Goal-Driven Mobile Robot Exploration Using Deep Reinforcement Learning

Kanishk Kaushal

Abstract

This project introduces a fully autonomous navigation system for goal-driven exploration in unknown environments using Deep Reinforcement Learning (DRL). The proposed method selects optimal waypoints from the environment, integrates a DRL-based motion policy into a navigation framework, and effectively mitigates local optimum challenges. The results show significant improvements in navigation efficiency and reliability, with real-world applications in autonomous mapping, exploration, and navigation tasks.

1 Overview

This project presents a novel autonomous exploration system enabling robots to navigate toward specified global goals in unknown environments without prior knowledge. The system integrates waypoint selection strategies with a Twin Delayed Deep Deterministic Policy Gradient (TD3)-based neural network for local navigation. Extensive experiments in static and dynamic environments validate the system's performance, demonstrating improved efficiency, safety, and adaptability compared to traditional methods.

2 Problem Definition

Efficient autonomous navigation and exploration in unknown environments pose significant challenges, especially in scenarios requiring dynamic decision-making without prior knowledge or human intervention. Current methods struggle with local optima, safety, and real-time adaptability.

2.1 Why is this problem interesting?

This problem has widespread implications for autonomous systems in real-world applications, such as disaster response, warehouse automation, and space exploration. Solving this enhances society's ability to perform complex, hazardous, or resource-intensive tasks autonomously and safely.

3 Deep Reinforcement Learning (DRL)

Deep Reinforcement Learning (DRL) combines the principles of reinforcement learning (RL) with deep learning. RL involves training agents to make decisions by interacting with an environment to maximize cumulative rewards. In DRL, neural networks approximate the policy or value functions, enabling complex decision-making tasks.

3.1 Key Components of DRL

- **State** (s_t): Represents the environment at time t.
- **Action** (a_t): Decision made by the agent.
- **Reward** (r_t): Feedback received after taking an action.
- **Policy** (π): Mapping from states to actions.
- **Value Function**: Measures the expected cumulative reward from a state.

4 Manipulation of Mobile Robot Position using DRL

In this project, DRL manipulates the mobile robot's position by learning an optimal motion policy. The robot uses its current position and sensor inputs (e.g., laser readings) to decide the next movement. The DRL model adjusts the robot's linear and angular velocities, which directly influence its position and orientation in the environment.

4.1 Robot Kinematics and DRL

The robot's state consists of its position (x, y) and orientation θ, which change over time as the robot moves based on the control signals from the DRL model. The key kinematic equations that describe the robot's motion are:

$$\dot{x} = v \cos \theta, \quad \dot{y} = v \sin \theta, \quad \dot{\theta} = \omega$$

where:

- v: Linear velocity, determined by the DRL model's action.
- ω: Angular velocity, determined by the DRL model's action.

The DRL model learns to manipulate these velocities (v and ω) based on the sensor inputs (e.g., laser data) and the position of obstacles. By adjusting v and ω, the robot can navigate to the goal or avoid obstacles.

4.2 Action Output by DRL Policy

The TD3-based actor network receives input data, which includes the robot's state (position, velocity, etc.) and waypoint data. The actor network then outputs two action parameters: linear velocity v and angular velocity ω. These actions control the robot's motion, manipulating its position as it moves towards the target.

The relationship between the wheel velocities (v_l, v_r) and the robot's motion is:

$$v = \frac{r}{2}(v_r + v_l), \quad \omega = \frac{r}{L}(v_r - v_l)$$

where:

- r: Wheel radius.
- L: Distance between the wheels.

Thus, the actions v and ω output by the DRL model directly control the wheel velocities, determining the robot's movement through the environment.

4.3 Training the DRL Model for Navigation

During training, the DRL model learns to take actions that maximize the cumulative reward. The reward function $r(s_t, a_t)$ encourages the robot to reach the global goal while avoiding obstacles. The reward is calculated as:

$$r(s_t, a_t) = \begin{cases} r_g & \text{if } D_t < \eta_D \\ r_c & \text{if collision} \\ v - |\omega| & \text{otherwise} \end{cases}$$

where:

- r_g: Positive reward for getting closer to the global goal.
- r_c: Negative reward for collisions.
- $v - |\omega|$: Immediate reward based on velocity and angular velocity.

The DRL model adjusts its policy over time, improving its ability to navigate efficiently to the goal by learning from past actions and rewards.

5 Twin Delayed Deep Deterministic Policy Gradient (TD3)

TD3 is a model-free actor-critic algorithm that improves upon the Deep Deterministic Policy Gradient (DDPG) method. It handles continuous action spaces effectively.

