A Hybrid GP/GA Approach for Co-evolving Controllers and Robot Bodies to Achieve Fitness-Specified Tasks

Wei-Po Lee John Hallam Henrik H. Lund
Department of Artificial Intelligence
University of Edinburgh
Edinburgh, U. K.
{weipol,john,henrikl}@aifh.ed.ac.uk

Abstract— **Evolutionary approaches have been advocated to automate robot design. Some research work has shown the success of evolving controllers for the robots by genetic approaches. As we can observe, however, not only the controller but also the robot body itself can affect the behavior of the robot in a robot system. In this paper, we develop a hybrid GP/GA approach to evolve both controllers and robot bodies to achieve behavior-specified tasks. In order to assess the performance of the developed approach, it is used to evolve a simulated agent, with its own controller and body, to do obstacle avoidance in the simulated environment. Experimental results show the promise of this work. In addition, the importance of co-evolving controllers and robot bodies is analyzed and discussed in this paper.**

I. INTRODUCTION

The behavior-based approach has been proposed as a methodology for building autonomous robots. Although many successful robots have been built based on this approach, increasing robot complexity makes the design difficult. Consequently, the evolutionary approach was advocated to provide some kind of design automation for building behavioral modules [3][4].

The first work proposing to use the genetic approach to synthesize programs for robot control is [6]. Due to being overly simplified, however, it has been criticized as being not complicated enough to control a real robot [3]. Another example of using a GP approach to evolve control programs is given in a series of papers by Reynolds. In the final version [8], the author applied arithmetic operations, such as $+$, $-$, $*$, %, and the conditional operation "iflte" to calculate a single output value from sensor values and interpreted it as the steering direction. The robot is then assumed to move for a fixed forward distance in the steering direction.

In our work, the structure of a control program differs from theirs. Our control programs are defined at the logic level: a control program is much like a boolean network, which maps *conditional structures* (constructed on sensor information) into appropriate motor commands. Instead of using compound robot actions, such as moving forward 1 foot or turning left 30 degrees, we use the outputs of the boolean network to directly drive left and right motors to revolve at different speeds. This results in a vehicle moving forward, backward, turning left, right smoothly and continuously.

The other difference between this work and all the others is that this work considers co-evolving the structure of a mobile robot, which has not yet been taken into account in the literature so far as we can discern. In previous work, the authors change just their controllers if the performance of a robot is not satisfactory. But our approach to improve is to change not only the controller but also the robot body itself because both affect the robot's behavior. Although Sims has evolved creatures with controllers and morphologies [9], his creatures are actually constituted from rigid parts which are not practical in the real world. Our way is to extract the determining parameters in designing a robot body then to apply the evolutionary algorithm to decide these values. The main goal of this work is then to investigate how to evolve the controllers and the robot bodies together to achieve a specified task.

II. THE SIMULATION

Agents can be equipped with different kinds of sensors for different tasks. In our current implementation, the agents are restricted to use only infra-red sensors (IRs) to acquire distance information. We assume that the robot has a physical round body and the IRs are positioned around the body pointing radially outward. The characteristic of an IR sensor is that it can only sense objects within a certain distance and a certain bearing. The visual distance is 30 cm and the bearing is 20 degrees in our simulation.

In this simulator, the motion system of the vehicle is regarded as a process with natural dynamics. It converts motor commands with required speeds (Full/Half, Forward/Reverse) into actual motions, and is modeled by related first-order differential equations. Once the time constant of the process is specified and the speed commands are determined by the control program, the rotational velocity of each motor can be calculated. Then the moving speed, the turning speed, and the new position of a vehicle are calculated by applying appropriate kinematic equations, with the specified wheel radius and wheelbase. The vehicle is driven by two independent motors with separate speed commands. This results in a vehicle able to move forward, backward, turn right and left at any possible speed. In order to make the simulation more realistic and to enhance the robustness of the evolved solutions, 5% random noise ($+5\% \sim -5\%$ uniformly) is injected to the perceptions and the motions.

III. GENETIC IMPLEMENTATION

A. System Overview

Our genetic system is a hybrid of Genetic Programming [6] and Genetic Algorithms [5]. An individual in this genetic system consists of a controller and a robot body, treated as $< brain, body >$, in which a brain is a tree-like program and a body is described by a string of real numbers. The GP part of this system evolves the tree-like program and the GA part evolves the floating point string. The aspect of an individual is shown in Figure 1.

