Controlling a Mobile Robot Using ROS2 and Machine Learning

Thesis · January 2024

DOI: 10.13140/RG.2.2.20125.73448

3 authors:

Saleh Abuali
GIPSA-lab
6 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Mickyas Tamiru Asfaw
Grenoble Institute of Technology
25 PUBLICATIONS **1,138** CITATIONS
SEE PROFILE

Perla Sammour
Notre Dame University
2 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Controlling a Mobile Robot Using ROS2 and Machine Learning

Saleh Abuali [1], Perla Sammour[1], Mickyas Tamiru Asfaw [1], Bogdan Robu[2], Ahmad Hably[2]
[1]Univ. Grenoble-INP UGA, ENSE3, Grenoble, France
[2]Univ. Grenoble Alpes, CNRS, Grenoble-INP, GIPSA Lab, Grenoble, France

Abstract:
The use of mobile robots for autonomous exploration and mapping has been an important research topic in recent years due to their ability to reduce labor costs and access restricted areas that are either inaccessible or hazardous to humans. Additionally, there has been a growing trend toward employing deep reinforcement learning (DRL) techniques for autonomous navigation in unknown environments due to their advanced experience-learning abilities. The purpose of this project was to apply DRL techniques to control a mobile robot using the Robot Operating System 2 (ROS2) framework, with the aim of exploring and mapping an unknown room. As a preliminary step into this new framework, a comprehensive study of ROS2 communication and tools was undertaken. This involved constructing a map of a Gazebo-simulated environment using the Simultaneous Localization and Mapping (SLAM) technique and utilizing the Navigation2 Stack planning server for autonomous navigation within the known environment. Subsequently, a thorough study of reinforcement learning (RL) and DRL techniques was conducted, leading to the implementation of a novel DRL algorithm for navigation in an unknown environment. After the implementation of the complete system, the SLAM toolbox was utilized to generate a map while the robot navigated towards randomly generated goals in the unknown room. The results showed that the robot successfully explored and navigated in the unknown room with an exceptionally limited number of collisions and a clean map of the simulated environment was generated.

Index Terms—**Mobile robots, Autonomous exploration, Deep Reinforcement Learning, Robot Operating System, Simultaneous Localization and Mapping.**

I. Introduction

In a large-scale unknown environment, exploration and map construction based on human guidance is cumbersome. Therefore, a key issue in robot research is the study of how robots can efficiently and reliably explore an unknown environment and construct an environment map [1]. For that, several navigation and exploration strategies were developed throughout the years. For instance, [2] introduced an autonomous exploration approach based on the concept of the frontiers or the region separating the explored and unknown areas in a map. Another approach is the use of multiple Rapidly-exploring Random Trees (RRTs) which can be generalized to three-dimensional spaces [3]. Additionally, the use of machine learning methods, including reinforcement learning and deep neural networks, has emerged as a significant development in robotic path planning [4]. This includes navigation in unknown environments. In fact, recent developments in deep reinforcement learning (DRL) for robot navigation have made it possible for autonomous agents to make high-precision decisions [5]. The use of such methods not only leads to precise results but also takes advantage of the experience-learning ability of the algorithm, which makes it adaptable to new situations.

For this reason, this project aimed to control a mobile robot using a novel DRL technique and the Robot Operating System 2 (ROS2) framework, to explore and map an unknown room. This new DRL technique known as the Twin-Delayed Deep Deterministic (TD3) algorithm was implemented in [5] for a goal-driven autonomous exploration. However, their work was done on ROS1 which is becoming relatively outdated due to the multiple new advancements in ROS2. Thus, this project combines the novelty in DRL with the new ROS2 framework to solve the challenging problem of autonomous exploration and mapping.

The work presented is divided into five main sections. The first section presents a comprehensive study of ROS2 communications and tools. After that, an overview of the different reinforcement learning methods is presented with an explanation of the implemented method. Then, the full system implementation will be explained. Finally, the results, conclusion and envisioned future work are discussed.

II. Getting started with ROS2

ROS2 is a set of software libraries and tools for building robot applications. It is designed to be a distributed and decentralized framework, allowing various nodes to communicate with each other to perform tasks. [6] As a preliminary step of this project, a first dive into the different ROS2 communication tools and packages was done.

A. ROS2 Nodes and Topics

Nodes are individual programs that perform a specific task. In ROS 2, complex systems are broken down into modular nodes, each responsible for a specific function, such as controlling a motor or processing sensor data.

