SSRG International Journal of Electrical and Electronics Engineering
ISSN: 2348 – 8379 / https://doi.org/10.14445/23488379/IJEEE-V8I4P106

Volume 8 Issue 4, 30-35, April 2021

Original Article

Research and Apply Deep Reinforcement Learning Technology to Control Mobile Robot

Roan Van Hoa[1], Nguyen Duc Dien[2], Lai Khac Lai[3]

[1,2]*University of Economics - Technology for Industries, Viet Nam*

[3]*Thai Nguyen University of Technology, Viet Nam*

Received Date: 12 March 2021
Revised Date: 15 April 2021
Accepted Date: 28 April 2021

Abstract *- Today, mobile robots are being strongly developed and widely used in life such as cargo robots, medical robots, wheelchairs for the disabled, etc. However, letting the robot move intelligently in dynamic environments without knowing the map in advance is a new research area of interest by scientists. The paper presents the application of deep reinforcement learning (DRL) to navigate mobile robots in an unknown environment. The system is built on robot operating system (ROS). The simulation results on the Gazebo software have verified the effectiveness of the proposed method.*

Keywords — *Robot operating system (ROS), Deep reinforcement learning, Navigation, Simultaneous localization and mapping (SLAM).*

I. INTRODUCTION

In Industry 4.0, mobile robot is an area of interest for its important role in everyday life as well as in manufacturing work and automated lines in industrial factories. Mobile robot is defined as a robot vehicle capable of self-movement and self-mobility to perform assigned tasks well. Usually an intelligent navigation system for a robot will consist of two main parts: simultaneous localization and mapping (SLAM) and a motion trajectory system. Fully portable robot systems are designed based on the robot programming operating system [1-3]. ROS is optimal for the design of unified robotic systems, especially combining, calibrating, and transmitting data from sensors with a central microcontroller circuit. Currently, DRL technology has been widely used in games [4, 5] and studied to improve robot performance instead of using both SLAM and motion trajectory system in environments unknown [6-8]. However, the deep reinforcement learning algorithm requires a large amount of data to train the robot . Therefore, it is recommended to use a simulation environment in order to speed up the training and not to tire the robot. In a simulation, creating an accurate model robot and its environment is a challenge and often requires a lot of enough data samples. To solve these dilemmas, robot learning is simulated and designed in Gazebo as it is compatible with the complex structure of the robot. Furthermore, Gazebo allows the construction of a virtual environment [9], which is imperative in the process of scrutinizing the reinforcement learning algorithms.

The remainder of the paper is organized as follows. The second part introduces the theoretical basis of deep reinforcement learning. Experimental setup will be given in the third section. The fourth part presents the architecture of the mobile robot control system. Finally, the fifth part is the conclusion of this article.

II. THEORETICAL BASIS

A. Reinforcemenr Learning Agent-Environment

A reinforcement learning task is about training an agent which interacts with its environment. The agent arrives at different scenarios known as states by performing actions. Actions lead to rewards which could be positive and negative. The agent has only one purpose here – to maximize its total reward across an episode. This episode is anything and everything that happens between the first state and the last or terminal state within the environment. We reinforce the agent to learn to perform the best actions by experience. This is the strategy or policy.

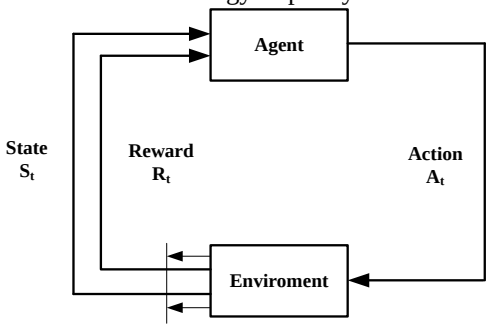


Fig. 1 Interaction diagram between agent and environment

B. Markov Decision Process (MDP)

An important point to note – each state within an environment is a consequence of its previous state which in turn is a result of its previous state. However, storing all this information, even for environments with short episodes, will become readily infeasible. To resolve this, we assume that each state follows a Markov property, i.e., each state depends solely on the previous state and the transition from that state to the current state.

