Learning to run faster in a humanoid robot soccer environment through reinforcement learning

Miguel Abreu[1 [0000−0002−6342−2054]], Luis Paulo Reis[1 [0000−0002−4709−1718]], and Nuno Lau[2 [0000−0003−0513−158X]]

[1] University of Porto, LIACC/FEUP, Artificial Intelligence and Computer Science Lab, Faculty of Engineering of the University of Porto, Portugal,
{m.abreu, lpreis}@fe.up.pt
[2] University of Aveiro, DETI/IEETA, Institute of Electronics and Informatics Engineering of Aveiro, University of Aveiro, Portugal,
nunolau@ua.pt

Abstract. Reinforcement learning techniques bring a new perspective to enduring problems. Developing skills from scratch is not only appealing due to the artificial creation of knowledge. It can also replace years of work and refinement in a matter of hours. From all the developed skills in the RoboCup 3D Soccer Simulation League, running is still considerably relevant to determine the winner of any match. However, current approaches do not make full use of the robotic soccer agents' potential. To narrow this gap, we propose a way of leveraging the Proximal Policy Optimization using the information provided by the simulator for official RoboCup matches. To do this, our algorithm uses a mix of raw, computed and internally generated data. The final result is a sprinting and a stopping behavior that work in tandem to bring the agent from point a to point b in a very short time. The sprinting speed stabilizes at around 2.5m/s, which is a great improvement over current solutions. Both the sprinting and stopping behaviors are remarkably stable.

Keywords: Reinforcement Learning · Machine learning · Humanoid · Robot · Soccer · Running

1 Introduction

Reinforcement learning is constantly expanding its reach to replace outdated solutions. Its ability to overcome problems with large state and action spaces is becoming more relevant as the computational power increases and new optimization algorithms are developed. Competitive environments are particularly prone to adopt this trend given the intense demand for new and improved solutions. This is particularly noticeable in the RoboCup competition. Every year, new robotic skills are designed, making the competition more exciting. Ultimately, developing and optimizing high performance behaviors can be useful for real life situations in the future.

One of the most important skills in a 3D soccer simulation match is running. The first player to reach the ball has an advantage over the opposing team, which may prove to be decisive if done consistently. To improve the odds, researchers should focus on developing fast and stable running mechanisms. With this in mind, every competing team has perfected their algorithms along the years. However, none is making use of the full potential of the robotic soccer agents. This is clear in the main event as well as secondary challenges of the RoboCup competition. Recent work has introduced new perspectives on this long-established problem [1]. Nonetheless, the proposed solution is not mature enough to be integrated in existing teams.

To bring the robotic soccer agents closer to their full potential, this paper introduces a way of leveraging the Proximal Policy Optimization using the information provided by the simulator for official RoboCup matches. To do this, our algorithm uses a mix of raw, computed and internally generated data. To complement the sprinting stage, we also optimize a stopping algorithm which brings the robot back to a stationary pose. We expect to improve the running performance and bring a new pace for future RoboCup competitions.

In section 2 we present some work related with the RoboCup competition. Then, we introduce the methods used to train and test our approach (section 3), including a description of the simulator and agent, the employed state and action spaces and the optimization method. In section 4 we describe the results concerning the learned sprinting and stopping behaviors. Then we discuss those results (section 5) and present some conclusions and future work (section 6).

2 Related Work

RoboCup is an annual international robotics competition which gathers teams from multiple countries annually since 1997 [2]. Since the adoption of the 3D Soccer Simulation League (3DSSL) in 2004, low level behaviors have evolved considerably. Every year, each participating team releases their binary to a public repository [3], which contains every team until the 2018 champion [4]. This repository is important to analyze competing teams and develop improved solutions. One of the most important research fields in the competition is bipedal locomotion. Contributions in this area extend over different areas, including omnidirectional walk [8, 9], contextual-based walking [10, 11], and fast walking [12]. Currently, there is a lot of room for improvement concerning running speed.

There is a RoboCup competition called Gazebo running challenge [5], which tests the agents' speed and stability using the Gazebo simulator. The last RoboCup event that included this challenge was in 2017. Every participating team would run for 20 seconds at the highest possible speed. Only two of the four participants were able to complete the challenge [6]. UTAustinVilla averaged $1.19m/s$ on the best three of four runs, while magmaOffenburg averaged $0.38m/s$ on the same conditions. Previously, in 2014, the RoboCup's running challenge was conducted in SimSpark. In this version UTAustinVilla also had the best results, achieving a speed of $1.343m/s$ [7].

