Autonomous Navigation Robots

Project Overview:

Everyone's apartment room is different, imagine you purchased a sweeping robot, how can it quickly understand the environment of your home so that after sweeping the dust in your home he can effectively avoid the walls and return from the room back to where it should stay? This project is the answer.

The project base on Robot Operating System (ROS2) to design and implement an **autonomous navigation system** for **mobile robots** using **LiDAR & Deep Reinforce Learing(DRL)**. In order to improve the development efficiency, we reproduce and optimize based on the previous models from github.

Project Objectives:

The **primary objective** is to develop a robot capable of ***Autonomous Navigation***, applicable to tasks such as those **performed by a daily cleaning robot.** Other object are belows.

***Sensor Data Processing*:** The robot can utilize it **LiDAR** sensors to gather real-time environmental data, enabling dynamic obstacle and wall detection.

***Decision-making*:** By integrate **DRL** technologies, the robot can make decision which allow it to intelligently respond to environmental changes.

***Simulation Environment*:** Demonstrate the robot's capabilities in the **Gazebo**, ensuring its effectiveness and reliability in a virtual environment.

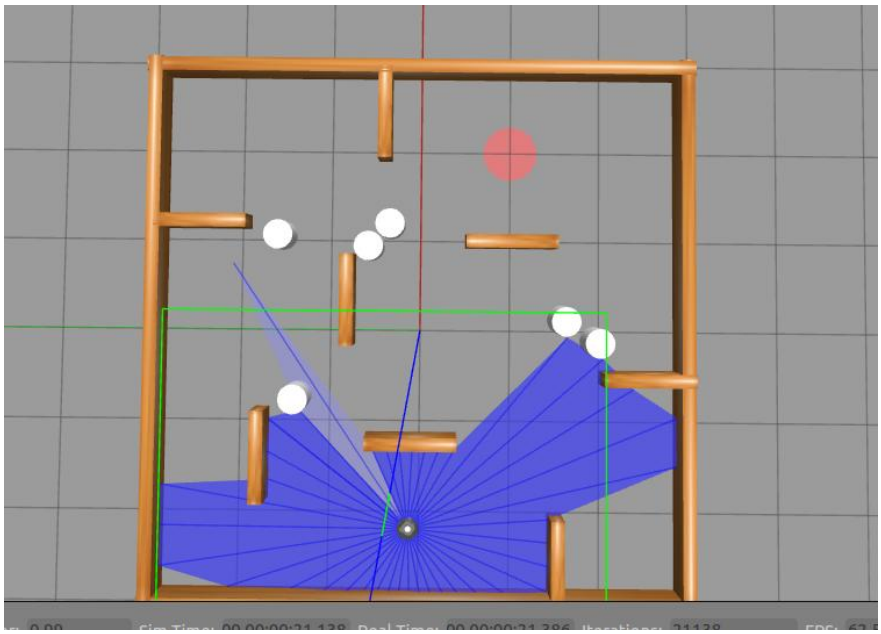
Scoring Criteria Achieves:

Simulation Software Usage

The project uses **Gazebo** for simulation display of the robot. In the simulation environment, the robot is able to avoid obstacles automatically and navigate to the target point with the

Autonomous Operation

The robot operates completely **autonomously** without any human control. It relies on **integrated sensors and AI algorithms(will mention below)** for real-time decision-making and independent navigation



Autonomous Robot in Gazebo with LiDAR

Sensor-based Behavior

The robot moves based on real-time data acquired by its **LiDAR sensors**. The sensor allows the robot to dynamically detect and avoid obstacles, map its environment and navigate efficiently.

Artificial Intelligence Integration

The robot integrates **DRL algorithms -- Deterministic Policy Gradient(DDPG)** that allow it to learn from its environment and improve its performance over time

Map Utilization

The robots utilize our predefined **map data** to understand and navigate their spatial environment. Using maps allows robots to effectively and quickly improve performance in otherwise complex environments

Description and Explanation:

1) Control System

This project is based on the **ROS2 framework.** ROS2 is an open source framework for the development of robotic systems. It provides a range of software libraries and tools for building robotic applications.

In this project, the ROS 2 framework is used to coordinate the operation of the robot, manage the acquisition and processing of sensor (LiDAR& IMU) data, and pass it to the learning module.

