|  |  |
| --- | --- |
| Immagine che contiene orologio  Descrizione generata automaticamente | |
| UNIVERSITY OF CATANIA | |
| **DIPARTIMENTO DI INGEGNERIA ELETTRICA ELETTRONICA E INFORMATICA** | |
| MASTER’S DEGREE IN AUTOMATION ENGINEERING AND  CONTROL OF COMPLEX SYSTEMS | |
|  | |
|  | |
| **Reinforcement learning for behaviour based navigation and physical implementation** | |
| ROBOTICS LABORATORY REPORT 2024 | |
| *Authors*: | *Supervisors:* |
| E. Garozzo, D. Gravina, G. Lecci, A.Motta | *Prof. G. Muscato, Prof. D. Guastella* |
|  |  |
|  |
|  |
| ACADEMIC YEAR 2023/2024 | |

# Contents

[Contents I](#_Toc169605051)

[Introduction 1](#_Toc169605052)

[Chapter 1 Reinforcement Learning 2](#_Toc169605053)

[Chapter 2 Physical Implementation 4](#_Toc169605054)

[2.1 ROS2 node map 4](#_Toc169605055)

[2.2 Multinode Node 4](#_Toc169605056)

[2.2.1 Lidar scan Callback 5](#_Toc169605057)

[2.2.2 Odometry Callback 5](#_Toc169605058)

[2.3 Lidar Node 7](#_Toc169605059)

[2.4 MicroROS 8](#_Toc169605060)

[Conclusions 10](#_Toc169605061)

[Bibliography 11](#_Toc169605062)

# Introduction

The core objective of our robotics project is to develop an advanced navigation system using a reinforcement learning algorithm. This system is designed to enable a robot, specifically the Pioneer model in our laboratory, which is equipped with four driving wheels in a differential drive configuration, to navigate through an unknown environment. The robot must avoid obstacles autonomously while striving to reach a predetermined target location.

To implement this project, we utilized the Robot Operating System (ROS) as the primary environment, with a MicroROS Agent facilitating communication. The low-level control of the robot's peripherals, including DC motors, is managed by an ESP32 microcontroller. This microcontroller handles the generation of PWM signals and integrates a PID controller in conjunction with encoders to ensure precise motor control.

For obstacle detection, we employed a Light Detection and Ranging (LIDAR) sensor. The LIDAR provides real-time data on the robot's surroundings, allowing the reinforcement learning algorithm to make informed decisions and navigate effectively. This sophisticated setup ensures that the robot can operate smoothly and efficiently in dynamic and unpredictable environments.

# Reinforcement Learning and Simulation

Our robotics project focuses on developing a behavior-based navigation system using a reinforcement learning algorithm. The robot, a Pioneer model equipped with four driving wheels and two DC motors with encoders, is designed to navigate an unknown environment. The objective is for the robot to avoid obstacles and successfully reach a predetermined target. The workspace relative to this simulation is **deep-rl-navigation\_ws**.

## TD3 Network introduction

To achieve this, a Twin-Delayed Deep Deterministic Policy Gradient (TD3) network has been used which is a model-free reinforcement learning method designed to find an optimal policy for maximizing long-term rewards. TD3 is an enhancement of the Deep Deterministic Policy Gradient (DDPG) algorithm, which addresses the issue of overestimating value functions that can lead to suboptimal policies. The key modifications in TD3 include learning two Q-value functions and using the minimum value during policy updates to reduce overestimation, updating the policy and targets less frequently than the Q-functions, and adding noise to the target action during policy updates to prevent the exploitation of actions with artificially high Q-value estimates.

## Training

During each episode, the robot executes a maximum number of steps before considering it a failure. An evaluation consists of an episode where the weights are not updated but only the AI agent's performance is assessed. The number of evaluations is determined by a parameter and is distributed evenly throughout the episodes. To enhance exploration, especially near obstacles, random actions were sometimes enforced when the robot encountered an obstacle, preventing it from getting stuck in local minima.

An episode starts with the robot at the initial position (0,0) and the goal placed nearby. It ends when the robot either crashes into an obstacle, reaches the goal, or exhausts the available steps. Exploration noise is added to each episode to encourage the agent to explore the map in search of the goal, while still basing its actions on the reward function.

The TD3 network was trained using gradient ascent, where the actor loss corresponds to the opposite of the worst-case critic output. By minimizing this, we maximize the critic output, which predicts the reward. The network inputs include the robot's state and the laser state (from a 180-degree lidar), and outputs a Twist message containing linear velocity along the robot's x-axis and angular velocity along the z-axis.

We also implemented a system to save the network every set number of episodes (10 in our case), optimized obstacle detection distances for collision consideration, and gradually increased the distance at which goals were generated from the robot's initial position as training progressed. Furthermore, we adjusted the mechanism for saving the network, now basing it on the number of episodes rather than the number of steps. We checked the “.world” file in Gazebo to locate obstacle positions, ensuring goals were not placed on them.

