applied sciences

MDPI

Article

Autonomous Robot Goal Seeking and Collision Avoidance in the Physical World: An Automated Learning and Evaluation Framework Based on the PPO Method

Wen-Chung Cheng, Zhen Ni *, Xiangnan Zhong and Minghan Wei

Department of Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton, FL 33431, USA; wcheng3@fau.edu (W.-C.C.); xzhong@fau.edu (X.Z.); weim@fau.edu (M.W.)
* Correspondence: zhenni@fau.edu; Tel.: +1-561-297-0035

Abstract: Mobile robot navigation is a critical aspect of robotics, with applications spanning from service robots to industrial automation. However, navigating in complex and dynamic environments poses many challenges, such as avoiding obstacles, making decisions in real-time, and adapting to new situations. Reinforcement Learning (RL) has emerged as a promising approach to enable robots to learn navigation policies from their interactions with the environment. However, application of RL methods to real-world tasks such as mobile robot navigation, and evaluating their performance under various training–testing settings has not been sufficiently researched. In this paper, we have designed an evaluation framework that investigates the RL algorithm's generalization capability in regard to *unseen* scenarios in terms of learning convergence and success rates by transferring learned policies in simulation to physical environments. To achieve this, we designed a simulated environment in Gazebo for training the robot over a high number of episodes. The training environment closely mimics the typical indoor scenarios that a mobile robot can encounter, replicating real-world challenges. For evaluation, we designed physical environments with and without unforeseen indoor scenarios. This evaluation framework outputs statistical metrics, which we then use to conduct an extensive study on a deep RL method, namely the proximal policy optimization (PPO). The results provide valuable insights into the strengths and limitations of the method for mobile robot navigation. Our experiments demonstrate that the trained model from simulations can be deployed to the previously *unseen* physical world with a success rate of over 88%. The insights gained from our study can assist practitioners and researchers in selecting suitable RL approaches and training–testing settings for their specific robotic navigation tasks.

Keywords: automated learning framework; platform development; autonomous robot; deep reinforcement learning; physical implementation; collision avoidance

Citation: Cheng, W.-C.; Ni, Z.; Zhong, X.; Wei, M. Autonomous Robot Goal Seeking and Collision Avoidance in the Physical World: An Automated Learning and Evaluation Framework Based on the PPO Method. *Appl. Sci.* **2024**, *14*, 11020. https://doi.org/10.3390/app142311020

Academic Editor: Vincent A. Cicirello

Received: 13 September 2024
Revised: 7 November 2024
Accepted: 23 November 2024
Published: 27 November 2024

1. Introduction

Mobile robot navigation is a fundamental research topic in the realm of robotics, with applications across a wide range of domains [1,2]. The goal of enabling robots to autonomously navigate through intricate and dynamic environments has spurred the exploration of various methodologies, among which reinforcement learning (RL) has emerged as a promising paradigm. The utilization of RL techniques for mobile robot navigation builds upon a rich body of research. Prior studies have demonstrated the potential of RL in addressing navigation challenges.

Among various RL methods, proximal policy optimization (PPO) has gained popularity due to its efficiency and robustness [3–6]. PPO is a policy gradient method that updates the policy by taking a step that is close to the previous policy, while ensuring a bounded policy change. This helps to avoid large policy updates that can harm the algorithm's performance or cause instability. Several previous works have explored the use of PPO for mobile robot navigation, with different aspects and objectives. For instance, the authors

of [7] proposed a qualitative comparison of the most recent autonomous mobile robot navigation techniques based on deep reinforcement learning, including PPO. However, according to [8,9], PPO struggles in continuous action space and has a slow covergence rate.

According to the literature, deep reinforcement learning (DRL) has the potential to enable robots to learn complex behaviors from high-dimensional sensory data without explicit programming, and deep Q-network (DQN) [10–13] and deep deterministic policy gradient (DDPG) [14–17] are two promising DRL algorithms for mobile robot navigation. For instance, the authors of [18] devised the dynamic epsilon adjustment method integrated with DRL to reduce the frequency of non-ideal agent behaviors and therefore improved the control performance (i.e., goal rate). The authors of [19] proposed an improved deep deterministic policy gradient (DDPG) path planning algorithm incorporating sequential linear path planning (SLP) to address the challenge of navigating mobile robots through large-scale dynamic environments. The proposed algorithm utilized the strengths of SLP to generate a series of sub-goals for the robot to follow, while DDPG was used to refine the path and avoid obstacles in real time. This approach also had a better success rate than the traditional DDPG algorithm and the A+DDPG algorithm [20]. The authors of [21] proposed a new DRL algorithm called a dueling Munchausen deep Q network (DM-DQN) for robot path-planning. DM-DQN combined the advantages of dueling networks and Munchausen deep Q-learning to improve exploration and convergence speed. In static environments, DM-DQN achieved an average path length that was shorter than both DQN and Dueling DQN. In dynamic environments, DM-DQN achieved a success rate that was higher than DQN and Dueling DQN. The authors of [22] proposed a global path planning algorithm for mobile robots based on the prior knowledge of PPoAMR, a heuristic method that used prior knowledge particle swarm optimization (PKPSO) [23–25]. They used PPoAMR to find the best fitness value for the path planning problem and then used PPO to optimize the path. Moreover, they showed that their algorithm can generate optimal and smooth paths for mobile robots in complex environments. However, as [26] pointed out, implementing DQN or any extended DQN algorithms on physical robots is more challenging. They are also often more resource-intensive. Moreover, DQN algorithms require extensive training data, are sensitive to environmental changes, and necessitate careful hyperparameter tuning. On the other hand, DDPG requires a large number of samples to converge to the optimal policy [27].

The authors of [28] proposed a generalized computation graph that integrates both model-free and model-based methods. Experiments with a simulated car and real-world RC car demonstrated the effectiveness of the approach. In recent years, there has been a growing interest in developing platforms that enable the evaluation of multi-agent reinforcement learning and general AI approaches. One such platform is SMART, introduced by [29], an open-source system designed for multi-robot reinforcement learning (MRRL). SMART integrated a simulation environment and a real-world multi-robot system to evaluate AI approaches. In a similar vein, the Arcade Learning Environment (ALE) was proposed by [30] as a platform for evaluating domain-independent AI. The ALE provided access to hundreds of Atari 2600 game environments, offering a rigorous testbed for comparing various AI techniques. The platform also includes publicly available benchmark agents and software for further research and development. However, these approaches require specialized hardware or resources, limiting accessibility for those with standard PC setups to conduct research from home.

Sampling efficiency and safety issues limit robot control in the real world. One solution is to train the robot control policy in a simulation environment and transfer it to the real world. However, policies trained in simulations often perform poorly in the real world due to imperfect modeling of reality in simulators [31].

The authors of [32] developed a training procedure, a set of actions available to the robot, a suitable state representation, and a reward function. The setup was evaluated using a simulated real-time environment. The authors compared a reference setup, different goal-oriented exploration strategies, and two different robot kinematics (holonomic and

differential). With dynamic obstacles, the robot was able to reach the desired goal in 93% of the episodes. The task scenario was inspired by an indoor logistics setting, where one robot is situated in an environment with multiple obstacles and should execute a transport order. The robot's goal was to travel to designated positions in the least amount of time, without colliding with obstacles or with the walls of the environment. However, this was all done in the simulated setting. The authors of [33] designed an automated evaluation framework for Rapidly-Exploring Random Tree (RRT) frontier detector for indoor space navigation, which addressed the performance verification aspect. However, the method being verified was not of the RL category. The authors of [34] also designed a simulation framework for training the *Turtlebot* robot agents. However, the framework was only designed for simulation purposes. These important works have set the stage for the examination of PPO in our study since the algorithm has a decent performance in robotic tasks and is easier to implement in mobile robots.

