

# Combining Simulation and Reality in Evolutionary Robotics

Juan Cristóbal Zagal · Javier Ruiz-del-Solar

Received: 10 October 2006 / Accepted: 1 February 2007 /  
Published online: 6 March 2007  
© Springer Science + Business Media B.V. 2007

**Abstract** Evolutionary Robotics (ER) is a promising methodology, intended for the autonomous development of robots, in which their behaviors are obtained as a consequence of the structural coupling between robot and environment. It is essential that there be a great amount of interaction to generate complex behaviors. Thus, nowadays, it is common to use simulation to speed up the learning process; however simulations are achieved from arbitrary off-line designs, rather than from the result of embodied cognitive processes. According to the reality gap problem, controllers evolved in simulation usually do not allow the same behavior to arise once transferred to the real robot. Some preliminary approaches for combining simulation and reality exist in the ER literature; nonetheless, there is no satisfactory solution available. In this work we discuss recent advances in neuroscience as a motivation for the use of environmentally adapted simulations, which can be achieved through the co-evolution of robot behavior and simulator. We present an algorithm in which only the differences between the behavior fitness obtained in reality versus that obtained in simulations are used as feedback for adapting a simulation. The proposed algorithm is experimentally validated by showing the successful development and continuous transference to reality of two complex low-level behaviors with Sony AIBO<sup>1</sup> robots: gait optimization and ball-kicking behavior.

**Keywords** AIBO robots · Evolutionary robotics · Mobile robotics · Organismically inspired robotics · RoboCup · Simulation of solid dynamics

---

<sup>1</sup>AIBO is a trademark of Sony Corporation

J. C. Zagal (✉) · J. Ruiz-del-Solar  
Department of Electrical Engineering, Universidad de Chile,  
Av. Tupper 2007,  
6513027 Santiago, Chile  
e-mail: jzagal@ing.uchile.cl

J. Ruiz-del-Solar  
e-mail: jruizd@ing.uchile.cl

## 1 Introduction

Evolutionary robotics (ER) is a methodology intended for the automatic design and generation of robots [18]. They are interpreted as autonomous artificial agents capable of developing their skills and behaviors as a result of the tight interaction and coupling with their environment. Using a genetic algorithm (GA) a population of artificial controllers or morphological descriptors, specified by artificial genotypes, evolves along several generations under behavior-based selection rules. The experimenter defines a fitness function in order to evaluate each individual's actual behavior depending on the desired task to be accomplished.

When using real robots however, there is a key constraint, which is that the individual's behavior evaluation and discrimination is a time consuming task. Therefore the controller complexity or dimensionality of search spaces is limited to the experimental time available. Another major and practical limitation is given by the short life span of existing robotic hardware, given that extensive evaluation trials may bring about important changes or even harm the robot platform.

The alternative of using simulation, with its potential for capturing different levels of organization, and speeding up learning, has been explored; however most of the reported simulators are fairly unsophisticated. Fortunately, current computing power and recent advances in graphics and rigid solid dynamic modeling libraries allow the generation of highly accurate simulators of robots and their environment. One problem is that strong bias can be introduced when designing simulation models. Brooks [3] argues, for example, that after transferring a controller generated under simulation to reality it will be very difficult to obtain a similar robot behavior, more over a great deal of effort is devoted to modeling aspects that in practice are not relevant to the behavior execution.

Transferring controllers generated in simulation to real robots is difficult, and it is considered as one of the major challenges when using ER and simulation. There are several examples such as reported by Nolfi [17] in which the behaviors developed under simulation fail to perform similarly in reality. Given that genetic search is free to explore any possibility given by the simulator in order to maximize fitness, it usually ends up exploiting its peculiarities. The evolved controllers are not directly transferable, and they usually fail in real environments. This is called the *reality gap* problem in ER, meaning that a behavior in reality yields a different evaluation than in simulation.

This work presents the *back to reality* (BTR) algorithm for ER, which allows improving automatically the coupling of robot simulator and environment along the behavioral learning process. Main novelties of this approach are that (1) simply individual fitness comparisons (of reality versus simulation) are used as a feedback to adapt the simulator model, (2) its co-evolutionary construction allows continuous robot behavior adaptation interleaved with simulator adaptation, (3) it produces simulator models which are coupled with resulting behaviors, (4) due to its simplicity it is the first method showing successful transfers of complex behaviors to real robots, and (5) it shows a remarkable learning performance as compared to other machine learning methods. The remainder of this article is structured as follows: Section 2 presents main motivations for this type of environmentally coupled simulation and descriptions of main proposals and methods given so far. Section 3 is the presentation of the proposed BTR algorithm. Section 4, illustration of the validation experiments using BTR and UCHILSIM [25, 28] for generating two behaviors with AIBO robots. Finally, Section 5, is the presentation of the conclusions and projections of this work.

## 2 Simulation and Cognition

Recent research in the cognitive neuroscience of sleep shows evidence of the role dreaming plays in neuroplasticity and cognition [10]. It has been proposed [23] that the human state of wakefulness can be interpreted as a process of *acquisition of information*, while the NREM<sup>2</sup> and REM<sup>3</sup> sleeping states respectively correspond to processes of *iteration of information* and *integration of information*. The *threat simulation theory* [20] proposes that “*dream consciousness is an organized and selective simulation of the perceptual world*”. Revonsuo argues that the adaptive function of dreaming is the simulation of threatening events and the repeated rehearsal of threat perception and avoidance skills; “*the realistic rehearsal of these skills can lead to enhanced performance*.” More fitness-enhancing aspects of dreams are presented in [6] as an extension of this theory.

Simulations, similarly as dreams for humans, can play a fitness-enhancing role for cognitive robots. If simulations are considered as part of the robot itself, they should be obtained as a result of a structural coupling of the robot and its environment. This consequence comes from the theory of autopoiesis<sup>4</sup> [24] which proposes that cognition is the operation of a system in a domain of structural coupling. Interestingly this idea is in conflict with traditional interpretations of simulation in the robotics’ literature; where simulation is commonly presented as the opposite of embodiment i.e. result of arbitrary off line design.

Studying the use of simulation, there is a first group of researchers that proposes to identify a priori a set of properties that the simulation should contain; Jakobi [13] proposed the minimal simulation approach to ER, consisting on modeling just those aspects of the simulation which are relevant for the arousal of a desired behavior and to introduce a high level of noise in the simulation and the robot model. Similarly Di Paolo [4] proposes to increase the neural plasticity of the robot controller in order to produce a smooth adaptation when downloading a robot controller to reality. Bedau [1], proposes that simulations can provide both simplicity and universality if their designers “*abstract away from the micro-details in the real system*” in order to provide “unrealistic” models that suffice to provide instances of the phenomena of interest. Then there is a second group of researchers that have identified the need of validating simulation but without proposing any formal methodology; Brooks [3] has suggested the need of some kind of interleaved procedure in order to validate the simulation by combining simulated and real robot evolution. Husbands [12] points out that: “*The simulation should be based on large quantities of carefully collected empirical data, and should be regularly validated*.” Kitano [15] suggests that detailed simulation models, or the “*physics model*,” can be a source of new scientific hypothesis, simulations should be implementations of the up to date theory and, must be validated with real data. In a third group of researchers we find only two clear methodologies for doing regular validations of the simulation model during behavior adaptation. In this group Grefenstette [8] proposes the *anytime learning* approach as a way of integrating *execution* and *learning* modules. A robot-learning module is in charge of testing new robot strategies against a simulation model of the task environment. The robot execution module controls the robot’s interaction with the environment, and includes an external monitor of environmental properties contained in the simulation. These modules

<sup>2</sup> Non-rapid eye movement.

<sup>3</sup> Rapid eye movement.

<sup>4</sup> One of the theories which best describes the concept of life (auto (self-) and poiesis (creation; production)).

interact by means of two communication channels. In the first one, a decision maker module determines when the results of the learning module can be translated into the execution module based on the performance obtained during the simulation. In the second channel, another decision maker module determines when and how the simulation model must be modified, taking into account the measured differences of simulator parameters obtained by the monitoring system. In that case the learning process is interrupted and the entire learning process must be restarted in the learning module.

