Learning Biped Locomotion from First Principles on a Simulated Humanoid Robot Using Linear Genetic Programming

Krister Wolff and Peter Nordin

Dept. of Physical Resource Theory, Complex Systems Group,
Chalmers University of Technology,
S-412 96 Göteborg, Sweden
{wolff, nordin}@fy.chalmers.se
http://www.frt.fy.chalmers.se/cs/index.html

Abstract. We describe the first instance of an approach for control programming of humanoid robots, based on evolution as the main adaptation mechanism. In an attempt to overcome some of the difficulties with evolution on real hardware, we use a physically realistic simulation of the robot. The essential idea in this concept is to evolve control programs from first principles on a simulated robot, transfer the resulting programs to the real robot and continue to evolve on the robot. The Genetic Programming system is implemented as a Virtual Register Machine, with 12 internal work registers and 12 external registers for I/O operations. The individual representation scheme is a linear genome, and the selection method is a steady state tournament algorithm. Evolution created controller programs that made the simulated robot produce forward locomotion behavior. An application of this system with two phases of evolution could be for robots working in hazardous environments, or in applications with remote presence robots.

1 Introduction

Dealing with humanoid robots requires supply of expertise in many different areas, such as vision systems, sensor fusion, planning and navigation, mechanical and electrical hardware design, and software design only to mention a few. The objective of this paper, however, is focused on the synthesizing of biped gait.
The traditional way of robotics locomotion control is based on derivation of an internal geometric model of the locomotion mechanism, and requires intensive calculations by the controlling computer, to be performed in real time. Robots, designed in such a way that a model can be derived and used for controlling, shows large affinity with complex, highly specialized industrial robots, and thus they are as expensive as conventional industrial robots. Our belief is that for humanoids to become an everyday product in our homes and society, affordable for everyone, there is needed to develop low cost, relatively simple robots. Such robots can hardly be controlled the traditional way; hence this is not our primary design principle.

E. Cantú-Paz et al. (Eds.): GECCO 2003, LNCS 2723, pp. 495–506, 2003.

A basic condition for humanoids to successfully operate in human living environment is that they must be able to deal with unpredictable situations and gather knowledge and information, and adapt to their actual circumstances. For these reasons, among others, we propose an alternative way for control programming of humanoid robots. Our approach is based on evolution as the main adaptation mechanism, utilizing computing techniques from the field of Evolutionary Algorithms.

The first attempt in using a real, physical robot to evolve gait patterns was made at the University of Southern California. Neural networks were evolved as controllers to produce a tripod gait for a hexapod robot with two degrees of freedom for each leg [6]. Researchers at Sony Corporation have worked with evolving locomotion controllers for dynamic gait of their quadruped robot dog AIBO. These results show that evolutionary algorithms can be used on complex, physical robots to evolve non-trivial behaviors on these robots [3] and [4].

However, evolving efficient gaits with real physical hardware is a challenge, and evolving biped gait from first principles is an even more challenging task. It is extremely stressing for the hardware and it is very time consuming [17]. To overcome the difficulties with evolving on real hardware, we introduce a method based on simulation of the actual humanoid robot.

Karl Sims was one of the first to evolve locomotion in a simulated physics environment [13] and [14]. Parker use Cyclic Genetic Algorithms to evolve gait actuation lists for a simulated six legged robot [11], and Jakobi et al has developed a methodology for evolution of robot controllers in simulator, and shown it to be successful when transferred to a real, physical octopod robot [7] and [9]. This method, however, has not been validated on a biped robot. Recently, a research group in Germany reported an experiment relevant to our ideas, where they evolved robot controllers in a physics simulator, and successfully executed them onboard a real biped robot. They were not able to fully realize biped locomotion behavior, but their results were definitely promising [18].