The domain of mobile robot navigation presents a multitude of complexities, including navigating through cluttered spaces, avoiding obstacles, and making decisions in real time. To address these challenges, this study employs an automated evaluation framework tailored for Turtlebot robots, designed to rigorously assess the performance of RL techniques in a controlled environment. The investigation is anchored in an extensive analysis, focusing on the PPO algorithm. This technique is scrutinized in a discrete action space. The contributions of this paper are as follows:

- We redesigned and implemented a personalized automated learning and evaluation framework based on an existing repository [34], allowing users to specify custom training and testing parameters for both simulated and physical robots (*Turtlebot*). This makes the platform adaptable and accessible to a wider range of users with limited resources.
- To establish a robust benchmark for robotic research study, we design a simulated environment in Gazebo to train the agent. The training environment closely mimics typical indoor scenarios encountered by *Turtlebot* robots with common obstacles such as walls and barriers, replicating real-world challenges.
- Our implementation provides statistical metrics, such as goal rate, for detailed comparison and analysis, with the flexibility to extend output to additional data such as trajectory paths and robot LiDAR readings. Moreover, our implementation can be extended to output additional metrics such as trajectory data, robot LiDAR data, etc. The insights gleaned from this study are invaluable to practitioners and researchers for their specific robotic navigation tasks.
- We conducted extensive physical experiments to evaluate the real-world navigation performance under varied environment configurations. Our results show that the agent trained in simulation can achieve a success rate of over 88% in our physical environments. Simulated training is not restricted by physical constraints such as the robot's battery power, and is more efficient in training data collection. This finding demonstrates the value of simulated training with RL for real world mobile navigation.

The organization of the paper is provided as follows. The algorithm background of PPO is presented in Section 2. The automated framework development and the integration of it with the PPO algorithm for robot navigation is presented in Section 3. The experimental set-up along with the set parameters for each environment is presented in Section 4. The results with relevant figures are provided in Section 5. Finally, Section 6 concludes the work.

2. PPO Algorithm Background

Schulman et al. [3] proposed the Proximal Policy Optimization (PPO) algorithm, which presents a novel approach to reinforcement learning that focuses on improving the stability and sample efficiency of policy optimization. PPO belongs to the family of policy gradient methods and aims to address some of the limitations of earlier algorithms like TRPO [35].

PPO introduces the concept of a clipped surrogate objective function, which plays a central role in its stability. This function limits the extent to which the policy can change

during each update, effectively preventing overly aggressive updates that can lead to policy divergence:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]. \quad (1)$$

where $\hat{E}_t$ represents the empirical average across time t. $\epsilon = 0.2$ is a hyperparameter. $clip(\cdot)$ is the clipping function that restricts $r_t(\theta)$ into the interval $[1-\epsilon, 1+\epsilon]$. $r_t(\theta)$ denotes the probability ratio, which is the ratio between the current policy, $\pi_\theta(a_t|s_t)$, and the policy from the previous time step, $\pi_{\theta_{old}}(a_t|s_t)$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}, \quad (2)$$

$\hat{A}_t$ denotes the advantage estimator, which is defined as follows:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \quad (3)$$

$$\delta_t = R_{t+1} + \gamma V_\mu(s_{t+1}) - V_\mu(s_t), \quad (4)$$

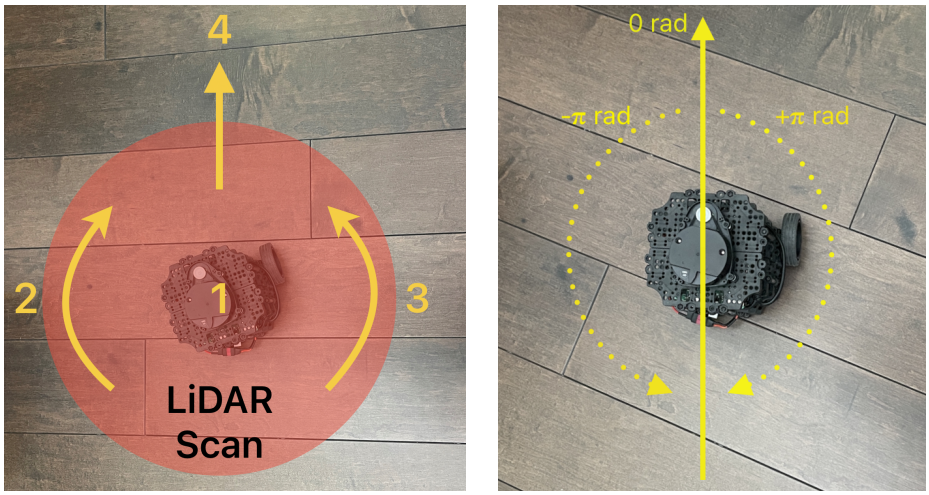
where $\gamma = 0.99$ is the discount factor. $\lambda = 2 \times 10^{-4}$ is the learning rate. t is the current time step. T is the total number of time steps in the end. R_{t+1} is the reward received in the next time step. $V_\mu(s_{t+1})$ is the value function of the next time step, and $V_\mu(s_t)$ is the current value function. Both functions are output by the critic network with parameter set μ. By constraining policy changes, PPO ensures smooth learning and reliable convergence.

Another crucial aspect of PPO is its use of importance sampling. This technique enables a balance between exploration and exploitation by adjusting the policy updates based on the ratio of probabilities between the new and old policies. This allows the agent to explore new actions while still staying close to its current policy, preventing excessive deviations. The PPO algorithm also boasts high sample efficiency, making it particularly suitable for applications where data collection is resource-intensive. It efficiently learns from a relatively small amount of data, which is vital for real-world scenarios where collecting extensive data can be costly or time-consuming.

3. Integration of the PPO Algorithm for Robot Navigation

For both the simulation and physical experiments study, a TurtleBot machine learning development package for ROS (Robot Operating System) has been used [36]. It is a collection of software tools and libraries that enable the development and deployment of machine learning algorithms on TurtleBot robots, and provides a wide range of functionalities, such as data collection, data pre-processing, training, and evaluation of machine learning models. At the beginning of each training episode, the agent is assigned a random goal. The agent then learns to navigate to goals based on its LiDAR sensor data. This demonstration is shown in Figure 1a.

The agent takes 360-degree LiDAR scan data, the current heading to the goal, the current distance to the goal, the current distance to the closest obstacle, and the current heading to the closest obstacle as inputs. The LiDAR scan data input is composed of 360 infrared readings of distances between the agent and the obstacles around it. More details about the sensor can be found in Appendix A. The heading of any detected point with respect to the agent takes value in $[-\pi, \pi)$, where 0 rad corresponds to the forward direction of the agent, as shown in Figure 1b. The heading to the point of interest increases as the agent turns counterclockwise towards the point, and decreases as the agent turns the other way. The agent can perform four actions as shown in Figure 1a. Action 1 is when the agent is stationary ($lws = 0$ m/s, $rws = 0$ m/s, where lws represents left wheel speed, and rws represents right wheel speed). Actions 2 and 3 are turning clockwise ($lws = 0.4$ m/s, $rws = 0$ m/s) and counterclockwise ($lws = 0$ m/s, $rws = 0.4$ m/s), respectively. Action 4 is going forward only ($lws = 0.4$ m/s, $rws = 0.4$ m/s), so the agent's movement is limited to moving forward.