As the authors state the performance of this approach relies dramatically on the design of decision makers; a large amount of knowledge is required when deciding how to update the simulation model in order to match the monitor observations. It is extremely important to determine a priori procedures for measuring environment parameters affecting the simulation, and to provide the experiment with a monitoring device that allows these measurements. In our opinion such external device is unpractical when using real robots, it is not clear how to relate the measurements of variables (such as mass, friction, body length, etc.) with the behavior execution. The experiments done with anytime learning are carried out using a simulation model in the role of the external environment and learning system. In this case, of course, generating a monitor device is just a matter of making certain variables explicit. Another policy to be determined is how to reinitialize the learning system when the simulation model has changed. This depends chiefly on the particular learning method being used. The validation experiments used for anytime learning are far too simple for those required in real robotics applications. They apply a cat and mouse simplistic two dimensional simulator, in which just the mean and standard deviation of the prey velocity are used as elements of the simulation to monitor the “reality” of another simulation.

In a similar trend Bongard [2] have proposed the *estimation exploration* algorithm consisting on evolving morphological aspects of a simulated robot by comparing recorded sensor data time series with those resulting from the target robot during behavior execution. In the experiments presented the physical robot is also simulated, but it is considered different than the simulated robot in ways unknown to the overall algorithm. The input of the algorithm is a target robot and its initial simulation.

The algorithm consists of an exploration phase in which neural network robot controllers are evolved, and an estimation phase, in which improvements to the initial simulation model are evolved using the differences of the best-evolved robot sensor time series (recorded in simulated versus target robot) as feedback. However, as the authors state “*slightly differences between the two machines (simulated and target robot) rapidly leads to uncorrelated signals,*” making extremely difficult to derive a meaningful fitness metric for searching over the simulators space. The experiments are done using a three-dimensional dynamics simulator of a legged robot barely feasible in the real world. The authors plan to extend their experiments to a real physical robot.

Table 1 shows essential comparisons of BTR with the existing methodologies that allow updating a simulator model during behavior adaptation. We consider that the existing methodologies are unpractical for real robot experiments (using an external monitor or trying to match uncorrelated data). On the other hand if we take a look at biology, the adaptation of the nervous system, and therefore the generation of illusions and dreams, is achieved indirectly by relying on behavioral fitness. BTR relies only on behavior performance comparisons in order to adapt a simulation model. This will be described in the next section.

**Table 1** Essentials of related work in combining simulation and reality in evolutionary robotics

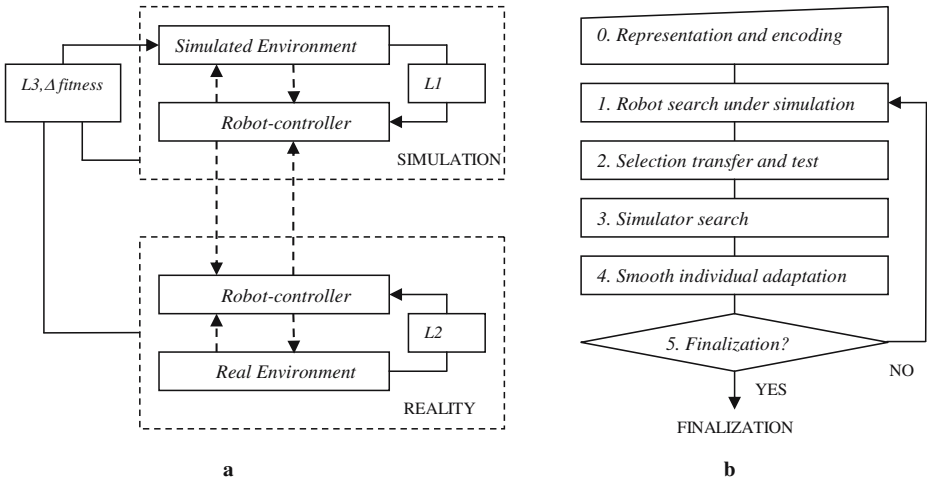
Method	1	2	3	4	5	6	7	8	9	Comments
Any time learning	×	–	–	–	–	×	–	–	–	Relies on the use of an external monitoring system, making the methodology unpractical. The adaptation of simulator is not necessarily linked to the behavior.
Estimation exploration	–	×	×	–	×	×	×	–	–	Relies on sensor data comparisons, which for slightly differences produces uncorrelated signals. This might be useful for simulated experiments but hardly realizable in reality.
Back to reality	–	×	×	×	–	–	–	×	×	Although it might suffer from disengagement, a clear methodology for preventing or solving it is given. It has been validated with real practical robot experiments.

(1): Requires an external monitoring device of simulator properties. (2): Allows uninterrupted behavioral learning. (3): Resulting simulator model is necessarily coupled with robot behavior. (4): Model feedback signal is always available. (5): Simulator requires of accurate sensor models. (6): Performance relies on behavior dependant decision makers. (7): Requires time series of sensors recordings. (8): Transfers of behaviors to real environment have been performed as a practical validation of the method. (9): Model feedback uses simply differences of behavioral fitness.

### 3 Back to Reality Algorithm

Motivated by the need of bridging the gap between simulation and reality in ER we propose an algorithm for performing regular validations and adaptations of a simulation model throughout a robot behavioral adaptation process. The idea is to co-evolve two systems; (1) the simulator (intended as a simulation of the robot and its environment), and (2) the robot (intended as the robot controller and/or its morphology). This algorithm provides the competition mechanisms that are required to produce a structural coupling between these systems as well as a practical definition of simulation model fitness. During this process real and virtual robot experiences are interleaved and mutual information is indirectly transferred across the systems achieving mutual adaptation. The robot evolution is performed in the real environment and through its simulation by maximizing a fitness function provided by the experimenter in view of the desired task to be accomplished.

For evolving the simulation this algorithm uses as fitness function the average difference of individual’s fitness (robot controller’s fitness) obtained in simulation versus their corresponding fitness resulting when evaluated in the real environment; we call this quantity  $\Delta fitness$ . This overall mechanism can be better understood when considering a robot as an embodied cognitive agent composed of a sensory-motor simulation of itself and its surroundings. In this respect a robot system is defined by two components: the robot controller and the robot simulator. As a consequence of the adaptation carried out in reality the robot controller is structurally coupled to the real environment. The adaptation occurring during the simulations couples the robot controller structurally with the simulator. Furthermore, considering the embodiment of cognition, a structural coupling of the robot controller-simulator system (simulation world) with the robot controller-reality system (real world) takes place. Note the circularity of this adaptation process, as Fig. 1a shows there are three underlying learning processes that results in the robot’s overall adaptation to its environment. First the robot maximizes its fitness by adjusting its behavior in the simulated environment ( $L1$ ), once transferred to reality, the robot adapts itself with real experience ( $L2$ ), and finally the simulator model is adapted by minimizing  $\Delta fitness$  at each iteration of



**Fig. 1** a: Three learning processes of the back to reality algorithm. b Flow diagram of the algorithm

the algorithm ( $L3$ ). In this way the simulation parameters are continuously tuned narrowing the reality-gap during the behavior adaptation process.

Note the simplicity of this approach, since measuring  $\Delta fitness$  corresponds to just measuring behavior fitness rather than performing any explicit measurement of simulation related variables. In this respect our approach does not require us to clarify any simulation opacity, but to merely measure individual performances, in the same way as implicit biological adaptation. It is neither required to measure an extensive set of sensor data recordings, as proposed by Bongard [2], nor to monitor a set of external explicit variables by means of an external monitoring device, as proposed by Grefenstette [8]. Figure 1b shows a flow diagram of the algorithm steps, a detailed description of each step of the algorithm is as follows:

*Step 0 – Representation and encoding* The vector domains where the search will be conducted should be defined in view of the parameters considered relevant by the experimenter (such as mass, force, gravity, limb lengths for a simulator or any controller parameter for the robot). This involves making a clear distinction between what is assumed to be known about a system and what is left to be determined by this algorithm. The valid range in which the search shall be performed should be defined for each parameter. Then a simulation/robot is represented by a vector  $s/r$  in the space  $S/R$  of possible simulators/robots. Another relevant setting is the encoding scheme to be used, since genetic search will be performed in the discrete space of points that can be represented by the given encoding, it is important that the experimenter defines a proper number of bits for each parameter. Once this information is provided it is possible to generate a bit genome representing the  $s/r$  vectors.