Recent work on this subject has brought improvements to the running performance [1]. The Proximal Policy Optimization algorithm was used to develop running behaviors from scratch using data provided by the robot's sensors, as well as the real orientation and coordinates provided by the server. This last type of data is not allowed in official RoboCup matches, and thus the resulting behavior is not ready to be integrated in a team. Furthermore, some components can be improved, including speed, stability and integration convenience.

3 Methods

Two complementary behaviors are being developed – sprinting and stopping. The former is concerned with making the transition from a stationary position to a running gait, while the latter is responsible for reversing that transition. Most of the features introduced below while describing the used methods are common for both behaviors.

3.1 Simulator and agent

SimSpark [13] – the official simulator for the RoboCup 3D Soccer Simulation League – was the chosen simulator to conduct the learning process. This option ensures that the optimization results can run seamlessly in official RoboCup matches. Also, SimSpark uses a realistic and stable physics engine – the Open Dynamics Engine (ODE). The learning agent is a version of the NAO humanoid robot [14], created by SoftBank Robotics.

Since the objective of this work is to integrate the learned behaviors in a competing team, the simulator was configured with current official RoboCup rules. However, during the learning stage, to accelerate the optimization, the real-time mode was disabled to allow synchronous communication with the agent. Comparing with previous work [1], in our approach, the noise was not disabled and the visual perception was not extended (i.e., the position and orientation were not obtained from the server, since this kind of features is not available during the competition).

3.2 State Space

The state space is composed of 80 continuous variables represented by the single-precision (32b) floating-point data type. These variables are grouped in Table 1, according to their meaning. The gyroscope, accelerometer and joints data is obtained directly from the simulator. There are a total of 22 joints in the NAO robot version without toes, as seen in Fig. 1, from which only 20 are being optimized. The head pitch and yaw are excluded from the state space. The feet force data is acquired directly when the feet are touching the ground, and it includes 3D contact coordinates and 3D force vectors for each foot. When the agent's feet are not touching the ground, SimSpark will not provide information

Table 1. State Parameters

Parameter	**Data size** (×32b)	**Acquisition Method**
counter	1	generated
z-coordinate	1	computed
orientation*	1	computed
gyroscope*	3	raw
accelerometer*	3	raw
feet force*	12	raw
joint's position*	20	raw
***differentiation**	39	computed

*differentiation is performed on parameters followed by an asterisk

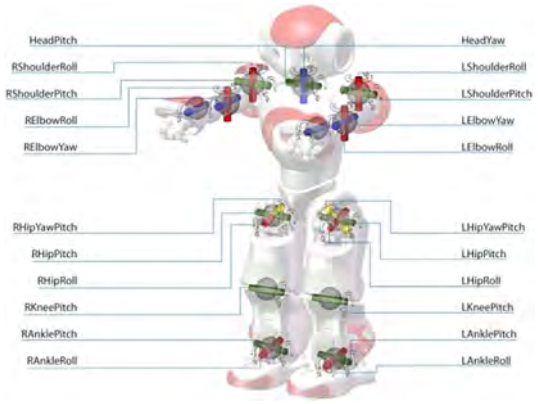

Fig. 1. NAO robot – Joints and Actuators (adapted from [22])

in the respective cycle and our agent assumes that the corresponding variables are all set to zero.

The counter is generated by the agent. It starts from zero, when the robot is stopped, and it is incremented at every 3 time steps (which is related to the control cycles explained in the next section). This counter works like a timer for the optimization algorithm, and, during our preliminary tests, its inclusion in the state space yielded about 20% of speed improvement after the optimization was concluded. In this phase, we also attested the importance of the robot torso's z-coordinate and orientation. The former provides useful information when the feet data is unavailable and the orientation is essential to keep the robot aligned with the objective.

The orientation is easily determined from the visual references provided by the simulator. However, these references are not good enough to provide accurate z-coordinate estimates. Therefore, this variable is computed with a linear predictor function that was modeled using linear regression on some of the other state space variables. The best single features and features crosses were initially filtered based on their correlation with the ground-truth, which was obtained directly from SimSpark. Then, an algorithm based on the Akaike information

criterion was used to refine the feature selection [15, 16]. Currently, this predictor uses 368 features (including raw observations and feature crosses). In a new data set, corresponding to a reinforcement learning optimization with 20M time steps, it yielded an average absolute error of 0.011m. This is accurate, considering that all the data provided to the agent by SimSpark (under the configuration for official matches) has an associated error of 0.005 units, due to rounding precision.

