A New Deep Q-Learning Method with Dynamic Epsilon Adjustment and Path Planner Assisted Techniques for Turtlebot Mobile Robot

Wen-Chung (Andy) Cheng, Zhen Ni, and Xiangnan Zhong

Eletrical Engineering and Computer Science
Florida Atlantic University
777 Glades Rd, Boca Raton, FL 33473 US

ABSTRACT

Deep Q-learning (DQL) method has been proven a great success in autonomous mobile robots. However, the routine of DQL can often yield improper agent behavior (multiple circling-in-place actions) that comes with long training episodes until convergence. To address such problem, this project develops novel techniques that improve DQL training in both simulations and physical experiments. Specifically, the Dynamic Epsilon Adjustment method is integrated to reduce the frequency of non-ideal agent behaviors and therefore improve the control performance (i.e., goal rate). A Dynamic Window Approach (DWA) global path planner is designed in the physical training process so that the agent can reach more goals with less collision within a fixed amount of episodes. The GMapping Simultaneous Localization and Mapping (SLAM) method is also applied to provide a SLAM map to the path planner. The experiment results demonstrate that our developed approach can significantly improve the training performance in both simulation and physical training environment.

Keywords: Reinforcement learning, autonomous navigation, goal seeking, deep Q-Learning, SLAM mapping, machine learning and artificial intelligence, and ϵ-greedy exploration.

1. INTRODUCTION

Deep Reinforcement Learning (DRL) algorithm has been a promising method for training agents with unstructured data obtained through the interaction with environment. This method builds on top of the traditional reinforcement learning techniques by incorporating deep neural networks, which can learn to represent complex features of the environment and make predictions about future states and rewards. The combination has enabled significant breakthroughs in a variety of domains, including robotics and autonomous systems. Several factors such as hardware level and simulation setup can affect the performance of the training process. Aveen et al.[1] proposed reward classes with different reward functions. The reward classes were chosen based on the percentage of positive and negative rewards received by an agent. However, the performance comparison of the reward classes was based solely on the variance of the cumulative rewards. Thus, the goal rate and other metrics of the agent during training were difficult to obtain.

In 2022, Raz et al.[2] developed a Scenario-Assisted Deep Reinforcement Learning technique for enhancing the reinforcement learning training process, which allowed engineers to directly contribute their domain knowledge, making the agent under training more likely to comply with various relevant constraints. The authors modified the reward calculation based on the constraints relevant to internet traffic control domain knowledge. The results showed that the original framework violated the constraints about 9-11% of the time, while the enhanced framework with the proposed method has a violation rate of 0.34%. Corsi et al.[3] then applied the above technique to the robotic mapless navigation problem. The authors used Lagrangian Proximal policy optimization (Lagrangian-PPO) as the agent's neural network structure. The results showed that the enhanced model on

Further author information: (Send correspondence to Zhen Ni)
Andy Cheng: E-mail: wcheng3@fau.edu, Telephone: +1 561 542 7948
Zhen Ni: E-mail: zhenni@fau.edu, Telephone: +1 561 297 0035
Xiangnan Zhong: E-mail: xzhong@fau.edu, Telephone: +1 561 297 3412

average has a slightly higher success rate (95%) compared to the baseline model (87%). Moreover, the constraint violation frequency of the enhanced model almost diminished when compared to the baseline model. But these works presented enhanced frameworks that have over-fitting issues and prolonged convergence to the optimal policy during training.

The Deep Q-Learning approach (DQL), which is one of the most common DRL approaches for robot exploration, was patented to Deep Learning as an application of Q-learning by Google DeepMind in 2014.[4] Other variations such as Double and Duelling DQN were introduced to reduce the overestimation of Q-values that can occur in traditional Q-learning and improve the learning speed and stability[5,6]. Since both variations were model-free and off-policy, they were good for learning in environments with unknown dynamics and large state-space. One of the DRL methods, Advantage Actor-Critic (A2C) of,[7] combined the actor-critic neural network architecture with the advantage function to improve learning stability and efficiency. An asynchronous version of the A2C algorithm called Asynchronous Advantage Actor-Critic (A3C) was also proposed.[7] A3C used multiple parallel agents to improve sample efficiency and reduce correlations between samples. A family of policy gradient methods for reinforcement learning called Proximal Policy Optimization (PPO) were proposed.[8] PPO updated the policy function by taking a step that was close to the previous policy, which helped to improve stability and sample efficiency. In earlier work, a new deep reinforcement learning algorithm named Deep Deterministic Policy Gradient (DDPG) was proposed for continuous control tasks.[9] The algorithm extended the Q-learning and actor-critic methods to continuous action spaces and used a deterministic policy function that was learned through gradient ascent. DDPG combined an off-policy approach that allowed it to learn from a replay buffer with a target network that improved stability during learning. Jesus et al.[10] proposed a novel approach to mobile robot navigation using Soft Actor-Critic (SAC) reinforcement learning. SAC was a variant of actor-critic method that used a soft update rule to balance exploration and exploitation in the learning process of continuous control tasks. All of these works did not involve any domain knowledge in the degrees of action exploration. Consequently they all required the exploration conditions and stochastic policies, and could become unstable when used with a function approximator.[11]

