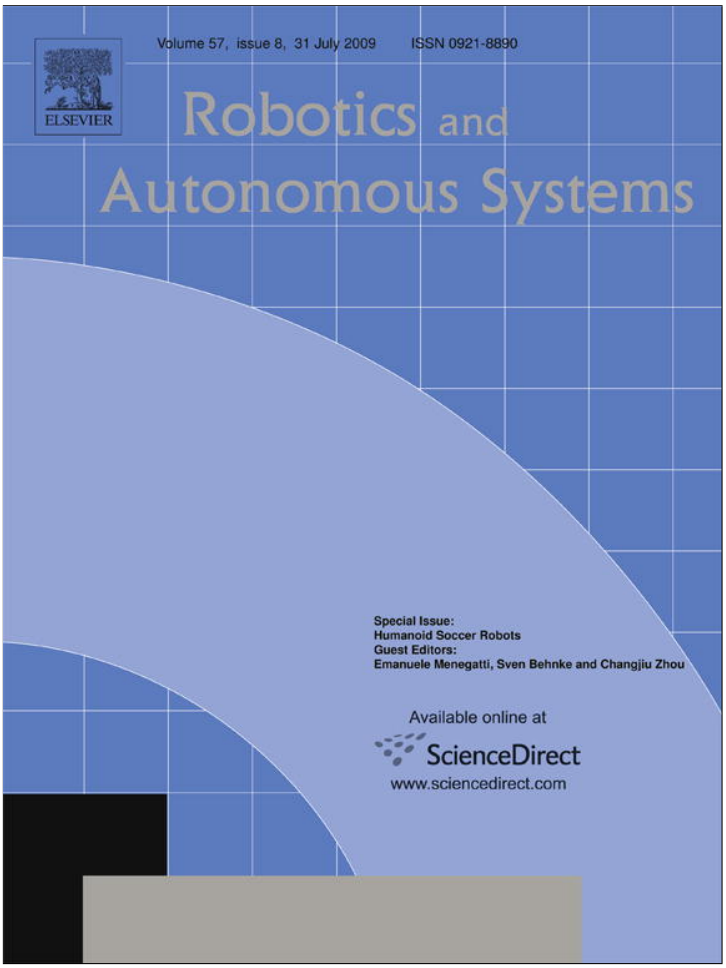

Robotics and Autonomous Systems 57 (2009) 808–818

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

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

Policy gradient learning for a humanoid soccer robot

A. Cherubini, F. Giannone, L. Iocchi *, M. Lombardo, G. Oriolo

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Ariosto 25, 00185 Roma, Italy

ARTICLE INFO

Article history:
Available online 5 April 2009

Keywords:
Humanoid robotics
Machine learning
Motion control

ABSTRACT

In humanoid robotic soccer, many factors, both at low-level (e.g., vision and motion control) and at high-level (e.g., behaviors and game strategies), determine the quality of the robot performance. In particular, the speed of individual robots, the precision of the trajectory, and the stability of the walking gaits, have a high impact on the success of a team. Consequently, humanoid soccer robots require fine tuning, especially for the basic behaviors. In recent years, machine learning techniques have been used to find optimal parameter sets for various humanoid robot behaviors. However, a drawback of learning techniques is time consumption: a practical learning method for robotic applications must be effective with a small amount of data. In this article, we compare two learning methods for humanoid walking gaits based on the Policy Gradient algorithm. We demonstrate that an extension of the classic Policy Gradient algorithm that takes into account parameter relevance allows for better solutions when only a few experiments are available. The results of our experimental work show the effectiveness of the policy gradient learning method, as well as its higher convergence rate, when the relevance of parameters is taken into account during learning.

1. Introduction

In order for robots to be useful for real-world applications, they must adapt to novel and changing environments and situations. However, the ability to deploy a fully autonomous robot in an unstructured, dynamic environment over an extended period of time remains an open challenge in the field of robotics. For this purpose, a popular research activity is the annual RoboCup competition, where different kinds of robots compete on standard reference testbeds in soccer, rescue and @home scenarios. Among the RoboCup disciplines, we focus here on humanoid robot soccer, and among the capabilities that a humanoid robot needs for playing soccer, walking is obviously the most important. Consequently, the speed, stability and precision of the walking gait, are crucial factors that determine the success of a team.

At first, gait improvements for legged robots centered around hand-tuning. However, in case of a change of robot hardware and/or of walking surface (e.g., the carpet, or the floor), the motion parameters need to be re-calibrated, and this is a process that can easily take several hundred trials for an expert. One alternative to hand-tuning a parameterized gait, while enabling the robot to adapt to changes in its surroundings, is to use machine learning to automate the search for good parameters. In the past, various machine learning techniques have proved useful in improving control policies for a wide variety of robots, as reported in Section 2. In fact, machine learning approaches generate solutions with little human interaction, and explore the search space of possible solutions in a systematic way, whereas humans are often biased towards exploring a small part of the space.

Learning in the robotic soccer domain has to overcome several challenges, such as a continuous multi-dimensional state space, noisy sensing and actions, multiple agents (including adversaries), and the need to act in real-time. Nevertheless, this approach has been somewhat fruitful: for example, since the inception of the RoboCup legged league in 1998, the speed of the quadruped robots has increased significantly [1]. In most cases, the knowledge achieved in the Four-Legged League can be transferred to the Humanoid League [2]. Despite this growing interest, considerable work remains to be done, due to the difficulties associated with applying machine learning in the real world. Compared to other machine learning scenarios, such as classification or action learning in simulation, learning on physical robots must be effective with a small amount of data, and should converge in short time [1]. Indeed, it is often prohibitively difficult to generate large amounts of data due to the maintenance required on robots, such as battery changes, hardware repairs, and, usually, constant human supervision.

Following up on all these considerations, in this article, we present a learning method for humanoid walking gaits based on Policy Gradient [3]. We focus on learning a specific humanoid

* Corresponding author.
E-mail addresses: cherubini@diag.uniroma1.it (A. Cherubini), iocchi@diag.uniroma1.it (L. Iocchi), oriolo@diag.uniroma1.it (G. Oriolo).

0921-8890/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2009.03.006

robot task (namely, curvilinear biped walking), that requires, at the same time: a relatively small set of parameters, and high precision from a control viewpoint. To learn this task, we compare Policy Gradient with an extension that takes into account parameter relevance [4], showing, through experiments on a 3D simulator, the higher convergence rate of the extended policy gradient method and, through experiments on real robots, the performance of Policy Gradient learning algorithms and the effective use of the results of learning on the simulator. While Policy Gradient learning methods have been compared on a quadruped robot in [4], the main contribution of the present article is the application of these algorithms to biped walking. We have successfully applied the presented learning method in preparation of RoboCup 2008, within the Standard Platform League (two-legged division)[1] using Aldebaran NAO robots.[2]

The remainder of this article is organized as follows. Section 2 surveys existing approaches for generating and optimizing legged gaits. Section 3 presents the static parameterized biped motion control scheme that we used to generate curvilinear trajectories with a humanoid robot. In Section 4, we describe the policy gradient learning algorithms. Section 5 defines the learning scheme applied to curvilinear walking gait, while experimental results are discussed in Section 6. We close with a discussion and possible avenues for future work in Section 7.

2. Related work

In this section, we provide a brief survey in the field of legged robot gait learning. In particular, we focus on two aspects: gait generation (in particular, we focus on biped gaits), and gait learning. For each aspect, we first review the existing literature, and then outline the original contribution of our work.

2.1. Gait generation

The design of controllers enabling biped robots to walk autonomously on uneven and variable terrains in a robust way (e.g. in daily life) remains a crucial research topic in robotics. Recent works in the field of legged robot gait modeling and control have been surveyed in [5,6]. In summary, two main classes of methods have been used for biped gait control: static and dynamic control approaches.

The most commonly used dynamic approaches are based on the zero moment point (ZMP). ZMP refers to the location within the base of support of the center of pressure of the floor reaction force. By controlling this location, the robot may induce forward motion while maintaining dynamic balance. In the dynamic walking pattern used in [7], for instance, ZMP is used during the double support phase to guarantee stability. In [8], ZMP is integrated in a whole body cooperative dynamic biped walking, that includes trunk motion to compensate for the moment generated by the motion of the whole body (lower, as well as upper limbs). For a survey of the history and characteristics of ZMP, the reader is referred to [9].