2) Sensor

The robot's sensors are modeled based on **turtlebot3**

a. LiDAR

The LiDAR sensor creates an environment map by emitting a laser and measuring its return time to obtain real-time environment data

```
<link name="base_scan">
  <inertial>
    <pose>-0.052 0 0.111 0 0 0</pose>
    <inertia>
      <ixx>0.001</ixx>
      <ixy>0.000</ixy>
      <ixz>0.000</ixz>
      <iyy>0.001</iyy>
      <iyz>0.000</iyz>
      <izz>0.001</izz>
    </inertia>
    <mass>0.114</mass>
  </inertial>

  <collision name="lidar_sensor_collision">
    <pose>-0.052 0 0.111 0 0 0</pose>
    <geometry>
      <cylinder>
        <radius>0.0508</radius>
        <length>0.055</length>
      </cylinder>
    </geometry>
  </collision>

  <visual name="lidar_sensor_visual">
    <pose>-0.064 0 0.121 0 0 0</pose>
    <geometry>
      <mesh>
        <uri>model://turtlebot3_waffle/meshes/lds.dae</uri>
        <scale>0.001 0.001 0.001</scale>
      </mesh>
    </geometry>
  </visual>
```

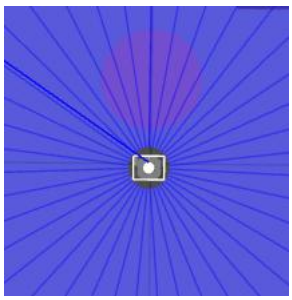
sdf Code of LiDAR

Data Reception

The Lidar sensor disseminates environmental scan data via the ROS 2 topic mechanism, specifically through messages of the type LaserScan. These messages encapsulate a series of distance measurements, each corresponding to the detected distance from the sensor to an object or obstacle. Collectively, these data points provide a *360-degree panoramic view* of the robot's surrounding environment, essential for comprehensive spatial awareness and navigation.

Data Integrity Verification

Prior to processing the received data, it is crucial to verify the integrity of the data points. Specifically, the number of data points received in each scan must match the expected count. This step ensures the consistency and reliability of the data processing workflow, which is vital for accurate analysis and subsequent decision-making processes in autonomous navigation.



LiDAR in Gazebo

LiDAR configuration

The configuration parameters of the LiDAR are critical for optimising the performance of the reinforcement learning model in a real environment. These parameters ensure accurate interpretation of the LiDAR data, which is essential for the robot to effectively navigate and interact with its surroundings.

Table1. Configuration of the LiDAR

Parameter	Description	Value
REAL_N_SCAN_SAMPLES	Number of scan samples provided by the LiDAR	40
REAL_LIDAR_CORRECTION	Correction factor subtracted from raw LiDAR measurements	0.40 m
REAL_LIDAR_DISTANCE_CAP	Maximum distance value for LiDAR measurements	3.5 m
REAL_THRESHOLD_COLLISION	Minimum distance considered as a collision	0.11 m

By appropriately configuring these parameters, the navigation ability and overall performance of the robot in dynamic environments can be improved.

Data Normalization

In order to apply the raw LiDAR data to the reinforcement learning model, normalization is required. For each distance measurement a predetermined correction factor is subtracted and then divided by a predetermined maximum distance value. This normalization not only preserves the relative distance

information but also increases the efficiency of data processing and improves the generality of the learning algorithm.

Updating Obstacle Distance

Concurrent with the data normalization process, the algorithm calculates and updates the distance to the nearest obstacle detected by the Lidar. This distance is a critical parameter for the robot's obstacle avoidance strategies and directly influences the effectiveness of the navigation algorithms. By continuously updating this distance, the robot can dynamically adjust its path in real-time to avoid collisions, thereby enhancing navigational safety and efficiency. This process involves checking the normalized distances for the minimum value, which represents the closest obstacle, and then recalculating it back to the actual distance using the maximum range parameter. This real-time updating of obstacle distance is integral to the robot's ability to navigate complex environments safely and efficiently.

b. IMU

IMU sensor measures the angular velocity and linear acceleration of the robot to obtain the position information and motion status of the robot.