(**a**) Agent actions and input schematic (**b**) Agent coordinate frame convention

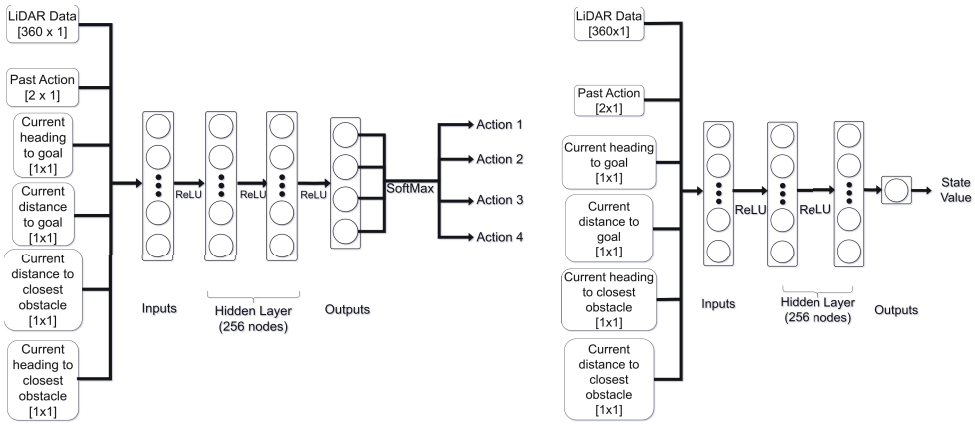
Figure 1. (**a**) Agent learning its surroundings using LiDAR data while navigating to the goal. The agent has four discrete actions available in this environment, where Action 1 is not moving, Action 2 is turning clockwise, Action 3 is turning counter-clockwise, and Action 4 is moving forward. (**b**) Coordinate frame of the agent, where 0 rad is the forward direction of the agent.

3.1. Agent Model

Figure 2 is the neural network structure of the PPO method used in the training. Note that the PPO method includes both an actor and a critic networks. As can be seen in Figure 2, the actor network is made up of an input layer followed by a ReLU activation function, a hidden layer of 256 nodes followed by another ReLU activation function, which is followed by another hidden layer of 256 nodes and a ReLU activation function, and an output layer followed by a SoftMax activation function that outputs one of the four actions at each step. The critic network has a similar structure as the actor network, except that its output is a single state value of the current state. Note that the heading and distance to the goal and obstacles are obtained using the odometry data of the robot agent.

3.2. PPO Workflow

The PPO update script is structured as in Figure 3. At each time step, the actor network within the PPO agent exhibits some action a_t and receives a tuple of the current state s_t, the action taken a_t, the reward received R_{t+1}, the logarithmic probability of the action taken $log_{prob}(a_t)$, and the binary episode termination flag $Flag$ (the flag takes a value of either 1 or 0). The tuple is then stored into a trajectory, which will be used to update the PPO agent every other 500 time steps, where each time step is at the millisecond scale. During each update, the actor network outputs the policy of the current state π_θ, which will be used to calculate the probability ratio $r_t(\theta)$. The state value returned by the critic network is then used to estimate the advantage function $\hat{A}_t$. Then both $\hat{A}_t$ and $r_t(\theta)$ are used to calculate the actor network objective function $L^{CLIP}(\theta)$, which is used to update the actor neural network weights. The aforementioned calculations are boxed in blue to show that they are specific to the updates of the actor network. At the end, the current policy π_θ is saved as the previous policy $\pi_{\theta_{old}}$, and the current policy π_θ is updated using stochastic gradient descent (SGD). The current value function $V_\mu(s_t)$ is updated using SGD, based on the error between predicted and actual values. The aforementioned calculations are boxed in red to show that they are specific to the updates of the critic network. Note that the solid arrow lines represent direct maneuver of the variables, whereas the dotted arrow lines represent the updates of the networks.



(**a**) Actor Network (**b**) Critic Network

Figure 2. (**a**) Structure of the actor network. The network takes 6 inputs. The LiDAR data input is composed of 360 infrared readings of distances between the agent and the obstacles around it. The past action input is composed of left and right wheel speeds from the previous time step. The other 4 inputs are of single values. The inputs are sent through a ReLU activation function to a hidden layer of 256 nodes. Afterwards, they are sent through another ReLU activation function to another hidden layer of 256 nodes. The output is then sent through first a ReLU and then a SoftMax activation function, which would be either of the 4 action values. (**b**) Structure of the critic network. The network takes the same inputs as the actor network. The output is the value of the agent's current state.

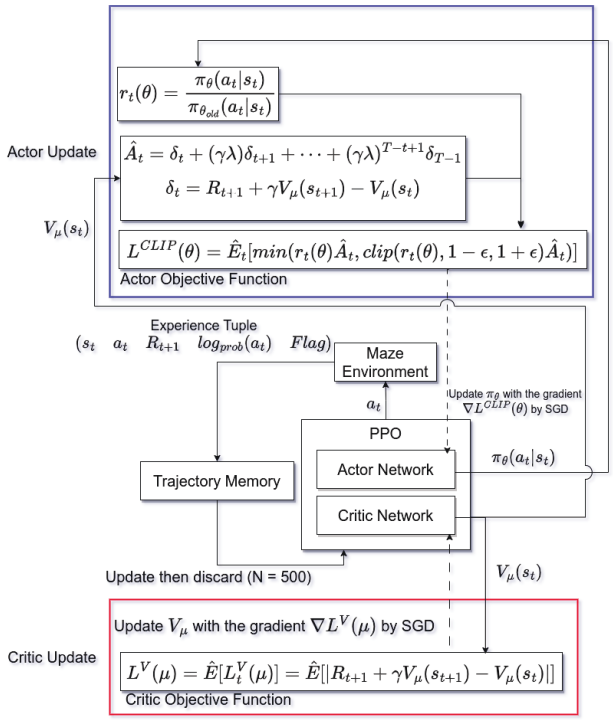


Figure 3. PPO agent update workflow. The PPO agent is updated every other 500 time steps. Each of the trajectory memory tuple contains the current state s_t, the action taken a_t, the reward received R_{t+1}, the logarithmic probability of the action taken $log_{prob}(a_t)$, and the binary episode termination flag $Flag$. The goal of the actor network here is to maximize the value function output by the critic network. The operations within the blue box are specific to the actor network updates, whereas the operations within the red box are specific to the critic network updates. Note that the solid arrow lines represent direct maneuver of the variables, whereas the dotted arrow lines represent the updates of the networks.

3.3. Overall Training Workflow

To make the whole evaluation more streamlined and autonomous, we have redesigned an evaluation framework. This framework is structured as in Figure 4. At the beginning of the current training round, the user can specify several parameters. For instance, the user can specify whether to conduct testing or training on the agent. An environment choice index can be specified as well, allowing the user to select different simulated environments for training/testing. The user can also choose whether to conduct the study in a simulated or real environment. The number of training/testing episodes (N) and the option to load a previously trained model can also be specified. When the episode termination condition is met, the current goal rate will be calculated and saved to an array of N episodes of goal rates, which will be saved to a CSV file for further analysis. Algorithm 1 provides pseudocode for the entire training process. Action A in the pseudocode can be any one of the four actions shown in Figure 1a and explained in Section 3, whereas the current state S and the new state S' (the state from the next step) are the array containing the six inputs shown in Figure 2 and explained in Section 3.1.

