Simbad : an Autonomous Robot Simulation Package for Education and Research

Louis Hugues, Nicolas Bredeche

Simbad : an Autonomous Robot Simulation Package for Education and Research

Louis Hugues[1] and Nicolas Bredeche[2]

[1] Ginkgo-networks, Paris, France
louis.hugues@wanadoo.fr - http://www.louis.hugues.xunuda.com/
[2] Equipe Inférence et Apprentissage, TAO / INRIA Futurs
Laboratoire de Recherche en Informatique, Bat.490,
Université de Paris-Sud, 91405 Orsay Cedex, France
bredeche@lri.fr - http://www.lri.fr/ bredeche

Abstract. Simbad is an open source Java 3d robot simulator for scientific and educational purposes. It is mainly dedicated to researchers and programmers who want a simple basis for studying Situated Artificial Intelligence, Machine Learning, and more generally AI algorithms, in the context of Autonomous Robotics and Autonomous Agents. It is is kept voluntarily readable and simple for fast implementation in the field of Research and/or Education.
Moreover, Simbad embeds two stand-alone additional packages : a Neural Network library (feed-forward NN, recurrent NN, etc.) and an Artificial Evolution Framework for Genetic Algorithm, Evolutionary Strategies and Genetic Programming. These packages are targeted towards Evolutionary Robotics.
The Simbad Package is available from http://simbad.sourceforge.net/ under the conditions of the GPL (GNU General Public Licence).

1 Introduction

This paper provides an introduction to Simbad, a new mobile robot simulator written in Java for Research and Education, and a set of tools for Evolutionary Robotics [7]. The main motivation is to provide an easy-to-use all-in-one package for Evolutionary Robotics. The Simbad package includes a mobile robot simulator with complex 3D scene modelling and simulation engine (Simbad), a Neural Network library (feed-forward and recurrent NN) as well as a complete Evolutionary Algorithm Library (Genetic Algorithm, Evolutionary Strategy, Genetic Programming with trees or graphs). All tools have been written to be efficient and easy to use by Java and/or Python programmers.

Section 2 present the state of the art in mobile robot simulation and highlights the specific characteristics of existing simulators. Section 3 describes the key motivations and characteristics of the Simbad simulator as well as its specificity compared to other simulators. This section also introduces the PicoNode and PicoEvo packages as well as implementation issues. In section 4, classic experiments are described using the Simbad packages. The last section concludes

this paper and refers to current applications of the Simbad package both for Research and Education.

2 Available Mobile Robot Simulators

Due to the spreading of the Open/GL API several tools provide some kind of 3D visualisation. However there are quite few simulators providing a complete 3D simulation and, in particular, sensing in the 3D space (vision, sonars, bumpers and lasers).

Player/Gazebo : Player and Gazebo projects have been developed at the USC (University of Southern California) [2, 3]. Those two components can be used in conjunction so as to obtain a powerful multi-robot simulation in a 3D environment. The Player project provides an abstract programming framework for real robots and is widely used in the robotics community. Gazebo is a 3D simulator able to simulate several Player-based Robots. Gazebo uses ODE (Open Dynamic Engine - a rigid body dynamics simulator) to compute physicals interaction with objects. Player, Stage and Gazebo are freely available via SourceForge with both binary and source code. Player/Gazebo constitutes a powerful package but relies on several complex components and thus requires some time to learn.

Webots [4] : is a product proposed by the Cyberbotics company from an initial basis developed by the LAMI laboratory at the EPFL (Ecole Polytechnique Federale de Lausanne). Its functionalities are very similar to those of Player/Gazebo but with the look and feel of a commercial package. As Gazebo, it uses ODE for physical simulation. The main disadvantages of Webots are its cost and the fact that source code is not available for standard users. This last point prevents experimenters to have a precise insight of the simulator behaviour.

Other tools : One can also mention the Robocup soccer simulator, EyeSim for the EyeBot Robot, SimBot and WoB [5]. Besides the 3D robot simulators, numerous tools exist which perform simulation in the 2D space. Among the very good ones are Stage simulator (which can be used in conjunction with Player instead of using Gazebo), Khepera Simulator the precursor of Webots and TeamBots, a multi robot Simulator.

3 The Simbad Package : Main Features

In this section, we describe the three main packages included in the Simbad Packages:

1. the Simbad simulator ;
2. PicoNode, the neural network library ;
3. PicoEvo, the Evolutionary Algorithm library.