5.1 Key Features of TD3

- **Actor-Critic Framework**: The actor network selects actions, while the critic network evaluates the quality of actions (Q-value).
- **Target Networks**: Maintains a slowly updated version of actor and critic networks to stabilize training.
- **Twin Critics**: Uses two critics to reduce overestimation bias by taking the minimum Q-value.
- **Delayed Updates**: Updates the actor and target networks less frequently to improve stability.

5.2 Policy Gradient

The policy gradient optimizes the policy π by maximizing the expected reward:

$$\max_{\theta} J(\theta) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid \pi_\theta\right]$$

where $\gamma \in [0, 1]$ is the discount factor, and θ represents the policy parameters.

The gradient is computed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim d^\pi, a\sim\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a \mid s) Q^\pi(s, a)\right]$$

6 Proposed Approach

6.1 Global Navigation

Waypoints are selected using the *Information-based Distance Limited Exploration (IDLE)* method:

$$h(c_i) = \tanh\left(e^{\left(\frac{d(p_t, c_i)}{l_2 - l_1}\right)^2} - e^{\left(\frac{l_2}{l_2 - l_1}\right)^2}\right) l_2 + d(c_i, g) + e^{I_{i,t}}$$

Where:

- $d(p_t, c_i)$: Euclidean distance between the robot's position p_t and POI c_i.
- $d(c_i, g)$: Distance from POI to the global goal g.
- $e^{I_{i,t}}$: Map information score calculated with a kernel k over candidate POIs.

6.2 Local Navigation

The local motion policy is trained using a TD3-based neural network:

- **Input**: Bagged laser readings and waypoint coordinates.
- **Actor-Critic Architecture**: The actor outputs actions (a_1, a_2) for linear and angular velocities, scaled as:

$$a = \begin{bmatrix} v_{\max}\left(\frac{a_1+1}{2}\right) \\ \omega_{\max} a_2 \end{bmatrix}$$

- **Reward Function**:

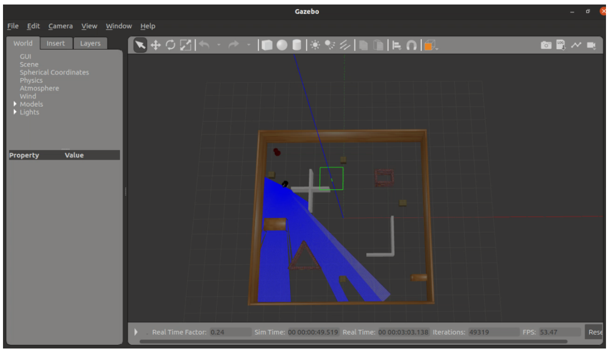
$$r(s_t, a_t) = \begin{cases} r_g & \text{if } D_t < \eta_D \\ r_c & \text{if collision} \\ v - |\omega| & \text{otherwise} \end{cases}$$

6.3 Algorithm

The following is the algorithm used.

Algorithm 1 Goal-Driven Autonomous Exploration

```
Input: Set globalGoal and threshold $\delta$
while reachedGlobalGoal = False do
    Read sensor data
    Update map from sensor data
    Obtain new POIs
    if $D_t < \eta_D$ then
        if waypoint = globalGoal then
            reachedGlobalGoal = True
        else
            if $d(p_t, g) < \delta$ then
                waypoint = globalGoal
            else
                Calculate $h(i)$ for all POIs
                waypoint = POI with smallest $h$
    Get action from TD3 policy
    Perform action
```



Real Time Factor: 0.24 Sim Time: 00 00:00:49.519 Real Time: 00 00:03:03.138 Iterations: 49319 FPS: 53.47

Figure 1: Gazebo simulation

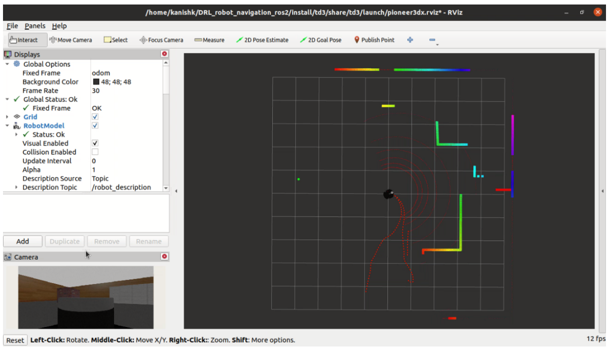
7 Experiment Setup

7.1 Dataset and Implementation

- **Dataset**: Simulated environments with randomized obstacle positions.
- **Implementation**: Training performed in Gazebo with ROS.
- **Model Architecture**: Actor-Critic TD3 with fully connected layers and ReLU activations. Tanh activation is used at the output for bounded action space.