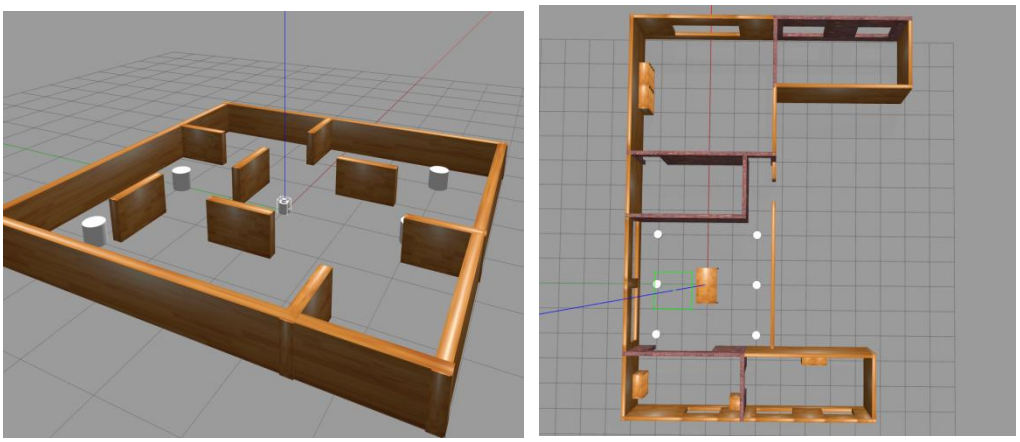
```
<link name="imu_link">
  <sensor name="tb3_imu" type="imu">
    <always_on>true</always_on>
    <update_rate>66</update_rate>
    <imu>
      <angular_velocity>
        <x>
          <noise type="gaussian">
            <mean>0.0</mean>
            <stddev>2e-4</stddev>
          </noise>
        </x>
        <y>
          <noise type="gaussian">
            <mean>0.0</mean>
            <stddev>2e-4</stddev>
          </noise>
        </y>
        <z>
          <noise type="gaussian">
            <mean>0.0</mean>
            <stddev>2e-4</stddev>
          </noise>
        </z>
      </angular_velocity>
      <linear_acceleration>
        <x>
          <noise type="gaussian">
            <mean>0.0</mean>
            <stddev>1.7e-2</stddev>
          </noise>
        </x>
        <y>
          <noise type="gaussian">
            <mean>0.0</mean>
            <stddev>1.7e-2</stddev>
          </noise>
        </y>
        <z>
          <noise type="gaussian">
            <mean>0.0</mean>
            <stddev>1.7e-2</stddev>
          </noise>
        </z>
      </linear_acceleration>
    </imu>
    <plugin name="turtlebot3_imu" filename="libgazebo_ros_imu_sensor.so">
      <ros>
        <!-- <namespace>/tb3</namespace> -->
        <remapping>~/out:=imu</remapping>
      </ros>
    </plugin>
  </sensor>
</link>
```

Code of IMU

IMU helps the robot estimate its own posture, maintain balance, and transmit information to the learning module to assist the robot in making path decisions

3) Simulation Environment

The robot is trained and evaluated in the **Gazebo** simulation environment. Gazebo has a realistic physics engine and sensor simulation, and rapid environment modeling. The team have added more walls and obstacles to the previous map model to speed up the training efficiency.



Map Utilization and Simulation

In Gazebo, teams can **train** a reinforcement learning navigation model and **perform** a model **evaluation** to test its performance and stability in different scenarios. Optimize the model parameters during trial and error to improve the effectiveness of the navigation strategy

4) Deep Reinforce Learning

The project enables robots to autonomously learn and improve their navigation strategies by implementing and training **DDPG** algorithms using the **PyTorch framework**.

This project uses ROS 2 and PyTorch frameworks to develop and deep reinforcement learning algorithms to train and evaluate models in a simulation environment:

a.Model Description

Actor Network

The Actor network generates actions given a state. It consists of three fully connected layers in this implementation:

```python
class Actor(Network):
    def __init__(self, name, state_size, action_size, hidden_size):
        super(Actor, self).__init__(name)

        self.fa1 = nn.Linear(state_size, hidden_size)
        self.fa2 = nn.Linear(hidden_size, hidden_size)
        self.fa3 = nn.Linear(hidden_size, action_size)


        self.apply(super().init_weights)

    def forward(self, states, visualize=False):

        x1 = torch.relu(self.fa1(states))
        x2 = torch.relu(self.fa2(x1))
        action = torch.tanh(self.fa3(x2))

        if visualize and self.visual:
            self.visual.update_layers(states, action, [x1, x2], [self.fa1.bias, self.fa2.bias])
        return action
```

Actor Class

Layer 1: Processes the input state vector and outputs hidden features.
Layer 2: Processes features from the first layer.
Layer 3: Generates the action vector.