Finally, the variables in Table 1 followed by an asterisk are differentiated with respect to time, considering the length of one time step (20ms). The majority of them constitute first-order derivatives of linear or angular positions. The exceptions are the rate of force change (df/dt); the torso's angular acceleration; and the torso's jerk (time derivative of acceleration).

3.3 Action Space

All joints of the simulated NAO robot (Fig. 1) have an associated actuator. Analogously to the state space, 20 actuators were used. The head pitch and yaw actuators were excluded since their optimization was not part of the objective. To control each actuator, the simulator requires angular speeds between -7°/s and +7°/s, provided through continuous variables. Our agent produces these values through an indirect control approach [1], where the optimization algorithm produces target angles for each joint instead of speed values for each actuator. The scaled target angle for joint i (θ_i^{target}) and the output angular speed for actuator i at time step t (ω_i^t) are obtained by the following equations:

$$\begin{aligned} \theta_i^{target} &= (x_i + 1) * \frac{b_i - a_i}{2} + a_i, \\ \omega_i^t &= (\theta_i^{target} - \theta_i^{t-1})/k, \end{aligned} \tag{1}$$

where x_i is the optimization algorithm output value for joint i, clipped to the range $[-1, 1]$; a_i and b_i are the lower and upper bounds, respectively, of the joint's angular range; θ_i^{t-1} is the angular position of joint i in the previous time step; and k is a constant that adjusts the speed at which the target angle is reached. Finally, ω_i^t is clipped to the range $[-7, 7]$ before being sent to the simulator. After a random search, k was set to 7.

In official RoboCup matches, the NAO robot's vision perception is limited to an interval of 3 time steps, and all observations are delayed 1 time step. The latter limitation allows the simulator to process the current action at the same time the agents compute the next action using the last available observations. This improves computational time but hinders the performance of reinforcement learning algorithms, since the current action must take into account the last action without knowing its result. To manage these issues, the agent control cycle was synchronized with the arrival of visual information. The observations fed to the optimization algorithm carry information about the last action, and the following yielded action is repeated for the next 3 time steps. This solves the disruption of time continuity and reduces computational requirements from the agent's side.

3.4 Optimization Method

We chose the Proximal Policy Optimization (PPO), introduced by Schulman et al. [17], as the preferred optimization method. This decision was based on its simplicity and good performance on producing high quality behaviors [17].

The chosen implementation was obtained from OpenAI baselines [18], which uses the clipped surrogate objective:

$$L(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t],$$
$$where \;\; r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}(a_t|s_t)}}, \tag{2}$$

where $\hat{A}_t$ is an estimator of the advantage function at timestep t. The clip function clips the probability ratio $r_t(\theta)$ in the interval given by $[1-\epsilon, 1+\epsilon]$. This implementation alternates between sampling data from multiple parallel sources, and performing several epochs of stochastic gradient ascent on the sampled data, to optimize the objective function.

Hyperparameter optimization requires a large number of iterations to cover a significant part of the hyperparameter space. Therefore, due to the similarity between case studies, we opted to base our hyperparameter tuning on the best solution found by OpenAI, when training the mujoco 3D humanoid [18]. They proposed the optimization of a multilayer perceptron with two hidden layers with 64 neurons. On top of that solution, we changed to 4096 time steps per actor batch and increased the maximum number of time steps to 200M (to learn the sprinting behavior) and 40M (to learn the stopping behavior).

3.5 Experimental Setup and Testing

The simulator was configured identically for the experimental and test setups. The real-time mode was switched on during testing to allow for visual inspection of the resulting behavior. Otherwise, it remained off to reduce computational time. For the sprinting behavior, the experimental and test setups were the same. The agent is initially placed near the field's left goal at (-14m, 0m), facing the field's right goal. The initial pose is shown in Fig. 2**a**. To achieve this pose all joints must be set to 0°, with the exception of: knees pitch (-60°), hips pitch (30°), ankles pitch (30°), shoulders pitch (-90°), elbows yaw (L:-90°,R:90°), elbows roll (L:-90°,R:90°). The agent is encouraged to run as fast as possible in a straight line since the reward is only given by the robot torso's x-coordinate difference from the last time step. As the traveled distance in the x-axis increases, so does the reward. The episode ends if the robot falls (torso's z-coordinate is below 0.27m) or reaches the finish line (torso's x-coordinate is above 14m).