A typical MDP is represented using a 6-tuple (S, A, T ,γ , D, R), where S is a (finite) set of possible states that represent a dynamic environment, A is a (finite) set of

available actions that the agent can select at a certain state, T is the state transition probability matrix that provides the probability of the system transition between every pair of the states, γ ∈ [0, 1) is the discount rate that guarantees the convergence of total returns, D is the initial-state distribution, and R is the reward function that specifies the reward gained at a specific state by taking a certain action.

C. Q-Learning

Let's say we know the expected reward of each action at every step. This would essentially be like a cheat sheet for the agent! Our agent will know exactly which action to perform. It will perform the sequence of actions that will eventually generate the maximum total reward. This total reward is also called the Q-value and we will formalise our strategy as:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q(s',a) \qquad (1)$$

The above equation states that the Q-value yielded from being at state s and performing action a is the immediate reward *r(s,a)* plus the highest Q-value possible from the next state s'. Q(s',a) again depends on *Q(s'',a)* which will then have a coefficient of γ^2. So, the Q-value depends on Q-values of future states as shown here:

$$Q(s,a) \rightarrow \gamma Q(s',a) + \gamma^2 Q(s'',a) + ... + \gamma^n Q(s^{,,...n},a) \quad (2)$$

Adjusting the value of γ will diminish or increase the contribution of future rewards. Since this is a recursive equation, we can start with making arbitrary assumptions for all Q-values. With experience, it will converge to the optimal policy. In practical situations, this is implemented as an update:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha\left[R_{t+1} + \gamma \max_{a} Q(S_{t+1},a) - Q(S_t,A_t)\right] \quad (3)$$

where α is the learning rate or step size. This simply determines to what extent newly acquired information overrides old information.

D. Deep Q-Learning

In deep Q-learning, we use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning and deep Q-learning is wonderfully illustrated below:

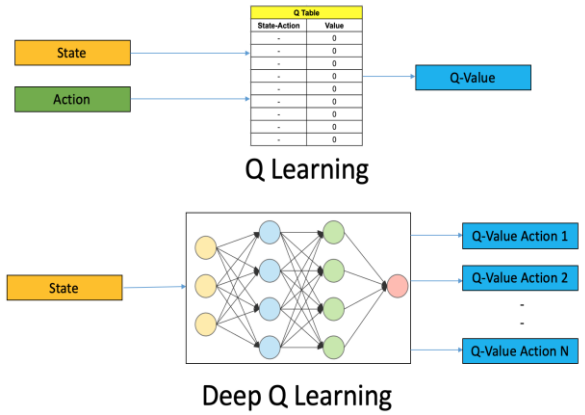


Fig. 2 From Q-learning to deep Q-learning

So, the steps involved in reinforcement learning using a deep Q-learning network (DQN) are:

- All the past experience is stored by the user in memory.
- The next action is determined by the maximum output of the Q-network.
- The loss function here is mean squared error of the predicted Q-value and the target Q-value – Q*. This is basically a regression problem. However, we do not know the target or actual value here as we are dealing with a reinforcement learning problem. Going back to the Q-value update equation derived fromthe Bellman equation. we have:

$$Q^* = R_{t+1} + \gamma \max_{a} Q(S_{t+1},a) \qquad (4)$$

In order for the neural network to learn how to estimate Q-Value for actions correctly, the Loss function must calculate the error between the actual and predicted Q-value:

$$Loss = \left[r + \gamma \max_{a'} Q(s',a';\theta_i') - Q(s,a;\theta_i)\right]^2 \quad (5)$$

E. Learning Procedure for Deep Q Learning of Mobibe Robot in ROS

The system observes the current scene including depth frames, and take action based ε-greedy strategy. The interaction experience $e = (s_i, a_i, r_i, s_{i+1})$ is stored in replace memory M keeps N most recent experiences by interacting with the environment , then DQN agent samples the mini batch from replay memory and train network on this mini batch.

