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 DRL Algorithms

A. 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.

B. Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) algorithm is a technique designed for environments with continuous action spaces. It combines concepts from actor-critic architecture and deterministic policy gradients by leveraging neural networks to approximate both policy and value functions:

- **Policy Function** $\mu(s|\theta^\mu)$: This is represented by the actor network parameterized by θ^μ, which maps states s to specific actions a.
- **Value Function** $Q(s, a|\theta^Q)$: This is represented by the critic network parameterized by θ^Q, which evaluates the quality of action a taken in state s.

Both networks are trained simultaneously to optimize the agent's performance.

To ensure stability during training, DDPG employs target networks:

- **Target Actor Network** $\mu'(s|\theta^{\mu'})$
- **Target Critic Network** $Q'(s, a|\theta^{Q'})$

These target networks are time-delayed copies of the original networks, updated more slowly to avoid drastic policy and value shifts. Initially, the target networks are identical to their counterparts:

$$\theta^{\mu'} \leftarrow \theta^\mu, \quad \theta^{Q'} \leftarrow \theta^Q$$

The updates for the target networks follow a soft-update mechanism:

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$
$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

where τ is a small constant (e.g., $\tau = 0.001$).

The training process involves interactions between the environment, agent (actor and critic networks), and a replay buffer that stores past experiences.

- At time step t, the agent observes the current state s_t and selects an action using the actor network with added noise for exploration:

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$$

  where $\mathcal{N}_t$ represents Ornstein-Uhlenbeck noise.
- The action a_t is executed in the environment, resulting in a new state s_{t+1} and reward r_t.
- The experience tuple (s_t, a_t, r_t, s_{t+1}) is stored in the replay buffer.
- **Critic Network Update:** The critic network minimizes the loss function:

$$L(\theta^Q) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$$

  where the target value y_i is computed as:

$$y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$$

- **Actor Network Update:** The actor network is updated by maximizing the expected return using policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s_i, a|\theta^Q)\Big|_{a=\mu(s_i)} \nabla_{\theta^\mu}\mu(s_i|\theta^\mu)$$

C. Pros and Cons of DDPG and PPO

DDPG is highly effective for tasks involving continuous action spaces, such as robotics and autonomous systems, due to its deterministic policy and sample-efficient off-policy updates. By leveraging experience replay and target networks, DDPG achieves high precision and efficiency, making it ideal for control tasks requiring fine-grained actions. However, its reliance on external noise for exploration and sensitivity to hyperparameter tuning can hinder performance in environments requiring extensive exploration. Additionally, DDPG can suffer from training instability due to overestimation bias in Q-value approximation.

PPO, on the other hand, is a versatile and stable reinforcement learning algorithm suited for both discrete and continuous action spaces. Its clipped objective function ensures stability during training, making it more robust and easier to tune compared to DDPG. PPO promotes natural exploration without requiring additional noise, making it effective for a wide range of environments, including games and simulations. However, its on-policy nature results in lower sample efficiency and higher computational costs, as it requires frequent interactions with the environment. While DDPG is more specialized for precise control in continuous domains, PPO's stability and generality make it a strong choice for broader applications.

IV. Understanding Why DRL Works in Robot Navigation

A. Problem setup

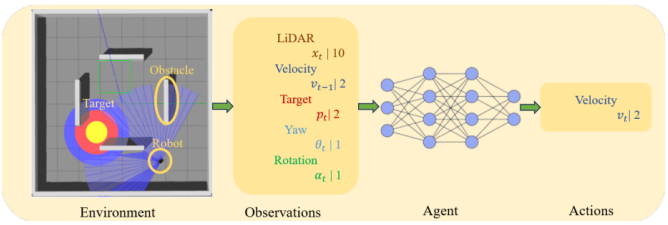


Fig. 1: Mapless motion robot trained with DRL agents for collision-free navigation, using observations that include LiDAR data x_t, previous robot velocity v_{t-1}, robot position p_t, yaw angle θ_t, and rotation angle α_t relative to the target, with each observation dimension indicated in the panel. 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 Noetic, and the Gazebo simulator, which together offered a highly realistic and flexible simulation framework. This combination enabled precise control over the virtual environment's physics and robot dynamics, ensuring a close approximation to real-world scenarios. As depicted in Fig.1, the training environment consisted of an 8×8 square meter indoor space populated with strategically positioned obstacles to test and enhance the robot's navigation capabilities.

These obstacles were designed to simulate common indoor challenges, such as narrow passages and dynamic barriers, ensuring robust learning. The Turtlebot3 robot, illustrated in Fig.2, was chosen as the experimental platform due to its popularity in academic research and compatibility with ROS and Gazebo. This setup provided a balance between simplicity and effectiveness, allowing the DRL model to be validated in a realistic yet controlled environment.

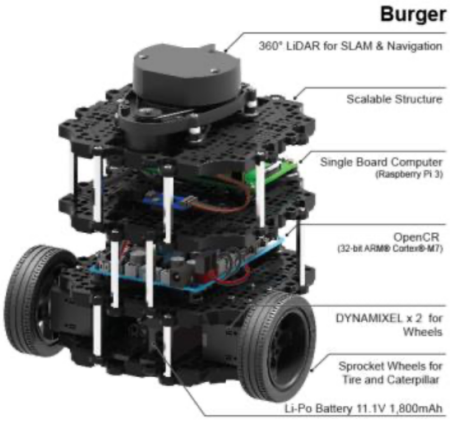


Fig. 2: The design and structure of the Turtlebot3 robot.

The mapless motion planner, illustrated in Fig.1, serves as a practical framework to explore the effectiveness of DRL in robot navigation by seeking the optimal policy for motion control. The policy is defined mathematically as:

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

where the robot's state and environment data are mapped to an appropriate velocity action. The robot's previous velocity v_{t-1} encompassing linear and angular components, is used as part of the observation, while the current velocity v_t forms the action, facilitating seamless decision-making across time steps. As shown in Fig.3, the LiDAR sensor provides 30-dimensional distance measurements, which are processed into a more compact 10-dimensional representation by grouping measurements into batches and selecting the minimum value within each batch. This condensed representation enhances computational efficiency while retaining critical environmental features. The target's position p_t, yaw angle θ_t, and rotation angle α_t relative to the robot provide additional key observations to guide navigation. These features align with the Actor-Critic framework of DDPG and PPO, leveraging a 16-dimensional observation space and a 2-dimensional action space, both continuous, to ensure precise and adaptable navigation in dynamic environments.

The reward function in [1] is designed to effectively guide the agent toward its target while avoiding obstacles, and balancing safety and goal-oriented behavior. The reward $r(s_t, a_t)$ is defined as:

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

Here, the reward r_{arrive} is a large positive reward granted, typically $r_{\text{arrive}} = 100$, when the agent reaches within a critical distance c_d of the target, typically $c_d \in [0.1, 0.2]$, directly reinforcing successful navigation. Conversely, a significant negative reward $r_{\text{collision}}$ is issued, typically $r_{\text{collision}} = -120$, if the maximum sensor reading x_t indicates a distance less than c_o, typically $c_o \in [0.1, 0.2]$, representing a collision or imminent collision. For all other scenarios, the agent receives a reward proportional to $c_r\left(d_{t-1} - d_t\right)$ with $c_r = 500$, reflecting the reduction in the distance d_t to the target since the previous time step d_{t-1}. This encourages steady progress toward the goal and penalizes regression or movement that reduces proximity to the target. Here, c_o is named as the collision threshold, c_d is the target proximity threshold, and c_r is the reward coefficient. The parameters r_{arrive}, $r_{\text{collision}}$, c_o, c_d, and c_r control the sensitivity of the reward system to collisions, target proximity, and progress, ensuring the agent learns to navigate effectively in complex environments. By penalizing collision risks and rewarding forward progress, this reward function addresses key challenges in environments with obstacles and dynamic constraints.