Topics are a communication mechanism that allows nodes to exchange messages. They act as a bus for nodes to publish and subscribe to data. Topics are used for continuous data streams, such as sensor data or robot state. Messages are published via topics and subscribed to via topics.

The ROS2 tutorial, found in [7], was followed to create nodes and use services and topics to control a simple ”turtlesim” robot in a 2D simulated environment. Then, the knowledge grasped from the tutorial was used to control the

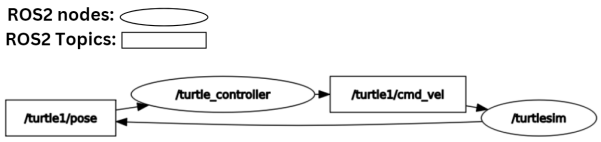


Fig. 1. The RQT graph of the Closed-loop System Controlling the Position of the Turtlesim

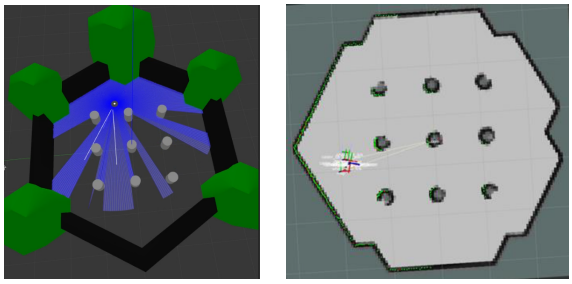


Fig. 2. Gazebo Simulated Environment and its SLAM-generated Map

"turtlesim" robot to reach randomly generated targets. This task aimed to introduce the concept of closed-loop control of the position variable in ROS2. This is illustrated in Figure 1, where the `/turtlesim` node feedbacks its pose to the `/turtle_controller` node, which in turn provides the right linear and angular velocities needed for the "turtlesim" to reach the goal.

B. Constructing a Map of Gazebo Environment

Gazebo is an open-source 3D robotics simulator that provides a platform for simulating robots in complex indoor and outdoor environments. It was integrated into ROS through a set of packages that provide a bridge between Gazebo's C++ API, and ROS messages and services. [8]

The ROS 2 Cartographer algorithm is used for simultaneous localization and mapping (SLAM) with TurtleBot3. It allows the robot to explore an unknown environment, navigate through it, and create a map. It is used to build a 2D binary occupancy grid map of the environment, which is then saved for navigational purposes. The simulated Gazebo environment, shown in Figure 2, was mapped using the Turtlebot3 Cartographer algorithm, where the robot was operated using teleop node which allowed to control the Turtlebot3 with keyboard inputs.

C. Autonomous Navigation using Navigation2 Stack

In ROS2, the Navigation2 (Nav2) stack provides tools for perception, planning, control, localization, visualization, etc. to build highly reliable autonomous systems [9]. The Nav2 stack tools include `Map_Server` to load a previously generated map, `AMCL` for localization, a Planner Server to generate a global path planner based on A* and other algorithms, and a Controller Server to control the robot's motion.

The Implementation of autonomous navigation within the previously mentioned Gazebo environment using NAV2 was successfully done. Figure 3 shows a Turtlebot3 which was able to generate obstacle-free trajectories and successfully follow predefined way-points within the environment.

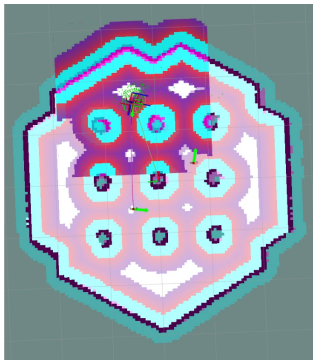


Fig. 3. Turtlebot3 Navigating Through Waypoints using NAV2 Stack

III. Using Reinforcement Learning for Navigation in Unknown Environments

A. Overview of Reinforcement Learning Algorithms

Reinforcement learning (RL) is a technique for machine learning that trains a set of neural networks to make sequential decisions in order to complete a given goal using a rewarding system [10]. More specifically, an RL agent is first given a series of observations that describe the task environment's state [11]. As a result, the RL agent decides how to go forward in order to accomplish the objective [11]. The agent's progress is assessed by an assessment procedure that is carried out after the action has been completed and is based on the updated condition of the environment. After that, the parameters of the agent are updated to enhance its performance.