In static approaches, instead, some a priori definition of the desired trajectories to follow is used. Standard and ad-hoc control techniques have been developed to cope with model uncertainties, obstacles and disturbances, in order to prevent the robot from falling. The desired (also called reference) trajectories can be either obtained from the capture of human motions, or purely computer-generated. In the second case, the usual approach includes two steps. First, a set of output variables (generally, the 3D coordinates of a few selected points on the robot) with adequate dimension is chosen. Second, after parametrization (e.g., under the form of splines) the desired trajectories of these variables are computed, for every phase of the gait. Computer-generated trajectories have been used in [10–13]. In [14], the trajectories are designed using Central Pattern Generators, similar to bio-inspired self-oscillating systems.

In this work, we utilize a static parameterized biped gait model to control the legs and arms in order to make NAO track arbitrary curvilinear trajectories. We have decided to adopt a static gait model, since no accurate dynamic model of NAO is available at this time. The trajectories are defined by the requested velocity command $(v, \omega) \in \mathbf{R}^2$, with v the forward linear velocity, and ω the angular velocity, as for non-holonomic (e.g., wheeled) robots. Our gait model is similar to the one presented in [11]. The model is based on arbitrary parameterized joint trajectories, and does not explicitly consider stability aspects; hence, the gait model performance must be assessed through its practical application. However, in contrast with most of the aforementioned works, which focus uniquely on pure rectilinear walks (with the exceptions of [11] and [12]), the motion control scheme that we propose is valid for generic curves of radius $R = v/\omega$. This includes the particular cases of pure rectilinear (ω null, thus $R = \infty$) and pure rotational (v null, thus $R = 0$) walks. With this approach, our walk reproduces a natural-looking human walk. Indeed, the close relationship between the shape of human walking paths in goal-directed movements and the kinematics model of a non-holonomic mobile robot has been shown in [15] (humans 'do not walk sideways'). From a comparative analysis, the authors of [15] infer that some constraints (mechanical, anatomical ...) act on human bodies restricting the way humans track trajectories, and that the trunk can be considered as a kind of steering wheel for the human body. Following up on these considerations, and on the fact that the NAO trunk is not actuated, we utilize the arms to add momentum for rotation, in contrast with the other (rare) curvilinear motion control schemes presented in the literature [11,12], which exploit only the lower limbs for rotations.

2.2. Gait learning

Robots should be able to respond to changes in the surroundings by adapting both their low-level skills (e.g., vision or motion control parameters) and the higher-level skills (e.g., the behaviors) which depend on them. Such adaptation should occur as autonomously as possible. However, in all cases where the complete analytical model is unknown, this is not trivial. Thus, machine learning techniques have been used in many robotic applications, both for finding optimal parameter sets of specific behaviors (*parameter learning*), and for determining the best choice of behaviors required to accomplish a task (*behavior learning*), whenever the model cannot provide an optimal solution to such problems. Clearly, one of the primary application areas of parameter learning is robot motion control, since the mathematical model is approximated, and traditional optimization methods cannot be used. In particular, gait optimization for legged soccer robots is not straightforward: creating effective motions is a challenging task, since there are many parameters to be set, and since successful motions strongly depend on many factors, which cannot be modeled with precision: playing surface, robot hardware, and game situation. For these reasons, in recent years, machine learning techniques have been used to find optimal parameter sets for legged gait optimization.

Hexapod robot walk generation has been solved in [16], with a Genetic Algorithm. Similar methods have been used for optimization of the vector of quadruped walk parameters, while avoiding the need for gradient approximation in [17–19]. Kohl and Stone,

[1] www.tzi.de/spl/.

[2] www.aldebaran-robotics.com/eng/.

from the University of Texas at Austin [20], empirically compared four different machine learning algorithms for quadruped walk optimization, and in [21], genetic algorithms were extended in order to improve omnidirectional gaits by switching and interpolating between vectors of parameters. A common feature of these approaches is that the robots time themselves walking across a known, fixed distance, thus eliminating the need for human supervision, other than battery changes. Parameter learning has proved very effective for improving other motion control tasks, such as robot grasping. This task is achieved in [22] by applying the *layered learning* paradigm [23]: grasping parameters rely on previously learned walk parameters. Similar approaches have been used for learning on humanoid platforms. Humanoid walk parameters are automatically learned using: various evolutionary strategies in [11], a particle swarm approach in [13], and Reinforcement Learning (RL) in [24]. Sato and others [14] design a new RL algorithm for central pattern generators, in order to improve biped gait stability. A method for optimally generating stable bipedal walking gaits, based on a Truncated Fourier Series Formulation, with coefficients tuned by a Genetic Algorithm, is presented in [25]. Policy gradient RL, and particle swarm optimization are compared in [26] for improving a biped gait. Morimoto and Atkeson [27] present a model-based RL algorithm for biped walking, in which the robot learns to appropriately modulate an observed walking pattern. In [10], a method for acquisition of highly energy-efficient walking, based on a two-stage genetic algorithm, is presented: in the first phase, the fitness function consists of a walking distance (longer is better), whereas in the second phase, the fitness function consists of a walking distance (longer) and energy consumption (less). Fundamental soccer skills for a real humanoid robot are improved, by extending standard RL with imitation of a teacher, and reevaluation of past experiences, in [28]. A humanoid robot, able to learn how to interact with the environment, and how to develop its perceptual, motor and communication capabilities, has been designed in the research project *RobotCub* [29]. Researchers at the University of Padova have developed an interface for teaching new motions to humanoid robots through touching, by directly manipulating the limbs of the robot [30].

Following up on these works and on our previous work on the Sony AIBO [4,31], in this article we propose an original parameter learning approach for improving curvilinear biped walks. In [31], we have shown the effectiveness of the layered learning approach [23], that is suitable with the large dimensions of the search space in complex robot learning scenarios. We have also studied how to use a 3D simulator for speeding up robot learning, and we have shown that the learned low-level parameters are strongly related to the desired behavior. In [4], we have proposed an extension of the policy gradient algorithm introduced in [3], that has been successfully used to optimize quadruped gait of AIBO robots. Our approach guarantees a higher convergence rate than the standard policy gradient, by exploiting additional information on the system properties, such as the *contiguities* between strategies, and the *relevance* of the behavior parameters. As reported in [4], the extended policy gradient has been tested in the application example of an attacker robot in the RoboCup Four-Legged League. Referring to the cited works, the original contribution of this article, from the learning point of view, is the experimental comparison between the policy gradient from [3], and the extended version introduced in [4], carried out for the first time on a *biped* platform. Moreover, in contrast with the algorithm presented in [4], in this article we only exploit parameter relevance, and not strategy contiguity, since the article focuses on parameter learning rather than behavior learning.

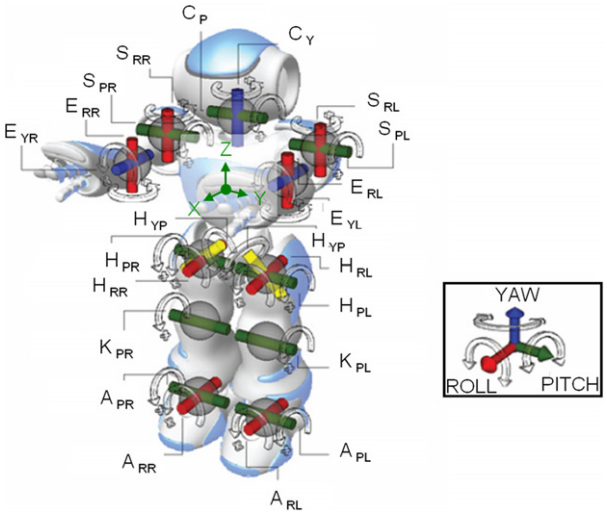

Fig. 1. The Aldebaran NAO robot with its 21 degrees of freedom, and reference frame $\mathcal{F}_{\mathcal{R}}$ (X, Y, Z) with origin in the robot center of mass.

3. Motion control

3.1. Outline

The robot used in this work is NAO from French manufacturer Aldebaran. It has been selected for the soccer competitions of the RoboCup Standard Platform League. NAO has a total of 21 degrees of freedom, shown in Fig. 1: 2 in the head, 4 in each arm, 1 in the pelvis (H_{YP}) and 5 in each leg. We define the reference frame $\mathcal{F}_{\mathcal{R}}$ (also shown in the figure) with origin in the robot center of mass, X axis in the sagittal plane and pointing forward, Y axis orthogonal to the sagittal plane and pointing to the left side of the robot, and Z axis orthogonal to the ground and pointing upwards. In this work, we assume for simplicity that, throughout the gait, the robot center of mass stays in a fixed position with respect to the robot trunk.