Algorithm 1 Training Pseudocode

User Input:
$TestTrainingFlag, Env, RealSimFlag, NumEp, LoadFlag$
$EpisodeSuccess = [\varnothing]$
if $RealSimFlag =$ Sim **then**
    Load the robot model and Gazebo model of Env
else
    Obtain the live LiDAR data from the physical robot in the real Env
end if
if $LoadFlag =$ TRUE **then**
    Load previously saved policy onto $PPOAgent$
else
    Initialize random policy of $PPOAgent$
end if
for $Episode = 1, 2, \ldots,$ **NumEp do**
    Initialize S from Env
    for $Step = 1, 2, \ldots, 500$ **do**
        Choose A from S using policy derived from $PPOAgent$
        Take action A, observe S'
        $S \leftarrow S'$
        if $TestTrainingFlag =$ Training **then**
            Update $PPOAgent$
        end if
        if $S' = Collision$ or $Step > 500$ **then**
            Break
        end if
    end for
    if $S' = Collision$ or $Step > 500$ **then**
        Append 0 to $EpisodeSuccess$
    else
        Append 1 to $EpisodeSuccess$
    end if
    $GoalRateCSV \leftarrow$ AVERAGE$(EpisodeSuccess)$
end for
Output $GoalRateCSV$

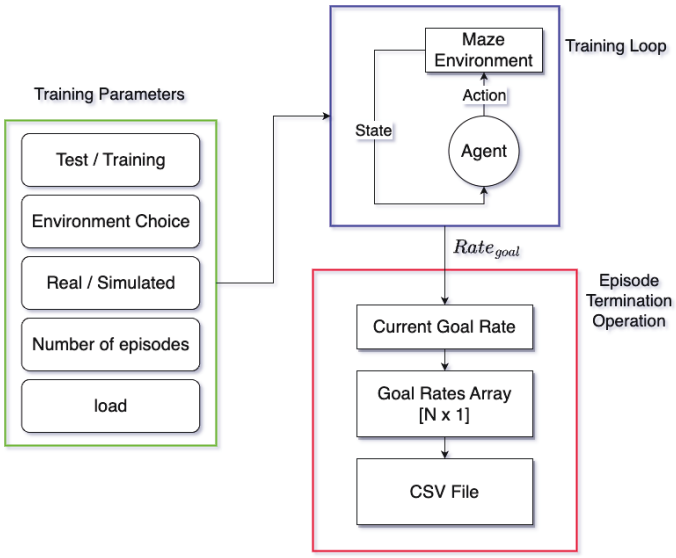


Figure 4. Overall workflow. The user can specify the parameters needed at the beginning of the current training round. In each episode, the agent receives the new state s_t, when taking action a_t. When the episode termination condition is met (i.e., the binary episode termination flag $Flag$ is set to 1), the current goal rate will be calculated and saved to an array, which will be saved to a CSV file for further analysis.

3.4. Reward Function

The reward function has two components, each influencing the agent's actions and learning process. The components include a goal-oriented reward function, which calculates the reward contributed by the location of the goal relative to the agent, and an obstacle-oriented reward function, which calculates the reward contributed by the location of the closest obstacle relative to the agent. The composition of these elements is carefully designed to meet the specific objectives and shape the agent's behavior accordingly.

The goal-oriented reward function is formally represented as Equation (5):

$$R_{t_{Goal}} = \begin{cases} -10 & \Delta d_{Goal} > 0.5 \text{ m}, \Delta d_{Goal} \leq 0 \text{ m} \\ 200 \times \Delta d_{Goal} \times \left(1 - 4 \times \left|0.5 - \left\{\frac{\mathcal{A}}{\pi}\right\}\right|\right) & 0 \text{ m} < \Delta d_{Goal} \leq 0.5 \text{ m} \end{cases}, \quad (5)$$

The goal distance rate is calculated as in Equation (6), which represents the change of agent to goal distance between steps:

$$\text{Goal distance rate} = \Delta d_{Goal} = PastGoalDistance - GoalDistance \quad (6)$$

Intuitively if $\Delta d_{Goal} \leq 0$ m, the agent is further away from the goal than the previous time step. $G_H = \arctan \frac{y_{goal} - y_{robot}}{x_{goal} - x_{robot}} - \theta$ represents the goal heading, where (x_{goal}, y_{goal}) and (x_{robot}, y_{robot}) are the Cartesian coordinates of the goal and robot, respectively, and θ is the robot's orientation. The $\{\cdot\}$ function takes the fraction part of any calculated result inside, where $\mathcal{A} = (0.5 \times (G_H + \pi)) \pmod{2\pi}$, which is the remainder after division by 2π. We partially follow the implementation in the original work [36] to define $\mathcal{A}$ with an offset value of π to ensure that the robot receives the maximum rewards when facing the goal directly. Figures 5 and 6 show the hardware representations of the robot, obstacles, and goal positions, where Figure 5 shows an example of how the goal distance and heading change for the agent. Figure 7a shows the plot of the goal-oriented reward function. As can be seen in the plot, the agent receives the most reward if it is facing the goal directly and is close to it. The agent receives the largest penalty when facing away from the goal. Note that the green arrow line represents the path of the agent.

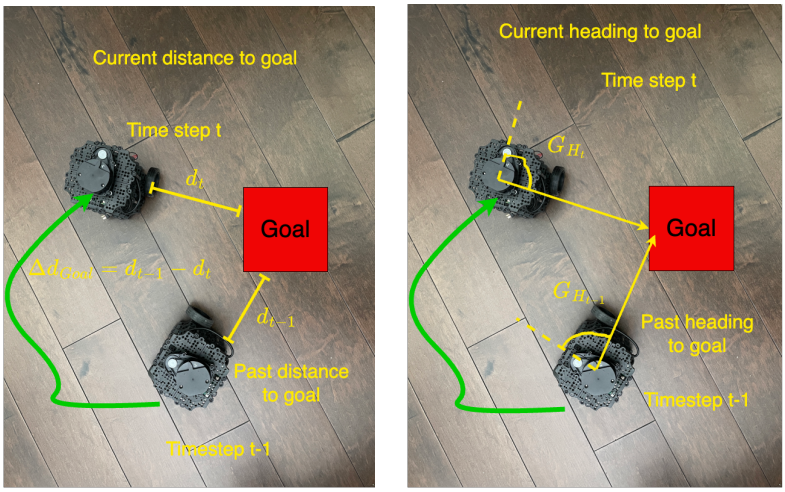
The obstacle-oriented reward function is formally represented as Equation (7):

$$R_{t_{Obstacle}} = \begin{cases} 5 & \Delta d_{Obstacle} \leq 0 \text{ m} \\ -400 \times \Delta d_{Obstacle} \times \left(1 - 4 \times \left|0.5 - \left\{\frac{\mathcal{B}}{\pi}\right\}\right|\right) & \Delta d_{Obstacle} > 0 \text{ m} \end{cases}, \quad (7)$$

The obstacle distance rate is calculated as in Equation (8), which represents the change of agent to closest obstacle between steps:

