Robot Navigation by Deep Reinforcement Learning

Kannak Sharma*, Lili Ye†, Zeming Zhou‡
Emails: [ksharm88, liliye, zzhou157]@asu.edu

***Abstract*—We are Team 10, and our project replicates key findings from our main reference paper [1], focusing on the challenge of robot navigation using deep reinforcement learning (DRL). Specifically, our goal is to achieve safe and efficient navigation for robots within the Gazebo simulation environment by leveraging data from LiDAR sensors. Our approach involves implementing and comparing two RL algorithms: Deep Deterministic Policy Gradient (DDPG) and Proximal Policy Optimization (PPO). To evaluate the effectiveness of RL for robot navigation, we analyze observations, rewards, and actions, along with the training process. Additionally, we replicate and run the code within a scaled-down version of the Gazebo environment. Our findings suggest that the custom PPO algorithm from our main reference paper [1] may not demonstrate significant advantages in low-dimensional environment settings.**

***Index Terms*—Autonomous Navigation, Deep Reinforcement Learning, Deep Deterministic Policy Gradient (DDPG), Proximal Policy Optimization (PPO), Actor-Critic Model, Dynamic Environments, Obstacle Avoidance, Real-Time Decision Making, LiDAR Sensors.**

I. Introduction

Autonomous navigation in robotics is crucial for applications ranging from industrial automation to complex real-world interactions in dynamic environments like warehouses. Traditional navigation methods, often reliant on static maps or heuristic-based planning, are limited in adaptability to real-time changes, making reinforcement learning (RL) a good alternative for developing responsive, self-adaptive navigation policies. In this assignment, we primarily leverage Proximal Policy Optimization (PPO) as a robust framework for autonomous mobile navigation in unstructured environments [1].

In this project, we implement the PPO algorithm within the Gazebo simulation environment via the Robot Operating System (ROS) framework, allowing our robot to navigate toward specified goals while avoiding obstacles. PPO's strengths lie in its stability and efficiency, allowing for continuous action spaces and safe navigation through Actor-Critic architecture, which learns an effective policy by adjusting the balance between exploration and exploitation. This allows robot to make adjustments in response to its environment, maintaining safety without needing pre-defined maps [1,2].

Complementary approaches in recent RL research also underscore the efficacy of RL for large-scale, dynamic environments. For example, Graph Relational Reinforcement Learning has demonstrated the advantages of combining relational graph structures with RL to manage interactions between multiple dynamic obstacles, such as humans in crowded environments. This method uses local observations aggregated into graphical inputs, enhancing decision-making by allowing the robot to understand both individual and collective object behaviors within its surroundings [2]. Additionally, research comparing PPO with other algorithms like Deep Q Networks (DQN) has highlighted PPO's effectiveness in learning robust, collision-free paths [3].

By focusing on PPO within a structured RL framework and integrating LiDAR-based observation data, our project aims to provide deeper insights into "why" PPO excels in autonomous navigation tasks. This research not only demonstrates PPO's ability to generate effective, adaptable navigation policies but also contributes to the broader understanding of RL's role in developing safer, more reliable robotic systems for complex, dynamic environments defined maps [1-3].

We have downloaded their codes and attempted to simulate their results. However, we found that reproducing their results for the 8×8 environment may take three to four days of computation. Consequently, we opted to simulate only the 4×4 Gazebo environment while maintaining the same obstacle configuration. Due to many errors in their original code, we created a smoothly running version and uploaded it to GitHub. Additionally, we provide two GIFs to illustrate the well-trained PPO algorithm online interacting with both the simple and complex Gazebo environments, available online for demonstration purposes:

https://github.com/liliyequantum/RL_in_robotics

II. Central Issue: Context, Challenges, and Approaches in Autonomous Robot Navigation

Problem Context: Autonomous mobile robots are increasingly used in dynamic environments where they interact with both stationary and moving obstacles. For safe and efficient navigation in such settings, a robot must avoid collisions and continuously adapt its path based on real-time sensory data.

Challenges in Autonomous Navigation: Navigating dynamic and unstructured environments presents several key challenges [1-3]: **1. Obstacle Avoidance:** Robots must detect and maneuver around obstacles, which could be either static (e.g., walls) or dynamic (e.g., people, other robots). **2. Real-Time Decision Making:** Efficient navigation requires split-second decision-making as robots process continuous streams of data. **3. Limited Sensory Data:** Robots often rely on local observations (e.g., LiDAR or ultrasonic sensors) that provide limited information, meaning the entire environment is rarely known at once. **4. Adaptability and Generalization:** Robots should generalize navigation strategies across different settings without extensive reprogramming or reliance on pre-built maps.