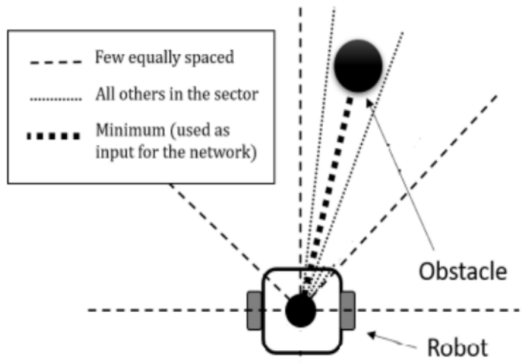


Fig. 3: Transforming raw sensor data into streamlined observations for effective decision-making.

B. Model architecture

The DRL framework utilized in this report implements the DDPG and PPO algorithms independently to address the same navigation problem, providing a basis for comparative analysis of their respective advantages and performance. Both algorithms adopt an actor-critic architecture, as outlined in Section III, where the actor is responsible for policy determination, and the critic evaluates the quality of actions taken.

The networks of DDPG and PPO Vanilla are structured as multilayer perceptrons (MLPs), which consist of fully connected layers designed to process state-action pairs efficiently.

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.

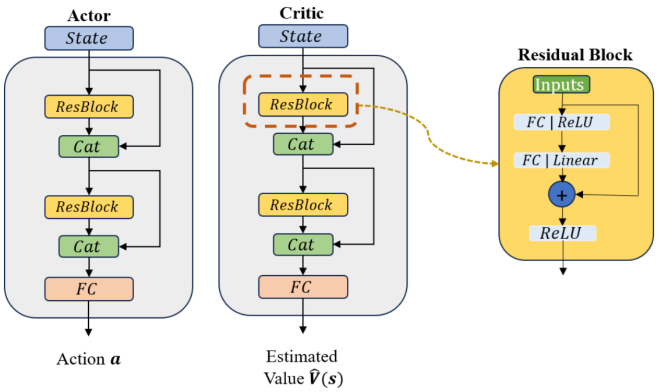


Fig. 4: Neural network architecture of the Proposed PPO algorithm: detailing layer types, dimensions, activations, and a unified output through the Merge layer. Only one fully connected hidden layer in each ResBlock containing 512 neurons, as described in [1]. The notation "FC" means a full connection map. So two "FCs" mean only one hidden layer.

This design aligns seamlessly with the Markov Decision Process (MDP) formulation, where the system's dynamics depend only on the current state and action, ensuring simplicity and computational efficiency. For DDPG, the deterministic nature of the policy benefits from the direct mapping capabilities of feedforward networks, while for PPO, the stochastic policy's adaptability is enhanced through clipped objective functions. The neural network configurations for both algorithms, including the number of layers, neurons, activation functions, and optimizer settings, are summarized in Table I, highlighting the architectural choices tailored to maximize the algorithms' performance in the navigation task.

In our main reference [1], the standard feedforward network structure in the PPO algorithm was replaced with a modified architecture incorporating Residual Blocks (ResBlocks) in both the actor and critic networks. This innovation aimed to address common issues in deep networks, such as gradient vanishing and degradation of performance as the network depth increases. ResBlocks are designed to facilitate efficient information flow by introducing skip connections that bypass intermediate layers, allowing the network to retain and refine key features from earlier stages. Specifically, each ResBlock processes its input through two fully connected (FC) layers: the first applies a ReLU activation to introduce non-linearity, while the second omits activation, ensuring minimal distortion of the data. The output of the second FC layer is then combined with the original input via a skip connection, and the resulting sum is passed through a final ReLU activation to produce the block's output. This architectural enhancement enables the network to learn complex transformations while preserving the integrity of important features, leading to improved stability and learning efficiency in reinforcement learning tasks. By incorporating ResBlocks, the modified PPO framework demonstrated enhanced performance and robustness, particularly in environments requiring deep policy representation and precise gradient updates.

By implementing DDPG, PPO vanilla, and the Proposed PPO with ResBlock-based architecture on the same problem, this report enables a comprehensive comparative analysis of these algorithms' performance, adaptability, and suitability for complex navigation tasks. The DDPG approach, with its deterministic policy and fully connected feedforward networks, is well-suited for continuous action spaces, emphasizing precision in action selection. Conversely, the vanilla PPO, leveraging a stochastic policy with simpler MLP-based actor-critic networks, excels in balancing exploration and exploitation, making it robust in dynamic and noisy environments. The Proposed PPO, enhanced with Residual Blocks, further builds on these strengths by addressing limitations like gradient vanishing and learning inefficiencies in deeper networks. This architectural innovation not only preserves critical features across layers but also facilitates the learning of more expressive policies, contributing to better stability and faster convergence. By contrasting these methodologies and their architectural nuances, the report sheds light on the trade-offs between deterministic and stochastic policies, as well as the impact of network complexity on the effectiveness of reinforcement learning in environments with varying challenges and constraints.

C. Performance comparison

The DRL agent, represented by the TurtleBot3 robot in Fig. 2, interacts dynamically with the Gazebo simulation environment to achieve the navigation task of reaching a target while avoiding collisions with walls. This task is carried out in an 8×8 square-meter indoor environment, as illustrated in Fig. 1, where the robot is initialized at the origin, the central point of the environment, at the start of each training episode. The agent generates policies using its DRL algorithm, leveraging observations, rewards, and actions in a continuous feedback loop referred to as a rollout. Episodes terminate when the robot either collides with a wall or reaches the maximum time step limit of $N = 800$, at which point it is reset to the origin. Upon successfully reaching the target, instead of resetting, the robot continues by starting from its last position, and a new target is generated, effectively creating a seamless transition to the next episode. This method ensures continuous exploration of the environment and enhances the

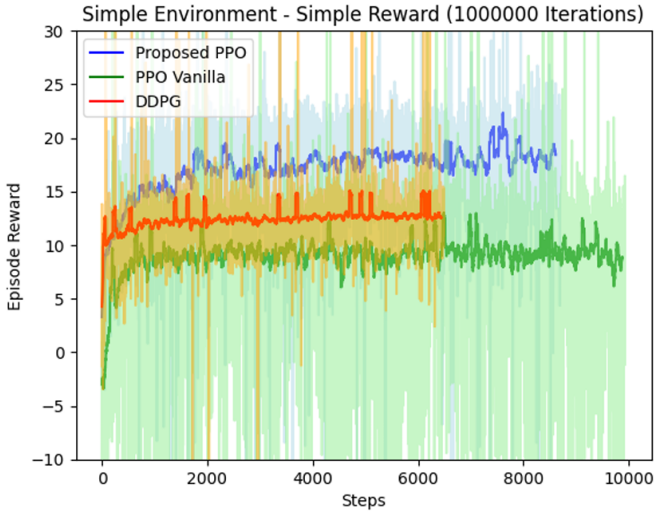


Fig. 5: Comparison of cumulative rewards achieved by the Proposed PPO (utilizing Residual Blocks), PPO Vanilla (employing MLP), and DDPG (employing MLP) in a simple 8x8 environment with a basic reward function. The "simple environment" refers to a setup where the four obstacles depicted in Fig.1 are 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].

