USING REINFORCEMENT LEARNING TO OPTIMIZE GAIT GENERATION PARAMETERS OF A HUMANOID ROBOT

Isaac J. Silva*, Danilo H. Perico*, Anna H. R. Costa†, Reinaldo A. C. Bianchi*

*Electrical Engineering Department
Centro Universitário FEI
São Bernardo do Campo, São Paulo, Brazil

†Escola Politécnica da Universidade de São Paulo
São Paulo, São Paulo, Brazil

Emails: isaacjesus@fei.edu.br, dperico@fei.edu.br, anna.reali@usp.br, rbianchi@fei.edu.br

Abstract— Humanoid robots use a gait pattern generator to control the servo motors during the gait preserving its dynamic balance. There are several gait generation techniques that have been developed for humanoid robots. The Darwin-OP robot uses a method to generate the gait pattern based on coupled oscillators that perform sinusoidal trajectories. However this gait pattern generation has several parameters that needs to be configured by hand. This research arises from the need to find an automatic way to adjust the values of these parameters. Therefore, this paper proposes a reinforcement learning algorithm with temporal generalization that aims to optimize the parameter values for the gait pattern generation of a humanoid robot. Experiments were performed in a simulated environment, and results showed that the algorithm was able to learn the best parameters values, through the evaluation of the humanoid robot's walk performance.

Keywords— Machine Learning for robotics, Reward structures for learning, Reinforcement Learning

1 Introduction

Researches on humanoid robots consists of a range of interests, ranging from the desire to replace humans in dangerous activities (mining, nuclear power, dangerous activities in disaster environments), and to aid in everyday activities (attendants, domestic activities). The advantage of humanoid robots is the fact that locomotion with legs is the best form of locomotion in environments with discontinuities in the floor, such as steps and stones (Westervelt et al., 2007).

The research of Marder and Bucher (2001) showed that for locomotion control in animals, there are neuronal circuits responsible for produce rhythmic motor patterns such as walking. These biological neuronal circuits are called Central Pattern Generators (CPG). Based CPG, researchers developed a gait generators for humanoid robots.

In DARwIn-OP robot (Ha, Tamura, Asama, Han and Hong, 2011) the gait pattern generation can be configured by parameters, the values of these parameters can be changed by the parameter's file, that needs to be configured by hand.

The goal of this work is optimize the parameter values of the gait pattern generation of a humanoid DARwIn-OP robot using reinforcement learning techniques, in order to seek a fast and dynamically balanced gait.

Temporal-Difference (TD) algorithm was chosen for this application, because the agent needs to learn the best parameter values only. Considering that states are composed by a set of parameters, the important thing here is to identify the best states (optimal values of the set of parameters), where the action taken is not important.

One of the most difficult problems to be solved currently in humanoid robots is the walking ability, and the origin of this research arises by the need to find an automatic way to adjust the gait generator parameter values of a humanoid robot for producing higher speeds gait with dynamically balanced gait.

The remaining of this paper is organized as follows: Section 2 presents theoretical background about Reinforcement Learning. Section 3 presents theoretical background about DARwIn-OP Gait Pattern Generation and reviews related work. Section 4 presents details of the proposed system, and Section 5 shows the experiments and results obtained. Finally, the section 6 concludes this work.

2 Reinforcement Learning and TD(λ)

In reinforcement learning, the agent learns through interaction with the environment, receiving negative or positive rewards, according to the actions taken (Mitchell, 1997).

The problem can be describe using a Markov Decision Process (MDP) (Sutton and Barto, 1998). A MDP is composed of a set of states $s \in S$, a set of actions $a \in A$ to each state s, a transition model $P(s'|s,a)$ and a reward function $R(s,a,s')$ (Russell and Norvig, 2010), where:

- S is a set finite of state;
- A is a set finite of action and $a(s)$ are the actions allowed for the state s;
- $P(s'|s,a)$ or $T(s,a,s')$ is the transition function;

- $R(s, a, s')$ is the reward get s' by performing the action a being the state s other reward $R(s')$ only the resulting state.