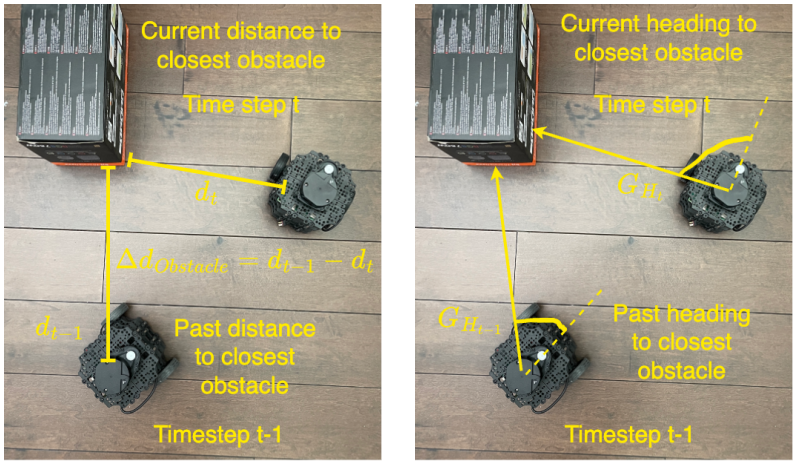
$$\text{Obstacle distance rate} = \Delta d_{Obstacle} = PastObstacleDistance - ObstacleDistance \quad (8)$$

Intuitively if $\Delta d_{Obstacle} \leq 0$ m, the agent is further away from the closest obstacle than the previous time step. O_H represents the obstacle heading, where $\mathcal{B} = (0.5 \times (O_H + \pi)) \pmod{2\pi}$. We partially follow the implementation in the original work [36] to define $\mathcal{B}$ with an offset value of π to ensure that the robot receives the minimum rewards when directly facing the closest obstacle. Figure 6 demonstrates pictorially one case of how the obstacle distance and heading might change for the agent. Figure 7b shows the plot of the obstacle-oriented reward function. As can be seen in the plot, the agent receives the most reward if it is facing away from the closest obstacle. The agent receives the largest penalty when facing the obstacle directly and is close to it.



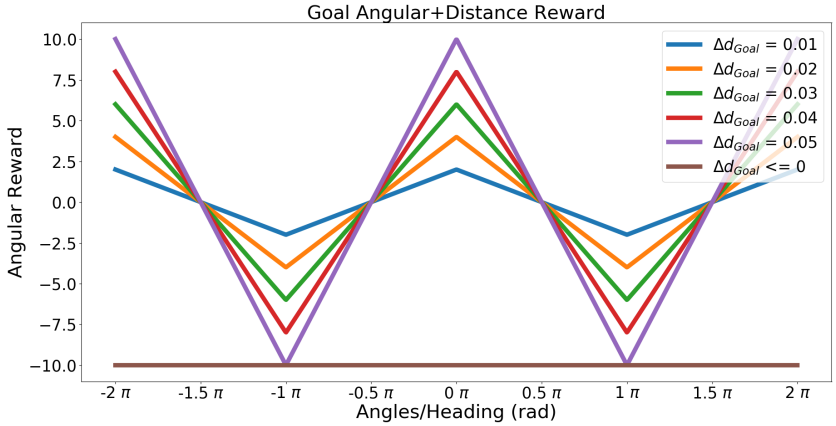
(**a**) Goal distance representation (**b**) Goal heading representation

Figure 5. (**a**) Current and past goal distance and (**b**) goal heading representation. The green arrow line represents the path of the agent.

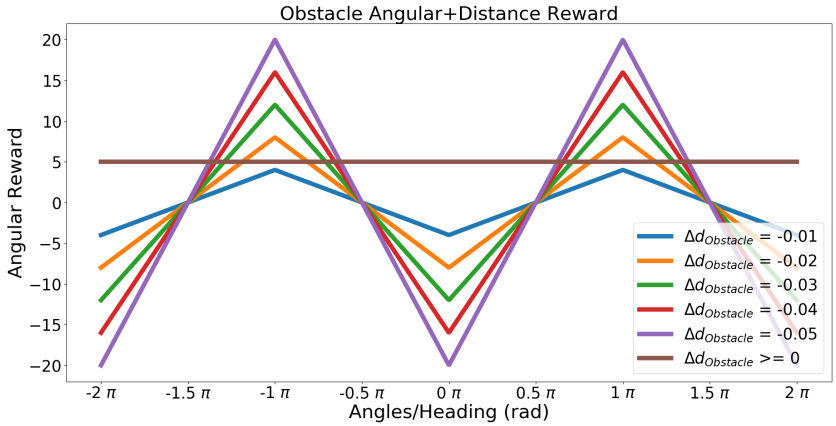


(**a**) Obstacle distance representation (**b**) Obstacle heading representation

Figure 6. (**a**) Current and past obstacle distance and (**b**) obstacle heading representation.



(**a**) Goal−Oriented Reward Function Plots



(**b**) Obstacle−Oriented Reward Function Plots

Figure 7. (**a**) The reward plots related to goal heading and distance. The agent receives maximum reward if it is facing the goal directly and is close to it. (**b**) The reward plots related to obstacle heading and distance. The agent receives the largest penalty when facing the obstacle directly and close to it.

4. Experimental Set-Up

4.1. Source Simulated Maze

The simulated training environment used for this framework was created using the Gazebo simulator [37], which provides realistic robotic movements, a physics engine, and the generation of sensor data combined with noise. The source simulated maze, shown in Figure 8, is a customized map with an area of approximately 82.75 m^2 (free space area). This simulated maze is an approximate replica of one of the target mazes. For additional information regarding the parameters chosen for the training phase in this particular simulation environment, see Appendix B.

4.2. Testing Physical Environments

The first physical maze, shown in Figure 9a, is a customized map free of obstacles. The second physical maze, shown in Figure 9b, is another customized map that is similar to the simulated environment. Note this maze has walls. The third physical maze, shown in Figure 9c, is another customized map that has a slab of wall in the middle that allows the robot to pass through on both sides. These mazes will be the physical testing environments for the agent. For additional information regarding the parameters chosen for the training and testing phases in this particular physical environment, see Appendix C.

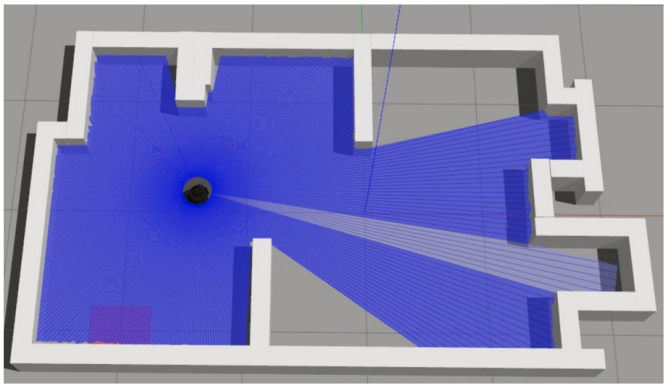
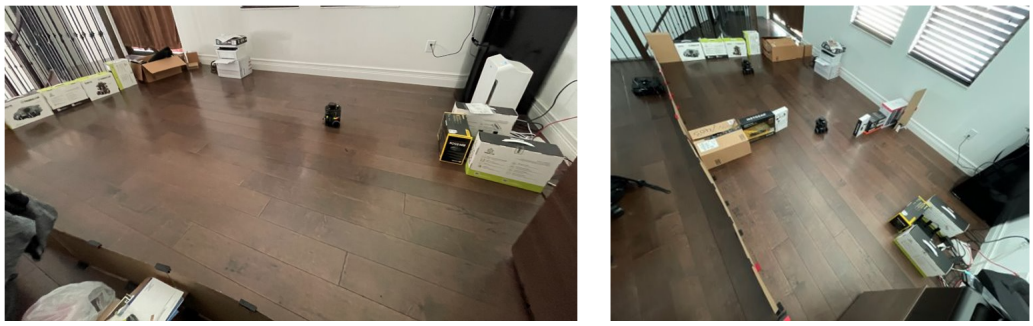
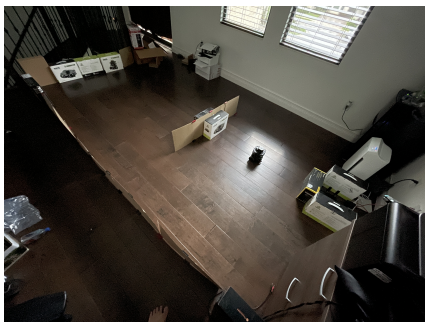


