Learning Humanoid Soccer Actions Interleaving Simulated and Real Data

Luca Iocchi and Fabio Dalla Libera and Emanuele Menegatti

Abstract— **This paper presents an approach for learning complex tasks on real robots, like walking or kicking in a humanoid soccer robot, profiting at most from the possibility to run simulations of a virtual model of the robot. This approach avoids to damage the real robot in the time consuming trials needed to learn a correct behavior and avoids to overfit the virtual robot model. The basic idea is to run most of the learning steps in simulation and to use a few learning steps on the real robot to assess discrepancies between the simulation and the reality. The calculated discrepancies are used to correct the fitness function used in simulation. Experiments on interleaving the learning between a real robot (Robovie-M by VStone) and its virtual model in USARSim are presented. They show that the proposed method is effective and significantly reduces learning time.**

I. INTRODUCTION

One of the first problems investigated by the humanoid robot community has been the creation of stable and human-like walking for humanoid robots. Several methods have been presented to address the gait stability problem. Some of them stem from control theory approaches and are based on the inverted pendulum model or on the calculation of the zero moment point (ZMP). All of these aim at planning the best trajectories for the different joints of the robot. There are several possibilities to calculate these trajectories like: offline pattern generation, offline pattern generation with online feedback compensation, and online pattern generation with online feedback control [10].

Most of these methods cannot be applied on small humanoid robot platforms due to their lack of high precision sensors and to the small computational resources they typically have on board. However, Manni and Indiveri proposed a simple control architecture for small humanoid robots based on the use of the FRI (Foot Rotation Indicator) point and the support polygon that allows to decouple the gait generation issue and the overall dynamic stability of the system [1].

The generation of stable walking for small humanoid robots is simplified by their relative intrinsic stability (i.e., relatively small size, low center of gravity, large feet). For most of the robots the gait is designed by hand with the tools (mostly graphical user interfaces) provided by their manufacturers (among the others: VStone, Kondo, ZMP, Robotis). However, hand-made motion generation is time consuming, can wear the robot due to the several trials and errors, and require deep understanding of the robot dynamics from the motion designer. For these reasons software for automatic generation of walking gaits (and in general of complex motions) is needed.

A possibility is to use learning or optimization algorithms to find the optimal parameters to control the gait of the robot. These methods can be divided into three different groups of approaches: (i) stochastic optimization methods (e.g., Genetic Algorithms, Evolutionary Algorithms, Simulated Annealing), (ii) sampling methods (e.g., Nelder-Mead simplex approach, Pattern Search), and (iii) surrogate optimization methods (e.g., [8]).

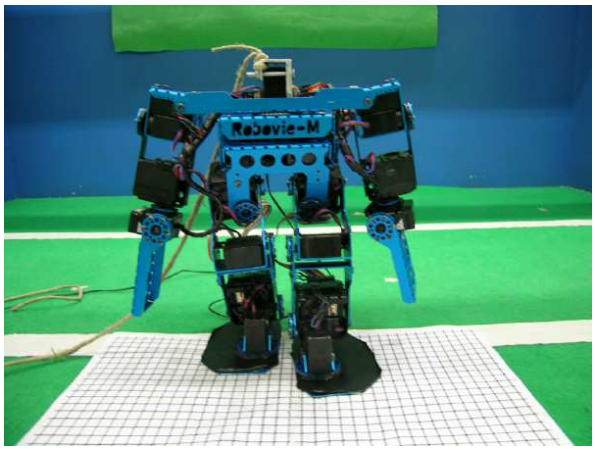
However, performing the several experiments needed by the learning algorithms on a real robot requires an extensive use and wearing of the robot, manpower for observing the results, and expensive lab time. For these reasons several researchers developed models of their robot in order to be able to perform the learning/optimization stage in simulation. However, the results of these simulations are strictly coupled to quality and accuracy both of the robot model and of the physical simulator. Most of the time the derivation of accurate enough robot models requires too much effort and in case of the small-humanoid robots might be not possible to obtain due to the low reliability/repeatability of the actuators used. Moreover, a learning process performed only in simulation might overfit the virtual model of the robot and its loose correspondence to the real robot, ending in a perfect gait for the virtual robot, but in a poor result for the real robot.