The Dynamic Window Approach (DWA) path planner was based on the idea of considering the robot's current velocity and the available space for movement around the robot, and then computing a safe velocity that allowed the robot to navigate to its goal without colliding with any obstacles.[12] One of the key contributions of DWA path planner was its ability to handle non-holonomic motion constraints, which were common in mobile robots that couldn't move laterally. The method was also computationally efficient and can be implemented on low-cost hardware. Such path planner has been used in an automated evaluation framework for Rapidly-Exploring Random Tree (RRT) frontier detection.[13] The existing DRL methods for navigation tasks did not require the accompany of a low-level local controller, such as DWA to compute the desired path for the mobile robots.[14] With either just a trained DQN or the DWA alone as a path planner, there's a compromise to be made between adaptive learning and precision in collision avoidance that could affect the agent's training duration.

With the aforementioned observation, this paper expands the capacity of DQL method to facilitate the training process. The major work of this paper are as follows. First, by adaptively adjusting ϵ, the exploration rate of the agent, according to the metrics which is designed based on the current goal rate and violation count, our developed method can reduce the frequency of non-ideal behaviors of the agent over the training period and therefore, improve the control performance. Second, the DWA path planner has also been implemented along with the agent in the physical experiment so that its training performance can be improved. The experiment results show the effectiveness of the proposed method.

The organization of the paper is provided as follows. The theoretical algorithm background is presented in Section 2. The Epsilon Adjustment and Path Planner Assist methods are designed in Section 3. Section 4 applies the developed method in the simulation studies. The physical experiment on an autonomous robot testbed is provided in Section 5. Finally, Section 6 concludes the work.

2. THEORETICAL ALGORITHM BACKGROUND

2.1 Deep Q-Learning (DQL)

The Q-Learning approach itself was first proposed by Watkins.[15] Q-Learning, according to Watkins, is a value-based method that can learn optimal policies in environments with delayed feedback. Deep Q-Learning, on

the other hand, is a reinforcement learning algorithm that combines the Q-learning algorithm with deep neural networks, and is used for training artificial agents to perform complex tasks[16].[17] In DQL, a deep convolutional neural network is used to approximate the optimal value function:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi], \tag{1}$$

This is the maximum sum of rewards r_t discounted by γ at each timestep t, bounded by policy $\pi = P(a|s)$, after making an observation (s) and taking an action (a). This is to ensure in long term that the agent will choose actions such that future cumulative reward is maximized. Thus at each i iteration of the Q-learning update the following loss function is used:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')}[r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)], \tag{2}$$

where (s, a, r, s') are samples of experience replay, θ_i the parameters of the Q-network at iteration i, and θ_i^- the network parameters used to compute the target at iteration i.

2.2 ϵ-greedy Exploration

ϵ-greedy exploration is a popular strategy in RL that balances exploration and exploitation. In this strategy, the agent chooses the action that yields (1) (i.e., the greedy action) with probability $1 - \epsilon$, and a random action (i.e., an exploratory action) with probability ϵ. The value of ϵ determines the degree to which the agent explores the environment. A high value of ϵ leads to more exploration, while a low value of ϵ leads to more exploitation. Typically, ϵ is gradually decreased over time as the agent learns more about the environment[18].[19] In this paper, ϵ will be dynamically adjusted based on several metrics of the agent in this project.

