Journal of Intelligent & Robotic Systems (2021) 102: 54
https://doi.org/10.1007/s10846-021-01355-9

REGULAR PAPER

Learning Humanoid Robot Running Motions with Symmetry Incentive through Proximal Policy Optimization

Luckeciano C. Melo[1] · Dicksiano C. Melo[1] · Marcos R. O. A. Maximo[1]

Received: 16 October 2020 / Accepted: 24 February 2021 / Published online: 3 June 2021

Abstract
This article contributes with a methodology based on deep reinforcement learning to develop running skills in a humanoid robot with no prior knowledge. Specifically, the algorithm used for learning is the Proximal Policy Optimization (PPO). The chosen application domain is the RoboCup 3D Soccer Simulation (Soccer 3D), a competition where teams composed by 11 autonomous agents each compete in simulated soccer matches. In our approach, the state vector used as the neural network's input consists of raw sensor measurements or quantities which could be obtained through sensor fusion, while the actions are the joint positions, which are sent to joint controllers. Our running behavior outperforms the state-of-the-art in terms of sprint speed by approximately 50%. We present results regarding the training procedure and also evaluate the controllers in terms of speed, reliability, and human similarity. Since the running policies with top speed display asymmetric motions, we also investigate a technique to encourage symmetry in the sagittal plane. Finally, we discuss key factors that lead us to surpass previous results in the literature and share some ideas for future research.

Keywords Deep reinforcement learning · Robotics · Proximal Policy optimization

1 Introduction

RoboCup is an international academic competition created with an ambitious long-term objective of a team of humanoid robots beating the human soccer World Cup champions by 2050 [1]. To accelerate progress, it is composed of many leagues with different game rules and constraints on robot design.

RoboCup 3D Soccer Simulation (Soccer 3D) is a league of RoboCup Soccer based on a robot soccer simulator with high-fidelity rigid-body dynamics. Each team consists of 11 simulated Nao humanoid robots [2]. Soccer 3D contributes to RoboCup Soccer by providing an interesting research environment for high-level multi-agent cooperative decision making and humanoid robot control [3]. Since researchers are not constrained by real robot hardware, they typically rely on model-free optimization and machine learning techniques to develop high-performance motions or complex decision making algorithms. Simulation permits automatic execution of many well-controlled experiments, which is hard to carry out in a real robot setting [4, 5] due to the following issues [4, 5]:

- Battery autonomy is very limited with current technology, so batteries need to be recharged frequently.
- Robots often need to be manually reallocated to initialize an experiment.
- Hardware damage is probable due to the exploratory nature of these algorithms.
- Experience is limited by real-time execution while simulation may run much faster than real-time in powerful hardware.
- Experience is also limited by the number of physical robots available, whereas many simulations may run in parallel.

In Soccer 3D, motion control has a huge impact on the team's ability. One of the main causes is that humanoid robot control is one of the hardest problems in robotics.

The authors Luckeciano C. Melo and Dicksiano C. Melo contributed equally to this work.

✉ Luckeciano C. Melo
luckeciano@gmail.com

[1] Autonomous Computational Systems Lab (LAB-SCA), Computer Science Division, Aeronautics Institute of Technology, Praça Marechal Eduardo Gomes, 50, Vila das Acácias, 12228-900, São José dos Campos, SP, Brazil

Therefore, there are huge gaps regarding motion skills between teams. Moreover, high-level behaviors are strongly dependent on strong low-level skills: good strategy plans are of no use if the agents are not able to execute them robustly.

In fact, the underactuated, nonlinear, and high dimensional dynamics of a humanoid robot with many degrees of freedom, such as the Nao, poses great challenges to state-of-the-art control techniques [6]. Most works have tackled humanoid robot walking and researchers were successful in developing controllers based on the Zero Moment Point (ZMP) concept and reduced-order dynamic models, such as the linear inverted pendulum model [7]. However, these algorithms restrict the robot to operate under a small region of its dynamics to avoid breaking the assumptions of the simplified models [8, 9].

Recent advances in machine learning have allowed model-free deep reinforcement learning (DRL) algorithms to learn humanoid robot motions on data directly sampled from a high-fidelity simulator. Since there is no need to consider an explicit mathematical model, these techniques can better exploit the humanoid robot dynamics and learn faster locomotion skills. In Soccer 3D, teams have recently explored model-free DRL to develop running motions [10, 11], opening new research directions.

This article contributes by enhancing the methodology proposed by Abrel et al. [12] for learning a running policy in the Soccer 3D domain. To the best of our knowledge, the learned policy obtained in this work yields the fastest running motion in Soccer 3D, surpassing the previous state-of-the-art top speed reported in [12]. Furthermore, our approach also reduces training time in comparison to [12]. For a more comprehensive comparison between the methods, please refer to Section 3. Our methodology uses model-free DRL through the Proximal Policy Optimization (PPO) [13] algorithm, to learn the policy from no prior knowledge. The state vector used as input to the neural network encompasses raw sensor measurements from the Nao robot and quantities obtainable from state estimate techniques. Moreover, we also contribute by investigating a technique to encourage symmetry in the sagittal plane based on ideas by Abdolhosseini et al. [14], which was not handled in [12]. Finally, we evaluate how incentivizing symmetry impacts the running performance.

We highlight to the reader that this is an extended version of the conference paper [15]. We expanded Sections 2 and 3, and added many text enhancements throughout the paper. Moreover, Sections 4.5 and 5.5 are completely new as they are related to our attempt in solving a problem not addressed in our previous paper, namely the asymmetry observed in the resulting motions. We tackled this problem by encouraging symmetric motions through modifications to the dataset, as suggested in [14].

The remainder of this work is organized as follows. Section 2 provides theoretical background. Section 3 presents related work. In Section 4, we explain the methodology used in this work. Furthermore, Section 5 presents simulation results to validate our approach. Finally, Section 6 concludes and shares our ideas for future work.

2 Background

Reinforcement Learning (RL) is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them [16].

RL is a subfield of Machine Learning that studies how an agent interacts with an environment receiving rewards. However, of all the forms of machine learning, reinforcement learning is the closest to the kind of learning that humans and other animals do, and many of the core algorithms of reinforcement learning were originally inspired by biological learning systems [16].

In RL, a behavior is learned through the interaction with the environment. An agent tries to maximize some reward and, consequently, learns to maps which action should be taken given its own observations of the external world. Thereby, the agent does not have access to any data source or experience, it learns directly from its interactions with the environment.

2.1 Markov Decision Processes

We consider the problem of learning a running motion as a Markov Decision Process (MDP). A Markov Decision Process is a mathematical formalization of sequential decision-making, where actions influence immediate rewards and subsequent situations, or states, and, through those, future rewards [16]. A MDP, is a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, \gamma, T)$, where:

- $\mathcal{S}$ is a state space.
- $\mathcal{A}$ is an action space.
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a transition probability distribution.
- $r : \mathcal{S} \times \mathcal{A} \to [-r_{bound}, +r_{bound}]$ is a bounded reward function.
- $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ is an initial state distribution.
- $\gamma \in [0, 1]$ is a discount factor.
- T is the length of the finite horizon.

During policy optimization, we typically optimize a policy $\pi_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$, parameterized by $\boldsymbol{\theta}$, with the objective of maximizing the cumulative reward throughout the episode:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_\tau \Big[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) \Big], \tag{1}$$

where τ denotes the trajectory, $s_0 \sim \rho_0(s_0)$, $a_t \sim \pi_{\boldsymbol{\theta}}(a_t \mid s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1} \mid s_t, a_t)$.

2.2 Policy Gradients

In Policy Gradient (PG) methods, the objective is to learn a parameterized policy that directly selects an action from the state space. We address policy learning in continuous action spaces. During training, such methods compute an estimate of the policy gradient and use it in an optimization algorithm, usually a stochastic gradient ascent algorithm.

Considering the cumulative reward as our objective function, we can derive the following equation for estimating the gradient [13]:

$$\hat{g} = \mathbb{E}_t \big[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t) \hat{A}_t \big], \tag{2}$$

where $\hat{A}_t$ corresponds to an estimator of the advantage function. Therefore, in PG algorithms, we basically collect some roll-outs from the current policy, estimate an advantage function using the received rewards and then estimate a policy gradient w.r.t the parameters $\boldsymbol{\theta}$, and finally update such parameters.

The major advantage is that the method is completely model-free, i.e., the gradient itself does not depend on the dynamics that governs the environment. In some applications, it is possible to represent the environment perfectly. However, in many others, including this work, knowing a sufficient representation of the environment's dynamics would be too complicated and error-prone. Thereby, the choice for model-free algorithms.

Nevertheless, applying such gradient directly will not result in a good policy because the estimation is very noisy, resulting in catastrophic updates that slow learning [16]. This obstacle can be quite challenging in environments with long horizons or high-dimensional action spaces.