The control software of the robot is based on the OpenRDK software framework.[3] [32] In the OpenRDK architecture, the *MotionControl* module is responsible for implementing all requested velocity commands, which are defined as $(v, \omega) \in \mathbf{R}^2$, with v the forward linear velocity along the X axis, and ω the angular velocity around the Z axis (positive counterclockwise). In the rest of this section, we describe the parameterized biped curvilinear gait model used in the *MotionControl* module. Our gait model (which is inspired by the model presented in [11]) has been designed in order to keep the number of parameters (and consequently the search space dimension for parameter optimization) as small as possible. It is based on arbitrary parameterized joint trajectories, and does not explicitly consider stability issues; its soundness will therefore be proved in practical applications.

The act of biped walking involves, for each gait cycle (i.e., for each step), both a single support (or swing) phase, and a double support phase. Therefore, two fundamental parameters of the step, are:

- T_{tot}: the total duration of a step,
- $ss_{\%} \in]0, 1[$: the ratio between single support duration and total step duration.

In the following, we describe the motion control task by dividing it into four subtasks: The design of the foot trajectories in the X–Z plane, the design of the center of mass trajectory in the lateral direction, the Hip Yaw/Pitch joint control used to make the robot turn, and the arm control.

[3] openrdk.sf.net.

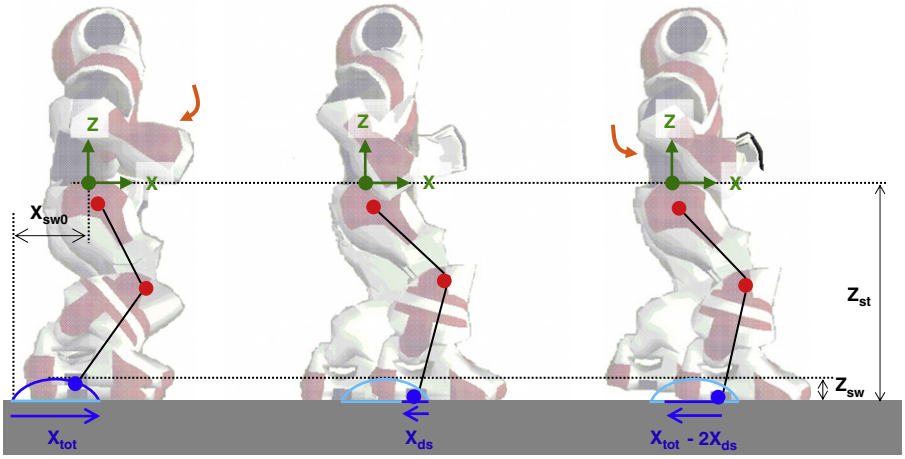

Fig. 2. Right foot trajectory in the X–Z plane, and characteristic parameters. Left to right: right foot swing, double support phase, and start of left foot swing. The right shoulder pitch movement is also outlined.

3.2. Foot trajectories in the X–Z plane

Here, we present the foot trajectory design for rectilinear walks $(v, 0)$. We show in the following that generic curvatures can be obtained by combining such trajectories with the appropriate Hip Yaw/Pitch control law necessary for pure rotations $(0, \omega)$. Since we focus on rectilinear walks, the foot trajectories belong to the X–Z plane. The legs move in two phases: the phase when the foot (*stance foot*) is on the ground to push the body forward, and the phase when the foot (*swing foot*) is in the air to prepare the next step. We design the trajectory of the stance foot in $\mathcal{F}_{\mathcal{R}}$ as a straight line, and the $\mathcal{F}_{\mathcal{R}}$ swing trajectory as a semi-elliptical trajectory (see Fig. 2). The joint angles required to enable the feet to track the rectilinear and semi-elliptical trajectories shown in Fig. 2 are calculated by means of inverse kinematics. Since the trajectories belong to the X–Z plane, the three leg pitch joints (hip, knee and ankle) are sufficient to control each foot during each phase.

As illustrated in Fig. 2, the foot trajectories depend on the following 5 parameters:

- X_{tot}: the total length of a step, which is related to the requested linear velocity v and to the step duration T_{tot} by equation:

$$X_{tot} = v\, T_{tot} \tag{1}$$

- X_{sw0}: the $\mathcal{F}_{\mathcal{R}}$ abscissa of the swinging foot at the beginning of the single support phase,
- X_{ds}: the portion of X_{tot} covered during one double stance phase,
- Z_{st}: the stance foot altitude in $\mathcal{F}_{\mathcal{R}}$,
- Z_{sw}: the maximum height from the ground reached by the swinging foot.

3.3. Using the leg roll joints to control the center of mass and the trunk

In order to guarantee stability during the single support phase, the robot must 'swing' sideways during motion, as is commonly done in static biped gaits [11,13]. This is done by shifting the center of mass of the robot in the direction of the stance foot, which is still on the ground. A qualitative representation of the desired trajectory for the projection of the center of mass in the X–Y plane during rectilinear walks is shown in Fig. 3(a). This trajectory comprises a sinusoid above the stance foot area during single support, and a linear trajectory during double support, which moves the center of mass to the next swinging foot, in order to prepare the next step.[4] This trajectory is obtained by moving the stance feet in $\mathcal{F}_{\mathcal{R}}$. We have seen that gait stability is also increased by letting the robot trunk roll around the X axis during the gait; this is done using a sinusoidal time law of amplitude K_R, which is driven by the stance hip roll joints. We have designed the foot and trunk movements by using the 6 parameters: X_{tot}, X_{ds}, Y_{ft}, Y_{ss}, Y_{ds}, and K_R. The first two parameters have been defined in Section 3.2. The four other parameters are:

- Y_{ft}: the distance between the feet during the gait,
- Y_{ss}: the maximum $\mathcal{F}_{\mathcal{R}}$ ordinate of the feet during the single support phase,
- Y_{ds}: the maximum $\mathcal{F}_{\mathcal{R}}$ ordinate of the feet during the double support phase,
- K_R: the amplitude of the trunk roll sinusoidal law.

3.4. Exploiting the NAO Hip Yaw/Pitch joint for turning

In order to implement rotational motions around Z (i.e., in order to realize velocity requests with $\omega \neq 0$), we utilize the single NAO Hip Yaw/Pitch joint H_{YP} (see Fig. 1). The movement is done by alternatively increasing and decreasing the joint value during consecutive single support phases (see Fig. 4), and locking it during the double support phases. The rotational movement is designed directly in the joint space, as opposed to the movements described previously, which were designed in the Cartesian space, and then mapped to the joint space via inverse kinematics.

Consider two consecutive steps. During the first swing phase, the Hip Yaw/Pitch joint value is increased smoothly (using a cosinusoidal law) from 0 to a fixed value H_{YPM}, whereas, during the following step, the yaw pitch joint value is decreased (again, using a cosinusoidal law) to restore parallelism (i.e., $H_{YP} = 0$) between the feet. Since the rotation axis of the NAO Hip Yaw-Pitch joint is the bisectrix of the Y and Z axis, the total angular width of a double rotation step driven by the above time laws is $\frac{H_{YPM}}{\sqrt{2}}$. Thus, H_{YPM} is related to the requested angular velocity ω and to the step duration T_{tot} by:

$$H_{YPM} = -2\sqrt{2}\,|\omega|\,T_{tot}. \tag{2}$$

Note that the sign of H_{YPM} is always negative. The sign of ω is determined by the initial foot swing (e.g., in Fig. 4, negative ω is obtained by swinging the right foot first). To countervail the effect of the NAO Hip Yaw-Pitch joint value $H_{YP}(t)$ on the foot trajectories in the X–Z plane, we subtract, throughout the gait cycle, $\frac{H_{YP}(t)}{\sqrt{2}}$ from the hip pitch joints calculated with the method described in Section 3.2.

[4] Sinusoidal trajectories have been chosen here (and elsewhere in our gait model) for their smoothness and energetic efficiency.

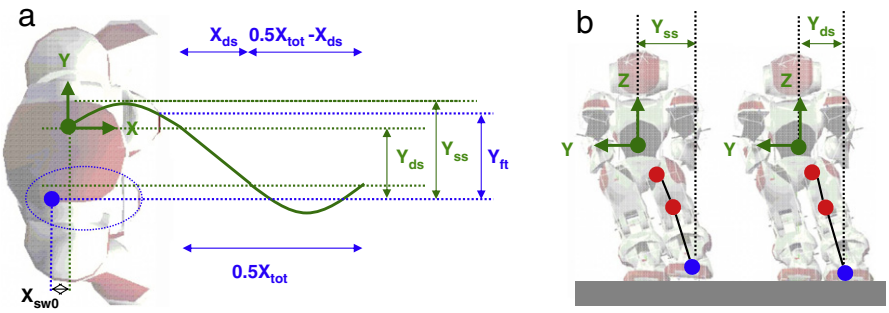

Fig. 3. Characteristic parameters of the leg roll joint control used for moving the center of mass sideways. (a) Top view, with qualitative trajectory of the projection of the center of mass in the X–Y plane and initially right foot (dotted) swinging. (b) Rear view during single support (left) and double support (right) phases.