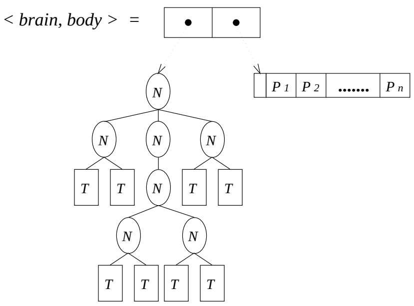

Fig. 1. The aspect of an individual defined in this work: in the tree structure, a node with the N/T is a non-terminal/terminal node; in the string representation, P_i is a real number.

Given an environment and a goal formulated as a fitness function, an initial population is created at random. Each individual has its own brain and body. To evaluate an individual is to execute the brain on the corresponding simulated robot body for a period of time and to measure the performance. The probability of the survival of an individual is then determined by how well the controller performs with the corresponding body to fit the evaluating criteria. After evaluating each individual, a certain selection method is employed to choose parent individuals and genetic operations are applied on them to create children individuals. In this work, the *tournment selection* method, which picks K individuals at random and chooses the fittest as the parent, is used.

Like other genetic work, genetic operations, such as *reproduction*, *crossover*, and *mutation*, are applied to the current population to create new individuals. The reproduction operation simply copies the selected parent individuals, without changing the controllers or the bodies, into the next generation. The crossover or mutation operation is allowed to to take place on the brain(s) or the body(bodies) at random. Due to the special structure of an individual here, however, the crossover is constrained to occur on both brains or both bodies from the involved parents in order to maintain the correctness of the structures.

Because of the different representations of the brain and the robot body, this work requires separate crossover and mutation operations for the tree expressions and the linear strings. Related techniques of GP and GA are applied independently for the brains and the bodies. The details are described in the following sections.

B. Evolving Brains with Sensors

Although sensors are parts of a robot body, they are closely associated with the control programs. Thus they are directly co-evolved with the control programs as described in this section. All other parts of a robot body are left and discussed in the next section.

B.1 Interpretation of the Control Program

A "brain" here means a *reactive* control program which controls the corresponding body. As we know, a reactive controller can be considered as a combinational logic system in which the output is determined completely by the current input states at each time step. It is well accepted that any combinational system can be described by a *boolean network* and a boolean network can be converted into a *boolean tree* [1][2], so that we define our control programs at logic level and use logic components, such as AND, OR, NOT, to map structured sensor conditionals into motor commands. Genetic Programming techniques are used to evolve such control programs.

We define a control program organized by three subtrees, and a non-terminal PROG is defined as a dummy root node to connect the three subtrees. The output of the first subtree is interpreted as the revolving direction of the left and right motors: if the output is 0, the control program will command both motors to revolve forward, otherwise it commands both motors to revolve backward. The evaluated results of the second and the third subtrees determine the speeds of the left and the right motor: 0/1 represents full/half speed. Three basic logic operations, AND, OR, and NOT, are defined as non-terminals to constitute the main frames of the three subtrees and map different combinations of structured sensor information (described below) into motor commands.

According to our program design, structured sensor-conditionals involve comparing the values from different sensors or comparing the sensor values to the numerical values as thresholds. Different kinds of sensors are required to build different behavioral modules. We use the task *obstacle avoidance*, which is the application task used in the experiment later, as the example to explain how the sensor information is structured.

We assume the agent uses simulated IR sensors to acquire distance information in this task. The characteristic of an IR sensor is that it can only sense the obstacle within a certain distance. Based on this, the sensor information from a certain sensor is converted to a signal indicating how safe the agent is in the direction which the sensor is pointing. So a safety signal can be 1, indicating that there is no obstacle within the maximum distance which an IR sensor can sense, a value between 0 and 1, which is the ratio of sensed distance to the maximum distance an IR sensor can sense, or 0, indicating that the agent has bumped against an obstacle.

We then define the symbol IR as one of the terminals. In this work, we intend to co-evolve sensor positions with the controller, so that there are no pre-defined position candidates for IR sensors. The agent is allowed to acquire distance information from IR sensors in *any* direction it wants. Each terminal IR is defined to be associated with a value between 0 and 1 which indicates the angle between the direction in which the IR is pointing and the agent's heading. In this way, whenever a terminal IR is evaluated, converted distance information (safety signal), in the direction indicated by the value associated with it, is returned. For instance, a terminal IR with a value 0.5 will return the converted sensor information in the direction 0.5 revolution (180 degrees) relative the agent's heading.