2.3 Dynamic Window Approach (DWA) Path Planner

The DWA will be used as a path planner that aids the DQL agent during the physical training process. The path planner is a motion planning algorithm used in robotics and autonomous vehicles to navigate in dynamic and uncertain environments. The DWA algorithm works by generating feasible trajectories for a robot based on its current velocity and the surrounding environment. The algorithm operates by defining a dynamic window, which is a subset of the robot's velocity space that takes into account the robot's maximum velocity and acceleration limits. The dynamic window is then used to generate a set of candidate trajectories for the robot to follow. Each trajectory is evaluated based on its proximity to obstacles, its distance from the goal, and its consistency with the robot's kinematic and dynamic constraints. The objective function G for the evaluation is as follows:

$$G(v,\omega) = c_1(c_2\ heading(v,\omega) + c_3\ dist(v,\omega) + c_4\ velocity(v,\omega)), \tag{3}$$

where c_1, c_2, c_3, c_4 are the constants to be determined by optimization, v the linear velocity, ω the angular velocity, $heading(v,\omega)$ the target heading, $dist(v,\omega)$ the distance to the closest obstacle on the trajectory, and $velocity(v,\omega)$ the forward velocity of the robot. After evaluating each candidate trajectory, the algorithm selects the trajectory that maximizes the objective function and executes it. The algorithm repeats this process at each time step, allowing the robot to adjust its trajectory in response to changes in the environment. The DWA algorithm is designed to be computationally efficient and can operate in real-time on resource-limited robots. It is widely used in robotics and autonomous vehicles, including mobile robots, drones, and self-driving cars.[12]

2.4 Scenario-Based DQL

Scenario-Based DQL is a DQL method that combines Scenario-Based Modeling (SBM)[20] with DQL. SBM is a method designed to aid the development of reactive systems that will behave as humans would expect them to under different conditions. To lower the agent violation frequency during training, the reward r_t is modified as follows at each time step t:

$$\tilde{r_t} = \begin{cases} \alpha r_t - \Delta & \text{if } a_t \text{ not permitted} \\ r_t & \text{otherwise} \end{cases}, \tag{4}$$

where $\tilde{r_t}$ is the DQL reward modified at time t, a_t the action made by agent at time t, α a constant in [-1, 1], and Δ the punishment. Now the DQL reward at each time step is decreased by some punishment if the agent performs an action that is not permitted. This approach will be used for comparative studies with the proposed approach.

3. PROPOSED METHODS

In this section we will describe the integration of Dynamic Epsilon Adjustment method for simulation training and the Path Planner Assisted method for physical experiments.

3.1 Agent

For both the simulation and physical experiments study, a TurtleBot machine learning development package for ROS (Robot Operating System) has been used.[21] It is a collection of software tools and libraries that enable the development and deployment of machine learning algorithms on TurtleBot robots, and provide 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 DQL agent is assigned a random goal. The agent then uses the laser data collected by its LiDAR sensor to learn its surroundings so it will be able to navigate to the goal. This demonstration is shown in Figure 1. The agent takes LiDAR scan data of size 360, current heading to the goal, current distance to goal, current distance to closest obstacle, and current heading to 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. The heading of the agent with respect to any point in space takes values in $[-\pi\ rad, \pi\ rad)$, where 0 rad is when the agent is facing directly at a point of interest, $-\pi$ rad or π rad being the agent facing away from the point of interest. The heading to the point of interest increases as the agent turns clockwise towards the point, decreases as the agent turns the other way. The agent can perform 5 actions as shown in Figure 1. Action 3 is just going forward ($v = 0.15\ m/s$, $\omega = 0\ rad/s$). Actions 1 and 2 are turning faster ($v = 0.15\ m/s$, $\omega = 1.5\ rad/s$) and slower ($v = 0.15\ m/s$, $\omega = 0.75\ rad/s$) to the left while moving forward. Actions 4 and 5 are turning slower ($v = 0.15\ m/s$, $\omega = -0.75\ rad/s$) and faster ($v = 0.15\ m/s$, $\omega = -1.5\ rad/s$) to the right while moving forward. So the agent in this case can only move forward.

As can be seen in Figure 2, the agent is made up of an input layer followed by a ReLU activation function, a hidden layer of 64 nodes followed by another ReLU function, a dropout layer with a dropout rate of 0.2, and an output layer followed by a linear activation function that outputs 1 of the 5 actions at each step.

3.2 Episode Termination Conditions

The agent is assigned one goal at a time during an episode. After a goal is reached, the agent will navigate towards the next immediately without going back to its reset position (coordinates $(0, 0)$). So there can be cases where the agent reaches 0 or more than 1 goals per episode. An episode ends when the agent spends more than 500 steps to navigate to the goal, or it collides with walls/obstacles. Afterwards the agent will reset back to its reset position instantaneously for the next episode in the simulation. For the physical experiments, the robot agent will be guided back to its reset position by the DWA path planner. And the DQN weight parameters are transferred to the next episode for both the simulation and physical experiments.