The stopping behavior is trained after the optimization of the sprinting behavior. It does not have a fixed initial position or pose. The robot starts from a stopped position and runs for a random period of time. Then, the control of the robot is assumed by the stopping optimization algorithm, which must ensure

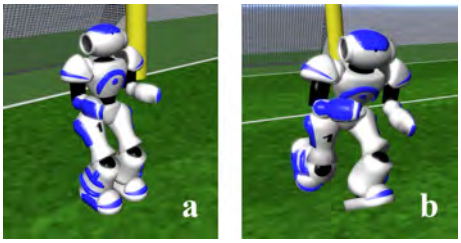

Fig. 2. Initial poses for the sprinting (**a**) and stopping (**b**) behaviors

that it goes back to a stationary position without falling. One possible initial position and pose is shown in 2**b**. The reward is given by the difference between the distance of the current joint's position to the desired pose, and the distance of the previous joint's position to the desired pose. In other words, the agent is compensated for getting closer to the desired pose, and penalized for getting farther. Furthermore, it also receives a fixed unitary bonus for surviving another control cycle without falling.

The target pose is the same as shown in 2**a** but with both arms lowered to bring the center of mass down and increase stability. To ensure that the final pose is completely achieved, after the episode reaches 50 time steps (1 second), an algorithm gradually takes over (using a weighted average) to force the joints to the desired angles. This also means that if the robot is not stable in under a second, it will probably fall and the episode will end. If successful, the episode ends when the robot achieves the final pose and remains stable for 100 time steps (2 seconds), receiving a fixed unitary bonus per control cycle.

During testing, the only difference is the extent of the running stage, before the stopping algorithm takes control. Instead of randomly assigning a value to the extent, it was gradually increased from 0.42s to 1.80s, allowing 100 episodes to run before incrementing 0.06s (3 time steps). This test was implemented to compare the stopping behavior performance under different initial conditions.

4 Results

This section presents the results obtained while testing the sprinting and testing behaviors, according to the conditions established in section 3.5. A video demonstration is available online at https://youtu.be/lkSVad_tjOY.

4.1 Sprinting

The learning process took 73h on a single Intel Xeon E5-2620 v3 at 2.40GHz. The results for this experiment were averaged over 1000 episodes executed under the conditions specified in section 3.5. After the optimization was concluded, the trained agent deviated, on average, -3° from the x-axis, with a standard deviation (SD) of $\pm 5.9 \times 10^{-2^\circ}$. This bias was compensated by adjusting the

orientation observation on the opposite direction. The following results were obtained after this fix was implemented.

Fig. 3 shows the robot's average speed along the x-axis (x-speed) from a given starting line (perpendicular to the x-axis) until the end of the episode. When the starting line is defined at 0m, the average x-speed covers the entire episode, including the initial stopped position. If the line is defined at 1m, only the trajectory performed after the robot torso's center crosses it is considered for the average x-speed computation.

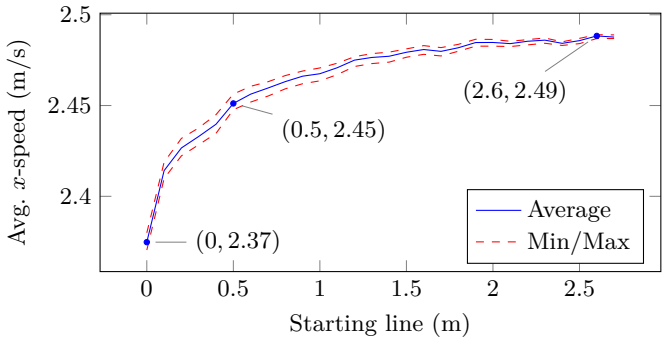

Fig. 3. Average robot speed along the x-axis from a given starting line (perpendicular to the x-axis) until the end of the episode (blue line). The dashed red lines represent the minimum and maximum values. These results were averaged over 1000 episodes.