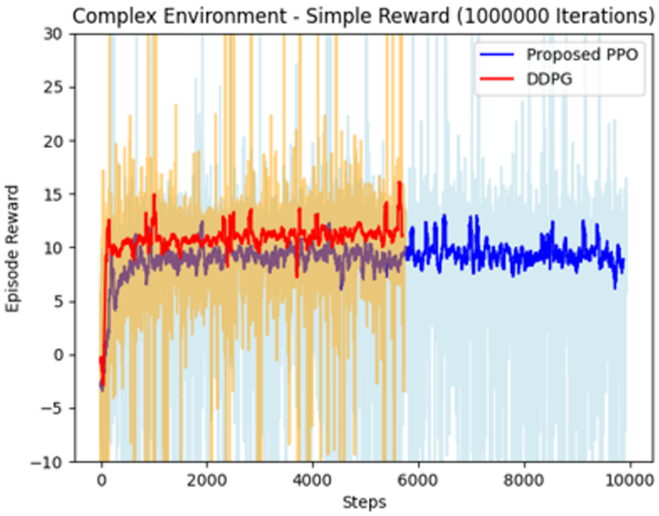


Fig. 6: Comparison of cumulative rewards achieved by the Proposed PPO (utilizing Residual Blocks) and DDPG (employing MLP) in a complex 8x8 environment with a basic reward function.

learning process. During training, the robot collects 8000 data points comprising states, actions, and rewards, which are used to update the actor and critic networks of the DRL model over 100 epochs with a batch size of 8000, utilizing the Adam optimizer for efficient gradient descent. he impact of the training approach is evaluated using results from Fig.5 and Fig.6, as presented in our main reference [1], which illustrate the robot's performance in both simple and complex environments. Fig. 5 depicts cumulative rewards versus episodes (referred to as Steps in [1]) for the Proposed PPO, PPO Vanilla, and DDPG algorithms. In the simple environment, PPO Vanilla exhibits the worst performance, with low cumulative rewards and the highest variance, indicating unstable behavior. In contrast, the Proposed PPO converges to the highest cumulative reward, demonstrating superior learning and stability. However, in the complex environment, while the Proposed PPO achieves more stable performance with lower variance, its cumulative reward slightly trails behind that of DDPG, highlighting a trade-off between stability and peak performance in challenging scenarios.

DDPG excels in high-dimensional continuous action spaces, leveraging its off-policy design for efficient sample reuse but struggles with exploration and requires careful tuning to address stability issues like overestimation bias. PPO provides stable updates through its clipped objective function, making it versatile for both discrete and continuous action spaces. However, its on-policy nature reduces sample efficiency, and its multiple gradient updates per step increase computational demands, especially in complex environments.

V. Conclusions

We obtained simulation results for DDPG and the Proposed PPO in simple and complex 4×4 environments, as detailed in Sec VI. Next, we plan to refine the learning rate, ResBlock neural network structure, and initial noise level in the actor-network to enhance the Proposed PPO's performance in Gazebo environments.

Kannak Sharma contributed Sections I and II with two slides (1/3 of the work), Zeming Zhou handled Section III with two slides (1/3), and Lili Ye prepared Section IV and the References table with eight slides (1/3).

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.

VI. Simulation Results

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

In our numerical simulations, we observed that both the DDPG and Proposed PPO algorithms typically required around ten updates for their learning curves to converge in both simple and complex 4×4 environments. However, the Proposed PPO did not exhibit a clear advantage over DDPG in these scenarios, as shown in Fig. 8 and Fig. 9. Notably, the convergence speed of the Proposed PPO was slower compared to DDPG, highlighting a potential inefficiency in its current configuration. To address this, we propose increasing the learning rate (currently set to 10^{-3}) and reducing the number of neurons in each layer of the Residual Block (presently set at 512 neurons per layer) to improve training efficiency. Furthermore, the final converged reward of the Proposed PPO was observed to be comparable to or slightly worse than that of DDPG in the 4×4 Gazebo environment. This result contrasts with findings from the main reference [1], where the Proposed PPO outperformed DDPG in a simple 8×8 environment and performed slightly worse in the complex 8×8 environment, as shown in Fig. 5 and Fig. 6. These discrepancies suggest that the performance of the Proposed PPO might be sensitive to environment complexity and scale, warranting further investigation into its parameterization and architecture.

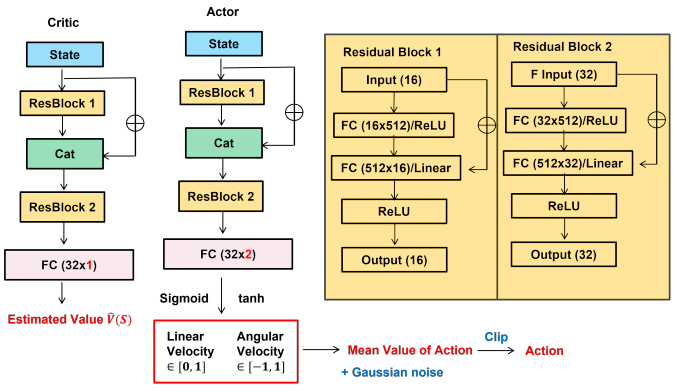


Fig. 7: The updated neural network architecture of the Proposed PPO algorithm in detail. Here the second Cat structure has been removed.

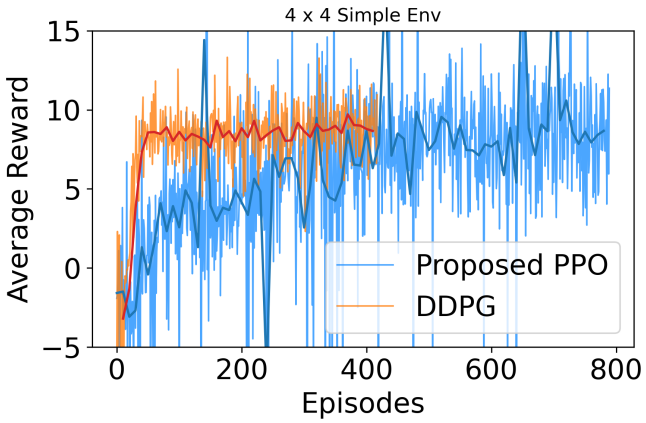


Fig. 8: Comparison of cumulative rewards achieved by the Proposed PPO (utilizing Residual Blocks), and DDPG (employing MLP) in a simple 4x4 environment with a basic reward function.