Some algorithms such as Q-learning (Watkins and Dayan, 1992) and SARSA (Rummery and Niranjan, 1994) require a very large number of iterations order to achieve convergence. To mitigate this amount of iterations required, one proposed acceleration method, the temporals generalization. The temporal generalization performs a spreading of learning: when the agent visits a state, the learning occurred in that state is spread to nearby states.

TD(λ) (Sutton, 1988) is an algorithm that uses eligibility traces, that is an additional memory variable associated to each state. We can see that the eligibility trace is incremented by 1 to the visited state, while all the others states are decremented by the factor $\gamma\lambda$, as described in equation 1.

$$Z_t(s) = \begin{cases} \gamma\lambda Z_{t-1}(s) & \text{if } s \neq S_t \\ \gamma\lambda Z_{t-1}(s) + 1 & \text{if } s = S_t \end{cases}, \quad (1)$$

where: λ is the factor that decays the eligibility trace ($0 \leq \lambda \leq 1$), determining how reward's future will be considered; $Z(S)$ is eligibility trace; γ is the discount factor ($0 \leq \gamma \leq 1$).

The error equation and the $V(S)$ value update is shown in equation 2 and 3, respectively.

$$\delta \leftarrow r + \gamma V(S') - V(S), \quad (2)$$

where: δ is the error; $V(S')$ is the state's future value selected by the taken action A; $V(S)$ is the state of the value that the agent is in; r is the reward.

$$V(s) \leftarrow V(s) + \alpha\delta Z(s) \quad \text{for all } s \in \text{ state space}, \quad (3)$$

where: $V(s)$ is the corresponding value to a state belonging to the state space; $Z(s)$ is the eligibility trace referring to a state belonging to the state space; α is the learning rate ($0 < \alpha < 1$).

3 Gait Pattern Generator

Locomotion control in vertebrate and invertebrate animals is made by neuronal circuits responsible for produce rhythmic motor patterns such as walking (Marder and Bucher, 2001). These biological neuronal circuits are called Central Pattern Generators (CPG). Based on the biological CPG, researchers developed gait generators for humanoid robots, working as a generic CPG model known as gait pattern generation. The most important task of this gait pattern generation is to preserve its dynamic balance during the walking (Vukobratović and Borovac, 2004).

Recent work have been using machine learning to improve robot performance. The work of MacAlpine et al. (2012) presents a learning algorithm to optimize the gait using an Aldebaran NAO humanoid robot in 3D simulation environment SimSpark (*Simulador SimSpark*, 2015). The authors describe that in order to achieve optimization of the parameter values, they used the algorithm Covariance Matrix Adaptation Evolution Strategy (CMA-ES). The CMA-ES uses a population generation approach similar to a genetic algorithm.

NAO robot has 40 parameters that can be optimized, but the authors selected 14 parameters to optimize and keep the other ones with fixed values, these 14 parameters are those that generate the greatest influence on speed and dynamically balanced gait. The authors reported that the simulation was performed in a cluster, and, because of the parallel learning, the training was about 150 times faster (MacAlpine et al., 2012).

Another work worth citing was conducted by Shafii et al. (2015) in the 3D simulation environment, using as main focus the effect of hip height movement on the walking speed of the robot, in a way that, the robot with varied height could walk faster than the robot with fixed hip height. The tests were made not only to obtain fast forward walk but also to obtain a fast side walk using CMA-ES. The parameters learned were tested on real NAO robots and on simulated ones.

