Robotics and Autonomous Systems 57 (2009) 819–827

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Self-modeling in humanoid soccer robots

Juan Cristóbal Zagal [a,b,*], José Delpiano [c], Javier Ruiz-del-Solar [c]

[a] Centro de Neurociencia de Valparaíso, Facultad de Ciencias, Universidad de Valparaíso, Gran Bretaña 1111, Playa Ancha, Valparaíso, Chile

[b] Instituto de Sistemas Complejos de Valparaíso, Artillería 470, Cerro Artillería, Valparaíso, Chile

[c] Departamento de Ingeniería Eléctrica, Facultad de Ciencias Físicas y Matemáticas, Universidad de Chile, Avenida Tupper 2007, Santiago, Chile

ARTICLE INFO

Article history:
Available online 8 April 2009

Keywords:
Self-modeling
Sensorimotor adaptation
Emergence of behavior
Humanoid soccer robots

ABSTRACT

In this paper we discuss the applicability, potential benefits, open problems and expected contributions that an emerging set of self-modeling techniques might bring on the development of humanoid soccer robots. The idea is that robots might continuously generate, validate and adjust physical models of their sensorimotor interaction with the world. These models are exploited for adapting behavior in simulation, enhancing the learning skills of a robot with the regular transference of controllers developed in simulation to reality. Moreover, these simulations can be used to aid the execution of complex sensorimotor tasks, speed up adaptation and enhance task planning. We present experiments on the generation of behaviors for humanoid soccer robots using the Back-to-Reality algorithm. General motivations are presented, alternative algorithms are discussed and, most importantly, directions of research are proposed.

1. Introduction

Humanoid soccer robots are challenged to overcome a series of complex sensorimotor tasks such as standing up, walking, kicking and trapping the ball or even diving for it. It appears that certain aspects of the fine dynamical interaction of a humanoid robot body with its environment cannot be properly modeled by an external designer. Most pre-engineered controllers fail to perform successfully if some of their parameters are not calibrated or adapted using real-world data.

The experience obtained on the generation of gaits for Sony AIBO four-legged robots shows that a large amount of sensorimotor interaction in combination with some learning method is required in order to achieve substantial improvements to gait speed (a comparison of different gait optimization methods is presented in [1]). The same situation has been observed in humanoid soccer robots, where gaits are optimized using stochastic methods [2–6].

Thus, solving complex sensorimotor tasks requires learning from the interactions of the robot body and the environment. However, there are some fundamental limitations in the implementation of this idea. First, the temporal limitation reduces the number of alternatives that a robot is able to explore during its ontogeny. While many researchers aim at discovering key algorithms to extract as much information as possible from the real-world interaction, it seems that the fundamental need to increase the number of samples is being forgotten. In contrast, the use of simulations [7,8] might considerably speed up the adaptation of humanoid robots, since the simulation of a robot acting under a particular environment can be easily accelerated by means of parallelization of candidate solution evaluations. The second limitation is related to the cost of robot hardware. Extensive periods of evolutionary search might bring about important damages to a humanoid robot due to their particular instability. In contrast to multi-legged stable robots or wheeled robots it appears rather difficult for a humanoid robot to develop a walking behavior from scratch. Improving a humanoid gait is particularly difficult since small parameter changes might cause instability and a robot to fall down. The third main limitation is the human time required for supervising the robot learning process. Humans must replace robot batteries, restart the setup when the robot falls down, and repair robot components when broken. Thus, the amount of interaction with the environment is limited due to the availability of human resources. In addition, for the case of soccer applications, adaptability speed is a key issue; it is necessary to be able to adapt the robot gait to different surface conditions in just a few hours after arriving at the competition site.

The use of simulations to implement parts of or the whole learning process and thus speed up the development of robot controllers, with the additional benefits of producing less damage of hardware components and less human intervention, dates back to the early 1990s. However, the fundamental problem of

* Corresponding author at: Centro de Neurociencia de Valparaíso, Facultad de Ciencias, Universidad de Valparaíso, Gran Bretaña 1111, Playa Ancha, Valparaíso, Chile.

E-mail addresses: jczagal@gmail.com (J.C. Zagal), jdelpian@ing.uchile.cl (J. Delpiano), jruizd@ing.uchile.cl (J. Ruiz-del-Solar).