2 Background and Motivation

In this section we summarize an on-line learning experiment performed with a humanoid robot. However this experiment was fairly successful in evolving locomotion controller parameters that optimized the robot's gait, it pointed out some difficulties with on-line learning. We summarize the experiment here in order to exemplify the difficulties of evolving gaits on-line, and let it serve as an illustrative motivation for the work presented in the remainder of this paper.

2.1 Robot Platform

The robot used in the experiments is a simplified, scaled model of a full-size humanoid with body dimensions that mirrors the dimensions of a human. It was originally developed as an alternative, low-cost humanoid robot platform,

intended for research [17]. It is a fully autonomous robot with onboard power supply and computer, and it has 14 degrees of freedom.

The robot has the 32-bit micro-controller EyeBot MK3 [2] onboard, carrying it as a backpack. All signal processing, including control, vision, and evolutionary algorithm, is carried out on the EyeBot controller itself. In its present status, the robot is capable of static walking.

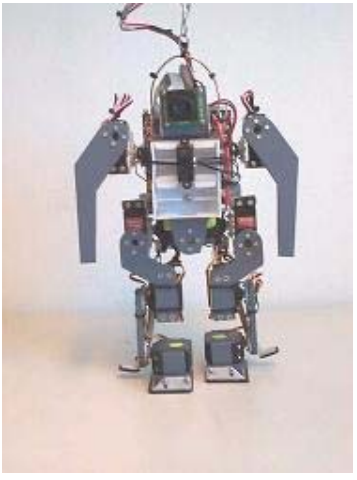

Fig. 1. Image of the real humanoid robot 'elvina'.

2.2 Gait Control Method

The gait control method for this robot involves repetition of a sequence of integrated steps. Considering fully realistic bipedal walk, two different situations arise in sequence: the statically stable double-support phase in which the robot is supported on both feet simultaneously, and statically unstable single-support phase when only one foot of the robot is in contact with the ground, the other foot being transferred from the back to front position. When this sequence of transitions has been repeated twice, one can consider a single gait cycle to be completed. That is, the locomotion mechanism's posture and limb's positions are the same after the completion as it was before it started to move, and hence it's internal state is the same.

If we now study only static walk, i.e. the projection of the center of mass of the robot on the ground always lie within the support polygon formed by feet on the ground, there is obviously a number of statically stable postures in between the internal state of the robot and it's final state, during completion of a single gait cycle. By interpolation between numbers of such, statically stable, consecutive states it is possible to make the robot to complete a single gait cycle. Then, by continually looping, biped gait is produced.

2.3 Evolutionary Gait Optimization Experiment

This experiment was performed in order to optimize a by hand developed set of state vectors, defining a static robot gait. The evolutionary algorithm used was a tournament selection, steady state evolutionary strategy [1] and [16] running on the robot's onboard computer. Individuals were evaluated and fitness scores automatically determined using the robots onboard digital camera and proximity sensor.
A population of 30 individuals, stemming from a manually developed individual was created with a uniform distribution over a given search range. The best-evolved individual and the manually developed individual were independently tested, and their performances were compared to each other's. The former one received a fitness score, averaged over three trials, of 0.1707, and the latter one, tested under equal conditions, got a fitness of 0.1051. Within this context, a higher fitness value means a better individual, and thus the best-evolved individual outperformed the manually developed individual both in its ability to maintain the robot in a straight course and in robustness, i.e. with a lesser tendency to fall over [17].

2.4 Observations

To run such an evolutionary experiment as described above span over several days, and requires manual supervision all this time. Between each generation of four individuals evaluated, the experiment was paused for about 15 minutes in order to spare the hardware and especially the actuators. The main reason for this is that the actuators accumulate heat when they are running continuously under heavy stress. They then run the risk of getting overheated and gradually destroyed. One way to handle this problem was, as mentioned above, to run the robot intermittent so that the servos maintain an approximately constant temperature.
Evolving efficient gaits with real physical hardware is a challenging task. During the experiments, the torso and both the ankle actuators were exchanged once as well as the two hip servos. The most vulnerable parts of the robot were proved to be the knee servos. Both these servos were replaced tree times.
Obviously there are a number of difficulties related with evolving biped walking behavior on a real, physical robot. In an attempt to overcome some of the problems, we want to use a physically realistic simulation of the robot. The central idea in this concept is to evolve control programs from first principles on a simulated robot, transfer the resulting programs to the real robot and continue to evolve efficient gait on the real robot. Of course, there will arise other problems applying this method, as simulation systems always imply some simplifications of the real world.