Algorithm for Deep Q-learning with replace memory

Initialize replay memory to size N
Initialize the Q- network *Q(s,a,θ)*
Initialize the TurtleBot system
For episodes = 1, M do
  Initialize sequence $s_1 = \{x_1\}$ and sequence $\phi = \phi(s_1)$
  For t, 1, T do
    With probability ε select a random action at a_t
    Otherwise select $a_t = \max Q^*(\phi(s_t),a;\theta_i)$
    $s_{t+1}, r_t \leftarrow$ Execute action a_t observe reward r_t
    Store transition $(\phi_t,a_t,r_t,\phi_{t+1})$ in M
    Sample random mini batch of
    $(\phi_t,a_t,r_t,\phi_{t+1})$ from M
    Set:

$$Q^* = \begin{cases} r_i & \text{for terminal } \phi_{i+1} \\ r_i + \gamma \max_{a'} Q(\phi_{j+1},a';\theta) & \text{for non-terminal } \phi_{i+1} \end{cases}$$

    Perform a gradient descent step on
    $\left[y_i - Q(s,a;\theta_i)\right]^2$ according to equation (5)
  End for
End for

III. EXPERIMENTAL SETUP

For the analysis of a DQN network was used the programming language Python [10]. 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. Robot Operating System (ROS)

ROS [1] is an open source software framework for robotics development. The main goal of ROS is to standardize and reuse robot software globally and create a community for robot developers. ROS manages and provides application packages for a variety of purposes, and it has formed a delivery ecosystem of user-developed packages. As depicted in Fig. 3, ROS is a robot and sensor control support system with abstract hardware and robot application development based on existing conventional operating systems. ROS data communication is not only supported by one operating system but also supported by multiple operating systems, hardware and programs, making it well suited for the development of robots when many different hardware combined.

Fig. 3 ROS-enabled operating systems

The main architecture of ROS is based on nodes, and each node is an independent process. The use of the nodes offers a number of benefits to the overall system. First of all, since each node is independent of each other, the failure of one node is not likely to result in a total system crash. Second, the ROS architecture also helps to improve code reusability. In ROS, communication is done by passing messages between nodes (Fig. 4). In general, nodes do not know who they are communicating with. All sent messages are published for a topic. If a node wants to receive messages, it must subscribe to specific topics. There can be many publishers and many subscribers to one button. In the case of multiple threads, a button will ignore all notifications from the threads it did not subscribe to.

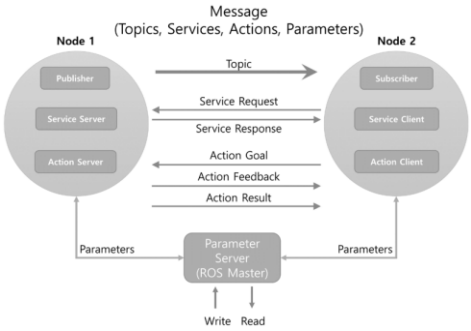


Fig. 4 Communication between buttons

In the paper, the authors use ROS Melodic version with Ubuntu 18.04 operating system on Dell Gaming G3 3590 computer with processor: Core i7 9750H, graphics card: Nvidia GTX1660 TI 6GB DDR5.

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. 5.

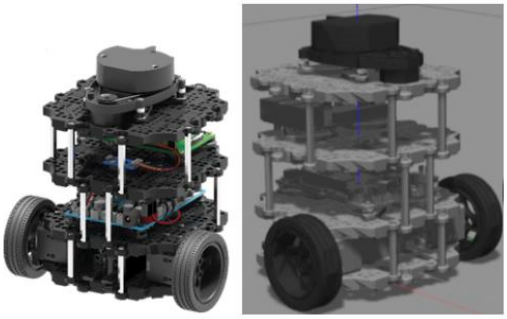


Fig. 5 Real and simulated TurtleBot3 Burger in Gazebo

The TurtleBot3 Burger uses 2 DYNAMIXEL motor 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. Software Gazebo

Besides, robotics experimentation is essential in robotics [11-13] as a final goal, 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 [9] is possible to simulate these environments easily and with the advantage of having an active community. This makes Gazebo a great tool on the area of robotic simulation.