For these reasons, we propose to interleave learning steps in simulation with learning steps on the real robot. We will show that most of the learning can be done in simulation and the steps on the real robot are used to adjust the learnt fitness function. As the basic learning technique we use Genetic Algorithms (GAs) to explore the parameter space of a fuzzy controller that implements the behavior of the robot. The learning approach proposed in this paper has been experimented on the humanoid robot Robovie-M and its virtual model in USARSim (see Figure 1).

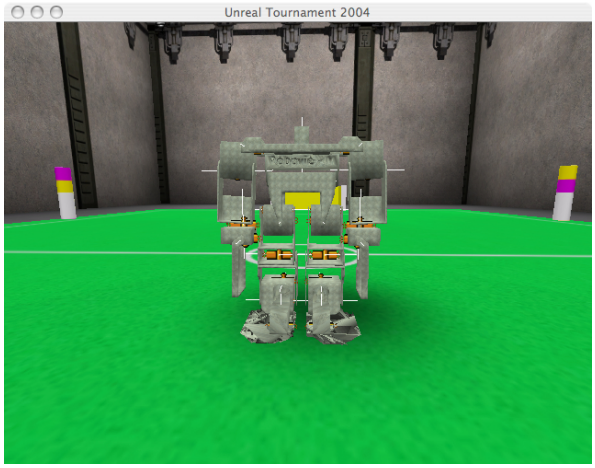
The remainder of the paper is organized as follows: Section II reports some of the previous works using learning algorithms on small humanoid robots. Section III describes the proposed algorithm for interleaved learning in simulation and in reality. Section IV introduces the learning process used to test the effectiveness of the proposed approach. In Section V we describe the experimental setting and the performed experiments that validate the proposed approach. Finally, Section VI summarizes the work done and hints the future directions of this research.

L. Iocchi is with DIS, Sapienza University, Via Ariosto 25, 00185 Rome, Italy ; e-mail: iocchi@dis.uniroma1.it

F. Dalla Libera and E. Menegatti are with the Department of Information Engineering The University of Padua via G. Gradenigo 6/A I-35131 Padova - ITALY emg@dei.unipd.it



(a) The real robot (RR)



(b) The virtual robot (VR)

Fig. 1. The real robot Robovie-M by VStone and the virtual robot rendered by the USARSim simulator

II. Related Work

Several authors applied learning techniques to biped walking: either to learn from scratch to walk or to optimize the gait to the particular speed of the robot or to the terrain on which the robot is moving. Tedrake et al. proposed a reinforcement learning approach which optimizes an online control policy, so their robot, due to its mechanical design, is able to acquire a robust policy for dynamic bipedal walking from scratch [14]. A similar approach for a passive dynamic walker was proposed in [13]. Pastrana used genetic algorithms to optimize the speed coordination and control of a humanoid robot's gait in order to make it walk in a stable manner at different walking speeds [11]. Other approaches to biped walking have used evolution strategies for optimizing a parametric walking model [6] or model-free methods based on neural oscillators [7]. Most of these approaches make use of a simulator for speeding up the learning process. The way in which the simulator is used is by performing preliminary experiments on it to find a suitable solution and then use this solution on the robot, and possibly performing additional learning processes on the real robot. The problem with this kind of approaches is that they strongly rely on the fidelity of the simulated model.

To improve the effectiveness of using a simulator for robot learning, some authors propose a different use of a simulator. Co-evolution of models and tests is presented in [3], [9], where evolution on the simulator is used to improve the performance in some behavior, while evolution on the real robot is used to improve the simulator model. Also Abbeel et al. [2] present a Reinforcement Learning technique in which real experiments are used to evaluate a policy, while simulated experiments are used to estimate the derivative of the evaluation with respect to the current policy. The two above approaches thus interleave simulated and real experiments to refine some model of the system: the simulator model in the first case, the dynamics of a Markov Decision Process in the second case. However, the problem of refining the model on the basis of the difference in performance between real and simulated behaviors is increasingly difficult as the complexity of the problem increases.