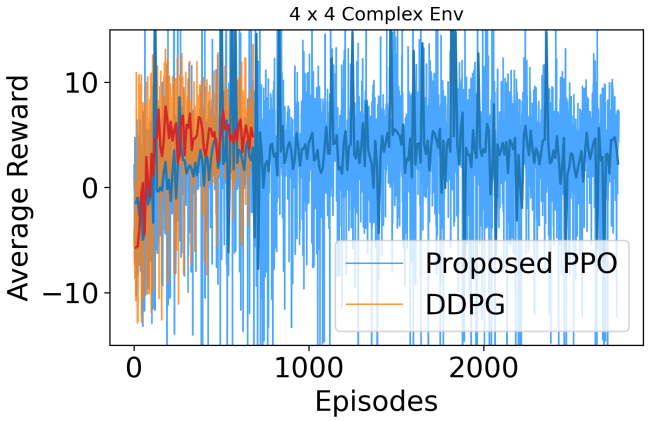


Fig. 9: Comparison of cumulative rewards achieved by the Proposed PPO (utilizing Residual Blocks) and DDPG (employing MLP) in a complex 4x4 environment with a basic reward function.

The final converged reward values in reinforcement learning are largely influenced by the delicate balance between exploration and exploitation, as this trade-off determines how effectively the agent discovers optimal policies while refining its actions. Both DDPG and PPO rely on mechanisms to achieve this balance—DDPG using noise-based exploration and PPO utilizing its stochastic policy structure. A well-calibrated trade-off ensures that the agent explores enough to avoid suboptimal policies while exploiting known strategies to maximize cumulative rewards. To improve the final converged reward in PPO, we propose adjusting the initial noise level in its stochastic policy. By fine-tuning the noise parameters, we aim to enhance the agent's exploration during the early training stages, enabling it to better discover high-reward trajectories. Simultaneously, this adjustment should maintain

a controlled transition to exploitation, preventing instability or over-exploration that might hinder convergence. This modification is expected to address observed limitations in PPO's performance and further optimize the learning outcomes.

The low-dimensional 4×4 Gazebo environment simplifies the navigation task, reducing the complexity of decision-making for the DRL agent. As a result, advanced DRL algorithms, such as the Proposed PPO, struggle to showcase a significant performance advantage over simpler methods, as the task does not demand their full capabilities. Conversely, in the more challenging 8×8 complex environment, the Proposed PPO also fails to demonstrate any notable superiority. This observation suggests that the Proposed PPO may not effectively handle environments with intricate obstacle configurations, potentially due to limitations in its architecture or hyperparameter settings. These findings highlight the need for further tuning or architectural improvements to enhance the algorithm's adaptability and performance in both simple and complex environments.

VII. Simulation Details and contributions

Here are Lili Ye's contributions:

- Created a Dockerfile to install ROS Noetic, Ubuntu 20.04, Python 3.8.10, and PyTorch 1.10, and used it to build the corresponding container image.
- Modified the target file to reduce the target size by half, located at `turtlebot3_simulations/turtlebot3_gazebo` directory: `/models/Target/model.sdf`.
- Modified the world file to change the 8×8 environment into a 4×4 environment, located at `turtlebot3_simulations/turtlebot3_gazebo` directory: `/worlds/train_world1.world` and `train_world_new.world`.
- Debugged the "roslaunch project ppo_stage_1.launch" error by renaming `main.py` to `ppo_stage_1.launch`, located in `project/src`.
- Debugged the training part and modified the hyperparameters in `ppo_stage_1.py`.
- Adjusted `threshold_arrive` and normalization constants in `environment_new.py`.
- Modified the output directories for models, records, and summaries in `ppo.py`, `ppo_stage_1.py`, and `environment_new.py`.
- Trained the Proposed PPO in both simple and complex environments, recorded two videos, and created corresponding GIFs.
- Updated `ddpg.py` and `ddpg_stage_1.py`.
- Fixed exploration noise in `ddpg_stage_1.py` to resolve the issue where the robot rotated in place without moving toward targets.
- Modified the DDPG training process to ensure updates only occur when the batch size equals 8000, consistent with PPO. Previously, DDPG updated at every episode. Changes made in `ddpg_stage_1.py`.
- Adjusted `threshold_arrive` and normalization constants in states within `environment.py`.
- Modified the output directories for records and models in `ddpg_stage_1.py` and `environment.py`.
- Trained the DDPG in both simple and complex environments.
- Plotted Fig. 8 and Fig. 9.
- Finished the Sec. VI.
- Fixed additional small bugs and uploaded all modified files to GitHub.
- Provided suggestions to the team to adjust the learning rate, neural network size, and initial noise level in the actor network to improve the Proposed PPO's performance in both Gazebo environments, as discussed in Sec. VI.

EEE 598, Fall 2024

Robot Navigation by Reinforcement Learning

Final project by Team 10

Kannak Sharma, Lili Ye, and Zeming Zhou

School of Electrical, Computer, and Energy Engineering,
Arizona State University
Contacts: [ksharm88, liliye, zzhou157]@asu.edu

Arizona State University

What is the issue/prompt under discussion

Objective

- Develop adaptable, collision-free navigation for robots in unstructured settings using RL and PPO algorithms.

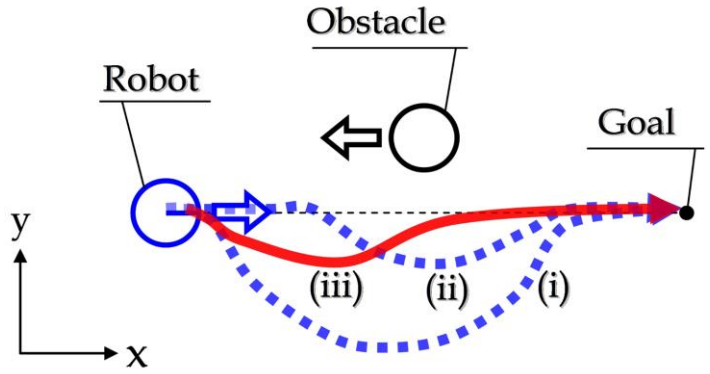


Fig. A situation of obstacle avoidance
Source: link

Key Challenges in Modern Navigation

- **Obstacle Avoidance**: Navigate around static (e.g., walls) and dynamic obstacles (e.g., people, other robots).
- **Real-Time Decision Making**: Rapidly process continuous data for immediate response.
- **Limited Sensory Data**: Operate with incomplete view of surroundings using local sensors (e.g., LiDAR).
- **Adaptability**: Generalize navigation strategies across various environments without reprogramming.

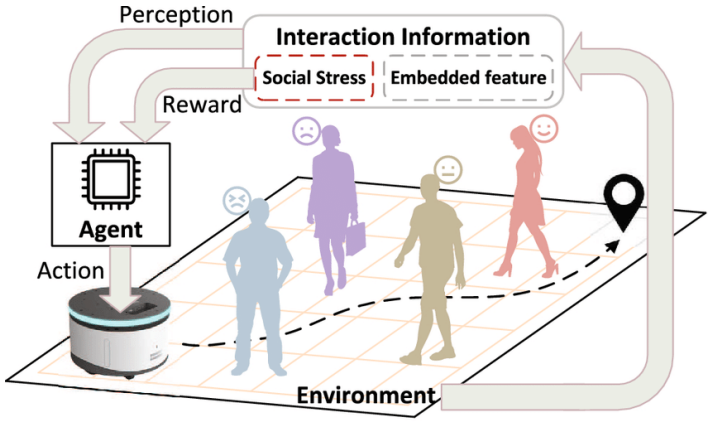


Fig. Robot navigation in the complex dynamic environment with multi-human interaction. Source: link

What is the issue/prompt under discussion

Why Traditional Methods Fall Short

- Lack of **flexibility** to handle unexpected obstacles or changes in environment layout.