0921-8890/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2009.03.010

the reality gap, which is the difference in behaviors obtained in simulation versus reality for a given robot controller, limits the applicability of these tools. The need for a continuous validation of simulation was noted by Brooks [9] and approaches that implement this idea have been already presented [10–12]. Although most current work that optimizes humanoid gaits make use of realistic simulators [2–4,6], their employment is rather restricted. The general idea is to implement the learning process in two consecutive stages: first, the parameter's optimization range is reduced using robot simulations, and then learning is carried out in the real robots. This process can be largely improved if experiments in simulation and in reality are fully integrated, and they run continuously, one after the other. In this way, robots, in addition to adjusting controller parameters, might generate self-models of their sensorimotor interaction, namely the body structure and physical laws. If this is the case, one could even think that the simulator is a part of the robot's cognitive structure, and that the robot can use the simulator while solving complex tasks, for example, playing soccer.

In this context the aim of this paper is to analyze the applicability, potential benefits, open problems and expected contributions that self-modeling techniques might bring on the development of humanoid soccer robots. We will explore how Nao humanoid soccer robots might generate self-models of their sensorimotor interaction using the *Back-to-Reality* (BTR) algorithm [11,1].

This paper is organized as follows. In Section 2 we discuss the concept of continuous self-modeling and how it might impact the field of robotics. In Section 3 we discuss methodologies for creating complex behaviors in animats and humanoid soccer robots. In Section 4 we explore how these methodologies affect the process of gait optimization with animats and Nao humanoid robots. Finally, in Sections 5 and 6 we discuss and make conclusions about the main results of this research work.

2. Self-modeling

As Metzinger proposes [13] in the context of phenomenological studies of consciousness, the study of self-models deals with questions such as: *How might a first-person perspective arise from an information-processing system? How do self-models appear as an interaction with the environment during evolution of nervous systems? How are the transitions from conscious to unconscious self-models related?* etc. These questions are starting to gather an increasing level of attention in the robotics community since (i) it appears that self-models might enhance the learning capabilities of robots, and (ii) in view of the complex dynamics of a mobile robot operation it appears attractive that models are self-generated rather than externally provided.

2.1. Self-modeling in robotics

2.1.1. Available techniques

In 2004, three self-modeling approaches intended for use in animats or robots were presented [12,10,11]. In [12], the question of whether there is an algorithm linked to an unknown body that can infer by itself information about the body and the world it is in was raised. According to experiments with a simulated mice head, they concluded that *sensorimotor laws possess intrinsic properties related to the structure of the physical world in which an organism's body is embedded*. In [10], the *Estimation Exploration* algorithm is proposed as a way to co-evolve a robot and its simulator. They later applied this algorithm to real robots [14]. In [11], the Back-to-Reality algorithm was presented in order to aid the autonomous generation of gaits with real Sony AIBO robots. The algorithm co-evolves robot simulators and robot controllers. Simulators are rewarded by their capability of providing a greater behavioral fitness consistency. A detailed explanation of the algorithm is presented in [1]. Besides gait generation the algorithm was applied to the autonomous generation of ball kick behaviors [15]. An analysis of properties of the fitness function used for simulation search is presented in [16]. We will explain the principles of operation of this algorithm in the following sections.

Converging approaches are presented in [17], where self-simulation is proposed for robot planning, and [18], where internal simulation of robot perception is explored.

In the case of humanoid robots, a method to interleave simulated and real data during robot adaptation is proposed in [5]. The strategy consists of estimating the discrepancies on fitness resulting from reality and simulators, and then using the estimated difference in order to correct the fitness values that result from a fixed simulation model during the robot adaptation process. This approach relies on the assumption that a linear estimate of the fitness discrepancies of simulation and reality can be established for a given population of robot controllers. Although the aim of this approach is not on robot self-modeling, the strategy is interesting for its potential to bridge fitness measurements related to the reality gap during robot ontogeny.

2.1.2. Embodied robot simulation

The specific properties of these techniques are still under investigation. With certain independence of the particular technique being used we subscribe in this paper to general principles that aim at generating what we call an embodied robot simulation [19]. The following are the principles of this view.