The approach presented in this paper also uses interleaved simulated and real experiments, but with the objective of refining the function used to evaluate different solutions instead of the model of the system. When complex tasks are considered (as the biped locomotion) this is an obvious advantage since approximating a function is much easier than refining a model.

III. Learning by interleaving simulated and real experiments

In this section we describe an algorithm for learning using interleaved simulated and real experiments. The method applies to real systems for which making experiments is expensive and time consuming, and for which a simulator is available. Obviously, the simulator cannot exactly reproduce the real system; however, our approach has no requirements for such a simulator, but it only assumes that the behavior in the simulator is similar to the real system. As an example, we apply the proposed technique to the problem of learning biped locomotion for a humanoid robot. .

The approach we will follow is to perform a set of experiments on the simulator, then some of them are repeated on the real robot in order to determine discrepancies in the results. Such difference is then used in the next steps of the learning process in order to make results obtained with simulated experiments closer to the real ones. This process is then repeated several times until some convergence criterion is met.

In this iterative process we do not aim at refining the robot model or the simulator. The main motivation is that refining of simulated models to make it more realistic is anyway a difficult and very time consuming process. Our solution, instead of modifying the simulated model of the system, it just dynamically computes a *correction function* that measures differences between the real and the simulated system outputs. This *correction function* is applied when scoring the simulated examples in order to make the results

Definitions
$P = \{X_1, ..., X_n\}$: genetic population
F_R: real evaluation function
F_S: simulated evaluation function
ΔF^i: correction evaluation function at iteration i
${F_C}^i \equiv F_S + \Delta F^i$: corrected simulated evaluation function at iteration i
$\delta_F^i(X) \equiv F_R(X) - F_C^i(X)$
σ : standard deviation of the correction term
h : number of simulation steps before a real evaluation

Algorithm
INPUT: F_R, F_S
OUTPUT: X^* : local maxima of F_R
CONSTANTS: σ, h

$P^0 \leftarrow$ random individuals
$i \leftarrow 0$
$\Delta F^i \leftarrow 0$
while (*not* termination condition) **do**
  for h steps **do**
    evaluate P^i with F_C^i
    $P^{i+1} \leftarrow$ genetic evolution from P^i
    $i \leftarrow i + 1$
    $\Delta F^{i+1} \leftarrow \Delta F^i$
  end for
  evaluate P^i with F_R
  $\Delta F^{i+1}(X) \leftarrow \Delta F^i(X) + \sum_{k=1}^{n} \delta_F^i(X_k^i)\, e^{-\frac{||X - X_k^i||}{\sigma^2}}$
  $i \leftarrow i + 1$
end while
return $X^* = \mathtt{a}rgmax_{X \in P^i} F_C^i(X)$

TABLE I
THE INTERLEAVED SIMULATED AND REAL DATA LEARNING ALGORITHM.

similar to the real system even when the simulator behaves differently.

In this work, we consider a genetic algorithm as the basic learning technique and the evaluation function is given by a fitness function that measures the quality of the solution. By interleaving experiments with the simulator and experiments with the real system, the proposed algorithm computes a *correction function* that represents the difference between the fitness values obtained with the real system and the fitness values obtained with the simulator. Since the scores of the simulated experiments are summed to the *correction function*, they tend to approximate scores that would have been obtained from real experiments.

The learning algorithm for interleaved use of simulated and real data is presented in Table I. We consider two evaluation functions: F_R, which is the evaluation function (or fitness function) when performing experiments on the real robot, and F_S, which is the evaluation function when performing experiments in a simulator.