  1. Dynamic Obstacles 2. Reactive Navigation Challenges 3. Real-Time Data Limitations

- Dependence on **predefined maps** makes traditional methods unsuitable for unstructured and dynamic settings.

  1. Static Nature 2. Unstructured Settings 3. Limited Scalability

Our Approach

- **Proximal Policy Optimization (PPO)**: Reliable for continuous action spaces, ideal for dynamic navigation.
- **Actor-Critic Model**: Enhances decision-making by evaluating actions for a balanced, adaptive response.
- **LiDAR Integration**: Enables real-time adjustments without reliance on pre-defined maps.

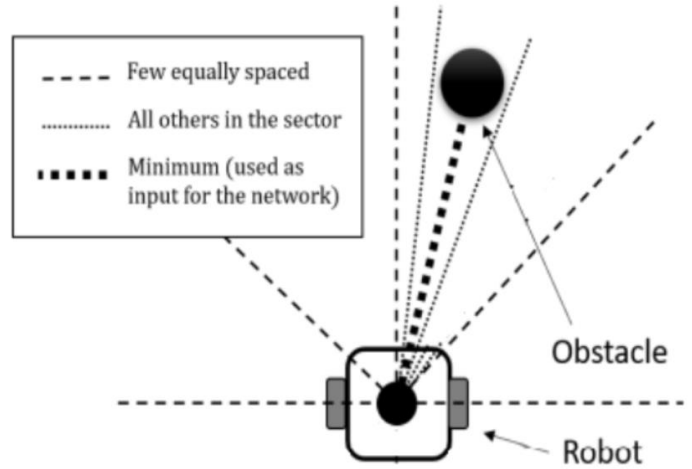


Fig. LiDAR-based obstacle detection: minimum distance in sector used as input for PPO network

PPO Algorithm - Key Components and Formulas

1.Generalized Advantage Estimation (GAE)

- To improve stability, GAE smooths the advantage estimates $\hat{A}_t$:

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

- Temporal difference δ_t:

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

2. Return Calculation

- Sum of discounted rewards:

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

3. Clipped Surrogate Objective

- Controls the update size to ensure stability:

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

4. Value Function Loss

- Minimizes the prediction error between the value function and actual returns:

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

Deep Deterministic Policy Gradient (DDPG)

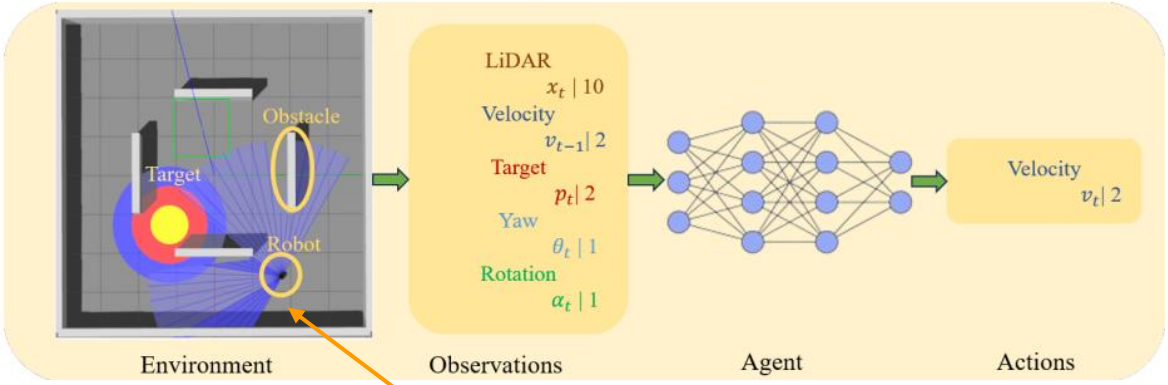
Key Mechanisms

1. **Target Networks**:
   - Includes Target Actor (μ') and Target Critic (Q').
   - Updated slowly using a soft update rule for stability: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
2. **Replay Buffer**:
   - Stores past experiences (s_t, a_t, r_t, s_{t+1}) to break data correlations.
   - Ensures efficient use of data through random sampling.

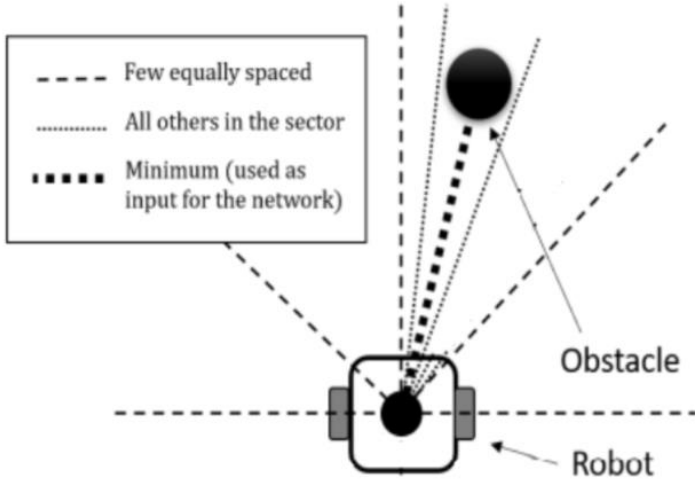
Training Process

- **Critic Network Update:**
  Minimizes the loss function to reduce Bellman error: $L(\theta^Q) = \frac{1}{N}\sum_{i=1}^{N}(y_i - Q(s_i, a_i|\theta^Q))^2$
  where $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'}))$.
- **Actor Network Update:**
  Maximizes the expected return by using the gradient of the Critic: $\nabla_{\theta^\mu} J \approx \frac{1}{N}\sum_{i=1}^{N} \nabla_a Q(s, a|\theta^Q)|_{a=\mu(s)} \nabla_{\theta^\mu}\mu(s|\theta^\mu)$

Problem Setup

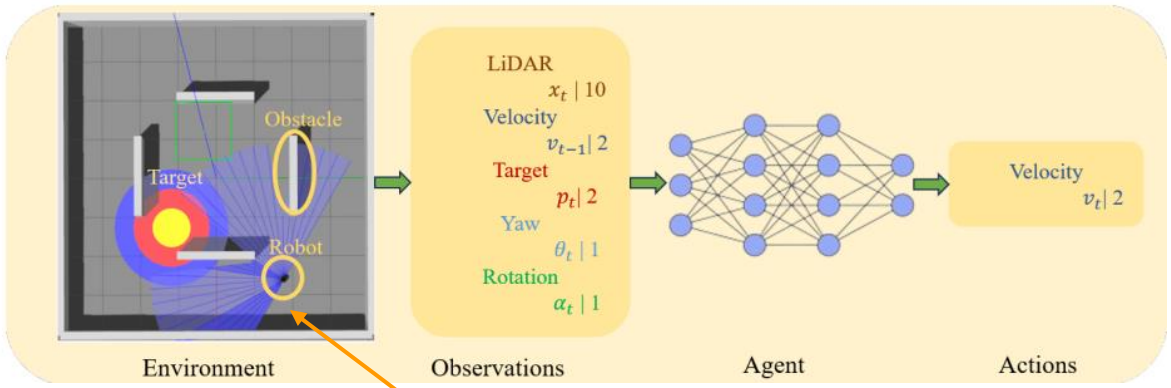


$$v_t = f(x_t, p_t, v_{t-1}, \theta_t, \alpha_t)$$



H. Taheri and S. R. Hosseini, “Deep re-inforcement learning with enhanced PPO forsafe mobile robot navigation,” arXiv preprintarXiv:2405.16266, 2024