1. *Physical simulation of sensorimotor interaction:* A physics sensorimotor simulation is used to expand the domain of sensorimotor activity of a robot; thus internal interaction is combined with real interaction towards exploring new behaviors.
2. *Grounded representations:* The body components and environment objects interacting with the robot are represented by a set of physical primitives that are initially parametrized by a designer. Such parametrization defines a space of possible realizations of the simulation. A grounding mechanism allows searching for candidate simulations as abduction over data collected during robot functioning.
3. *Behavioral and sensorimotor consistency:* The grounding mechanism seeks to increase the historical consistency of behavioral and/or sensorimotor data acquired during operation under real versus simulated environments. Searching over the space of simulations and robot behaviors can be compared to a process of affordances or exploiting a mastery of sensorimotor contingencies. A self-organizing re-parametrization process should allow augmenting or modifying the search space in the case that the sensorimotor activity cannot be reproduced by the simulation.

3. Creating complex behaviors

In this section we will describe the principle of operation of the Back-to-Reality algorithm. The first description of the algorithm was presented in [11], and a more detailed description is given in [1]. Previous descriptions of the algorithm were specific to the use of a fitness based functional to be minimized in order to adapt simulation. In this section we present a more general description that considers sensor based as well as fitness based functionals. Some definitions are first presented, the principle of operation is discussed, and then the algorithm steps are presented.

3.1. Definitions

Simulation:

The implementation of a robot simulation is defined by a vector $s = \{s_1, \ldots, s_{N_\mathcal{S}}\}$ in the space $\mathcal{S}$ of possible simulations. The dimensions of this space might be defined by morphological aspects such as the length, width, shape or weight of robot components as well as their topological relation. It might also include aspects such as the friction among different elements, gravitational forces, motor parameters such as PID servo constants, etc. The experimenter defines the $\mathcal{S}$ boundaries of each parameter $s_i \in [min_i, max_i]$ with $i = \{1, \ldots, N_\mathcal{S}\}$.

Reality:

Reality is the target operational environment of the robot. We present experiments in which reality can be either a particular realization of the simulation $s_r \in \mathcal{S}$ or reality itself. As will be described, s_r is unknown for the robot and it should be determined by the algorithm by relying on behavioral comparisons.

Controller:

The implementation of a robot controller is defined by the vector $c = \{c_1, \ldots, c_{N_\mathcal{C}}\}$ in a space $\mathcal{C}$ of possible robot controllers. The space $\mathcal{C}$ might include morphological descriptors of the robot as well as controller-related parameters. However, we have not performed experiments for the evolution of robot morphologies. The experimenter defines the $\mathcal{C}$ boundaries of each parameter $c_i \in [min_i, max_i]$ with $i = \{1, \ldots, N_\mathcal{C}\}$.

Behavior:

The set of actions that a robot executes in response to the environment E. The characterization and qualification of a robot's behavior necessarily depends on the observer. From a single viewpoint we can model behavior in discrete time $t_j = j \cdot \Delta t$ as a time series $B = \{X_0, \ldots, X_{N-1}\}$ of N vector states $X_j = \{x_1, \ldots, x_{N_D}\}$, each describing N_D dynamical parameters, such as position, rotation and velocity, of bodies composing the robot or interacting with it. If we assume a set of fixed initial conditions, a fixed reference system and fixed evaluation period $T_e = N \cdot \Delta t$, we can establish that the robot behavior B is a function of the robot controller c and the evaluation environment E. Thus we have $B = B(E, c)$. In this context E might be either a simulation defined by a point $s \in \mathcal{S}$ or reality itself. Clearly, in the later case "$E = reality$" is just an abstraction for the sake of consistency.

Sensorimotor history:

Assuming a robot with N_O motors and N_I sensors, the motor history time series are defined as $O^l = \{o_{l^0}, \ldots, o_{l^{N-1}}\}$, for $l = \{1, \ldots, N_o\}$, and the sensor time series as $I^l = \{i_{l^0}, \ldots, i_{l^{N-1}}\}$, for $l = \{1, \ldots, N_I\}$.

Fitness:

The behavioral evaluation provided by the experimenter. We ascribe $f(B(E, c))$ to the fitness function that the experimenter assigns to the behavior $B(E, c)$. From a set of M robot controllers we denote the fitness of the robot controller c_k, with $k = \{1, \ldots, M\}$, that elicits behavior $B(E, c_k)$ as f_{Ek}, with $E = r$ for reality and $E = s$ for simulation. For example, at the end of T_e, the fitness might be the distance traveled by the robot, the distance traveled by a ball that the robot kicks, the amount of consumed energy, etc.

