Deep Reinforcement Learning for Navigation of Mobile Robots

Junior Costa de Jesus
Federal University of Santa Maria
Santa Maria, Rio Grande do Sul
Email: dranaju@gmail.com

Marco Antonio de Souza Leite Cuadros
Federal Institute of Espirito Santo
Serra, Espirito Santo
Email: marcoantonio@ifes.edu.br

Jair Augusto Bottega
Federal University of Santa Maria
Santa Maria, Rio Grande do Sul
Email: jairaugustobottega@gmail.com

Daniel Fernando Tello Gamarra
Processing Department of Electricity
Federal University of Santa Maria
Santa Maria, Rio Grande do Sul
Email: daniel.gamarra@ufsm.br

***Abstract*—This paper presents a study of a deep reinforcement learning technique that uses a Deep Deterministic Policy Gradients network for application in navigation of mobile robots. In order for the robot to arrive to a target on a map, the network has 10 laser range findings, the previous linear and angular velocity, and relative position and angle of the mobile robot to the target as inputs. As outputs, the network has the linear and angular velocity. From the results analysis, it is possible to conclude that the deep reinforcement learnings algorithms, with continuous actions, are effective for decision-make of a robotic vehicle. However, it is necessary to create a good reward system for the intelligent agent to accomplish your objectives. This research uses different virtual simulation environments provided by ROBOTIS in the robot simulation software Gazebo in order to test the performance of the algorithm.**

A supplementary video can be accessed at the following link: https://youtu.be/NhGxEC3g7sU. That shows the performance of the proposed system.

***Index Terms*—Deep Deterministic Policy Gradients, Deep Reinforcement Learning, Robots Navigation.**

I. Introduction

Deep Reinforcement Learning (Deep-RL) is starting to achieve interesting results in different areas such as tasks involving the control on discrete systems [1], [2] continuous systems [3], [4], and more recently in robotics [5], [6]. The first applications of deep reinforcement learning in robotics were in the use of manipulation in a fully observable and stable environment [7], but tasks in mobile robotics involving obstacles interacting with physical environments and objects, turns the workplace more complex. In order to overcome this problem, Deep-RL methods normally try to discretize the actions to turn simpler the problem [8], [9]. Recent articles explore continuous control actions used for navigation of mobile robots with good results [10], [11].

In this paper, we try to demonstrate how effective can be the Deep-RL used on simulated environments. For that, three environments were used on Gazebo, which can provide us with a lot of resources for robot simulation, for example we can create an environment and insert a model of a real mobile robot [12], [13]. The mobile robot used on the simulation was the Turtlebot3.

The objective of this research is to show the efficiency of a Deep-RL network in the task of mobile robot navigation from an initial position to a target on an environment. To simplify this problem it was created a network which has 14 inputs and 2 outputs, as shown in Fig. 1. The 14 inputs are composed by 10 readings of the laser sensor, the previous linear and angular velocity, the distance and angle of the mobile robot related to the target. And the outputs of the network are the linear and angular velocity that are sent to the robot in order to get to the target. It is expected the intelligent agent won't collide with any obstacle on its trajectory to the target.

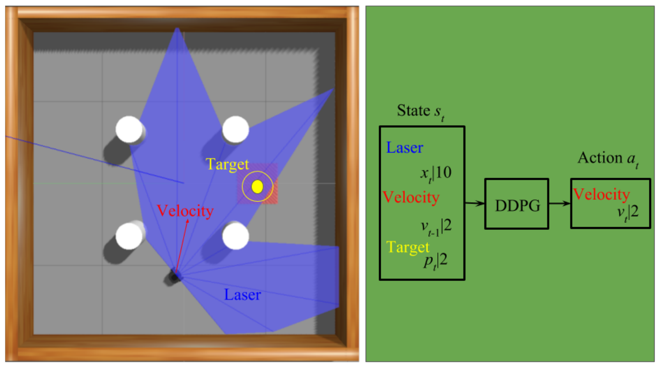


Fig. 1. System definition.

This work is divided in seven sections. After a brief introduction in the first section of the Deep-RL on the mobile robot navigation, the second section describes the work of authors on the field that inspired this research, the third section gives a background about the technique used, the fourth section makes an introduction of the tools used for the project, the fifth section summarizes the methods so that the robot can get to a target, the sixth section presents the results obtained on the Gazebo environments, the seventh section makes the discussion on the results and applications of Deep-RL.