Reinforcement Learning as a Solution: The Role of PPO Reinforcement Learning (RL) offers a powerful alternative to

traditional methods by allowing robots to develop navigation strategies through direct interaction with their environment. Among RL algorithms, Proximal Policy Optimization (PPO) is especially suited for this purpose due to its stability and efficiency in managing continuous action spaces. PPO employs an *Actor-Critic* framework, where the **Actor** suggests actions based on the current state (e.g., velocity and position relative to obstacles), and the **Critic** evaluates these actions using a reward signal. This evaluation guides the robot toward increasingly effective navigation policies over time. Through this iterative feedback loop, PPO enables the robot to learn from past experiences and refine its navigation strategies, ultimately allowing it to handle complex tasks such as avoiding obstacles while efficiently reaching a target.

III. How: Implementation of the Proximal Policy Optimization (PPO) Algorithm

Pseudo-algorithm: PPO Agent

Input: Initial policy $\pi(a|s;\theta)$, critic value function $V(s;\phi)$, clipping factor ϵ, policy learning rate α_θ, value function learning rate α_ϕ, number of episodes N_{ep}, number of epochs K, and number of mini-batches M.

Output: Optimized policy parameters θ.

1) **for** Episode $= 1$ to N_{ep} **do**:
   a. Collect trajectory D by running policy π_θ in the environment.
   b. Compute advantage estimates $\hat{A}_t$ based on $V(s;\phi)$.
   c. Calculate return G_t for each state in the trajectory.
   d. Update the policy π_θ using stochastic gradient ascent and the value function $V(s;\phi)$ by stochastic gradient descent.
   – **for** epoch $= 1$ to K **do**:
      i. Divide trajectory D into M mini-batches.
      ii. **for** each mini-batch in D **do**:
         · Compute probability ratio $r_t(\theta)$ for the new and old policies.
         · Calculate the clipped surrogate objective $L_{\text{CLIP}}(\theta)$.
         · Compute the square-error loss $L^{V^F}(\phi)$ for the value function.
         · Update policy parameters θ using $\theta \leftarrow \theta + \alpha_\theta \nabla_\theta L_{\text{CLIP}}(\theta)$.
         · Update value function parameters ϕ using $\phi \leftarrow \phi - \alpha_\phi \nabla_\phi L^{V^F}(\phi)$.
2) **return** Optimized policy parameters θ.

Generalized Advantage Estimation (GAE): To improve stability, we use Generalized Advantage Estimation (GAE), which smooths out the variance of advantage estimates $\hat{A}_t$ over the trajectory. The advantage $\hat{A}_t$ at time t is defined as:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-1-t}\delta_{T-1} \tag{1}$$

where δ_t represents the temporal difference (TD) error at time t and is computed as:

$$\delta_t = R_t + \gamma V(s_{t+1};\phi) - V(s_t;\phi) \tag{2}$$

Here, γ is the discount factor (e.g., 0.99), which discounts future rewards to balance short-term and long-term gains. λ is a weighting parameter (typically 0.95) that allows for control between high bias and high variance. When $\lambda = 0$, GAE reduces to a standard TD estimate with lower variance; when $\lambda = 1$, it uses the full Monte Carlo estimate, which can have higher variance. The GAE method aggregates these temporal differences over multiple steps, helping smooth the advantage estimate, which provides more stable policy updates.

Return Calculation: The return G_t is calculated as the discounted sum of rewards from time step t onward in a trajectory. For a time horizon T, it is given by:

$$G_t = \sum_{k=0}^{T-1} \gamma^k R_k \tag{3}$$

where R_k is the reward received at time k, and γ is the discount factor (e.g., 0.99). This return G_t represents the cumulative reward that the agent expects to gain from time t until the end of the episode. By summing discounted rewards, it captures both immediate and future rewards, allowing the agent to make decisions that balance immediate gains with long-term benefits. This return G_t is also used as a target for updating the value function, providing a benchmark for how well the current policy performs.