Darwin-OP robot uses a gait pattern generation based in coupled oscillators. The oscillators perform sinusoidal trajectory of movement synchronized in time, so the robot is capable to perform a dynamically balanced gait. Ha, Tamura and Asama (2011) developed the gait pattern generation for the Darwin-OP robot and also developed a closed loop control able to protect the robot from a possible fall, which is done by reading the gyro data. They developed three oscillators, one in each foot OSC_{move} (movement oscillator), and one in the center of mass OSC_{bal} (balance oscillator). Each oscillator have six sub-oscillators related to the axes X, Y, Z, α, β e γ, summing up 18 sub-oscillators (Ha, Tamura and Asama, 2011).

The oscillators can be configured by parameters, that can be divided into three groups, as showed in the Table 1.

Parameters containing the word *offset* in its name do not require dynamic test. These values are statically configured with the robot in the upright position, therefore, in this work these parameters were setting with default values.

The Feedback parameters are related to the gain on the gyro feedback, so in this work these parameters were setting with default values.

Table 1: Parameter values of the gait pattern generation.

Offset's parameter	Parameters of the oscillators	Feedback parameters
X_offset	*period_time*	*balance_knee_gain*
Y_offset	*dsp_ratio*	*balance_ankle_pitch_gain*
Z_offset	*step_forward_back_ratio*	*balance_hip_roll_gain*
roll_offset	*foot_height*	*balance_ankle_roll_gain*
pitch_offset	*swing_right_left*	
yaw_offset	*swing_top_down*	
hip_pitch_offset	*arm_swing_gain*	
pelvis_offset		

4 Using reinforcement learning to optimize gait pattern generation parameters

This paper proposes a reinforcement learning algorithm with temporal generalization that aims to optimize the parameter values of the gait pattern generation for a DARwIn-OP robot. To define the reinforcement learning environment, each parameter p must be discretized within a range (this range is determined according to the characteristic of each parameter). Therefore, a study was conducted to determine the minimum and maximum value for each parameter, and also the discretization step size of each parameter, in order to reduce the state space in a way that is not prejudicial to the expected result.

The MDP used in this work is composed of:

- S: the states s are a set of parameters p_n, therefore, $s(p_1, p_2, ..., p_n) \in S$;
- A: the actions $a(s)$: increment a parameter value, decrement a parameter value, and stay on the same state;
- $T(s, a, s')$: the transition function is a deterministic function that will allow the increment or decrement of only one parameter p_i ($1 \leq i \leq n$) of $s(p_1, p_2, ..., p_n) \in S$, and it also allows the system to stay in the same state;
- r: a negative reward is given to the robot when it falls, and a positive reward is determined by a function that has as input the speed of the gait ($positive\_reward = f(speed)$).

For example, if someone creates a state containing a set of three parameters $s(p_1, p_2, p_3) \in S$, there will be 3 actions of increment of parameter, 3 actions of decrement of parameter and one action to stay on the same state, summing up 7 actions $a(s) \in A$, where actions a_1, a_2 e a_3 are increasing actions; a_4, a_5 e a_6 are actions of decrement; and the action a_7 is to stay on the same state.

The algorithm to optimize the parameter values using the TD(λ) method, is shown in algorithm 1. The episode ends when the robot walks a certain number of footsteps, W, without falling or if it falls a certain amount of times. Before starting an episode, the state $s \in S$ is randomly chosen, allowing the robot to learn all the state space.

```
Initialize all $V(s)$ arbitrarily
while The desired number of episodes is not reached do
    Choose an initial state $s$.
    for for all $s$ do
        $Z(s) \leftarrow 0$
    end
    while Desired number of steps without falling is not reached or the maximum number of falls is not reached do
        Update the parameters $p_n$ w.r.t. the chosen state $s$.
        $X_i$ and $Y_i$ receive the current global position.
        Performs $W$ footsteps.
        $X_f$ and $Y_f$ receive the current global position.
        Calculate the speed of the gait.
        if robot fall then
            $r \leftarrow negative\_reward$
        end
        else
            $r \leftarrow positive\_reward$
        end
        $a \leftarrow$ action of the state $s$ using policy derived from $\epsilon - greedy$
        Take action $a$ observe the state $s'$
        $\delta \leftarrow r + \gamma V(s') - V(s)$
        $Z(s) \leftarrow Z(s) + 1$
        for $\forall s \in S$ do
            $V(s) \leftarrow V(s) + \alpha\delta Z(s)$
            $Z(s) \leftarrow \gamma\lambda Z(s)$
        end
        Update the state $s$:
        $s \leftarrow s'$
    end
end
```

Algorithm 1: Algorithm TD(λ) for parameter learning.