Figure 8. The custom simulated environment. Dimension = 12.50 m × 8.00 m. Environment size ≈ 82.75 m^2.



(**a**) (**b**)



(**c**)

Figure 9. (**a**) Physical Maze 1, (**b**) Maze 2, and (**c**) Maze 3. (**a**) Physical Maze 1. Dimension = 12.50 m × 8.00 m. Environment size ≈ 82.75 m^2. (**b**) Physical Maze 2. Dimension = 12.50 m × 8.00 m. Environment size ≈ 82.75 m^2. (**c**) Physical Maze 3. Dimension = 12.50 m × 8.00 m. Environment size ≈ 82.75 m^2.

5. Results Analysis

5.1. Simulation Training

The agent was first trained in a simulated environment as shown in Figure 8 for 5000 episodes. This number of episodes was chosen because a reasonable performance was reached (a goal rate of 0.78). Figures 10–12 show the success rate, the total number of goals reached, and the average rewards obtained over all training episodes. The x-axis in each of the figures represents the total number of episodes run, and the y-axis represents the respective metric being evaluated.

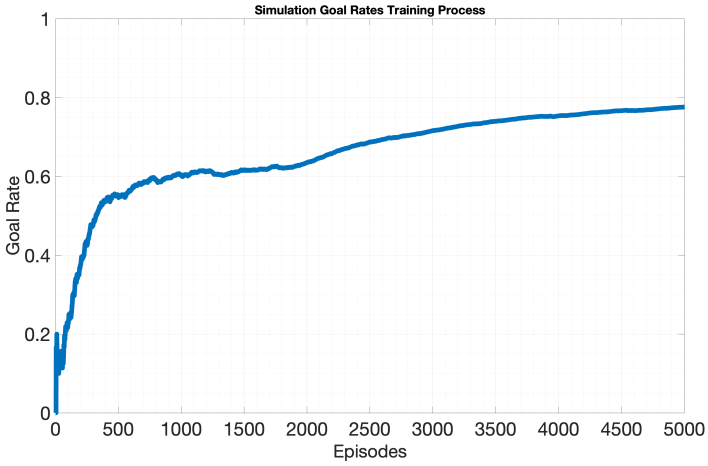


Figure 10. The goal rate for all training episodes in the simulated environment with obstacles. The simulated environment, shown in Figure 8, was used. One of the advantages of simulated training is abundant training episodes without a time-consuming setup. After about 5000 episodes, the agent's goal rate converged to about 79%.

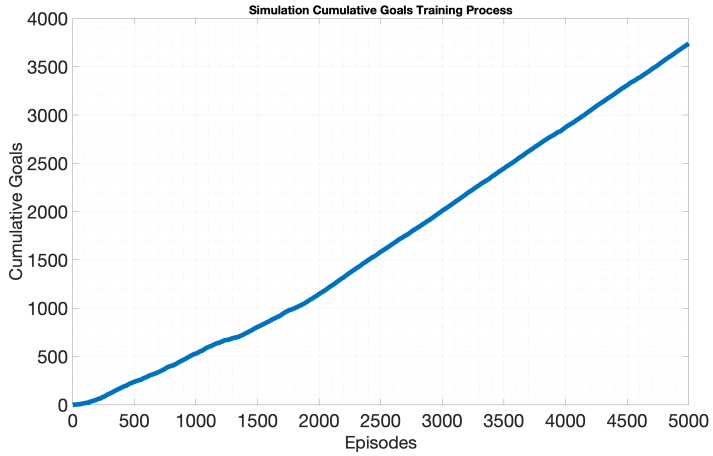


Figure 11. The total number of goals reached for all training episodes in the simulated environment with obstacles. The simulated environment, shown in Figure 8, was used. Again, as the agent improves at reaching the target, the curve should appear more linear, which is shown here close to the end of 5000 episodes.

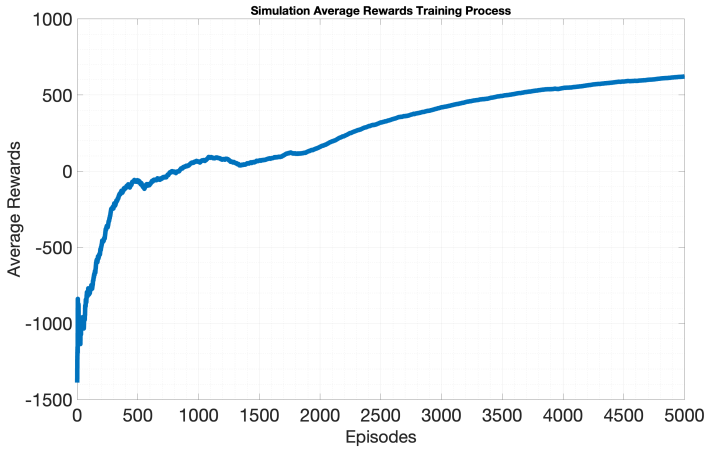


Figure 12. Average rewards for all training episodes in the simulated environment with obstacles. As the agent improves at the task at hand, the average rewards should become more positive and converge to a value, which is shown here.

All figures have shown sufficient performance by the agent because the agent showed a goal rate of approximately 79% at the end of the 5000-episode training, as demonstrated in Figure 10. In Figure 11, the line representing the total number of goals reached is almost at 45 degrees relative to the x-axis near the end of the training. This means that the agent is almost always reaching the goal at each episode. In addition, the average rewards obtained was becoming more positive and stabilized (the curve is becoming flatter in the positive region), as shown in Figure 12. The network weights at the end of this training were transferred to environments as shown in Figure 9 for physical testing. To assess the reliability of this training, we repeated the same training process for 100 runs, where each run lasts for 5000 episodes. The results are in Figure 13. The blue line shows the agent's mean goal rate over 5000 episodes for all 100 runs, whereas the shaded region indicates the 95% confidence interval across all runs. This interval provides insight into the variability of performance, which converges as the agent stabilizes its learning. We have also trained a basic DQN agent under the same settings, but the goal rate converged at around only 10%.

