Journal of Intelligent & Robotic Systems (2022) 105: 12
https://doi.org/10.1007/s10846-022-01619-y

REGULAR PAPER

Deep Reinforcement Learning for Humanoid Robot Behaviors

Alexandre F. V. Muzio[1] · **Marcos R. O. A. Maximo**[1] · **Takashi Yoneyama**[2]

Received: 21 February 2022 / Accepted: 27 February 2022 / Published online: 27 April 2022

Abstract
RoboCup 3D Soccer Simulation is a robot soccer competition based on a high-fidelity simulator with autonomous humanoid agents, making it an interesting testbed for robotics and artificial intelligence. Due to the recent success of Deep Reinforcement Learning (DRL) in continuous control tasks, many teams have been using this technique to develop motions in Soccer 3D. This article focuses on learning humanoid robot behaviors: completing a racing track as fast as possible and dribbling against a single opponent. Our approach uses a hierarchical controller where a model-free policy learns to interact model-based walking algorithm. Then, we use DRL algorithms for an agent to learn how to perform these behaviors. Finally, the learned dribble policy was evaluated in the Soccer 3D environment. Simulated experiments show that the DRL agent wins against the hand-coded behavior used by the ITAndroids robotics team in 68.2% of dribble attempts.

Keywords Deep reinforcement learning · Robot soccer · Humanoid robots · Robotics

1 Introduction

RoboCup is an international academic competition created to foster robotics and artificial intelligence research [27]. It has an ambitious long-term goal of having a team of humanoid robots beating the human soccer World Cup champions by 2050. There are many leagues with different game rules and constraints on robot designs to accelerate progress towards this objective.

RoboCup 3D Soccer Simulation (Soccer 3D) is a league of RoboCup based on a robot soccer simulator with a high-fidelity simulation model of the Nao humanoid robot. The particular contributions to RoboCup reside in being a research environment for high-level multi-agent cooperative decision-making, and humanoid robot control [44]. A simulation environment is convenient for machine learning algorithms due to their need for large amounts of data [36]. Dealing with real robots is time-consuming due to the need to recharge batteries or reallocate robots manually to set up experiments. Moreover, experience collection may be largely accelerated by running many simulations in parallel and executing in faster than real-time. Unfortunately, transferring behaviors learned in simulation to real robots is challenging due to the so-called reality gap. Still, some works have succeeded in doing so, usually by executing a final fine-tuning process on the real robot [36].

Recent deep reinforcement learning (DRL) techniques have solved complex continuous robotic control tasks in simulation [24, 54]. Indeed, many DRL benchmarks involve legged robot tasks. In contrast to model-based methods, DRL does not require an explicit mathematical model of the robot. This is not only advantageous from an implementation perspective but also makes room for better motions since explicit models often include simplifying assumptions. In Soccer 3D, DRL has became popular for developing high-performance motions [3, 12, 32, 41, 43, 44, 62]. Nevertheless, there is still little research in applying these techniques for high-level behaviors.

This article contributes by presenting how to use DRL to develop humanoid robot behaviors. Instead of learning a policy that issues joint commands directly, we seek an approach where the agent learns to command a model-based walking engine not subject to learning.

We highlight to the reader that this article is an extension of the conference paper [50], where the Proximal Policy

✉ Marcos R. O. A. Maximo
mmaximo@ita.br

[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

[2] Electronic Engineering Division, Aeronautics Institute of Technology, Praça Marechal Eduardo Gomes, 50, Vila das Acácias, 12228-900, São José dos Campos, SP, Brazil

Optimization (PPO) algorithm is used to make a humanoid robot learn how to dribble a single opponent. To the best of our knowledge, our previous work was the first one to use DRL to learn a high-level behavior in Soccer 3D. In this article, we enhanced the presentation of many parts. Furthermore, we also present a completely new task in Section 5.1, which we call the "Humanoid Racing Task". Finally, we also refer the interested reader to the dissertation [49] for a more detailed presentation of some parts.

The remainder of this paper is organized as follows. Section 2 provides theoretical background. Section 3 presents related works. Section 4 describes the methodology used for agent development and evaluation. In Section 5, results for the racing and dribbling tasks are shown. Finally, Section 6 concludes and shares our ideas for future work.

2 Theoretical Background

This section presents background about reinforcement learning (RL) and deep reinforcement learning, focusing on the algorithms actually used in this work.

2.1 Reinforcement Learning

In reinforcement learning, an agent learns by interacting with the environment [64]. The interaction happens in a discrete manner: at each timestep t, the agent is in state S_t and executes action A_t; then, the environment transits to state S_{t+1} and emits a reward signal R_{t+1}.

The agent's objective is to maximize the cumulative reward, which considers future rewards while interacting with the environment. These interactions may be episodic, i.e., the task may restart after some number of timesteps, or continuing, i.e., the task continues indefinitely.

Mathematically, RL is based on the concept of a Markov Decision Processes (MDP). An instance of an MDP is defined by

- State set $\mathcal{S}$.
- Action set $\mathcal{A}$.
- Initial state distribution $P(S_1) : \mathcal{S} \to [0, 1]$.
- Transition probability function $\mathcal{P}(S_{t+1}|S_t, A_t) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$.
- Reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

The return G_t from state S_t is defined as

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{1}$$

where $\gamma \in [0, 1]$ is the so-called discount factor, which introduces the concept of discounting, so the agent prefers immediate rewards instead of long-term gains. A stochastic policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is given by

$$\pi(a|s) = \mathcal{P}[A_t = a|S_t = s]. \tag{2}$$

For a policy, the state-value function is defined as the expected return starting at a given state:

$$v_\pi(s) = \mathbb{E}[G_t|S_t = s]. \tag{3}$$

The action-value function is similar, but also takes into account the action taken:

$$q_\pi(s, a) = \mathbb{E}[G_t|S_t = s, A_t = a]. \tag{4}$$

Hence, the goal of an RL problem may be reframed as obtaining the optimal value function (or, equivalently, the optimal policy). Classic value-based RL algorithms, such as Q-Learning and Sarsa, estimate the value function using a table [64]. Nevertheless, these methods do not scale to large or continuous state or action spaces. This is known as the *curse of dimensionality* [8]. To overcome this limitation, function approximators must be used [64].

2.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) is based on the idea of using neural networks as function approximators. In this subsection, we will explain the DRL techniques used in this work.

Deep Deterministic Policy Gradients (DDPG) [30] uses two neural networks:

- Actor function $\mu(s|\theta^\mu)$: represents the current deterministic policy.
- Critic function $Q(s, a|\theta^Q)$: approximates the action-value function.

DDPG extends the Deterministic Policy Gradient Algorithm (DPG) [61], which is based on the Deterministic Policy Gradient Theorem. DDPG innovation lies in incorporating ideas from DQN [46] to stabilize learning with deep neural networks (DNNs), namely:

- **Experience replay:** instead of immediately using the agent's experience, store it in a replay buffer and, at each learning update, sample a minibatch of experiences from the buffer.
- **Target networks:** use copies of the actor and the critic to compute the targets. These networks are updated at a slower pace.

DDPG (as an off-policy algorithm) can explore independently from learning and uses an exploration policy given by

$$\mu'(S_t) = \mu(S_t|\theta^\mu) + \mathcal{N}, \tag{5}$$

where $\mathcal{N}$ is a noise process. A modern approach for the problem of action exploration is to add noise to the parameter space rather than on the action space [55], which may be implemented by inserting adaptive noise in the parameters of the neural network policy. This is used in this work.

Trust Region Policy Optimization (TRPO) frames RL an optimization problem [57]. The algorithm optimizes a surrogate, i.e. alternate, objective function constraining the policy update to lie within a trust region to guarantee monotonic improvement. TRPO also uses neural networks as function approximators.

TRPO constrains the policy step size through the Kullback-Leibler (KL) divergence. The KL divergence $D_{KL}[P, Q]$ is a measure of how one probability distribution diverges from a second one and is defined as the relative entropy between two continuous random variables P and Q.

TRPO solves the following optimization problem:

$$\underset{\theta}{\text{maximize}}\ \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t\right] \tag{6}$$

$$\text{subject to}\ \hat{\mathbb{E}}_t[D_{KL}[\pi_{\theta_{\text{old}}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta, \tag{7}$$

where $\hat{\mathbb{E}}_t[...]$ is the empirical average over a finite batch of samples, and $\hat{A}_t$ is an estimator for the advantage function $A(S_t, A_t) = Q(S_t, A_t) - V(S_t)$ at timestep t. We define the policy $\pi_\theta(a|s)$ by a normal distribution $\mathcal{N}$ where the mean and log standard deviation are outputs of a neural network [57].

TRPO limits the step size of the policy (using the KL divergence) to guarantee policy improvement. However, despite being more data-efficient and reliable than DDPG, TRPO is computationally costly and hard to implement [59].