In this experiment, the robot has not fallen once in 1000 episodes. Considering the entire trajectory, from a stopped position until the robot travels 28m, the average x-speed ranged between 2.37m/s and 2.38m/s, with a SD of $\pm$2.8mm/s. Using the technique employed by the magmaChallenge benchmark tool [19], which starts measuring the robot's x-speed after it crosses the 0.5m line, our approach averaged 2.45m/s on SimSpark, with a SD of $\pm$2.1mm/s. After 2.5m the agent stabilizes at around 2.49m/s.

Fig. 4 allows the analysis of a single episode. Fig. 4**a** shows the instantaneous x-speed, represented by a continuous blue line. The average speed is defined by a dotted red line. It is computed by considering the current time instant t until the episode's end T, where the travelled distance is given, analogously, from x_t until x_T, as denoted by $(x_T - x_t)/(T - t)$. Note that this expression is dependent on time instants instead of x-coordinates.

Initially, the robot was standing still. Then it accelerated and crossed the 0.5m imaginary line. At that moment, 0.54s had passed, and the robot's instantaneous x-speed (for which we consider the last 3 steps = 60ms) was 1.23m/s. This point in the graph is near a local minimum, which is explained by the current running gait cycle. By observing Fig. 4**b**, it is possible to study the movement dynamics which leads to the speed oscillations while running.

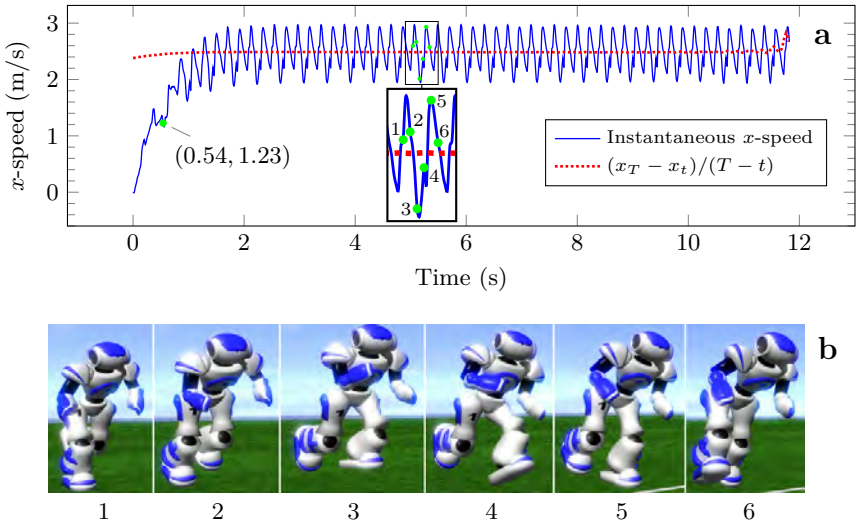

Fig. 4. Robot's speed along the x-axis in a single episode, on top (**a**). The continuous blue line indicates the instantaneous speed while the dotted red line represents the average speed from the current time value t until the end of the episode T. A sequence of frames from that episode is shown on the bottom (**b**). Each frame corresponds to one of the numbered time steps in (**a**).

As Adelaar and Novacheck defined [20, 21], running, in comparison with walking, is characterized by longer double floating periods, where there is no support from any limb. This characteristic is seen during 27% of the time for the episode shown in Fig. 4. The sequential frames displayed on the figure's bottom correspond to the numbered time steps on top. Each pose is associated with a specific cycle of the running gait. The lowest speed is registered when the robot is still floating (frame no. 3).

4.2 Stopping

The stopping behavior is coordinated with the sprinting behavior to bring the robot, once again, to a stationary pose. The results for this experiment were averaged over 2400 episodes executed under the conditions specified in section 3.5. Fig. 5 shows the stopping behavior results for different initial conditions. The robot has not fallen once during this evaluation. In the first batch of 100 episodes, the robot ran for 0.42s, achieving an average maximum speed of 1.24m/s. After that moment, the stopping algorithm took control and stopped the robot, on average, in 0.39m with an angle deviation of 115°. In the following batches, the initial speed increases, although not monotonically, for the reasons already stated in section 4.1. The fastest initial speed was obtained for a running extent of 1.74s, for which the average max. speed was 2.79m/s. With these initial conditions, the robot stopped, on average, in 0.72m with an angle deviation of 109°.