Clipped Objective Function: To ensure stable training, the PPO algorithm employs a clipped objective function that prevents drastic changes to the policy during updates. The clipped surrogate objective $L_{\text{CLIP}}(\theta)$ is defined as:

$$L_{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{4}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio between the updated policy and the previous policy, and ϵ is a clipping parameter (e.g., 0.2) that limits how much the policy can change in a single update. The clipping operation, $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$, restricts $r_t(\theta)$ to lie within the range $[1-\epsilon, 1+\epsilon]$. This ensures that the policy update is not too aggressive, avoiding potential instability from large changes. By taking the minimum of $r_t(\theta)\hat{A}_t$ and the clipped value, PPO effectively controls policy updates, balancing exploration and exploitation.

Value Function Loss: The value function loss $L^{V^F}(\phi)$ is designed to minimize the prediction error between the estimated value $V(s_t;\phi)$ and the actual return G_t. It is defined as:

$$L^{V^F}(\phi) = \hat{\mathbb{E}}_t \left[\left(V(s_t;\phi) - G_t \right)^2 \right] \tag{5}$$

This loss function ensures that the value function $V(s;\phi)$ provides an accurate prediction of the expected return for each state. By minimizing the square error between the predicted and actual returns, the value function becomes a reliable measure of state quality, supporting effective advantage estimation in the GAE method. The value function is updated by adjusting parameters ϕ to minimize $L^{V^F}(\phi)$, leading to a more accurate baseline for advantage calculations, which improves the stability of policy updates in PPO.

IV. Understanding Why DRL Works in Robot Navigation

A. Problem setup

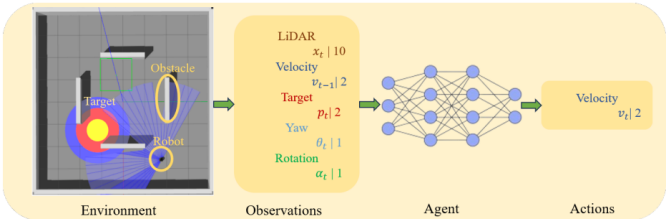

Fig. 1: Mapless motion robot trained with DRL agents for collision-free navigation, using observations that include LiDAR data x_t, previous robot velocity v_{t-1}, robot position p_t, yaw angle θ_t, and rotation angle α_t relative to the target, with each observation dimension indicated in the panel. The well-trained DRL agent provides an optimal policy for safely and efficiently guiding the robot to its target.

In our main reference [1], the DRL model was trained in virtual environments using the Robot Operating System (ROS), specifically ROS Neotic, and the Gazebo simulator, providing a realistic and customizable setting. Fig. 1 shows a 8×8 square meter indoor environment with strategically placed obstacles to challenge the robot's navigation. The Turtlebot3, as shown in Fig. 2, served as the robotic platform in the Gazebo-based experiments.

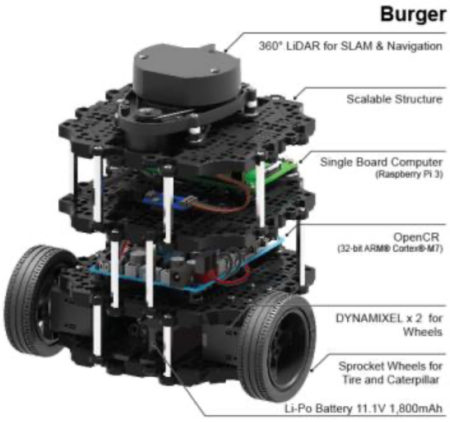

Fig. 2: The design and structure of the Turtlebot3 robot.

The mapless motion planner is shown in Fig. 1 to explore why DRL works in robot navigation, aiming to find the optimal policy:

$$v_t = f(x_t, p_t, v_{t-1}, \theta_t, \alpha_t), \tag{6}$$

where the robot's previous velocity v_{t-1} serves as the observation and the current velocity v_t as the action, both including linear and angular velocities as shown in Fig. 1. LiDAR data x_t, as shown in Fig. 3, providing 30-dimensional distance measurements, is condensed to 10 dimensions by grouping the measurements into 10 batches and selecting the minimum distance in each. Target information is encoded as position p_t, yaw angle θ_t, and rotation angle α_t between the robot and the target, aligning with the DDPG and PPO's Actor-Critic architecture. Thus, the observation dimension is 16, and the action dimension is 2, both defined within a continuous space.