3 Evolution of Control Programs

Our primary goal is to utilize Genetic Programming [5] and [8] for evolving locomotion control programs from first principles for our simulated biped robot,

i.e. with no a priori knowledge for the robot on how to walk, information of morphology etc. The evolved programs take the robot's *current* internal state parameter values as input vector and return a vector predicting it's *next* internal state parameter values, in order to produce robust biped gait.

3.1 Dynamic Physics Simulation

The Open Dynamics Engine (ODE) is a free library for simulating articulated rigid body dynamics, developed by Russell Smith [15]. An articulated structure is created when rigid bodies of various shapes are connected together with joints of various kinds.
The robot model is qualitatively consistent with the real robot in the aspect of geometry, mass distribution, and morphology. See [17] for details of the robot. It consists of 12 actuated joints and 13 body elements. It is constructed with its mass concentrated to the main body elements, which in the real robot correspond to the servo actuators, batteries and computer. The plastic body parts, interconnecting the servos to each other, are not rendered in the simulation, since their mass is very low compared to the total mass.

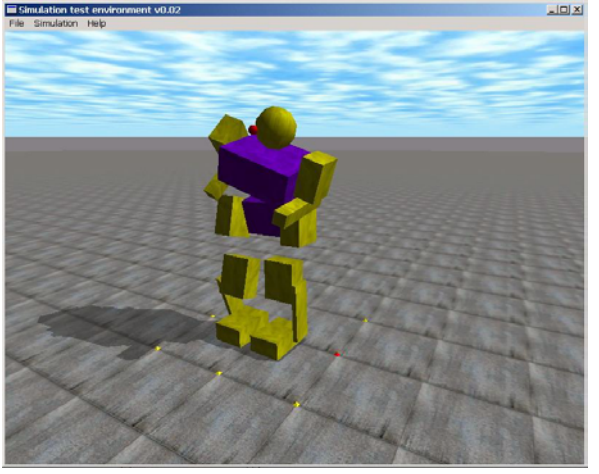

Fig. 2. Snapshot of the simulated humanoid robot. The body elements are directly connected to each other, although this is not visualized here.

3.2 Virtual Register Machine

The Genetic Programming representation used for this problem of robot control program induction is an instance of a Virtual Register Machine, VRM(k, l) [10].

It has k I/O registers and l internal work registers. In the current implementation of our system, l equals k. The function set consists in the present of arithmetic functions ADD, SUB, MUL, DIV, where DIV is protected division, and SINE. We now define a register state vector $\mathbf{Reg} \equiv [Reg_1, ..., Reg_k]$ of k integers, each of the elements corresponding to one of the actuated joints of the simulated robot. All program input/output is communicated through the states of the I/O registers. That is, program inputs are supplied in the initial state Reg, and output is taken from the final register state Reg'. Further, the I/O register state vector is initially copied into the internal work registers. We can do this in a straight forward manner, since we have imposed that the number of I/O registers, k, equals the number of work registers, l. The Virtual Register Machine is allowed writing only to the internal work registers when looping the program instructions. The I/O registers are write-protected in this phase, and their final state is updated after the end of the program execution cycle, before they are passed to the robot and then updating it's internal state.