2.3 Advantage Function and the GAE Algorithm

To estimate the gradient using Eq. 2, we need to first compute an estimator of the advantage function $\hat{A}_t$. In practice, one may use the temporal-difference (TD) error to estimate the advantage function:

$$\delta^t_{\pi_{\boldsymbol{\theta}}} = r(s_t) + \gamma \hat{V}(s_{t+1}) - \hat{V}(s_t), \tag{3}$$

where $\hat{V}$ is an estimate of $V_{\pi_{\boldsymbol{\theta}}}$. This is possible because the temporal-difference error estimates the advantage function without bias when $\hat{V}(s) = V_{\pi_{\boldsymbol{\theta}}}(s)$:

$$\begin{aligned} \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[\delta^t_{\pi_{\boldsymbol{\theta}}} \mid s_t, a] &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}}[r(s_t) + \gamma V_{\pi_{\boldsymbol{\theta}}}(s_{t+1})] - V_{\pi_{\boldsymbol{\theta}}}(s_t) & (4)\\ &= Q_{\pi_{\boldsymbol{\theta}}}(s_t, a) - V_{\pi_{\boldsymbol{\theta}}}(s_t) & (5)\\ &= A_{\pi_{\boldsymbol{\theta}}}(s_t, a). & (6) \end{aligned}$$

However, there is no guarantee that the condition $\hat{V}(s) = V_{\pi_{\boldsymbol{\theta}}}(s)$ holds. Thus, this estimate method induces bias. To solve this problem, we may define the advantage estimator in terms of discounted temporal-difference errors of higher orders:

$$\begin{aligned} \hat{A}^{(1)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) &= \delta^t_{\pi_{\boldsymbol{\theta}}}, & (7)\\ \hat{A}^{(2)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) &= \delta^t_{\pi_{\boldsymbol{\theta}}} + \gamma \delta^{t+1}_{\pi_{\boldsymbol{\theta}}}, & (8)\\ \hat{A}^{(3)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) &= \delta^t_{\pi_{\boldsymbol{\theta}}} + \gamma \delta^{t+1}_{\pi_{\boldsymbol{\theta}}} + \gamma^2 \delta^{t+2}_{\pi_{\boldsymbol{\theta}}}, & (9)\\ \hat{A}^{(k)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) &= \delta^t_{\pi_{\boldsymbol{\theta}}} + \gamma \delta^{t+1}_{\pi_{\boldsymbol{\theta}}} + \gamma^2 \delta^{t+2}_{\pi_{\boldsymbol{\theta}}} + \cdots + \gamma^k \delta^{t+k}_{\pi_{\boldsymbol{\theta}}}, & (10) \end{aligned}$$

where the $\delta^{t+k}_{\pi_{\boldsymbol{\theta}}}$ is the TD error of k steps [16]. The Generalized Advantage Estimator (GAE) is defined as the exponentially-weighted average of these higher order errors [17]. We introduce the λ factor, which is related to how many steps will effectively be considered in the estimate. In mathematical terms:

$$\begin{aligned} \hat{A}^{GAE(\gamma,\lambda)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) &= (1-\lambda) \\ &\quad \times \Big(\hat{A}^{(1)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) + \lambda \hat{A}^{(2)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) + \lambda^2 \hat{A}^{(3)}_{\pi_{\boldsymbol{\theta}}}(s_t, a) + \cdots \Big) & (11)\\ &= \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta^{t+k}_{\pi_{\boldsymbol{\theta}}} & (12) \end{aligned}$$

GAE with $\lambda = 0$ is the case of Eq. 3 that can induce bias. On the other side, GAE with $\lambda = 1$ does not induce bias regardless of the accuracy of $\hat{V}(s)$, but has high variance. Therefore, GAE with $0 \le \lambda \le 1$ makes a compromise between bias and variance, controlled by λ [17]. The complete mathematical demonstration from Eqs. 11 to 12 can be found in [17].

2.4 Proximal Policy Optimization

Proximal Policy Optimization is a on-policy algorithm which tries to take the biggest possible improvement step on a policy using the current experience, without moving so far from the current policy. The problem with large improvements in policy is that it can cause a catastrophic loss of performance. PPO is a family of first-order methods that take into account a few considerations to keep new policies close to old. It alternates between sampling data through interaction with the environment, and optimizing a “surrogate” objective function using stochastic gradient ascent [13]. It takes some benefits of trust region optimization in terms of reliability and stability by defining a “clipped” surrogate objective:

$$\mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_t\Big[\min(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big], \quad (13)$$

where ϵ is a clip hyperparameter, and $r_t(\boldsymbol{\theta})$ is the probability ratio defined in Equation 14:

$$r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a_t \mid s_t)}{\pi_{\boldsymbol{\theta}_{old}}(a_t \mid s_t)}, \quad (14)$$

where $\boldsymbol{\theta}_{old}$ represents the parameter vector of the old policy. In this way, the clip function avoids excessively large policy updates and reduces the problem of catastrophic steps. We use an actor-critic style of PPO, where we also predict the value function and use it to estimate the advantage function through the Generalized Advantage Estimation (GAE) algorithm [18]. The actor and the critic roles are implemented through two different neural networks, respectively.

Finally, we use an implementation for PPO [19] that collects data from multiple parallel actors and synchronize them by applying an average of computed gradients into a unified policy representation (neural network).

3 Related Works

Research in RoboCup has a long history of applying optimization and machine learning for developing high-level behaviors or low-level skills [5, 9, 20–25]. These techniques require exploration of the state space in search of optimal solutions. Therefore, they are even more interesting in simulated leagues where simulation permits automatic execution of many well-controlled experiments [4, 5].

In Soccer 3D, a very successful approach for motion design uses metaheuristic optimization algorithms for tuning parameters used to describe a movement. The most common and straightforward way to describe a motion is through the so-called keyframes, which define a sequence of target robot poses (complete descriptions of joint angles) together with the time interval between these poses [9]. However, other motion descriptions were also explored.

Maximo et al. define joint trajectories through periodic functions whose parameters are optimized through Particle Swarm Optimization (PSO) [4]. Urieli et al. used the optimization of individual skills in Soccer 3D as a benchmark for optimization algorithms, such as Genetic Algorithms, Hill Climbing, Cross-Entropy Method, and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [26]. The authors concluded that CMA-ES showed the best performance in this domain. This and subsequent works by the same research group had a huge influence in the Soccer 3D community, as most teams devised to CMA-ES to optimize skills.

MacAlpine et al. optimized parameters of a model-based omnidirectional humanoid walking engine using CMA-ES [27]. The resulting gait is considered the main element responsible for the remarkable performance team UT Austin Villa had in RoboCup 2011, where it won all 24 games scoring a total of 136 goals without conceding none. A similar approach was used to optimize keyframes sampled from other teams in [24]. Then, the same ideas were used to learn 19 behaviors which were optimized to work well together using an overlapping layered learning paradigm [28]. Dorer used Genetic Algorithms to teach an Nao robot endowed with toes to kick the ball 30% better than the regular robot could [29]. Reinforcement learning (RL) has been an active area of research for many years [16]. RL is a branch of machine learning suited for sequential decision-making problems, such as those found in robot motion control. However, classical tabular RL methods, such as SARSA and Q-Learning, suffer from the curse of dimensionality due to the need of maintaining a value function table [16]. Therefore, they were only able to solve toy problems, such as grid worlds and multi-armed bandits [16].

Function approximators were used to mitigate this scalability problem [16], but researchers struggled with convergence problems when using neural networks in RL for many years. Major breakthroughs introduced by the Deep Q-Networks work, namely experience replay and target network, stabilized Q-Learning with deep neural networks as function approximators, giving rise to the field of deep reinforcement learning (DRL) [30].

Recent DRL algorithms for continuous control tasks are based on policy gradients [16], an alternative RL formulation that permit continuous state and action spaces. Some of the most famous methods in this class are Advantage Actor-Critic (A2C) [31], Sample Efficient Actor-Critic with Experience Replay (ACER) [32], Deep Deterministic Policy Gradients (DDPG), Trust Region Policy Optimization (TRPO) [33], and Proximal Policy Optimization (PPO) [13].

These algorithms yield remarkable performance in continuous control tasks. Heess et al. applied PPO to teach a humanoid model to run and execute parkour movements in a simulated environment [34]. Nevertheless, the learned motions look unnatural. Peng et al. [35] used the same learning algorithm but with modifications to the reward function and some strategies to improve the training procedure. This work encourages the learning to mimic reference motions, which results in more human-like behaviors. Since humanoid locomotion skills learned through DRL are often asymmetric, Peng et al. recently proposed strategies to incentive learning of asymmetric locomotion [14]. Experimental results demonstrate that state-of-the-art DRL algorithms greatly outperform metaheuristic optimization techniques, such as CMA-ES, in control tasks in terms of sample efficiency [13, 18].