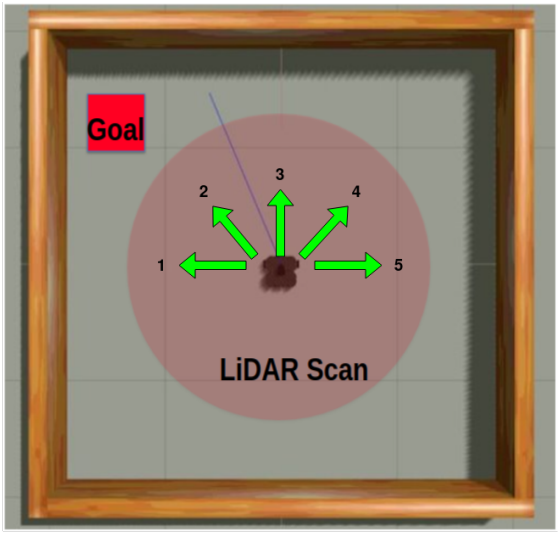


Figure 1. DQL agent learning its surrounding using LiDAR data while navigating to the goal. The agent has 5 actions available in this environment.

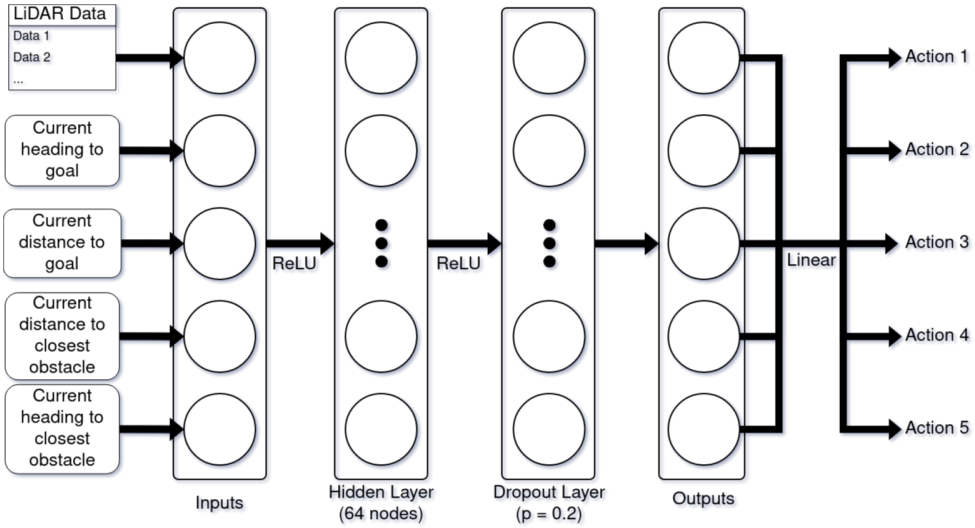


Figure 2. Structure of the DQL agent. The agent takes 5 readings of its surroundings as inputs. The LiDAR data input is composed of 360 infrared readings of distances between the agent and the obstacles around it. The other 4 inputs are of single values. The inputs are sent through a ReLU activation function to a hidden layer of 64 nodes. Afterwards, they are sent through another ReLU activation function to a dropout layer with a dropout rate of 0.2. The output is then sent through a linear activation function, which would be either of the 5 action values.

3.3 Dynamic Epsilon Adjustment for Simulation

When performing DQL, it was observed that the agent could circle in place very often after long episodes of training. To decrease the frequency of this behavior, the exploration rate ϵ was adjusted for each episode based on the goal rate and violations count (circling-in-place behavior) of the current episode, and the trend of the overall goal rate. The trend of the overall goal rate is calculated by the finding the difference between the current goal rate of the episode and the goal rate from the last episode. These metrics are calculated at the end of each episode. If the latest goal rate is at most 0.5, then ϵ is decayed by 1%. Otherwise both the trend of the goal rate and violations count of the current episode will be checked. If the goal rate is decreasing and the violations

count is at most 10, then ϵ is decayed by 1%. If the goal rate is decreasing but the violations count is more than 10, then the value of ϵ is checked. The value of ϵ is also checked when the goal rate is not decreasing and the violations count is more than 10. Since there could be cases where ϵ increases to more than 1, a condition has been set to prevent that from happening. If ϵ is already at least 1, it will be set back to 1. Otherwise, ϵ is increased by 1%. If neither is the goal rate decreasing nor is the violations count more than 10, ϵ remains the same. This workflow can be seen in Figure 3(a). This dynamic adjustment of ϵ allows tunable degrees of action exploration based on the agent performance metrics for the current episode, rather than just statically less action exploration over time. This helps lowering average violations count in the long run.