Table 2: Configuration variables for reinforcement learning

Maximum number of steps per episode	40
Number of footstep (W)	4
Learning rate (α)	0.10
Exploration rate (ϵ)	0.15
λ	0.05
γ	0.90

An exponential function was adopted to separate the parameter values that generate low speeds of gait.

$$r = \begin{cases} negative\_reward & \text{if robot fall} \\ f(speed) = K_1 10^{(\frac{speed}{K_2})} & \text{otherwise} \end{cases} \quad (4)$$

Where K_1 and K_2 are constants used to influence the reinforcement given. K_1 is a linear constant gain and K_2 is an exponential constant gain. Experimentally K_1 was defined as 0.33 and $K_2 = 10$.

5 Experiments

This section presents four experiments: The first experiment learns the best parameters value of $period\_time$ and $swing\_right\_left$. The second experiment checks the robot gait behavior using the parameter values learned from first experiment. The third experiment learns the best parameters value of $period\_time$ and $swing\_right\_left$ with the footstep length increased. The fourth experiment checks the robot gait behavior using the parameter values learned in the third experiment.

Two gait generator parameters were chosen to be optimized during the development of this work (*period_time* and *swing_right_left*), in order to have a small state space. The other parameters were set to default values. The learning variables were set with the same values that had already been used in previous works (Martins, 2007), as shown in Table 2.

The selection of actions is made using the $\epsilon - greedy$ rule as described in Sutton (Sutton and Barto, 1998). Line 7 of the algorithm 1, contains the condition for the end of the episode. In this experiment, the maximum number of falls was set to 10 and the maximum number of steps per episode is presented in Table 2.

Experiments used in Equation 4 as rewards. In algorithm, the negative reward value has relationship to the value of positive reward and the maximum number of steps per episode. $negative\_reward = -400$ was used for the maximum number of steps per episode equal to 40.

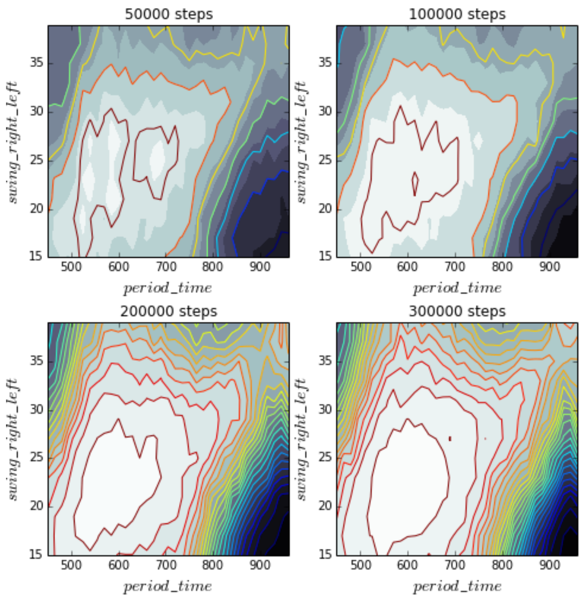

Figure 1: Experiment 1: Graphs of the values of $V(s)$ after 50.000; 100.000; 200.000; 300.000 steps.