3.3 Linear Genome Representation

Each individual is composed of simple instructions between input and output parameters. Each instruction consists of four elements, encoded as integers, and the whole individual is a linear list of such instructions:

```
 8, 22, 3, 12,
19, 11, 2, 16,
15, 12, 3, 12,
 8,  3, 4, 19,
12, 12, 4, 21,
 1,  6, 5, 12,
20,  3, 1, 19,
 9, 12, 2, 21,
23,  5, 3, 19,
16,  9, 3, 14,
13, 21, 5, 19,
 6, 13, 5, 14,
16, 22, 3, 16,
16,  3, 4, 18,
 8, 19, 2, 13,
20,  5, 3, 20,
13,  6, 1, 14,
```

The encoding scheme is as follows; the first and second elements of an instruction refers to the registers to be used as arguments, the third element corresponds to the operator, i.e. ADD=1, SUB=2, MUL=3, DIV=4, and SINE=5, and the last element is a register reference for where to put the result of the operation. The meaning of the first line (instruction) here is: multiply register 8 with register 22 and put the result in register 12. The operators take two arguments,

except when the operator is SINE, which of course only take one argument. In this case, the SINE operator is applied to the first element in the instruction, and the second element is simply discarded. A mutation on that element will thus have no effect on that individual's genotype. The register references 1-11 are assigned to I/O-registers, and register references 12-23 are assigned for the internal work registers. Parsing the individual above, and print out the first three instructions in 'C-style' looks like this:

```
Reg12 = Reg8 * Reg22;
Reg16 = Reg19 - Reg11;
Reg12 = Reg15 * Reg12;
```

3.4 Evolutionary Algorithm

At the beginning of the evolutionary process, the population is filled with randomly created individuals. The length, or number of instructions, of an individual is chosen randomly with Gaussian distribution, with expectation value 20. The maximum length is restricted to 256 instructions. The genes are created with a uniform distribution over their respective search range; 1-23 for the two first genes of an instruction, 12-23 for the last gene, and 1-5 for the third gene, which corresponds to the function set.
Our GP-system is a steady state tournament selection algorithm, with the following execution cycle:

```
1. Select four members of the population for tournament.
2. For all members in tournament do:
   a. Create an instance of the simulated robot.
   b. Record the position in 3d-space of all the robot's limbs.
   c. Execute the individual for 2500 simulation time steps.
   d. Record the final position of all the robot's limbs.
   e. Compute the fitness value (see below).
   f. Destroy the simulated robot.
3. Perform tournament selection.
4. Apply genetic operators on the winners to produce two children.
5. Replace the two losers in the population with the offspring.
6. Go to step 1.
```

The individuals are evaluated (evaluation cycle starting with point 2a. above) under identical conditions, since the simulation is entirely deterministic. They all start from the same standing upright pose, with the same orientation. The execution time for individuals are 2500 simulation time steps (corresponding to approx. 20 seconds of real time simulation), and if an individual cause the robot to fall before this time is completed, the evaluation is terminated. In the beginning of an experiment, a great majority of individuals are terminated before the intended time. Looping an individual once does *not* correspond to a single simulation time step, but to moving the robot's limbs between two consecutive internal states ('states' being referred to as in the subsection *Gait Control Method*).

Table 1. Koza style tableau, showing parameter settings for the evolution of locomotion control programs for the simulated humanoid robot.

Parameter	Value
Objective	Approximate a function that produce robust biped gait
Terminal Set	24 integer registers,
Function Set	ADD, SUB, MUL, DIV, SINE
Raw Fitness	According to eq. (2), scalar value
Standardized Fitness	Same as Raw Fitness
Population Size	800
Initialization Method	Random
Simulation Time	2500 simulation time steps
Crossover Probability	100%
Mutation Probability	80%
Initial Program Length	Gaussian distribution, expectation value 20.
Maximum Program Length	256 instructions
Maximum Tournament Number	None
Selection Scheme	Tournament, size 4
Termination Criteria	None (determined by the experimenter)