PPO is similar to TRPO but uses a different surrogate objective function [59] in order to be simpler to implement. Let

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}. \tag{8}$$

Therefore, PPO's surrogate objective is

$$L(\theta) = \hat{E}_t\big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\big], \tag{9}$$

where

$$\text{clip}(x, a, b) = \begin{cases} a, & \text{if } x < a, \\ x, & \text{if } a \leq x \leq b, \\ b, & \text{if } x > b. \end{cases} \tag{10}$$

Moreover, $L(\theta)$ is the loss function and $\hat{A}_t$ is an estimator for the advantage function at timestep t. The clip function restricts policy updates to avoid catastrophic steps. We use an actor-critic implementation with two different neural networks, i.e., one network for the actor and another one for the critic. The advantage function is estimated through Generalized Advantage Estimation (GAE) [58]. PPO's objective can be further improved by adding an entropy bonus to incentivize exploration [45, 68]. Empirically, PPO outperforms TRPO in terms of sample complexity.

2.3 Curriculum Learning

In many tasks, defining a reward function that provides adequate guidance for the learning algorithm and at same time represents the intended behavior is hard [18]. For example, sparse rewards make the task really hard since the agent does not receive sufficient feedback for learning while doing random exploration [35]. To circumvent this problem, we adopt curriculum learning [9].

In curriculum learning, the agent starts with an easy task and then moves to a harder one when it has learned to accomplish the current task. To apply curriculum learning, the designer must order tasks by difficulty and define heuristics to decide when a task has been mastered [35]. An interesting result is shown in [69], where the authors apply curriculum learning to evaluate short computer programs. Furthermore, Bansal *et al.* show that self-play may provide a natural curriculum in an environment with multi-agent competition [6].

3 Related Works

Model-free methods are very popular for developing behaviors in RoboCup [11, 32, 34]. They are especially useful in simulated leagues where data collection is easier [36]. In the grand scheme of RoboCup, researchers expect that the knowledge gained in simulation will transfer to real robots when better *sim2real* methods are developed [16].

In RoboCup Soccer Simulation 2D, a seminal work demonstrates how an agent may learn to hassle an opponent player in a defensive situation using RL [20]. Moreover, Hausknecht and Stone used DDPG to develop an agent for the Half Field Offense (HFO) subtask of Soccer 2D [23]. In IEEE Very Small Size Soccer (VSSS), Medeiros *et al.* used PPO augmented with curriculum learning to make a soccer robot learn to intercept the ball against a nearby opponent [39]. They used an approach similar to ours, where the RL agent interacts with a model-based controller. Finally, in RoboCup Small Size, a league with physical robots endowed by omnidirectional movement and kicking devices, some works also present behaviors developed through DRL [26, 60]. Notice that developing behaviors in these domains is expected to be easier since the agents do not need to deal with the complex underlying dynamics of a humanoid robot.

In Soccer 3D, metaheuristic optimization algorithms have been extensively used for developing motions and

behaviors [2, 17, 34, 47]. Maximo *et al.* use Particle Swarm Optimization (PSO) to optimize parameters that define joint trajectories based on periodic functions [36]. Urieli *et al.* evaluated many optimization algorithms in the context of optimizing individual skills in Soccer 3D [65]. The authors reported CMA-ES as the algorithm with the best performance in this domain. Since then, CMA-ES has been used for many optimization tasks in Soccer 3D [13, 31, 32].

Reinforcement Learning (RL) has been an active area of research for over 30 years. Many classical RL algorithms are based on estimating the value function [7]. A very popular classical method is Q-Learning [67], which learns the action-value function to determine an optimal policy.

On the other hand, policy gradient methods directly search for an optimal policy without necessarily estimating the value function. REINFORCE is a seminal algorithm in this class of methods [68]. Policy gradient algorithms have been pursued in DRL, since this approach handles continuous state and action spaces.

Deep Reinforcement Learning (DRL) scaled RL to problems with much larger state and action spaces by using deep neural networks (DNNs) as function approximators. The seminal work Deep Q-Networks (DQN) [46] introduced novel ideas to stabilize learning when a deep neural network is used to approximate the value function.

DQN allows a continuous state space but still restricts the action space to be discrete. In this regard, as a policy gradient algorithm, DDPG also includes an explicit policy, allowing continuous action spaces. DPPG is based on the Deterministic Policy Gradient (DPG) theorem [61] and is capable of solving continuous control problems with high dimensional visual inputs [30]. It uses an actor-critic approach where DNNs estimate both the policy and the value function.

Another recent approach frames DRL as an optimization problem. Since policy gradient methods are very sensitive to the size of the policy update, which may result in so-called catastrophic drops in performance if too large, these methods restrict policy updates within a trust region. This family of methods is mainly represented by Trust Region Policy Optimization (TRPO) [57] and Proximal Policy Optimization (PPO) [59].

Recently, the Soccer 3D community is seeing a shift towards reinforcement learning methods. One of the first works in this regard used TRPO to optimize a kicking motion [34]. The authors argue that descriptions with more parameters typically lead to better motions, and while CMA-ES is able to optimize a few hundred parameters, DRL can optimize motions with thousand or millions of parameters encoded as weights of a neural network. Melo contributed in a similar fashion where imitation learning encodes an existing kicking motion in a neural network, which is enhanced through DRL [42]. Later, Melo *et al.* proposed a meta-learning algorithm to develop a policy to precisely kick the ball to any desired distance [43]. Dorer *et al.* also used PPO for multi-directional kick-learning [62]. In an approach combining model-based and model-free methods, Melo *et al.* used PPO to learn push recovery strategies on top of a model-based walking engine [41].

DRL has also been used for developing running motions in Soccer 3D. Abrel *et al.* achieved a high-performance running motion by learning from scratch with PPO [3]. Then, Melo and Maximo enhanced this work by adding the center of mass' coordinates to the state space, tuning PPO's hyperparameters for the task, and changing how the policy roll-outs are collected; with these modifications, the learned policy surpassed, by approximately 50%, the top speed previously achieved by [3]. Later, Melo *et al.* extended this method by using a technique to encourage symmetry in the sagittal plane, obtaining a more natural-looking movement [44].

Some works investigated how a humanoid robot may learn to dribble a ball [5, 28, 29, 31]. Leottau and Ruiz-del-solar propose a methodology where the behavior is split into two subproblems: alignment and ball pushing. Then, they use a fuzzy controller for alignment while ball pushing is achieved by an RL controller. In Soccer 3D, MacAlpine and Stone used layered learning and the CMA-ES algorithm to learn how to conduct the ball without directly taking opponents into account [28, 33]. This approach is one of the key ingredients of the success of team UT Austin Villa in the league.

To the best of our knowledge, our previous work was the first one to apply DRL to develop a high-level behavior in Soccer 3D, namely how to dribble against a single opponent [50]. In this article, we extend this work by expanding our presentation and showing how to use DRL in another task in this domain.

4 Methodology

This section explains the methodology used in this work, including the simulation environment SimSpark [1], the model-based omnidirectional walking engine [38], how the learning environment is implemented, and the hierarchical approach combining model-based and model-free techniques used to solve the tasks.

4.1 Simulation Environment

The chosen simulation environment is SimSpark [1], the simulator used in Soccer 3D. It is a multi-agent simulator and uses Open Dynamics Engine [52] as its physics engine. In the Soccer 3D environment of SimSpark, two teams of eleven humanoid agents compete in a soccer match.

Fig. 1 SimSpark Simulation Environment running

Robotic simulations in SimSpark are stochastic since its implementation does not guarantee determinism of events and noise is added to the robot's sensor measurements. Thus, different trials of the same simulation setting may generate different results.

The server also publishes information regarding the simulation's state for visualization purposes. This interface provides ground truth information about the simulation, e.g., the global positions of the robots, which is useful for machine learning tasks. Figure 1 shows a game simulation between two teams within RoboViz [63], a visualization tool for SimSpark.

Agents communicate with the simulator through TCP. The agents receive perceptual data from the server and send desired actions back to the simulator. Then, the server executes a simulation step of $\Delta t = 0.020$ s. To move, the robot sends the desired velocity values of each joint [51]. During official matches, the server does not wait for the agents' commands before simulating a new step, which imposes time restrictions for the agent's decision-making. Nevertheless, during training, we run the simulator in *sync mode*, therefore the server waits an response from every agent reply before executing a new step. On the other hand, *sync mode* allows faster than real-time simulation since the server may promptly execute a new simulation step after receiving commands from all connected agents.

The simulated agent is based on the Nao humanoid robot from SoftBank Robotics [56]. It has 22 joints, which receive velocity commands.

SimSpark was chosen mainly because it is the official simulator of the RoboCup Soccer Simulation 3D competition.

We also use the ITAndroids Soccer3D code base, which was developed in C++ by our research group to compute in the Soccer 3D competition. For more information about this code base, we refer the interested reader to [40, 48].