IV. THE ARCHITECTURE OF CONTROL SYSTEM OF MOBILE ROBOT

In Deep Q-learning, use neural networks to approximate the Q-value function. The state is given as input and the Q-value of all actions can be generated as output.

The deep learning neural network model proposed in this study consists of four layers: input layer, two hidden layers, and output layer. The first hidden layer consists of 64 fully connected architecture, with 26 inputs taken from Laser Distance Sensor (LDS), distance to target and angle to target, There, there are 1728 trained parameters. The second hidden layer also has 64 neurons with 64 inputs from the first hidden layer so 4160 parameters are trained.

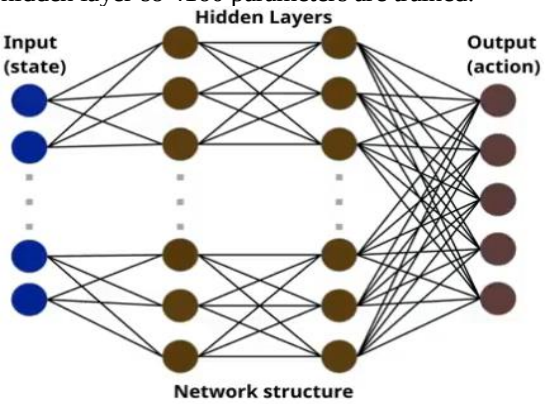
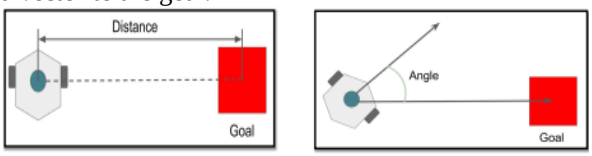


Fig. 6 Four-layer neural network structure

State: is an observation of environment and describes the current situation. This is vital for the agent because it would calculate and act depending on the state. The state size is 26 and 24 LDS (Laser Distance Sensor) values. The other two are distances to goal, and angle to goal. A mathematical approach for this is as follow: State = LDS (24 values) + Distance (1) + Angle (1) LDS denotes the (24) values that the lidar sensor emits. Distance represents the distance to the goal and Angle is the angle between the robot heading and vector to the goal.



Actions (Degrees of Freedom): The robot has five actions which can act on depending on the type of state. In here, the robot has a fixed linear velocity of 0.15m/s and the angular velocity is determined by the state.

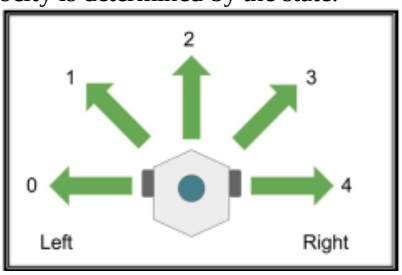


Fig. 7 Actions of mobile robot

Action	Angular velocity (rad/s)
0	-1.5
1	-0.75
2	0
3	0.75
4	1.5

Reward function: We need to define the reward and penalty system for the DRL network. Remember that rewards and penalty relationships are numbers attributed to smart agents. There are three different conditions for the reward system, which give better results for controlling the robot to automatically reach the target.

$$r = \begin{cases} 100 & \text{Goal} \\ -100 & \text{Collision} \\ 5r_d \cos\theta & \text{Other} \end{cases} \quad (6)$$