The reward function in [1] is set as:

$$r(s_t, a_t) = \begin{cases} r_{\text{arrive}} & \text{if } d_t < c_d, \\ r_{\text{collision}} & \text{if } \max_{x_t} < c_o, \\ c_r\,(d_{t-1} - d_t) \quad \text{Otherwise.} & \text{otherwise.} \end{cases} \tag{7}$$

The reward r_{arrive} is granted when the agent is within a critical distance c_d of the target, encouraging goal-oriented movement. A penalty $r_{\text{collision}}$ is applied if any sensor reading x_t signals a near-collision distance c_o, ensuring safety. Otherwise, the reward is based on the reduction in distance to the target since the last timestep, incentivizing progress. The agent is penalized for moving toward walls and rewarded linearly as it nears the target, addressing challenges in obstacle-filled environments. Here, c_o is the collision threshold, c_d is the target proximity threshold, and c_r is the reward coefficient.

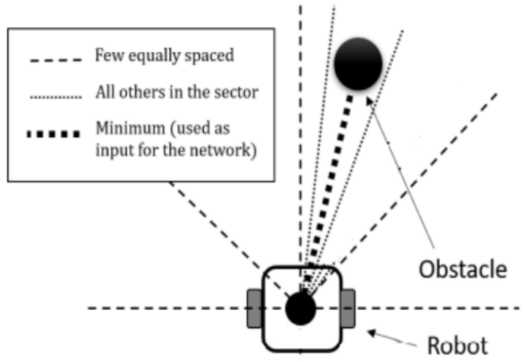

Fig. 3: Transforming raw sensor data into streamlined observations for effective decision-making.

B. Model architecture

Network	Layer	Input Dimensions	Output Dimensions
Actor	Linear (L_1)	`state_dim`	400
	Linear (L_2)	400	300
	Linear (L_3)	300	`action_dim - 1`
	Activation	ReLU (L_1,L_2), Sigmoid / Tanh (L_3)	
	Output	Concatenation of scaled Sigmoid and Tanh outputs	
Critic	Linear (L_1)	`state_dim + action_dim`	400
	Linear (L_2)	400	300
	Linear (L_3)	300	1
	Activation	ReLU (L_1, L_2)	
	Output	Single scalar value (Q-value)	

TABLE I: Parameters and structure of the Actor and Critic networks in the DDPG and PPO vanilla algorithm.

The DRL framework employed in this report involves independently solving the same problem using the DDPG and PPO algorithms. Each algorithm is implemented with an

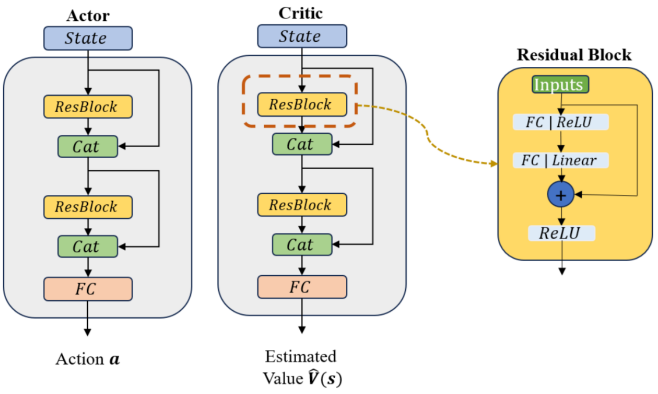

Fig. 4: Neural Network Architecture of the PPO Algorithm: detailing layer types, dimensions, activations, and a unified output through the Merge layer. Each fully connected (FC) layer has 512 neurons.

actor-critic architecture, as detailed in Section III. This dual approach allows for a comparative evaluation of the strengths and performance of each method. Traditionally, the actor and critic networks are implemented as fully connected neural networks, commonly referred to as multilayer perceptrons (MLPs). These networks are particularly suitable for DRL tasks, as DRL addresses Markov Decision Processes (MDPs), where the next state depends solely on the current state and action. The simplicity and efficiency of feedforward neural networks align well with this property of MDPs, making them a natural choice for the DDPG framework. The design and configuration of the DDPG and PPO vanilla networks are summarized in Table I.