8 Results and Discussion

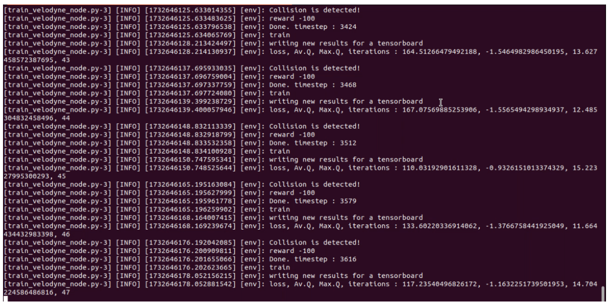
The proposed system achieved significant improvements in navigation efficiency and robustness. It successfully avoided local optima and performed reliably in cluttered environments. However, limitations include dependency on specific robot dynamics and lack of memory for large-scale exploration.



/home/kanishk/DRL_robot_navigation_ros2/install/td3/share/td3/launch/pioneer3dx.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Displays
Add
Duplicate
Remove
Rename
Camera
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.
12 fps

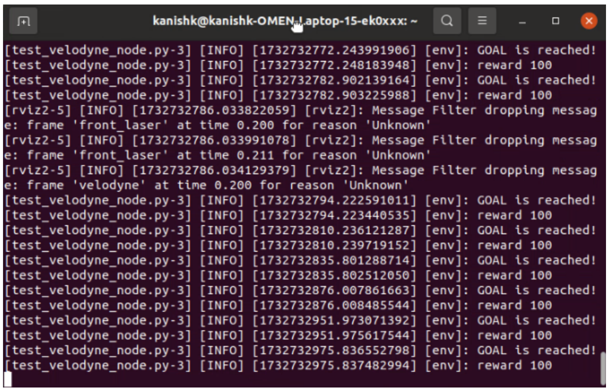
```
Global Options
  Fixed Frame            odom
  Background Color       48; 48; 48
  Frame Rate             30
Global Status: Ok
  Fixed Frame            OK
Grid                     ✓
RobotModel               ✓
  Status: Ok
  Visual Enabled         ✓
  Collision Enabled      □
  Update Interval        0
  Alpha                  1
  Description Source     Topic
  Description Topic      /robot_description
```

Figure 2: Rviz visualization



```
[train_velodyne_node.py-3] [INFO] [1732646125.633014355] [env]: Collision is detected!
[train_velodyne_node.py-3] [INFO] [1732646125.633483625] [env]: reward -100
[train_velodyne_node.py-3] [INFO] [1732646125.633796538] [env]: Done. timestep : 3424
[train_velodyne_node.py-3] [INFO] [1732646125.634065769] [env]: train
[train_velodyne_node.py-3] [INFO] [1732646128.213424497] [env]: writing new results for a tensorboard
[train_velodyne_node.py-3] [INFO] [1732646128.214130937] [env]: loss, Av.Q, Max.Q, iterations : 164.51266479492188, -1.5464982986450195, 13.627
458572387695, 43
[train_velodyne_node.py-3] [INFO] [1732646137.695933035] [env]: Collision is detected!
[train_velodyne_node.py-3] [INFO] [1732646137.696759004] [env]: reward -100
[train_velodyne_node.py-3] [INFO] [1732646137.697337759] [env]: Done. timestep : 3468
[train_velodyne_node.py-3] [INFO] [1732646137.697724080] [env]: train
[train_velodyne_node.py-3] [INFO] [1732646139.399238729] [env]: writing new results for a tensorboard
[train_velodyne_node.py-3] [INFO] [1732646139.400057946] [env]: loss, Av.Q, Max.Q, iterations : 167.07569885253906, -1.5565494298934937, 12.485
304832458496, 44
[train_velodyne_node.py-3] [INFO] [1732646148.832113339] [env]: Collision is detected!
[train_velodyne_node.py-3] [INFO] [1732646148.832918799] [env]: reward -100
[train_velodyne_node.py-3] [INFO] [1732646148.833532358] [env]: Done. timestep : 3512
[train_velodyne_node.py-3] [INFO] [1732646148.834100928] [env]: train
[train_velodyne_node.py-3] [INFO] [1732646150.747595341] [env]: writing new results for a tensorboard
[train_velodyne_node.py-3] [INFO] [1732646150.748525644] [env]: loss, Av.Q, Max.Q, iterations : 110.03192901611328, -0.9326151013374329, 15.223
27995300293, 45
[train_velodyne_node.py-3] [INFO] [1732646165.195163084] [env]: Collision is detected!
[train_velodyne_node.py-3] [INFO] [1732646165.195627999] [env]: reward -100
[train_velodyne_node.py-3] [INFO] [1732646165.195961778] [env]: Done. timestep : 3579
[train_velodyne_node.py-3] [INFO] [1732646165.196259902] [env]: train
[train_velodyne_node.py-3] [INFO] [1732646168.164007415] [env]: writing new results for a tensorboard
[train_velodyne_node.py-3] [INFO] [1732646168.169239674] [env]: loss, Av.Q, Max.Q, iterations : 133.60220336914062, -1.3766758441925049, 11.664
434432983398, 46
[train_velodyne_node.py-3] [INFO] [1732646176.192042085] [env]: Collision is detected!
[train_velodyne_node.py-3] [INFO] [1732646176.200909811] [env]: reward -100
[train_velodyne_node.py-3] [INFO] [1732646176.201655066] [env]: Done. timestep : 3616
[train_velodyne_node.py-3] [INFO] [1732646176.202623665] [env]: train
[train_velodyne_node.py-3] [INFO] [1732646178.052156215] [env]: writing new results for a tensorboard
[train_velodyne_node.py-3] [INFO] [1732646178.052881542] [env]: loss, Av.Q, Max.Q, iterations : 117.23540496826172, -1.163225173950953, 14.704
224586486816, 47
```