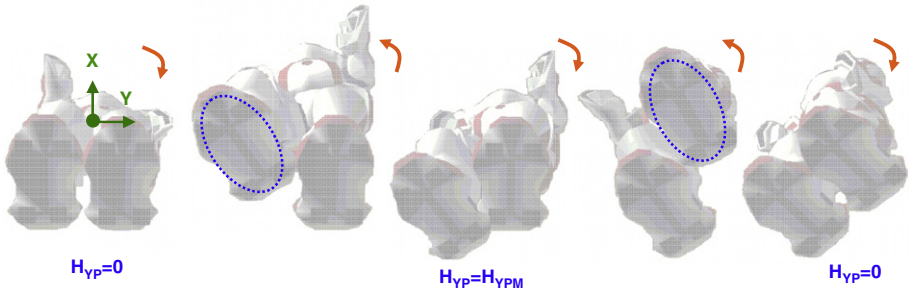

Fig. 4. Bottom view showing how the NAO Hip Yaw/Pitch joint is used for turning at $\omega < 0$ over two steps (the swinging foot is dotted). Left to right: double support, right swing, double support, left swing, double support. The verse of the angular momentum induced by shoulder pitch movement is also indicated by the curved arrows.

3.5. Arm control

As the human walk shows, arm movement synchronized with leg movement can improve gait performance. Similarly to leg yaw-pitch control, we have decided to design the arm movement directly in the joint space. We only consider the pitch shoulder joints, and lock the 4 elbow and the 2 shoulder roll joints. Since conservation of angular momentum implies that for rotating bodies, a decrease in the radius is accompanied by an increase in the angular velocity (e.g., the ice skater spins faster when the arms are drawn in, and slower when the arms are extended), we lock the shoulder and elbow roll joints to the zero positions, and fix the elbow yaw joint to 90°.

We design shoulder pitch movements coupled with the leg movement and symmetric with respect to the vertical downward arm position. Two different arm control laws are designed for pure rectilinear walks and pure rotations, and the linear combination of such laws is used for curvilinear walks. For rectilinear walks, the angular momentum induced by the arms must cancel the undesired momentum around the Y axis generated by leg swinging. We use a sinusoidal time law during the swing phases, and lock the shoulder pitch joint, during the double support phases. The arm corresponding to the swinging leg moves backward and vice-versa, as shown in Fig. 2. Instead, for pure rotations, the angular momentum induced by the arms should enhance the momentum around Z that is used to rotate. We use sinusoidal time laws during both phases. During the swing phase, the arm movement generates a momentum with the same verse of ω, while during the double support phase, the arms are moved back to prepare the next swing phase (see Fig. 4). For all walks (i.e., for all velocity commands (v, ω)) the arm is controlled by one parameter: the amplitude K_S of the shoulder pitch joint sinusoidal time laws.

4. Policy gradient learning

In this article, we experiment with learning techniques for improving the generation of walking behaviors, using the motion control scheme described in the previous section. In particular, we focus on Policy Gradient methods, that are recalled in this section. As in any robot learning problem, the applied method should minimize time consumption, which is often related to 'hardware consumption' in real robot applications. Hence, the algorithm must converge to the best solution (or to a good solution) after as few experiments as possible. When experimenting with learning techniques for humanoid robots, this issue is even more demanding, since bad walking behaviors can cause the robot to fall to the ground and possibly damage itself.

To overcome this problem, we start from this observation: during the learning process, the relevance of the parameters (i.e., their impact on finding an optimal solution) is not constant. In particular, in many cases, some of the parameters reach good or optimal values after few experiments, while other parameters need many experiments. If we are able to appropriately reduce the search space by removing the directions for which we have already found an optimal value (i.e., the directions that are not relevant anymore for determining the optimal solution), the number of experiments needed could be reduced, without losing the quality of the solution.

Unfortunately, this choice cannot be made in an ideal manner, and in practice reducing the search space implies a reduction in the quality of the solution. On the other hand, reducing the search space also means that, with the same number of experiments, it is possible to explore the search space in more depth. Consequently, there should be a balance between the number of experiments and the quality of the solution.

In this article, we want to verify that reducing the search space by removing irrelevant parameters during the learning process may lead to an optimal solution after few experiments. In order to verify this statement, we adopt the Policy Gradient learning method, which has been successfully used for robot learning [3,31] and is suitable for a modification that takes into account the relevance of parameters [4], and apply it to biped curvilinear walk.

In the remainder of this section, we recall the Policy Gradient algorithm and its extension that takes into account relevance of

parameters, while in the following section, we describe how these methods have been applied to learning curvilinear humanoid gaits.

4.1. Policy gradient

The learning problem considered in this article is formulated as a policy gradient reinforcement learning problem [3], that considers each possible set of parameter assignments, and defines an open-loop policy that can be executed by the robot. Assuming that the fitness function (or objective function) $F(\underline{X})$ is differentiable with respect to each of the parameters, the Policy Gradient (PG) algorithm estimates its gradient in the parameter space $\underline{X} \in \mathbf{R}^k$, and follows it toward a local optimum $\underline{X}^*$.

The parameter optimization approach based on Policy Gradient starts from an initial parameter set $\underline{X}^0$ and proceeds to estimate the partial derivative of $F(\underline{X})$ at $\underline{X}^0$ with respect to each parameter. From the initial set $\underline{X}^0$, p randomly generated *policies* ${}_m\underline{X}^0$ ($m = 1, \ldots, p$), near $\underline{X}^0$, are evaluated. The number of policies p is proportional to the search space dimension: $p = \Lambda k$. Each of the p policies is generated as: ${}_m\underline{X}^0 = \underline{X}^0 + [\rho_1, \ldots, \rho_k]^{\mathrm{T}}$, and each *perturbation* ρ_j is chosen randomly in the set $\{-\epsilon_j, 0, +\epsilon_j\}$. Each ${}_m\underline{X}^0$ is grouped into one of three sets for each j: $G_{-\epsilon,j}$, $G_{0,j}$ or $G_{+\epsilon,j}$, depending whether its j^{th} parameter was obtained by adding $-\epsilon_j$, 0 or $+\epsilon_j$. After evaluating the fitness function at each policy ${}_m\underline{X}^0$, average scores $\bar{F}_{-\epsilon,j}$, $\bar{F}_{0,j}$, and $\bar{F}_{+\epsilon,j}$ are computed for $G_{-\epsilon,j}$, $G_{0,j}$ and $G_{+\epsilon,j}$, respectively. These scores are used to construct an estimate of the *gradient* $\nabla\underline{X}^0$, which is then normalized, multiplied by a scalar *step-size*, noted η, and added to $\underline{X}^0$, to determine $\underline{X}^1$ and begin the next iteration ($i = 1, \ldots, N_{iter}$). The step-size η is fixed to a constant value throughout the learning process.

The algorithm usually terminates after a predefined number of iterations N_{iter}, and it is proved to converge to a local optimum $\underline{X}^*$ if a sufficient number of iterations are performed. In practice, at each iteration of the algorithm, p experiments must be executed to evaluate the fitnesses of the p policies necessary for estimating the gradient $\nabla\underline{X}$. Hence, the total number of experiments necessary to complete N_{iter} iterations is:

$$N_{PG} = pN_{iter} = \Lambda k N_{iter}. \tag{3}$$

4.2. Extended policy gradient with parameter relevance

Although effective, the method described above does not consider the variable relevance of parameters during the learning process. Here, we present an extension of the method, which takes into account relevance of parameters to speed up the learning process. The algorithm presented in this section is derived from the one described in [4], by considering only the relevance of the parameters, and not the contiguities among strategies, since strategies are not taken into account in this article.

Let us define the following metric for measuring the *relevance* of the parameters.

Definition. *Relevance* of parameter j at iteration $\bar{i} \neq 0$ is the norm of the weighted average of the jth gradient component of vector X:

$$R^{\bar{i}}(j) = \frac{\left| \sum_{i=1}^{\bar{i}} \lambda^{\bar{i}-i} \nabla X_j^i \right|}{\sum_{i=1}^{\bar{i}} \lambda^{\bar{i}-i}}$$

where $\lambda \in]0, 1[$ is a *forgetting factor*, that operates as a weight diminishing for the more remote data.