3.2. Principle of operation

If the behavior elicited by robot controller c in simulation s is similar to the behavior observed in reality we write

$$B(s, c) \approx B(r, c); \tag{1}$$

more precisely, this means that at any time step j we also have

$$X_{sj} \approx X_{rj} \quad \forall j = \{0, \ldots, N_B - 1\}, \tag{2}$$

where X_{sj} stands for the state vector of $B(s, c)$ at time step j, and X_{rj} is the state vector of $B(r, c)$ at time step j. In other words there should be a match along time of the robot state in both simulation and reality.

If we somehow measure the degree to which this match is obtained for each candidate simulation s, we can derive a useful distance function ϕ for adapting simulation to match reality. Unfortunately the state X_{rj} is generally unobservable.[1] The alternative is to estimate the quality of candidate simulations by assessing their ability to reproduce data collected during robot functioning.

The use of fitness data was explored for self-modeling in statically stable four-legged robots [11,20] and specific properties have been reported in [16]. The idea is to exploit the fact that behavioral fitness is always available in both simulation, f_{sk}, and reality, f_{rk}. Then a behavioral fitness discrepancy elicited by a robot c_k can be defined as

$$\delta_k = |f_{sk} - f_{rk}|. \tag{3}$$

If such behavioral discrepancies, as expressed in (3), are reduced for various behaviors, then it is natural to expect that simulation better approximates reality in those characteristics that are relevant to the execution of these behaviors. Thus, we define the first distance function ϕ_1 to be the average behavioral differences $\Delta_{fitness}$; this is

$$\phi_1 = \Delta_{fitness} = \frac{1}{m}\sum_{k=1}^{m} \delta_k = \frac{1}{m}\sum_{k=1}^{m} |f_{sk} - f_{rk}|. \tag{4}$$

Another alternative is to look at sensorimotor data that can be collected as the robot functions. Approaches in such a direction were first presented in [10,12]. A limitation of such ideas is given by the high decorrelation of sensorimotor data that results from candidate simulations that are dissimilar to reality, thus limiting comparisons to very narrow observation windows.

In this paper we will explore the alternative of exploiting sensorimotor data for self-modeling using some statistical measures. We will first use the information distance that was recently proposed in [21] in the context of defining a metric space of experience:

$$\mathrm{d}(X, Y) = H(X|Y) + H(Y|X), \tag{5}$$

where X and Y are random variables, and $H(X|Y)$ is the conditional entropy of X given Y. It has been shown that this information distance has the properties of a metric. Thus we define our second distance function ϕ_2 to be

$$\begin{aligned}\phi_2 &= D(O_r, O_s) + D(I_r, I_s) \\ &= \sum_{l=1}^{N_O} \mathrm{d}(O_r^l, O_s^l) + \sum_{l=1}^{N_I} \mathrm{d}(I_r^l, I_s^l),\end{aligned} \tag{6}$$

where O_r^l and O_s^l are the l-th real and simulated motor command time series, and I_r^l and I_s^l are the l-th real and simulated sensor readout time series.

3.3. Algorithm steps

The Back-to-Reality algorithm steps are shown in Fig. 1. A detailed description of each step is as follows.

[1] In control theory a system is observable if, for any possible sequence of state and control vectors, the current state can be determined in finite time using only the outputs.

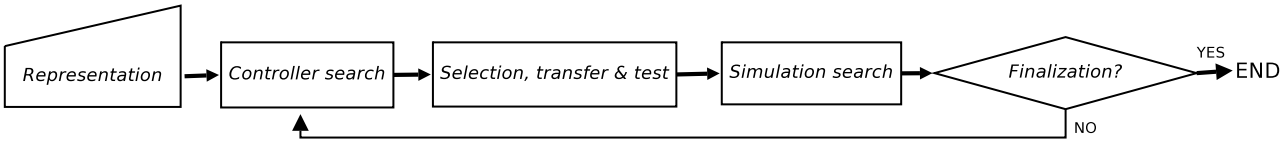

Fig. 1. Flow diagram of the Back-to-Reality algorithm.

Step 0 — *Representation:* The simulation $\mathcal{S}$ and controller $\mathcal{C}$ search spaces are established following the definitions presented in Section 3.1. Similarly, the fitness function, evaluation period T_e and distance function ϕ are defined.