Problem Setup



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

$r_{arrive} = 120$

$r_{collision} = -100$

$c_r = 500$

c_o: Collision threshold
c_d: Target proximity threshold
c_r: Penalty coefficient

Training details:

Episodic training: When collision or max time step = 800 or arrive the target

Batch size = 8000 time steps

Epochs = 100 steps with Adam optimizer for gradient descent

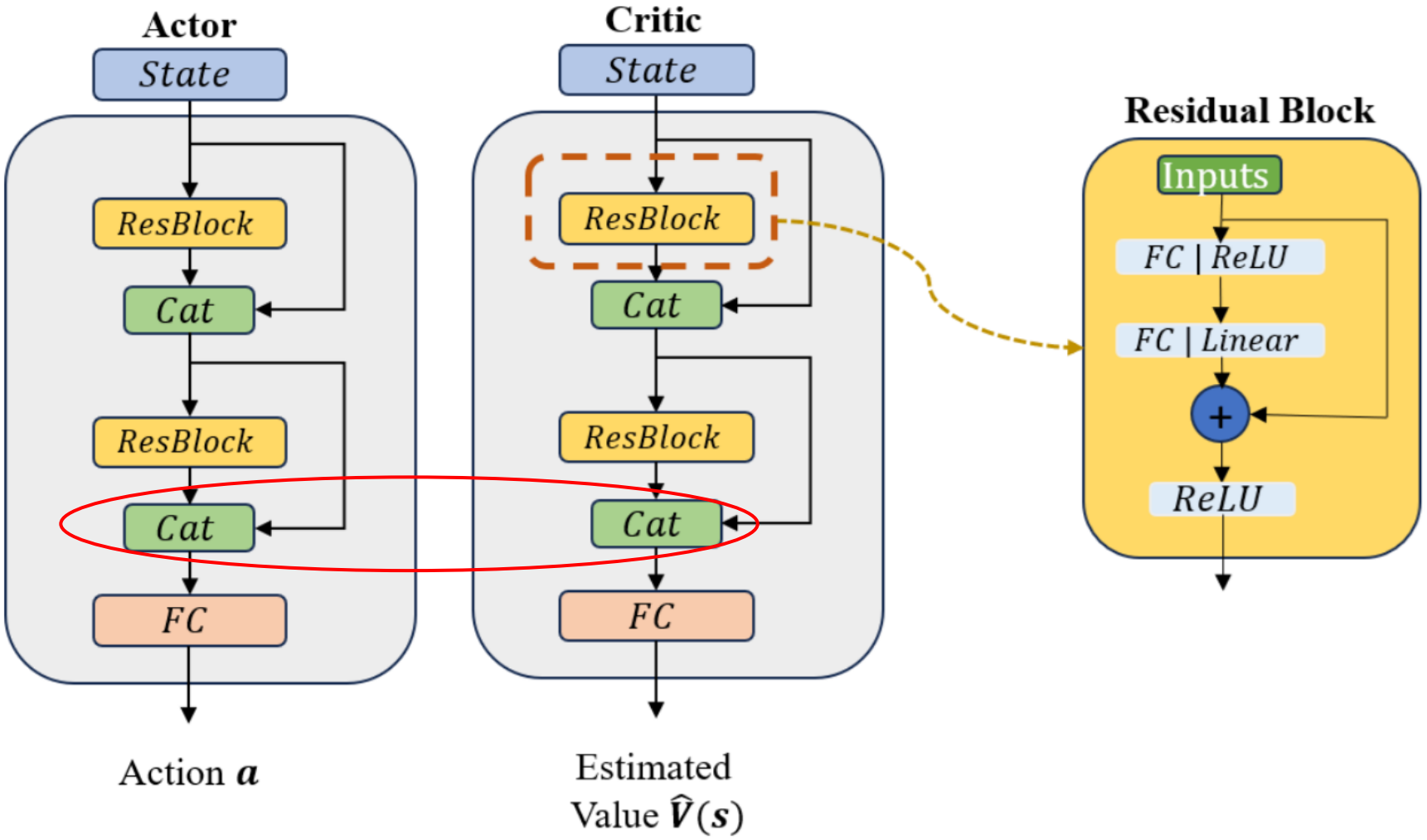
H. Taheri and S. R. Hosseini, “Deep re-inforcement learning with enhanced PPO forsafe mobile robot navigation,” arXiv preprintarXiv:2405.16266, 2024

Model Architecture: DDPG and PPO Vanilla in Code

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)	

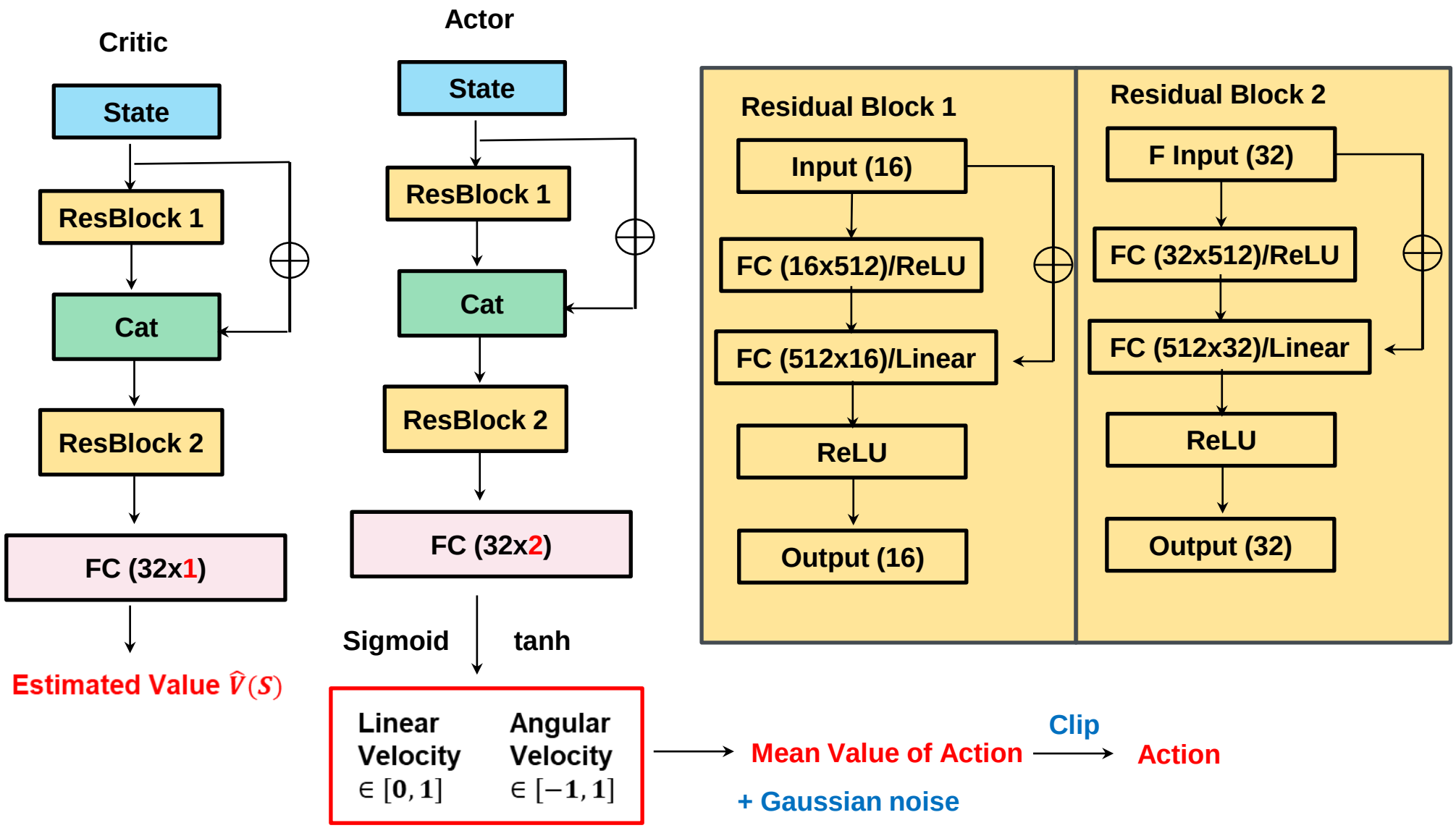
TABLE : Parameters and structure of the Actor and Critic networks in the DDPG and PPO vanilla algorithm.