*Step 1 – Robot search under simulation* Genetic search is used in this step for widely exploring the search space  $R$  of possible robots. In the case of the first iteration, a population of  $M$  robot individuals is generated at random or as biased by an initial known robot solution  $r_0$ . For the remainder iterations the best existing robot solution  $r_{i-1}$  obtained from step 4 is used in order to bias the population. The amount of generations during which genetic search is conducted depends upon two major factors: (1) the drift tendency of the

particular problem, and (2) the cost of performing real behavioral evaluations. Drift or disengagement is a pathology that sometimes arises in co-evolving systems; Ficici presents a detailed explanation [5]. It will happen for example when all the robots in a given population completely fail to be transferred to reality (showing fundamental discrepancies of behavior or fitness measure, such as a robot that falls during a walking experiment), in this case the gradient will be undetermined and the selection pressure of simulators will have no influence on the competence of robot behaviors. We can estimate the degree of disengagement by measuring the rate of individuals that fail to be transferred to reality in each iteration of the algorithm. In order to prevent this, or in case it is detected; the amount of generations during which this step is carried out must be reduced. On the other hand if the cost of performing real behavioral evaluations is high it will be desirable to extend the amount of generations carried out in this step. The experimenter should balance both factors. During the experiments presented in this paper we have chosen to use extensive learning periods under simulation given that we have not found major drift problems.

*Step 2 – Selection, transfer and test* The individuals from the last generation of step 1 are sorted in order of descending fitness. Starting from the best individual, they are transferred and evaluated in reality until  $m$  ( $m < M$ ) individuals are found to exhibit a non-zero fitness in the real robot. Corresponding fitness values  $f_{rk}$  ( $r$ : reality;  $k = \{1, \dots, m\}$ ) are stored for the next step. As previously discussed it is possible that an individual showing high fitness under simulation presents a dramatically low fitness when tested in reality. Such an individual will be discarded and the selection will continue with the next ranked individual. If it is not possible to find  $m$  transferable individuals from the last generation, they will have to be taken, in descending order, from the previous generations obtained in step 1. However, according to our experience this is a rare situation. The amount  $m$  of individuals being transferred plays an important role. The more individuals evaluated, the more information is implicitly passed through  $\Delta fitness$  to the simulator selection process. The genotypic diversity among individuals brings about slightly different robot behaviors (phenotypic diversity), each capturing different pieces of information relevant for the simulator adjustment. On the other hand if too many individuals are used, then individuals with low fitness will be selected from the population; thus the selection pressure will be set on undesirable characteristics of the simulator for the instigation of the best behaviors. In our experiments the GAs settings (mainly the probability of mutation) are controlling the diversity among population individuals. An alternative would be to explicitly incorporate a diversity factor in the individual's fitness function; however this alternative has not been investigated in the presented work.

*Step 3 – Simulator search* In this step GAs are used (similarly as in step 1) in order to widely search the space  $S$  of possible simulators. The best existing simulator solution  $s_{i-1}$  (at iteration  $i-1$ ) is used in order to bias a population of  $L$  simulator individuals. In the case of the first iteration this population is generated as random or as biased by a known starting simulator solution  $s_0$ . The optimization is steered towards minimizing  $\Delta fitness$ . For each realization of a simulator individual the set of  $m$  robot individuals selected in step 2 are re-evaluated under simulation computing their corresponding fitness  $f_{sk}$  ( $s$ : simulation;  $k = \{1, \dots, m\}$ ) and then computing  $\Delta fitness$  as expressed in Eq. 1. This procedure might appear expensive since evaluating each simulator individual's involves evaluating  $m$  robot behaviors, however, this stage is done in simulation and can be distributed in several

computers if required. In the presented experiments the evolution (in this case minimization) of  $\Delta fitness$  is performed until 20% of its initial value is obtained.

$$\Delta fitness = \frac{1}{m} \sum_{k=1}^m (f_{sk} - f_{rk}) \quad (1)$$

*Step 4 – Smooth individual adaptation* In this step the best robot solution obtained in step 1, which was successfully validated in step 2, is further improved in reality by carrying out smooth adaptations. The idea is to extract further information from real experiences in order to better tune the solution generated under simulation. Since the main goal in this step is to perform smooth adjustments of the solution, other learning mechanism can be used besides GAs, such as policy gradient reinforcement learning. The solution  $r_i$ , obtained in this step can be then used in order to bias the population of step 1 for the next iteration.

*Step 5 – Finalization* The algorithm finishes if (1) the robot behavior requirements are met, (2) certain number of iterations defined by the experimenter have been carried out or (3) there is a substantial disengagement in one of the iterations. The latter case indicates a failure in the procedure; an alternative is to go back some generations and repeat the last evaluations. Such a situation has not, however, surfaced during our experiments. If none of the previous conditions are met then the algorithm continues by proceeding to step 1, biasing the population with the best existing robot individual  $r_i$  obtained in step 4 and using the simulator  $s_i$  obtained in step 3.

As will be experimentally shown, this procedure allows to successfully and constantly transfers behaviors to reality; we have decided to title it “*Back to Reality (BTR)*” algorithm. We have presented some preliminary drafts of this procedure in [26, 27]. The proposed methodology should also allow evolving the robot morphology. Such type of experiments could be done by using a simulated reality; using a pair of simulators in which characteristic from one simulator are being discovered by the other parallel simulator. However, this is not yet tested since the focus of this work is on learning behaviors with real robots.

In the following sections, we will establish that this procedure allows solving complex problems (e.g. gait optimization) more efficiently than most of the commonly used methods, also being able to resolve problems, which due to their complexity, have not been solved by others (e.g. learning a ball-kicking behavior).

#### 4 Generating Complex Robot Behaviors

Here we describe experiments which by means of the BTR algorithm continuously validate and update a simulation implemented using the UCHILSIM simulator [25, 28]. We present the evolution of two different robot behaviors for Sony AIBO robots. These behaviors are quite relevant in the context of a soccer team competing in RoboCup<sup>5</sup>.

<sup>5</sup> RoboCup is the world’s premier annual event in adaptive multi-agent systems ([www.robocup.org](http://www.robocup.org)).

#### 4.1 Evolving AIBO Gaits

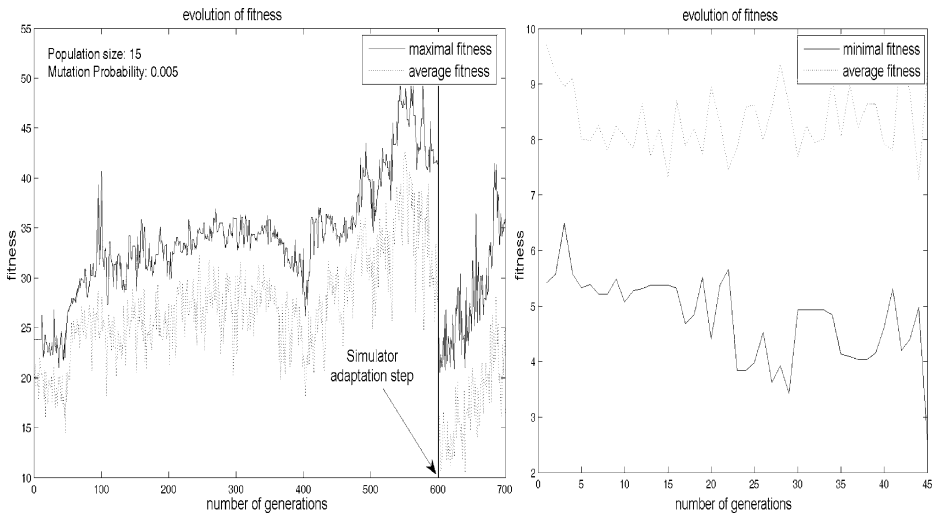
Since we are competing in the four legged league of RoboCup, we are motivated to improve the gait speed of our AIBO robots; there is a strong correlation between the speed of robot-players and the success of a robot soccer team. The problem of improving AIBO robots gaits with real robots has been addressed by several authors. We will review and compare the various performances. Note that there are no reported experiments on the evolution of AIBO gaits in simulation with successful transfers to reality. In this context we would like to test the capabilities of BTR in order to produce interesting behaviors comparable or better than those resulting from other approaches.