4.2 Omnidirectional Walking Engine

We adopt a hierarchical approach where the learning agent issues commands to a model-based omnidirectional walking engine. This walking algorithm was developed using control theory in previous works [37, 38] and was already present in ITAndroids code base. Despite not modifying the walking engine in this work, we provide a high-level description of the algorithm here since it influences how the robot walks, so it is part of the environment from the learning agent's perspective.

The walking engine is commanded with a desired velocity $\mathbf{v} = [u_x, u_y, u_\theta]^T$, where u_x, u_y and u_θ are desired velocities in the forward, lateral and rotational directions, respectively. The output is the joint angles trajectories so the robot walks at the desired velocity while maintaining balance. PID controllers at the joint level compute the joint velocities which are sent to the server. The algorithm is based on the Zero Moment Point (ZMP) stability criterion and uses the Linear Inverted Pendulum Model (LIPM) to approximate the robot dynamics [25]. A block diagram of the walking engine is shown in Fig. 2, where the following blocks are present:

- **Next Torso and Swing Foot Poses Selector:** based on the desired velocity, it plans the poses of the torso and the swing foot at the end of the step.
- **CoM Trajectory Generator:** computes the trajectory of the center of mass (CoM) so the robot achieves the planned torso and swing foot poses. The ZMP

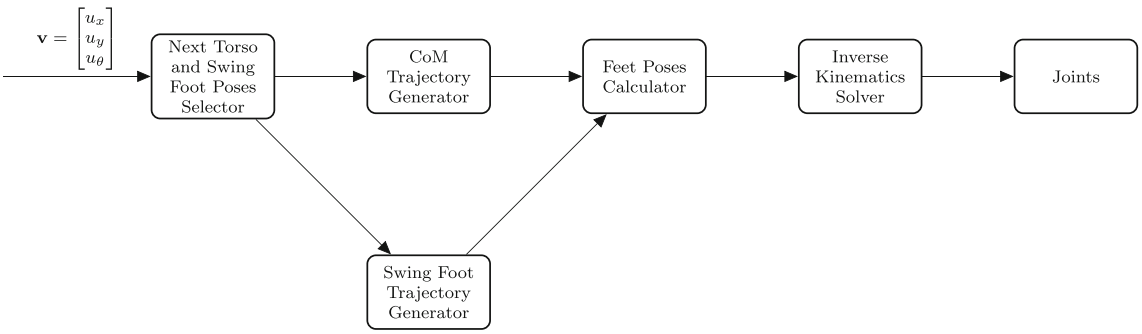

Fig. 2 Block diagram of the omnidirectional walking engine

is constrained to follow a polygonal trajectory within the support polygon during the step, while the CoM trajectory must match the CoM positions at the beginning and at the end of the step. Then, a boundary value problem is solved analytically to yield the trajectory of the CoM based on these constraints.
- **Swing Foot Trajectory Generator:** determines the swing foot's trajectory by interpolating between the initial and final poses of the swing foot.
- **Inverse Kinematics (IK) Solver:** computes the joint angles through an analytical IK method.

The walk cycle duration is constant, and the robot always alternates between the left and right feet as support foot. Moreover, the walking always undergoes a double support phase before changing the support foot. Furthermore, the acceleration is bounded since large velocity changes may destabilize the robot, so the robot may need multiple steps to reach the desired velocity. The parameters of this walking engine were optimized using CMA-ES [47]. For details, we refer the interested reader to [37, 38].

4.3 Environment and Implementation

For the implementations of the RL algorithms DDPG, TRPO, and PPO, we use the OpenAI Baselines repository [14], which is a set of high-quality implementations of reinforcement learning algorithms made by OpenAI.

Two modules compose the project, each one running as a separate process:

- **Learning Client:** runs the RL algorithms and makes remote procedure calls (RPCs) to the server exchanging state and action information regarding the soccer agent. This module was implemented in Python 3.5 and TensorFlow through the OpenAI Baselines [14] framework.
- **Soccer Agent:** acts as a bridge between SimSpark and the learning client. This module was implemented in C++ using the ITAndroids' code base.

The communication between server and client is based on Protocol Buffers [21] and uses RPCs to exchange information. Figure 3 shows a pictorial overview.

Therefore, another contribution of this work is the implementation of a framework for DRL in the context of the RoboCup 3D Soccer Simulation League. Other teams from the league may benefit from using the code to learn their own behaviors. The client's code is available as open-source at:

https://github.com/alexandremuzio/rlearning3d

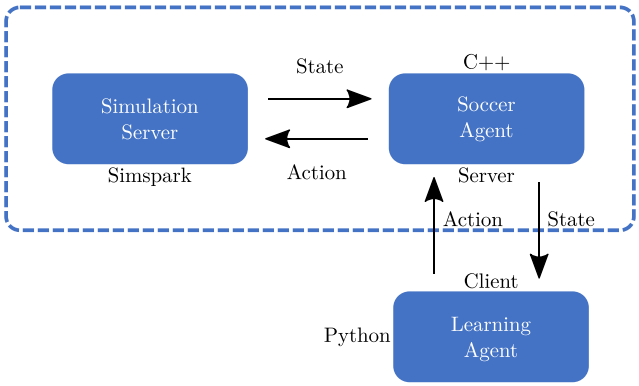

Fig. 3 Learning architecture diagram

4.4 Approach and Task Modeling

We use a hybrid hierarchical controller that combines a learned policy and a model-based algorithm (walking engine):

- **High-Level Controller:** the policy learned through DRL. It runs at a sampling rate of 10 Hz.
- **Lower Level Controller:** model-based walking controller that runs at a higher sampling rate of 50 Hz. This algorithm does not contain learning parameters.

Figure 4 presents this architecture.

The main task we intend to learn is soccer dribbling. We also learned a much easier task regarding completing a racing track. Learning this additional task acted as a warm-up, allowing us to get a better grasp on how to model tasks in this environment and how to provide good reward shaping.

A vector $[x, y, z, \theta]^T$ describes an agent's pose, as presented in Fig. 5. For learning, ground truth information is used, which is retrieved from the server using an entity called the "Wizard" in ITAndroids' base code. Velocities are obtained by deriving the positions numerically. Since numeric differentiation may introduce too much noise, we use low-pass filters to smooth out the signals.

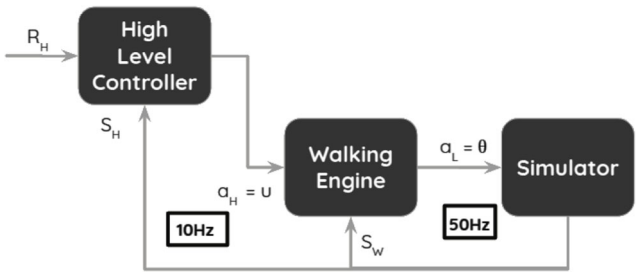

Fig. 4 Hierarchical hybrid controller

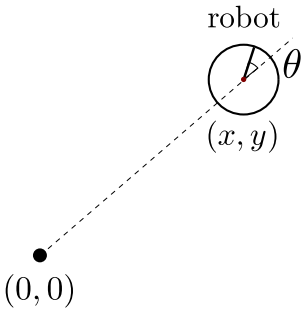

Fig. 5 Robot pose representation. It is composed of 3 variables: the robot's x and y coordinates and its orientation θ

The hardware used for all experiments consisted of a notebook with an Intel i7-7500U CPU and a GeForce 940MX GPU. Notice that SimSpark's physical simulation is the bottleneck in terms of computational cost in our case; therefore, the training time is mainly CPU-bound.

5 Experiments and Results

In this section, we describe how we modeled each task in a reinforcement learning framework. We also present and discuss the obtained results.

When using RL (and machine learning in general), it is important to use metrics to monitor the training performance. In this work, we mainly observed episode duration, accumulated reward, and each algorithm's data efficiency.

5.1 Humanoid Racing Task

In this task, the robot must reach the finish line of a race track of 18x4 m while remaining between the two lateral lines. Despite looking trivial at first glance, recall that to accomplish this task with high performance, the policy needs to take the walking engine close to its limits without falling over. ITAndroids' agent relies on many hand-coded heuristics to navigate at high walking speeds. Also, notice that the robot rarely finishes the race track when the walking engine is used in an open-loop fashion, as will be discussed later.

Figure 6 presents the task visually. The robot starts the race in the center of the green line and needs to reach the red finish line while staying within the region defined by the black border lines.

State space: consists of the x, y, z coordinates of the torso, the yaw angle of the torso θ, and the forward velocity v_x, sideways velocity v_y and rotational velocity v_θ. Notice that these are actual velocities, not the commanded ones.

Action space: consists of u_x, u_y and u_θ, which are velocity control signals in forward, sideways and rotational directions, respectively.