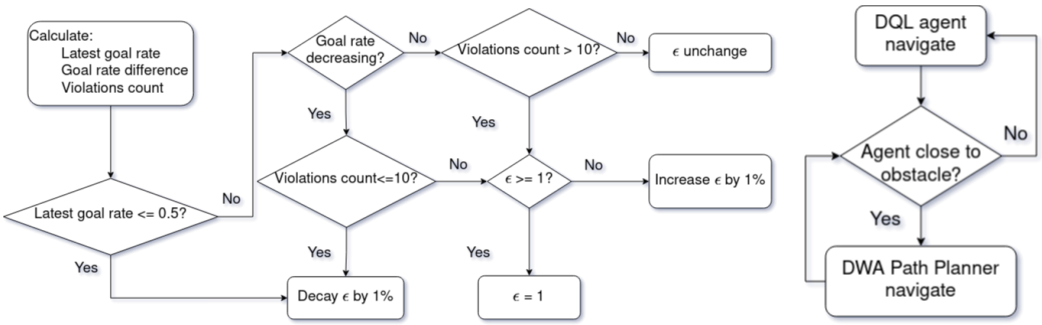


Figure 3. 3(a): Several metrics of the agent such as its latest goal rate, goal rate difference, and violations count are calculated at the end of the latest episode. They're then checked for several combinations of conditions in order for ϵ to be adjusted properly. 3(b): At each episode step, the control of the agent can switch between the DQL and the DWA path planner depending on the current distance between the agent and the closest obstacle. If the agent is close to an obstacle ($\leq 17\ cm$), the DWA path planner will take control of the agent.

3.4 DWA Path Planner Integration for Physical Experiments

For the physical training, a DWA path planner is used to improve the DQL training speed. The DWA path planner takes the generated SLAM map in the form of an occupancy map, where each grid has a value of 1 or 0. A grid has a value of 1 if there's an obstacle in the grid, 0 otherwise. The DWA path planner implemented here functions as explained in section 2.3, and is only activated when the DQL agent is close to an obstacle ($\leq 17\ cm$). This path planner activation check takes place at each step during an episode. This logic flow can be seen in Figure 3(b).

4. SIMULATION STUDY

4.1 ROS Simulation Setup

The environment used for this framework was created using the Gazebo simulator,[22] which provides realistic robotic movements, a physics engine, and the generation of sensor data combined with noise. The agent is assigned one goal randomly at a time within the environment shown in Figure 1 for each episode in the simulation. The environment has an area of 16 m^2. Note that there are no obstacles in this environment. The parameters used for the training are listed in table 1 of Appendix A.1. These parameters are set to ensure that the agent can achieve goal rate convergence within the shortest amount of episodes for all cases. Since Scenario-Based DRL is also implemented for the DQL agent as a comparison. The constraint imposed is that the agent has to avoid 7 consecutive right or left turns. The agent receives a punishment of 4.5 for every violation against the constraint. The theoretical implementation of this was explained in section 2.4.

4.2 Results and Analysis

To evaluate how the agent performed under the proposed method, we have conducted 818 episodes of training with the proposed and state-of-the-art methods. We use the same network architecture (see section 3.1), hyper-parameter values (see table 1) and learning procedure throughout to demonstrate that our approach drastically

lowers the average violations count and improves the agent's goal rate by only dynamically adjusting ϵ for each episode based on various performance metrics of the agent. We compare our method with the traditional DQL and Scenario-Based DRL methods from the RL literature on the same simulation training set-up mentioned in section 4.1. As illustrated by Figure 4, our method outperforms the existing RL methods on reducing average violations count per episode without the need of imposing additional constraints that punish the agent. Furthermore, our method yield the fastest convergence in terms of agent goal rate (see sub-figure 4(c)), and the most total number of goals reached of all 3 methods compared at the end of the 818 training episodes (see sub-figure 4(b)). As can be seen in sub-figure 4(a), the average number of violations per episode increases for the first 200 training episodes across all 3 methods. This is due to the agent recognizing the violation (circling-in-place) is the action that maximizes the reward, and the goal rate hasn't reached 0.5 (see section 3.3). However after the first 200 episodes, the average violations count per episode decreases almost monotonically only for the proposed method (the goal rate exceeded 0.5). This is due to agent being able to explore more or less options of actions depending on its current performance metrics, instead of exploring increasingly limited options of actions as the training goes on. This can be seen in Figure 4(d), where ϵ is decayed statically over time in traditional DQL and Scenario-Based DRL, but is instead adjusted dynamically based on different performance metrics of the agent in our method.