POLICY GRADIENT WITH PARAMETER RELEVANCE

INPUT: $\underline{X}^0$, η, N_{iter}, T_r, PR_{start}
OUTPUT: $\underline{X}^*$

```
1  begin
2  initialize X ← X^0, J^0 ≡ {1, ..., k}
3  for each iteration i = 1 to N_iter
4      generate p^{i-1} random policies _mX^{i-1} near X^{i-1} on J^{i-1}
5      evaluate F(X) at all p^{i-1} policies _mX^{i-1}
6      J^i ← ∅
7      for each parameter j ∈ J^{i-1}
8        evaluate F̄_{-ε,j}, F̄_{0,j}, and F̄_{+ε,j}
9        if F̄_{0,j} > F̄_{-ε,j} and F̄_{0,j} > F̄_{+ε,j}
10         ∇X^i_j ← 0
11       else
12         ∇X^i_j ← F̄_{+ε,j} − F̄_{-ε,j}
13       endif
14       evaluate R^i(j)
15       if (R^i(j) > T_r or i < PR_start)
16         J^i ← J^i ∪ j
17       endif
18     endfor                    %parameters
19     ∇X^i ← η × ∇X^i / |∇X^i|
20     X^i ← X^{i-1} + ∇X^i
21 endfor                        %iterations
22 return X^*
23 end
```

Fig. 5. Pseudo-code for Policy Gradient with Parameter Relevance algorithm.

The above definition of relevance is used to estimate how much the parameter can contribute to find an optimal solution. Small values of the relevance of parameter j imply that the estimated gradient varies 'slightly' along the jth component during the learning process: hence, we assume that parameter j has little relevance on the system performance, and that it is not necessary to search in this direction anymore.

The Policy Gradient with Parameter Relevance (PGPR) algorithm is shown in Fig. 5. The algorithm is similar to Policy Gradient, with the main difference that policies are computed only on the relevant parameters. More specifically, we denote with $J^i \subseteq \{1, \ldots, k\}$ the subset of relevant parameters at iteration i (i.e., parameters j with 'high' $R^i(j)$). After PR_{start} iterations, J^i is updated at each step during the learning process, depending on the values of $R^i(j)$ (lines 14–16). At each iteration, policies are computed only in the directions given by the relevant parameters, i.e. only by the parameters in J^{i-1} (line 7). Correspondingly, at each iteration, the number of policies generated is also updated according to the number of relevant parameters: $p^i = \Lambda|J^i|$.

An example of application of the algorithm is outlined in Fig. 6. The figure presents two cases of parameter search for the drawn 2D fitness functions. In each figure, the light colored dots show the values of $\underline{X}^i$ computed at each step (starting from the leftmost one), while the dark colored dot indicates the solution $\underline{X}^*$ found after a few iterations ($N_{iter} = 6$ in the figure). The horizontal dotted line shows the search direction after one of the parameters (the one corresponding to the vertical axis) is fixed. In the left picture, the advantage of the algorithm is clear: after a few iterations, the parameter corresponding to the vertical axis is fixed and the search continues only on the horizontal axis (dotted line), rather than on the entire 2D space. This obviously speeds up convergence and improves the quality of the solution when considering a small number of trials. The right picture shows a case where the algorithm does not behave in an optimal way. Here, after some iterations, the parameter associated to the vertical axis is fixed to a value such that the consequent 1-D search does not guarantee to find the optimal solution. However, even in this case, if we consider only a limited number of experiments, the extended algorithm provides a better solution than the classic algorithm. In fact, during the PGPR learning process, since the number of

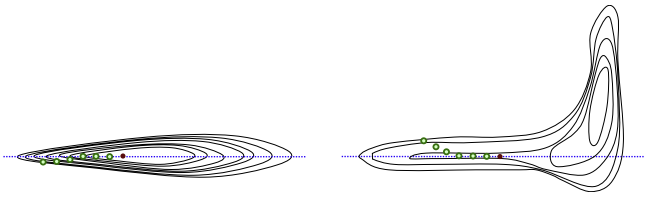

Fig. 6. Example of application of the PGPR algorithm.

relevant parameters (i.e., $|J^i|$) is reduced progressively, the number of policies $p^i = \Lambda|J^i|$ also diminishes. Hence, the total number of experiments necessary to complete N_{iter} iterations, is:

$$N_{PGPR} = \sum_{i=0}^{N_{iter}} p^i = \Lambda \sum_{i=0}^{N_{iter}} |J_i|. \tag{4}$$

Thus, if we assume that at least one parameter is considered irrelevant and therefore discarded during the learning process (otherwise, the two algorithms will behave identically) and if we fix the number of iterations N_{iter} (i.e., the number of times the gradient is estimated), it is obvious from Eqs. (3) and (4) that $N_{PGPR} < N_{PG}$.

This shows the main characteristic of the proposed PGPR algorithm: although it does not guarantee to reach a local optimum (as in the example on the right in Fig. 6), in many cases it requires less experiments to compute the same solution (in other words, it tends to find good solutions as early as possible).

As shown in the next sections, the application of this algorithm is very suitable for robot learning tasks, where experiments are expensive and a non-optimal solution obtained with few experiments is often preferred to an optimal solution that requires a larger number of experiments.

5. Learning configuration

5.1. Parameters

The objective of the learning process is to learn the optimal parameter vector $\underline{X}^*$ that ensures the best performance (i.e., the maximum value of fitness function $F(\underline{X})$) for any curvilinear walking gait (i.e., for arbitrary velocity requests $(v, \omega) \in \mathbf{R}^2$). The vector $\underline{X}$ is composed of the following eleven parameters (detailed in Section 3):

- $ss_{\%}$: the ratio between single support duration and total step duration,
- X_{tot}: the total length of a step,
- X_{sw0}: the $\mathcal{F}_{\mathcal{R}}$ abscissa of the swinging foot at the beginning of the single support phase,
- X_{ds}: the portion of X_{tot} covered during a double stance phase,
- Z_{st}: the stance foot altitude in $\mathcal{F}_{\mathcal{R}}$,
- Z_{sw}: the maximum height from the ground reached by the swinging foot,
- Y_{ft}: the distance between the feet during the gait.
- Y_{ss}: the maximum $\mathcal{F}_{\mathcal{R}}$ ordinate of the feet during the single support phase,
- Y_{ds}: the maximum $\mathcal{F}_{\mathcal{R}}$ ordinate of the feet during the double support phase,
- K_R: the amplitude of the trunk roll sinusoidal law,
- K_S: the amplitude of the swinging shoulder movement.

Note that in the above list of parameters to be learned, we have not considered T_{tot} and H_{YPM} since Eqs. (1) and (2) show that these parameters are constrained respectively to v and X_{tot}, and to ω and T_{tot}. Moreover, we decided to fix T_{tot} allowing for different velocities, obtained by varying X_{tot}.

Although these parameters allow for a curvilinear walking gait, assigning:

$$X_{tot} = X_{ds} = \mathbf{0} \qquad X_{sw0} = \bar{X}_{sw0}$$

where $\bar{X}_{sw0}$ is the X_{sw0} value that guarantees zero moment around the Y axis (i.e., that the ZMP is the projection of the robot center of mass on the ground), turns the gait into a pure rotational walk.

Moreover, all the parameters are box-constrained by the gait designer, due to the physical characteristics of the system, and we call Δ_j the range size for parameter j. Hence, a candidate solution of the optimization problem for curvilinear walk is:

$$\underline{X} = \left[ss_{\%}\ X_{tot}\ X_{sw0}\ X_{ds}\ Z_{st}\ Z_{sw}\ Y_{ft}\ Y_{ss}\ Y_{ds}\ K_R\ K_S\right]^{\mathrm{T}} \in \mathbf{\Theta} \subset \mathbf{R}^{11}. \tag{5}$$

5.2. Fitness function

The appropriate choice of the fitness function for optimization is fundamental. Here, we evaluate the quality of a walking gait by taking into account the speed and the precision of the robot motion with respect to the desired trajectory. Hence, we adopt the following function:

$$F(\underline{X}) = \lambda_L L(\underline{X}) + \lambda_P P(\underline{X}) \tag{6}$$

where $L(\underline{X})$ and $P(\underline{X})$ are metrics indicating respectively the length of the path covered in a fixed amount of time, and the precision with respect to the desired trajectory at the end of the experiment, measured for parameter set $\underline{X}$. The positive weights λ_L and λ_P indicate the importance of these two measures. Here, we simply report the derivation of L and P in the case of circular trajectories (i.e., case of finite, non-null R), without considering the cases of pure rotation and pure rectilinear walks. To test parameter set $\underline{X}$, we make NAO walk, for a fixed period of time, along a circular trajectory of radius $|R|$ (i.e., we apply a motion request (v, ω) such that $v = \omega R$). Refering to Fig. 7, we note $e(\underline{X})$ the Euclidean distance between NAO's final position and the circumference, and $d_{re}(\underline{X})$ the signed distance covered by the robot during the experiment. The sign of d_{re} is positive if NAO walked in the correct verse (i.e., if the average radius during the real walk had the sign of R). Then, the variables introduced in (6), can be computed as:

$$L(\underline{X}) = \frac{d_{re}(\underline{X})}{v}$$

$$P(\underline{X}) = 1 - \frac{e(\underline{X})}{|R|}$$

Note that $F(\underline{X})$ can become negative for bad parameter sets. Note also that, although this formulation of the fitness function does not take directly into account robot falls, it does reward 'late' falls, with respect to 'early' ones.