II. Related Works

Deep-RL has been previously applied on robotic tasks, among this applications we can refer to the work of [14]. Which is a survey of different works of this techniques applied to robotics. Mnih *et al.* [1] utilized a convolution neural network to estimate a value function for future rewards on the Atari games, this strategy was called deep Q-network (DQN) [1], [15], [16]. The DQN can only be used in a task with a discrete action. To extend it to a continuous control, Lillicrap *et al.* [3] proposed a deep deterministic policy gradients (DDPG). That became the basement for the application of Deep-RL in mobile robot navigation. Tai *et al.* [10] created a mapless motion planner for a mobile robot by taking the sparse 10-dimensional range findings and the target position with respect to the mobile robot coordinate frame as inputs and the continuous steering commands as output, but firstly being proposed as discrete steering commands on [9]. It was shown that, with the asynchronous Deep-RL method, a mapless motion planner can be trained and complete the task to get to a determined target.

Zhu *et al.* [8] proposed another model to apply the Deep-RL to the task of driving a mobile robot. The model created took the current observation of states and the image of the target as input and generated an action in a 3D environment as the output.

An activity that can be very challenging in robotics is to navigate a vehicle safely and efficiently in pedestrian-rich environments. Chen *et al.* [11] elaborated a Deep-RL model that can quantify what *to* do and *not to* do on the precise mechanism of human navigation. This work develops a time-efficient navigation policy that respects common social norms. Creating a method able to control a mobile robot moving at human walking speed in an environment with many pedestrians.

III. Theoretical Background

A. Deep Reinforcement Learning

The goal in deep reinforcement learning is to control an agent attempting to maximize a reward function. The deep Q-network (DQN) algorithm [1] was capable of human level performance on many Atari video games by estimating the actions of an agent. However, while DQN could solve problems on complex observation spaces, it only can handle discrete action spaces. It is noticeable that many tasks, on the robotic control, have continuous action spaces. So DQN cannot be applied to continuous domains and it is necessary to use another algorithm that can handle this type of problems.

The deep deterministic policy gradients (DDPG) algorithm consists of an actor-critic method that uses approximation functions that can learn continuous action space policies. The algorithm makes use of a neural network for the actor network and other for the critic network. These two networks computes the action prediction for the current state and generates a temporal-difference error signal for each time step. The input of the actor network is the current state, and the output is a real value representing an action chosen for a continuous action space. The output of the critic is simply the estimated Q-value of the current state and the action given by the actor.

The biggest challenge of learning in continuous action space is exploration. To confront this challenge is needed to construct a exploration policy μ' by adding a noise sampled from a noise process $\mathcal{N}$ to the actor network policy, defined as:

$$\mu' = \mu(s_t) + \mathcal{N} \tag{1}$$

where $\mathcal{N}$ can be chosen in a way to suit the environment. Being the Ornstein-Uhlenbeck process [17] the most used to generate temporally correlated exploration efficiency in physical control problems.

In general, to train and evaluate a policy function coming out of the actor network and a value function coming out of the critic network, with thousands of simulated trajectories temporally correlated, leads to the introduction of huge amounts of variance in the approximation of a true Q-function. It is suggested to use a replay memory to store the experiences of the agent during training [2]. This means, saving the states, actions, rewards and new states that the agent explored during the episode. And then, randomly sampling experiences to use for learning in order to break the temporal correlations within different training episodes.

The replay of experiences allows the intelligent agent to learn from recent memories, increasing the learning speed and breaking undesirable temporal correlations. Even with a short memory it is possible to see a substantial improvement in the agent performance. Despite the application of a replay memory slows the agent learning, a better performance is achieved.

Without counting the replay memory used, it is needed to make a target network to generate targets for the temporal-difference error that can regulate the learning and improve stability. The target network contains a copy of the actor and critic, however, with "soft" updates. This means that the target network do not copy directly the weights of the actor and critic network. The weights, represented by θ, of the target network are then updated by:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta \tag{2}$$

with $\tau \ll 1$. Where the target is represented by the apostrophe. So the target values of the network change slowly and allows an improvement in the stability of the learning.