## Reward Function

The definition of a proper and effective reward function is essential to obtain an efficient network. The reward function was designed to adjust the AI model's learning based on several factors:

- Obstacle distances detected by the lidar.

- The distance and angle error relative to the goal.

- Previous actions, where spins are penalized and straight movements are rewarded.

Additionally, we applied the reward function in a decreasing manner to previous states leading to success or failure, thereby reinforcing or punishing the actions that led to those outcomes. To further refine the system, we generated reachable goals, ensuring they were never placed inside or above an obstacle, and utilized pre-trained models to enhance the training process.

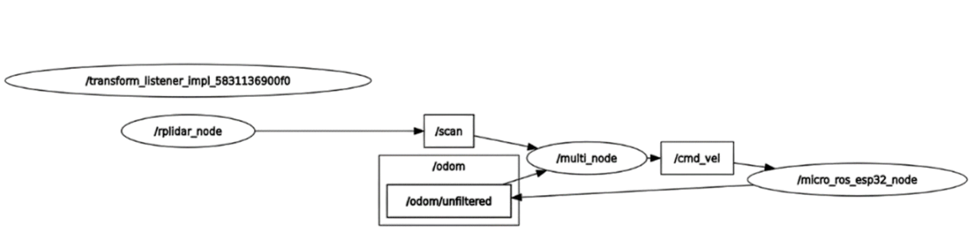
Multiple versions of the reward function were tested during training. The original reward was based on an inverted ReLU activated below a certain threshold. This function penalized proximity to obstacles and high angular velocities, while rewarding straight movements. Our modified reward function was defined as linear velocity minus the modulus of angular velocity, plus terms dependent on goal proximity, the absolute angle to the goal, and proximity to obstacles. Additional functions were added to reward high velocities when the distance to obstacles was large and to penalize high velocities when the distance was small.

Finally, we incorporated a mechanism to apply the final reward, whether negative for a collision or positive for reaching the goal, to preceding steps in a decreasing manner. This approach helps to reinforce or punish the actions that led to the outcome, thus improving the overall training effectiveness.

# Physical Implementation

## ROS2 node map

We used ROS2 Humble as software to implement our robot application. The image below illustrates the application's node and topic map. It was obtained using the rqt\_graph command, which shows three main nodes.



The first is the **/rplidar\_node**, whose main role is to publish the data from the RPLIDAR sensor to the **/scan** topic.

Subscriber this topic is the **/multi\_node**, which receives the scan data and process it to interface it with the network. This Td3 neural network is, in fact, embedded in this node. The network outputs are the angular and linear velocities (called actions) that the robot must have to reach the target. At this point, the node publishes this information to the **/cmd\_vel topic** so that the **/micro\_ros\_esp32\_node** can receive it.

The last node reads the high-level commands from the /cmd\_vel topic. Given the two actions, this node computes the desired angular velocities for the left and right wheel pairs using inverse kinematics. The second function performed by this node is to implement a velocity control loop, evaluating the control actions to maintain the wheels at a desired angular velocity. To maintain accurate knowledge of the current robot velocities, the /micro\_ros\_esp32\_node uses the encoders mounted on the wheels to measure their angular velocities and thus estimate robots velocities via direct kinematics. The information about the robot's velocity state is then published on the **/odom/unfiltered** topic.

## Multinode Node

This node workspace folder is **multinode\_ws**. Inside this workspace, there are two main scripts, one implements a “listener” node, one a “publisher” node. These names are due to the fact that this script was originally conceived as a test of publishing and subscribing to the topic /scan through these two nodes which created and received fake data to test if they functioned properly. In the end, the subscriber node became effectively the now called “multinode”, since it serve as a multipurpose node, and the “publisher” is just a complementary node to the first which subscribes to the topics the other node publishes to and publish in the topics the first subscribes to. Therefore, the publisher is not strictly a publisher anymore and serves only as a debugger when implementing new features to the main node (subscriber).

The two nodes can be run respectively through the following commands:

* ros2 run complete\_node complete\_subscriber
* ros2 run complete\_node complete\_publisher

The main functionalities of this node are implemented inside the callback functions associated with the publishing of messages in the different topics.