The other terminal defined in this task is a numerical value between 0 and 1 inclusive: these are used as thresholds. Thus a structured sensor-conditional in this work is defined as the *constrained syntactic structure* X $>$ Y, where X, Y can be any terminal. The symbol '$>$' is a non-terminal which performs the comparison operation and returns a boolean true/false according to the result.

To sum up, a reactive program here includes three boolean subtrees. The evaluation results of the boolean trees are interpreted as motor commands to drive left and right motors directly. A typical control program is shown and illustrated in Figure 2.

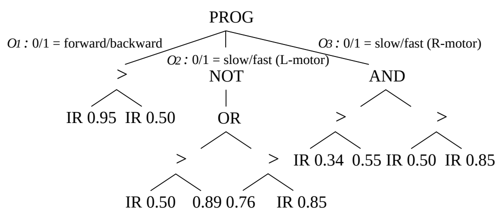

Fig. 2. The diagram of a typical control program.

B.2 The Genetic Operations

As we mentioned previously, the genetic system simply copies individuals, without changing the brains or bodies, into the next generation for reproduction. But separate crossover and mutation operations are required for tree structures (controllers) and the string of real numbers (robot bodies). The crossover and mutation operations for controllers are described as follows and those for robot bodies will be described in the next section.

The *crossover* operation involves swapping two subtrees from the parent control programs. Because there are some *constrained syntactic structures*, such as X $>$ Y, defined in this work, the crossover operation must be restricted. If the selected crossover point in the first parent is the root node, the second crossover point must be a root node as well (this means two individuals exchange their brains to become new individuals); if the chosen crossover point in one parent is an internal node, then the crossover point in the other parent must be an internal node too; otherwise if the selected crossover point in the first parent is a terminal node, the crossover point for the second parent is restricted to be a terminal node. In the last case there is an additional operation, *averaging*, that will possibly occur. If the types of the two terminals are different, i.e., a IR and a numerical constant, the system swaps them as described. But if the two terminals are the same type then *swapping* or *averaging* would occur randomly. The latter averages the two values associated with IRs or the two numerical constants. The *mutation* operation deletes a subtree at a randomly selected point and re-creates a random subtree to substitute it in the selected individual.

When the above genetic operations occur to the brains of the individuals, their corresponding bodies are not changed. This means, a changed brain with an unchanged robot body is put together to constitute a new individual in the next generation.

C. Evolving Robot Bodies

An agent is made up of by a brain and a body; both can affect the behavior. The performance of an agent is measured by how well the task is achieved by executing the brain on the corresponding body. In this section, we describe how to represent a robot body and to employ the genetic approach to evolve such a body.

In order to evolve a robot body we need to analyze and extract the determining elements, which affect the behavior of a robot profoundly, from the structure of a robot. In mobile robot design, for instance, there are some determining elements such as the wheel radius, the width of the wheel base, the time constant of the motion system, the body size (the diameter of the body, if we assume the robot body is round), and positions (with orientations) of the sensors, etc. The wheel radius affects the speed of the robot and determines the maximum and minimum moving speed for the specified motor commands; the width of the wheel base determines the turning rate of a robot; the time constant affects the response of the robot and determines the acceleration of the robot; the size of a robot body should be task-oriented: to avoid obstacles it may need to be smaller but to push boxes it may need to be larger; and the positions and orientations of the sensors allow the robot to acquire the perceptual information it needs. To evolve a robot body, in fact, means to decide these determining *structural parameters* of a robot genetically.

The structural parameters can be arranged as a linear string, in which each position is a real number representing the value of the corresponding parameter. Due to hardware limitations and performance considerations, each structural parameter has its lower bound and upper bound. When we build a robot, the value of each structural parameter must be between its bounds. Thus a robot body can then be expressed as

$$P_1 P_2 .....P_n$$

where

$$Min(P_i) \leq P_i \leq Max(P_i) \;\; ; \;\; 1 \leq i \leq n$$

For the linear string representation, a Genetic Algorithm can be employed to determine the value of each structural parameter P_i in its range.