IV. Experimental Setup

For the analysis of a DDPG network was used the programming language Python [18]. The Python language has as priority the legibility of the code under speed. The vast library and frameworks provided by Python makes it an exquisite tool for machine learning and data analysis purposes.

A. ROS

The robot operations system (ROS) is a flexible framework to write software for robots. ROS [19] is a collection of tools, and libraries. ROS provides operational system's standard services, like hardware's abstraction, device low level control, messages between processes and package management. The set of ROS processes in execution are represented by graphs architecture where the processing is performed on nodes that receive and send messages as sensors, control, state, planning, actuator and others.

Despite the importance of low latency on the robots control, ROS is not a real-time operational system, although it is possible to integrate ROS with real-time code. This lack of real-time system is being addressed on the development of ROS 2.0.

B. *TurtleBot*

TurtleBot is a ROS standard platform robot, and there are 3 version of the series. TurtleBots are affordable and programmable mobile robots for use in education, research, hobby, and product prototyping. The third version was used on this project, shown in Fig. 2.

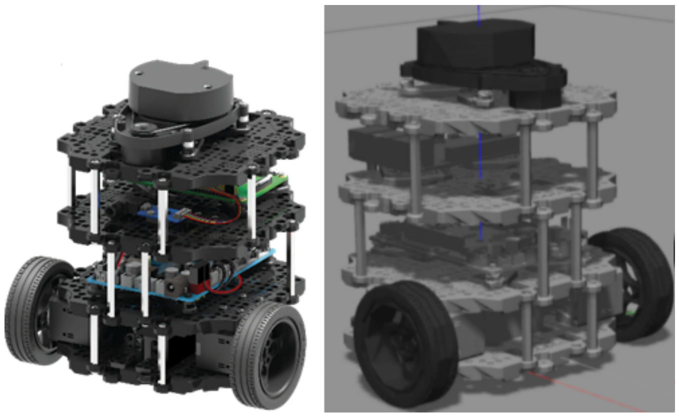


Fig. 2. Real and simulated TurtleBot3 Burger in Gazebo.

The TurtleBot3 Burger uses 2 DYNAMIXEL motord series XL, for the object detection the TurtleBot3 utilize a 360 degree sensor laser LiDAR, and it has an IMU sensor for the odometry calculations. All the control is made by the open source controller board OpenCR1.0 and Raspberry Pi 3 microprocessor.

C. *Gazebo*

Robot simulation is an essential tool on all roboticist's toolbox. A good simulator makes possible to test algorithms quickly, to design robots, and to train systems with artificial intelligence using realistic scenarios. With Gazebo [12] is possible to simulate this environments easily and with the advantage of having an active community. This makes Gazebo a great tool on the area of robotic simulation.

V. Methodology

The intention of this work is to create a mobile robot that can plan its movements without any map knowledge on the environment. Its translation function is defined as:

$$v_t = f(x_t, p_t, v_{t-1}) \tag{3}$$

where x_t is the observation from the raw sensor information, p_t is the relative position of the target, and v_{t-1} is the velocity of the mobile robot in the last time step. All variables specified, previously, can be defined as the current state s_t of the mobile robot. With this model it is possible to get the actions that the robot will make, given its current state. However, it is needed to ensure a minimum reading frequency of the input data to control the movement of the robot because if the robot get a slow read frequency of the inputs it cannot react to an obstacle in the trajectory to a target. In this way, the robot can react to new states quickly. This methods was first explored by Tai *et al.* [10].

A. *Network Structure*

Once the system of state and actions has been defined, it is possible to create a DDPG network capable of resolving the problem. The DDPG network has 14 inputs as presented in Fig. 3, in which 10 corresponds to the laser range findings, 2 corresponds to the linear and angular velocity, and the other 2 corresponds to the relative position and angle of the mobile robot to the target. The sample of the laser range findings are between -90 and 90 degrees in relation to the robot. The output of the network is the action of linear and angular velocity that will be applied on the mobile robot.