### Lidar scan Callback

The scan callback is a callback function that is called every time a new message is published in the topic /scan. Since the network cannot be overloaded with too many inputs, the part of the state regarding the lidar information is limited to only 20 elements. Given also the fact that only forward motion in conceived, the simulated lidar sensor only gives information on the front side of the robot, with a field of view of 180° and very coarse resolution (180/20=9°). Since the lidar used has a field of view of 360° with finer resolution, this callback must discard the 180° of the field of view regarding the posterior side of the robot and down-sample the ranges from 180 to 20. This can be done by dividing the remaining 180° FOV in 20 sectors, where each sector j is defined as the lower bound angle (**self.gaps[j][0]**) and its upper bound angle (**self.gaps[j][1]**). Then for every range i from the message ranges vector, we check in which sector the corresponding angle (angles[i]) that measurement falls in (simply by checking if self.gaps[j][0]<angles[i]<self.gaps[j][[1]). Once the correct sector j has been found, the corresponding element in the down-sampled vector (**self.lidarData[j]**) is updated with the minimum between the current stored value and the current range reading in analysis. The choice of the minimum means that the worst-case scenario is always considered which is crucial when trying to avoid the collision with an obstacle which would result in the termination of the current episode.

### Odometry Callback

For an autonomous navigation problem, especially in a behavior-based task , robot odometry is crucial to estimate the robot position in the space using the velocities published by /micro\_ros\_esp32. Three important variables of the MultiNode class are x and y positions, and the angle theta. Starting from an initial condition at the fixed reference frame origin, with orientation along the x-axis, robot odometry is evaluated iteratively whenever a message is published on the /odom/unfiltered topic

A math equations with symbols

Description automatically generated with medium confidence

Regarding the computation of the new pose, the second order Runge-Kutta integration equations shown above have been used. There are two other quantities that are very important to compute: the distance and the angle to the goal. The former is obtained by computing the length of the segment from the robot's position to the goal's position. The next step is to compute the angle β between the evaluated segment just obtained and the fixed reference frame’s x-axis. Lastly, we can obtain the angle α by computing β – θ (the current robot heading with respect to the fixed reference frame). This important angle is the correction angle to steer towards the target (angle\_to\_goal). These two quantities are very important because, along with the information provided by the /rplidar\_node, and the current linear and angular velocities, they are used by the TD3 network to obtain the next actions.

A diagram of a mechanical system

Description automatically generated with medium confidence

## Lidar Node

Crucial inputs to the network are the robot distances from the obstacles obtained thanks to the lidar sensor. For this purpose, the lidar **a1m8-r5** has been used, with 1° angle resolution and a FOV of 360°. The manufacturer provides the piece of software implementing the ROS2 node (contained in the folder workspace: lidar\_a1\_ws, the script can be launched with the command: ros2 launch rplidar\_ros view\_rplidar\_a1\_launch.py) concerning the publishing of the laser reading data to the topic **/scan** as a **sensor\_msgs/LaserScan** message. This message contains various fiels as shown in the figure below. A screenshot of a computer program

Description automatically generatedHowever, the fields of interest are pratically only *angle\_min, angle\_max, angle\_increment* and *ranges*. The last parameter is a vector containing the distances read for each angle to the obstacle, while the first two are the minimum and maximum angle read from the lidar and the third is the increment of angle between two successive readings. By dividing the difference between the maximum angle and the minimum angle by the angle increment, we get the total numbers of reading, 360. Thanks to these three parameters, the vector containing all the angles read from the lidar can be constructed. This means that now given an reading index i between 0 and 359, we can have both the information about the angle (angles[i]), and corresponding range (msg.ranges[i]) of that reading.

A part from publishing data to the topic /scan, this script also permits the visualization of the lidar readings through RVIZ as a point cloud. This visualization, along with further scripts implemented by us, has been crucial to associate the angles readings to the orientation of the lidar, in order to align the sensor to the front of the robot when mounting it to the chassis.

## MicroROS

MicroROS is a specialized middleware designed to extend the capabilities of the Robot Operating System (ROS) to resource-constrained microcontrollers. It provides a flexible and scalable framework that enables seamless communication between microcontrollers and the ROS ecosystem, thereby facilitating the integration of low-level hardware control with high-level robotic applications.