Motivated by the success of DRL, the robot soccer community has been experimenting with these algorithms. MacAlpine and Stone optimized a kicking motion using the Trust Region Policy Optimization (TRPO) algorithm [25]. The authors verified that adding more parameters to the kick optimization improves the optimized kick performance. Nevertheless, they were unable to scale CMA-ES to more than a few hundred parameters, whereas DRL is often used to optimize policies with thousands or millions of parameters. Luckeciano extended this work by proposing a learning method where imitation learning is used to first capture a kick motion seed, which is then optimized through PPO [15, 36]. Many different RL approaches were tried and imitation learning proved essential to obtain high-performance kick motions in this setting. An extension to this method used meta-learning through the Bottom-Up Meta-Policy Search (BUMPS) algorithm to generalize specialized policies trained by RL for kicking to specific distances to a policy that precisely kicks to any desired distance within a range [20].

Regarding running motions in Soccer 3D, Fischer and Dorer [10] applied a modified version of the method presented in [29], based on Genetic Algorithms, to learn a running behavior from scratch using the Nao agent with toe joints. Furthermore, Abrel et al. achieved an incredible running motion by learning from scratch with PPO [12]. To the best of our knowledge, they have the state-of-the-art running skill in terms of top speed in Soccer 3D, which is reported as 2.5 m/s in their paper. They use a state space containing a global time step counter, the global torso's height and orientation, the joint positions, the acceleration and angular velocity measured by the inertial measurement unit (IMU), and feet force sensor readings. They also include numeric differentiation of some of these quantities. The action space consists of commanded joint positions. Finally, they use a simple reward signal that incentives the robot to run as fast as possible in the forward direction.

This work is an extended version of a conference paper [15], which enhances a previous work by Abrel et al. on the same subject [12]. We also use PPO with the same action space and optimization task. Nonetheless, we enhance their approach by expanding the state space (by adding the center of mass' coordinates at each time step), tuning PPO's hyperparameters for the task, and changing how the policy roll-outs are collected. Opposed to [12], our method does not rely on any modification to Simspark [37], the Soccer 3D simulation server. Our methodology generates a running motion which surpasses the state-of-the-art top sprint speed reported in [12] by approximately 50%, while reducing the training time at the same time. Furthermore, in comparison to this previous work [12], we also contribute by presenting alternative formulations which generate more natural running gaits.

Interestingly, we have been previously working on running policies with little success. However, we were bootstrapping our policies with our control-based omnidirectional walking engine [38] through imitation learning. Counter-intuitively, in opposition to the kicking motion, learning from scratch yields better running motions, which is an interesting insight we have drawn from [12].

4 Methodology

In this section, we provide details about our methodology: the formulation of running motion as an MDP; the description of the optimization tasks used to obtain our final policy and how we evaluate it; and the configuration regarding the PPO training.

4.1 Domain Description

The RoboCup 3D simulation environment is based on SimSpark [39], a generic physical multi-agent system simulator. SimSpark uses the Open Dynamics Engine (ODE) library for its realistic simulation of rigid body dynamics with collision detection and friction. The Nao robot has a height of approximately 57 cm and weights 4.5 kilograms.

The agent sends speed commands to the simulator and receives perceptual data. Each robot has 22 joints with sensors and actuators, and the monitoring/control of such joints happens at each cycle (20 ms). Visual information is obtained by the agent in periods of 60 ms through noisy measurements of the distance and angle to objects within a restricted vision cone of 120 degrees. The agent also receives noisy data from sensors: gyroscope, accelerometer, and feet pressure. Communication between agents and the server happens at the frequency of 50 Hz [40].

The coordinate system of the field is defined with origin in the center of the soccer field, with the x-axis pointing to the opponent's goal and the y-axis pointing to the left of someone oriented to the opponent's goal.

4.2 MDP Description

In this work, we aim to obtain a policy that provides running skills for a humanoid robot in the RoboCup 3D Soccer Simulation environment. This policy uses sensor data to infer a state for the robot and map it to possible joint positions at each time step, to make the robot run with maximum forward velocity, while being stable enough. To achieve such an objective, we need to model each component of the MDP.

4.2.1 State Space

We modeled the state space with the features reported in Table 1. We aim that, through optimization, the policy learns about the dynamics to perform running with enough stability by using such low-level information.

We start by creating counters to model temporal correlation. First, a general counter to explicit the sequential decision-making nature of the problem, which also improves velocity [12]. Secondly, there are counters, one for each foot, which explicit the motion period of each leg as helpful features to incentive symmetry.

The actual joint's values are also part of the observation since we would like to obtain a running motion with actions in the joint space. Torso's height, orientation, velocity, acceleration, and center of mass position aim to provide useful information regarding the robot's kinematics. The feet force data is related to the center of pressure, which indicates stability.

Finally, we noticed that only the sensor data from the current time step is not enough to model the whole dynamics of this problem. In fact, this is a partially observed setting. Therefore, we need to compute features that represent the recent past, such as the rate of change of each feature. We obtain them by applying first-order numeric differentiation.

4.2.2 Action Space

In terms of action space, we use the same indirect approach described by Abrel et al. [12]. We firstly bound the neural network's output to the interval $[-1, 1]$. We then linearly project this space onto the joint space, considering the range of possible values for each joint. Finally, we use these target angles θ_i^{target} and the current angles $\theta_i^{current}$ for each joint i to compute the commanded angular speed ω_i^t of each actuator, using a proportional controller with constant

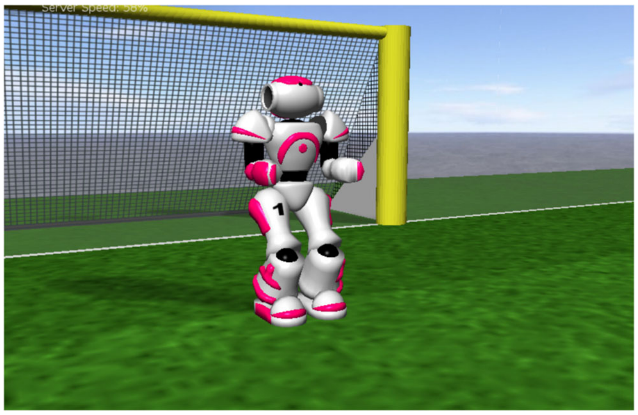

Fig. 1 Initial robot's joints configuration

$K_p = 7\ s^{-1}$, as presented in Eq. 15, where the quantities are given in the International System of Units (SI). We also saturate the speed of each joint using the limits provided in Simspark's documentation [37].

$$\omega_i^t = K_p \left(\theta_i^{target} - \theta_i^{current}\right). \tag{15}$$

4.2.3 Reward Function, Episode Horizon and Initial State Distribution

We consider different reward functions depending on the optimization task. We opted to shape it as simple as possible, to evaluate the efficiency of the RL optimization. In other words, we would like to validate if this formulation can obtain good locomotion skills from a high-level reward function (such as the speed in the global x-axis). Similarly, the episode horizon is also task-dependent. Both aspects will be described minutely in the next section.

In terms of initial state distribution, we considered the initial robot's joints configuration that enables the robot to start upright and easily explore bipedal balance and locomotion (see Fig. 1).

Table 1 State Space

Feature	Description	Size
General Counter	A counter that increments at each time step	1
Left/Right Foot Counter	A counter that restarts when the left/right foot touches the ground and increments at each time step	2
Joints' Values	Nao Joints, except neck yaw and pitch	20
Torso's Height and Orientation	The height (relative to ground) and yaw orientation of Nao's torso at the moment	2
Center of Mass	The coordinates of the center of mass at each time step	3
Torso's Velocity	Torso's angular velocity provided by the gyroscope sensor	3
Torso's Acceleration	Torso's acceleration provided by the accelerometer sensor	3
Left/Right Foot Pressure Data	The force and origin coordinates computed by left/right foot pressure feet sensors	12
Rate of Change	The rate of change w.r.t to the last time step of each feature previously described, except for the counters	43

4.3 Optimization Task and Evaluation

To achieve a sprint motion that runs as fast as possible, we create two similar optimization tasks. In both cases, we started the robot at $(-14, 0)$ using the initial joints' configuration previously described. The reward is just the forward distance traveled w.r.t. the last time step.

In Task I, the policy does not have any prior knowledge, thus we use Early Termination [35] by finishing the episode when the robot falls. This technique helps in two ways: first, it avoids to collect data from a bad terminal state, which the robot is not able to recover itself; second, we explicitly reinforce the agent to keep going forward as long as possible, obtaining more reward. Additionally, we also finish this task when the robot reaches the finish line placed at $x = 14$, which avoids that the agent crashes into the goal post.