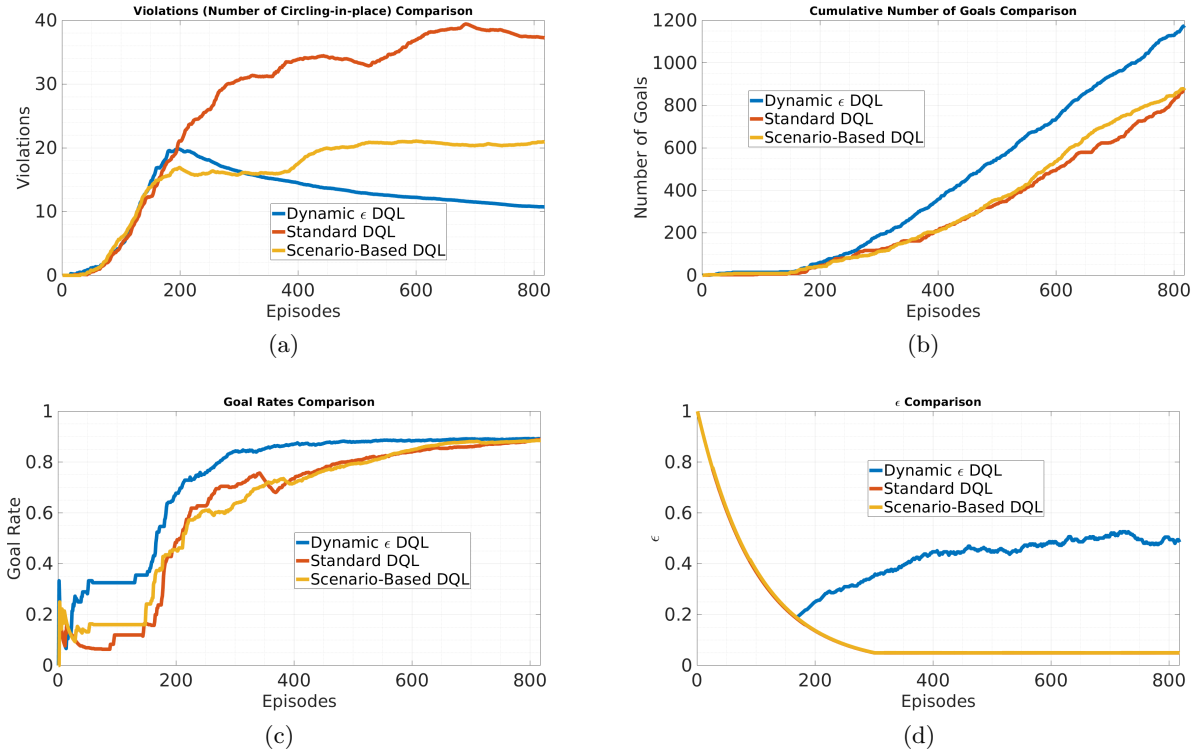


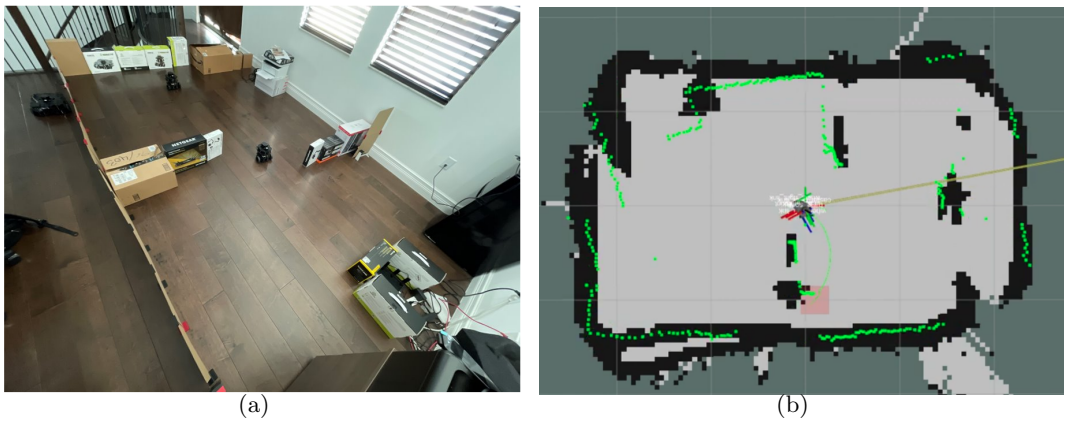
Figure 4. 4(a) Violations count plot comparison for different methods. 4(b) Total number of goals reached over all episodes. 4(c) Goal rate plot comparison for different methods. 4(d) ϵ comparison, where DQL and constraint-based learning are overlapping with each other