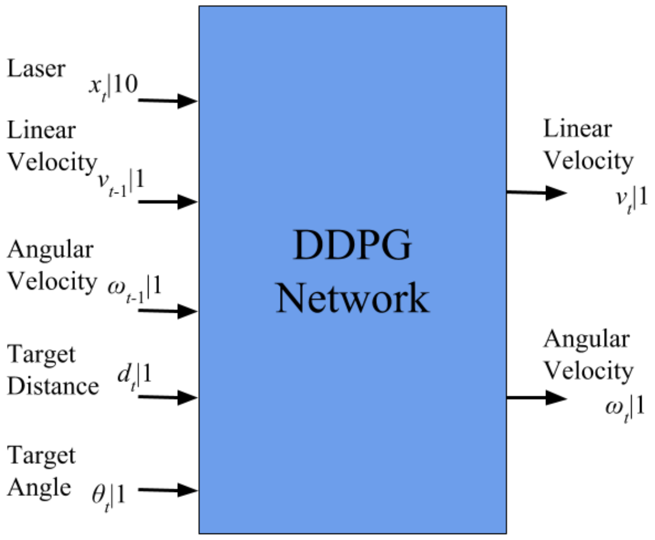

Laser $x_t|10$
Linear Velocity $v_{t-1}|1$
Angular Velocity $\omega_{t-1}|1$
Target Distance $d_t|1$
Target Angle $\theta_t|1$
DDPG Network
Linear Velocity $v_t|1$
Angular Velocity $\omega_t|1$

Fig. 3. DDPG network inputs and outputs.

The network structure of the DDPG is shown in Fig 4. The actor-network has as input the current state of the mobile robot followed by 3 fully-connected neural networks layers with 512 nodes. This input of the networks is transformed on the linear and angular velocity that will be the commands sent to the motor of the mobile robot. The angular velocity range is constrained between $(-1, 1)$ and the hyperbolic tangent function $(tanh)$ is used as activation function. For the linear angular range, that it is constrained between $(0, 1)$, the sigmoid function was used. As there is no laser readings in the back of robot, the backward move is not necessary. The output actions are then multiplied with two hyperparameters to decide the final linear and angular velocity executed by the mobile robot. For this, it was used as maximum linear velocity 0.22 m/s and maximum angular velocity 1 rad/s on the TurtleBot3 robot version Burger.

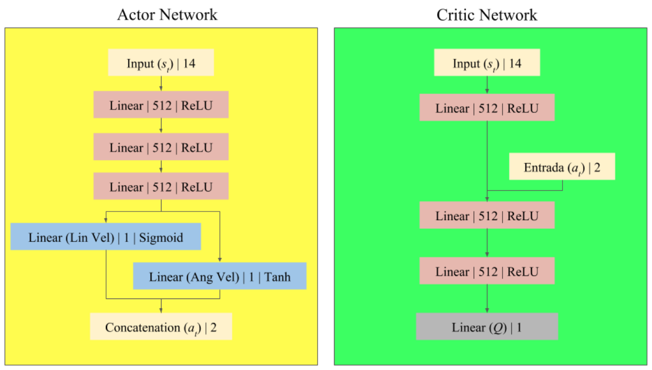


Fig. 4. DDPG network structure model.

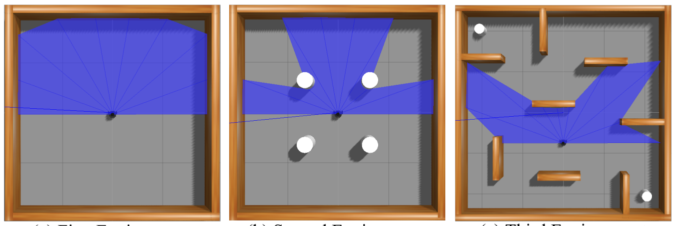
In the critic-network, the Q-value of the current state and action are predicted. Using only 3 fully-connected neural networks layers to process the input state. Where the action, output of the actor-networks, is concatenated on the second neural network layer. The Q-value is activated through a linear activation function:

$$y = kx + b \tag{4}$$

where x is the input of the last layer, y is the predicted Q-value, and k and b are the trained weights and bias of this layer, respectively.

B. Simulation Environments

There were used three environments for the simulations. The first environment is shown in Fig. 5(a), the environment represents a free area for the robot to move. The walls of this environment are the only things where the robot can collide. If the mobile robot collide with the wall or any obstacle, a negative reward is given for this action and the current episode stops.