Model architecture: Proposed PPO in Paper

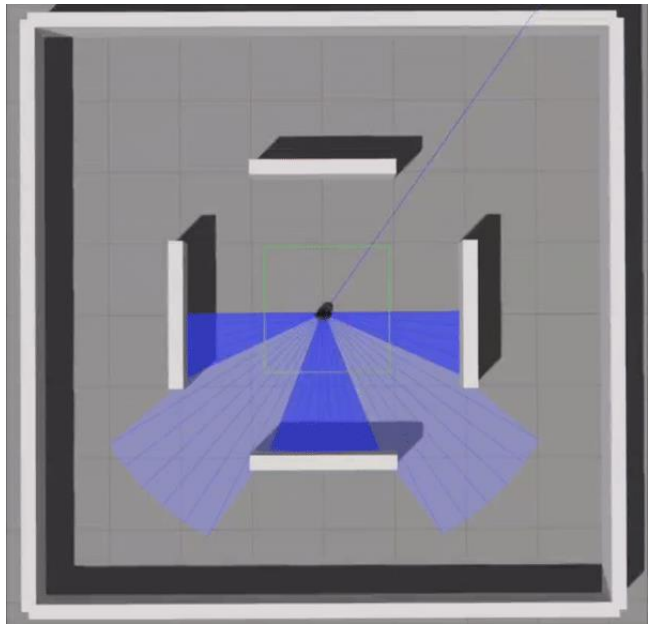


H. Taheri and S. R. Hosseini, “Deep re-inforcement learning with enhanced PPO forsafe mobile robot navigation,” arXiv preprintarXiv:2405.16266, 2024