5. PHYSICAL EXPERIMENT STUDY ON AN AUTONOMOUS ROBOT TESTBED

5.1 Physical Environment Setup

We develop an autonomous robot test-bed to verify the effectiveness of the designed method. The goals are hard-coded to be randomly assigned under the condition that they don't overlap with the walls and obstacles within the test-bed. The test-bed is shown in Figure 5(a) with an experiment dimension of 3.75 $m \times 2.75$ m. 2

walls, a static robot, and a router walled by 3 boxes have been added to this maze to act as obstacles. The DWA global path planner along with SLAM mapping have also been integrated with this physical training set-up as shown in figure 5(b). The DWA path planner parameters used for the physical training are listed in table 2 of Appendix A.2. These are preset parameters which haven't been modified. We are using Turtlebot as a robot agent for the physical experiments. TurtleBot,[23] developed by Open Robotics, is based on the ROS (Robot Operating System), a popular open-source framework for building robots. The robot is equipped with various sensors, such as a 360-degree LIDAR, a camera, and an IMU (Inertial Measurement Unit), that allow it to perceive its environment and navigate autonomously.



(a) (b)

Figure 5. 5(a): The physical maze. Dimension = 3.75 $m \times 2.75$ m. Environment size ≈ 10.31 m^2. 5(b): SLAM mapping with DWA path planner.

5.2 Results and Analysis

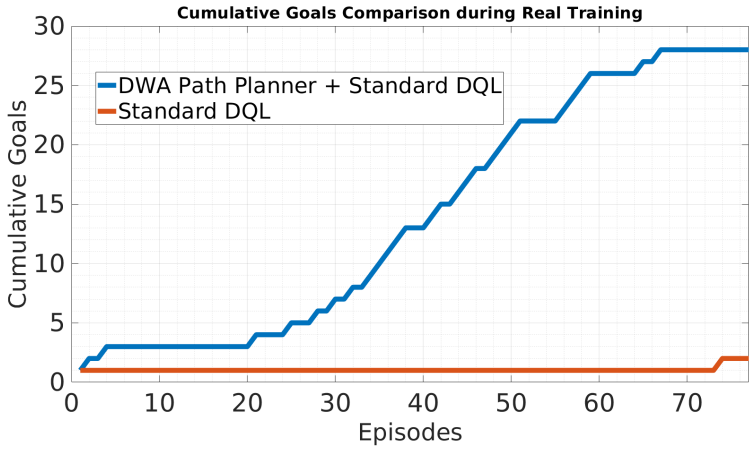


Figure 6. Cumulative goals comparison plot between the proposed method and standard DQL method. The DQL agent integrated with DWA path planner reached 28 goals within 77 episodes. Whereas standard DQL agent alone reached only 2 goals in this timeframe.

To evaluate how well the agent can be integrated with a DWA path planner under the proposed method, we have conducted 77 episodes of physical experiments with the path planner enabled and disabled cases (standard DQL). Due to battery constraint, the number of physical experiment episodes is less than the simulation. We used the same network architecture (see section 3.1), hyperparameter and DWA parameter values (see tables 1 and 2) and learning procedure throughout to demonstrate that our approach drastically increases the total

number of goals reached across all episodes. We compared our method with the standard DQL method from the RL literature on the same physical experiment set-up mentioned in section 5.1. As illustrated by Figure 6, our method outperforms the traditional DQL methods on reaching the most total number of goals throughout the episodes. This is due to the agent having less frequent collisions via the help of DWA path planner as explained in section 3.4.

6. CONCLUSION

In this work, we've developed the Dynamic Epsilon Adjustment method that is implemented in an ROS Gazebo simulation environment. The new method is able to decrease the average violations count per episode while also improving the agent training performance in terms of total number of goals reached and goal rate convergence. In contrast to previous work, our method allows tunable degrees of action exploration based on agent performance metrics of the current episode. Furthermore, we've integrated a DWA path planner with the DQL agent to improve its training performance in a physical experiment setting. Depending on the distance between the agent and the closest obstacle, the control of the agent can now switch between the DQN and the DWA path planner at each step. Because of this, the agent now has less collisions in physical experiments. Our approaches for both the simulation and physical DQL training have demonstrated degrees of improvements from previous works.