Fitness Calculation. As in all GP-applications, finding a proper fitness function that guides the artificial evolution in the desired direction is of great importance. The primary goal for the experiment was to produce a "human-like", bipedal gait without the robot falling. To accomplish this task, the individual controlling the robot should; (i) locomote the robot as straight forward as possible, and (ii) keep the robot in an upright pose during the movement. Hence, the proper measurements to feed the fitness function with are related to the height maintained by the robot, and the covered distance during simulation. Explicitly formulated in mathematical terms, the proper fitness function was found to be:

$$f = W\left[1.0 - \frac{h_{start}}{h_{stop}}\right] + (d_{left} + d_{right}) \tag{1}$$

where h_{start} is the height of the robot at the starting position, h_{stop} is the height when evaluation terminates (either the simulation is fully completed, or it is terminated before the intended time, caused by the robot falling). The height measure is applied to the position of the robot's head, however one could take the height of any body part. The second term is a measure of the distance covered by the robot during evaluation, applied to its feet. The robot is always starting with its feet in origo (in xy-plane). The first term will give a positive contribution to fitness if $h_{stop} > h_{start}$, negative contribution in the case when $h_{stop} < h_{start}$, and zero contribution if $h_{stop} = h_{start}$. Thus we have a fitness function rewarding forward locomotion and keeping the upright pose, and punishing backward movements and falling. The W in the first term is a weight,

scaling the mutual relation of rewarding and punishing. After some tweaking, it was found to work best when set to a value in the order of 10.

Genetic Operators. We use only two-point string crossover, with 100% probability for crossover, divided mutually on the rate 4:1 on homologous and non-homologous crossover.
When an individual is chosen for mutation, the mutation operator works by randomly selecting one single instruction from the individual, and make a change in the selected instruction. It makes that change either by changing any of the register references to another randomly chosen register reference from the register set, or the operator in the instruction may be changed. The probability for an individual to undergo mutation is 80%.

4 Results

When observing the experiments in run-time, it is compelling how quickly the simulated robot learns. In the first couple of hundred tournaments, a great majority of the individuals cause the robot to fall almost immediately in the beginning of the evaluation cycle, and the greater part of them tip over backwards. Maybe one out of ten individuals fall to the fore, which is a good starting point of taking a step ahead. Rather soon, however, one can observe the opposite situation, one out of ten individuals' overturn backwards and the rest fall ahead. This was not the desired goal for the evolution, but we regard this as being the first refined behavior that emerged.
The next observable stage of development in the evolution is when a large fraction of individuals is keeping the robot at a standstill, almost motionless, on its feet. In the beginning of our experiments, we faced some problems with evolution converged to this state. By increasing the population size and making some adjustments to the fitness function (mainly by decreasing the weight w, giving lesser punishment for tipping over), we could guide the evolution towards the desired goal. The mix of individuals showing this behavior, and individuals with a more 'energetic' behavior guarantee sufficient diversity of the population for evolution to proceed.
The final results of these experiment was indeed consistent with our initial objectives. That is, evolution created controller programs that made the simulated robot produce forward locomotion behavior. Some of the resulting programs made the robot walking forward in a spiral manner, with small movements, and others produced gait patterns with more lively movements. When tested, some of the individuals managed to keep the robot on its feet for the whole evaluation time (2500 simulation time steps), but when executed for a longer time, the robot usually ended up overturned. Nevertheless, a division of evolved programs could accomplish the task during the test run, without ever tipping over the robot.
Figures 3 and 4 displays some statistics from a representative run. In these experiments we did more than thirty independent runs, ranging from a few thousand