Task II is very similar to the first one, but we consider a fixed episode length of 400 time steps instead of a finish line. In the first task, when the policy is able to achieve the finish line without fall, it starts trying to obtain reward by improving its forward velocity, but it also reduces the episode length and therefore the cumulative reward. A fixed horizon, on the other hand, will avoid this trade-off. We also maintained the Early Termination in case of agent fall.

We evaluate policies by two factors. First, we measure how fast the robot can run by computing its forward velocity. Secondly, how reliable and stable is such locomotion skill, by using the information about angle deviation from the global x-axis direction, once that this is the reference line which the agent should align with. The results reported in the next section will use such metrics during and after training.

4.4 Hyperparameters and Training Procedure

For training, we used a distributed version of PPO [19], with few modifications. The code is available in github[1]. This distributed variant parallelize agents as MPI processes. The learning updates happen synchronously as a reduce operation among all gradient estimations from each agent. We trained in a cluster of Intel Xeon scalable processors, using Intel DevCloud [41], with 19 worker nodes collecting data and a head node running the PPO algorithm.

Each experiment collected 200M time steps, during approximately 20 hours in the described setup. In terms of hyperparameters, we opted to use a set that is very similar to other benchmark environments for locomotion skills, presented in Table 2. However, we highlight the importance of hyperparameter search as future work, since PPO is very sensitive to them [36].

Table 2 PPO Hyperparameters

Hyperparameter	Value
Timesteps per actor batch	4096
Clip parameter	0.1
Entropy Coefficient	0.0
Optimization epochs	10
Learning rate	0.0001
Batch size	64
Discount factor	0.99
GAE λ	0.95
Learning rate decay	No decay

For neural network architecture, we used the same model for both actor and critic, composed of 2-layer dense networks with 64 neurons and hyperbolic tangent as non-linear activation. We initialize weights with a simple normal distribution with unit variance.

4.5 Symmetry

Healthy human gaits are usually symmetric [14]. Nevertheless, the methodology described so far produces high performance motions but with visible asymmetries. As demonstrated in [14], by encouraging symmetric motion, the learning may converge to more human-like motion policies. Therefore, we aim to induce motion symmetry and evaluate its final impact on the learning process.

In formal terms, two trajectories may be defined as symmetric if for each state-action tuple (s_t, a_t, r_t) from one trajectory, the corresponding symmetric state-action tuple is given by $(\psi_s(s_t), \psi_a(a_t), r_t)$ for the other trajectory. $\psi_s : \mathcal{S} \rightarrow \mathcal{S}$ is a bijective function which associates each state to its mirrored counterpart. Similarly, $\psi_a : \mathcal{A} \rightarrow \mathcal{A}$ associates each action to its mirrored counterpart.

A symmetric policy produces a mirrored action when given a mirrored state as input. Formally, we can define a symmetric policy to be one where, for all states $s \in \mathcal{S}$, the policy follows the property described in Eq. 16.

$$\pi(\psi_s(s)) = \psi_a(\pi(s)), \forall\, s \in \mathcal{S} \tag{16}$$

Abdolhosseini et al. [14] describes four methods for enforcing symmetry: using data augmentation, auxiliary losses, a time-indexed motion phase, or an architecture-based method. As an initial attempt in stimulating symmetry, we avoided the time-indexed motion phase method as it involves modifying the environment and the architecture-based method as it requires changing the neural network architecture. The other methods can be implemented

[1]https://github.com/alexandremuzio/baselines/tree/neural-engine-dynamics

directly in the learning algorithm, without changing the learning task or the neural network architecture.

Finally, we discarded the auxiliary loss method as it involves the addition of an extra hyperparameter. Our goal was to slightly encourage symmetry making no change in the optimization task, the neural network architecture, or the hyperparameters.

Thus, we opted for the data augmentation technique. In this approach, each trajectory tuple is duplicated, mirrored, then added as a valid experience tuple along with the original.

Formally, let $\tau = (s_1, a_1, r_1, \ldots, s_T)$ be a trajectory sampled from the environment. Next, we compute the mirrored trajectory of τ, i.e. $\tau_{sym} = (\psi_s(s_1), \psi_a(a_1), r_1, \ldots, \psi_s(s_T))$. Both τ and τ_{sym} are added to the roll-out memory buffer for learning. Notice that the rewards, $r_1, ..., r_{T-1}$ are the same in both τ and τ_{sym}. Algorithm 1 shows the modification to the backbone of the Proximal Policy Optimization algorithm [13] in order to achieve a running policy which is more symmetric. Differences from the standard PPO algorithm are marked in blue.

Algorithm 1 PPO Algorithm with Data Augmentation to Incentivize Symmetry.

```
while stopping criteria not met do
    foreach agent do
        Sample $\tau$: run policy $\pi_{\boldsymbol{\theta}_{old}}$ for $T$ timesteps
        Compute $\tau_{sym} = (\psi_s(s_1), \psi_a(a_1), r_1, ..., \psi_s(s_T), \psi_a(a_T), r_T)$
        $\tau \leftarrow \tau \cup \tau_{sym}$
        Estimate $\hat{A}_t$ using truncated GAE algorithm
    end for
    Apply $\nabla\mathcal{L}^{CLIP+S}(\boldsymbol{\theta})$ to policy network and $\nabla\mathcal{L}^{VF}(\boldsymbol{\phi})$ to value function network (parameterized by $\boldsymbol{\phi}$) using the Adam algorithm
    $\boldsymbol{\theta}_{old} \leftarrow \boldsymbol{\theta}$
end while
```

4.5.1 Symmetry in the Sagittal Plane

As mentioned in Section 1, the achieved policy lacks similarity with the natural human gait. The running motion does not present symmetry in the sagittal plane, as expected for a humanoid gait. Thus, we will investigate the effects of enforcing mirrored actions in the sagittal plane to achieve a more natural gait. In consequence, we have to define functions $\psi_s : \mathbb{R}^{89} \rightarrow \mathbb{R}^{89}$ and $\psi_a : \mathbb{R}^{20} \rightarrow \mathbb{R}^{20}$ to mirror our states and actions, respectively. From now, we will describe the transformation of each observation and action into its mirrored version.

– **Joint's values**

  As mentioned, the agent has 22 joints. The two neck joints (pitch and yaw) are not used as observations and actions since their impact on a running motion is negligible. The other 20 joints have to be mirrored

Table 3 Joint symmetry

Joint	Value After Applying Symmetry
Left Shoulder Pitch	Right Shoulder Pitch
Left Shoulder Yaw	**- Right Shoulder Yaw**
Left Arm Roll	**- Right Arm Roll**
Left Arm Yaw	**- Right Arm Yaw**
Right Shoulder Pitch	Left Shoulder Pitch
Right Shoulder Yaw	**- Left Shoulder Yaw**
Right Arm Roll	**- Left Arm Roll**
Right Arm Yaw	**- Left Arm Yaw**
Left Hip Yaw Pitch	**- Right Hip Yaw Pitch**
Left Hip Roll	**- Right Hip Roll**
Left Hip Pitch	Right Hip Pitch
Left Knee Pitch	Right Knee Pitch
Left Foot Pitch	Right Foot Pitch
Left Foot Roll	**- Right Foot Roll**
Right Hip Yaw Pitch	**- Left Hip Yaw Pitch**
Right Hip Roll	**- Left Hip Roll**
Right Hip Pitch	Left Hip Pitch
Right Knee Pitch	Left Knee Pitch
Right Foot Pitch	Left Foot Pitch
Right Foot Roll	**- Left Foot Roll**

Bold is used to indicate the joints where the symmetry operation also changes the signal