In which, r: reward function
r_d: reward from distance
θ: angle to target

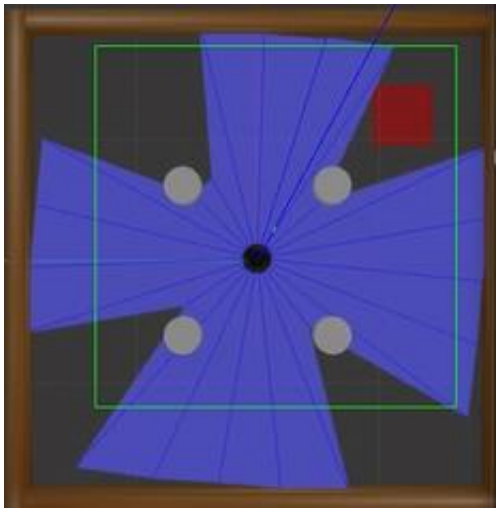
V. SIMULATION

In order to examine adequately the capability of the mobile robot, some tasks are performed in simulations by designing a robot simulator in Gazebo.

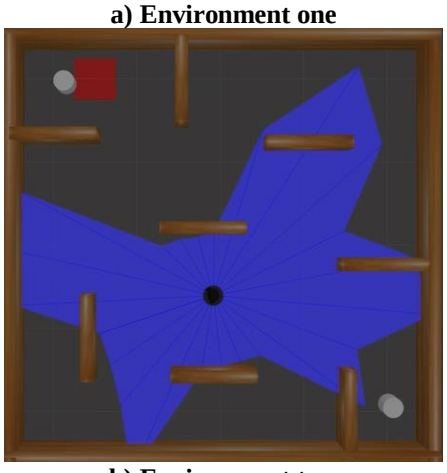
A. Simulation Environments

The training environment is set up to demonstrate the task of navigating and avoiding obstacles (including static and dynamic obstacles) for the robot in Gazebo. The black circle represents the robot, the chestnut-like walls, the white cylinder is the obstacle, the red square represents the robot's target, and the blue lines represent the scanning capabilities of the sensor. Laser way (LDS) from robot.

There are two environments used for the simulations. The first environment is shown in Fig. 8 (a), which represents an area of free movement for the robot to move. There are four fixed obstacles surrounded by the wall. The second environment, is shown in Fig. 8 (b). The number of walls increases, moving obstacles, represented by white blocks, make the environment more dynamic, closer to the environment in the real world. If the mobile robot collides with a wall or any obstacle, a negative reward will be awarded for this action and the current learning will stop. Conversely, if the mobile robot reaches the target, a positive reward will be awarded and the learning process resumes.



a) Environment one



b) Environment two

Fig. 8 Training environments used on Gazebo simulation

B. Simulation Result

Fig. 9 a sequence of the actions made by the TurtleBot from an initial position until it could arrive to the target after the training episodes.