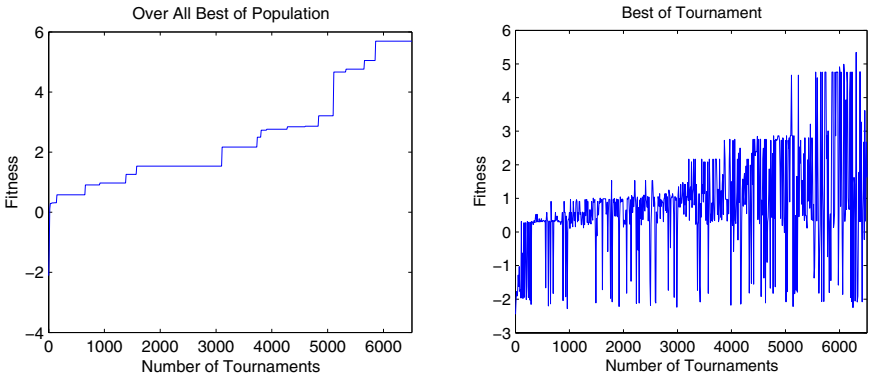

Fig. 3. a,Fitness value of over all best individual in the population (left) and **b**,fitness value of the best individual in every tournament (right).

tournaments, up to more than 80000 tournaments. The way fitness was defined (eq. 2), a fitness value < 0 correspond to the robot falling backward, and a small positive value (typically ranging from ~ 0.3 to ~ 0.6) correspond to the robot immediately falling ahead, while a value around 1.5 indicate a standstill. In figure 3a, one can observe how the best individual performed those behaviors; falling backward in the first few hundred tournaments, falling ahead in the first thousand tournaments, and standing still up to the 3000 tournaments. Fitness values in the range of ~ 1.5 to ~ 2.5 indicate some good locomotion, but usually ended up with the robot overturned, and fitness > 2.5 was successful locomotion behavior.

As depicted in figure 3a, the currently best individuals in the population showed progress from the beginning of the evolution and continued to develop over time. The program length typically decrease below the initialization length in the beginning of a run, but after a short while it starts to increase above that threshold, and finally it stabilize around some value. See figure 4. In all experiments we used the same initialization program length, with gaussian distribution and expectation value 20. It was observed that the program length, averaged over the whole population, did never go below the value 13, and never above 50, and it usually stabilized somewhere around 30.

5 Summary and Conclusions

We describe the first instance of an approach for control programming of humanoid robots. It is based on evolution as the main adaptation mechanism, utilizing computing techniques from the field of Evolutionary Algorithms. The central idea in this concept is to evolve control programs from first principles on a simulated robot, transfer the resulting programs to the real robot and continue to evolve efficient gait on the real robot. As the key motivation for using

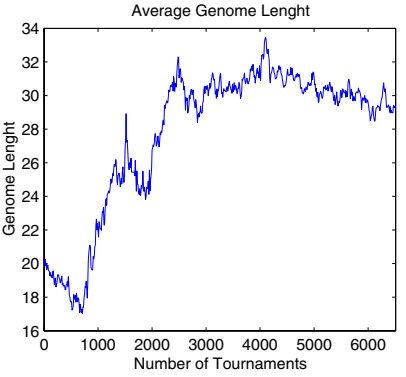

Fig. 4. Average genome length of all individuals in the population, length being defined as the number of instructions in an individual.

simulators, we briefly describe an on-line learning experiment performed with a biped humanoid robot.

The Evolutionary Algorithm is an instance of Genetic Programming, implemented as a Virtual Register Machine with 12 internal work registers and 12 external registers for I/O operations. The individual representation scheme is a linear genome, encoded as an array of integers. The selection method is a steady state tournament algorithm, with size four. The final results of these experiment was consistent with our initial objectives. That is, evolution created controller programs that made the simulated robot produce forward locomotion behavior. Current versions of the simulation system and the robot, however, do not allow the evolved programs to be directly downloaded to the robot. Further investigations and improvements are needed. To begin with, we must implement a subsystem of the simulated robot's control system and program interpreter on the real robots micro controller. Further, the real robot has an active feedback system, consisting of a color camera and a distance sensor, which will be implemented on the simulated robot as well. The development of the robot platform is an ongoing process, hence other sensors will be implemented on the robot. Then, the simulated robot should of course reflect all aspects, morphological and perceptual, of the real robot.
With this system of two phases of evolution, it will be possible to have a flexible adaptation mechanism that can react to hardware failures in the robot, e.g. if an actuator or sensor break down. By extracting information about malfunctioning parts and do off-line evolution with a modified model of the robot, it will become possible to react to the changes in the robot morphology. Another approach in this spirit, called Punctuated Anytime Learning, has been proposed by Parker [12]. For robots working in hazardous environments, or in applications with remote presence robots, this feature would be very useful.