3.1 The Simbad Simulator

Simbad [1] is a Java 3d multi-robots simulator developed for scientific and educational purposes. It is mainly dedicated to researchers and students who want a simple basis for studying Situated Artificial Intelligence, Machine Learning, and more generally AI algorithms, in the context of Autonomous Robotics and Autonomous Agents. Simbad enables programmers to write their own robot controllers, modify the environment and use the available sensors (or build their own). It is not intended to provide a real world simulation and is kept voluntarily readable, simple and extensible.

Simbad is written in java and requires only a standard Java development kit (version 1.4.2 or higher) and Sun Java 3d (version 1.3.1). This last one provides a well-documented 3D graph on which are constructed Simbads objects. Those two components are easily available for a wide range of platforms (Windows, Mac Os X , Linux, IRIX, AIX). The Simbad project is hosted at SourceForge and available at http://simbad.sourceforge.net/. It is *free to use and modify* under the conditions of the **GPL** (GNU General Public Licence).

The performances of the simulator are excellent and enable experimenters to use batch simulation in the context of heavy computation for learning (e.g. evolutionary algorithms). The 'Java is slow' remark is outdated and Java applications are now clearly equivalent to C++ ones [6] but do not require numerous external libraries. The simulator provides the following functionalities:

- Single or multi-robot simulation.
- Color Mono-scopic cameras.
- Contact and range sensors.
- Online or batch simulation.
- Python scripting with jython.
- Extensible user-interface.
- Simplified physical engine

Table 1 is a brief comparison of Gazebo, Webots and Simbad functionalities.

To sum it up, Simbad is very useful, and has proven to be so, for studying models in middle size projects as well as teaching in AI and Robotics classrooms while Gazebo and Webots are very good options if you are involved on a long-term project with real robots.

As for implementation, the user only provide an environment derived from the EnvironmentDescription class and a robot controller derived from Robot class. This last one has an initialisation method (*initBehavior*) and a method to be called on each simulation step (*performBehavior*). The simulator then execute the motor control orders in a similar way as a real robot controller (i.e. repetitive calls to the micro-controller). The following code shows how to create a simulator, settle the environment with a single box in it and create a robot in the environment (Figure 1 shows a comparable yet more complex environment):

```
import simbad.sim.*; import simbad.gui.*; import javax.vecmath.*;

public class MyEnv extends EnvironmentDescription {
```

Table 1. Comparaison between available simulators

	Player/Gazebo	Webots	Simbad
3D simulation	yes	yes	yes
3D vision	yes	yes	yes
multi-robots	yes	yes	yes
Physics	ODE	ODE	built-in
Source available	yes	no	yes
Freeware	yes	no	yes
Ease of use	needs expertise	quite good	easy
Platforms	win/linux/mac	win/linux/mac	win/linux/mac
Real Robots	yes	yes	limited

```
public MyEnv() {
 /* create four walls and the robot */
 Wall w1 = new Wall( new Vector3d(9, 0, 0), 19, 1, this);
 w1.rotate90(1);  add(w1);
 Wall w2 = new Wall( new Vector3d(-9, 0, 0), 19, 2, this);
 w2.rotate90(1);  add(w2);
 Wall w3 = new Wall( new Vector3d(0, 0, 9), 19, 1, this);
 add(w3);
 Wall w4 = new Wall( new Vector3d(0, 0, -9), 19, 2, this);
 add(w4);
 add(new MyRobot( new Vector3d(0, 0, 0), "my robot"));
}

public class MyRobot extends Robot {
 MyRobot (Vector3d _position, String _name )
 {    super (_position, _name);   }

 public void initBehavior()
 {  /* your init code goes here */}

 public void performBehavior()
 { /* code perfomed on each step goes here */  }
}

public static void main(String[] args)
{ Simbad frame = new Simbad( new MyEnv(), false); }
}
```

Simbad can also be used with Python, a very popular scripting language[3].

As a tutorial, a list of examples with sources is available from the menu in Simbad that gives direct access to many of Simbad features. Examples are:

- BaseDemo : demo with camera sensor, sonars and bumpers. The robot wanders and stops when it collides.
- ImagerDemo : shows how to capture the camera image , process it and display it in a dedicated window.

[3] The above example written in Python can be found on the web site.