The algorithm provides a way to find a (local) optimal solution to F_R performing many evaluation steps on F_S and only a few evaluation steps on F_R. This is achieved by modifying at each iteration the function that evaluates experiments with the simulator, taking into account discrepancies from the evaluation on the real system. More specifically, at each iteration individuals in the genetic population are evaluated with the fitness function ${F_C}^i \equiv F_S + \Delta F^i$, where F_S is the value obtained by the simulated experiments and ΔF^i represents the difference obtained with respect to the real experiments. As we will show in the next paragraphs, after a sufficient number of iterations, F_C^i approximates F_R around some local maximum of F_R, therefore an optimal solution of F_C^i is also a local sub-optimal solution of F_R.

ΔF^i is defined as the sum of a set of Gaussian-like functions $\delta_F^i(X_k^i)\, e^{-\frac{||X - X_k^i||}{\sigma^2}}$, with $\delta_F^i(X_k^i) \equiv F_R(X) - F_C^i(X)$ representing the difference between real and simulated evaluation. These functions are determined and added to ΔF^i periodically (every h iterations) during the algorithm. Each of these functions represents a local correction of F_C^i towards F_R. More formally, in correspondence with points X_k for which a local correction exists, the value of $\Delta F^i(X_k)$ is a good approximation of $F_R - F_S$, and thus $F_C^i(X_k) \simeq F_R(X_k)$.

Therefore, the presented algorithm allows for finding a local optimal solution of F_R by performing evaluations on the real and on the simulated system with a ratio $1 : h$ (i.e., 1 experiment on the real system every h experiments in the simulator). The result is obvious in the ideal case $F_R = F_S$ and also when the derivatives of the two functions F_R and F_S have the same sign for each point, but it works well even if this condition is not met.

To better describe the algorithm and its properties let us present an example in one dimension. We consider two one dimensional functions $F_R(x)$ and $F_S(x)$ and apply the above algorithm. $F_R(x)$ is the fitness function of the real system, i.e., the one we want to use only rarely, and $F_S(x)$ is the fitness function of the simulated system, i.e., the one we want to use most of the times during the learning process. We have experimentally determined that a good value for σ depends on the random step used to produce mutations in the genetic evolution step. In these tests we set the random step of the genetic algorithm to 0.2 and $\sigma = 0.6$. However, we have similar behaviors, but with different convergence rates, using σ varying from 0.1 to 3.0. We use $n = 10$ individuals that are uniformly selected in the interval $[-20, 20]$ and $h = 10$.

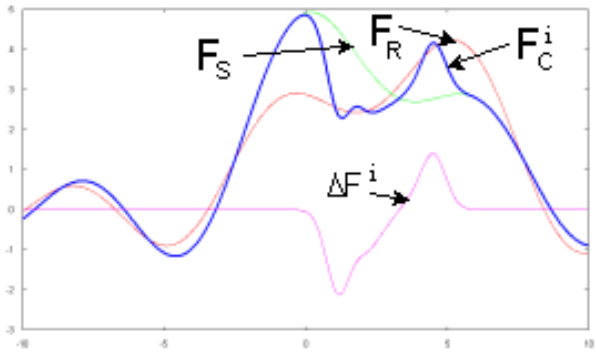


Fig. 2. Algorithm Execution Example in 1-D

Figure 2 shows an example of execution of the algorithm on 1-D functions. The red and green plots represent F_R and

F_S functions respectively. The bold blue plot shows F_C^i after 250 iterations (but only 25 iterations with the real function F_R), while the lower function shows ΔF^i. As shown in the figure, the algorithm approximates F_R with F_C^i near one of its local maximum, but also near the local maximum of F_S. In fact, in the early iterations values that maximizes F_S, but not F_R, are retrieved, and thus correction is needed to enable the algorithm to explore other areas of the search space.