References

1. Banzhaf, W., Nordin, P., Keller, R.E., and Francone F. D.: Genetic Programming An Introduction: On the Automatic Evolution of Computer Programs and Its Applications. San Francisco: Morgan Kaufmann Publishers, Inc. Heidelberg: dpunkt verlag. (1998)
2. Bräunl, T. 2002: EyeBot Online Documentation. Last visited: 01/21/2003. http://www.ee.uwa.edu.au/∼braunl/eyebot/
3. Hornby, G.S., Fujita, M. Takamura, S., Yamamoto, T., and Hanagata, O.: Autonomous evolution of gaits with the Sony quadruped robot. Proceedings of the Genetic and Evolutionary Computation Conference. San Francisco: Morgan Kaufmann Publishers, Inc. (1999)
4. Hornby, G.S., Takamura, S., Yokono, J., Hanagata, O., Yamamoto, T., and Fujita, M.: Evolving robust gaits with AIBO. IEEE International Conference on Robotics and Automation, New York: IEEE Press, pages 3040–3045. (2000)
5. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA, USA: MIT Press. (1992)
6. Lewis, M. A., Fagg, A. H., and Solidum, A.: Genetic programming approach to the construction of a neural network for control of a walking robot. Proceedings of the IEEE International Conference on Robotics and Automation. New York: IEEE Press. (1992)
7. Lund, H., and Miglino, O.: From Simulated to Real Robots. Proceedings of IEEE 3rd International Conference on Evolutionary Computation. New York: IEEE Press. (1996)
8. Langdon, W. B., and Poli, R.: Foundations of Genetic Programming. New York: Springer-Verlag. ISBN 3-540-42451-2, 274 pages. (2002)
9. Miglino, O., Lund, H., and Nolfi S.: Evolving Mobile Robots in Simulated and Real Environments. Technical Report, Institute of Psychology, C.N.R., Rome. (1995)
10. Nordin, P.: Evolutionary Program Induction of Binary Machine Code and its Applications. Ph.D. Thesis, der Universität Dortmund am Fachbereich Informatik, Germany. (1997)
11. Parker, G. and Rawlins, G.: Cyclic Genetic Algorithms for the Locomotion of Hexapod Robots. Proceedings of the World Automation Congress, Volume 3, Robotic and Manufacturing Systems. (1996)
12. Parker, G.: Punctuated Anytime Learning for Hexapod Gait Generation. Proceedings of the 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems. (2002)
13. Sims, K.: Evolving Virtual Creatures. Proceedings of Siggraph, pp.15–22. (1994)
14. Sims, K.: Evolving 3D Morphology and Behavior by Competition. Proceedings of Artificial Life IV, Brooks and Maes, editors, MIT Press, pp.28–39. (1994)
15. Smith, R.: Open Dynamics Engine v0.030 User Guide. Last Visited: 03/27/2003. http://opende.sourceforge.net/ode-0.03-userguide.html
16. Schwefel, H. P.: Evolution and Optimum Seeking. New York, USA: Wiley. (1995)
17. Wolff, K., and Nordin, P.: Evolution of Efficient Gait with an Autonomous Biped Robot using Visual Feedback. Proceedings of the Mechatronics Conference. University of Twente, Enschede, the Netherlands. (2002)
18. Ziegler, J., Barnholt, J., Busch, J., and Banzhaf W.: Automatic Evolution of Control Programs for a Small Humanoid Walking Robot. 5th International Conference on Climbing and Walking Robots. (2002)