Step 1 — *Controller search under simulation:* Using the best known simulation s_{i-1} as the environment, a genetic search is conducted over the controller solution space $\mathcal{C}$. A starting population of M controller individuals (usually $M \approx 30$) is obtained by performing random mutations with probability p_m over the solution c_{i-1} which is given as a seed. For the first iteration the population can be generated as random or biased by a known starting solution c_0. The number of generations during which the genetic search is conducted in this step should be small if the problem presents a high tendency for drift[2]. The output of this step is a new controller c_i that maximizes the behavioral fitness $f(B(s_{i-1}, c_i))$.

Step 2 — *Selection, transfer and test:* A set of controllers derived from the previous step is selected for being tested in the real robot. The idea is to collect the data that is required for assessing the quality of candidate simulators (required in Step 3) according to the distance function ϕ being used. In the case of using ϕ_1 a set of $(m < M)$ controllers (usually $m \approx 60\%M$) is selected in order of descending fitness and tested in reality. Corresponding fitness values f_{rk}, with $k = \{1, \ldots, m\}$, are stored. If it is not possible to find m transferable individuals from the last generation, they are taken, in descending order, from the previous generations obtained in Step 1. There is a genotypic similarity among the m controllers that triggers a phenotypic similarity among corresponding behaviors. A probability p'_m per bit controls the rate of mutation for a population constructed from a given controller c_i. An analysis on the right selection of p'_m is presented in [16]. If ϕ_2 is used, far fewer controllers need to be transferred to reality $(m \approx 1 - 2)$. In this case, however, time series of sensorimotor data are being collected (see 3.1).

Step 3 — *Simulation search:* The best existing simulator solution s_{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. A simulation s_i is obtained by steering the evolution towards minimizing the distance function ϕ. The algorithm continues by taking the simulation obtained in Step 3 as a new environment for Step 1 in the next iteration.

Step 4 — *Finalization:* The algorithm finishes if the robot behavioral requirements are met, if the number of iterations defined by the experimenter have been completed, or if there is disengagement on the co-evolving populations. The latter case might occur for example if the space of simulations does not account for phenomena that arise in reality. Explanations of how to deal with disengagement are presented in [1].

4. Experimental results

4.1. Analysis of the algorithm using an artificial ant morphology

First we explore the idea of self-modeling with the Back-to-Reality algorithm using an artificial morphology that models an ant (see the complete description in [16]). This morphology allows us to explore the generation of complex but generally stable walking behaviors while using an interesting six-legged animat. We center the analysis on exploring how the algorithm allows, at the same time, learning a walking gait from scratch and adjusting a simulation model to match reality. This adjustment allows identifying the ant morphology that best suits the correspondence of simulated and real behaviors. In this case we will also explore the behavior of the distance function ϕ_1. Fig. 2 shows the morphology of random ants that are obtained over a four-dimensional search space $\mathcal{S}$ defined by the parameters corresponding to the length of four of the twelve ant segments.

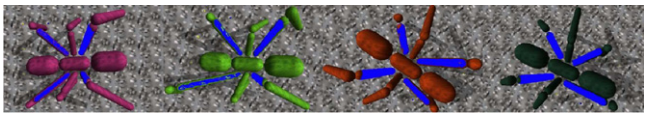

Fig. 2. Random realizations of the ant simulator. Each ant simulator is defined by varying the length of the four leg segments marked in blue.

4.1.1. Controller definition

Each one of the six ant legs is composed by a segment representing the femur and another representing tibia and tarsus. Joints connect each rigid body. Each independent axis of motion i (a total of 18) is motorized and torque is applied according to the output of PID dynamic compensators that follow a motion reference signal $r_i = \theta_i + a_i sin(wt + \psi_i)$, where θ_i is a pre-defined central angle of oscillation for each motor i.

The amplitude a_i and motion phase ψ_i are defined by the ant controller vector under search, $c = \{a_1, \psi_1, \ldots, a_{18}, \psi_{18}\}$ of 36 elements. These parameters are in the range $a_i \in \{a_{min}, a_{max}\}$ and $\psi_i \in \{\psi_{min}, \psi_{max}\}$ for each joint.

4.1.2. Analysis

Fig. 3 presents results corresponding to 13 iterations of the Back-to-Reality algorithm. On each iteration 11 generations of controller evolution, with population size of 20 individuals, and 11 generations of simulation search, with population size of 15 individuals, were performed. The figure shows the evolution of maximal and average controller fitness (gait speed) as well as the evolution of minimal and average $\Delta_{fitness}$. For each BTR stage the figure shows the corresponding value of the ant simulator, namely the estimated length of each segment resulting from the minimization of $\Delta_{fitness}$. We make the following observations from the results presented in this figure.