In order to select the suitable algorithm for the purpose of this project, extensive research on reinforcement learning algorithms was then conducted to choose the novel Twin-Delayed Deep Deterministic (TD3) algorithm. To enhance comprehension of this choice, a brief overview of the different explored algorithms is presented.

Q-Learning is a basic algorithm that aims to compute, through value iteration using the Bellman equation, the optimal Q-values of each state-action pair and thus deduce an optimal policy by following the actions that maximize these Q-values [12]. Optimal Q-values are the maximum expected rewards obtained from a certain state and following a certain action under the optimal policy [12]. The limitation of this algorithm is that it requires a small state space as well. as a discrete action space since it consists of iteratively updating Q-values from a Q-table that combines the different sets of states and actions.

The deep Q-learning algorithm solves this issue that arises from having a complex state space by using neural nets as a Q-function estimator (instead of value iteration) to compute the Q-values of the different actions obtained from inputting a certain state (that can be of large size). This network is called a critic network. It also uses a target critic network that is a duplicate of the original network but its parameter updates are performed after a certain number of updates to the original network; this is done to prevent instability of the algorithm [13]. The limitation of this algorithm is that a discrete action space is required, which is not the case in robotic applications

since the robot actions like linear and rotational speeds of a wheeled robot have continuous values.

Deep Deterministic Policy Gradient (DDPG) algorithm uses continuous action space by using two networks: the actor and the critic networks. The actor network (or policy) takes the state vector as an input and outputs the action to be taken, while the critic network takes the actions and states and outputs a single Q-value of this state-action pair [14].The limitations of this algorithm are that it can be sensitive to hyperparameter tuning and may suffer from instability making it challenging to find a good set of parameters that lead to effective learning. Moreover, it tends to overestimate Q-values, which then leads to policy-breaking as it exploits the Q-function.

The Twin Delayed Deep Deterministic Policy Gradient (TD3) addresses the limitations of DDPG by incorporating three key techniques [15]:

Clipped Double-Q Learning: The algorithm employs two critic networks (hence "twin") to independently estimate Q-values for a given state-action pair. The smaller Q-value obtained from the target critic networks is used to compute the target Q in the Bellman error loss functions. This approach helps mitigate overestimation bias, resulting in faster and more accurate algorithm training. Both critic network parameters ϕ_1 and ϕ_2 are updated through backpropagation to minimize the loss functions of each network, computed using the same target. Equation (1) illustrates the Bellman equation used to calculate the target Q value denoted as y, while Equations (2) and (3) represent the loss functions to be minimized for each critic network. In these equations, r is the reward obtained upon reaching the state s', and d specifies whether this step was terminal. γ is the learning rate, and s' and a' are the state and action performed in the next time step. D represents the replay memory containing the experiences used for training:

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_i,\text{target}}(s', a'(s')) \quad (1)$$

$$\mathcal{L}(\phi_1, \mathcal{D}) = E_{(s,a,r,s',d)\sim\mathcal{D}} \left[\left(Q_{\phi_1}(s, a) - y(r, s', d) \right)^2 \right] \quad (2)$$

$$\mathcal{L}(\phi_2, \mathcal{D}) = E_{(s,a,r,s',d)\sim\mathcal{D}} \left[\left(Q_{\phi_2}(s, a) - y(r, s', d) \right)^2 \right] \quad (3)$$

Delayed Policy Updates: The updates to the parameters of the actor network are delayed compared to the critic network. This ensures that the critic network gets more accurate before doing a back-propagation to update the actor network and so it increases the network stability.

Target Policy Smoothing: To address the exploitation issue in the Q-function error, noise is introduced to the target action during training, creating smoother Q-values in response to action variations. This is represented in Equation (4), where μ_θ^{target} is the output of the target policy network, and ϵ is a zero-mean Gaussian noise with standard deviation σ. The parameter c denotes the maximum value of the noise, and a_{Low} and a_{High} are the lower and upper limits of the action.

$$a'(s') = \text{clip}\left(\mu_\theta^{target}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High} \right) \quad (4)$$

B. Implementation of TD3 Algorithm

To better illustrate how the TD3 algorithm is implemented, its pseudo-code is described in Algorithm 1, which can be found from [16]. To be able to understand this algorithm, an illustration of the implemented TD3 algorithm in [5] is provided in Figure 4. Note that this network, consisting of an actor-network that computes the action from the state and the two critic networks that evaluate the expected reward or Q-value of this action/state pair, is actually duplicated to have a separate "target" network. The purpose of this target network is to compute the target Q-values. In fact, after the convergence of the algorithm, the estimated Q-values of this target network should converge to the true expected reward of each state/action pair. The reason why a separate network is used to compute the target Q is because, in that way, the update of the parameter of this network can be delayed, which will prevent instability in the convergence of the algorithm.