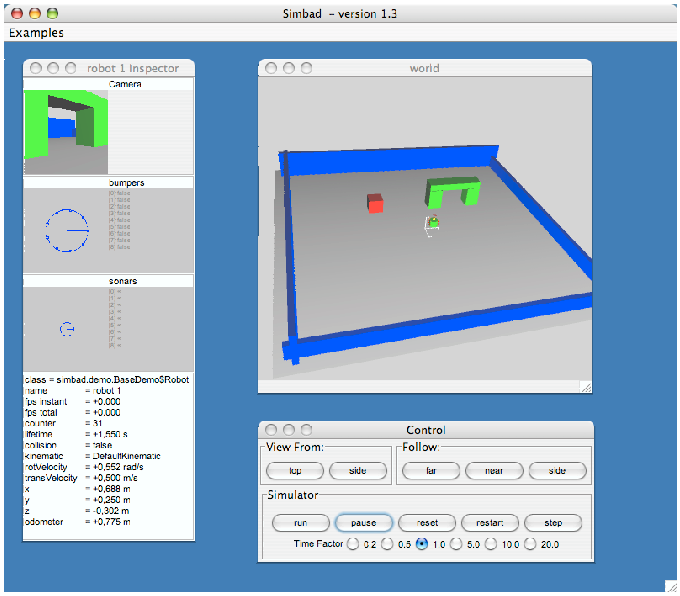

Fig. 1. Simbad interface - Simbad can be run either as a standalone program (as shown) or in batch mode with no display when simulation speed is crucial (e.g. Evolutionary Robotics setup).

- AvoidersDemo : shows several robot with sonars and bumpers performing a collision avoidance behavior.
- BlinkingLampDemo : the lamp on each robot blinks when the robot is approaching an obstacle.
- BumpersDemo : several robots bumping.
- DifferentialKinematicDemo: a differential drive (two wheels) kinematics demo.
- KheperaDemo : a Khepera robot demo.
- LightSearchDemo : robots search a light using light sensor.
- PickCherriesDemo : show a robot picking cherries. When touched, the cherries are removed.
- PushBallsDemo : shows a robot pushing balls.

3.2 The Neural Network Library : PicoNode

In the field of autonomous robotics, the seminal work of Rodney Brooks on Reactive Robotics and Subsomption Architecture [8] have lead the way toward a vast amount of works in the field of machine learning and robotics where the goal is to build the control function defined as : $f(sensoryinputs\{, internalstate\}) \rightarrow motorcontrol$ where the optional term *internal state* may represent some kind of memory. Popular learning approach for robot control includes Reinforcement Learning [9], Learning in Bayesian and Markov Model and Learning in Neural Network [7].

Neural Networks provide a very powerful representation framework in the scope of robot control due to their ability to handle continuous noisy data with little computation. Moreover, internal states can be represented in recurrent architectures and the important literature in this field attests for a wide range of possible controllers.

The PicoNode library provides a general graph-based representation framework along with two implementations: feed-forward and recurrent neural networks. The use of PicoNode is not limited to robot control; it has been designed so as to ease building of simple (e.g. multi-layered perceptrons) as well as less simple (e.g. N-layers recurrent nets) neural networks.

The following code illustrates how a simple feed-forward neural networks with 2 hidden units, 2 input nodes and 1 output node is built:

```
/* STEP 1 : Initializing and building a neural net */

// step 1a : create a network
FeedForwardNeuralNetwork network = new FeedForwardNeuralNetwork(
     new ActivationFunction_logisticSigmoid() );

// step 1b : create some neurons
Neuron in1 = new Neuron( network,
 new ActivationFunction_logisticSigmoid());
Neuron in2 = new Neuron( network,
 new ActivationFunction_logisticSigmoid());
Neuron hidden1 = new Neuron( network,
 new ActivationFunction_logisticSigmoid());
Neuron out1 = new Neuron( network,
 new ActivationFunction_logisticSigmoid());

// step 1c : declare I/O neurons
network.registerInputNeuron( in1 );
network.registerInputNeuron( in2 );
network.registerOutputNeuron( out1 );

// step 1d : create the topology (random weight values)
network.registerArc( new WeightedArc( in1 , hidden1 ,
                     Tools.getArcWeightRandomInitValue()));
network.registerArc( new WeightedArc( in2 , hidden1 ,
                     Tools.getArcWeightRandomInitValue()));
network.registerArc( new WeightedArc( hidden1 , out1,
                     Tools.getArcWeightRandomInitValue()));

// step 1e : initialize the network (perform some integrity
// checks and internal encoding)
network.initNetwork();

/* STEP 2 : using the network (feed-forward signal) */

// step 2a : loading the input values (i.e. sensory inputs)
ArrayList inputValuesList = new ArrayList();
inputValuesList.add(new Double (0.5));
inputValuesList.add(new Double (0.5));

// step 2b : computing the output values (i.e. motor outputs)
network.step( inputValuesList );
System.out.println("Output value : " + out1.getOutputValue());
```

Of course, recurrent architectures may be defined as well by just adding recurrent arcs and use the available RecurrentNeuralNetwork object instead of FeedForwardNeuralNetwork.