1. In (a) we can observe how the controller fitness monotonically increases along different iterations of BTR. This is interesting if we consider the important changes in the simulation of the ant that are introduced by varying the length of the four segments. In total, $11 \times 13 \times 15 = 2145$ controller evaluations were performed under simulation while just $12 \times 15 = 180$ evaluations were performed in the real environment. The methodology allows searching a solution space nearly 12 times larger than performing experiments in pure reality.

[2] A pathology of co-evolving systems in which the selection pressure of one population has no influence in the co-evolving population.

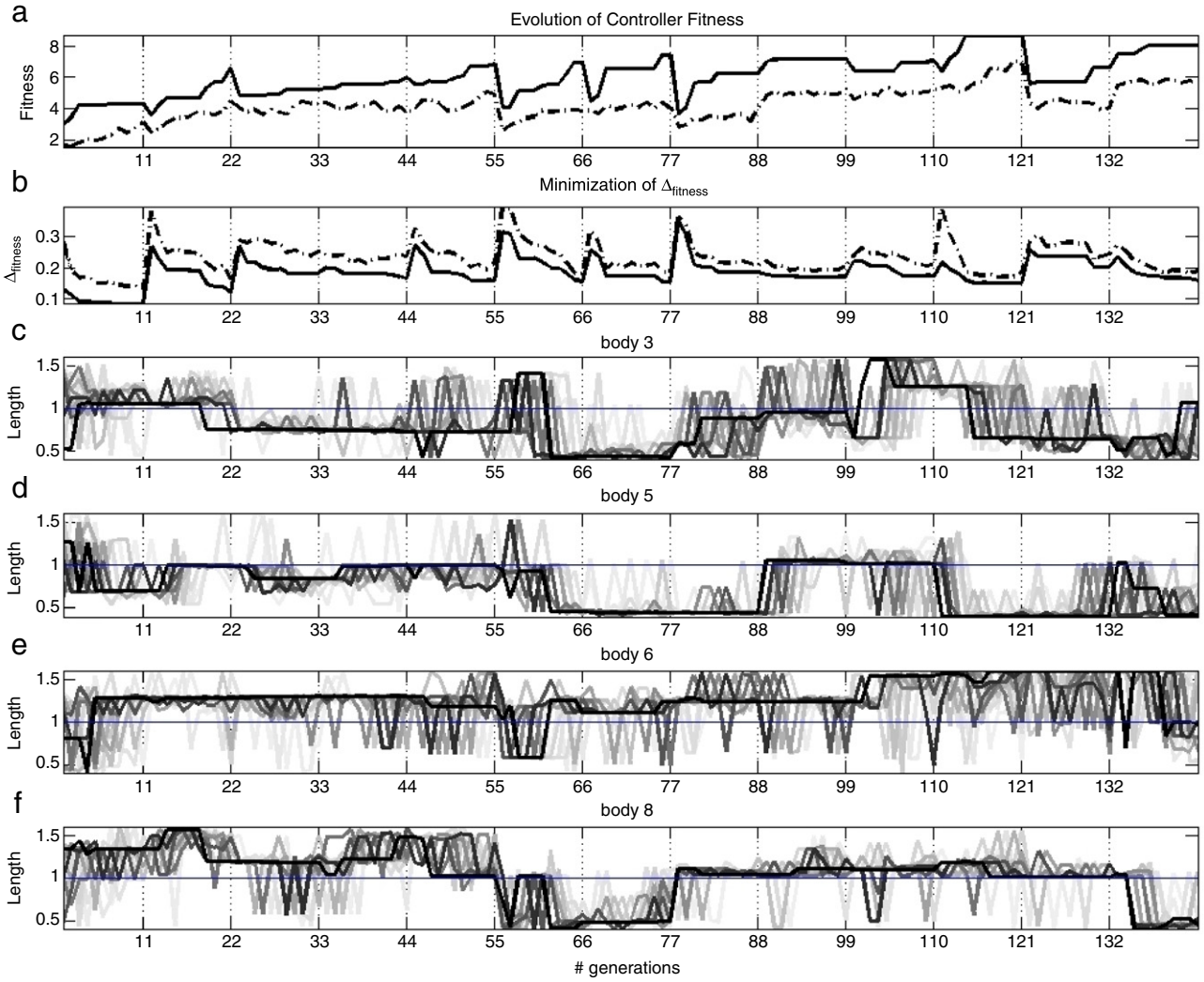

Fig. 3. Experimental results on the ontogeny of an artificial ant. The figure present 13 iterations of the Back-to-Reality algorithm. On each iteration 11 generations of controller evolution, with population size of 20 individuals, and 11 generations of simulation search, with population size of 15 individuals, were performed. In (a) the evolution of maximal and average controller fitness is shown with continuous and dashed lines. Similarly, in (b) the resulting evolution of minimal and average $\Delta_{fitness}$ is presented. In (c) to (f) the variation of parameters defining the simulation is shown. Black lines represent the estimation that was obtained by the individual with lower $\Delta_{fitness}$. The shade of gray decreases as the individual $\Delta_{fitness}$ increases. Reality is represented by the central line in which each parameter takes the value 1.

2. In (b) we can observe how the minimization of $\Delta_{fitness}$ takes place after each iteration of the algorithm. We observe a smooth minimization of $\Delta_{fitness}$ at each algorithm iteration stage. We observe, for the most of the iterations, that a low value of $\Delta_{fitness}$ is obtained, and therefore we conclude that a consistency of behavior on simulation and reality is obtained. Therefore, resulting behaviors of the overall adaptation process are transferable to reality. This is a very relevant result, considering our main hypothesis that a continuous self-modeling process *might allow for the regular transference of controllers developed in simulation to reality*.