Two-point crossover and one-point mutation operations are used to create new body strings. The crossover operation here, like the standard two-point crossover, involves two parents and two crossover points for parents. But its function is slightly different from the standard one: it is defined to perform operations of *exchanging* or *averaging* at random. The *exchanging* operation exchanges the P_i between two crossover points, but the *averaging* operation averages the corresponding P_i for the two parent strings instead. The mutation operation randomly picks a P_i for the selected parent and substitutes it with a re-generated random number, which satisfies its upper and lower bounds, so generating a new string.

IV. EXPERIMENTS AND RESULTS

In the experiment, we define "obstacle avoidance" behavior as the application task to evaluate the developed approach. The experiment is arranged in two phases. In the first phase, we concentrate on how to evolve an individual to move without collision; and in the second phase, we investigate the importance of the appropriate brain-body coupling.

A. Fitness Measures

As mentioned before, to evaluate an individual is to execute the control program on the corresponding robot body for a given period of time and to measure the performance according to certain criteria (fitness function). In this work, a fine time-slice technique is used. At each time step, the control program is evaluated once and drives its body to move; then the corresponding fitness is calculated. The accumulated fitness of an individual during the given time steps is then used to measure the performance.

An obstacle avoidance behavior means that an agent can keep moving without collision. A straightforward way to formulate this is to keep it as safe as possible at each time step. To achieve this, each agent is equipped with eight IR sensors, which point towards -$\frac{3}{4}\pi$, -$\frac{1}{2}\pi$, -$\frac{1}{4}\pi$, 0, $\frac{1}{4}\pi$, $\frac{1}{2}\pi$, $\frac{3}{4}\pi$, and π relative to the heading of the agent, and is trained to keep the safety signals from these IRs as close to 1 as possible. (The eight IRs mentioned here are used for fitness assessment only: they are independent of those sensors evolved as parts of the controllers. After training, these eight IRs are removed.) The term *minimum_safety*, which is the minimum of the eight safety signals, is used in the fitness function for this purpose. In order to keep it safe, the vehicle is punished whenever it begins getting dangerous (that is, the *minimum_safety* is less than 1), and the lower this value, the larger the penalty. In addition, in order to avoid the degenerate situation when an agent sticks at a certain position, or the situation when an agent spins, an agent is encouraged to move straight at high speed, and discouraged from rotation. Thus, the fitness function is defined as a penalty function. For an individual I,

$$Fitness(I) = \sum_{j \in Cases} \{\sum_{i=1}^{k} penalty(t_i) + (N-k) * penalty(t_k)\}_j$$

where

$$penalty = [\alpha * (1 - minimum\_safety) + \beta * (1 - v) + \gamma * w]$$

In this function, $Cases$ is the set of fitness evaluations done on this controller (fitness cases), k is the time the vehicle hits an obstacle, N is the given number of time steps that the obstacle avoidance behavior should last for, v is the normalized forward speed (backward is regarded as negative), and w is the normalized rotating speed. This would keep a vehicle safe and moving forward as straight as possible.

B. Evaluation and Testing

In this experiment, we trained the robot to avoid obstacles using the evolutionary procedure then tested the evolved pair of controller and robot body to examine the performance. Before training, we defined a training set including M starting positions. For a certain generation, each individual was trained to move from C starting positions, which were chosen randomly from the predefined training set, and ran for N time steps for each start position. The cumulative fitness is designated as the fitness of an individual. In our experiment, M was 30, C was 15 and N was 500.

The structural parameters we hope to evolve in this work are *time constant*, *wheel base*, *wheel radius*, and the *body size*, but each structural parameter has its own limitations. In this experiment, the value of time constant was restricted between 0.5 and 2.5 second; the lower bound and upper bound of wheel radius was 1.0 cm and 3.5 cm; the value of body size was limited from 10 cm to 25 cm; and the wheel base was constrained to be not larger than the body size.

A simple *island* model GA[10] is implemented in this work. It allowed us to use multiple populations to maintain diversity. We used two populations of 40 individuals each.

The number of generations was 50 and the best individual appearing in the *last* generation is designated as the final solution. The best individual evolved from this procedure and its typical behavior is shown in Figure 3.

```
( PROG
  (> IR 0.52  IR 0.97)
  ( OR ( OR ( AND (> IR 0.19  IR 0.97)
                  (> IR 0.21  IR 0.97))
              (> IR 0.60  IR 0.97))
         (> IR 0.23  IR 0.21))
  (> IR 0.71  IR 0.83))

time constant  : 0.68
wheel base     : 10.53
wheel radius   : 1.56
body size      : 10.53
```

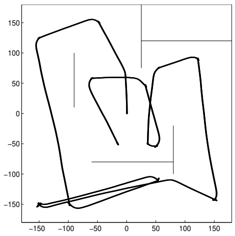

Fig. 3. The evolved solution(including the control program and the structural parameters); and the emergent behavior.