The parameters that have been optimized in this experiment are *period_time* and *swing_right_left*. The *period_time* was discretized into 35 values in a range of 450 to 960 with step 15. The *swing_right_left* was discretized into 13 values in a range of 15 to 39 with step 2. Therefore, the total number of states was 455 (=$13 \cdot 35$).

The graphs to represent $V(s)$ values (Figures 1, 3, 4, 7) use a color pattern, where the lighter color is the region that has the highest value.

All the experiments were performed in an Intel i5-4690 3.5GHz computer, comprised of 8GB of RAM, 120GB of SSD, NVIDIA GeForce GTX660 2GB DDR5, running Linux Ubuntu 14.04. During all experiments the algorithm proposed was implemented in the simulator Webots 8.0.3, using C++ programming language. Webots is a 3D simulation environment used to model, program and simulate mobile robots.

5.1 *First experiment: optimizing parameters values.*

The aim here is to learn new parameters values that makes the robot move with higher speed. In this experiment the robot performs the walking on the simulator. The graphs depicted in Figure 1 show the $V(s)$ values for all states. The analysis of these graphs allow us to understand that the algorithm was able to find the parameter values responsible for producing higher speeds and the best dynamically balanced gait (the areas of lighter color have the highest values of $V(s)$).

It is also possible to observe in the graphs (Figure 1) that only 100.000 steps has the region of the best values, which is approximately

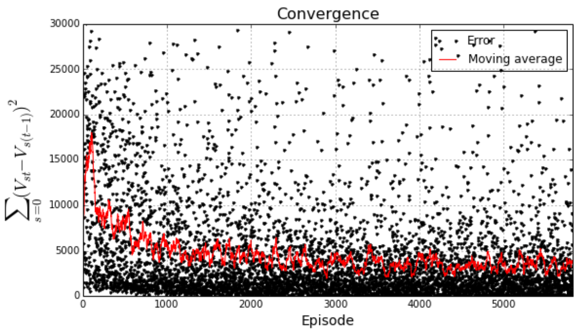

Figure 2: Experiment 1: Convergence graph.

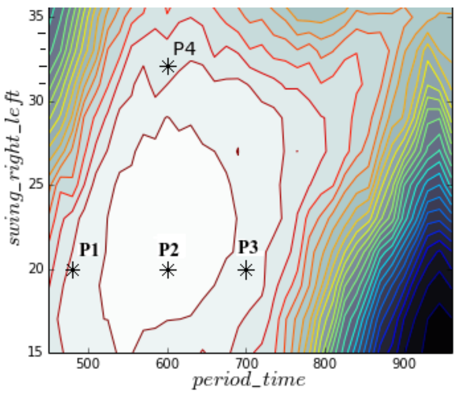

Figure 3: Experiment 2: Graph of values parameters learned.

the same region found in 300.000 steps. Focusing on the central area of the best values region of these graphs, the values are approximately *period_time* $\approx$ 600 and *swing_right_left* $\approx$ 22. In Webots simulator the default values of these two parameters are *period_time* = 600 and *swing_right_left* = 20. Therefore, the execution of the proposed algorithm could find values of *period_time* and *swing_right_left* that were approximately the same default values of the simulator.

The graph of Figure 2 shows the sum of the difference between the current value of $V_t(s)$ to the previous value of $V_{t-1}(s)$ per episode for all states s. The moving average used in these two graphs was 50 episodes.

5.2 Second experiment: Testing the robot with the optimized parameters

This experiment is made to check the robot gait behavior in the simulator, using the parameter values learned by the proposed learning algorithm. To perform this experiment we first set up the robot with the values of *period_time* and *swing_right_left* related to point P_n chosen. For each point the robot performs a gait during a certain time interval. This time interval should be the same for all points P_n to be tested.