Reward signal: the reward is defined by

$$r(s,a) = v_x - 0.005(v_\theta^2 + v_y^2) - 0.05(u_x^2 + u_y^2 + u_\theta^2) - 0.05y^2 + 50\mathbb{I}^{\text{finish line}} - 10\mathbb{I}^{\text{leave track}} - 10\mathbb{I}^{\text{robot fell}}, \quad (11)$$

where $\mathbb{I}^{\text{finish line}} = 1$ if the robot arrives the finish line and 0 otherwise; and $\mathbb{I}^{\text{leave track}} = 1$ if the robot leaves the track and 0 otherwise.

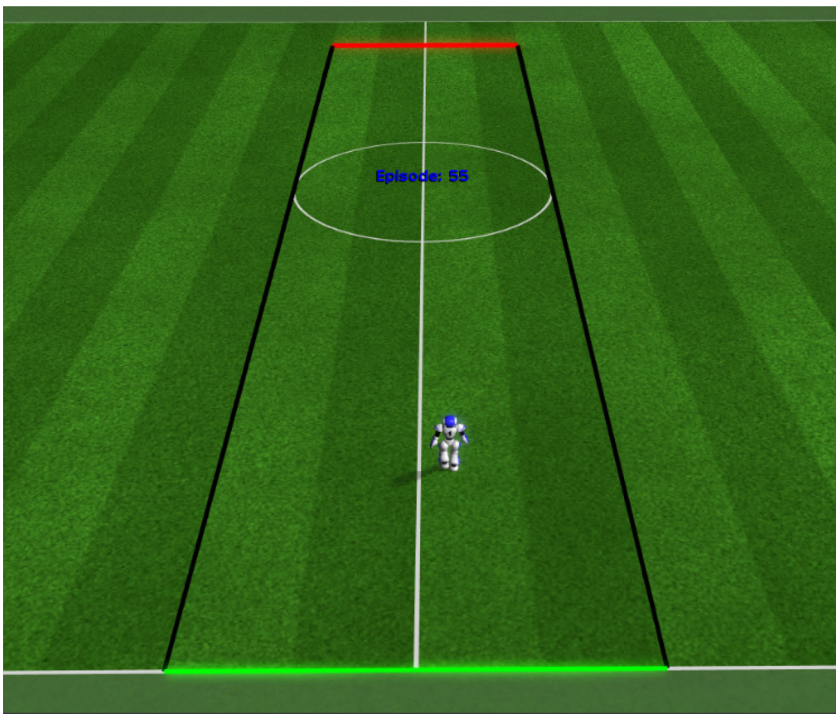

Fig. 6 Humanoid Racing Domain

Table 1 Experiments parameters

Hyperparameter	Value
Horizon (T)	2000
Discount (γ)	0.99

Table 3 TRPO hyperparameters used for the two tasks

Hyperparameter	Value
Stepsize (D_{KL})	0.01
GAE parameter (λ)	0.98
Timesteps per batch	1024

The term v_x rewards the robot for moving in the forward direction of the race track, while the terms $-0.005(v_\theta^2 + v_y^2)$ and $-0.05y^2$ penalize deviation from this direction. We took inspiration from [15] for these terms.

The episode duration is 2,000 steps, and the episode terminates if the robot falls or if the robot leaves the race track. We consider the agent to have fallen if its CoM gets below 0.2 m.

In this domain, we ran DDPG, TRPO, and PPO during 10^6 timesteps. Each experiment needed around 3 hours to complete.

Table 1 shows parameters used to configure the experiments. Moreover, Tables 2, 3, and 4 present hyperparameters for DDPG, TRPO, and PPO, respectively. These values were mainly based on the hyperparameters from [15] and [59].

To obtain more statistically significant results, we ran the experiments 5 times for each algorithm and use the mean and standard deviation over all runs. Figure 7 shows the episode rewards during the training phase for all runs. The dark line and the transparent filling represent the mean and the standard deviation, respectively, of reward considering the 5 executions. Likewise, Fig. 8 presents the mean and the standard deviation of the episode length over the 5 runs, respectively.

By looking to Figs. 7 and 8, we conclude that the performance of PPO is consistently higher than DDPG and TRPO in this task. However, we need to clarify that this may be due to a lack of hyperparameter tuning for DDPG and TRPO.

For TRPO and PPO, the episode length in the first 200,000 timesteps is shorter than at the end of training. Intuitively, we may say that the robot is initially falling down or leaving the race track until it learns to avoid falls and to stay on track, so the episodes get longer over time. Also, notice that the DDPG agent barely learns.

During evaluation, we noticed that the only agent capable of consistently reaching the finish line was the one trained with PPO. Therefore, we chose one of the policies trained with PPO and compared it against a baseline consisting of an open-loop "go-straight" controller (i.e., we command only forward velocity to the omnidirectional walking pattern generator). We generated 20 trajectories for each controller and compared them graphically, as shown in Fig. 9.

Finally, we summarize the results obtained for these trajectories:

- The **total race average speed** is $\overline{v_x} \approx 0.6$ m/s. This expresses how fast the agent completed the race on average. Note that the walking engine limits the forward speed command to 0.9 m/s due to stability concerns. Therefore, when compared to both the baseline and the theoretical fastest agent, the learned policy has a good performance on this task.
- The **horizontal displacement average** is $y_{\text{mean}} \approx 0.47$ m. This measures, on average, how much the agent has deviated from the centerline of the race track when it reaches the finish line. When we take into account that the race track has a width of 4 m, the average horizontal displacement looks small, as presented in Fig. 9. Furthermore, the robot remained within the track limits in all 20 executions.

Therefore, the agent trained with PPO accomplishes the task with a performance far superior than the baseline, since the learned policy compensates deviations that occur during execution. Nevertheless, Fig. 9 shows that there is a significant bias in the lateral direction, so there is still room

Table 2 DDPG hyperparameters used for the two tasks

Hyperparameter	Value
Batchsize	64
Actor learning rate	10^{-4}
Critic learning rate	10^{-3}
Adaptive param. noise	0.2

Table 4 PPO hyperparameters used for all two tasks

Hyperparameter	Value
Adam stepsize	3×10^6
Num. epochs	10
Minibatch size	64
GAE parameter (λ)	0.95
Timesteps per batch	2048

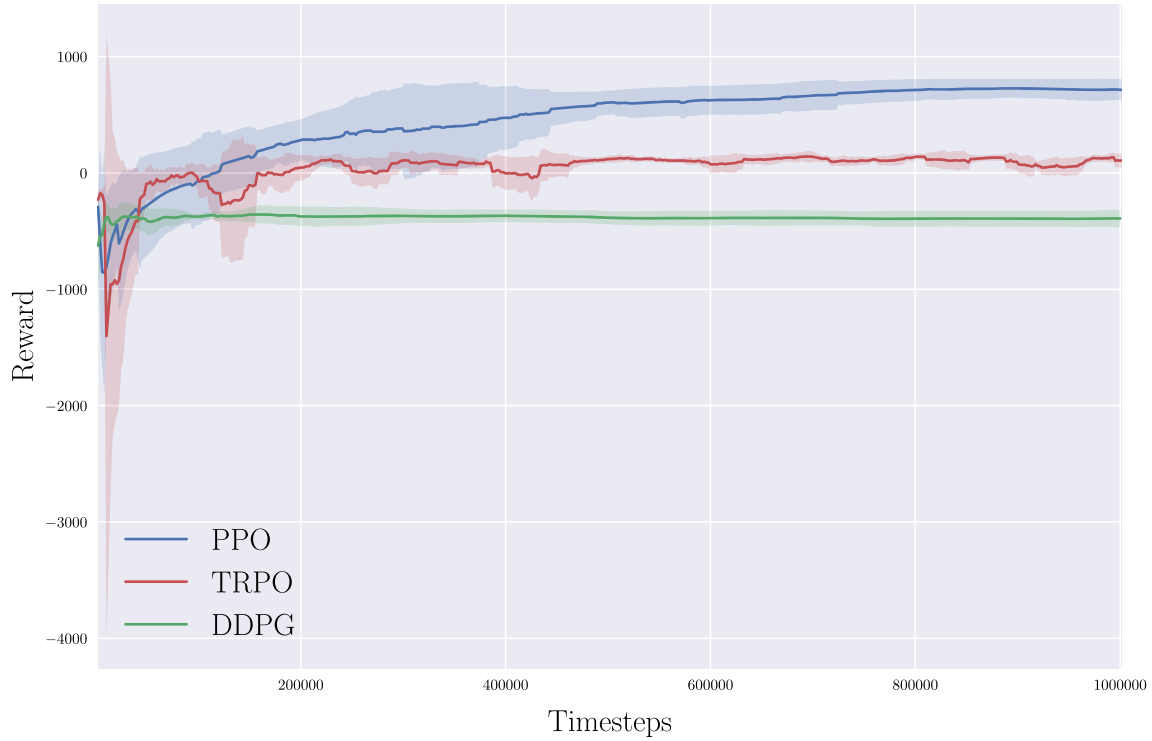

Fig. 7 Average reward for PPO, TRPO, and DDPG in the Racer task. The dark line and the transparent filling are the mean and the standard deviation over the 5 trials, respectively