3.3 The Evolutionary Framework : PicoEvo

PicoNode provides standard supervised learning algorithm (e.g. Back-Propagation), however such learning algorithms are usually of little use[4] in the context of sparse, noisy, delayed and asynchronous reinforcement signals that are usually part of the task of control learning in mobile robotics (e.g. a binary reward (success/failure) is provided only when the robot may reach the exit of a maze).

The Evolutionary Robotics [7] approach addresses this problem by relying on population-based stochastic optimisation algorithms, i.e. evolutionary algorithms. Such algorithms are particularly well fitted when the objective function (i.e. the task) is difficult to describe. These stochastic optimisation algorithms perform on a generational basis (i.e. optimisation at step i depends on step $i-1$) and rely on the exploration of the space of possible solutions through a population of candidate solutions by combining selection operators (most fitted candidates are likely to survive from one generation to another) and ideally well-suited variation operators (candidates may be recombined and/or altered to diffuse supposedly good characteristics as well as to efficiently explore the search space). These algorithms have been shown to be very efficient and to achieve human-competitive results on numerous problems where standard learning algorithm are difficult to apply, which is a key advantage in robotics[5].

The second advantage of Evolutionary Algorithm consists in that such algorithms can deal with a wide range of representation formalism to be optimised, from bit vectors to tree and graphs (and thus, programs). In the scope of neural networks optimisation, it is then possible to optimise network weights (see next section) or even the whole network topology [13]. It is important to notice that even if only neural network optimisation is addressed in the scope of this paper, other representation may be optimised as well (e.g. Bayesian network, markov models, etc. – where topology learning is often an issue).

The PicoEvo library is a general Evolutionary Algorithm library that embeds several kinds of algorithms such as Genetic Algorithm, Evolutionary Strategies, tree-based and graph-based Genetic Programming. As for PicoNode, it has been conceived to ease the implementation of new operators. Then again, implementation has been done in such a way that the underlying concepts are straightforward to understand, even if the user has little knowledge in Evolutionary Algorithm (e.g. during an AI course). The typical main loop for evolving a population of vectors containing bit values is the following (the following example concerns the classic max-one problem - the reader may refer to [10] for details on the task and parameters):

[4] Some architectures, such as the auto-teaching network [11] and AAA [12], do nevertheless rely on both back-propagation and evolution.

[5] note that the Simbad simulator can be launched in fast-mode when performing such experiment, i.e. no user interface, so as to compute simulation much faster than human real-time mode – indeed, Simbad has been benchmarked to run at about 15000 steps per second on a Pentium 2.8ghz with 512mo RAM under Linux (knoppix) and Windows XP for an experimental setup involving evolution of neural network weights.

```
/* STEP 1 - INITIALISATION */
// create an Evolution environment with a single Population

// the parameter container (may be load from file)
ParameterSet_Evolution_mulambdaES parameterSet =
  new ParameterSet_Evolution_mulambdaES();

// setup evolution parameters
parameterSet.setGenomeSize(512);
parameterSet.setMu(5);
parameterSet.setLambda(195);
parameterSet.setMutationRate(1.0/512.0);
parameterSet.setMuPlusLambda(true);
parameterSet.setGenerations(250);
parameterSet.setInitPopSize(200);

// setup evolution operators
parameterSet.setSelectionOperator(
 new SelectionOperator_MuLambda("MuLambda"));
parameterSet.setEvaluationOperator(
 new EvaluationOperator_MaxOne_StaticArray_Bit("MaxOne"));
parameterSet.setPopulationInitialisationOperator(
 new InitialisationOperator_Population_SimplePopulation("MaxOne"));
parameterSet.setIndividualInitialisationOperator(
 new InitialisationOperator_Individual_StaticArray_Bit("MaxOne"));
parameterSet.setElementInitialisationOperator(
 new InitialisationOperator_Element_StaticArray_Bit());
parameterSet.setPopulationStatisticsOperator(
 new StatisticsOperator_Population());

World myWorld = new World ("myEvolution", parameterSet);
Population_SimplePopulation maxOnePop =
  new Population_SimplePopulation("max-one population",myWorld);
myWorld.registerPopulation(maxOnePop);

maxOnePop.performInitialisation();

/* STEP 2 - RUNNING */

for ( int i=0 ; i!=myWorld.getTemplate().getGenerations() ; i++ )
    myWorld.evolveOneStep(true);
maxOnePop.displayInformation();
```

Of course, PicoEvo may also be considered as a stand-alone module for non-robotics-related optimisation tasks (e.g. classic examples are provided such as max-one, symbolic regression, etc.).