Table 3 shows a simulation performed with some parameter values to compare the speed

Table 3: Performance of the parameter values

	P1	**P2**	**P3**	**P4**
period_time	480	600	700	600
swing_right_left	20	20	20	32
Travelled distance [m]	49.0	54.6	42.2	51.61
Mean speed [cm/s]	17.7	19.7	15.2	18.61

and the stability. In Figure 3, the values *period_time* = 600 and *swing_right_left* = 20 related to point P2 that is inside the region of the best values. Values of the points P1, P3 and P4 are outside the region of the best values.

Point P4 showed a speed slower when compared to P2, and a speed higher than that of P1 and P3 points, but the algorithm leaves point P4 out of the best values region because during the gait there was a fall of the robot. Points P1, P2 and P3 performed the gait without falling during this experiment.

5.3 Third experiment: Optimizing parameters values to increase the speed gait

Here we aim is to learn new parameters values that makes the robot move with higher speed with the robot's footstep length increased (footstep length is the maximum distance between the two legs at the instant one leg is ahead of the other during the gait). The method *setXAmplitude* (*setXAmplitude* is the forward footstep length) was used to increase the footstep length. However, this method is delimited in the range of -1 to 1, then, to perform this experiment we needed increase the range for -1.5 to 1.5. In previous experiments, the value used in the method *setXAmplitude* was 1.

In this experiment, the value used in the method *setXAmplitude* was 1.5. Table 2 shows the configuration variables of the reinforcement learning algorithm.

It can be seen in the graph of Figure 7 that there was a shift in the region of the best values of the parameters *period_time* and *swing_right_left*. However, there are two small areas of the graphic that stand out with a lighter color (because these two small regions have the highest values of $V(s)$). Besides the shift of the region of the best values, it can be seen that there was a decrease in the size of the area of the region (the two small regions have the largest values of $V(s)$), so the best region decreased when compared to the $V(s)$ graph of Figure 3.

The graph of the Figure 4 shows two graphs with the maximum values for each value of *period_time* and *swing_right_left*, Figure 5 also shows the amounts of $V(s)$ for each value of *period_time* and *swing_right_left*; However, in Figure 4 we can see more precisely where is the region with the highest values of $V(s)$.

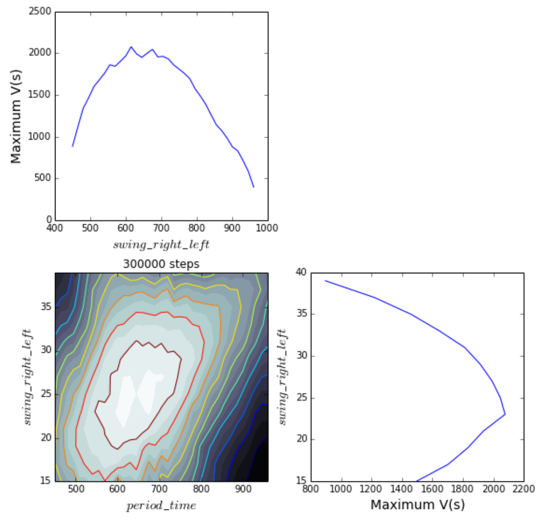

Figure 4: Experiment 3: Graph with the maximum values of $V(s)$.

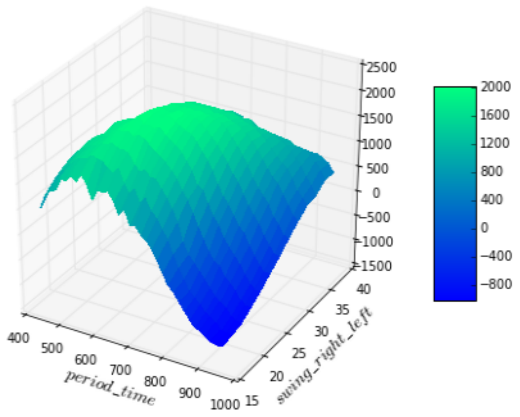

Figure 5: The graph in three dimensions $V(s)$.