Algorithm 1 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High} \right), \quad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_{\phi_i}(s, a) - y(r, s', d) \right)^2 \qquad \text{for } i = 1, 2$$

      if j mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s\in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

      end if
    end for
  end if
until convergence

As a first step, all network parameters are set to zero, and the replay buffer, which contains a set of experiences used to train the network, is also emptied. In addition, the target network parameters are also initialized to zero by setting them equal to the network parameters. Afterward, a sequence of steps is repeated to update the network's parameters until the convergence of the algorithm. First, the state consisting of the lidar measurements and the polar coordinate to the goal position is fed into the target actor network, which outputs $\mu_\theta(s)$. A noise is then added to this action as described in equation (4), and the action is executed in the environment. This leads to a new state (s'), reward (r) and a "done" signal (d) indicating if that action was terminal. All of these are then

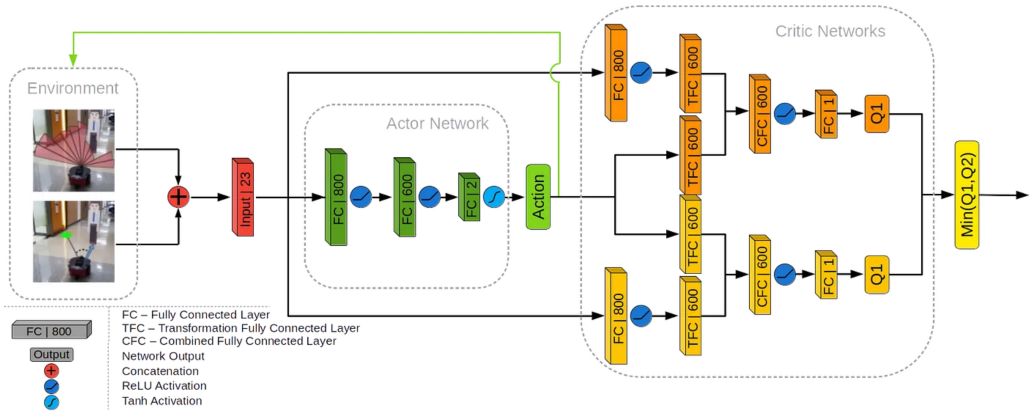


Fig. 4. Full Implemented TD3 Network Structure Consisting of the Actor and Critic Networks along with the Type of Layers and Number of Parameters. [5]

stored with the previous state and action in the replay buffer D. Of course, if the state s' was terminal, the environment should be reset.

Then, if it is time to update the parameters of the network the following steps are performed: First a batch is sampled from the replay buffer, and then the target actions are computed by feeding the next state s' into the actor target network then computing the action from equation (4). Afterwards, the target expected reward is computed using the Bellman equation and by taking the minimum of the two Q-values obtained from the two target critic networks. Then, a forward pass is done in the original network to obtain the Q value of that network, and the weights of the original critic networks are then updated by gradient descent to minimize the loss between the obtained Q-value and the target one. The policy or actor-network parameter update is performed next but after two updates of the critic network (or when the policy delay becomes 0). This network is updated through gradient ascent to maximize the expected reward obtained by feeding the state into this network and obtaining an action that maximizes the Q-value. Finally, the target network parameters of the actor and critic networks are updated (after two updates of the original critic network) using a soft update, which means that only a subset of these parameters will be updated. The latter is also done to avoid instability in the learning since updating all parameters at once will make them optimized for different sets each time the update is performed.

As previously mentioned, the algorithm used in this project was based on the one implemented in [5] since it showed successful results. Thus, the network architecture was the same as described in Figure 4. The actor-network consisted of two fully-connected layers, each followed by a Rectified linear unit (ReLU) activation function. This network has two outputs, which are the linear and angular velocities composing the "action" variable. In the critic network, the state is first fed into a fully connected layer followed by a Relu activation function. The output of this function as well as the action are fed into two separate transformation fully-connected layers (TFC) which are then joined through a combined fully connected layer (CFC). Then, a Relu activation is applied and finally, it is connected to the output with one parameter to represent the Q value.