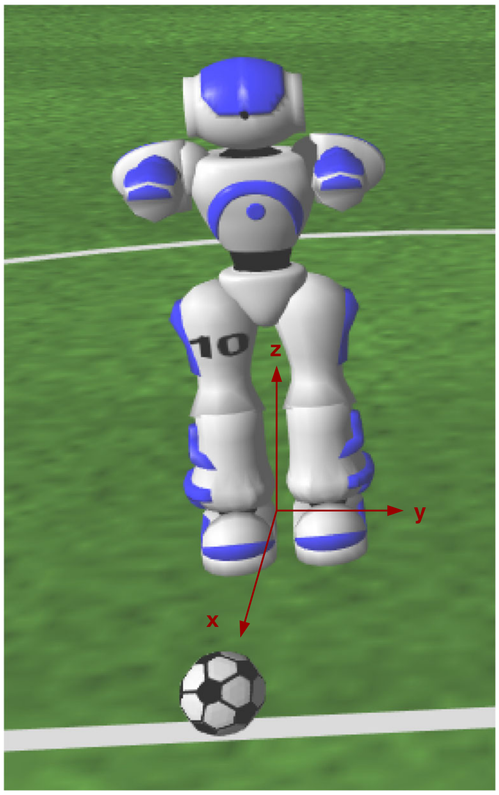

Fig. 2 Coordinate system representation

Table 4 Observation symmetry

Observation	Mirrored observation
Center of Mass Y	**- Center of Mass Y**
Torso Angular Velocity X	**- Torso Angular Velocity X**
Torso Angular Velocity Z	**- Torso Angular Velocity Z**
Torso Acceleration Y	**- Torso Acceleration Y**
Left Foot Pressure Data Origin X	Right Foot Pressure Data Origin X
Left Foot Pressure Data Origin Y	**- Right Foot Pressure Data Origin Y**
Left Foot Pressure Data Origin Z	Right Foot Pressure Data Origin Z
Right Foot Pressure Data Origin X	Left Foot Pressure Data Origin X
Right Foot Pressure Data Origin Y	**- Left Foot Pressure Data Origin Y**
Right Foot Pressure Data Origin Z	Left Foot Pressure Data Origin Z
Left Foot Pressure Data Force X	Right Foot Pressure Data Force X
Left Foot Pressure Data Force Y	**- Right Foot Pressure Data Force Y**
Left Foot Pressure Data Force Z	Right Foot Pressure Data Force Z
Right Foot Pressure Data Force X	Left Foot Pressure Data Force X
Right Foot Pressure Data Force Y	**- Left Foot Pressure Data Force Y**
Right Foot Pressure Data Force Z	Left Foot Pressure Data Force Z
Right Foot Counter	Left Foot Counter
Left Foot Counter	Right Foot Counter

Bold is used to indicate the observations where the symmetry operation also changes the signal

in the sagittal plane. Intuitively, the symmetry occurs by exchanging the right joints by their left side counterparts and vice versa. Besides that, the yaw and roll joints must have their values inverted. Table 3 describes the application of ψ_s and ψ_a for each joint. Notice that a minus signal is used to indicate that a value needs to be inverted. The coordinate system convention is shown in Fig. 2.

- **General Counter**
  As a simple counter, its value is the same for symmetric states.
- **Left/Right Foot Counter**
  The values for left and right are exchanged.
- **Torso's Height and Orientation**
  Those values are the same for symmetric states. Notice that only the torso's yaw orientation is used here.
- **Center of Mass**
  As the coordinates $\mathbf{CoM} = [x_{CoM}, y_{CoM}, z_{CoM}]^T$ are calculated relatively to the agent's local coordinate system, we invert the value of the y coordinate. Hence, $\psi(\mathbf{CoM}) = [x_{CoM}, -y_{CoM}, z_{CoM}]^T$.
- **Torso's Angular Velocity**
  The torso's angular velocity has a component in each axis, i.e., $\boldsymbol{\omega} = [\omega_x \ \omega_y \ \omega_z]^T$. As the sagittal plane is defined as the plane XZ (see Fig. 2), in order to mirror the torso's angular velocity, we invert the values of the x and z components, while the y component remains the same, i. e., $\psi(\boldsymbol{\omega}) = [-\omega_x \ \omega_y \ -\omega_z]^T$.
- **Torso's Acceleration**
  The torso's linear acceleration also has a component in each axis, i.e., $\boldsymbol{a} = [a_x \ a_y \ a_z]^T$. As it is a linear measure, we only invert the y component, i. e., $\psi(\boldsymbol{a}) = [a_x \ -a_y \ a_z]^T$.
- **Left/Right Foot Pressure Data**
  Again, we exchange the values between the left and right feet. We also invert the value of the y coordinate of the data computed by each pressure sensor.
- **Rate of Change**

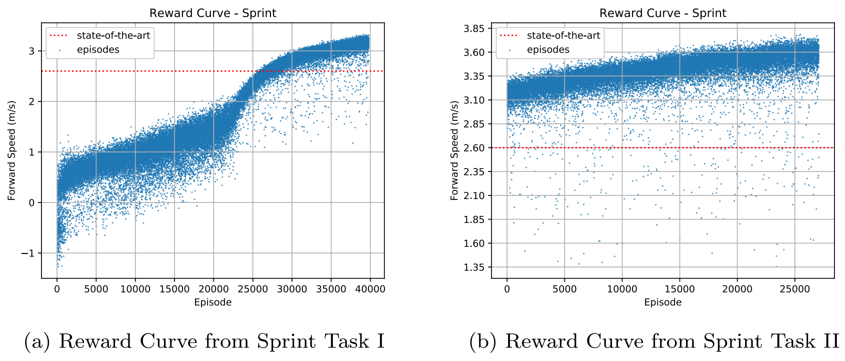

Fig. 3 Reward curves from sprint tasks

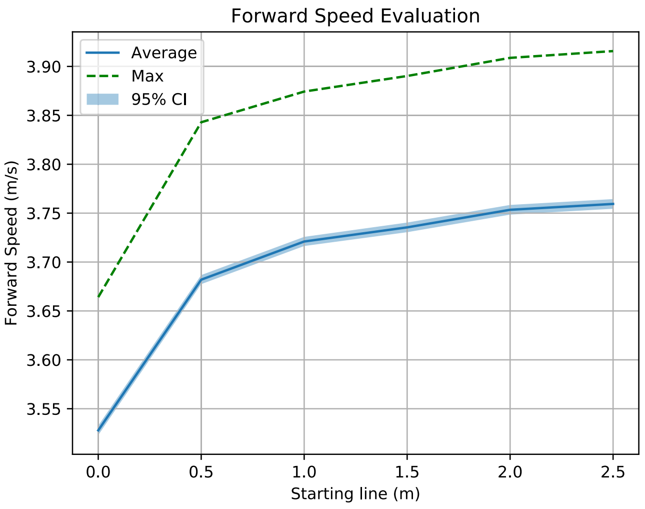

Fig. 4 Forward speed evaluation from a given starting line. All statistics were collected from 1,000 episodes

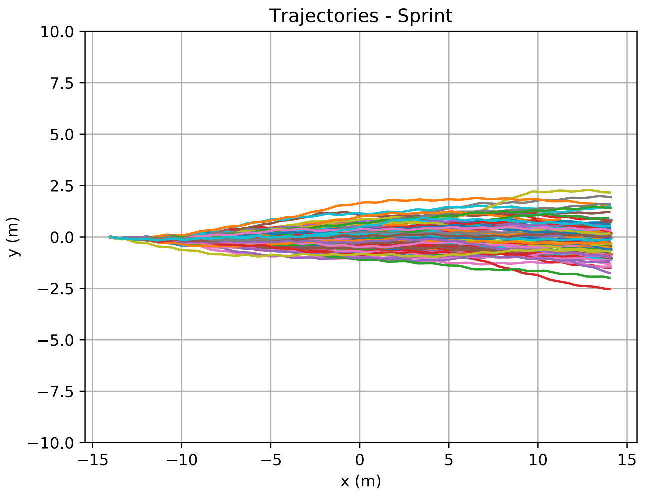

Fig. 5 Plot from the trajectories followed by the agent in 100 episodes

Each rate of change follows the same rule as its original observation.

Table 4 summarizes the mirrored representation of each observation. The joints are already represented in Table 3. The observations that remain constant in symmetric states are not represented for the sake of conciseness.

The code that applies symmetry to actions and observations is open source.[2]

4.5.2 Code Implementation

To learn the movement policy, we modified PPO's implementation from OpenAI Baselines [19]. With slight modifications, available in gitlab[3], we incremented the previous PPO's implementation with the data augmentation step, right after the trajectory sampling and before the optimization step.

As consequence, we produce experiments without changes in the MDP (neither the states nor the actions were changed) or in the optimization task. Hence, we could evaluate the individual impact of the data augmentation in the learning process, as the other major components were not altered. Also, the hyperparameters were the same described in Table 2, used for the previous experiments.

[2]https://gitlab.com/itandroids/open-projects/baselines/-/blob/run-policy-symmetry/baselines/ppo1/symmetry.py

[3]https://gitlab.com/itandroids/open-projects/baselines/-/tree/run-policy-symmetry

5 Results and Discussion

In this section, we present the results regarding our methodology during training and evaluation, in the light of the metrics previously described: how fast the robot can run (forward velocity) and how reliable and stable is such locomotion skill, as mentioned in Section 4.3.

In terms of reproducibility, we open source all the training logs (in Tensorboard [42] format), evaluation data, and scripts that computed the following results, as well as the trained models.[4] We also present some videos to illustrate the locomotion skill.