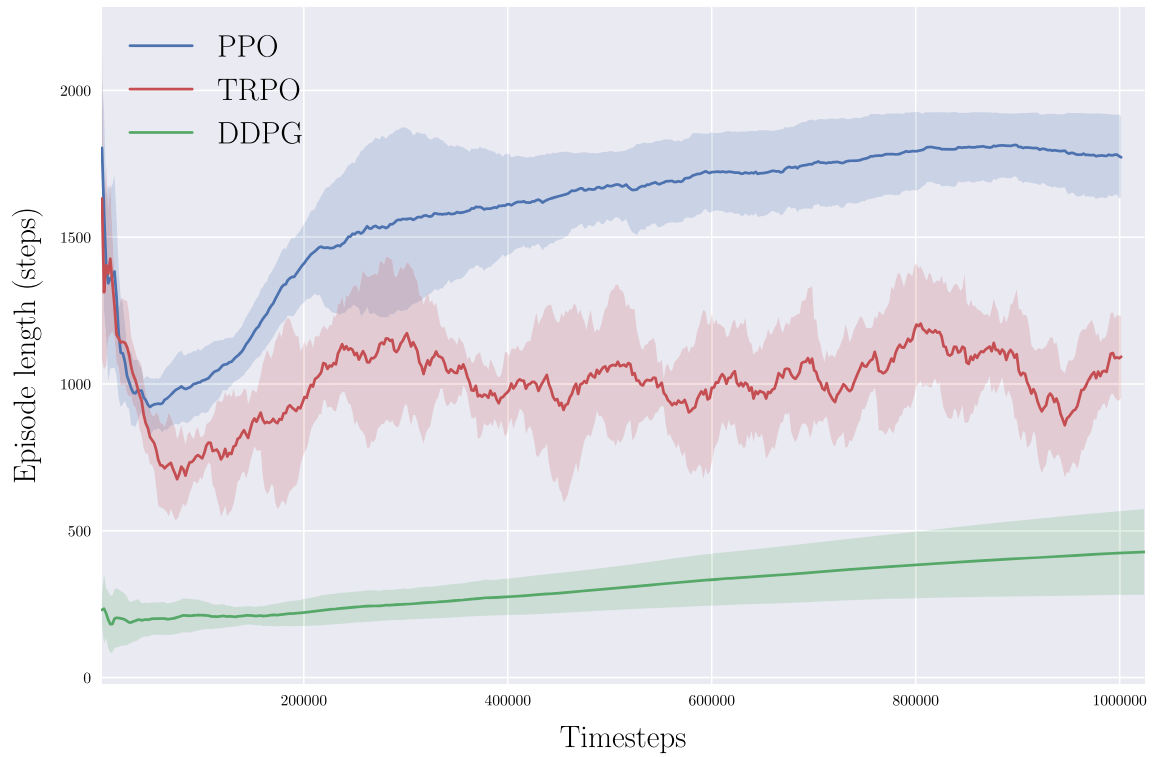

Fig. 8 Episode length for PPO, TRPO, and DDPG in the Racer task. The dark line and the transparent filling are the mean and the standard deviation over the 5 runs, respectively

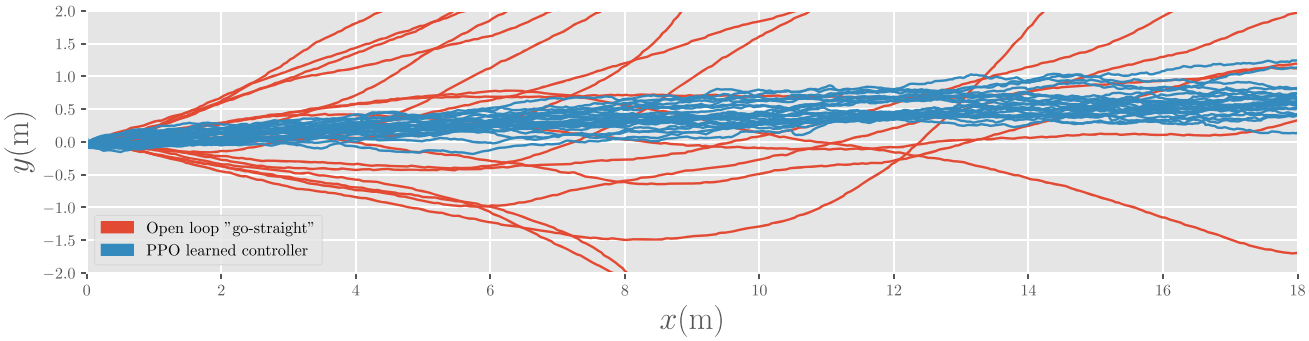

Fig. 9 Trajectories comparison between one of the trained PPO policies and the open-loop "go-straight" controller

for improvement. A video of the agent executing this task may be seen at [1].

5.2 Humanoid Soccer Dribbling Task

In this task, a learning agent and an opponent dispute the control of the ball within a rectangle of 4 × 3 m. This task's goal is for the learning agent to take control over the ball and take it to the right edge of the rectangle. This task is used as a surrogate for the actual dribbling behavior, assuming that an agent that learns to accomplish this task will also perform well in dribbling in an actual soccer match. Notice that the opponent does not learn.

Figure 10 presents this task. The yellow lines represent the 4 × 3 m rectangle while the blue and red agents are the learning and opponent agents, respectively. Moreover, notice that the blue or red line in front of an agent denotes the respective robot's torso orientation. These lines are drawn for visualization and debugging purposes.

State space: consists of the x, y, z coordinates of the robot's torso, the yaw angle of the torso θ, the forward velocity v_x, sideways velocity v_y and rotational velocity v_θ. Moreover, we also consider the ball's position with respect to (w.r.t.) the agent, the opponent's pose w.r.t. the agent, and the ball's position w.r.t. the opponent. After some trial and error, we also included the ball-agent, ball-opponent and agent-opponent distances as observations.

Action space: the desired velocity vector $\mathbf{v} = [u_x, u_y, u_\theta]^T$ used to command the walking engine.

Reward signal: defined by

$$\begin{aligned} r(s,a) \;=\; & (d^{t-1}_{\text{ball-agent}} - d^{t}_{\text{ball-agent}}) + 0.05e^{(-d^{t}_{\text{ball-agent}})} \\ & -(d^{t-1}_{\text{opp-agent}} - d^{t}_{\text{opp-agent}}) + 5\Delta x_{\text{ball}} \\ & +10\mathbb{I}^{\text{agent completed dribble}} - 10\mathbb{I}^{\text{opp completed dribble}} - 1\mathbb{I}^{\text{agent fell}}, \end{aligned} \quad (12)$$

where $\mathbb{I}^{\text{agent completed dribble}} = 1$ if the agent successfully completed the dribble and 0 otherwise; and $\mathbb{I}^{\text{opp completed dribble}} = 1$ if the opponent successfully completed the dribble and 0 otherwise. The term Δx_{ball} represents the difference between the current x position of the ball and the one on the previous timestep, which incentivizes the agent to move the ball forward. The term $e^{(-d^{t}_{\text{ball-agent}})}$ incentivizes the robot stay close to the ball. Finally, $(d^{t-1}_{\text{ball-agent}} - d^{t}_{\text{ball-agent}})$ rewards the agents for getting closer to the ball while the equivalent term for the opponent punishes the agent if the opponent gets closer to the ball.

Unfortunately, some reward engineering was required since dribbling inherently has very sparse rewards. Notice that a random agent has a negligible chance of performing a successful dribble, so it would almost never receive rewards to guide its learning. We based the reward signal on some ideas from [15, 53].

The episode duration is set to 5,000 steps. The episode may also terminate earlier if the agent falls (CoM below 0.2 m) or leaves the rectangle used to define the task. On the other hand, if the opponent leaves the rectangle, we respawn it back within the arena. A video of the learned agent performing dribbles may be seen at [2]. Furthermore, some failures cases are shown at [3].

To successfully learn this task, we employed curriculum learning [9]. It helped to guide exploration since the agent has a higher probability of obtaining positive reward from random actions in an easier task [6]. The curriculum considers two features:

– **Agent-ball distance at the beginning of the episode:** the agents start at a random position. Nonetheless, our curriculum dictates that the maximum distance the learning agent can spawn from the ball increases over the duration of the training, while the distance the opponent can start from the ball decreases. The intention here is to help the agent to touch the ball, especially at the beginning of the training, when the policy is mostly random. Figure 11 shows this

[1] https://www.youtube.com/watch?v=IF8kUpi96lE&feature=youtu.be

[2] https://www.youtube.com/watch?v=2i2q9fLjQBY

[3] https://www.youtube.com/watch?v=jtmxJuAu2dk

Fig. 10 Humanoid Soccer Dribbling Domain

curriculum – during training, the distance d_{ag} between the agent and the ball increases while the distance d_{opp} between the opponent and the ball decreases.