The reward function used is described below, [5]:

$$r(s_t, a_t) = \begin{cases} r_g = 100 & \text{if distance to goal} < \text{threshold} \\ r_c = -100 & \text{if collision} \\ r = v - |\omega| & \text{otherwise} \end{cases}$$

Where r_g is the reward given when the goal is reached, r_c is the negative reward given when a collision occurs, and r is the reward obtained otherwise. The last case was used to encourage the robot to move forward to explore instead of turning around itself.

IV. System Implementation in Ros 2

A. System Setup

The training was done on an Asus ZenBook Pro Duo UX582HS, Core i9-11900HK 5GHz (16 CPUs), equipped with an NVIDIA GeForce RTX 3080 Laptop GPU (8GB Gddr6 V RAM), 32GB DDR4 RAM, Hard disk: 1TB pcie NVme Gen 4 Samsung SSD. It took about eight hours to train the TD3 network through 1237 episodes in the Gazebo simulator. However, due to time limitations and the need for powerful computational resources, available previously trained parameters were used for better accuracy. These parameters were obtained from a model that was trained for 3703 episodes for about 20 hours. During the training, every session (or episode) came to an end after 500 steps were completed, a target was met, or a collision was identified. The values of $\omega_{\max}$ and $v_{\max}$ were established at 1 rad per second and 0.5 meters per second, respectively. The parameter update delay was set to two episodes, and the delayed rewards were updated during the previous $n = 10$ steps. Evaluation of the training model was repetitively performed after 10 episodes and the average Q value and maximum Q value was stored in Tensorflow for visualization of their evolution with time.

Moreover, the training was conducted in a simulated 10×10 m-sized environment provided by [5]. This environment consisted of complex obstacles of different shapes and sizes, like tables, shelves, and triangular walls, as depicted in Figure 5. In addition, during training, cardboard boxes were displaced to let the model adapt to a new unknown environment.

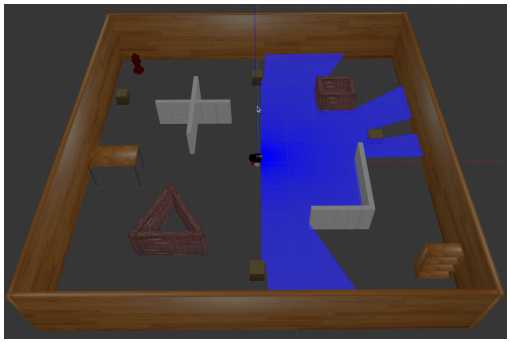


Fig. 5. Gazebo Environment

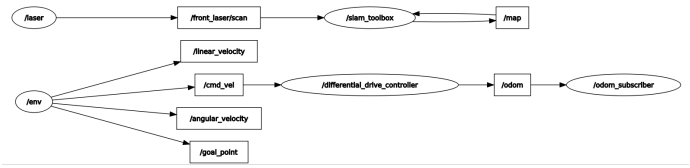


Fig. 6. RQT Graph of the Implemented Nodes

The robot chosen for this study was the Pioneer P3DX simulated mobile robot, a non-holonomic, differential-drive base with two controllable directions for linear (forward/backward) and angular (left/right) motion. These motions are numerically expressed as linear velocity in meters per second and angular velocity in radians per second, facilitating processing by a neural network.

Finally, the navigation of the robot was performed using the trained actor-network implemented in the `/env` node. This node generates random goals and computes the action to be taken by the robot at each timestep. These actions are published on the `/cmd_vel` topic to be used by the robot controller. In addition, this node publishes to the `/goal_point`, `/linear_velocity` and `/angular_velocity` topics that are used for visualization in Rviz. All these communications are illustrated in Figure 6. Moreover, to achieve Simultaneous Localization and Mapping and obtain a map of the environment, the `/slam_toolbox` node was implemented. This node subscribes to the `/front_lazer/scan` and publishes to the `/map` topic which is used to display the map in Rviz.

V. Results and Discussion

As previously mentioned, during training, the average Q-value was computed during the evaluation step. This stored value was plotted using Tensorboard to visualize the convergence after training. Figure 7 compares the training achieved on the previously-mentioned Laptop with the previously-trained model. It is clear that the model was training correctly however, it needed more time to converge. In addition, the average Q value obtained after convergence was around 30 and the model needed at least 3000 steps to converge. This result has led to the choice of the previously trained parameters for the testing phase.