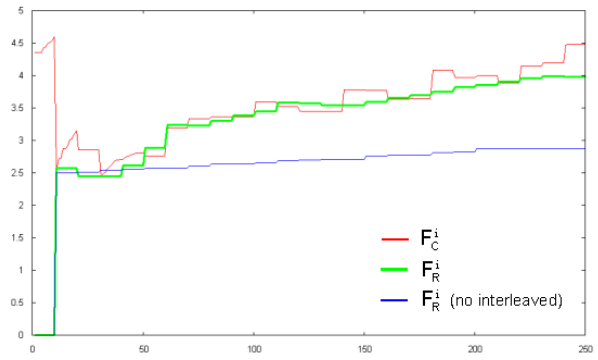


Fig. 3. Learning process for the example in 1-D

In Figure 3 the evolution of the learning system is shown. The red plot shows the value of F_C^i for the best individual at each iteration, while the green bold plot shows the value of F_R^i that is computed every 10 iterations. The figure shows how the function F_C^i approximates F_R^i, reaching its local maximum. The lower blue function instead shows the learning evolution obtained with the same number of experiments on real data, but without the simulation experiments interleaved. The figure illustrates the advantage of using simulated experiments and the effectiveness of the correction function ΔF^i computed in the algorithm. It is also evident from this example that the best solution obtained by considering only F_S gives a poor result on F_R, due to an overfitting to the simulated function. Finally, notice also that good solutions are obtained since the early stages of the learning process. For example, in the figure we can see that a good solution is already obtained after 60 iterations (6 iterations on real data).

IV. Implementation of the Learning Process

The learning process we have chosen for the experiments described in this paper is based on the development of a genetic algorithm for optimizing a fuzzy control system. This approach has been largely used in the literature (see for example [12], [15]) and has produced good results in the task of biped locomotion. However, it is interesting to point out that the approach described in the previous section could be also applied with other base learning algorithms.

Two behaviors have been implemented and tested: walking and kicking. They have been extensively tested in the simulator described in the next section, which does not require human assistance and thus allows for performing extensive learning processing.

The fuzzy controllers designed for such behaviors have been optimized by using a genetic algorithm. In the experiments reported below we used a population of 20 individuals. For each generation, the best individual is replicated (*elitism*), 70% of the individuals are obtained by *crossover*, the remaining by *mutation*. Individuals for crossover and mutation are probabilistically selected considering the fitness functions. In order to speed up the process, we found useful to select only those individuals that are above 50% of the fitness value of the best individual. Crossover is implemented with a random selection of the parameters from each of the two individuals, while mutation is implemented by adding a small random value to the parameters. The initial generation is computed by randomizing an initial set of parameters that allows for a stable, but not very effective behavior.

A. Implementation of walking

The fuzzy control system that we have used to implement a walking gait is based on an oscillation function

$$J_b(t) = A \sin(2\pi\omega t)$$

where J_b is the joint of the torso that makes the robot swing left-to-right, A and ω are parameters of the learning process. The value of J_b determines three different fuzzy variables: *body_left*, *body_center*, *body_right* (see Figure 4).

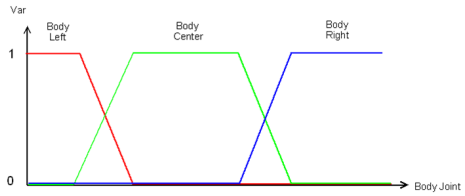
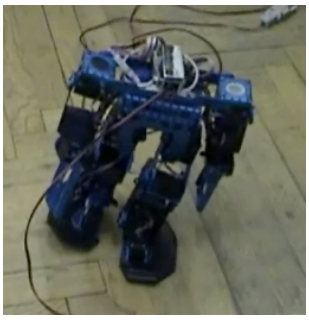


Fig. 4. *body_left*, *body_center*, *body_right* fuzzy variables.

In addition, 6 control variables are used for controlling 6 joints: 2 for the leg, 2 for the foot, 1 for the torso (bending ahead), and 1 for the arm. The fuzzy rules are designed to behave as follows: when the body is on the center, then the joints move to a stable standing position; when the body is on the left, then the right leg and the left arm are moved forward (right step forward); when the body is on the right, then the left leg and the right arm are moved forward (left step forward).

More specifically, in our implementation, each rule has a fuzzy variable (either *body_center*, or *body_left*, or *body_right*) as the antecedent and a pair $<$ control variable, target value $>$ (related to one of the above joints) as the consequent. The evaluation of the antecedent of the fuzzy rule determines the gain that is applied to the control variable to reach the target value specified in the rule.