Model architecture: Proposed PPO in Code

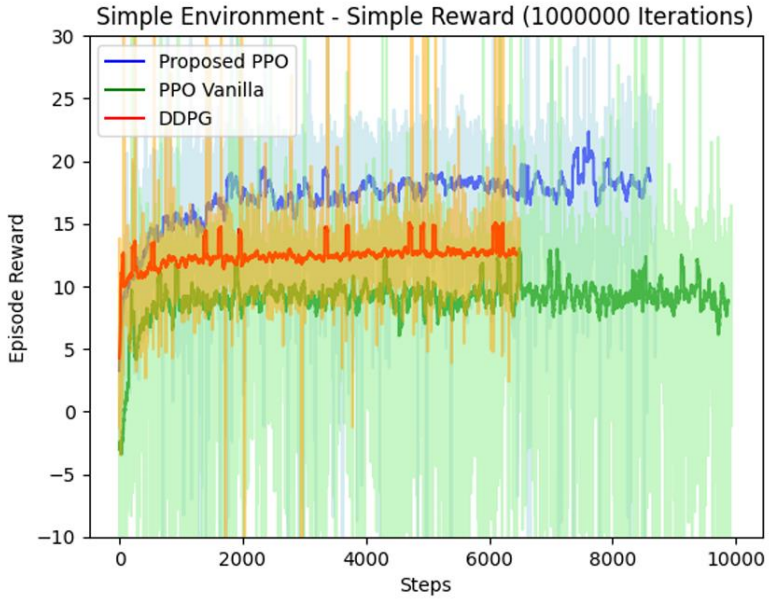


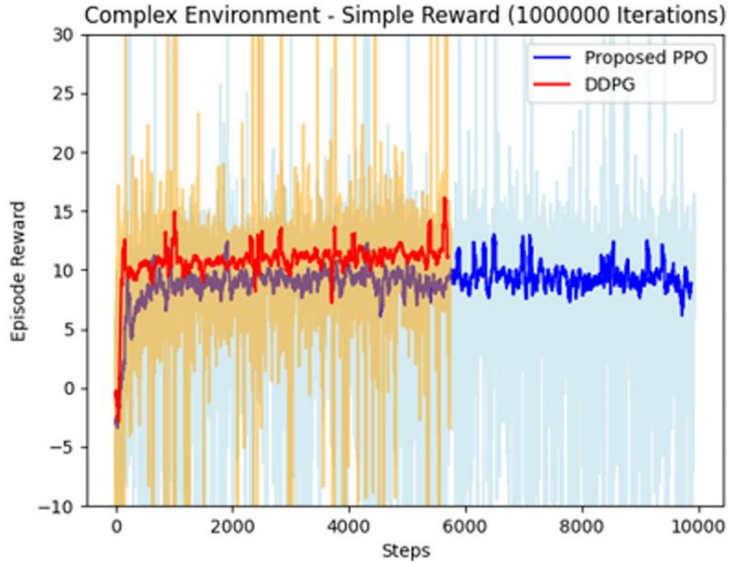
Performance in Paper



.gif

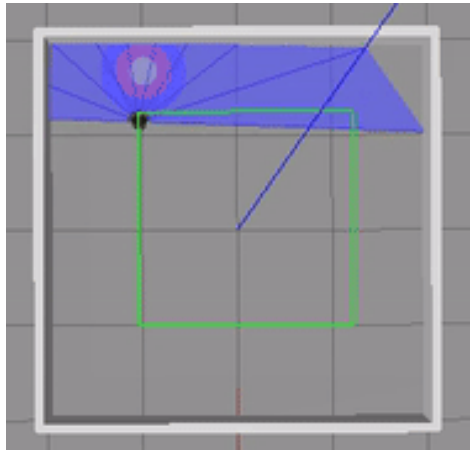
https://github.com/hamidthri/navbot_ppo

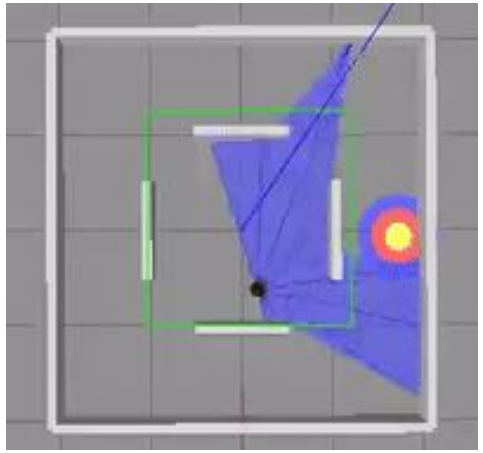




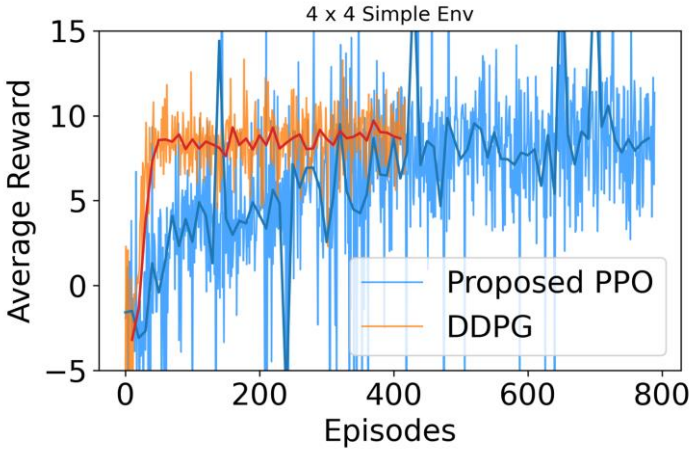
H. Taheri and S. R. Hosseini, “Deep re-inforcement learning with enhanced PPO forsafe mobile robot navigation,” arXiv preprintarXiv:2405.16266, 2024

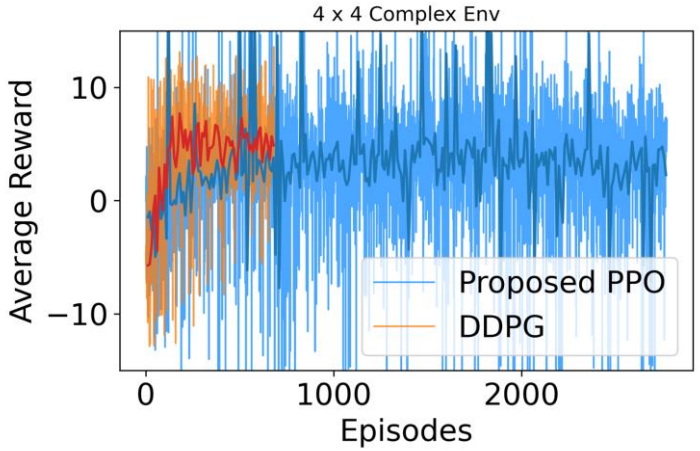
Performance in Code

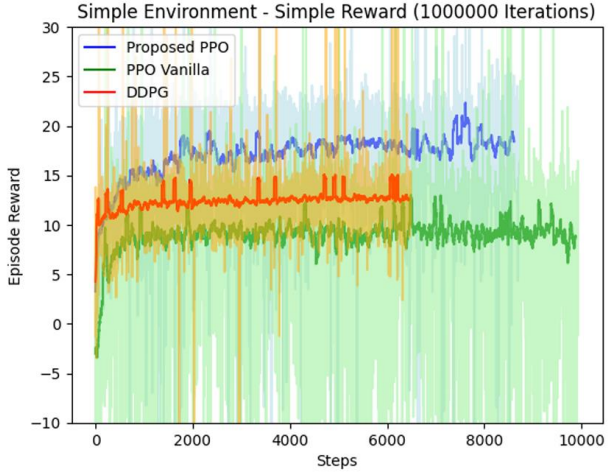


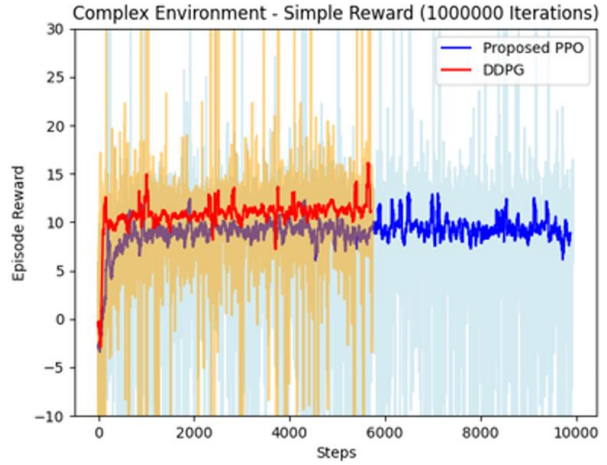


.gif





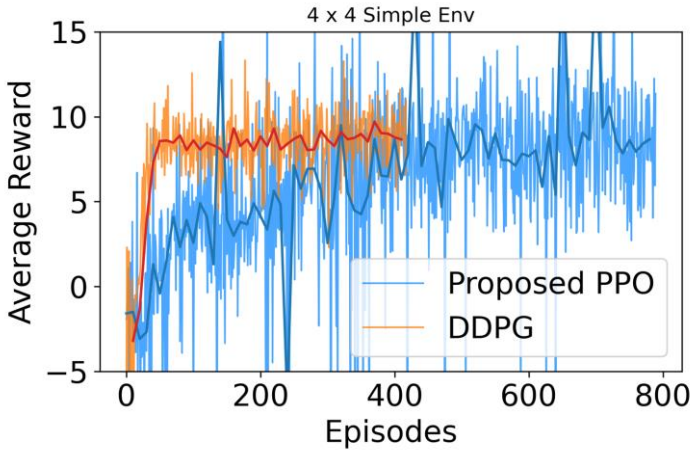


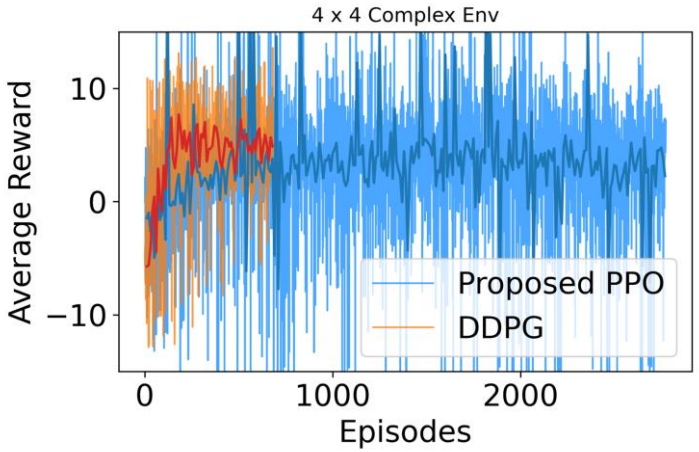


https://github.com/liliyequantum/RL_in_robotics

H. Taheri and S. R. Hosseini, “Deep re-inforcement learning with enhanced PPO forsafe mobile robot navigation,” arXiv preprintarXiv:2405.16266, 2024

Future Work





Q1: The proposed PPO shows the slower convergence speed compared to DDPG

A1: Increase learning rate, reduce neurons in Residual Block layers or change the network structure of it

Q2: The Converged reward of the proposed PPO is similar or slightly worse than DDPG in 4 x 4 environments

A2: Adjust PPO's initial noise level in stochastic policy to enhance reward convergence.

Exploration vs. Exploitation trade-off plays a crucial role in final reward values.

References

Papers cited	How is the paper used (specifics)	Remarks (if any)
[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.	