(a) First Environment (b) Second Environment (c) Third Environment

Fig. 5. Training environments used on Gazebo simulation.

The second environment is shown in Fig. 5(b) and has 4 fixed obstacles. It means that this environment is more complex in a way that the intelligent agent have to make a better strategy to not collide.

The third environment, shown in Fig. 5(c), is more complex than the previous environments. The number of walls and the mobile obstacles, represented by the white blocks, makes the environment more dynamical, approaching it to a real-world environment.

C. Reward Function

Once the environment have been defined it is possible to simulate a controlled mobile robot for a navigation task. Now it is necessary to define the reward and penalty system to the Deep-RL network. Remembering that the rewards and penalties are attributed numbers passed to the intelligent agent. So, the network will make a feedforward and backpropation step in order to learn the hyperparameters.

There are four different conditions for the reward system that presented better results for the problem's resolution and are the following:

$$r(s_t, a_t) = \begin{cases} r_{arrive} \text{ if } d_t < c_d \\ r_{collide} \text{ if } min_x < c_o \\ c_{r1}(d_{t-1} - d_t) \text{ if } (d_{t-1} - d_t) > 0 \\ c_{r2} \text{ if } (d_{t-1} - d_t) \leq 0 \end{cases} \tag{5}$$

If the robot gets to the target through threshold checking c_d, a positive reward (r_{arrive}) is given, but if the robot collides with an obstacle through a minimum range readings checking, a negative reward $(r_{collide})$ is given. Both conditions are sufficient to end the training episode. Otherwise, the reward is based on the distance difference from the target compared to the last time step $(d_{t-1} - d_t)$. If this difference is positive the reward given is the distance traveled multiplied by the hyperparameter (c_{r1}), and if the distance is negative is used the hyperparameter (c_{r2}). This motivates the mobile robot to get closer to the target position and encourages it to avoid the obstacles in the environment.

VI. Results

This section deals with the presentation and discussion of results. The collected data are related to the obtained reward from the artificial neural network of the project. With these data is possible to see what is the learning degree of the agent in the environment because the gained reward is intrinsically attached to the performance of the agent on the environment that it must go. All the training environments were arranged by ROBOTIS, however, some alterations were done on the source code of the Gazebo simulation in order for the simulated mobile robot to use the reward system defined on this paper.

A mobile robot with a DDPG network was trained in order to make the experiments in the environments presented on Fig. 5, where the goal of the mobile robot is to get to the target. For the first test was used the first environment that shows, in Fig. 6, a sequence of frames of the robot and the target. We can observe how the robot start in an initial position distant from the target, navigating in order to reach the target.

The training results of the reward function of the first environment are shown in Fig. 7. On the first episodes can be

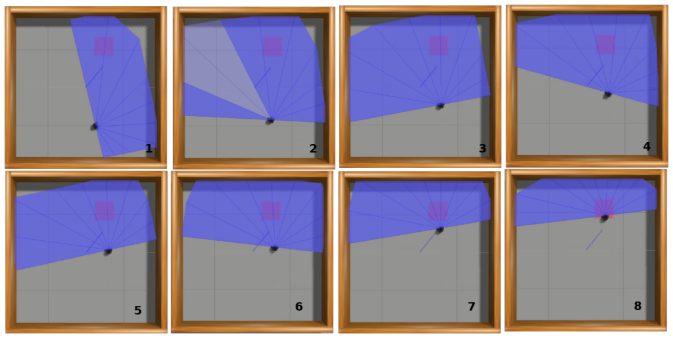


Fig. 6. Images sequence in the first environment.

noticed a negative reward, this happens because the algorithm started and it was still learning. This reward by episode means that the robot is trying to maximize the reward to complete the task. It is observed in the episode 400 a fall. This is a probable correction on some maximized parameters of the network and it resulted in a low reward. Results like this means that the network has a continuous learning trying to correct itself.