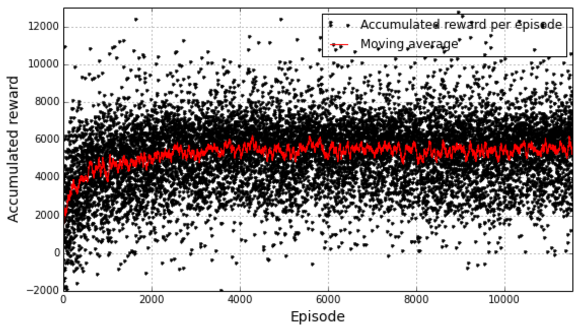

Figure 6: Experiment 3: Graph of reward accumulated per episode.

In Figure 6 the algorithm executed approximately 540,000 steps and we can see the accumulated reward per episode.

5.4 *Fourth experiment: Testing the robot with the optimized parameters*

To perform this experiment, the robot was first setup with the values of *period_time* and *swing_right_left* related to point P_n chosen. For each point, one type of gait is performed with the robot during a certain period of time. This interval should be equal for all points P_n to be tested. During this gait, it's checked if the robot falls, and every four steps it was calculated the speed of the gait; when the experiment was concluded, the average speed was calculated.

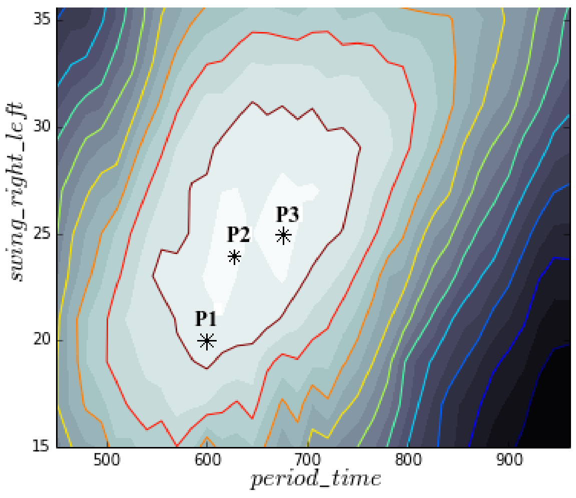

Figure 7: Experiment 4: Graph of values parameters learned.

Table 4: Performance of the parameter values

	P1	**P2**	**P3**
period_time	600	625	675
swing_right_left	20	24	25
Travelled distance [m]	17.4(fall)	17.9	16.8
Travelled distance [m]	101.2(fall)	102.2	99.0
Travelled distance [m]	110.4(fall)	113.0	108.9
Travelled distance [m]	124.1(fall)	127.5	123.4
Mean speed [cm/s]	28.4	28.8	27.8

Table 4 shows the results when using the parameters values that the algorithm found as the best values (parameter values of $period\_time = 625$ and $swing\_right\_left = 24$ for the P2 point, and period $period\_time = 675$ and $swing\_right\_left = 25$ for the P3 point). Table 4 shows that the values of P1 ($period\_time = 600$ and $swing\_right\_left = 20$) showed a good speed gait, however in some times the robot falls. Therefore, the speed is high but the robot does not have a dynamically balanced gait.

Observing the points position in Figure 7 taken from Table 4 and used in the experiment, one can see that all 3 points are located in a region considered as having good values. However, P1 does not have as good dynamically balanced gait as the dynamically gait shown by P2 and P3.

Figure 8 shows the traveled path by the robots. In graph, we can observe that the parameter values of the robot 1 (P1) makes the robot performs a walk with slip.