3. In (c) to (f) the corresponding parameter's convergence is shown. We observe that although the parameters are not fully identified, the algorithm improves the estimation with time. However, some oscillations in the estimated values are observed. Probably the main reason for the observed oscillations is that in the experiments a convergence is not achieved, due to the low number of generations that are taken for minimizing $\Delta_{fitness}$.

4.2. Experiments with real Nao robots

In the previous section we have demonstrated the effectiveness of the Back-to-Reality algorithm as a whole co-evolutionary process. We have observed the continuous transference of behaviors obtained in simulation to reality and how the algorithm allows increasing the animat controller fitness while reducing the number of tests performed in reality. The algorithm's *simulation search* step is a key component that needs to be validated with a real humanoid robot. A natural question is whether a minimization of a distance function ϕ will lead to a better identification of aspects of a real robot. In this section we address this fundamental question, building a realistic Nao robot simulator and testing the identification power of the different metrics. We first describe the implementation of the new Nao simulator and then we describe the details of the experiments performed.

4.2.1. Building a Nao simulator

The first step for testing self-modeling with real Nao robots is to find the best available robot simulator that allows modifying the simulation parameters and executing the same robot controllers

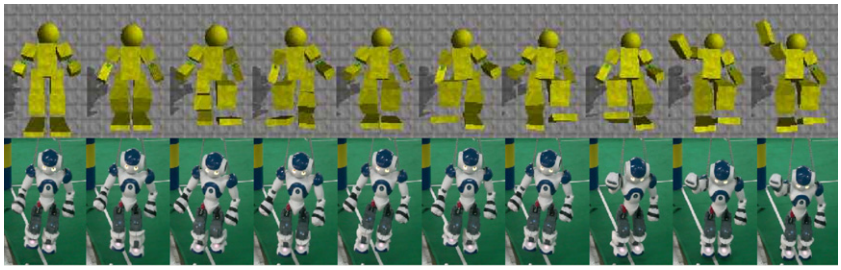

Fig. 4. Illustration of the complete walking behavior executed with the simulated Nao robot (top), and with the real Nao robot (bottom).

Table 1
Specification of Nao robot body components: mass, absolute positions, rotations, and joint orientations. Types of geometries are labeled as 2 for bounding box, 3 for cylinder and 4 for spheres. Simulation search space is defined by bodies that are labeled with a star, "*".