6. Experimental results

The objectives of the experiments reported in this section are: (1) demonstrating that the Policy Gradient algorithm is adequate for learning optimal parameters of the curvilinear walk described in this article; (2) evaluating the approach based on parameter relevance and assessing its advantage when only a limited number of experiments is available. The experiments have been performed both on the Webots 3D simulator[5] and on real NAO robots.

[5] www.cyberbotics.com.

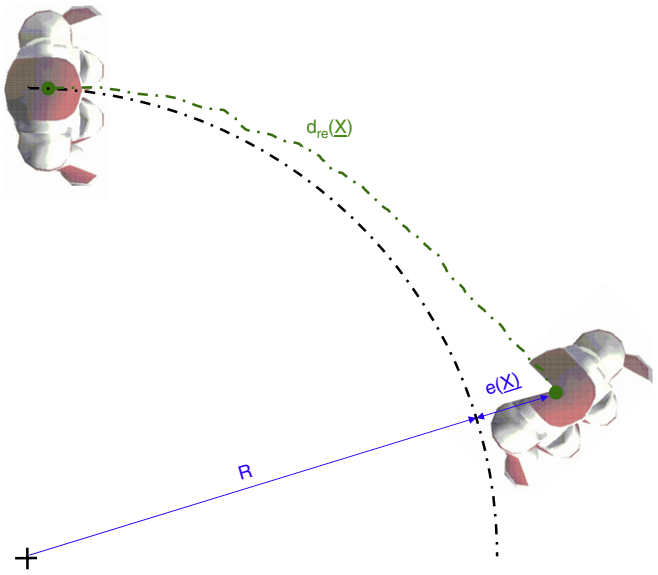

Fig. 7. Relevant variables used to compute the fitness of a parameter set $\underline{X}$. The robot center of mass is indicated with a circle. The ideal and real trajectories are represented respectively by the dark and light dashed curves.

For the experimental evaluation described in this section, we took advantage of a result from our previous work [31]: the utility of using a 3D simulator for speeding up the learning process. In [31], we had shown that simulated learning can be used in a layered approach as a starting point for learning on the real robot. Moreover, simulations are in general less biased by noise, and require less resources (both in terms of human operators and in terms of hardware) than real world experiments. On the other hand, running very long runs on the simulation does not guarantee better parameters for the real robot. In fact, we found out that after a certain number of learning iterations we obtained specific solutions for the simulator that did not correspond to good solutions for the real robot. Therefore, the length of the experiments in simulation has been limited to 20–25 iterations.

In the following, we will thus present first a set of learning sessions using Webots, and then sessions on the real NAO. Simulations provide information about the convergence of the algorithms for a robot learning application, while real world experiments are used to show that learning is effective also on real robots.

In both simulated and real experiments, we have focused on learning the best parameter set with fixed T_{tot} and variable velocity v, which depends on the step length X_{tot}, in order to maximize both velocity and precision in executing a curvilinear walk.

6.1. Experiments with Webots simulator

For performing quantitative measures of the proposed learning approaches, we made use of the Webots 3D simulator, that includes an official model of the NAO humanoid robot. The learning task described in the previous section has been performed several times, and we report here some relevant results that highlight the characteristics of the Policy Gradient (PG) and the Policy Gradient with Parameter Relevance (PGPR) algorithms. Referring to the 11 parameters shown in (5), we have decided to fix parameters $ss\%$, K_S and Y_{ft} to hand tuned values, and learn the remaining parameters (thus, $k = 8$). The number of policies used to estimate the gradient is set to 16 (i.e. $p = \Lambda k$, with $\Lambda = 2$).

With this configuration, we performed two series of experiments: (1) running several learning sessions starting from the same initial parameter set, in order to attenuate the effects of the randomized choices of the algorithms, while comparing PG and PGPR; (2) running different learning session for different initial values to evaluate the different performance of PG vs. PGPR.

6.1.1. Experiments with the same initial value

In the first set of experiments, we have performed many learning sessions starting from the same initial parameter set, in order to reduce the randomness of the results. The objective is to analyze the relevance of parameters, and to compare PG and PGPR in terms of both the number of experiments performed and the quality of the solution.

The experimental procedure is the following:

- choose an initial parameter set that is the same in all runs;
- perform 5 complete learning sessions using PG algorithm for a fixed number of iterations ($N_{iter} = 20$) and analyze the variance of the results (for the fitness values as well as for the final parameter set);
- identify the parameters with low relevance during the PG learning process;
- run PGPR starting from intermediate results of the PG learning run and fixing the parameters with low relevance
- evaluate the reduction in the number of experiments and the differences in the quality of the solutions between PG and PGPR.

The results of the simulations are summarized below. Further details are given in www.diag.uniroma1.it/~iocchi/RobotExperiments/HumanoidLearning.

- The fitness increases in a regular way and there is low variance between the 5 simulations (see Fig. 8). The learning process allows for an increase of performance from policies with fitness 1.55 ± 0.05 at the first iteration to final results of 2.34 ± 0.06.
- The five simulation runs do not converge towards the same parameter set (see Table 1). In particular, some parameters converge towards the same values (low variance in the final solutions), while others tend to assume different values (high variance). This can be explained by the fact that in the simulator some parameters are less relevant and do not contribute in a significant way to the optimal solution. For example, the height of the foot from the ground Z_{sw} does not affect the quality of the solution since the model of the simulator is not precise enough to take into account uneven floor and frictions between the foot and the ground.
- Between iteration 10 and iteration 20, 3 to 5 parameters have been considered not relevant. As shown in Table 2, some of them are different in the 5 simulations and stabilize at different iterations. In almost all cases, parameters that in the final set of the PG learning process have high variances (i.e., Z_{sw}, X_{ds}, K_r) have been identified by PGPR as not relevant. This confirms the ability of PGPR algorithm to detect non-relevant parameters.
- Applying PGPR from iterations where non-relevant parameters have been detected and therefore fixed allows for a decrease in the number of experiments between 7% and 17% with respect to PG. The best fitness obtained using PGPR and PG are very similar (i.e., within $\pm 5\%$) thus indicating that PGPR guarantees the same performance of PG but using less experiments.

6.1.2. Experiments with different initial values

In the second set of experiments we repeat the approach described before, but running different learning processes starting from different initial parameter sets. We start with PG algorithm and when the relevance of a parameter is below the given threshold T_r, we split the learning task in two: (1) we continue with PG algorithm; (2) the irrelevant parameter is fixed and learning is pursued with the remaining parameters (we call this run PGPR-1). While running PGPR-1, when the relevance of a parameter is below

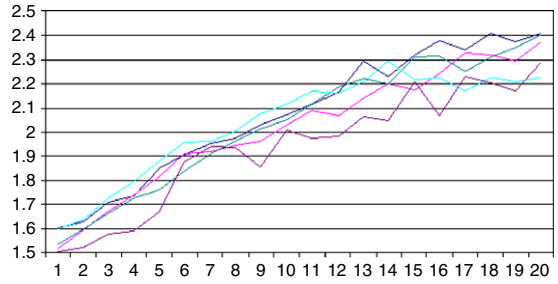

Fig. 8. Learning curves for the 5 learning sessions.

Table 1
Final parameter sets $[Z_{st}\ Y_{ds}\ Y_{ss}\ Z_{sw}\ X_{tot}\ X_{sw0}\ X_{ds}\ K_r]$ for the 5 learning sessions.