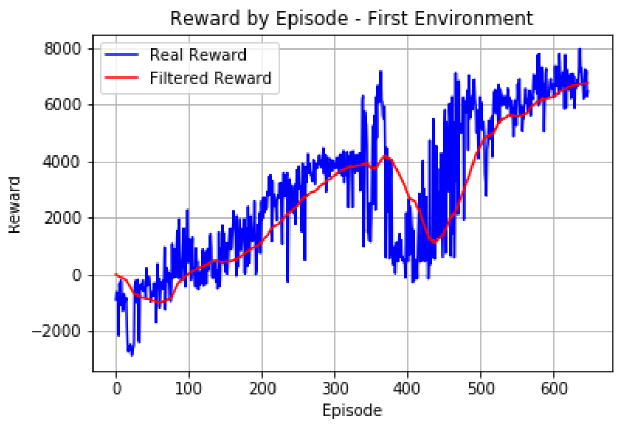


Fig. 7. Rewards of the first environment.

In Fig. 7 the x axis represents the past episodes on the simulation, an episode is defined when the mobile robot arrives to the target in the map or collides with some obstacle. The y axis in Fig. 7 represents the total value of the reward that the robot received on the episode. The reward, with the blue color, has a great variance, it was decided to use a moving average filter for better visualization of the results.

After the mobile robot has been trained on the first environment, the experiment was done in the second environment and tested. It is shown in Fig. 8 a sequence of the actions made by the TurtleBot from an initial position until it could arrive to the target after the training episodes.

The reward function results of the training process for the second environment are shown in Fig. 9. Comparing this results with the last environment, we can observe that it needed more episodes so the robot could present good results. It is noticed that with a more complex environment, exists the possibility that an agent could take a longer time to get a good performance.

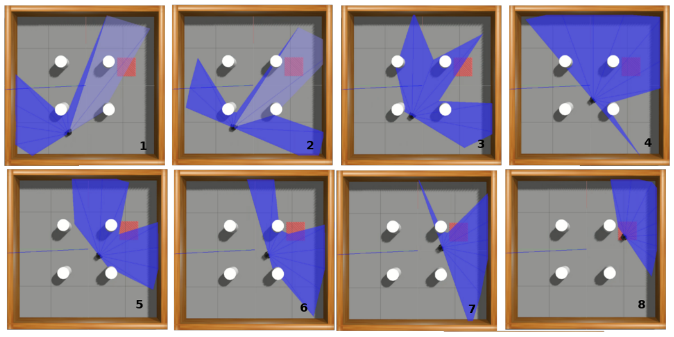


Fig. 8. Images sequence in the second environment.

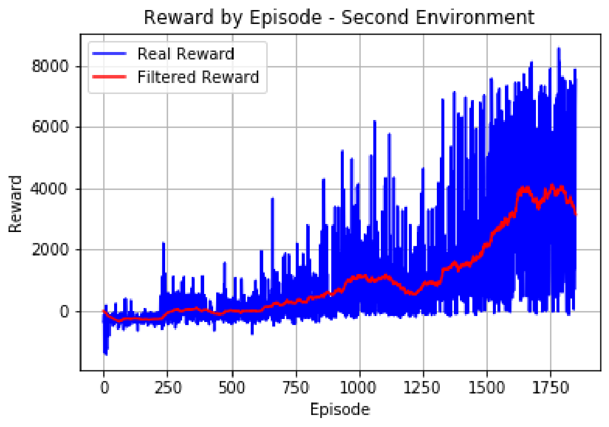


Fig. 9. Rewards of the second environment.

For the final test and the sequence of the actions made by the TurtleBot to arrive to the target after the training process it was used the third environment. A sequence of the actions made by the TurtleBot to arrive to the target after the training episodes are shown in Fig. 10.

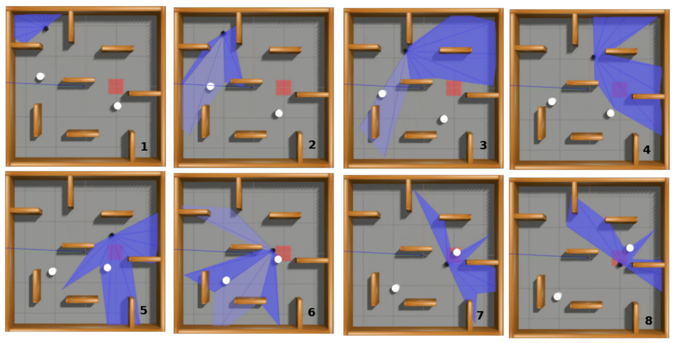


Fig. 10. Images sequence in the third environment.