Body part	Mass	Translation			Rotation			Type	Object Shape		
Torso	1.22	0.00	0.07	−0.02	0	1	0	2	0.10	0.18	0.10
HeadYaw	0.05	0.00	0.16	−0.02	0	1	0	3	0.01	0.08	0.00
HeadPitch	0.35	0.00	0.22	0.00	1	0	0	4	0.07	0.00	0.00
RShoulderPitch	0.05	−0.09	0.14	−0.02	1	0	0	4	0.01	0.00	0.00
RShoulderRoll	0.11*	−0.10	0.14	0.00	0	1	0	2	0.07	0.06	0.08
RElbowYaw	0.03	−0.10	0.15	0.07	0	0	1	4	0.01	0.00	0.00
RElbowRoll	0.06	−0.10	0.15	0.12	0	1	0	2	0.05	0.05	0.11
LShoulderPitch	0.05	0.09	0.14	−0.02	1	0	0	4	0.01	0.00	0.00
LShoulderRoll	0.11*	0.10	0.14	0.00	0	1	0	2	0.07	0.06	0.08
LElbowYaw	0.03	0.10	0.15	0.07	0	0	1	4	0.01	0.00	0.00
LElbowRoll	0.06	0.10	0.15	0.12	0	1	0	2	0.05	0.05	0.11
RHipYawPitch	0.10	−0.06	−0.04	−0.03	0.707	0.707	0	4	0.01	0.00	0.00
RHipRoll	0.14*	−0.06	−0.04	−0.03	0	0	1	4	0.01	0.00	0.00
RHipPitch	0.29*	−0.06	−0.09	−0.02	1	0	0	2	0.07	0.14	0.07
RKneePitch	0.42*	−0.06	−0.13	−0.01	1	0	0	2	0.08	0.11	0.07
RAnklePitch	0.06	−0.06	−0.23	−0.01	1	0	0	4	0.01	0.00	0.00
RAnkleRoll	0.10	−0.06	−0.27	0.02	0	0	1	2	0.08	0.02	0.16
LHipYawPitch	0.10	0.06	−0.04	−0.03	0.707	−0.707	0	4	0.01	0.00	0.00
LHipRoll	0.14*	0.06	−0.04	−0.03	0	0	1	4	0.01	0.00	0.00
LHipPitch	0.29*	0.06	−0.09	−0.02	1	0	0	2	0.07	0.14	0.07
LKneePitch	0.42*	0.06	−0.13	−0.01	1	0	0	2	0.08	0.11	0.07
LAnklePitch	0.06	0.06	−0.23	−0.01	1	0	0	4	0.01	0.00	0.00
LAnkleRoll	0.10	0.06	−0.27	0.02	0	1	0	2	0.08	0.02	0.16

that are used in the real robot. After testing current available simulators we found that there is no simulator that provides all the above-mentioned requirements with Nao robots. Thus we decided to implement our own Nao simulator using the Open Dynamics Engine together with a library of functionalities provided with the simulator reported in [20].

The simulator was constructed using public data provided by the manufacturers [22]. Table 1 shows the specification of each one of the 23 robot body components, including the mass, absolute position, translations, rotations and the corresponding geometric object that are used in the simulation as best approximation. The specification also contains the orientation of each robot joint. The same data is used by the commercially available existing simulators.

We have identified two critical factors for achieving a good simulation of Nao robots. The first is finding the right tuning of PID joint servos. Fig. 5 (left) shows examples of the achieved reference following performance of joints before adapting simulation with BTR. The second critical factor is selecting the right number of physical integration steps to be taken between each pair of controller reference signals. The real robot receives controller references every 55 ms. We found that in order to achieve a good simulation performance, a total of 7 physical integration steps must be performed between each controller reference. This means that our physical simulation uses a temporal discretization of 7 ms.

4.2.2. Test description

Following the steps of the Back-to-Reality algorithm we have defined the robot simulation search space $\mathcal{S}$ (Step 0) by the set of 8 body weights that are indicated in Table 1. We have considered a range of variation of $+/-50\%$ of their original values. We have observed that the behavior of the real robot is quite sensitive to variations of body weight. In this case we will test the distance function ϕ_2 that was described in Section 3.1. As robot controller we have used the walking behavior implemented by Aldebaran-Robotics (Step 1). The main idea of this controller is to transform ZMP[3] targets into COM[4] targets. NaoQi, the SDK[5] developed by

[3] Zero Moment Point.
[4] Center of mass.
[5] Software development kit.

Fig. 5. Controller reference following capabilities of the simulated robot before (left) and after (right) executing Step 3 of the algorithm. Examples of three different Nao joints are shown.

Aldebaran, allows the acquisition of motor commands and sensor readouts at each motorized robot joint. We have defined a set of $N = 4024$ time steps for collecting joint sensor data at each one of the 22 real Nao motors. Then we have accumulated the set of readouts from the real robot by testing the above-mentioned controller (Step 2), and we have performed a genetic search over