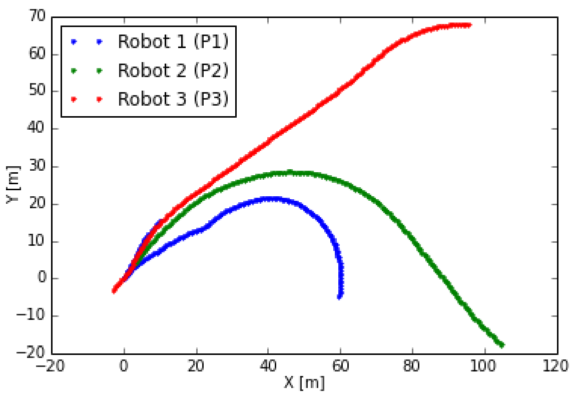

Figure 8: Traveled path by the robots in the simulator.

6 Conclusions

This paper showed a reinforcement learning algorithm that has been able to optimize the gait generator parameter values of a humanoid robot in a simulated environment, was also possible to identify the relationship between the parameters.

In the first experiment it is possible to conclude that the parameter values of *period_time* and *swing_right_left* used by Webots simulator are the best values since they combine speed with dynamically balanced gait. In the second experiment we concluded that it is possible to increase the speed by changing the values parameters, however, increase the footstep length result in a difficult to perform a dynamically balanced gait.

The deep reinforcement learning may also be implemented to decrease the learning time, making it possible to increase the number of parameters to be learned. In Webots simulator we can include more than one robot in the simulation environment. In such configuration it will be possible to have the agents sharing the knowledge to achieve their goals.

Acknowledgments

The authors acknowledge the Centro Universitário FEI and São Paulo Research Foundation (FAPESP Grant 2016/21047-3) for supporting this project. The authors acknowledge the support of NVIDIA Corporation with the donation of the Jetson TX1 developer kit. The authors would also like to thank the scholarships provided by CAPES and CNPq.

References

Ha, I., Tamura, Y. and Asama, H. (2011). Gait pattern generation and stabilization for humanoid robot based on coupled oscillators, *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, IEEE, pp. 3207–3212.

Ha, I., Tamura, Y., Asama, H., Han, J. and Hong, D. W. (2011). Development of Open Humanoid Platform DARwIn-OP, *Proceedings of SICE Annual Conference (SICE)*, IEEE, pp. 2178–2181.

MacAlpine, P., Barrett, S., Urieli, D., Vu, V. and Stone, P. (2012). Design and optimization of an omnidirectional humanoid walk: A winning approach at the robocup 2011 3d simulation competition., *AAAI*.

Marder, E. and Bucher, D. (2001). Central pattern generators and the control of rhythmic movements, *Current biology* **11**(23): R986–R996.

Martins, M. F. (2007). *Aprendizado por reforço acelerado por heurísticas aplicado ao domínio do futebol de robôs*, Master's thesis, Centro Universitário FEI.

Mitchell, T. M. (1997). *Machine Learning*, 1 edn, McGraw-Hill, Inc., New York, NY, USA.

Rummery, G. A. and Niranjan, M. (1994). *Online Q-learning using connectionist systems*, University of Cambridge, Department of Engineering.

Russell, S. J. and Norvig, P. (2010). *Artificial intelligence: a modern approach*, 3 edn, Prentice Hall, Upper Saddle River, NJ.

Shafii, N., Lau, N. and Reis, L. P. (2015). Learning to walk fast: Optimized hip height movement for simulated and real humanoid robots, *Journal of Intelligent & Robotic Systems* **80**(3-4): 555–571.

Simulador SimSpark (2015).
URL: *http://simspark.sourceforge.net/*

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences, *Machine learning* **3**(1): 9–44.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning*, 1st edn, MIT Press, Cambridge, MA, USA.

Vukobratović, M. and Borovac, B. (2004). Zero-moment point—thirty five years of its life, *International Journal of Humanoid Robotics* **1**(01): 157–173.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning, *Machine learning* **8**(3-4): 279–292.

Westervelt, E. R., Grizzle, J. W., Chevallereau, C., Choi, J. H. and Morris, B. (2007). *Feedback control of dynamic bipedal robot locomotion*, Control and automation, CRC Press, Boca Raton.