Critic Network

The Critic network evaluates the value of a given state-action pair. It consists of four fully connected layers.

```python
class Critic(Network):
    def __init__(self, name, state_size, action_size, hidden_size):
        super(Critic, self).__init__(name)

        self.l1 = nn.Linear(state_size, int(hidden_size / 2))
        self.l2 = nn.Linear(action_size, int(hidden_size / 2))
        self.l3 = nn.Linear(hidden_size, hidden_size)
        self.l4 = nn.Linear(hidden_size, 1)

        self.apply(super().init_weights)

    def forward(self, states, actions):
        xs = torch.relu(self.l1(states))
        xa = torch.relu(self.l2(actions))
        x = torch.cat((xs, xa), dim=1)
        x = torch.relu(self.l3(x))
        x = self.l4(x)
        return x
```

Critic Class

Layer 1: Processes the input state vector.
Layer 2: Processes the input action vector.
Layer 3: Combines features from state and action vectors and processes.
Layer 4: Outputs the Q-value.

Then, implements the main parts of the DDPG algorithm, including action selection, training processes, and network updates. Based on the DDPG algorithm, the robot can learn the real-time data obtained by the sensor through the algorithm to make decisions.

b.Model Configuration

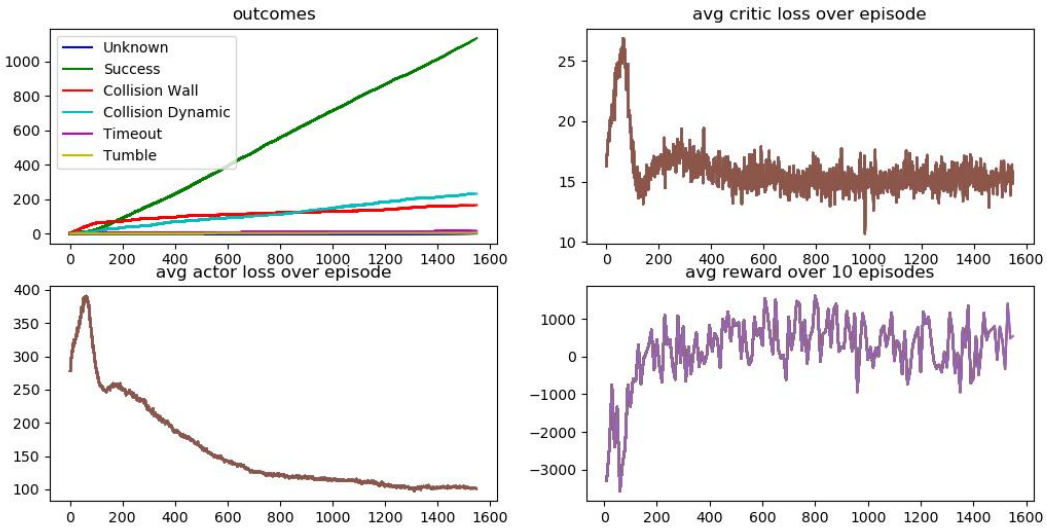
In the development of Deep Deterministic Policy Gradient (DDPG) algorithms, specific parameters play a pivotal role in determining the effectiveness and efficiency of the training process. Below is a table summarizing the essential parameters that directly influence the model's performance in a reinforcement learning environment.

Table2. Configuration of the DDPG

Parameter	Description	Value
HIDDEN_SIZE	Number of neurons in hidden layers	512
BATCH_SIZE	Number of samples per training batch	128
BUFFER_SIZE	Maximum number of samples stored in the replay buffer	1,000,000
DISCOUNT_FACTOR	Discount factor for calculating the present value of future rewards	0.99
LEARNING_RATE	Rate at which the model weights are updated	0.003
TAU	Coefficient for soft updates of the target networks	0.003
EPSILON_DECAY	Decay rate of epsilon in epsilon-greedy policy	0.9995
EPSILON_MINIMUM	Minimum epsilon value for the epsilon-greedy policy	0.05

By appropriately configuring these parameters, the DDPG algorithm can effectively learn and adapt to complex environments.

c.Model Evaluation



Evaluation of DDPG

The agent shows significant learning progress as indicated by the increase in successful outcomes and the decrease in actor loss.

With the DRL algorithm, the robot can learn and optimize its navigation strategy by independently interacting with it in dynamic and uncertain environments. Then, the team run the trained model on the Gazebo, and the robot can quickly find the way to reach the Goal.