The testing of the robot was first achieved by just testing the navigation algorithm and observing the robot reaching its goals. Then, the SLAM was initiated and the robot started mapping the environment while reaching the randomly-generated targets as seen from Figure 8 on the left. After some steps, the robot successfully mapped the whole room and a clean map was generated. The map obtained is presented in 8 on the right. Note that due to the fact that the navigation algorithm implemented suffers from local optimum, sometimes the robot would get stuck specially when the goal point is close to an obstacle. In that case, the algorithm generates random actions to try to mitigate this problem. However, this was not always effective as sometimes the robot would collide. A way to solve this problem is to make the generation of the goal points further away from the obstacle. Moreover, an implementation of a global planner like the one introduced in [5] would not only mitigate the problem of local optimum but also lead to a more efficient exploration of the room.

VI. Conclusion and Future works

This report outlines the implementation of a mobile robot control system employing ROS2 and the Twin-Delayed Deep Deterministic Policy Gradient (TD3) Reinforcement Learning algorithm. The focus is on autonomous exploration and mapping in unfamiliar environments. Key components include Gazebo for simulation, ROS2 for communication, and deep reinforcement learning for autonomous navigation. The TD3 algorithm is detailed, addressing the specific problems of previous DRL algorithms and overcoming challenges associated with other reinforcement learning methods. Experimental tests on a simulated Pioneer P3DX mobile robot demonstrate successful navigation and mapping, yielding a final map of the environment using the SLAM toolbox. Future project extensions could involve leveraging the map information generated by the SLAM toolbox to establish goal points beyond the currently covered area. This enhancement aims to prevent the robot from revisiting previously explored areas during its autonomous operations and can be used to mitigate the local optimum problem.

Several skills were acquired during this project: Python, ROS2, Gazebo, Rviz, Navigation2 Stack, SLAM, Autonomous Navigation Techniques, Q-learning, Deep-Reinforcement Learning, Twin-Delayed Deep Deterministic Policy Gradient. Teamwork, Time management, communication, and presentation skills.

Finally, the project timeline summarizing all milestones accomplished during this project is presented in table I. The weekly work was usually divided as follows: 7-hour team meeting at school, 3 hours of personal work and 1 hour of meeting with the supervisor.

Video: https://youtu.be/IRASuKMiOvw?si=n0sKMkWmtI97jMKv

Acknowledgment

We express our sincere gratitude to Dr. Bogdan Robu for his invaluable supervision and guidance throughout this project. Special thanks go to Dr. Amaury Negre from Gipsa, and we extend our appreciation to Reinis Cimurs, the author of the