Run	X	$F(X)$
1	[0.158 0.059 0.053 0.001 0.168 0.289 0.239 0.026]	2.407
2	[0.159 0.060 0.053 0.008 0.163 0.310 0.250 0.018]	2.378
3	[0.161 0.059 0.055 0.006 0.165 0.287 0.243 0.022]	2.402
4	[0.156 0.062 0.050 0.003 0.154 0.294 0.243 0.030]	2.296
5	[0.153 0.056 0.051 0.006 0.170 0.267 0.203 0.028]	2.289
Average	[0.157 0.059 0.053 0.005 0.164 0.289 0.236 0.025]	2.354
Std.dev.	[0.003 0.002 0.002 0.003 0.006 0.015 0.019 0.005]	0.058
Std.dev. %	[1.8% 3.6% 3.3% 55.9% 3.8% 5.3% 7.9% 18.8%]	2.45 %

Table 2
Non-relevant parameters: values and iteration in which they have been fixed.

Run 1	$K_r = 0.024\,(10)$ $X_{ds} = 0.250\,(15)$ $X_{sw0} = 0.285\,(15)$ $Z_{sw} = 0.002\,(16)$ $Y_{ds} = 0.056\,(17)$	$N_{PGPR} = 266\,(-17\%)$
Run 2	$Z_{sw} = 0.005\,(14)$ $X_{ds} = 0.245\,(15)$ $K_r = 0.018\,(20)$ $Y_{ss} = 0.051\,(20)$	$N_{PGPR} = 298\,(-7\%)$
Run 3	$X_{ds} = 0.250\,(11)$ $Z_{sw} = 0.004\,(14)$ $K_r = 0.021\,(14)$ $Y_{ds} = 0.054\,(17)$	$N_{PGPR} = 272\,(-15\%)$
Run 4	$Z_{sw} = 0.005\,(13)$ $K_r = 0.027\,(14)$ $X_{ds} = 0.241\,(16)$ $Z_{st} = 0.159\,(20)$	$N_{PGPR} = 286\,(-11\%)$
Run 5	$Y_{ss} = 0.049\,(14)$ $X_{sw0} = 0.274\,(17)$ $K_r = 0.026\,(18)$	$N_{PGPR} = 298\,(-7\%)$

the threshold, we split again in two learning tasks: (1) we continue with PGPR-1 algorithm; (2) we fix this parameter and start a learning session with the remaining ones (we call this run PGPR-2), and so on, until a fixed number of iterations is reached.

In the tables, graphs and descriptions we use the terms PG, PGPR-1, PGPR-2, ... to refer respectively to the result of the standard PG algorithm, to the PGPR with 1 parameter fixed, to the PGPR with 2 parameters fixed, etc. Thus, the PG run corresponds to the standard PG algorithm, the PGPR-(n) run (with n being the maximum value reached during the procedure) corresponds to the PGPR algorithm proposed in this article (Fig. 5), while the runs PGPR-(i) ($1 \leq i \leq n-1$) represent intermediate processes where only a subset of the irrelevant parameters has been fixed.

This procedure aims at verifying the effectiveness of the choice of fixing non-relevant parameters at a certain iteration of the PG algorithm. It is important to notice here that after the learning process is split, the two sessions proceed in parallel with different randomized choices. Therefore, it is not possible to directly compare the results of the two sessions independently from these random choices.

Table 3
Results of simulation with different initial values.

Initial parameter set	F_0	PG	PGPR-1	PGPR-2
1	1.607	2.104	2.161 (14)	2.241 (18)
2	1.497	1.895	1.986 (13)	–
3	1.107	1.814	1.998 (12)	–
4	0.698	1.584	1.640 (10)	1.589 (17)

Starting from a manually coded initial parameter set ${}^0\underline{X}$ (with a fitness of 1.607 in the first iteration), 14 iterations of PG have been executed on all the parameters, reaching a fitness value of 1.865. Then one parameter has been fixed, because of its low relevance, and the learning process has been split in two: one execution with all the parameters, one execution with one parameter less (PGPR-1). At iteration 18, the values of the fitness were 1.954 for PG and 1.985 for PGPR-1, thus showing a slight increase of performance for PGPR. At this iteration a second parameter was below the relevance threshold and we split the execution again: we continued with PGPR-1 and we started a PGPR-2 session. At iteration 25 we stopped the learning process. The final fitness values for PG, PGPR-1, and PGPR-2 were respectively 1.935, 2.029, and 2.049, while the maximum values reached during all the executions were respectively 2.104, 2.161, and 2.241.

Three additional learning processes, starting from different initial policies, have been run in Webots. The maximum fitness values obtained by each execution of the algorithm in the four runs have been summarized in Table 3. The value F_0 in the table reports the fitness function computed at the first iteration, and indicates how good the initial parameter set was. As shown in the table, the initial sets used in the runs have been chosen in order to evaluate the effectiveness of the approach when starting both from good parameters (high values of F_0) and from bad ones (low values of F_0). The first irrelevant parameter was detected/fixed respectively at iteration 14, 13, 12 and 10 for runs 1 to 4 (values in parenthesis). A second irrelevant parameter was detected only in runs 1 and 4, respectively at iteration 18 and 17. Different final results in the table mostly depend on the initial parameter set and on the randomness of the algorithms.

As shown in the table, the general trend is that PGPR outperforms PG, and PGPR-($i+1$) outperforms PGPR-(i). However, there are also cases in which this is not confirmed (for example, with initial set 4). One reason is certainly the randomness of the algorithm. A second reason is that when starting from an initial set that is far from the optimum, fixing a parameter may be more risky, since it can lead to worse solutions.

Moreover, some parameters have been fixed early in the runs that start from lower quality initial policies. This is explained by the fact that when starting from a low fitness solution, some of the parameters contribute more significantly to increase the fitness (i.e., the derivative of the learning curve is higher), causing the other parameters to be fixed at earlier stages. As already mentioned, some of these choices may not be optimal (see again PGPR-2 for initial set 4).

The overall results of the experiments reported in this section are that (1) Policy Gradient methods are effective to improve performance of the walking gait described in this article, (2) the PGPR method generally guarantees better solutions with respect to PG, since in most cases where the relevance of a parameter is low, fixing this parameter value allows for speeding up the search toward the optimal solution.

Among all the tests performed, the best parameter set allowed for a top speed of above 12 cm/s, which is better than the (currently) standard walk in Webots (4.5 cm/s) and the one built-in in the URBI[6] universal robotics platform (10 cm/s).

[6] www.urbiforge.com.

6.2. Experiments with the real robot

The main objective of this work is to implement an effective walking gait on the real NAO robot, by exploiting the results obtained in simulation. Obviously, since the model of the NAO in the simulator is only approximated, the parameter sets that return best results in the simulation do not guarantee the same performance for the real robot. However, simulated tests still help to discard bad values of the parameters, to further limit the parameter ranges and to estimate the directions in which the fitness function tends to increase. Moreover, the implementation of the learning algorithms has been fundamental to improving important aspects of the robot gait, such as stability and precision with respect to the desired trajectory.

One important issue we have considered when moving to experiments on the real robots is the lack of symmetry in the response of the motors. While with the simulator it is possible to consider the same set of parameters for driving both the left and the right leg, with the real robot we had to duplicate some of the parameters in order to characterize the control of the left and the right leg. Moreover, we have noticed different behaviors of our four robots with the same parameters; this is again due to the different responses of the motors.

The experiments on the real robot have been performed as follows.

(1) The best solutions obtained in the simulations have been tested on the real robot and we have observed the different behavior of the robot with respect to the simulator.
(2) We have chosen a "good" initial set of parameters by visual inspection, evaluating mostly the walk stability.
(3) We have considered a rectilinear walk, and have adapted the computation of the fitness function to the real robot, on which we do not have a GPS-like device determining the real position of NAO. In practice, the fitness score is given by a combination of the performed distance and a stability value assigned by an operator that visually supervises the experiment.
(4) We then apply the PG algorithm by considering only the parameters that have shown to be more relevant during simulated experiments.

The results of two sessions of PG algorithm on two different real robots are reported here. In the first experiment (on robot 'Nao-34'), six iterations of the learning process have increased the fitness of the best policy from 45.75 to 57.00, while in the second experiment (on robot 'Nao-40') the increase of performance in 3 iterations has been from 55.0 to 64.5. Moreover, we have verified that the parameters learned for the rectilinear walk can be effectively used for curvilinear walk as well.
Details on the experiments and videos are available at: www.diag.uniroma1.it/~iocchi/RobotExperiments/HumanoidLearning

7. Conclusions

In this article, we have compared two learning methods for humanoid walking gaits based on Policy Gradient. In contrast with most works in the field of biped gait generation and biped gait learning, we have focused on curvilinear trajectories. We have shown that the PGPR algorithm, that takes into account parameter relevance, allows for better solutions than classic policy gradient, when only a few experiments are available, since it reduces the search space size during learning. The experimental results confirm the effectiveness of our biped motion control scheme, the performance of Policy Gradient reinforcement learning methods, and the higher convergence of the PGPR algorithm with respect to classic PG. In fact, we have applied this learning task to biped walking, obtaining, both in a simulated environment and on real robots, notable improvements in the execution of walking gaits, even after a limited number of experiments. These results are fundamental in robot learning applications, where time and hardware consumption is a major issue.