Our main reference [1] designed a modified network architecture for the proposed PPO algorithm, replacing the standard feedforward network structure with Residual Blocks (ResBlocks) in both the actor and critic networks. ResBlocks are advanced neural network components designed to enhance information retention across layers, addressing challenges like gradient vanishing that often arise in deep networks. Each ResBlock operates as follows: the input is passed through two fully connected (FC) layers, with the first layer applying a ReLU activation function and the second layer omitting any activation. A skip connection bypasses these two layers, directly adding the original input to the output of the second FC layer. This combined output then passes through a final ReLU activation. The ResBlock structure helps preserve essential features from earlier layers while enabling the network to learn additional transformations, improving stability and overall performance in deep reinforcement learning tasks.

By applying the DDPG, the PPO vanilla, and the proposed PPO to the same problem, this report facilitates a comparative analysis of their performance and suitability for solving the task at hand. The distinct neural network architectures employed for each algorithm highlight their unique characteristics and capabilities, providing valuable insights into their respective advantages in reinforcement learning scenarios.

C. Performance comparison

Fig. 5: Comparison of cumulative rewards achieved by the proposed PPO (utilizing Residual Blocks), PPO Vanilla (employing MLP), and DDPG (employing MLP) in a simple 8x8 environment with a basic reward function. The "simple environment" refers to a setup where the four obstacles depicted in Fig.1 are removed, leaving only the wall boundary. In contrast, the "complex environment" corresponds to the full setup shown in Fig.1. Note here the horizontal axis "Steps" corresponds to the Episode in the main reference [1].

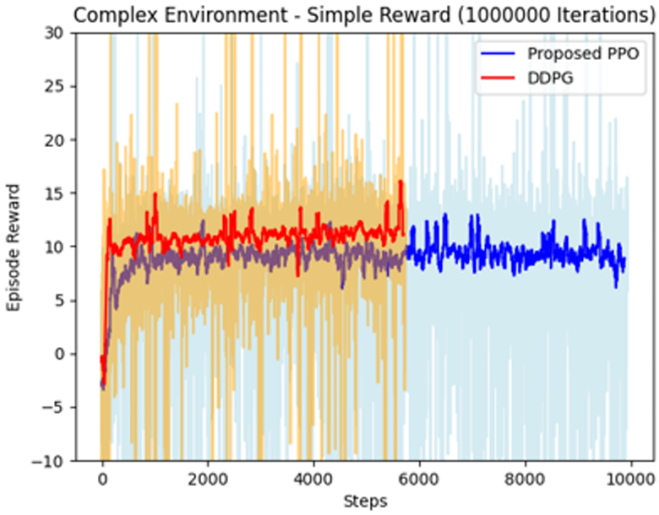

Fig. 6: Comparison of cumulative rewards achieved by the proposed PPO (utilizing Residual Blocks) and DDPG (employing MLP) in a complex 8x8 environment with a basic reward function.

The DRL agent, represented by the TurtleBot3 robot in Fig. 2, interacts with the Gazebo environment to accomplish

its task. The objective of the task is to reach the target without colliding with the walls. During the training phase, the robot is initialized at the origin point, which is the central point of the 8x8 environment shown in Fig. 1. The robot then generates a policy based on its DRL algorithm, collects observations and rewards, and performs actions. This iterative process is referred to as a rollout. If the robot crashes into a wall or reaches the maximum time step limit of $N = 800$, it is re-initialized at the origin point, marking the beginning of a new episode. However, if the robot successfully reaches the target, it is not re-initialized. Instead, a new target is generated, and the robot starts from its last position to search for and reach the newly generated target, effectively initiating a new episode. When the robot collects 8000 data points consisting of states, actions, and rewards, the DRL actor and critic networks are updated using a batch size of 8000 with 100 epochs for gradient descent. The training results are displayed in Fig. 5 and Fig. 6.

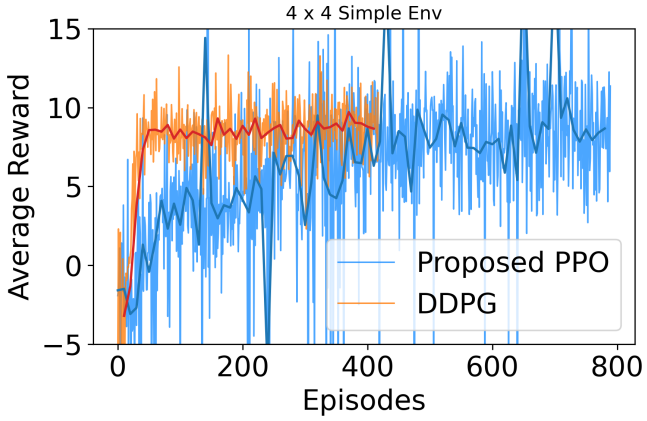

Fig. 7: Comparison of cumulative rewards achieved by the proposed PPO (utilizing Residual Blocks), and DDPG (employing MLP) in a simple 4x4 environment with a basic reward function.