As a behavior fitness measure we use the robot speed measured in centimeters per second during evaluation trials of 20 s. The following set of 20 parameters defines the AIBO's gait in our experiments: the locus shape (three parameters: length, shape factor and lift factor.); the front locus shape modifier (three parameters: lift height, air height and descending height); the rear locus shape modifier (three parameters: lift height, air height and descending height); the front locus operation point (three parameters:  $x$ ,  $y$  and  $z$ ); the rear locus operation point (three parameters:  $x$ ,  $y$  and  $z$ ); locus skew multiplier in the  $x$ - $y$  plane (for turning); the speed of the feet while on the ground; the fraction of time each foot spends in the air; the time spent on the lift stage (its equal to the descending time); the number of points in the air stage of which the inverse kinematics is calculated.

The robot simulator is defined by a set of 12 parameters, which determine the simulator and robot dynamics model. There are four parameters for the mass distribution in the robot: head mass, neck mass, body mass and leg mass; four parameters of the dynamic model: friction constant, gravity constant, force dependent slip in friction direction 1 and force dependent slip in friction direction 2; and finally four parameters for the joint leg model: proportional, integral and differential constants of the PID controller and maximum joint torque.

Specifically, a conventional GA employing fitness-proportionate selection with linear scaling, no-elitism scheme, two-points crossover with probability  $P_c=0.7$  and mutation with probability  $P_m=0.005$  per bit was used.

This experiment was performed according to BTR algorithm as follows: First iteration, step 1: Evolution of the 20 robot controller parameters was carried out using genetic search along 600 generations under simulation, this allowed the extensive exploration of the gait solution space. A population size of 15 individuals per generation was used. As starting point we used one of our hand tuned solutions (10 cm/s), although our approach is also suitable when starting from scratch as it is described in the next experiment. The resulting evolution of maximal and average fitness is shown in Fig. 2 (left). Individuals under simulation immediately obtain larger fitness than in reality, this is clearly due to the initial discrepancies between simulation and reality. Step 2: Individuals from the last generation of step 1 were transferred and tested in reality until 10 individuals were found to exhibit non-zero fitness. In this case a few individuals (such as the best resulting from simulation) were falling down when tested in reality (producing zero fitness) and thus, according to our algorithm were discarded. The group of 10 successfully transferred individuals averaged a speed of 18 cm/s, the best individual of this group performed 20 cm/s in reality. The maximum resulting  $\Delta fitness$  was 10 cm/s. The fitness of each individual was measured by averaging three evaluations with real robot, therefore a total of 30 evaluations were performed in this step. Step 3: The evolution of the 12 simulator parameters was performed by minimizing  $\Delta fitness$  and re-evaluating each one of the 10 individuals under each one of the simulation individuals. A population size of 20 individuals per generation was used for



**Fig. 2** *Left*: Fitness evolution of individuals tested with the UCHILSIM simulator. Before adapting the simulator, the individuals obtain larger fitness in simulation than in reality. After the adaptation of the simulator (performed at generation 600), the fitness of individuals in simulation is reduced and fits better reality. *Right*: adaptation of the simulator, it is possible to observe how the minimization of the average differences of fitness obtained in reality versus simulations takes place

the evolution of the simulator. As Fig. 2 (right) indicates, the best evolved simulator resulted with  $\Delta fitness$  of 2 cm/s. Step 4: The best individual from step 2 was taken as a starting point for a policy gradient reinforcement learning [16] adaptation carried out in reality. In this case we were interested on performing smooth transitions of the solution. We used 13 variations of the policy (solution) along 10 repetitions with a total of 130 evaluations with real robots. The best individual resulting from this stage got a fitness speed of 23.5 cm/s. Step 5: It is decided to continue to the next iteration. Second iteration: The best robot individual from step 4 of previous iteration is taken back to the best available simulator that resulted in step 3 from previous iteration, and then the procedure is repeated from step 1. This time the step 1 is executed along 100 generations; corresponding evolution of fitness under simulation is shown in Fig. 2 (left). Step 2: The 10 successfully transferred individuals were tested three times each. During this stage one of the best evolved individuals got a fitness of 24.7 cm/s in reality. The overall procedure up to this point involved 190 evaluations with real robots.

It should be noticed that these improvements were achieved using our own AIBO controlling software (UChile1 package [22]), and therefore, the resulting speed is not directly comparable to that obtained by others. In addition to the gait locus used by the robot controller, other important matters are the efficiency in the computations, the low level refinements in motor control, the inverse kinematics models being used, etc.

#### 4.2 Learning Performance Comparisons

Several machine learning methods have been applied to the AIBO gait optimization problem; Table 2 presents a summary of methods and their performances. Initially the researchers from Sony Corporation, in the seminal work of Hornby [11], applied GAs to

**Table 2** Performance comparisons of machine learning methods applied to gait optimization on AIBO robots

Method and authors	Num par adj: <sup>a</sup>	Rob mod: <sup>b</sup>	Num indiv gen: <sup>c</sup>	Tot num gen: <sup>d</sup>	Tot eval rob: <sup>e</sup>	Tot num eval sim: <sup>f</sup>	Tim inv per rob: <sup>g</sup>	Fit imp (cm/s): <sup>h</sup>	Imp factor:(IF) <sup>i</sup>	Learn perf factor <sup>j</sup> (LP)
GA Trot gait	20 r	Prototype	30×3	21	1,890	–	21 h	4.5–10	0.55	0.582
Hornby et al. 1999 Pace gait	20 r		30×3	11	990	–	11 h	8.7–12	0.275	0.556
GA Golubovic and Hu 2002	13 r	ERS110	20	50	1,000	–	8 h	4–11.7	0.658	0.855
PGRL Kohl and Stone 2004	12 r	ERS210	15×3	23	1,035	–	9 h	24.5–29	0.155	0.180
EHCLS Quilan et al. 2003	11 r	ERS210	1	80	80	–	2 h	24.9–29	0.141	1.939
GA Röfer 2004	26 r 26 r	ERS210 ERS7	15×3 15×3	48 48	2,160 2,160	– –	4 h 4 h	23–31 30 <sup>k</sup> –40	0.258 0.25	0.310 0.301
Powell Direction Set Kim and Uther 2003	24 r	ERS210	24	24	576	–	5 h	22.7–27	0.159	0.663
BTR (r: robot, s: simulator)	20 r 12 s	ERS210	15 r 20 s	700 r 45 s	190	10,500 r 900 s	3 h	10–24.7	0.595	6.263

<sup>a</sup> Number of parameters under adjustment (r: robot, s: simulator). <sup>b</sup> Robot model. <sup>c</sup> Number of individuals evaluated per generation. <sup>d</sup> Total number of generations. <sup>e</sup> Total number of evaluations performed on real robots. <sup>f</sup> Total number of evaluations performed under simulation. <sup>g</sup> Estimation of the time invested per robot in real experiments (derived from data provided in corresponding publications). <sup>h</sup> Fitness improvement. <sup>i</sup> Fitness improvement factor. <sup>j</sup> Learning performance factor of the method. <sup>k</sup> Since no information is provided in corresponding publication, it is here assumed that the starting point was 30 cm/s.

develop populations of 30 gait controller individuals defined by 20 parameters, each evaluated three times over a prototype of the AIBO robot. As the starting point populations of non-falling individuals were manually selected from a random generator. It was reported that several broken parts of the robot were replaced during the experiments. This prevented researchers from outside Sony on carrying out similar studies until 2002 when Golubovic and Hu [7] started again with real robot experiments on the AIBO ERS-110 model. They used GAs to evolve 13 parameters of their gait engine employing population sizes of 20 individuals during 50 generations achieving 11.7 cm/s. In the promising work of Hardt and Stryk [9] a simulation of the AIBO ERS-210 based on a recursive multibody algorithm was used in order to optimize gait stability and speed. By using their own numerical optimization tool (DIRCOL) it was possible to obtain speeds of 67 cm/s, unfortunately this was just theoretical work done under an equational model without there being any transfers performed to the real robot and without validation of the simulation. In reality such high speed has never been achieved with an AIBO robot. In the work of Kohl and Stone [16] policy gradient reinforcement learning (PGRL) was used to improve their AIBO ERS-210 robots gait speed. Starting from their best available hand coded policy; they evaluated 15 individuals, three times each, using three robots simultaneously along 23 iterations. Kim and Uther used their implementation of the Powell Direction Set method [14] in order to improve their gait solution for the AIBO ERS-210 starting from their best 22.7 cm/s solution up to 27 cm/s. They evaluated 24 direction variations 24 times. Röfer [21] used GAs in real AIBO ERS-210 robots starting from his 2003 best hand coded solution of 23 cm/s and achieving the ERS-210 record of 31 cm/s. Following a similar methodology he achieved 40 cm/s on the newest model of AIBO ERS-7, however there is no information with respect to the starting solution used for this robot (30 cm/s are assumed). The work of Quilan [19] using evolutionary hill climbing with line search (EHCLS) is surprising in terms of efficiency. As can be derived from their report, they managed to improve a hand coded solution from 24.9 cm/s up to 29 cm/s investing just 80 evaluations in real robots.