In order to examine the performance of the evolved solution, it is necessary to test it in different test cases. In our testing procedure, the evolved individual was tested 100 times, to control for random effects of perceptual and motor noise. Each time it moved from a new starting position with a given orientation and was allowed to move for 10000 time steps. The evolved best individual did not bump any obstacle in all the 100 test cases.

In addition, we tested the evolved agent in somewhat different environments to examine whether it still had reasonable performance. In 10 different tests, the evolved agent did not have any collision in all of the test cases. Figure 4 illustrates two examples.

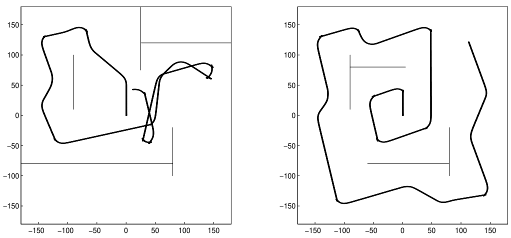

Fig. 4. Two examples of testing the evolved agent in changed environments.

C. The Importance of Appropriate Brain-Body Coupling

We have shown that the controller and the robot body can be co-evolved to achieve the behavior-specified task. In order to investigate how the evolved control program relies on the co-evolved body, we tested the evolved program on different robot bodies which are the various combinations of structural parameters available in their own ranges.

For each robot body, we tested it with the evolved control program 100 times. Each time it moved from a new starting position and was allowed to move 10000 time steps, like the testing procedure described in the above section. Each entry in Table 1 shows the number of successes (cases when the robot did not bump any wall during the 10000 time steps) from the 100 test cases for a certain robot body.

From Table 1, we find that the combination where the evolved controller was executed on the co-evolved robot body (marked with an asterisk) has the highest success rate (actually it is 100%). The inappropriate brain-body couplings can not achieve the specified task perfectly. This demonstrates that the evolved controller relies on the co-evolved robot body.

body-size	*wheel-base*	*wheel radius* 1.56	2.34	3.12	*wheel radius* 1.56	2.34	3.12
10.53	1.05	83	74	75	86	79	64
	5.26	96	83	70	84	74	71
	10.53	100*	99	97	96	86	81
15.79	1.05	64	61	52	71	59	42
	5.26	76	56	74	52	45	27
	10.53	89	95	92	95	64	51
	15.79	90	88	92	94	92	84
21.05	1.05	41	33	26	52	46	34
	5.26	51	24	11[d]	31	13	6
	10.53	30	53	65[c]	91	34	17
	15.79	24[b]	28	27[e]	67	75	68
	21.05	87[a]	19	30	62	59	41

Table 1: The testing results of different brain-body couplings. The *time constants* of the left and right sets of columns are 0.68 and 1.35 seconds.

C.1 Further Investigation

Some data in Table 1 attracts our attention. For the case a (indicated by the mark a at the corresponding number), for instance, the number of success increases dramatically, comparing to case b. We can explore the reasons by observing the behaviors emerging from the two pairs of brains and bodies.

In case b, the robot always bumped the wall due to the inappropriate enlargement of the body size and wheel base. But if the wheel base is enlarged more, the robot became more difficult to rotate, especially with the relatively small wheels which slow down the motion of the robot. This causes the robot to get stuck easily — it oscillates forward and backward with a little turning at a certain position, so it is safe in most of the test cases. The typical behaviors of case a and case b are shown in Figure 5.1 and Figure 5.2.

The other example we investigated is case c, whose performance is much better than cases d and e. After examining their behaviors, we found that there are actually two types of failure caused by certain kinds of robots which bump the wall easily. The first is a robot with a large body, small wheel base, and large wheels. A large body inevitably increases the bumping probability; the small wheel base with large wheels makes the behavior of the robot unstable: it is easy to turn at high speed. Consequently, it often drove the robot to bump the wall suddenly. Figure 5.3 shows this situation.

The second is a robot with a large body, wide wheel base, and large wheels. As in the first situation, a large body increases the bumping probability for the robot but the wide wheel base with large wheels, on the contrary, drives the robot forward faster with small turning rate. This results in the robot bumping the wall in most of the test cases although it can detect the wall efficiently and tries to move away from the wall. Figure 5.4 shows this situation.