A genetic algorithm has been used to optimize the parameters X of such a fuzzy controller. As fitness function, we used the distance walked by the robot during an experiment with a constant time slot (in our case, 20 seconds) with a penalization for walking gaits that are not straight:



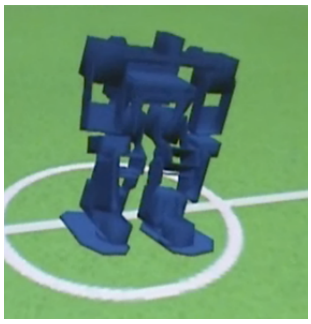


Fig. 5. Examples of walking.

Fig. 6. Examples of kicking.

$$F(X) = x_{\mathrm{e}} - |y_{\mathrm{e}}|$$

where $(x_{\mathrm{e}}, y_{\mathrm{e}})$ is the position reached after the experiment, x is the axis in the direction of motion, y is orthogonal to it, and $(0, 0)$ is the initial position of the robot. If the robot falls down during an experiment the fitness function is decreased to zero and this individual will not be part of selection in the next generation.

Examples of execution of the walking motion on both the robot and the simulator are shown in Figure 5.

B. Implementation of kicking

The implementation of the kicking behavior is based on the oscillation function

$$J_b(t) = A \sin(2\pi\omega\, t)$$

that controls the swing left-to-right of the body.

The fuzzy variables for this behavior include those representing the position of the body (*body_center*, *body_left*, *body_right*) and those representing the position of the kicking leg (*leg_backward*, *leg_straight*, *leg_forward*). The control variables are again a subset of the joints that will be operated by the behavior.

The fuzzy rules are designed in such a way to swing the kicking leg when the body is in the appropriate position; depending on the position of the leg, other rules determine the position of the knee, the ankle and the arm joints as well.

As in the previous case all the parameters of the fuzzy rules are optimized using a genetic algorithm. The experiments have been performed by putting the robot close in front of a soccer goal and with a ball just ahead the kicking foot. The fitness function used in this case measures the velocity of execution of the behavior, penalizing those executions that make the robot fall down or that do not score the goal.

Examples of execution of the kicking action on both the robot and the simulator are shown in Figure 6.

V. Experiments

A. The Robovie-M humanoid platform

Robovie-M is a small humanoid robot platform produced by the Japanese company VStone[1]. In this work, the Robovie-M version 2 was used. The size of the robot are 290x240x65mm, and its weight is 1.9 Kg. Robovie-M version 2 has 22 degrees of freedom (DOF) activated by 22 Sanwa servomotors (six in each leg, four in each arm, and two in the trunk) . The motors are driven by three Microchip PIC16F877 controlled by a Renesas CPU H8/3684 working at 16MHz. The board mount a two axis accelerometer and a RS232 serial port. The firmware controlling the H8/3684 is proprietary.

The control board's power supply is given by five AA batteries of 1.2 V and 2300 mA that gives a power supply at 6 V with peak-current of 6 A.

B. The USARSim Simulator

USARSim[2] is a realistic simulator developed by the Urban Search and Rescue (USAR) RoboCup community [4], that has been recently extended with the support for legged robots [16]. It is intended as a research tool for the study of human-robot interaction (HRI) and multi-robot coordination. This robot simulator is based on the industrial Unreal Engine game engine released by Epic Games with Unreal Tournament 2004. The simulation engine and its development tools can be inexpensively obtained by buying the game. Unreal Editor enables the user to rapidly develop objects and environment. Unreal Script, an ad-hoc script language, enables the user to control the behavior of the simulated objects.