Although we are not able to release the whole agent code (due to competition reasons), we released the portion that corresponds to the training agent, which details the whole MDP implementation.[5]

5.1 Training Procedure

Figures 3a and b present the reward curves from both training procedures, each of them with 200M time steps. These data were collected using one training actor. We also highlight the state-of-the-art forward speed reported inside the Soccer 3D environment.

Using the training setup previously described, we use approximately 20.5 and 18.5 hours for training tasks I and II, respectively. The first task achieved the previous best speed between episodes 25,000 and 30,000, which

[4]https://drive.google.com/open?id=1wDEWSQv48qEM8Q17ydPQsrLM7sbtxtwz

[5]https://github.com/luckeciano/humanoid-run-ppo

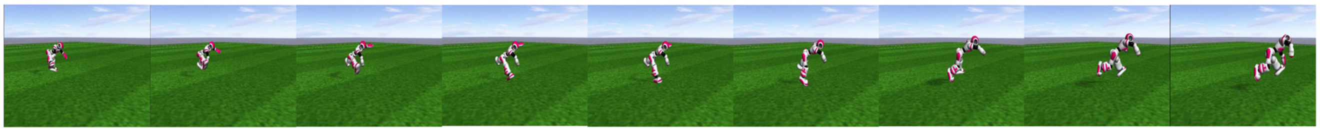

Fig. 6 Sequential frames illustrating the running motion from the best reported results, in terms of forward speed

corresponds to approximately 72M time steps. This shows improvement, in terms of sample efficiency, considering the results reported in [12] (i.e. we reduce the number of samples needed to achieve the same performance). We also observed that approximately 4 hours of training is enough for the agent to cross the whole soccer field.

5.2 Speed Evaluation

Figure 4 presents the data regarding speed evaluation. We collected them by reproducing the running motion during 1,000 episodes of sprint task I, using the deterministic policy after both training tasks. We present the average and maximum velocities across all episodes. Finally, we also show the 95% bootstrap confidence interval, symbolized by the blue shaded area.

Accordingly to Fig. 4, we report a top speed of 3.91 m/s, which surpasses the best velocity reported in the Soccer 3D environment by approximately 50.3%. The standard deviation for this top speed is 0.07 m/s. Furthermore, we observe a small confidence interval, which reinforces the reliability of the presented metric.

5.3 Reliability and robustness

We also present results about the reliability and robustness of the running motion. We evaluate them by plotting the followed trajectories and evaluating the final deviation.

Figure 5 shows the trajectories followed by the agent in 100 episodes. We preferred not to plot all 1,000 episodes for the sake of readability. Nevertheless, we report the mean of final deviation across all 1,000 collected episodes: 1.52 degrees (from the x-axis), with a standard deviation of 1.27 degrees. We did not employ any additional compensation regarding the agent's pose to reduce this deviation.

In the worst cases presented in the Fig. 5, there is a deviation of approximately 2.5 meters, that we do not conceive as harmful considering the length of the trajectory and the game conditions in the RoboCup 3D Soccer Simulation environment. Furthermore, we consider that such deviation can be reduced by applying compensation using the agent's pose as feedback.

5.4 Human Similarity

Finally, we need to present qualitatively ideas about how similar the running motion is in comparison to humans. As previously stated, we released videos about the motion.[6] We observed that, although the running motion has fast locomotion skills, the agent's torso is not completely erect, being less human-like.

We then reproduced all training procedures previously described but constraining the minimum robot torso's height to 0.33 m (in contrast to 0.27 m). We implemented this constraint as the condition that triggers Early Termination: if the torso height is less than this minimum value, the episode ends. It resulted in a more human-like motion, at the cost of some stability (the robot falls in more episodes) and forward velocity (top speed of 3.81 m/s). Figures 6 and 7 present both motions as sequences of frames. We also released all data and plots from both motions, to provide a further comparison between them.

5.5 Symmetry Experiments

The first experiments with data augmentation failed in the primary goal: the agent was not able to learn to run through the entire field. It learned to walk, but it fell after some steps. As can be observed in Fig. 8, the reward of the learning process with data augmentation does not reach the same level achieved by the experiment without symmetry incentive and with the same setup.

We believe that the problem is that by augmenting the data, we break the on-policy property assumed by the PPO algorithm to encourage symmetry. Abdolhosseini et al. [14] argued that one drawback of using this approach is that the mirrored tuples are not strictly on-policy, because at the training time the policy is not guaranteed to be symmetric at all. As consequence, the probability of sampling action$\psi_a(\pi(s))$ from $\pi(\psi_s(s))$ could be low, effectively corresponding to an off-policy action [14].

By comparing the reward evolution from the current experiment with the previous one, we observed that initially both experiments present similar progress, although they

[6]https://youtu.be/FLkVNh_I3UA

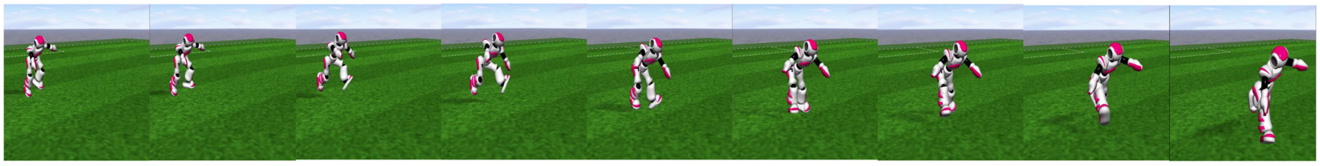
Fig. 7 Sequential frames illustrating the erect running motion, which is more similar to human locomotion

diverge over time, see Fig. 8. Inspired by that empirical observation, we choose a strategy to enforce symmetry without compromising the success of the learning process. We decided to split our training into two phases: a small pre-training phase and the proper training phase.

Those phases can be regular, i.e., without data augmentation or with data augmentation to encourage symmetry. A priori, we had no intuition on which would be the best strategy: first pre-train regular and then train with symmetry incentive or the contrary. Therefore, we run experiments with the two possibilities. We call the version with a pre-training phase without data augmentation as **PtRTS**: **Pre-t**raining **R**egular and **T**raining with **S**ymmetry. Moreover, the experiment with the pre-training phase without data augmentation is called **PtSTR**: **Pre-t**raining with **S**ymmetry and **T**raining **R**egular, as described in Table 5.

We ran PtRTS and PtSTR for the robot's torso height of 0.33 m and 0.27 m. The results were quite interesting: in both experiments, PtRTS failed as can be seen in Fig. 9, while the PtSTR could achieve the desired behavior of running through the soccer field. We also noticed an improvement in the gait of the agent, mostly in the experiment of height 0.27 m. The previous policy achieved with a torso's height of 0.33 m was more human-like, thus the gain with the data augmentation was not as impressive as in the case

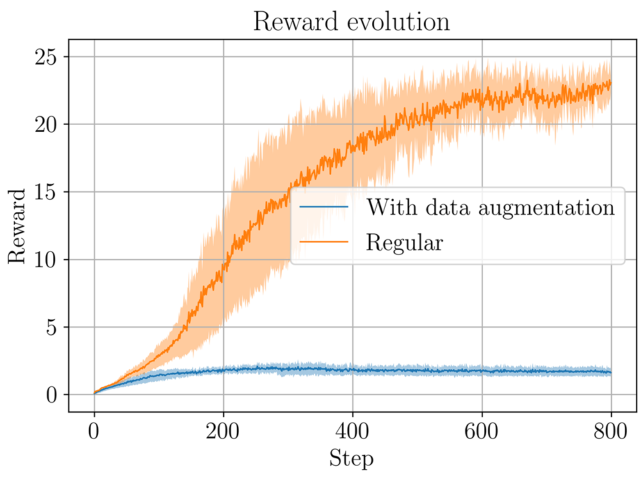

Fig. 8 In blue: reward evolution with data augmentation. In orange: reward evolution for the regular formulation, without data augmentation. Each curve shows the mean (darker color) and the 95% bootstrapped confidence interval (lighter color) from 4 training sessions repeated in the same conditions

of height 0.27 m. The results and comparisons from our symmetry experiments can be seen at the video.[7]

As can be observed, the best policy achieved without symmetry incentive produces a movement in which the right leg is put in front of the left leg and the running motion recalls a gallop, see Fig. 10. Meanwhile, the policy achieved with symmetry incentive present similar actions for the left and right, therefore the gait is more natural as can be observed in Fig. 11.

The small pre-training phase (45,000 environment timesteps) encouraging symmetry through data augmentation was capable of producing a much more natural gait movement which presents symmetry in the sagittal plane. This strategy was mandatory to the success of the experiment as the initial experiment was only able to produce a movement policy without symmetry and the initial symmetry experiment was not able to produce a stable running motion.