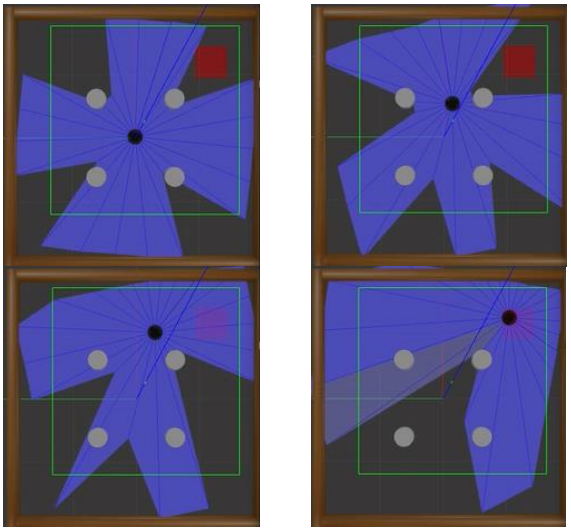


Fig. 9 Image series in the first simulation environment

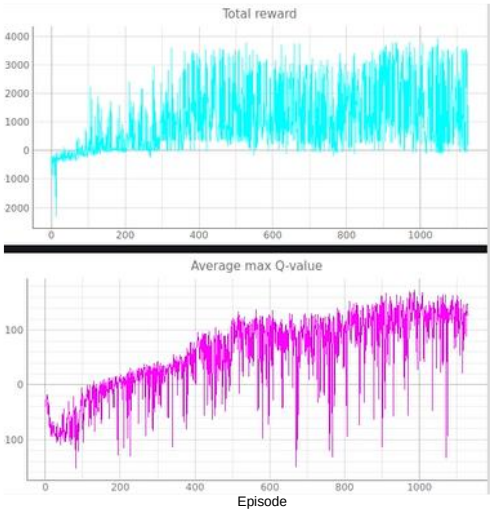
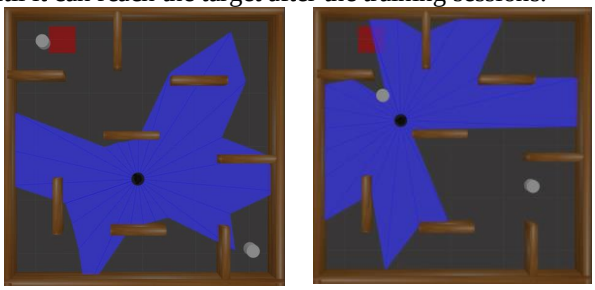


Fig. 10 The total rewards and average max Q-value of each epoch

In the first episodes it can be noticed that the total bonus points fluctuate around the negative value, which happens because the robot receives incomplete environmental information. This reward per episode means the robot is trying to maximize its reward for completing the quest.

After the autonomous robot is trained in the first environment, the experiment is carried out in the second environment and tested. It is shown in Fig. 11 a sequence of actions performed by TurtleBot from the starting position until it can reach the target after the training sessions.



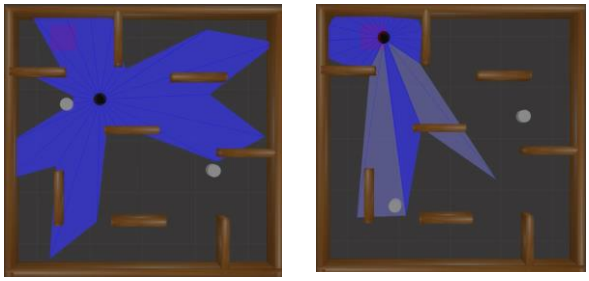


Fig. 11 Series of images in the second simulation environment

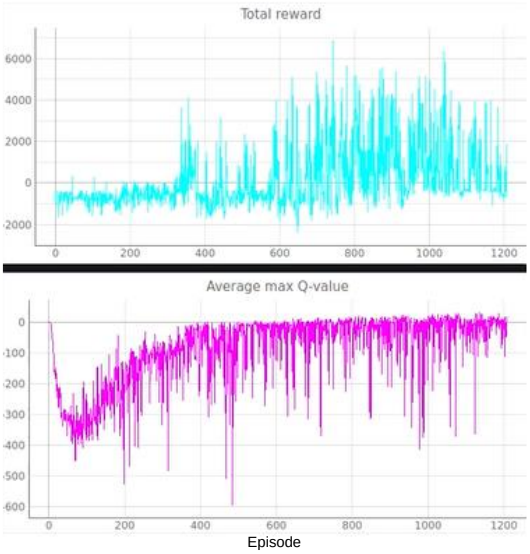


Fig. 12 The total reward and the average maximum Q value in the second environment

Fig. 12 shows the total reward that the robot gets in the second environment using the DQN algorithm. Comparing this result with the first environment, we can see that the robot needs more learning for it to achieve good results. From there we see with a more complex environment, the agent can take longer training to achieve good performance.