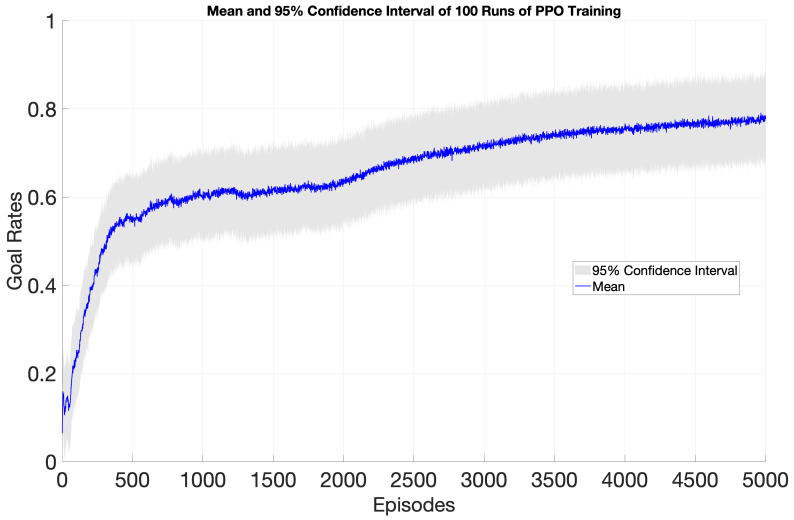


Figure 13. Confidence interval plot of 100 independent PPO training runs across 5000 episodes. The shaded region around the mean represents the 95% confidence interval, quantifying run variability and training process robustness.

5.2. Physical Testing

We evaluate the performance of the agent in the physical environments as shown in Figure 9 for 5 runs of 100 trials. A run consists of any number of trials (in this case, 100), whereas the robot attempting to navigate to the goal position is counted as a trial. Table 1 shows the success rates over all the 5 runs of 100 trials in the environments mentioned. We can see that the agent has the highest average goal rate in Maze 1 since Maze 1 is an empty maze. In Maze 2, the agent has a goal rate of 0.724, which is comparable to the converged goal rate in the simulation training. The agent has a goal rate of 0.881 in Maze 3 since there is a slab of wall in the middle and it is easier for the robot to pass through either side. Although the agent achieved a success rate of approximately 79% in the simulated environment (Simulated Maze 2), its performance dropped slightly to 72.4% in the physical environment (Maze 2). This difference is attributed to the environmental noise present in the physical setting, which is absent in simulation.

Table 1. Average Goal Rate. The goal rate value is first calculated out of 100 trials and then averaged over 5 runs.

	Maze 1	**Maze 2**	**Maze 3**
Average Goal Rate	0.952	0.724	0.881

Figure 14 shows that the agent reached its goals the quickest in Maze 1, which is an empty maze, as indicated by the steep slope of the blue line. Maze 2, with a goal rate of 0.724, proved to be the most challenging due to its narrower central open space compared to the other mazes. In Maze 3, the agent reached its goals at a moderate pace, navigating around a central wall by successfully learning to pass on either side. This is reflected by the more gradual slope of the yellow line in the figure.

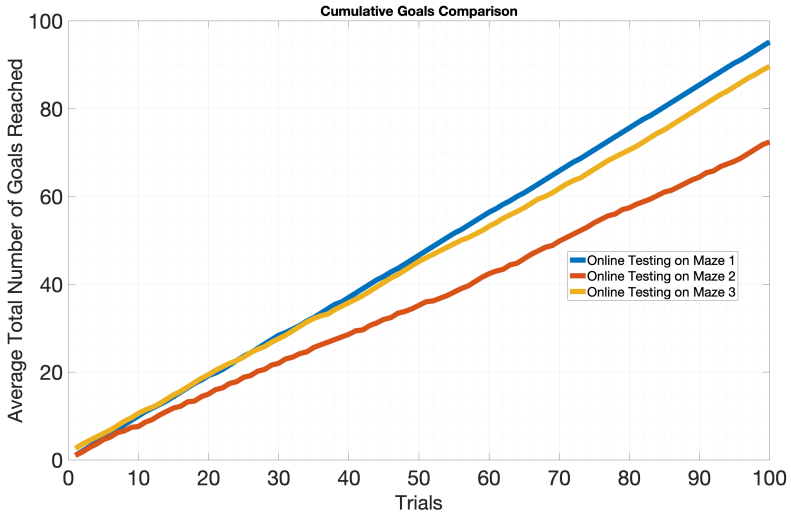


Figure 14. Average cumulative goals for all 5 runs of 100 trials in physical environments shown in Figure 9. It can be seen clearly that the agent reaches goals more often in Maze 1 since it is a free maze, whereas the agent reaches the goals least often in Maze 2 since there are more walls.

6. Conclusions

In this paper, we design an existing automated evaluation framework so that the user can input more specific training parameters before the actual testing, allowing for easily repeatable experiments with any number of iterations. Our work also allows the framework to evaluate the agent's testing performance. We show this by first defining a source simulated environment that is similar to one of the target real-world environments. We then set up three physical environments (an empty maze as Maze 1, a maze similar to the simulation maze as Maze 2, and the maze with one slab of wall in the middle as Maze 3) to show the performance difference of the agent in different physical environments including familiar and unforeseen environments. Our experiments show that sufficient training in simulation can greatly improve the agent's performance when transferred to physical environments. With our framework, statistical results such as goal rates can be output as a CSV file for later analysis.

Author Contributions: W.-C.C., Z.N., and X.Z. identified the topic, formulated the problem, and decided the system-level framework. W.-C.C. integrated the method, conducted the simulation and physical experiments, collected the data, and initialized the draft. M.W. and Z.N. provided related literature, suggested the experiment designs, and discussed the comparative results. All authors analyzed the experimental data, such as figures and tables, and proofread the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by the National Science Foundation under Grant 2047064, 2047010, 1947418, and 1947419.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Source code and case study data are available at https://github.com/Ac31415/RL-Auto-Eval-Framework, accessed on 22 November 2024.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A. Sensor Specifications

In this Appendix, we present the specifications of the LiDAR sensor used. More details can be found in the manufacturer's datasheet [38].

- Hardware: 360 Laser Distance Sensor LDS-01, Hitachi-LG Data Storage, Inc., Tokyo, Japan
- Quantity: 1
- Dimensions: 69.5 (W) × 95.5 (D) × 39.5 (H) mm
- Distance Range: 120–3500 mm
- Sensor Position: 192 mm from the ground

Appendix B. Simulation Training Parameters

In this appendix, we specify the hyperparameters used for simulation training, alongside various implementation details. These parameters are set so that the algorithm can converge within a reasonable time frame.

- Episode limit: 5000
- Update time step (memory limit): 500
- Policy update epochs: 50
- PPO clip parameter: 0.2
- Discount factor (γ): 2×10^{-4}

Appendix C. Physical Training and Testing Parameters

In this appendix, we specify the hyperparameters used for physical training and testing, alongside various implementation details. Note that during physical testing, update parameters were not used.