4 Experiments

In this section we detail two experiments that illustrates the use of the Simbad Package. The first experiment is a straightforward use of the Simbad simulator and relies on a classic wall-avoidance and random wandering behaviour. The second experiment involves the PicoNode and PicoEvo packages in order to automatically build controllers for the same wander/wall-avoider task.

4.1 Experiment 1 : a simple hand-written wander behavior

In this experiment, we have implemented a simple wander behaviour. The robot we consider is equipped with Infrared and bumper sensors and two motor commands: directional and translation velocities. The code is the following:

```
public void performBehavior() {
 if (bumpers.oneHasHit()) { setTranslationalVelocity(-0.1);
  setRotationalVelocity(0.5-(0.1 * Math.random())); }
 else
  if (collisionDetected()) {
   // stop the robot
   setTranslationalVelocity(0.0);
   setRotationalVelocity(0); }
  else
   if (sonars.oneHasHit()) {
    // reads the three front quadrants
    double left = sonars.getFrontLeftQuadrantMeasurement();
    double right = sonars.getFrontRightQuadrantMeasurement();
    double front = sonars.getFrontQuadrantMeasurement();

    // if obstacle near
    if ((front  < 0.7)||(left  < 0.7)||(right  < 0.7)) {
     if (left < right)
      setRotationalVelocity(-1);
     else
      setRotationalVelocity(1);
     setTranslationalVelocity(0);
   } else {
    setRotationalVelocity(0);
    setTranslationalVelocity(0.6); }
  } else {
   setTranslationalVelocity(0.8);
   setRotationalVelocity(0); }
 }
```

As shown here, accessing sensor and motor information is very intuitive and complex behaviours may be implemented on this basis. In this setup, this code is executed 20 times per "virtual"second[6] and the simulator impact the motor commands on the robot. An example of the resulting trace is shown in figure 2.

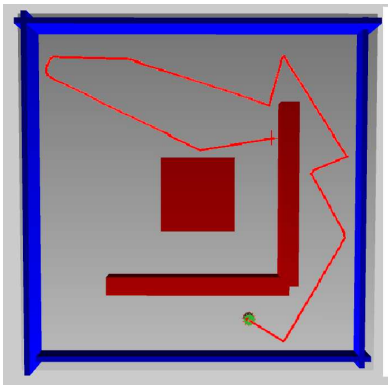

Fig. 2. trace for hand-written wander behavior

[6] As stated before, "virtual" simulation time can be accelerated more than 1000 times depending on the machine used.

4.2 Experiment 2 : evolving a wander behaviour

In this experiment, we use a Multi-Layered Perceptron with four sensory inputs (four quadrants of an Infrared sensors belt) and two motor outputs (translational and rotational velocity), i.e. same as before except that bumpers are not used. Problem properties and parameters are : 11 neurons (4 inputs, 4 hidden, 2 outputs, 1 bias) fully connected (i.e. 26+6 = 32 weights) ; 20 individuals, (2+18)-ES, mutation rate is 0.1 ; in the same environment as previous experiment (see fig. 2). The objective for a given robot is optimal if the robot does not hit walls, maximise translation, minimise rotation and minimise wall proximity. That is :

if hit wall : $fitness_{robot_i} = 0$ else : $fitness_{robot_i} = \sum(translation_{speed} + (1 - rotational_{speed}) + (1 - max(IR_{value}))$

Results are shown in fig. 3 – Due to the task simplicity, the best individual quickly achieve near-optimal control and performance keeps on improving over time. This is of course a very basic, yet classic, experiment. This basic Evolutionary Robotics introduction task is taken from the classic reference [7] to illustrate the simultaneous use of Simbad, PicoNode and PicoEvo.