Using USARSim problems like modeling, animation, and virtual environment's rendering are automatically solved. High quality 3D rendering at low cost is made possible by building the simulation on top of a game engine and this solution outperform open-source simulation projects. USARSim has the advantage that a full effort can be devoted to the robotics-specific tasks of modeling platforms, control systems, sensors, interface tools and environments. These tasks are in turn, accelerated by the advanced editing and development tools integrated with the game engine leading to a virtuous spiral in which a widening range of platforms can be modeled with greater fidelity in less time. The current release of the simulation consists of: various environmental models (levels), models of commercial and experimental robots, and sensor models. The high quality of the 3D

[1] www.vstone.co.jp

[2] usarsim.sourceforge.net

rendering of USARSim enables it to be used also to test image processing algorithms to be run by the robot in simulation. In fact, USARSim can generate views of the environment as exocentric view (i.e. an external camera observing the robot, like in Fig.1(b)) or in egocentric view (i.e. the camera mounted on the robot).

In a previous work, we showed that USARSim is a good simulator for small humanoid robot platforms [5] and that the simulated robot approximates quite well the behavior of the real robot, as can be seen in the video available at the link `http://www.dei.unipd.it/~emg/downloads/penaltyComparison.wmv`.

C. Experimental Results

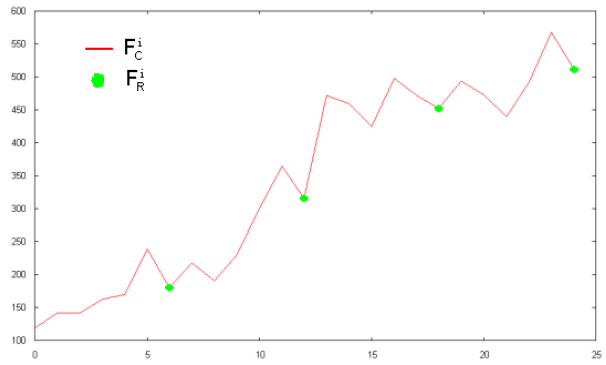


Fig. 7. Learning process with interleaved simulator and real experiments.

In this section we report a learning session for the walking behavior of the above presented robot. Genetic Algorithms have been used as the base learning technique and the algorithm proposed in this paper for interleaving simulated and real experiments has been used. Genetic Algorithms are configured as described in Section IV, and the ratio between real and simulated experiments has been set to 1:5.

Figure 7 shows the value of the fitness function (distance in [mm] moved in 20 seconds) over the number of iterations of the learning process. We have performed 24 iterations (20 on the simulator and 4 on the real robot); for the simulated experiments we evaluated 20 individuals for each generation, while for the real robot only the best 10 individuals have been considered. Therefore, we performed a total of 480 runs on the simulator and only 40 runs on the real robot. The red plot in the Figure shows the corrected fitness function, while the green dots the values obtained with the experiments on the real robot. The best solution after these iterations has a fitness value of 512 mm.

To demonstrate the effectiveness of our approach, we have determined two other measures: 1) the speed-up in the learning process with respect to using only real data (with the same number of runs); 2) the overfitting if using only simulated data.

For the first point, we have run genetic algorithm from the same initial parameters setting used before, running 4 generations of 10 individuals (i.e., 40 runs in total). The fitness value of the best solution has been 130 mm, so the speed-up for this experiment has been 3.94.

For the second point we run the same number of iterations (20) with simulated data only. We obtained in this way a value for the best fitness of 668. However, when evaluating this solution on the real robot we had a slightly worse result than the one given with the interleaving method: 488 mm. This can be explained by a specialization (overfitting) of the solution to the simulated model of the robot and the environment, that instead is not present when the corrected fitness function is used.

VI. CONCLUSIONS AND FUTURE WORKS

In this paper we have presented an algorithm and experimental results for learning behavior of a complex system using both the real device and a simulated environment. In particular, we focus our work on biped locomotion for a humanoid robot and we have used a Robovie-M humanoid robot and its simulated model in USARSim to perform the experiments. The proposed approach makes use of interleaved experiments from a simulator and from the real robot aiming at obtaining good results minimizing the number of experiments on the real robot. As shown by reported experiments, both on a synthetic simple problem and with a real robot, the proposed method allows for effective speed-up of the learning process using a simulator, but at the same time it avoids overfitting to the simulated model by periodically computing a correction evaluation function that takes into account differences between the simulated environment and the real robot.