The learning performances of all machine learning methods applied to the AIBO gait optimization problem can be compared by considering their corresponding (1) improvement in fitness, (2) the cost or number of evaluations with real robots, and (3) the complexity of the gait controller solution. A natural measure of the fitness improvement should consider the difference in fitness achieved during learning  $f_{\text{end}}$  with respect to the starting fitness  $f_0$ , divided by the maximum known fitness  $f_{\text{max}}$  of the particular gait implementation. For the methods being compared the fitness values achieved during learning correspond exactly to the maximum fitness known for each particular implementation of gait engines, thus, for our comparisons  $f_{\text{max}}=f_{\text{end}}$ . Equation 2 shows an expression of the *improvement factor* (IF) which includes a scaling factor of 100 for ease of interpretation. The learning cost or number of evaluations performed in real robots  $e$  is a fundamental consideration; according to the experience of Hornby serious damage of the robot platform can be expected at around 1000 evaluations. As Table 2 shows this scenario was probably encountered in most of the reported experiments. The complexity of the gait controller solution under search must also be taken into account. According to the Kolmogorov criterion<sup>6</sup>, the dimensionality  $d$  of a space can be considered as an estimate of the complexity of a point

<sup>6</sup> Kolmogorov complexity, also known as descriptive complexity or algorithmic entropy, is the shortest description of an object.

in such space. Equation 3 expresses our *learning performance* (LP) metric by considering the abovementioned factors in a single measure.

As it can be seen in Table 2 the BTR algorithm achieves the highest performance for the task of improving gaits with AIBO robots. Our method (1) allows the extensive exploration of the high dimensionality space of 20 parameters of our gait engine (since a high number of evaluations are performed under simulation), (2) it requires few evaluations with real robots as compared with the other methods, and (3) allows producing a considerable increase in fitness. Moreover from the last column of the table BTR shows the highest learning performance according to our LP metric. By taking this into account is possible to conclude that this algorithm solves this problem more efficiently (in terms of real robot evaluations) than all the other machine learning methods which have been applied.

$$IF = \frac{(f_{\text{end}} - f_0)}{f_{\text{max}}} \cdot 100 \quad (2)$$

$$LP = \frac{d \cdot IF}{e} \quad (3)$$

### 4.3 Learning to Kick the Ball

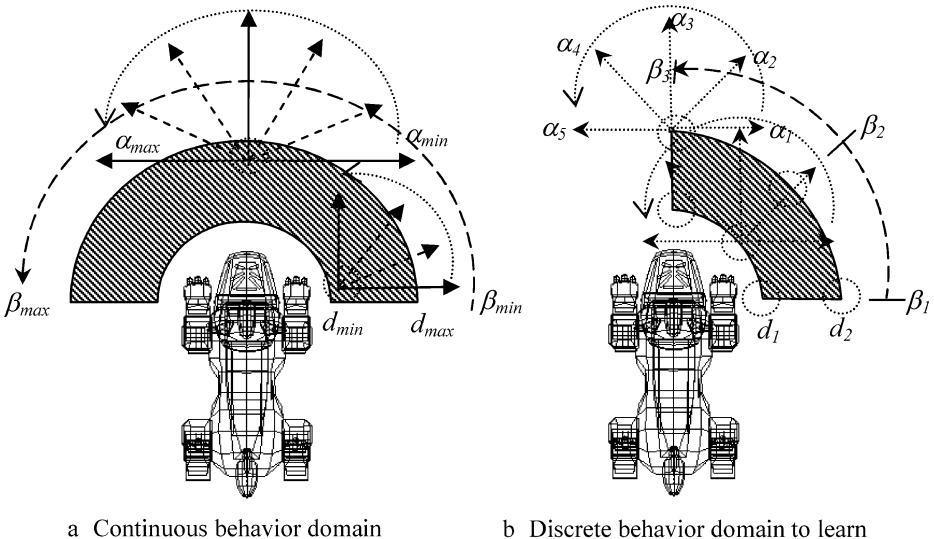
Kicking the ball with high power, short reaction time and accuracy are fundamental requirements for any soccer player. Until now all the four-legged teams use more or less the same method for kicking the ball, which works as follows: “First localize the robot in position the ball and targets, and then trigger some of the recorded ball kicks.” We consider that this approach is quite restrictive since firstly, it requires the ball to be placed relative to the robot into a discrete set of positions, the problem with this is that the robot will have to invest valuable time on repositioning itself; and second, there is only a discrete set of target directions to choose from. Human soccer players are able to kick the ball freely, i.e. from almost whatever relative position and with any target direction. Moreover within the four legged league we can notice a direct relation between the amount of ball-kicking alternatives the players have and the team success in robot soccer. Another important aspect we wish to address here is learning. Human players acquire these fine low-level sensory motor coordination abilities through extended training periods that might last for years. We believe that this type of reactive behavior can be derived from a tight interaction between the robot and the environment. However, learning to kick the ball with real legged robots seems to be an expensive and time-consuming task. Therefore we analyze here if the proposed BTR algorithm can be used in order to address this complex problem by searching widely for a solution while maintaining a simulation model adapted to reality. Our goal is to produce an optimal ball-kicking behavior for any given robot relative ball position and target direction that the robot might access without repositioning itself.

We aim at learning the best ball-kicking behavior that the AIBO robot is able to execute while introducing a minimum of designer bias. As Fig. 3a shows, the desired behavior should operate over a continuum of robot accessible ball positions and target directions. We define a ball-kicking behavior domain as a continuous three dimensional space made up by the dimensions of the starting ball distance  $d$  (in [14 cm, 18 cm]) measured from the robots neck, the robot relative ball angle  $\beta$  (in [0°, 180°]) measured from the robots right side, and the ball target direction defined by an angle  $\alpha$  (in [0°, 180°]) measured from the robots right side.

In order to generate an optimal kick for each point (task) in this domain we will consider a discrete set of points for learning, as Fig. 3b illustrates; two ball distances  $\{d_1, d_2\}$ , three relative angles  $\{\beta_1, \beta_2, \beta_3\}$  and five target angles  $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5\}$ , i.e. thirty points in total. We have selected only the right half portion of the range of  $\beta$  since we expect solutions to be symmetric around  $\beta=90^\circ$ . For each one of these thirty points, a specific ball-kicking behavior will be generated by learning the set of parameters that define the corresponding sequence of robot limb displacements giving rise to a ball-kick. The final continuous behavior for an arbitrary point, i.e. ball position  $\{d, \beta\}$  and target angle  $\{\alpha\}$ , will be obtained by interpolating, in parameter space, reaching solutions at the neighboring points. In all cases the idea is to maximize the distance traveled by the ball while maintaining high accuracy in the resulting shooting direction.

The way in which the ball-kick is parameterized might be a strong source of designer bias; therefore care must be taken to provide flexibility. We will present a ball-kick parameterization that aims to reach a compromise between simplicity for learning and flexibility for allowing several solutions. Under a physical perspective we should consider that a larger amount of energy might be transferred to the ball when all the robot limbs are allowed to move. For example, the legs supporting the body can displace the torso and its resulting ground relative speed might be added to the torso-relative speed of the limb extreme that finally kicks the ball. It is also important grant freedom as to where on the ball surface the forces are applied. This freedom might allow the robot to discover, just as an example, that kicking the ball on its upper section is more efficient for gaining speed than kicking it on their lower section (as happens in the billiard game). Notice that the UCHILSIM simulator is well suited for such high detail level.