In our numerical simulations, we observed that both the DDPG and PPO algorithms typically require around ten updates for their learning curves to converge in both simple and complex 4×4 environments. Furthermore, we found that the proposed PPO algorithm does not demonstrate a significant advantage over DDPG, as shown in Fig. 7 and Fig. 8. Firstly, the convergence speed of the proposed PPO is considerably slower compared to DDPG. To address this issue, we plan to explore the use of a larger learning rate (the current value is 10^{-3}) and reduce the number of neurons in each layer of the Residual Block (currently set to 512 neurons per layer). Secondly, the converged reward achieved by the proposed PPO is either similar to or slightly worse than that of DDPG in the 4×4 Gazebo environment, as illustrated in Fig. 7 and Fig. 8. However, in the main reference [1], their results from Fig. 5 and Fig. 6 indicate that the proposed PPO outperforms DDPG in the simple environment and performs slightly worse in the

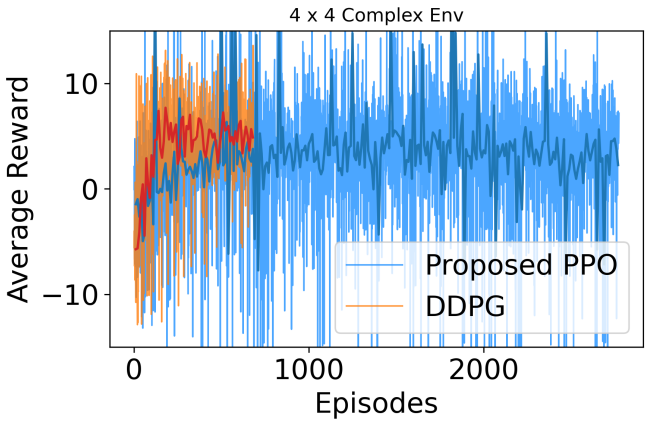

Fig. 8: Comparison of cumulative rewards achieved by the proposed PPO (utilizing Residual Blocks) and DDPG (employing MLP) in a complex 4x4 environment with a basic reward function.

complex environment.

Based on our understanding, the final converged reward values are determined by the balance between exploration and exploitation. The well-balanced trade-off between these two factors in both DDPG and PPO could significantly influence the final converged value. In this context, we plan to adjust the initial noise level in the stochastic policy of PPO to potentially improve the final converged reward.

The low-dimensional Gazebo environment simplifies the task for the DRL agent, making it challenging for advanced DRL algorithms to exhibit a clear advantage. On the other hand, in the 8×8 complex environment, the proposed PPO does not demonstrate any notable advantage. This suggests that the proposed PPO may not perform better in environments with more complex obstacle configurations.

V. Future Work

We will adjust the learning rate, the neural network size, and the initial noise level in the actor network to enhance the performance of our proposed PPO algorithm in both the simple and complex Gazebo environments.

References

Citation	Discussion Aspect
[1] H. Taheri and S. R. Hosseini, "Deep reinforcement learning with enhanced PPO for safe mobile robot navigation," arXiv preprint arXiv:2405.16266, 2024	We relied extensively on this paper across Sections I through V to develop the PPO agent for robot navigation.
[2] Z. Liu, Y. Zhai, J. Li, G. Wang, Y. Miao, and H. Wang, "Graph relational reinforcement learning for mobile robot navigation in large-scale crowded environments," IEEE Transactions on Intelligent Transportation Systems, 2023.	We use this paper to introduce Graph Relational Reinforcement Learning (GRRL) for robot navigation in Section I.
[3] Z. Wang, Y. Wang, Z. Wang, H. Yan, Z. Xu, and Z. Wu, "Research on autonomous robots navigation based on reinforcement learning," in 2024 3rd International Conference on Robotics, Artificial Intelligence and Intelligent Control (RAIIC). IEEE, 2024.	We use this paper to introduce Deep Q-Networks for robot navigation in Section I.