– **Opponent skill:** the opponent's skill level changes. At first, the opponent does not move. Then, after some time, we change to the opponent to the ITAndroids agent our team actually uses in competitions. Its dribble behavior is based on hand-coded heuristics and control theory.

Curriculum learning was essential for this task. A skilled opponent at the beginning would completely overpower a policy initialized randomly, leaving the learning agent deprived of the high rewards that come with accomplishing the task.

We ran DDPG, TRPO and PPO during $1.7\text{x}10^6$ timesteps. We compare these algorithms and analyze how the learned policy rivals with the baseline dribble behavior implemented by the ITAndroids team. In this task, each training required around 8 hours to finish, being much longer than in the previous task. We hypothesize that this is due to the additional opponent present in this task. This made testing new ideas much slower, therefore solving this task was very challenging.

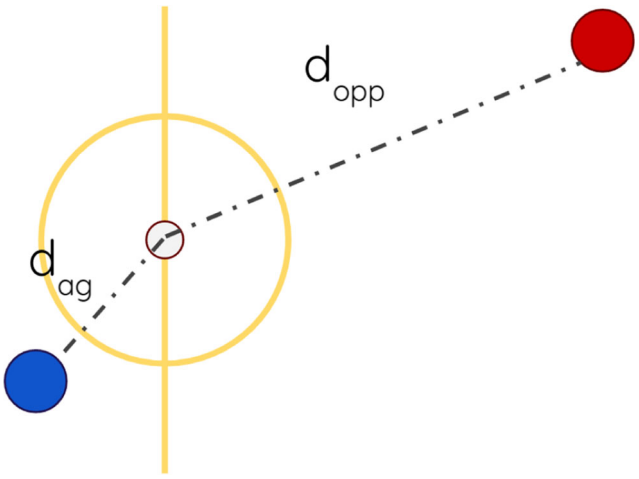

Fig. 11 Agent-ball distance curriculum. During training, we gradually increase d_{ag} and decrease d_{opp}

In a similar manner, we executed the experiments 5 times for each algorithm and computed means and standard deviations. Figure 12 presents the episode rewards during the training phase for all executions. The dark line and the transparent filling represent the mean and the standard deviation, respectively, of the reward taking all executions into account.

For PPO, there is a sudden reduction in reward when the opponent's skill is changed. This is expected since a better opponent makes completing the dribble harder, so the agent gets less reward. Similarly, Fig. 13 illustrates the mean and the standard deviation of episode length over the same five runs.

The curriculum change also induces a sharp drop in the episode duration. Notice that in this case, the agent must learn to dribble faster to beat the more skilled opponent. Moreover, the average reward here has a higher variance than in the previous task. This behavior indicates that the space of successful policies in this task is bigger than in the previous task because it is significantly more complex.

To evaluate our results, we ran the learned policy against the baseline agent for 2,000 episodes. As a metric of performance, we consider the ratio of successful dribbles:

$$M = \frac{N_{\text{agent dribbles}}}{N_{\text{agent dribbles}} + N_{\text{opp dribbles}}}, \tag{13}$$

where $N_{\text{agent dribbles}}$ and $N_{\text{oppdribbles}}$ are the number of successful dribbles by the agent and the opponent, respectively. A successful dribble happens when an agent manages to push the ball out of the task region towards its attacking side. We also computed how long, on average,

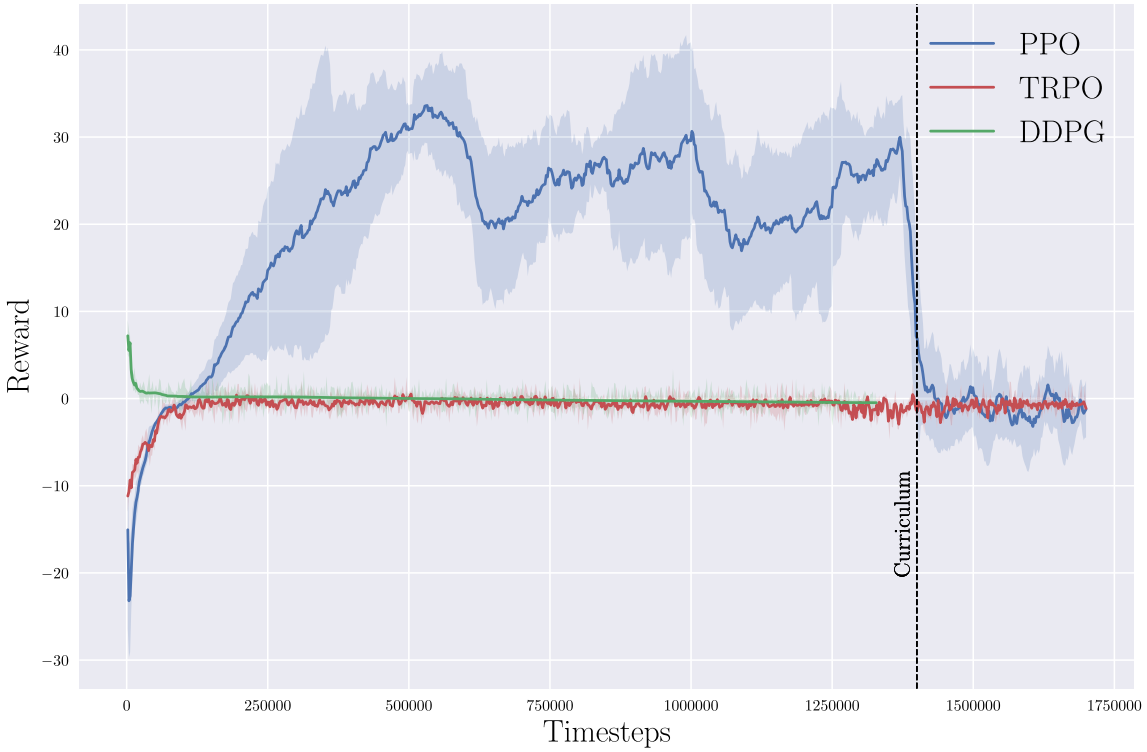

Fig. 12 Average reward for PPO, TRPO and DDPG in the dribbling task during training. The dark line is the mean over the 5 runs, and the transparent filling is the standard deviation in regards to the mean and the dotted line represents when the curriculum changes

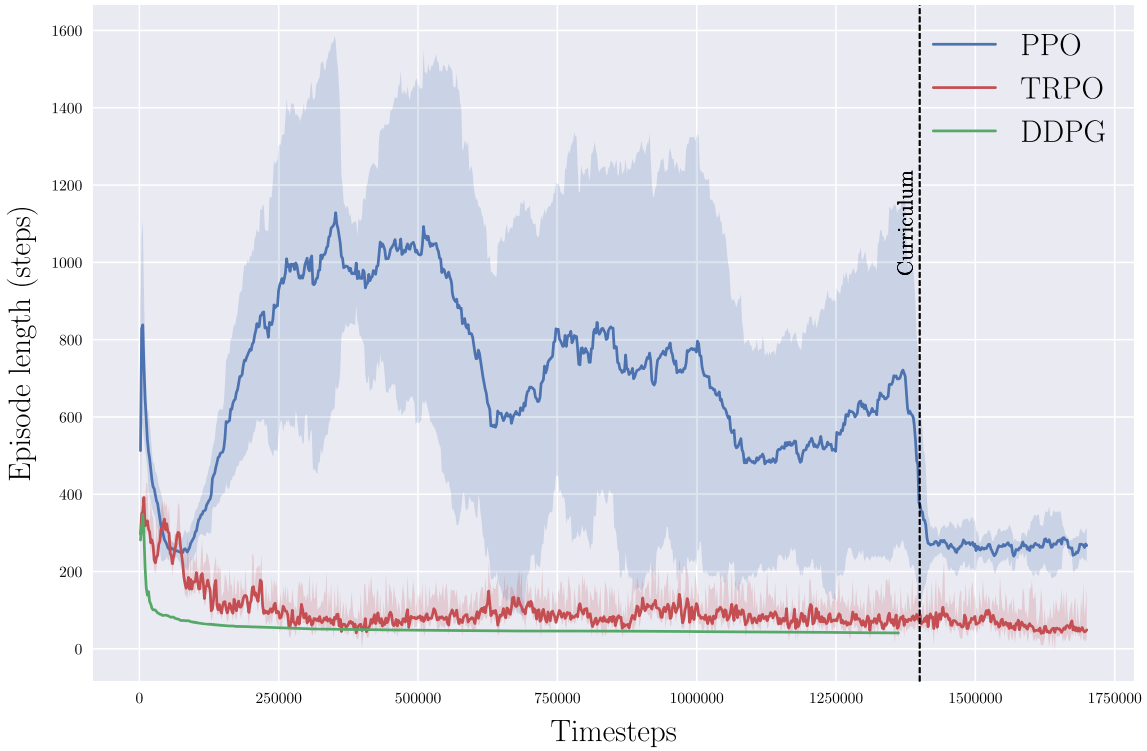

Fig. 13 Episode length for DDPG, TRPO and PPO in the dribbling task during training. The dark line is the mean over the 5 runs and the transparent filling is the standard deviation in regard to the mean and the dotted line represents when the curriculum changes

each robot takes to accomplish the task. Table 5 presents the results.