The configuration state of an AIBO limb is defined by the angular state of its three joints  $\{v, \omega, \gamma\}$ . The course of a limb during a ball-kick might be described using hundreds of points in joint space, however we are analyzing only the case of fast reactive ball-kicks



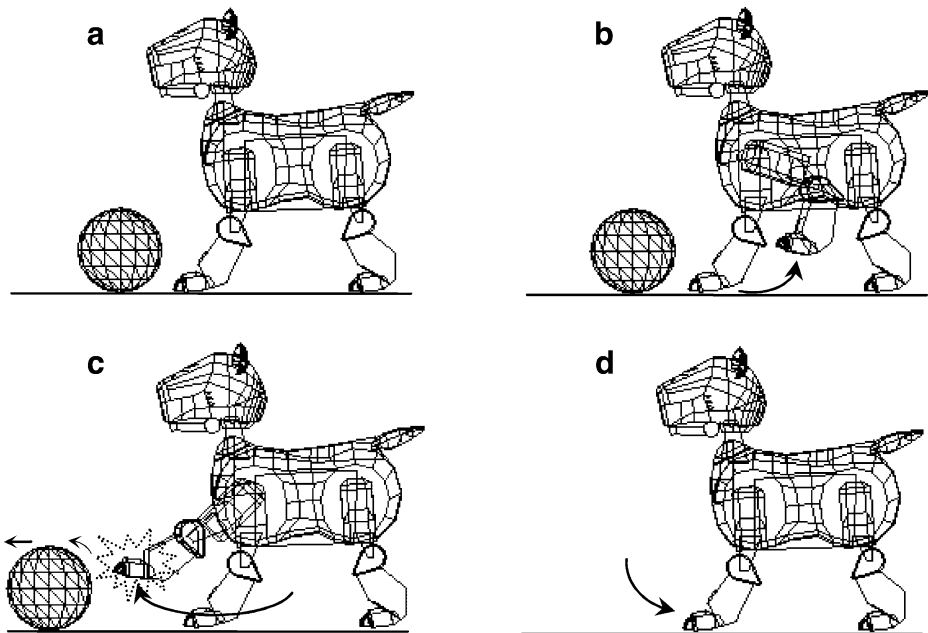
**Fig. 3** Illustration of the behavior domain defined by various robot relative ball positions and ball target angles. They are first shown **a** for the expected continuous behavior to be generated and **b** for the discrete set of points that we consider for learning

where no ball repositioning is required prior to ball-kick. We estimate that it is sufficiently good to consider just four extreme points of displacement for each limb (see Fig. 4), which are (a) the starting equilibrium limb configuration which we have selected to be the same as our walking equilibrium configuration, (b) the limb energy accumulation configuration  $\{v_{i1}, \omega_{i1}, \gamma_{i1}\}$  with  $i=[1, \dots, 4]$ , (c) the limb energy liberation configuration (ball contact instant)  $\{v_{i2}, \omega_{i2}, \gamma_{i2}\}$ , with  $i=[1, \dots, 4]$ , and (d) the ending limb configuration that we consider to be the same as the starting limb configuration.

Twenty-five controller parameters should be obtained through adaptation in this work. They correspond to six parameters  $\{v_{i1}, \omega_{i1}, \gamma_{i1}, v_{i2}, \omega_{i2}, \gamma_{i2}\}$  for each one of the four robot legs, and one parameter which controls the speed of joints by incorporating certain amount of points in the trajectory which are passed as intermediate references to the joint servo motors. A greater amount of interpolation points makes the motion of legs slower.

Each parameter is represented as an 8 bit string. We generate chromosomes of length  $8 \times 25$  for the controller and  $8 \times 12$  for the simulator. The algorithm used for the evolution is a fitness-proportionate selection GA with linear scaling, no-elitism scheme, two-point crossover with a crossover probability  $Pc=0.75$  and mutation with a mutation probability  $Pm=0.015$  per bit. The 12 parameters for the simulation correspond to those described in Section 4.1.

For any given ball position and target angle we aim to find a ball-kicking behavior that maximizes the ball displacement while maintaining high accuracy in the resulting direction. Deriving a good fitness function for this purpose might seem quite straight forward; however some considerations must be taken.



**Fig. 4** Example of the subsequent configurations that are taken by one of the robot limbs during the execution of a ball-kicking behavior. The limb starts from an equilibrium configuration **a**, second it passes to a energy accumulation configuration **b**, third it triggers the ball-kick **c** and finally the robot returns to the starting equilibrium configuration **d**. Notice that for simplicity this illustration contains the action of just one limb, however the displacement of the four legs is considered in the presented experiments

Equation 4 corresponds to the first fitness function that we used during our experiments, it corresponds to the measured Euclidean distance between the final ball position and the robot neck's position multiplied by the cosine of the error angle  $\psi$  between the resulting direction vector and the target direction vector. This function is maximized when the error is zero.

$$\text{fitness}_1 = d\cos(\psi) \quad (4)$$

This function however has shown to be useless for finding accurate ball-kicking behaviors since genetic search first concentrates on finding individuals which are able to shoot the ball at great distances without having an accurate angle. The reason is that the cosine function is quite flat around zero. In order to solve this inconvenience we choose instead an exponential function as shown in Eq. 5. By using this function we observed that genetic search first concentrates on producing the right angle and then maximizes the distance.

$$\text{fitness}_2 = de^{-k\psi^2} \quad (5)$$

A different ball-kicking behavior will be achieved for each one of the 30 points in the discrete behavior domain space. They will be obtained through genetic search over the 25 controller parameters space using the BTR algorithm. However the desired behavior should carry out the right accurate ball-kick for an arbitrary point  $\{\alpha, \beta, \delta\}$  on the continuous behavior domain. For doing this we use trilinear interpolation over the obtained points for any desired point. A requirement for this procedure to work is that neighboring grid solutions should produce a similar set of limb configurations; otherwise the resulting interpolated ball-kick will not make sense. We measure this similarity as the Euclidean distance of the parameter vectors.

For learning a particular isolated ball-kicking behavior we first randomly generate an initial population of controllers. A known initial solution was used for defining the set of 12 simulator parameters (this solution was obtained from previous experiments on learning to walk), then according to the BTR algorithm we perform the following steps: Step 1: Evolution of the 25 robot controller parameters was carried out using genetic search in simulation computing individual fitness according to Eq. 5. The evolutionary process runs first along 50 generations of 15 individuals and then it stops triggering a user alarm. Step 2: Human intervention is required in order to transfer to reality a group of 10 individuals in order of descending fitness resulting from simulation and to measure their corresponding fitness in reality. Notice that this process can be automated; however it is not the main goal of this research. Step 3: The evolution of the simulator takes place in order to minimize  $\Delta\text{fitness}$ . For this, each tested behavior is repeatedly executed using different simulator parameters derived from the genetic search process. This stage runs along 10 generations each time. Step 4: The best individual obtained on step 2 is taken as a starting point for a smooth adaptation (GA using  $Pm=0.005$  per bit) performed in the real environment for just five generations of 10 individuals. Once the simulator parameters have been updated the whole controller adaptation process continues going back to step 1, taking as a starting point the best individual resulting from step 4.

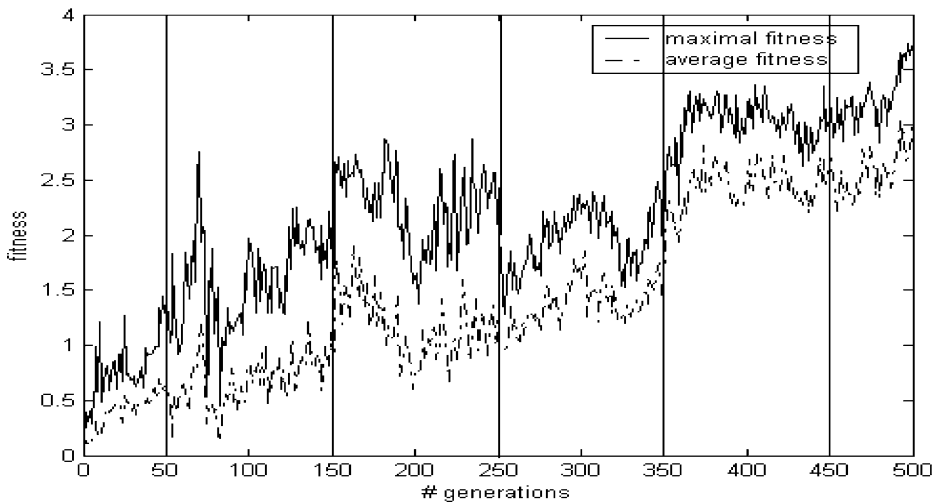
Performing real robot evaluations is expensive for this task; some evolved ball-kicking behaviors require high motor torque and might involve crashing the robot torso against the floor. We observed that the drift rate (amount of individuals transferred that fail to touch the ball) was higher during the early generations of the adaptation process. Taking this into account, we estimate that a good compromise of drift rate versus cost was obtained when

using 50 generations of genetic search in step 1 for the first iteration, and 100 generations for the remainder iterations of the BTR algorithm.