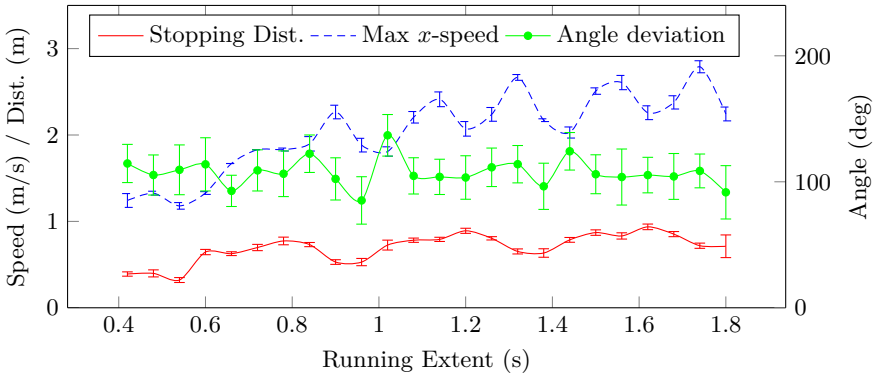

Fig. 5. Stopping results as a function of the initial running extent. The robot starts from a stopped position and runs for a certain extent, between 0.42s and 1.80s. Then, the stopping algorithm (SA) tries to bring the robot back to a predefined stationary pose. The max. x-speed ± 1 SD (dashed blue line) is obtained immediately before the SA starts. The stopping distance ± 1 SD (cont. red line) is the 2D Euclidean distance traveled by the robot while the SA is operating. Analogously, the angle deviation ± 1 SD (green dots) is the absolute rotation angle measured while the SA is running. In total, 24 running extents were tested, with 100 episodes each, totalling 2400 episodes.

5 Discussion

Regarding the sprinting behavior, the obtained results were a considerable improvement over current RoboCup standards. Our agent is 82% faster than the 2014 RoboCup's running challenge champion [7]. It is also considerably faster than the 2017's champion [6], although the comparison is not entirely fair due to differences between SimSpark and Gazebo. To the best of our knowledge, there is currently no team in the RoboCup 3D Soccer Simulation League that can run at a similar speed. Also, in comparison with previous work [1], several metrics were improved. The average speed after the robot crosses the 0.5m line increased from 1.46m/s to 2.45m/s and the standard deviation reduced from 0.13m/s to 0.002m/s. This can be translated as greater speed, determinism and reliability, which is supported by the fact that the robot did not fall in 1000 consecutive episodes. The conditions tested in this work were harder, since the agent had no access to the original robot torso's x, y and z coordinates. However, this also made the solution more robust, since it is easier to run towards a different direction without having to align the Cartesian plane with the agent's initial orientation. Two factors that largely contributed to the optimization quality were the new initial pose introduced in section 3.5 and the longer optimization session with 200M time steps.

The stopping behavior is directly linked with the running style, and it is harder to compare with existing solutions. From the obtained results, it is possible to conclude that the agent learned to rotate its body approximately 100°, while trying to stop, to avoid falling. From a physics standpoint, this trick seems

logical. The NAO robot is more prone to tip forward than sideways, because its base is wider than it is long. Nonetheless, this is not ideal, since most times the agent is expected to end the sprint with the same orientation that it had on the beginning. We tried penalizing the orientation deviation but the quality of the solution decreased, as the robot would sometimes fall after the optimization was concluded. Despite this limitation, the proposed solution always kept control of the robot, which did not fall in 2400 consecutive episodes.

6 Conclusion and Future Work

We have successively trained a sprinting and stopping behavior that is now integrated into our team and can be used in official RoboCup matches. This solution was obtained from scratch using the Proximal Policy Optimization algorithm. The sprinting performance is highly stable. It is largely superior to previous running challenge champions and, to the best of our knowledge, it is also considerably better than current running algorithms used by RoboCup 3D Soccer Simulation League teams.

Regarding the stopping behavior, there is still room for improvement on multiple fronts. Despite its impeccable ability to stop the agent without falling, its ending orientation is, on average, very different from when the stopping algorithm takes control. Moreover, the stopping distance can also be improved.

On the future, the focus will be set on improving the stopping algorithm and expanding both behaviors to other versions of the NAO robot, with special emphasis on the version with controllable toes. We expect that the larger solution space may produce even better results.

7 Acknowledgment

The first author is supported by FCT under grant SFRH/BD/139926/2018. The work was also partially funded by COMPETE 2020 and FCT Portuguese Foundation for Science and Technology under project UID/CEC/00027/2019 (LIACC) and UID/CEC/00127/2019 (IEETA).