6 Conclusions and Future Work

In this work, we showed a methodology based on Deep Reinforcement Learning to learn running skills without prior knowledge. We used the Proximal Policy Optimization algorithm to learn a neural network policy whose inputs are related to the robot's dynamics. The results show that the learned motion is able to surpass the previous top forward speed in Soccer 3D by approximately 50.3%, considering the results reported in a previous work [12], assumed as state-of-the-art. Moreover, the running gait is learned in few hours.

We consider that this large margin from previous state-of-the-art is mostly due to our improvements in the observation space. We empirically found that providing the center of mass was very important to improve learning efficiency and asymptotic performance. This information helps to balance the agent during the exploration of a high performing policy. Additionally, the torso height and the imposed constraints are crucial to not exploit sub-optimal policies where the agent does not stand up in early stages of training.

We also share some insights we extracted from our experiments. Adding the torso's height and the center of mass to the state space are very important for speeding

[7] https://youtu.be/QPqrY1OK-No

Table 5 Symmetry experiments

Pre-training	Training	Name
Regular	With data augmentation	PtRTS
With data augmentation	Regular	PtSTR

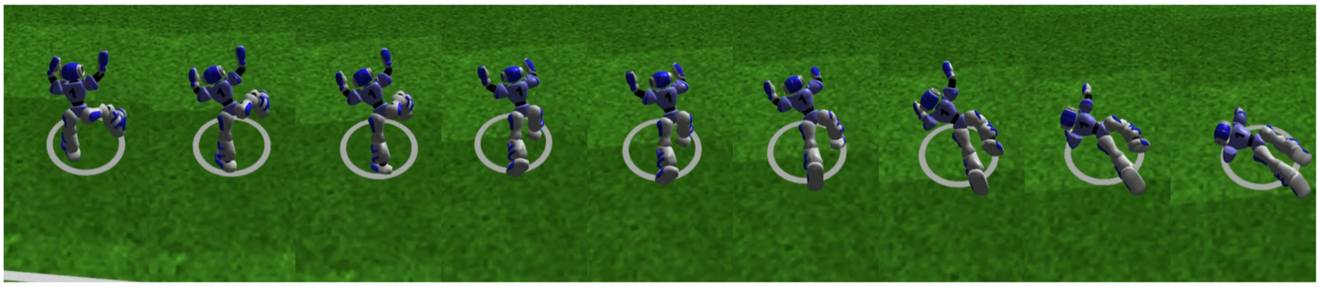
Fig. 9 Sequential frames illustrating the running motion from PtRTS. Notice that the agent is able to perform some steps, but it falls down

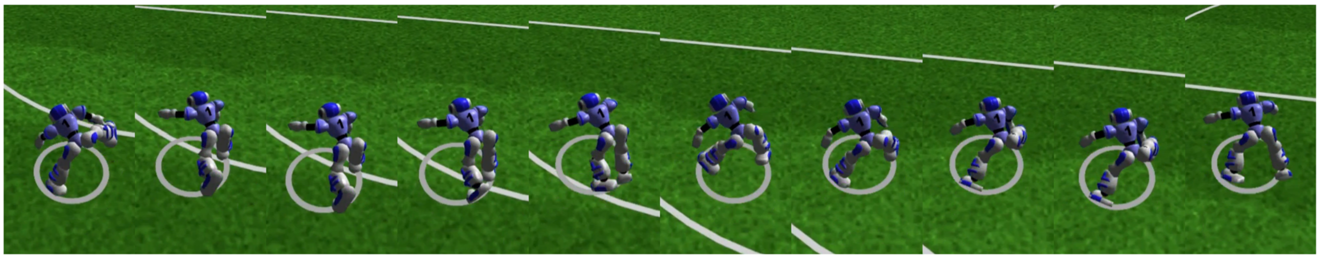
Fig. 10 Sequential frames illustrating the running motion from the best reported results. Notice that the right leg is always in front of the left leg

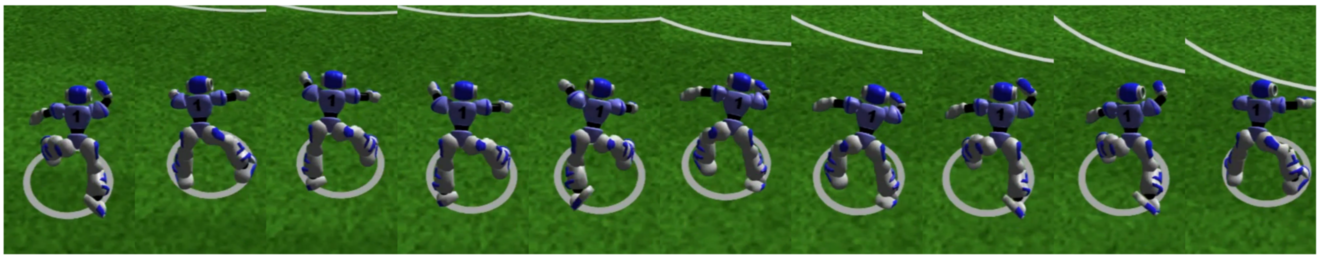
Fig. 11 Sequential frames illustrating the running motion from PtSTR, which is more similar to human locomotion. Notice that in the first and eighth frames the left leg is much more in front of the right leg. On the other hand, in the fifth frame the right leg is much more in the front of the left leg

up training and obtaining faster motions. Moreover, using many parallel actors in PPO improves gradient estimation, avoiding optimization steps in bad directions.

We also found that PPO is very sensitive to hyperparameters tuning. Distinct sets of hyperparameters result in very different policies. Furthermore, The resulting gait is also very sensitive to hyperparameters related to the agent, such as the minimum height of the torso and the proportional gain of the joints' controllers. Finally, we noticed that incentivizing symmetry yielded a more natural running gait, but with lower top speed.

We envision paths for future investigation. A curriculum learning approach may be applied to obtain high-level behaviors that emerges from this running policy, such as navigation and ball conduction skills. A possible path could involve starting with the trained running policy and changing the task objective to have the agent navigate to a given pose or conduct the ball. We also intend to integrate this running skill with the soccer agent behavior. This would require transition motions to start and stop running when needed, which could also be obtained through deep reinforcement learning.

We could learn other low-level skills using a similar methodology, such as kicking and getting up motions. In this regard, the reward signal need to be changed to account for the new task. We could also try other methods to encourage symmetric motions, such as another ones described by Abdolhosseini et al. [14].

Acknowledgements The authors thank ITAndroids' sponsors: Altium, Cenic, Intel, ITAEx, MathWorks, Metinjo, Micropress, Polimold, Rapid, SolidWorks, STMicroelectronics, Wildlife Studios, and Virtual.PYXIS. A special thanks goes to Intel for providing the computational resources and specialized AI software. Finally, we are also grateful to all members of the ITAndroids team, especially those from Soccer 3D, for developing the base code used in this research.

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

1. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for ai. AI Mag. **18**(1), 73 (1997). https://doi.org/10.1609/aimag.v18i1.1276, https://aaai.org/ojs/index.php/aimagazine/article/view/1276
2. Gouaillier, D., Hugel, V., Blazevic, P., Kilner, C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B.: Mechatronic design of nao humanoid. In: 2009 IEEE International conference on robotics and automation, pp. 769–774 (2009)
3. Melo, L.C., Maximo, M.R.O.A., da Cunha, A.M.: Learning humanoid robot motions through deep neural networks. In: Proceedings of the II brazilian humanoid robot workshop (BRAHUR) and II brazilian workshop on service robotics (BRASERO), pp. 74–79 (2019). https://fei.edu.br/brahurbrasero2019/Proceedings_BRAHUR_BRASERO_2019.pdf
4. Maximo, M.R.O.A., Colombini, E.L., Ribeiro, C.H.: Stable and fast model-free walk with arms movement for humanoid robots. Int. J. Adv. Robot. Syst. **14**(3), 1729881416675135 (2017). https://doi.org/10.1177/1729881416675135
5. Farchy, A., Barrett, S., MacAlpine, P., Stone, P.: Humanoid robots learning to walk faster: From the real world to simulation and back. In: Proc. of 12th Int. Conf. on autonomous agents and multiagent systems (AAMAS) (2013)
6. Kuindersma, S., Permenter, F., Tedrake, R.: An Efficiently Solvable Quadratic Program for Stabilizing Dynamic Locomotion. In: Proceedings of the 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE, Hong Kong, China (2014)
7. Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., Hirukawa, H.: The 3D Linear Inverted Pendulum Mode: A simple modeling for a biped walking pattern generation. In: Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, Hawaii, USA (2001)
8. Collins, S., Ruina, A., Tedrake, R., Wisse, M.: Efficient bipedal robots based on passive dynamic walkers. Science Magazine **307**, 1082–1085 (2005)
9. Muniz, F., Maximo, M.R.O.A., Ribeiro, C.H.C.: Keyframe movement optimization for simulated humanoid robot using a parallel optimization framework. In: 2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR), pp. 79–84 (2016)
10. Fischer, J., Dorer, K.: Learning a walk behavior utilizing toes from scratch. https://archive.robocup.info/Soccer/Simulation/3D/FCPs/RoboCup/2019/magmaOffenburg_SS3D_RC2019_FCP.pdf (2019)
11. Abreu, M., Simes, D., Lau, N., Reis, L.P.: Fast, human-like running and sprinting. https://archive.robocup.info/Soccer/Simulation/3D/FCPs/RoboCup/2019/FCPortugal_SS3D_RC2019_FCP.pdf (2019)
12. Abrel, M., Reis, L.P., Lau, N.: Learning to run faster in a humanoid robot soccer environment through reinforcement learning. In: Proceedings of the 2019 RoboCup symposium. RoboCup, Sydney, Australia (2019)
13. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR abs/1707.06347. arXiv:1707.06347 (2017)
14. Abdolhosseini, F., Ling, H.Y., Xie, Z., Peng, X., Panne, M.V.D.: On learning symmetric locomotion. Motion, Interaction and Games (2019)
15. Carvalho Melo, L., Omena Albuquerque Máximo, M.R.: Learning humanoid robot running skills through proximal policy optimization. In: 2019 Latin american robotics symposium (LARS), 2019 Brazilian symposium on robotics (SBR) and 2019 workshop on robotics in education (WRE), pp. 37–42 (2019)
16. Sutton, R.S., Barto, A.G. Reinforcement learning: An introduction, 2nd edn. The MIT Press, Cambridge (2018). http://incompleteideas.net/book/the-book-2nd.html
17. Schulman, J., Moritz, P., Levine, S., Jordan, M.I., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. CoRR abs/1506.02438. arXiv:1506.02438 (2015)
18. Schulman, J., Moritz, P., Levine, S., Jordan, M.I., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. In: Bengio, Y., LeCun, Y. (eds.) 4th International conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016). arXiv: 1506.02438
19. Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P.: Openai baselines. GitHub, San Francisco (2017). https://github.com/openai/baselines