We ran this process for the 30 different points of the behavior domain grid that we aim to create. The above mentioned process was performed during 500 generations for each point. Figure 5 shows the evolution of fitness for the point  $\{d=14\text{ cm}, \beta=90^\circ, \alpha=90^\circ\}$ , i.e. the task was to kick the ball when it was placed at 14 cm right in front of the robot neck with a frontal target direction. The vertical lines represent the interruption stages where the simulator was adapted using as feedback the fitness measures of the resulting 10 best behaviors in the real environment. One can observe how updates in the simulator produce differences in the maximum and average fitness during the controller adaptation process. We observed that at the beginning of the adaptation process the behaviors obtained in the simulator were not directly transferable into reality, i.e. their corresponding fitness was quite different, however at the end of the process (beyond generation 300) the resulting behaviors were almost directly transferable to reality.

Similar observations were made in most of the remaining 29 points of the behavior domain, however learning was not successfully accomplished in some points like  $\{d=14\text{ cm}, \beta=0^\circ, \alpha=180^\circ\}$ . Certainly it is hard to imagine the robot managing to kick the ball when placed on its right side with a left side target direction!. A few neighbors to this point also failed to exhibit good results during the learning process. We consider that this is a design problem which can be easily solved by restricting the behavior domain. On the presented experiment we replaced the neighboring good solutions in those points where solution was not found (just four cases).

The resulting behaviors were quite interesting, even more so considering that they were obtained from scratch. For example the robot learned that one way of shooting the ball was to jump onto it with its torso or with its legs as well. We obtained several behaviors where the concept of “accumulating energy” was heavily exploited. Some resulting ball-kicking

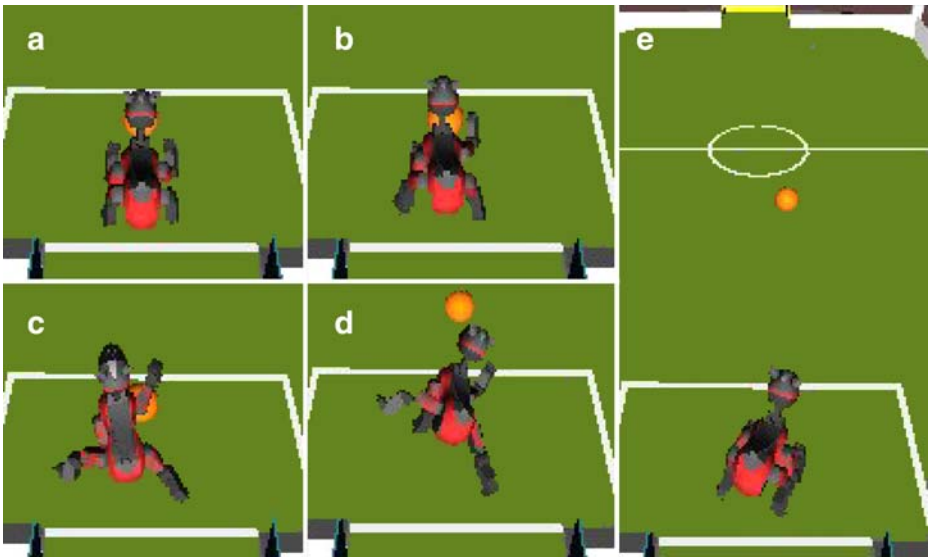


**Fig. 5** Evolution of maximal and average fitness for the adaptation of the behavior domain point  $\{d=14\text{ cm}, \beta=90^\circ, \alpha=90^\circ\}$ , i.e. the task was to kick the ball when it is placed at 14 cm right in front of the robot neck with a frontal target direction. The vertical lines represent the interruption stages where the simulator is adapted using the feedback obtained when measuring the fitness of resulting best behaviors on the real environment. It is possible to observe how updates in the simulator produce differences in the maximal and average fitness during the controller adaptation process

behaviors do not look very nice since the robot ends up in strange configurations, however they were nonetheless successful. Figure 6 shows a screenshot of UCHILSIM while executing a ball-kicking behavior.

One can directly generate a continuous behavior for any point  $\{d, \beta, \alpha\}$  using tri-linear interpolation from the resulting grid of behaviors. Unfortunately there is no guarantee of success when the points vary from the grid points. This is due to the great variety of solutions obtained. For example two neighboring points in the grid producing a similar behavior, one sends the ball at direction  $0^\circ$  and the other at  $45^\circ$ , however one uses the left leg while the other uses the right leg. Interpolating these two behaviors gives rise to a behavior which allowed the robot to just fall over without even touching the ball. Under this scenario we visualize three alternatives: (1) manually select similar solutions for the behavior grid disregarding at some point their optimality, (2) guide the evolutionary process by contaminating populations with manually selected individuals from neighboring points, and (3) enforce similarity during the learning process by redefining the fitness function. We have produced a sufficiently good continuous overall behavior by using the second methodology. Starting from the previously generated behaviors we contaminated their neighbor's populations with an 80% of individuals belonging to this successful point, and then we repeated the process over the complete domain. Although this latter approach is quite time consuming, it works well generating a continuous ball-kicking behavior over the entire domain.

It should be stressed that to the best of our knowledge, BTR is the only machine learning method that has been applied to this problem. Due to its complexity it is necessary to extensively search over highly dimensional solution spaces. Given the nature of the resulting behavior, it is a task that might easily destroy a robot when using traditional methods. In addition to the here presented autonomous learning methodology, the only



**Fig. 6** Illustration of the use of the UCHILSIM simulator for learning to kick the ball. In this case it can be seen a curious behavior where the robot first pushes its body forwards and then kicks the ball with its right leg. The order of actions is as indicated in the figures

existing alternative to address the ball kick problem is manual design. This methodology is extensively used by all four-legged RoboCup teams.

## 5 Conclusions and Projections

In this work organismically inspired robotics concepts were described, amid them we remark that robot cognition is a matter of interacting in the way a robot is able to interact with its environment as a result of its experience (structural coupling). ER researchers have used simulation to increase the amount of interaction of a robot and to speed up the robot learning process; however simulations are the result of arbitrary design. We have reviewed a set of viewpoints for the significance and use of simulations in ER, most of researchers agree on the fundamental need for validation when using simulation.

We have analyzed the use of simulation as a cognitive process in which robot environment interaction is applied to validate and update a simulator model. In particular we have proposed the BTR algorithm that allows performing regular simulation adaptations by relying only on simple fitness measurements. We have also made use of the highly realistic UCHILSIM simulator. We can conclude as a result from the presented experiments the following:

1. Using BTR and a realistic robot dynamics' simulator (such as UCHILSIM) allows generating complex robot behaviors, with regular validations and updates of the simulation model. This approach presents a significant speedup with regard to evolving on real robots.
2. The BTR algorithm allows maintaining a simulator adapted for the arousal of a particular robot behavior by relying on simple behavior fitness measurements, unlike other approaches (such as using a set of extensive sensor measurements or using complex monitoring of generally inexplicit simulator variables).
3. Different optimal simulators are achieved for different behaviors; this shows that the adaptation of the simulator depends on the particular behavior being created. Thus, the generation of behaviors guides the way in which the simulator is adapted. Another interpretation is that *different representations of the surroundings are generated for addressing different tasks*.
4. Peculiarities of the simulator exploited by genetic search over the space of robot behaviors are diminished after performing the adaptation of the simulator by minimizing the differences of behavior fitness obtained in reality versus simulation ( $\Delta fitness$ ).
5.  $\Delta fitness$  decreases during the BTR adaptation process, which indicates an increase of the structural coupling between the simulation and the real environment.
6. BTR allows solving a complex learning problem such as gait optimization using less evaluations in real robots than most of existing approaches, achieving the highest learning performance, measured using the here-defined LP metric.
7. BTR allows addressing new learning problems, such as autonomous generation of ball-kicking behaviors, which due to their complexity have not been addressed yet by other machine learning methods, and are solved using manual design.
8. This is the first work in which a simulator is used for generating AIBO gaits with successful and constant transfers to reality.
9. The presented approach is convenient in terms of hardware expenses. Since it requires few evaluations in reality, it saves the robot platform from potential damage.