References

1. Abreu, M., Lau, N., Sousa, A., Reis, L. P.: Learning low level skills from scratch for humanoid robot soccer using deep reinforcement learning, In: 19th IEEE International Conference on Autonomous Robot Systems and Competitions (IEEE ICARSC'2019), Gondomar, Porto, Portugal, April 24-26 (2019)
2. Noda, I., Suzuki, S. J., Matsubara, H., Asada, M., Kitano, H.: RoboCup-97: The first robot world cup soccer games and conferences. AI magazine **19**(3), 49–49 (1998)
3. Glaser, S.: RoboCup Soccer - 3D Simulation League, https://archive.robocup.info/Soccer/Simulation/2D/binaries/RoboCup/2018/. Last accessed 19 Apr 2019
4. MacAlpine, P., Torabi, F., Pavse, B., Sigmon, J., Stone, P.: UT Austin Villa: RoboCup 2018 3D Simulation League Champions. In: RoboCup 2018: Robot Soccer World Cup XXII, pp. 77–88. Springer, Berlin, Heidelberg (2018)

5. Gazebo support for the RoboCup 3D simulator league, https://bitbucket.org/osrf/robocup3ds. Last accessed 19 Apr 2019
6. MacAlpine, P., Stone, P.: UT Austin Villa: RoboCup 2017 3D Simulation League Competition and Technical Challenges Champions. In: RoboCup 2017: Robot Soccer World Cup XXI, pp. 473–485. Springer, Japan (2017)
7. MacAlpine, P., Depinet, M., Liang, J., Stone, P.: UT Austin Villa: RoboCup 2014 3D Simulation League Competition and Technical Challenge Champions. In: RoboCup-2014: Robot Soccer World Cup XVIII, pp. 33–46. Springer, Berlin (2015)
8. Snafii, N., Abdolmaleki, A., Lau, N., Reis, L. P.: Development of an omnidirectional walk engine for soccer humanoid robots. International Journal of Advanced Robotic Systems **12**(12), 193 (2015)
9. Moradi, K., Fathian, M., Ghidary, S. S.: Omnidirectional walking using central pattern generator. International Journal of Machine Learning and Cybernetics **7**(6), 1023–1033 (2016)
10. Abdolmaleki, A., Lau, N., Reis, L. P., Peters, J., Neumann, G.: Contextual policy search for linear and nonlinear generalization of a humanoid walking controller. Journal of Intelligent & Robotic Systems **83**(3), 393–408 (2016)
11. Abdolmaleki, A., Lau, N., Reis, L. P., Peters, J., Neumann, G.: Contextual policy search for generalizing a parameterized biped walking controller. In: 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, pp. 17–22. IEEE (2015)
12. Shafii, N., Lau, N., Reis, L. P.: Learning to walk fast: Optimized hip height movement for simulated and real humanoid robots. Journal of Intelligent & Robotic Systems **80**(3), 555–571 (2015)
13. Xu, Y., Vatankhah, H.: SimSpark: An open source robot simulator developed by the RoboCup community. In: Robot Soccer World Cup, pp. 632–639. Springer (2013)
14. SoftBank Robotics: Nao the humanoid robot, https://www.softbankrobotics.com/emea/en/nao. Last accessed 19 Apr 2019
15. Akaike, H.: A new look at the statistical model identification. IEEE Transactions on Automatic Control **19**(6), 716–723 (1974)
16. Sugiura, N.: Further analysis of the data by Akaike's information criterion and the finite corrections, Communications in Statistics - Theory and Methods **7**(1), 13–26 (1978)
17. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR, vol. abs/1707.06347 (2017)
18. Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., Zhokhov, P. Zhokhov: Openai baselines, https://github.com/openai/baselines. Last accessed 20 Apr 2019
19. The MagmaOffenburg RoboCup 3D Simulation Team: magmaChallenge: Benchmark tool for RoboCup 3D soccer simulation. https://github.com/magmaOffenburg/magmaChallenge. Last accessed 19 Apr 2019
20. Adelaar, R. S.: The practical biomechanics of running. The American journal of sports medicine **14**(6), 497–500 (1986)
21. Novacheck, T. F.: The biomechanics of running. Gait & posture **7**(1), 77–95 (1998)
22. SoftBank Robotics: Aldebaran documentation: Nao – actuator & sensor list, http://doc.aldebaran.com/2-1/family/nao dcm/actuator sensor names.html