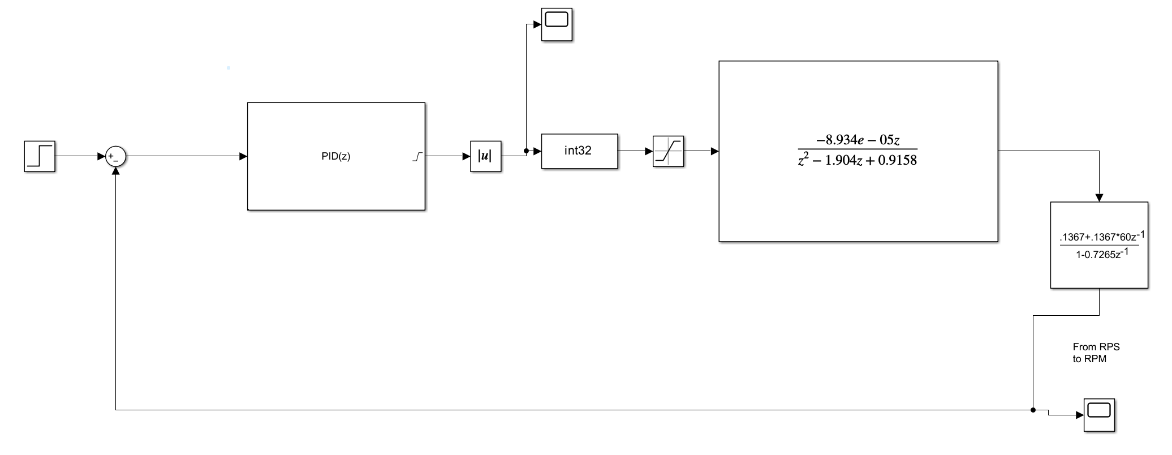
In our project, the adopted microcontroller was an ESP32. The ESP32 played a critical role in implementing the closed-loop controller, specifically a Proportional-Integral-Derivative (PID) controller. It was also responsible for reading the angular velocity (omega) of the wheels from the encoders and sending this information via serial communication to the master ROS system. The master ROS then utilized this data for integration and odometry calculations, ensuring tracking of the robot's position and movement.

The ESP32 publishes the omega wheel's velocity to the odom/unfiltered topic and subscribes to the cmd\_vel topic. By subscribing to cmd\_vel, it obtains the desired linear and angular velocity outputted from the network running on ROS. Using inverse kinematics, it computes the desired omega angular speed, which is then set as the reference for the PID controller.

**Inverse kinematics equations:**

v and are respectively the desired robot linear and angular speed while and the required linear velocity for the two couple of wheels.

We began by implementing an open-loop controller and acquiring data from the encoders. By sending a linear PWM signal ranging from 17 to 255, we gathered necessary data for system identification. Using the System Identification Toolbox in MATLAB, we developed a model of the system. This model was then used to simulate the controller in Simulink, allowing us to determine the parameters of the closed-loop controller needed to achieve the desired performance.



To address the issue of noise from the encoder, which would result in intermittent changes in velocity, we employed a Low-Pass (LP) first order Butterworth filter. This filtering process was essential to smooth out the velocity readings and ensure stable operation of the PID controller. Several filters were tested to find the best trade-off between bandwidth and performance. We eventually opted for a cutoff frequency around 7Hz. The presence of such low-frequency noise suggests that its origin might be due to mechanical vibrations. An FFT study on the encoder’s data could help design a better filter for future iterations of the project.

# Conclusions

In this project several results where obtained starting from the simulation scripts to train the TD3 network.

First and foremost, the reward function has been updated to include also proximity to goal and heading towards the goal to better guide the network during the train, achieving networks with better performances and enviroment awareness.

Furthermore, this network has been implemented on a real robot, starting from low level closed-loop PWM motor control to high level action calculation. Wheels velocity measurement through encoders coupled to the wheels motors permitted estimation of the robot position through odometry calculations.

A real lidar has been used to provide information about enviroments and its obstacoles. Differences between the real and the simulated lidar lead to necessary further elaborations on the upcoming data (down-scale and minimum range calculations).

Through the robot position estimated with odometry and the interfaced lidar data, the state vector needed by the TD3 network can be constructed, thus providing the needed high level velocity commands. However, this position estimation is fairly noisy and inaccurate. For this reason, the goal was difficult to reach but obastacle avoidance worked effortelessly.

In the future, integration of further sensors and systems such as GPS could enhance accuracy of the position estimation by etiher substituing odometry or correct its estimation using the Kalman filter. Another improvement, given the lidar used, could be to also include the removed 180° back FOV and implement backward motion to improve the robot manovrability in the enviroment.

Ottenimento di una rete più performante nel trovare il goal rispetto all’orginale (come lo dimostriamo? - (Enrico) Può essere dimostrato confrontando un set di trials ad esempio 100 goals e comparando le caratteristiche finali legate al numero di trial fissato: Ad esempio, si prende una rete, si fanno raggiungere 100 goal e si osserva in quanto tempo vengono raggiunti e quanti di questi effettivamente vanno a buon fine. Un altro tipo di confronto può essere effettuato verificando la robustezza alla variazione di mappe e combinazione di ostacoli. A parità di condizioni si testano le altre reti.), controllo a basso livello e lettura dagli encoder delle velocità, interfacciamento tra lidar e rete.

Prospetti futuri: introdurre gps per migliore stima della posizione o in sostituzione all’odometria (da sola soffre problema di deriva della stima) o in aggiunta all’odometria (sensor fusion, filtro di kalman), possibilità di aggiungere anche la possibilità di andare in retromarcia, aggiungere altri 20 elementi dal lidar relativi alla zona posteriore del robot.