APPENDIX A. PARAMETERS

A.1 Physical and Simulation Training Parameters

Table 1. Training Parameters

Hyperparameter	**Value**
Replay memory size	1000000
Target network update frequency	2000
Discount factor	0.99
Learning rate	2.525×10^{-4}
Starting ϵ	1.00
Minimum ϵ	0.05
Batch size	64
Episodes	818
Max steps per episode	500

A.2 DWA Path Planner Parameters

Table 2. DWA Parameters

Parameter	**Value**
Maximum acceleration in the x direction	2.5 m/s^2
Maximum acceleration in the y direction	2.5 m/s^2
Maximum angular acceleration	3.2 rad/s^2
Maximum velocity in the x direction	0.55 m/s
Minimum velocity in the x direction	0 m/s
Maximum angular velocity	1.00 rad/s
Minimum angular velocity	-1.00 rad/s

ACKNOWLEDGMENTS

This work was partially supported by the National Science Foundation under Grant 1947418, 2047064 and 2047010.

REFERENCES

[1] Dayal, A., Cenkeramaddi, L. R., and Jha, A., "Reward criteria impact on the performance of reinforcement learning agent for autonomous navigation," *Applied Soft Computing* **126**, 109241 (2022).

[2] Yerushalmi, R., Amir, G., Elyasaf, A., Harel, D., Katz, G., and Marron, A., "Scenario-assisted deep reinforcement learning," *arXiv preprint arXiv:2202.04337* (2022).

[3] Corsi, D., Yerushalmi, R., Amir, G., Farinelli, A., Harel, D., and Katz, G., "Constrained reinforcement learning for robotics via scenario-based programming," *arXiv preprint arXiv:2206.09603* (2022).

[4] Google Inc., Mountain View, C. U., "Methods and apparatus for reinforcement learning," (U.S. Patent 2015/0100530 A1, Apr. 2015).

[5] Van Hasselt, H., Guez, A., and Silver, D., "Deep reinforcement learning with double q-learning," in [*Proceedings of the AAAI conference on artificial intelligence*], **30**(1) (2016).

[6] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N., "Dueling network architectures for deep reinforcement learning," in [*International conference on machine learning*], 1995–2003, PMLR (2016).

[7] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K., "Asynchronous methods for deep reinforcement learning," in [*International conference on machine learning*], 1928–1937 (2016).

[8] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O., "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347* (2017).

[9] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971* (2015).

[10] de Jesus, J. C., Kich, V. A., Kolling, A. H., Grando, R. B., Cuadros, M. A. d. S. L., and Gamarra, D. F. T., "Soft actor-critic for navigation of mobile robots," *Journal of Intelligent & Robotic Systems* **102**(2), 31 (2021).

[11] Nguyen, T. T., Nguyen, N. D., and Nahavandi, S., "Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications," *IEEE transactions on cybernetics* **50**(9), 3826–3839 (2020).

[12] Fox, D., Burgard, W., and Thrun, S., "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine* **4**(1), 23–33 (1997).

[13] Andy, W.-C. C., Marty, W.-Y. C., Ni, Z., and Zhong, X., "An automated statistical evaluation framework of rapidly-exploring random tree frontier detector for indoor space exploration," in [*2022 4th International Conference on Control and Robotics (ICCR)*], 1–7, IEEE (2022).

[14] Jiang, H., Esfahani, M. A., Wu, K., Wan, K.-w., Heng, K.-k., Wang, H., and Jiang, X., "itd3-cln: Learn to navigate in dynamic scene through deep reinforcement learning," *Neurocomputing* **503**, 118–128 (2022).

[15] Watkins, C. J. C. H., "Learning from delayed rewards," (1989).

[16] 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* **518**(7540), 529–533 (2015).

[17] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P., "Trust region policy optimization," in [*International conference on machine learning*], 1889–1897, PMLR (2015).

[18] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M., "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602* (2013).

[19] Sutton, R. S. and Barto, A. G., [*Reinforcement learning: An introduction*], MIT press (2018).

[20] Damm, W. and Harel, D., "Lsc's: Breathing life into message sequence charts," in [*Formal Methods for Open Object-Based Distributed Systems: IFIP TC6/WG6. 1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS), February 15–18, 1999, Florence, Italy*], 293–311, Springer (1999).

[21] ROBOTIS-GIT, "turtlebot3_machine_learning." https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning (2018).

[22] Gazebo, "Open Source Robotics Foundation." http://gazebosim.org/ (2014). [Online; accessed 24-Apr-2022].

[23] Foundation, O. S. R., "Turtlebot." https://www.turtlebot.com/ (2014). [Online; accessed 6-Apr-2023].