Notice that the learned agent greatly outperforms the baseline, despite taking longer on average to accomplish the task. Qualitatively, we can say that the learned policy has a good performance on average, but makes terrible decisions sometimes. Moreover, intelligent behaviors appear to emerge from the RL agent: for example, when the opponent tries to steal the ball, the agent appears to use its body to block the opponent's movement.

5.3 Concluding Remarks

As expected, dribbling presented itself as a much more challenging task compared to the other task. While the agent learned to accomplish the racing task with no curriculum learning, it was unable to learn how to dribble without a curriculum. Moreover, TRPO obtained a working policy in the racing task, but it failed to learn in the dribbling task. Finally, the dribbling task required much more data for the agent to learn an adequate policy.

As discussed in Section 3, to the best of our knowledge, this is the first work to adopt DRL to develop high-level behaviors in Soccer 3D. Therefore, we unfortunately do not know of related works similar enough to permit a performance comparison. However, the conclusions we obtained are supported by the literature. In robot soccer, many works have obtained good results for continuous control tasks using actor-critic DRL methods [3, 12, 39, 41, 43], especially PPO, as we did. Moreover, PPO is known to outperform DDPG and TRPO in complex tasks [59].

Despite using a different robot soccer competition (VSSS), Medeiros *et al.* is close in terms of methodology to ours [39], while also reaching similar conclusions. They also used PPO augmented by curriculum learning (CL) for an agent to learn to play against an opponent by commanding a model-based controller. In their problem, CL helped achieve a policy with a better final performance, but, differently from our case, the agent was also able to learn a good policy without CL. We hypothesize that the complexity of dealing with a humanoid robot makes our problem harder.

Table 5 Evaluation results of the dribbling task

Agent	Successful dribble rate (M)	Avg. dribble duration (timesteps)
Learning Agent	68.2%	298.2
Baseline Agent	31.8%	321.6

6 Conclusions

Our main objective was to learn high-level soccer behaviors using reinforcement learning in this work. We addressed the problem with state-of-the-art model-free deep reinforcement learning algorithms, namely DDPG, TRPO, and PPO. Therefore, we learned behaviors while dealing with the complex dynamics of a humanoid robot.

To facilitate, we used a hierarchical approach where the agent learns to command a model-based walking engine based on the Zero Moment Point (ZMP) concept. The walking engine receives the desired velocities in forward, lateral and rotational directions and outputs the joint angles.

We developed a DRL framework for integrating DRL algorithms with the RoboCup 3D Soccer Simulation environment to accomplish our objective. In our results, PPO achieved the best performance, which was expected, and effectively learned humanoid robot behaviors.

This work could be extended by enhancing the performance of the learned tasks or by learning new behaviors in this environment. There are also many approaches to enhance the dribbling soccer agent, for example:

- Add more information to the state space. For example, we may include feet pressure measures to try to capture walking stability.
- Instead of considering the walking engine's parameters as fixed, we could also try to learn these parameters in a hierarchical manner, similar to [33].
- Using a walking engine makes the tasks easier to be learn, but it also restricts the robot's movement to follow a certain gait pattern. A hierarchical end-to-end approach that directly learns to command the robot's joint angles could lead to better performance [19].
- In the soccer task, the (fixed) opponent is assumed as part of the environment. A novel approach for competitive environments is to employ self-play, i.e., both the agent and its opponent are learning the task. For example, [4] shows it is possible to use meta-learning in non-stationary tasks such as those of self-play.
- Try different state-of-the-art DRL algorithms, such as Actor-Critic with Experience Replay (ACER) [66] or Soft Actor-Critic (SAC) [22].
- Execute hyperparameter search for better learning performance [10].

Finally, the same learning framework can be used to learn other high level soccer behaviors, such as stealing the ball or goalkeeping. An even harder challenge is to transfer the skills learned on the simulated robot to a real robot. Unfortunately, learning algorithms are known to overfit to inaccuracies present even in high-fidelity simulators [16], making this transfer impracticable. This

problem is exacerbated with humanoid robots, due to their complex dynamics. For example, optimizing walking skills in SimSpark often leads to tiny step sizes, generating walking patterns that do not look realistic [31].

Acknowledgements The authors acknowledge the ITAndroids' Soccer3D team for developing the code base used in this work. Moreover, we would like to thank ITAndroids' sponsors: Altium, Cenic, Intel, ITAEx, Mathworks, Metinjo, Micropress, Polimold, Rapid, Solidworks, ST Microelectronics, WildLife, and Virtual Pyxis.

Author Contributions All authors have contributed to the concept and design of the research. Alexandre Muzio is the main contributor: he developed the RL formulations, implemented the source code, and executed the experiments. Marcos Maximo and Takashi Yoneyama assumed advisor roles during the research, discussing ideas and providing insights when needed. Marcos Maximo prepared this manuscript based on material previously written by Alexandre Muzio. Takashi Yoneyama further contributed by revising the text. The final manuscript was revised and approved by all authors.

Funding Alexandre Muzio received an Master's scholarship from CAPES (number 88882.161989/2017-01). Takashi Yoneyama is partially funded by CNPq – National Research Council of Brasil through the grant 304134/2-18-0.

Availability of data and material No extra data or material is available.

Code Availability The source code for the client is available at: https://github.com/alexandremuzio/rlearning3d

Declarations

Conflict of Interests The authors declare that they have no conflicts of interest/competing interests.

References

1. Simspark. http://simspark.sourceforge.net/wiki/index.php/Main_Page (2004)
2. Abdolmaleki, A., Simões, D., Lau, N., Reis, L.P., Neumann, G., Sarıel, S., Lee, D.D.Behnke, S., Sheh, R. (eds.): Learning a humanoid kick with controlled distance. Springer International Publishing, Cham (2017)
3. 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, Australia (2019)
4. Al-Shedivat, M., Bansal, T., Burda, Y., Sutskever, I., Mordatch, I., Abbeel, P.: Continuous adaptation via meta-learning in nonstationary and competitive environments. arXiv:1710.03641 (2017)
5. Alcaraz-Jiménez, J., Herrero-Perez, D., Barberá, H.: A closed-loop dribbling gait for the standard platform league (2014)
6. Bansal, T., Pachocki, J., Sidor, S., Sutskever, I., Mordatch, I.: Emergent complexity via multi-agent competition. arXiv:1710.03748 (2017)
7. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems (1983)
8. Bengio, Y., Courville, A.C., Vincent, P.: Unsupervised feature learning and deep learning: A review and new perspectives. arXiv:1206.5538 (2012)
9. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, pp. 41–48. ACM, New York (2009). https://doi.org/10.1145/1553374.1553380
10. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. J. Mach. Learn. Res. **13**, 281–305 (2012). http://dl.acm.org/citation.cfm?id=2188385.2188395
11. Carvalho Melo, D., Quartucci Forster, C.H., Omena de Albuquerque Máximo, 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)
12. 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)
13. Depinet, M., MacAlpine, P., Stone, P.Bianchi, R.A.C., Akin, H.L., Ramamoorthy, S., Sugiura, K. (eds.): Keyframe sampling, optimization, and behavior integration: Towards long-distance kicking in the Robocup 3D simulation league. Springer, Berlin (2015)
14. Dhariwal, P., Hesse, C., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Openai baselines. https://github.com/openai/baselines (2017)
15. Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P.: Benchmarking deep reinforcement learning for continuous control. arXiv:1604.06778 (2016)
16. 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). AAMAS, Saint Paul (2013)
17. 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)
18. Florensa, C., Held, D., Wulfmeier, M., Abbeel, P.: Reverse curriculum generation for reinforcement learning. arXiv:1707.05300 (2017)
19. Frans, K., Ho, J., Chen, X., Abbeel, P., Schulman, J.: Meta learning shared hierarchies. arXiv:1710.09767 (2017)
20. Gabel, T., Riedmiller, M., Trost, F.: A Case Study on Improving Defense Behavior in Soccer Simulation 2D: The NeuroHassle Approach, pp. 61–72. Springer, Berlin (2009). https://doi.org/10.1007/978-3-642-02921-9_6
21. Google: Protocol buffers. https://developers.google.com/protocol-buffers/ (2017)
22. Haarnoja, T., Zhou, A., Abbeel, P., Levine, S.: Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor (2018)
23. Hausknecht, M., Stone, P.: Deep reinforcement learning in parameterized action space. In: Proceedings of the International Conference on Learning Representations (ICLR). ICLR, San Juan (2016)
24. Heess, N., TB, D., Sriram, 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 (2017)
25. 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 (2001)
26. Kim, J., Kim, B., Yoon, J., Lee, M., Jung, S.Y., Choi, J.: Robot soccer using deep q network. In: 2018 International Conference on Platform Technology and Service (Platcon), pp. 1–6 (2018)

27. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., Osawa, E., Matsubara, H.: Robocup: A challenge problem for ai. AI Magazine **18**(1), 73 (1997). https://doi.org/10.1609/aimag.v18i1.1276. https://aaai.org/ojs/index.php/aimagazine/article/view/1276
28. Leottau, D.L., del Solar, J.R., MacAlpine, P., Stone, P.: A study of layered learning strategies applied to individual behaviors in robot soccer. In: Almeida, L., Ji, J., Steinbauer, G., Luke, S. (eds.) RoboCup-2015: Robot Soccer World Cup XIX, Lecture Notes in Artificial Intelligence. Springer, Berlin (2016)
29. Leottau, L., Celemin, C., del solar, J.R.: Ball dribbling for humanoid biped robots: A reinforcement learning and fuzzy control approach (2014)
30. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. arXiv:1509.02971 (2015)
31. 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). AAAI, Toronto (2012)
32. MacAlpine, P., Stone, P.: Overlapping layered learning. Artificial Intelligence **254**, 21–43 (2018). https://doi.org/10.1016/j.artint.2017.09.001. https://www.sciencedirect.com/science/article/pii/S0004370217301066
33. MacAlpine, P., Stone, P.: Overlapping layered learning. Artificial Intelligence **254**, 21–43 (2018). https://doi.org/10.1016/j.artint.2017.09.001. https://www.sciencedirect.com/science/article/pii/S0004370217301066
34. 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, Berlin (2018)
35. Matiisen, T., Oliver, A., Cohen, T., Schulman, J.: Teacher-student curriculum learning. arXiv:1707.00183 (2017)
36. Maximo, M.R., 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
37. Maximo, M.R.O.A.: Omnidirectional Zmp-based walking for a humanoid robot. Master's thesis, Instituto Tecnológico de Aeronáutica (2015)
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, Brazil (2016)
39. de Medeiros, T.F., de Máximo, A., M.R.O., Yoneyama, T.: Deep reinforcement learning applied to ieee very small size soccer strategy. In: 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE), pp. 1–6 (2020). https://doi.org/10.1109/LARS/SBR/WRE51543.2020.9306954
40. Melo, D., Soares, E.E., Moreira, E., Muniz, F., Marra, G., Nahum, G., Lopes, H., Saraiva, J.L., José Otávio Vidal, J.F., Melo, L., Maximo, M.: Itandroids soccer3d team description paper 2017. https://www.robocup2017.org/file/symposium/soccer_sim_3D/ITAndroids3D_TDP.pdf (2017)
41. Melo, D.C., Máximo, M.R.O.A., da Cunha, A.M.: Push recovery strategies through deep reinforcement learning. In: 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE), pp. 1–6 (2020). https://doi.org/10.1109/LARS/SBR/WRE51543.2020.9306967
42. Melo, L.C.: Imitation Learning and Meta-Learning for Optimizing Humanoid Robot Motions. Master's Thesis, Instituto tecnológico de aeronáutica, são josé dos Campos, SP Brazil (2019)
43. 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)
44. Melo, L.C., Melo, D.C., Maximo, M.R.O.A.: Learning humanoid robot running motions with symmetry incentive through proximal policy optimization. Journal of Intelligent &, Robotic Systems **102**(3), 54 (2021). https://doi.org/10.1007/s10846-021-01355-9
45. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. arXiv:1602.01783 (2016)
46. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015). https://doi.org/10.1038/nature14236. Letter
47. Muniz, F., Maximo, M.R., Ribeiro, C.H.: 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). https://doi.org/10.1109/LARS-SBR.2016.20
48. Muzio, A., Melo, D., Henrique, E., Muniz, F., Marzzo, I., Saraiva, J.L., Melo, L., Aguiar, L.G., Maximo, M., Bertolino, M.: Itandroids soccer3d team description paper 2016. http://www.robocup2016.org/media/symposium/Team-Description-Papers/Simulation3D/RoboCup_2016_Sim3D_TDP_ITAndroids3D.pdf/ (2016)
49. Muzio, A.F.V.: Curriculum-based Deep Reinforcement Learning Applied to Humanoid Robots. Master's Thesis, Instituto tecnológico de aeronáutica, são josé dos Campos, SP Brazil (2018)
50. Muzio, A.F.V., Maximo, M.R.A., Yoneyama, T.: Deep reinforcement learning for humanoid robot dribbling. In: 2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE), pp. 1–6 (2020). https://doi.org/10.1109/LARS/SBR/WRE51543.2020.9307084
51. Obst, O., Murray, J., Boedecker, J., Rollmann, M., Ebrahimi, M., Vatankhah, H., van Dijk, S., Yuan, X.: Simspark effectors. https://gitlab.com/robocup-sim/SimSpark/wikis/Effectors (2004)
52. ODE: Open dynamics engine (ode). http://www.ode.org/ (2004)
53. Peng, X.B., Berseth, G., Yin, K., van de Panne, M.: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Transactions on Graphics Proc SIGGRAPH **36**(4), 2017 (2017)
54. Peng, X.B., Chang, M., Zhang, G., Abbeel, P., Levine, S.: Mcp: Learning composable hierarchical control with multiplicative compositional policies. In: Wallach, H., Larochelle, H., Beygelzimer, A., D'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 32, pp. 3681–3692. Curran Associates Inc (2019). http://papers.nips.cc/paper/8626-mcp-learning-composable-hierarchical-control-with-multiplicative-compositional-policies.pdf
55. Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R.Y., Chen, X., Asfour, T., Abbeel, P., Andrychowicz, M.: Parameter space noise for exploration. arXiv:1706.01905 (2017)
56. Robotics, S.: Nao robot. https://www.ald.softbankrobotics.com/en/robots/nao (2018)
57. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization. arXiv:1502.05477 (2015)

58. Schulman, J., Moritz, P., Levine, S., Jordan, M.I., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. In: Bengio, Y. (ed.) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings (2016). arXiv:1506.02438
59. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv:1707.06347 (2017)
60. Schwab, D.: Robot deep reinforcement learning: Tensor state-action spaces and auxiliary task learning with multiple state representations. Ph.D. thesis, Carnegie Mellon University (2020)
61. Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M.: Deterministic policy gradient algorithms. In: Proceedings of the 31st International Conference on International Conference on Machine Learning - vol 32, ICML'14, pp. I–387–I–395. JMLR.org (2014). http://dl.acm.org/citation.cfm?id=3044805.3044850
62. Spitznagel, M., Weiler, D., Dorer, K.: Deep reinforcement multi-directional kick-learning of a simulated robot with toes. In: 2021 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), pp. 104–110 (2021). https://doi.org/10.1109/ICARSC52212.2021.9429811
63. Stoecker, J.: Roboviz. https://github.com/magmaOffenburg/RoboViz (2011)
64. Sutton, R.S., Barto, A.G. Introduction to Reinforcement Learning, 1st edn. MIT Press, Cambridge (1998)
65. 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)
66. 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)
67. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, King's College (1989)
68. Wiliams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning (1992)
69. Zaremba, W., Sutskever, I.: Learning to execute. arXiv:1410.4615 (2014)

Alexandre F. V. Muzio received the BSc degree in Computer Engineering and the MSc degree in Electronic and Computer Engineering from Aeronautics Institute of Technology (ITA), Brazil, in 2017 and 2018, respectively. Alexandre Muzio is currently at Microsoft working on large scale multilingual machine translation.

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.

Takashi Yoneyama received the bachelor's degree in electronic engineering from the Aeronautics Institute of Technology (ITA), São José dos Campos, Brazil, the M.D. degree in medicine from the Universidade de Taubaté, Taubaté, Brazil, and the Ph.D. degree in electrical engineering from the Imperial College London, London, U.K., in 1983. He is with the Electronic Engineering Department, ITA as a Professor of Control Theory. He has published more than 100 journal papers, 300 articles in scientific events, four books and has supervised more than 110 theses and dissertations. His research concerns mainly stochastic optimal control theory. Prof. Yoneyama served as the President of the Brazilian Automatics Society from 2004 to 2006.