As we can observe, if the wheel base of the robot becomes larger, failure of the first kind decreases, but failure of the second kind increases. In case d, most of the failures are because of the first situation but in case e most of the failures belong to the second situation. Regarding the two situations which cause the bumping behavior together, we find case c happens to be the best case with a small number failures for both situations. So its performance is better than case d and case e.

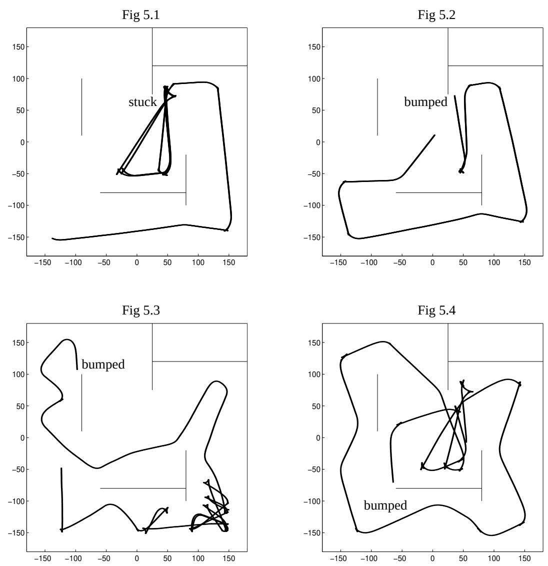

Fig. 5. Some faults caused by the inappropriate brain-body couplings (see text for explanation)

V. CONCLUSION AND FUTURE WORK

In this paper, we have developed a hybrid approach of Genetic Programming and Genetic Algorithms to co-evolve a reactive control program and its corresponding robot body to achieve the specific behavior. A boolean-tree has been well-defined to represent a control program and the determining structural parameters of a physical robot are extracted and arranged as a linear string of real numbers to represent a robot body. The GP part of the system is used to evolve the tree-structure of a control program and the GA part of the system is applied to determine the string of the structural parameters. Experimental results have shown the promise of the developed approach.

In addition, we have also analyzed the importance of appropriate brain-body coupling in designing a robot system. The evolved controller is successful only performing in the co-evolved robot body. This means, in the evolutionary process, the robot body itself also plays an important role because they both have adapted to the environment in order to achieve the task. For the simple tasks, a human designer may be able to design a robot body according to his prediction of the difficulty of the tasks, then design (or evolve) the controller. But for more complicated tasks, co-evolving controllers and morphologies for robot systems may provide a potential alternative.

Some aspects of future work are important. First of all, because our work is done in simulation, it is necessary to build a real robot (Lego-like) and download the evolved controller to it to observe the performance. Although there are gaps between simulated and real worlds, research [7] has shown that we could sample the real sensor data and the robot motion in the real world to build a more realistic simulator to develop evolutionary systems. Our future work also involves integrating these considerations into our simulator. Finally, since our experiment is focused on a certain behavior, we can go on to consider more difficult behaviors. Based on what we learned from this, we shall furthermore see if we can successfully evolve more new combinations of controllers and robot bodies for different tasks.

References

[1] S. B. Ackers. Binary Decision Diagrams. In *IEEE Transactions on Computers*, C-27(6), p509-516, 1978.

[2] R. E. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. In *ACM Computing Survey*, 24(3), p293-318, 1992.

[3] R. A. Brooks. Artificial Life and Real Robots. In *Proceedings of the First European Conference on Artificial Life*, p3-10, 1991.

[4] D. Cliff, I. Harvey, P. Husbands. Explorations in Evolutionary Robotics. In *Adaptive Behavior*, 2(1), p71-104, 1993.

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

[6] J. R. Koza. *Genetic Programming: on the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[7] O. Miglino, H. H. Lund, S. Nolfi. Evolving Mobile Robots in Simulated and Real Environments. To appear in *Artificial Life*, 1996.

[8] C. W. Reynolds. Evolution of Corridor Following Behavior in a Noisy World. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, p402-410, 1994.

[9] K. Sims. Evolving 3D Morphology and Behavior by Competition. In *Proceedings of Artificial Life IV*, p28-39, 1994.

[10] R. Tanese. Distributed Genetic Algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, p434-439, 1989.