We hope this work serves to remark the effectiveness of performing regular validations on simulation for ER. In this context it is of utmost importance to use highly realistic simulators and a practical feasible framework in order to combine real and virtual (simulated) experiences. Similarly as occurs in nature, the behavior feedback of an agent can be used as a tool for generating representations of the surrounding world.

In future work we plan to analyze and optimize some design options of BTR, and to employ it in the autonomous development of other highly complex behaviors for mobile robots. We also believe that this kind of tool can be of interest for the neuroscience community, because it can be employed for the practical and fully observable testing of theories on the cognitive function of dreaming. Therefore, one of the long-term goals of this research is to provide practical tools in such direction.

**Acknowledgments** This research was partially supported by FONDECYT (Chile) under Project Number 1061158.

## References

1. Bedau, M.A.: Can unrealistic computer models illuminate theoretical biology? In: Wu, A.S. (ed.) Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program, pp. 20–23. Morgan Kaufmann, San Francisco, CA (1999)
2. Bongard, J.C., Lipson, H.: Once more unto the breach: co-evolving a robot and its simulator. In: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9), pp. 57–62 (2004)
3. Brooks, R.: Artificial life and real robots. In: F.J. Varela and P. Bourgine. (ed.) Toward a Practice in Autonomous Systems. Proceedings of the First European Conference on Artificial Life, pp. 3–10. MIT Press, Cambridge, MA (1992)
4. Di Paolo, E.A.: Homeostatic adaptation to inversion of the visual field and other sensorimotor disruptions. In: J-A Meyer, A. Berthoz, D. Floreano, H.L. Roitblat, and S.W. Wilson (eds.) From Animals to Animats, Vol. 6. Proceedings of the VI International Conference on Simulation of Adaptive Behavior, pp. 440–449. MIT Press, Cambridge, MA (2000)
5. Ficici, S.: Solution concepts in coevolutionary algorithms. PhD thesis, Brandeis University (2004)
6. Franklin, M.S., Zyphur, M.J.: The role of dreams in the evolution of the human mind. *Evolutionary Psychology* 3, 59–78 (2005)
7. Golubovic, D., Hu, H.: A hybrid evolutionary algorithm for gait generation of Sony legged robots. In: Proceedings of the 28th Annual Conference of the IEEE Industrial Electronics Society, Seville, Spain, 5–8 November 2002
8. Grefenstette, J.J., Ramsey C.L.: An approach to anytime learning. In: Proceedings of the Ninth International Machine Learning Workshop, Morgan Kaufmann, San Mateo, CA (1992)
9. Hardt, M., Stryk, O.: The role of motion dynamics in the design, control and stability of bipedal and quadrupedal robots. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) Lecture Notes in Computer Science, vol. 2752, pp. 206–223. Springer, Berlin Heidelberg New York (2002)
10. Hobson, J.A., Pace-Schott, E.F.: The cognitive neuroscience of sleep: neuronal systems, consciousness and learning. *Nat. Rev., Neurosci.* 3(9), 679–93 (2002)
11. Hornby, G.S., Fujita, M., Takamura, S., Yamamoto, T., Hanagata, O.: Autonomous evolution of gaits with the Sony quadruped robot. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1297–1304. Morgan Kaufmann, San Mateo, CA (1999)
12. Husbands, P., Harvey, I., Cliff, D.: An evolutionary approach to situated AI. In: A. Sloman et al. (ed.) Proceedings of the 9th Conference of the Society for the Study of Artificial Intelligence and the Simulation of Behavior (AISB), pp. 61–70. IOS Press, Amsterdam, The Netherlands (1993)
13. Jakobi, N.: Minimal simulations for evolutionary robotics. PhD thesis, University of Sussex (1998)
14. Kim, M., Uther, W.: Automatic gait optimisation for quadruped robots. In: Roberts, J., Wyeth, G. (eds.) Australasian Conference on Robotics and Automation, Brisbane (December 2003)
15. Kitano, H., Hamahashi, S., Kitazawa, J., Takao, K., Imai, S.: The virtual biology laboratories: a new approach of computational biology. In: Husbands, P., Harvey, I. (eds.) Proceedings of the Fourth European Conference on Artificial Life, pp. 274–283. MIT Press, Cambridge, MA (1997)

16. Kohl, N. and Stone, P.: Policy gradient reinforcement learning for fast quadrupedal locomotion. In: Proceedings of IEEE International Conference on Robotics and Automation (2004)
17. Nolfi, S., Floreano, D., Miglio, O., Mondada, F.: How to evolve autonomous robots: different approaches in evolutionary robotics. In: Brooks, R., Maes, P. (eds.) *Artificial Life IV*, pp. 190–197. MIT Press, Cambridge, MA (1994)
18. Nolfi, S., Floreano, D.: Evolutionary robotics – the biology, intelligence, and technology of self-organizing machines. In: *Intelligent Robotics and Automation Agents*. MIT Press, Cambridge, MA (2000)
19. Quilan, M.J., Chalup, S.K., Middleton, R.H.: Techniques for improving vision and locomotion on the Sony AIBO robot. In: *Australasian Conference on Robotics and Automation (ACRA 2003)* (2003)
20. Revonsuo, A.: The reinterpretation of dreams: an evolutionary hypothesis of the function of dreaming. *Behav. Brain Sci.* **23**(66), 877–903 (2000)
21. Röfer, T.: Evolutionary gait-optimization using a fitness function based on proprioception. In: *Proceedings of RoboCup 2004: Robot Soccer World Cup VIII. Lecture Notes of Artificial Intelligence*, pp. 310–322. Springer, Berlin Heidelberg New York (2004)
22. Ruiz-del-Solar, J., Zagal, J.C., Guerrero, P., Vallejos, P., Middleton, C., Olivares, X.: UChile1 team description. In: *Proceedings of RoboCup 2003: Robot Soccer World Cup VII. Lecture Notes of Artificial Intelligence*. Springer, Berlin Heidelberg New York (2003)
23. Stickgold, R., Hobson, J.A., Fosse, R., Fosse, M.: Sleep, learning and dreams: off-line memory reprocessing. *Science* **294** (5544), 1052–1057 (2001)
24. Varela, F.J.: *Principles of Biological Autonomy*, New York: Elsevier (1979)
25. Zagal J.C., Ruiz-del-Solar J.: UCHILSIM: A dynamically and visually realistic simulator for the RoboCup four legged league. In: *Proceedings of RoboCup 2004: Robot Soccer World Cup VIII. Lecture Notes of Artificial Intelligence*, pp. 34–45. Springer, Berlin Heidelberg New York (2004)
26. Zagal, J.C., Ruiz-del-Solar, J.: Learning to kick the ball using back to reality. In: *Proceedings of RoboCup 2004: Robot Soccer World Cup VIII. Lecture Notes of Artificial Intelligence*, pp. 335–346. Springer, Berlin Heidelberg New York (2004)
27. Zagal, J.C., Ruiz-del-Solar, J., Vallejos, P.: Back to reality: crossing the reality gap in evolutionary robotics. In: *Proceedings of the 5th IFAC Symposium on Intelligent Autonomous Vehicles, IAV 2004*. Elsevier, Amsterdam, The Netherlands (2005)
28. Zagal, J.C., Sarmiento, I., Ruiz-del-Solar, J.: An application interface for UCHILSIM and the arrival of new challenges. In: *Proceedings of RoboCup 2005: Robot Soccer World Cup IX. Lecture Notes of Artificial Intelligence*, pp. 464–471. Springer, Berlin Heidelberg New York (2005)