20. Melo, L.C., Maximo, M.R.O.A., da Cunha, A.M.: Bottom-up meta-policy search. In: Proceedings of the deep reinforcement learning workshop of NeurIPS 2019 (2019)
21. Carvalho Melo, D., Quartucci Forster, C.H., Omena de Albuquerque Mximo, M.R.: Learning when to kick through deep neural networks. In: 2019 Latin american robotics symposium (LARS), 2019 Brazilian symposium on robotics (SBR) and 2019 workshop on robotics in education (WRE), pp. 43–48 (2019)
22. MacAlpine, P., Collins, N., Lopez-Mobilia, A., Stone, P.: Ut austin villa: Robocup 2012 3d simulation league champion. In: Chen, X., Stone, P., Sucar, L.E., van der Zant, T. (eds.) RoboCup 2012: Robot soccer world cup XVI, pp. 77–88. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
23. Abdolmaleki, A., Simões, D., Lau, N., Reis, L.P., Neumann, G.: Learning a humanoid kick with controlled distance. In: Behnke, S., Sheh, R., Sarıel, S., Lee, D.D. (eds.) RoboCup 2016: Robot world cup XX, pp. 45–57. Springer International Publishing, Cham (2017)
24. Depinet, M., MacAlpine, P., Stone, P.: Keyframe sampling, optimization, and behavior integration: Towards long-distance kicking in the robocup 3d simulation league. In: Bianchi, R.A.C., Akin, H.L., Ramamoorthy, S., Sugiura, K. (eds.) RoboCup-2014: Robot soccer world cup XVIII, Lecture Notes in Artificial Intelligence. Springer Verlag. Berlin (2015)
25. MacAlpine, P., Stone, P.: UT Austin Villa: RoboCup 2017 3D simulation league competition and technical challenges champions. In: Sammut, C., Obst, O., Tonidandel, F., Akyama, H. (eds.) RoboCup 2017: Robot soccer world cup XXI. Lecture Notes in Artificial Intelligence, Springer (2018)
26. Urieli, D., MacAlpine, P., Kalyanakrishnan, S., Bentor, Y., Stone, P.: On optimizing interdependent skills: A case study in simulated 3d humanoid robot soccer. In: Tumer, K., Yolum, P., Sonenberg, L., Stone, P. (eds.) Proc. of 10th Int. Conf. on autonomous agents and multiagent systems (AAMAS), vol. 2, pp. 769–776. IFAAMAS (2011)
27. MacAlpine, P., Barrett, S., Urieli, D., Vu, V., Stone, P.: Design and optimization of an omnidirectional humanoid walk: A winning approach at the RoboCup 2011 3D simulation competition. In: Proceedings of the twenty-sixth AAAI conference on artificial intelligence (AAAI) (2012)
28. MacAlpine, P., Stone, P.: Overlapping layered learning. Artif. Intell. **254**, 21–43 (2018). https://doi.org/10.1016/j.artint.2017.09.001. https://www.sciencedirect.com/science/article/pii/S0004370217301066
29. Dorer, K.: Learning to use toes in a humanoid robot. In: Akiyama, H., Obst, O., Sammut, C., Tonidandel, F. (eds.) RoboCup 2017: Robot world cup XXI, pp. 168–179. Springer International Publishing, Cham (2018)
30. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv: 1312.5602, Cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013 (2013)
31. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd international conference on international conference on machine learning - Volume 48, ICML'16, pp. 1928–1937. JMLR.org (2016)
32. Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N.: Sample efficient actor-critic with experience replay. arXiv:1611.01224 (2016)
33. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization (2017)
34. Heess, N., TB, D., S, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, S.M.A., Riedmiller, M., Silver, D.: Emergence of locomotion behaviours in rich environments. arXiv (2017)
35. Peng, X.B., Abbeel, P., Levine, S., van de Panne, M.: Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. ACM Trans. Graph. **37**(4), 1–14 (2018). https://doi.org/10.1145/3197517.3201311
36. Melo, L.C.: Imitation learning and meta-learning for optimizing humanoid robot motions. Master's Thesis, Instituto Tecnológico de Aeronáutica (2019)
37. Vatankhah, H., Lau, N., MacAlpine, P., van Dijk, S., Glaser, S.: Simspark. Gitlab, San Francisco (2018). https://gitlab.com/robocup-sim/SimSpark
38. Maximo, M.R.O.A., Ribeiro, C.H.C.: ZMP-based humanoid walking engine with arms movement and stabilization. In: Proceedings of the 2016 Congresso Brasileiro de Automática (CBA). SBA, Vitória, ES, Brazil (2016)
39. Xu, Y., Vatankhah, H.: Simspark: An open source robot simulator developed by the robocup community. In: Behnke, S., Veloso, M., Visser, A., Xiong, R. (eds.) RoboCup 2013: Robot world cup XVII, pp. 632–639. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
40. MacAlpine, P., Collins, N., Lopez-Mobilia, A., Stone, P.: UT Austin Villa: RoboCup 2012 3D simulation league champion. In: Chen, X., Stone, P., Sucar, L.E., der Zant, T.V. (eds.) RoboCup-2012: Robot soccer world cup XVI, Lecture notes in artificial intelligence. Springer Verlag, Berlin (2013)
41. Intel: Intel devcloud. https://software.intel.com/en-us/ai-academy/devcloud (2018)
42. et al, M.A.: TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, Software available from tensorflow.org (2015)

Luckeciano C. Melo received the BSc degree in Computer Engineering and the MSc degree in Electronic and Computer Engineering from Aeronautics Institute of Technology (ITA), Brazil, in 2018, and 2019, respectively. His research focus is to develop agents that learn behaviors through interaction, in an efficient, generalist, and adaptive way. Specifically, his primary focus of research and interest is to develop and apply meta-learning and reinforcement learning algorithms for high dimensional control tasks, aiming data efficiency and reusability of prior knowledge.

Dicksiano C. Melo received the BSc degree in Computer Engineering and the MSc degree in Electronic and Computer Engineering from Aeronautics Institute of Technology (ITA), Brazil, in 2019 and 2021, respectively. Besides humanoid robotics, his research interests also include Reinforcement Learning, Computer Vision, and Unsupervised Learning.

Marcos R. O. A. Maximo received the BSc degree in Computer Engineering (with Summa cum Laude honours) and the MSc and PhD degrees in Electronic and Computer Engineering from Aeronautics Institute of Technology (ITA), Brazil, in 2012, 2015 and 2017, respectively. Maximo is currently a Professor at ITA, where he is a member of the Autonomous Computational Systems Lab (LAB-SCA) and leads the robotics competition team ITAndroids. He is especially interested in humanoid robotics. His research interests also include mobile robotics, dynamical systems control, and artificial intelligence.