Figure 3: training



```
[test_velodyne_node.py-3] [INFO] [1732732772.243991906] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732772.248183948] [env]: reward 100
[test_velodyne_node.py-3] [INFO] [1732732782.902139164] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732782.903225988] [env]: reward 100
[rviz2-5] [INFO] [1732732786.033822059] [rviz2]: Message Filter dropping messag
e: frame 'front_laser' at time 0.200 for reason 'Unknown'
[rviz2-5] [INFO] [1732732786.033991078] [rviz2]: Message Filter dropping messag
e: frame 'front_laser' at time 0.211 for reason 'Unknown'
[rviz2-5] [INFO] [1732732786.034129379] [rviz2]: Message Filter dropping messag
e: frame 'velodyne' at time 0.200 for reason 'Unknown'
[test_velodyne_node.py-3] [INFO] [1732732794.222591011] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732794.223440535] [env]: reward 100
[test_velodyne_node.py-3] [INFO] [1732732810.236121287] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732810.239719152] [env]: reward 100
[test_velodyne_node.py-3] [INFO] [1732732835.801288714] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732835.802512050] [env]: reward 100
[test_velodyne_node.py-3] [INFO] [1732732876.007861663] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732876.008485544] [env]: reward 100
[test_velodyne_node.py-3] [INFO] [1732732951.973071392] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732951.975617544] [env]: reward 100
[test_velodyne_node.py-3] [INFO] [1732732975.836552798] [env]: GOAL is reached!
[test_velodyne_node.py-3] [INFO] [1732732975.837482994] [env]: reward 100
```

Figure 4: testing

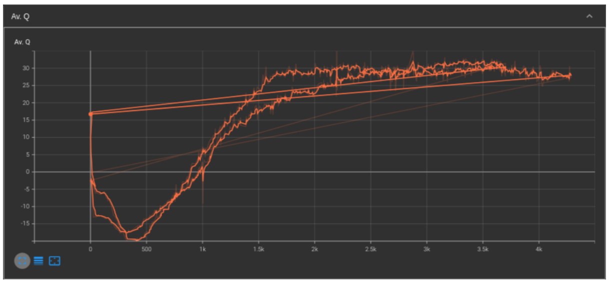


Figure 5: average Q converging

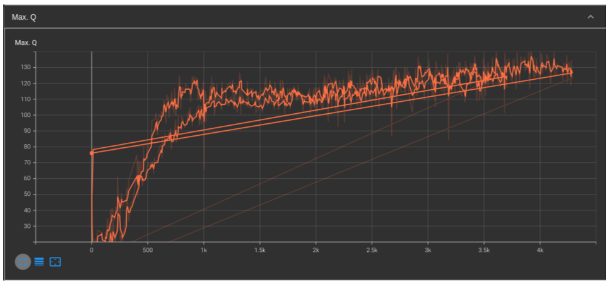


Figure 6: maximum Q converging

9 Conclusion

This project demonstrates a hybrid navigation system that integrates DRL and heuristic planning to address goal-driven exploration challenges in unknown environments. The system achieves a balance between efficiency, reliability, and adaptability.

References

1. Reinis Cimurs, Il Hong Suh, Jin Han Lee, "Goal-Driven Autonomous Exploration Through Deep Reinforcement Learning," *arXiv*, 2021.