Future work will consist of using PGPR to identify the parameters characterizing pure rectilinear and pure rotational gaits. We also plan to estimate the dynamic model of the NAO robot and to exploit its sensing capabilities in order to develop a ZMP gait controller, whose parameters could also be tuned using PGPR.

References

[1] S.K. Chalup, C.L. Murch, M.J. Quinlan, Machine learning with AIBO robots in the four-legged league of RoboCup, IEEE Transactions on Systems, Man and Cybernetics, Part C (2006).
[2] T. Laue, T. Rofer, Getting upright: Migrating concepts and software from four-legged to humanoid soccer robots, in: Proc. of First Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, 2006.
[3] N. Kohl, P. Stone, Policy gradient reinforcement learning for fast quadrupedal locomotion, in: Proc. of IEEE International Conference on Robotics and Automation, 2004.
[4] A. Cherubini, F. Giannone, L. Iocchi, P.F. Palamara, An extended policy gradient algorithm for robot task learning, in: Proc. of IEEE/RSJ International Conference on Intelligent Robots and System, 2007.
[5] S. Kajita, B. Espiau, Legged robots, in: B. Siciliano, O. Khatib (Eds.), Handbook of Robotics, Springer, 2008, pp. 361–390 (Chapter 16).
[6] A.D. Kuo, Choosing your steps carefully, IEEE Robotics and Automation Magazine (June) (2007).
[7] J.-Y. Kim, I.-W. Park, J. Lee, M.-S. Kim, B.-K. Cho, J.-H. Oh, System design and dynamic walking of humanoid robot KHR-2, in: Proc. of IEEE International Conference on Robotics and Automation, 2005.
[8] J. Yamaguchi, E. Soga, S. Inoue, A. Takanishi, Development of a bipedal humanoid robot-control method of whole body cooperative dynamic biped walking, in: Proc. of IEEE International Conference on Robotics and Automation, 1999.
[9] M. Vukobratovic, B. Borovac, Zero-moment point — Thirty five years of its life, International Journal of Humanoid Robotics (2004).
[10] F. Yamasaki, K. Endo, H. Kitano, M. Asada, Acquisition of humanoid walking motion using genetic algorithm — considering characteristics of servo modules, in: Proc. of IEEE International Conference on Robotics and Automation, 2002.
[11] M. Hebbel, R. Kosse, W. Nisticò, Modeling and learning walking gaits of biped robots, in: Proc. of First Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, 2006.
[12] L. Hu, C. Zhou, B. Wu, T. Yang, P.K. Yue, Locomotion planning and implementation of humanoid robot Robo-Eectus Senior (RESr-1), in: Proc. of IEEE-RAS International Conference on Humanoid Robots, 2007.
[13] C. Niehaus, T. Rofer, T. Laue, Gait optimization on a humanoid robot using particle swarm optimization, in: Proc. of Second Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, 2007.
[14] M. Sato, Y. Nakamura, S. Ishii, Reinforcement learning for biped locomotion, in: Proc. of International Conference on Artificial Neural Networks, 2002.
[15] G. Arechavaleta, J.P. Laumond, H. Hicheur, A. Berthoz, The nonholonomic nature of human locomotion: A modeling study, in: Proc. of IEEE International Conference on Biomedical Robotics and Biomechatronics, 2006.
[16] G.B. Parker, D.W. Braun, I. Cyliax, Learning gaits for the stiquito, in: Proc. of International Conference on Advanced Robotics, 1997.
[17] M.S. Kim, W. Uther, Automatic gait optimisation for quadruped robots, in: Proc. of Australasian Conference on Robotics and Automation, 2003.
[18] S. Chernova, M. Veloso, An evolutionary approach to gait learning for four-legged robots, in: Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004.
[19] T. Rofer, Evolutionary gait-optimization using a fitness function based on proprioception, in: Proc. of 8th International Robocup Symposium, 2004.
[20] N. Kohl, P. Stone, Machine learning for fast quadrupedal locomotion, in: Proc. of 19th National Conference on Artificial Intelligence, 2004.
[21] U. Duffert, J. Hoffmann, Reliable and precise gait modeling for a quadruped robot, in: Proc. of 9th International Robocup Symposium, 2005.
[22] P. Fidelman, P. Stone, The chin pinch: A case study in skill learning on a legged robot, in: Proc. of 10th International Robocup Symposium, 2006.
[23] P. Stone, M. Veloso, Layered learning, in: Proc. of 11th European Conference on Machine Learning, 2000.
[24] J. Peters, S. Vijayakumar, S. Schaal, Reinforcement learning for humanoid robotics, in: Proc. of IEEE-RAS International Conference on Humanoid Robots, 2003.
[25] L. Yang, C.M. Chew, A.N. Poo, T. Zielinska, Adjustable bipedal gait generation using genetic algorithm optimized Fourier series formulation, in: Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006.
[26] F. Faber, S. Behnke, Stochastic optimization of bipedal walking using gyro feedback and phase resetting, in: Proc. of IEEE-RAS International Conference on Humanoid Robots, 2007.
[27] J. Morimoto, C.G. Atkeson, Learning biped locomotion, IEEE Robotics and Automation Magazine (June) (2007).

[28] T. Latzke, S. Behnke, M. Bennewitz, Imitative reinforcement learning for soccer playing robots, in: Proc. of 10th International Robocup Symposium, 2006.
[29] G. Sandini, G. Metta, D. Vernon, RobotCub: An open framework for research in embodied cognition, in: Proc. of IEEE-RAS International Conference on Humanoid Robots, 2004.
[30] F. Dalla Libera, T. Minato, I. Fasel, H. Ishiguro, E. Menegatti, E. Pagello, Teaching by touching: An intuitive method for development of humanoid robot motions, in: Proc. of IEEE-RAS International Conference on Humanoid Robots, 2007.
[31] A. Cherubini, F. Giannone, L. Iocchi, Layered learning for a soccer legged robot helped with a 3D simulator, in: Proc. of 11th International Robocup Symposium, 2007.
[32] A. Farinelli, G. Grisetti, L. Iocchi, Design and implementation of modular software for programming mobile robots, International Journal of Advanced Robotic Systems (March) (2006).

A. Cherubini received the M.Sc. degree (“Laurea”) in Mechanical Engineering in 2001 from the Università di Roma “La Sapienza”, the M.Sc. degree in Control Systems in 2003 from the University of Sheffield, U.K., and the Ph.D. degree in Systems Engineering in 2008 from the Università di Roma “La Sapienza”. During his PhD programme (2004-2007), he was a visiting scientist at the Lagadic group at IRISA/INRIA in Rennes (France), where he is currently working as post-doctoral fellow. His research interests include: visual servoing for mobile robotic applications, assistive robotics, legged locomotion, nonholonomic robot navigation, and robot learning.

F. Giannone received the B.Sc. degree in Computer Engineering from Università di Roma “La Sapienza” in December 2005. Her main interests are machine learning and behavior modeling through petri nets applied to Cognitive Robotics.

L. Iocchi received his master (Laurea) degree in 1995 and his Ph.D. in 1999 from Università di Roma “La Sapienza”. He is currently Assistant Professor at Department of Computer and System Science, Università di Roma “La Sapienza”, Italy. His main research interests are in the areas of cognitive robotics, action planning, multi-robot coordination, robot perception, robot learning, stereo vision, and vision based applications. He is author of more than 100 referred papers in international journals and conferences.

M. Lombardo received the B.Sc. degree in Electronic Engineering in 2005 from the Università di Roma “La Sapienza”, Italy. His interests are Artificial Intelligence, Legged locomotion, Robotics applications in medicine.

G. Oriolo received the Ph.D. degree in Control Systems Engineering in 1992 from the Università di Roma “La Sapienza”, Italy. He is currently an Associate Professor of Automatic Control and Robotics at the same university, where he also heads the Robotics Laboratory. He has been Associate Editor of the IEEE Transactions on Robotics and Automation from 2001 to 2005, and he is currently Editor of the IEEE Transactions on Robotics. His research interests are in the area of planning and control of robotic systems, in which he has published two books and more than 120 papers.