As future work, we plan to perform a more systematic evaluation of the proposed approach also using other robotic platforms, e.g. AIBO robots, for which we have both the real robot and the simulated model. Furthermore, it is interesting to study in more detail the properties of the developed algorithm, and in particular to provide sufficient conditions on the functions F_R and F_S to find with the proposed algorithm a local maximum of F_R.

REFERENCES

[1] A. di Noi A. Manni and G. Indiveri. A control architecture for dynamically stable gaits of small size humanoid robots. *Proc. Of the 8th International IFAC Symposium on Robot Control SYROCO 06, Bologna, Italy*, (1), September 6-8 2006.

[2] P. Abbeel, M. Quigley, and A. Y. Ng. Using inaccurate models in reinforcement learning. In *Proc. of International Conference on Machine Learning (ICML)*, 2006.

[3] J. Bongard and H. Lipson. Nonlinear system identification using coevolution of models and tests. *IEEE Transactions on Evolutionary Computation*, 9(4):361–384, 2005.

[4] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSim: A robot simulator for research and education. In *Proc. of Int. Conf. on Robotics and Automation (ICRA)*, 2007.

[5] N. Greggio, G. Silvestri, S. Antonello, E. Menegatti, and E. Pagello. A 3d model of a humanoid robot for usarsim simulator. In *Proceedings of the First Workshop on Humanoid Soccer Robots, Genoa, ITALY*, pages 17–24, December 2006.

[6] M. Hebbel, R. Kosse, and W. Nistico. Modeling and learning walking gaits of biped robots. In *Workshop on Humanoid Soccer Robots*, 2006.

[7] Daniel Hein, Manfred Hild, and Ralf Berger. Evolution of biped walking using neural oscillators and physical simulation. In *RoboCup 2007: Proceedings of the International Symposium*, LNAI. Springer, 2007.

[8] Th. Hemker, H. Sakamoto, M. Stelzer, and O. von Stryk. Hardware-in-the-loop optimization of the walking speed of a humanoid robot. In *CLAWAR 2006: 9th International Conference on Climbing and Walking Robots*, pages 614–623, Brussels, Belgium, September 2006.

[9] H. Lipson and J. Bongard. An exploration-estimation algorithm for synthesis and analysis of engineering systems using minimal physical testing. In *Proc. of ASME Design Engineering Technical Conferences (DETC)*, 2004.

[10] I.W. Park, J.Y. Kim, and J.H. Oh. Online Biped Walking Pattern Generation for Humanoid Robot KHR-3 (KAIST Humanoid Robot-3: HUBO). *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, pages 398–403, 2006.

[11] Julio C. Pastrana Perez. Gait optimization for humanoid robots. Master's thesis, Albert-Ludwigs-Universitat Freiburg Institut Fur Informatik, 2005.

[12] D.T. Pham and D. Karaboga. Optimun design of fuzzy logic controllers using genetic algorithms. *Journal of Systems Engineering*, 1:114–118, 1991.

[13] E. Schuitema, DGE Hobbelen, PP Jonker, M. Wisse, and JGD Karssen. Using a controller based on reinforcement learning for a passive dynamic walking robot. *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*, pages 232–237, 2005.

[14] R. Tedrake, TW Zhang, and HS Seung. Stochastic policy gradient reinforcement learning on a simple 3D biped. *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, 3, 2004.

[15] J. Velasco and L. Magdalena. Genetic algorithms in fuzzy control systems. In G. Winter, J. Periaux, M. Galan, and P. Cuesta (Eds.), editors, *Genetic Algorithms in Engineering and Computer Science*, pages 141–165. John Wiley & Sons, 1995.

[16] M. Zaratti, M. Fratarcangeli, and L. Iocchi. A 3d simulator of multiple legged robots based on usarsim. In *Proc. of RoboCup Symposium*, 2006.