- Number of runs: 5
- Number of trials of each run: 100
- Update time step (memory limit): 500
- Policy update epochs: 50
- PPO clip parameter: 0.2
- Discount factor (γ): 2×10^{-4}

References

1. Gonzalez-Aguirre, J.A.; Osorio-Oliveros, R.; Rodriguez-Hernandez, K.L.; Lizárraga-Iturralde, J.; Morales Menendez, R.; Ramirez-Mendoza, R.A.; Ramirez-Moreno, M.A.; Lozoya-Santos, J.d.J. Service robots: Trends and technology. *Appl. Sci.* **2021**, *11*, 10702. [CrossRef]
2. O'Brien, M.; Williams, J.; Chen, S.; Pitt, A.; Arkin, R.; Kottege, N. Dynamic task allocation approaches for coordinated exploration of Subterranean environments. *Auton. Robot.* **2023**, *47*, 1559–1577. [CrossRef]
3. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
4. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv* **2015**, arXiv:1506.02438.
5. Heess, N.; Tb, D.; Sriram, S.; Lemmon, J.; Merel, J.; Wayne, G.; Tassa, Y.; Erez, T.; Wang, Z.; Eslami, S.; et al. Emergence of locomotion behaviours in rich environments. *arXiv* **2017**, arXiv:1707.02286.
6. Wang, Y.; Wang, L.; Zhao, Y. Research on door opening operation of mobile robotic arm based on reinforcement learning. *Appl. Sci.* **2022**, *12*, 5204. [CrossRef]
7. Plasencia-Salgueiro, A.d.J. Deep Reinforcement Learning for Autonomous Mobile Robot Navigation. In *Artificial Intelligence for Robotics and Autonomous Systems Applications*; Springer: Berlin/Heidelberg, Germany, 2023; pp. 195–237.
8. Holubar, M.S.; Wiering, M.A. Continuous-action reinforcement learning for playing racing games: Comparing SPG to PPO. *arXiv* **2020**, arXiv:2001.05270.
9. Del Rio, A.; Jimenez, D.; Serrano, J. Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments. *IEEE Access* **2024**, *12*, 146795–146806.
10. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
11. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef]

12. Kim, K. Multi-agent deep Q network to enhance the reinforcement learning for delayed reward system. *Appl. Sci.* **2022**, *12*, 3520. [CrossRef]
13. Pérez-Gil, Ó.; Barea, R.; López-Guillén, E.; Bergasa, L.M.; Gomez-Huelamo, C.; Gutiérrez, R.; Diaz-Diaz, A. Deep reinforcement learning based control for Autonomous Vehicles in CARLA. *Multimed. Tools Appl.* **2022**, *81*, 3553–3576. [CrossRef]
14. Lillicrap, T.P.; Hunt, J.J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; Wierstra, D. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
15. Barth-Maron, G.; Hoffman, M.W.; Budden, D.; Dabney, W.; Horgan, D.; Tb, D.; Muldal, A.; Heess, N.; Lillicrap, T. Distributed distributional deterministic policy gradients. *arXiv* **2018**, arXiv:1804.08617.
16. Egbomwan, O.E.; Liu, S.; Chaoui, H. Twin Delayed Deep Deterministic Policy Gradient (TD3) Based Virtual Inertia Control for Inverter-Interfacing DGs in Microgrids. *IEEE Syst. J.* **2022**, *17*, 2122–2132. [CrossRef]
17. Kargin, T.C.; Kołota, J. A Reinforcement Learning Approach for Continuum Robot Control. *J. Intell. Robot. Syst.* **2023**, *109*, 1–14. [CrossRef]
18. Cheng, W.C.A.; Ni, Z.; Zhong, X. A new deep Q-learning method with dynamic epsilon adjustment and path planner assisted techniques for Turtlebot mobile robot. In Proceedings of the Synthetic Data for Artificial Intelligence and Machine Learning: Tools, Techniques, and Applications, Orlando, FL, USA, 13 June 2023; Volume 12529, pp. 227–237.
19. Chen, Y.; Liang, L. SLP-Improved DDPG Path-Planning Algorithm for Mobile Robot in Large-Scale Dynamic Environment. *Sensors* **2023**, *23*, 3521. [CrossRef]
20. He, N.; Yang, S.; Li, F.; Trajanovski, S.; Kuipers, F.A.; Fu, X. A-DDPG: Attention mechanism-based deep reinforcement learning for NFV. In Proceedings of the 2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS), Tokyo, Japan, 25–28 June 2021; pp. 1–10.
21. Gu, Y.; Zhu, Z.; Lv, J.; Shi, L.; Hou, Z.; Xu, S. DM-DQN: Dueling Munchausen deep Q network for robot path planning. *Complex Intell. Syst.* **2023**, *9*, 4287–4300. [CrossRef]
22. Jia, L.; Li, J.; Ni, H.; Zhang, D. Autonomous mobile robot global path planning: A prior information-based particle swarm optimization approach. *Control Theory Technol.* **2023**, *21*, 173–189. [CrossRef]
23. Hamami, M.G.M.; Ismail, Z.H. A Systematic Review on Particle Swarm Optimization Towards Target Search in The Swarm Robotics Domain. *Arch. Comput. Methods Eng.* **2022**, 1–20. [CrossRef]
24. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95-International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948.
25. Wang, H.; Ding, Y.; Xu, H. Particle swarm optimization service composition algorithm based on prior knowledge. *J. Intell. Manuf.* **2022**, 1–19. [CrossRef]
26. Escobar-Naranjo, J.; Caiza, G.; Ayala, P.; Jordan, E.; Garcia, C.A.; Garcia, M.V. Autonomous navigation of robots: Optimization with DQN. *Appl. Sci.* **2023**, *13*, 7202. [CrossRef]
27. Sumiea, E.H.; Abdulkadir, S.J.; Alhussian, H.S.; Al-Selwi, S.M.; Alqushaibi, A.; Ragab, M.G.; Fati, S.M. Deep deterministic policy gradient algorithm: A systematic review. *Heliyon* **2024**, *10*, e30697. [CrossRef] [PubMed]
28. Kahn, G.; Villaflor, A.; Ding, B.; Abbeel, P.; Levine, S. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, Australia, 21–25 May 2018; pp. 5129–5136.
29. Liang, Z.; Cao, J.; Jiang, S.; Saxena, D.; Chen, J.; Xu, H. From multi-agent to multi-robot: A scalable training and evaluation platform for multi-robot reinforcement learning. *arXiv* **2022**, arXiv:2206.09590.
30. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.* **2013**, *47*, 253–279. [CrossRef]
31. Ju, H.; Juan, R.; Gomez, R.; Nakamura, K.; Li, G. Transferring policy of deep reinforcement learning from simulation to reality for robotics. *Nat. Mach. Intell.* **2022**, *4*, 1077–1087. [CrossRef]
32. Gromniak, M.; Stenzel, J. Deep reinforcement learning for mobile robot navigation. In Proceedings of the 2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS), Nagoya, Japan, 13–15 July 2019; pp. 68–73.
33. Andy, W.C.C.; Marty, W.Y.C.; Ni, Z.; Zhong, X. An automated statistical evaluation framework of rapidly-exploring random tree frontier detector for indoor space exploration. In Proceedings of the 2022 4th International Conference on Control and Robotics (ICCR), Guangzhou, China, 2–4 December 2022; pp. 1–7.
34. Frost, M.; Bulog, E.; Williams, H. Autonav RL Gym. 2019. Available online: https://github.com/SfTI-Robotics/Autonav-RL-Gym (accessed on 24 April 2022).
35. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 7–9 July 2015; pp. 1889–1897.
36. ROBOTIS-GIT. turtlebot3_machine_learning. 2018. Available online: https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning (accessed on 24 April 2022).

37. Gazebo. Open Source Robotics Foundation. 2014. Available online: http://gazebosim.org/ (accessed on 24 April 2022).
38. ROBOTIS-GIT. LDS Specifications. Available online: https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#components (accessed on 24 April 2022).