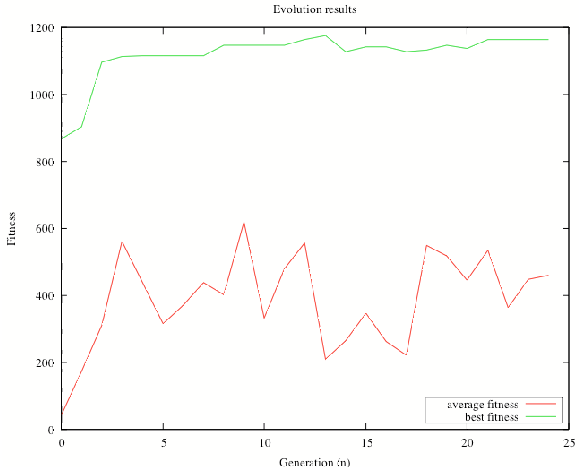

Fig. 3. Evolving a NN-based wander behaviour : Results

Due to the necessarily limited size of this paper, only simple experiments are described to illustrate the use of the Simbad package. However, the reader may refer to [14,15] which present Evolutionary Robotics experiments performed using elements from this package.

5 Conclusion

In this paper, we have presented the Simbad package for mobile robot simulation and evolutionary robotics. This package provides three stand-alone important components:

- Simbad : an extensible mobile robot simulator which can handle complex 3D scene and physics-based interaction;
- PicoNode : a library for graph-based controller such as (but not limited to) feed-forward and recurrent neural networks;
- PicoEvo : an Evolutionary Algorithm library which includes Genetic Algorithm, Genetic Programming (tree and graphs) and other popular approach in the field of population-based stochastic optimisation algorithms;

The Simbad package provides a powerful and easy-to-use framework written in Java for researchers and teachers in Evolutionary Robotics. To date, several works in the field of Evolutionary Robotics have already been achieved using the Simbad package [14, 15] and current works relies on this package. Moreover, the Simbad package is also used in the scope of Education in the last year of IFIPS, the engineering school at Universite Paris-Sud (France), as well as at Ecole Polytechnique (France), for a set of courses and projects on Reactive and Evolutionary Robotics on simulated and real Khepera robots.

The Simbad package is freely distributed with sources and provides an ideal solution for both researchers and teachers in the field of Autonomous and Evolutionary mobile robotics. Current on-going works on the Simbad Package includes extensions of the Simbad simulator towards easy switching capability from simulation to in situ robotics and of the PicoNode and PicoEvo libraries to other machine learning algorithms (Reinforcement Learning, topology learning in bayesian network for robot control, dynamical systems modelling, etc.).

Acknowledgment

The authors would like to thank Cedric Hartland, Justine Lebrun and Thomas Delquie for their contributions to the Simbad framework. This work was supported in part by the PASCAL Network of Excellence and by a CNRS TCAN grant (Nerobot project).

References

1. Simbad Simulator - http://simbad.sourceforge.net
2. B. Gerkey, R. T. Vaughan and A. Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. Proceedings of the 11th International Conference on Advanced Robotics, 2003.
3. N. Koenig and A. Howard. Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Sendai, Japan, 2004.
4. Webots - http://www.cyberbotics.com/

5. http://www.lri.fr/∼mary/WoB
6. P.Lewis and U. Neumann. Performance of Java versus C++. http://www.idiom.com/ zilla/Computer/javaCbenchmark.html
7. S. Nolfi and D. Floreano. Evolutionary Robotics. MIT Press, 2000.
8. R. Brooks. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, Vol. 2, No. 1, pp.14-23 1986.
9. R. Sutton and A. Barto. Reinforcement Learning. MIT Press, 1998.
10. D. Goldberg. Genetic Algorithms in search. Addison-Wesley, 1989.
11. S. Nolfi and D. Parisi. Auto-teaching : networks that develop their own teaching input. Proceedings of the second ECAL, 1993.
12. N. Godzik, M. Schoenauer and M. Sebag. Robustness in the long run : Auto-teaching vs. Anticipation in Evolutionary Robotics. Proceedings of PPSN, 2004.
13. F. Gruau. Modular Genetic Neural Networks for six-legged Locomotion. Proceedings of Artificial Evolution (EA), 1995.
14. N. Bredeche and L. Hugues. Speeding up learning with Dynamic Environment Shaping in Evolutionary Robotics. Proceedings of the fifth International Workshop of Epigenetics Robotics (EpiRob), 2005.
15. N. Bredeche and L. Hugues. Evolutionary Robotics : incremental learning of sequential behavior. Proceedings of the fourth IEEE International Conference on Developmental Learning, 2005.