The results of the reward function of the last training environment is shown in Fig. 11. On this environment, due to the high complexity, it was necessary a higher number of training episodes. If compared the reward function with the previous Fig. 7 and Fig. 9, it is possible to notice an average reward bellow two thousand. This happens because to reach the target, which has the highest reward possible on comparison with the others, it is necessary that the agent could execute actions that do not generate too many points, however, that still makes it to arrive to the target. Nevertheless, it was noticed by simulating the trained robot some actions that caused it to collide. Many of these collisions were due to the moving dynamic obstacle close to the target. This resulted in the agent making the decision of that mobile robot would collide or would try to avoid the obstacle. The occurred behavior may have been caused because of the system created. In order to resolve these actions, probably, it would be necessary to create a reward system that could bypass the error of the network.

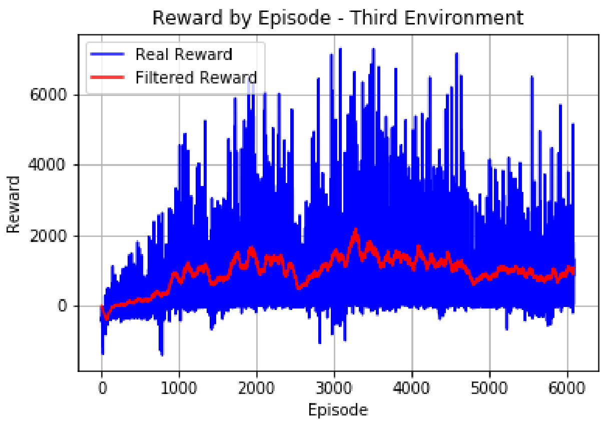


Fig. 11. Rewards of the third environment.

VII. Conclusion

In this paper was developed a neural network, DDPG, to be used on navigation of a mobile robot through continuous control in a virtual environment. Thus achieving a deep reinforcement learning network structure able to solve the problem of robot navigation. It was proposed as a task that the robot could reach a target position in different simulated environments and it created a reward function so that the DDPG network could give as results, the linear and angular velocity for the robot. All the network structure created was applied on the Gazebo simulation environments with success.

With the training results obtained on the simulation environments, it was analyzed the performance of the intelligent agent algorithm in the task of avoiding obstacles and getting to the final goal. On the three environments proposed, the algorithm had a good performance, however, in the last environment it was observed that sometimes even after many training episodes the mobile robot would still collide with some obstacle.

It is possible to conclude that DDPG networks are suitable for the development of applications that need a continuous control on the robotics. The Deep-RL networks can produce excellent results if the reward system is well-made for the problem that it wants to solve. It was proved that intelligent agents can move around in a complex simulated environment without any previous knowledge of the environment. The network based on deep deterministic policy gradients can provide means of unifying the machine learning for the control of robotic systems. This technique can be applied in function as manipulation of robotic arms, pendulums, games among others. As future works, it is intended to use the deep reinforcement learning technique for the navigation of the TurtleBot3 Burger robot in a real environment, thus, being able to validate the training in a simulated environment in a real environment.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[2] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.
[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.
[4] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.
[5] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3389–3396.
[6] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking reinforcement learning algorithms on real-world robots," *arXiv preprint arXiv:1809.07731*, 2018.
[7] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.
[8] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3357–3364.
[9] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," *arXiv preprint arXiv:1610.01733*, 2016.
[10] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 31–36.
[11] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 1343–1350.
[12] C. Fairchild and T. L. Harman, *ROS Robotics By Example*. Packt Publishing Ltd, 2016.
[13] L. Joseph, *Mastering ROS for robotics programming*. Packt Publishing Ltd, 2015.
[14] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.
[15] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in *2015 AAAI Fall Symposium Series*, 2015.
[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
[17] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Physical review*, vol. 36, no. 5, p. 823, 1930.
[18] D. Ascher and M. Lutz, *Learning Python*. O'Reilly, 1999.
[19] Y. Pyo, H. Cho, R. Jung, and T. Lim, "Ros robot programming," *Seoul, ROBOTIS Co*, 2015.