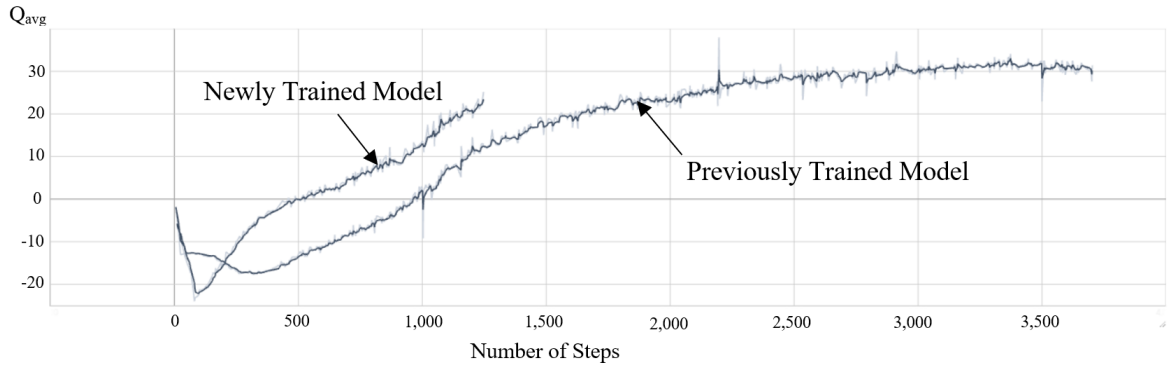


Fig. 7. Average Q-Value Comparison After Training

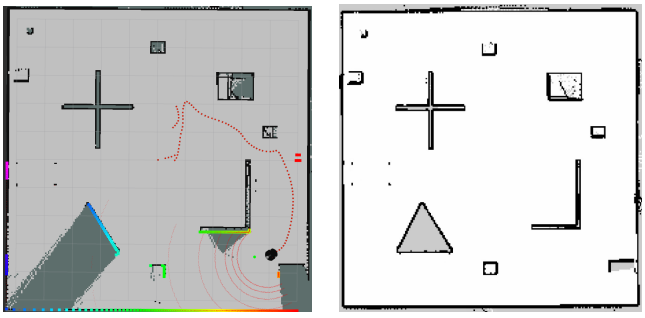
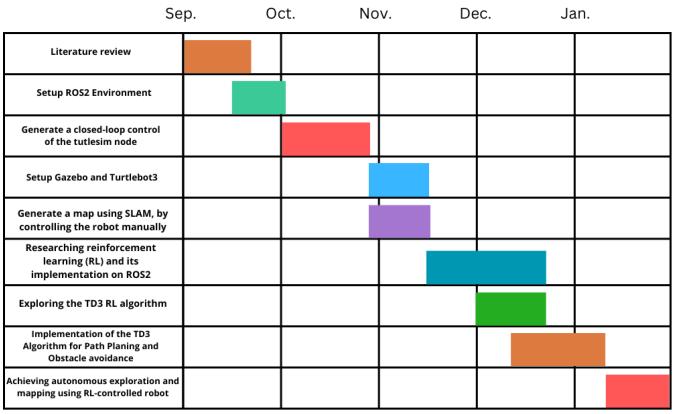


Fig. 8. The Robot Reaching the Goals and Mapping (right). Final Map of the Environment (left).

TABLE I

Integrator Project Timeline



paper [5], for generously sharing his knowledge through a GitHub tutorial.

REFERENCES

[1] H. Qin, S. Shao, T. Wang, X. Yu, Y. Jiang, and Z. Cao, "Review of autonomous path planning algorithms for mobile robots," *Drones*, vol. 7, no. 3, 2023. [Online]. Available: https://www.mdpi.com/2504-446X/7/3/211

[2] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, 1997, pp. 146–151.

[3] H. Umari and S. Mukhopadhyay, "Autonomous robotic exploration based on multiple rapidly-exploring randomized trees," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1396–1402.

[4] L. Liu, X. Wang, X. Yang, H. Liu, J. Li, and P. Wang, "Path planning techniques for mobile robots: Review and prospect," *Expert Systems with Applications*, p. 120254, 2023.

[5] R. Cimurs, I. H. Suh, and J. H. Lee, "Goal-driven autonomous exploration through deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 730–737, 2022.

[6] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[7] "Ros 2 documentation: Rolling documentation." [Online]. Available: https://docs.ros.org/en/rolling/Tutorials/Beginner-CLI-Tools.html

[8] "Gazebo ros 2 integration." [Online]. Available: https://classic.gazebosim.org/tutorials?cat=connect_ros&tut=ros2_overview

[9] S. Macenski, F. Martín, R. White, and J. Ginés Clavero, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[10] M. S. D. P. D. S. M. S. E. U. Y. Matsuo, Y. LeCun and J. Morimoto, "Deep learning, reinforcement learning, and world models," neural networks," *[Online].Available:https://www.sciencedirect.com/science/article/pii/S0893608022001150*, vol. 152, no. 267–275, 2022.

[11] M. C. . O. A. H. M. A. H. I. B. Rana Azzam1, 2 and Y. Zweiri, "Learning-based navigation and collision avoidance through reinforcement for uavs," *IEEE Transactions on Aerospace and Electronic Systems, available: DOI 10.1109/TAES.2023.3294889*, 2023.

[12] T. Ribeiro, F. Gonçalves, I. Garcia, G. Lopes, and A. F. Ribeiro, "Q-learning for autonomous mobile robot obstacle avoidance," in *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2019, pp. 1–7.

[13] M.-F. R. Lee and S. H. Yusuf, "Mobile robot navigation using deep reinforcement learning," *Processes*, vol. 10, no. 12, 2022. [Online]. Available: https://www.mdpi.com/2227-9717/10/12/2748

[14] ——, "Mobile robot navigation using deep reinforcement learning," *Processes*, vol. 10, no. 12, 2022. [Online]. Available: https://www.mdpi.com/2227-9717/10/12/2748

[15] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[16] "Twin delayed ddpg - spinning up documentation." [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/td3.html