VI. CONCLUSION

This paper has proposed a deep reinforcement learning method to implement a robot's operating process in a virtual environment built in Gazebo. The robot system and reinforcement network are built into the ROS. A network-based task with laser scan signals generated from the LDS sensor as input data. The simulation results showed significant performance of the proposed method in obstacle avoidance and finding the way to the destination for Turtlebot. Furthermore, the robot's performance can be monitored through a visualization tool and opens up a highly probable DQN algorithm in various specific environments. The reinforcement learning network will be deployed in the robot's navigation mission in the real environment for future work.

ACKNOWLEDGMENT

This study was supported by University of Economics - Technology for Industries, Viet Nam; http://www.uneti.edu.vn/.

REFERENCES

[1] YoonSeok Pyo, HanCheol Cho, RyuWoon Jung, TaeHoon Lim, ROS Robot Programming, ROBOTIS Co.,Ltd (2017).

[2] Đỗ Quang Hiệp, Ngô Mạnh Tiến, Nguyễn Mạnh Cường, Lê Trần Thắng, Phan Xuân Minh, Xây dựng hệ thống nhận thức môi trường cho robot tự hành Omni bốn bánh dựa trên thuật toán EKF-SLAM và hệ điều hành ROS, Tạp chí Nghiên cứu KH&CN quân sự, Trang 30-37, Số Đặc san Hội thảo Quốc gia FEE (2020).

[3] Nguyen Duc Dien, Nguyen Duc Duong, Vu Anh Nam, Tran Thi Huong, Building Environmental Awareness System for Mobile Robot Operating in Indoor Environment on ROS Platform, SSRG International Journal of Electrical and Electronics Engineering (SSRG-IJEEE), (2021) 8(1) 32-36.

[4] Hosu I-A and Rebedea T, Playing Atari games with deep reinforcement learning and human checkpoint replay, 2016. ArXiv, abs/1607.05077.

[5] Mnih. V, Kavukcuoglu. K, Silver. D, Rusu. A.A, Veness. J, Bellemare. M.G, Graves. A, Riedmiller. M, Fidjeland. A.K, Ostrovski. G, et al, Human-level control through deep reinforcement learning, Nature 2015, pp. 518-529.

[6] Roan Van Hoa, Dinh Thi Hang, Tran Quoc Dat, Tran Dong, Tran Thi Huong, Autonomous Navigation for Mobile Robot Based on Reinforcement Learning, SSRG International Journal of Electronics and Communication Engineering, 8(1)(2021) 1-5.

[7] Roan Van Hoa, L. K. Lai, Le Thi Hoan, Mobile Robot Navigation Using Deep Reinforcement Learning in Unknown Environments, SSRG International Journal of Electrical and Electronics Engineering (SSRG-IJEEE), 7(8) (2020) 15-20.

[8] L. Tai and M. Liu, A robot exploration strategy based on qlearning network, in Proc. 2016 IEEE International Conference on Real-time Computing and Robotics (RCAR), 2016, pp. 57-62.

[9] I. Zamora, N. G. Lopez, V. M. Vilches, and A. H. Cordero, Extending the openai gym for robotics: A toolkit for reinforcement learning using ros and gazebo, arXiv preprint arXiv:1608.05742, 2016.

[10] D. Ascher and M. Lutz, Learning Python. O'Reilly, 1999.

[11] M. Pfitscher, D. Welfer, M. A. d. S. L. Cuadros, and D. F. T. Gamarra, Activity gesture recognition on kinect sensor using convolutional neural networks and fastdtw for the msrc-12 dataset, in International Conference on Intelligent Systems Design and Applications. Springer, (2018) 230–239.

[12] R. M. da Silva, M. A. d. S. L. Cuadros, and D. F. T. Gamarra, Comparison of a backstepping and a fuzzy controller for tracking a trajectory with a mobile robot, in International Conference on Intelligent Systems Design and Applications. Springer, (2018) 212–221.

[13] M. Cuadros, P. De Souza, G. Almeida, R. Passos, and D. Gamarra, Development of a mobile robotics platform for navigation tasks using image processing, in Asia-Pacific Computer Science and Application Conference (CSAC 2014), Shangai, China, 2014.