Reinforcement Learning for Obstacle Avoidance

Arvinder Singh*, Uday Raghuvanshi†
CS 5180 Reinforcement Learning, Northeastern University
Email: *singh.arv@northeastern.edu, †raghuvanshi.u@northeastern.edu

***Abstract*—This study addresses the fundamental challenge of autonomous obstacle avoidance in mobile robots through Reinforcement Learning (RL). The project focuses on developing a system that facilitates safe navigation for robots and autonomous vehicles, within dynamic environments. The methodology employed involves the utilization of RL algorithms and leveraging sensor inputs to make informed decisions. The study explores and compares various RL methods and parameter settings to optimize safe navigation while minimizing collisions. Algorithms include Q-Learning, Deep Q-Learning and Double Deep Q-Learning.**

Quantitative analyses are conducted to evaluate the performance of different RL approaches in terms of successful navigation and collision avoidance. Metrics including plots concerning episodes and time steps are implemented to gauge performance. Additionally, a virtual environment containing obstacles is simulated to visualize trained policies. Results indicate that RL-based models exhibit considerable promise in achieving safe navigation. The comparison between different methods showcases differences in adaptability and efficiency and the strengths and limitations of each approach.

This study contributes valuable insights into the effective utilization of RL for autonomous obstacle avoidance, offering potential applications in real-world scenarios.

I. Problem Addressed

We developed an autonomous obstacle avoidance system for mobile robots. This includes autonomous vehicles, warehouse robots, drones etc. The challenge is to design a system that makes decisions to ensure safe navigation of robot in the simulated environment, and also in simulated unseen environments. The key components of our system are:

1) **Environment**- The physical operating space of robot. It includes obstacles, path to follow and sensor inputs. We used Pygame and Pymunk to simulate and visualize policies.
2) **Robot/Agent**- The mobile robot/ agent would be equipped with sensors to perceive surroundings. It would take decisions to avoid obstacles. The robot is equipped with 3 radar sensors which help it to take decisions in real-time.
3) **State space**- It is an array of size 3, indicating the sensor readings from the 3 sensors.
4) **Action space**- It includes the set of actions that robot could take. The action space is discrete with 3 possible values: Left, Right and do nothing.
5) **Reward**- This would provide feedback to our agent based on the outcomes of actions it takes. If agent crashes, a reward of -500 is received. Else, the reward is proportional to the distance from obstacles. Through the reception of rewards based on action outcomes, the system establishes a feedback loop.

Input to the algorithm would include the data received from sensors as the robot moves in the environment. These would be the state observations. Reward obtained after performing the actions would also be input. The output would be the next action (right, left or do nothing) taken by robot that would help it to navigate.

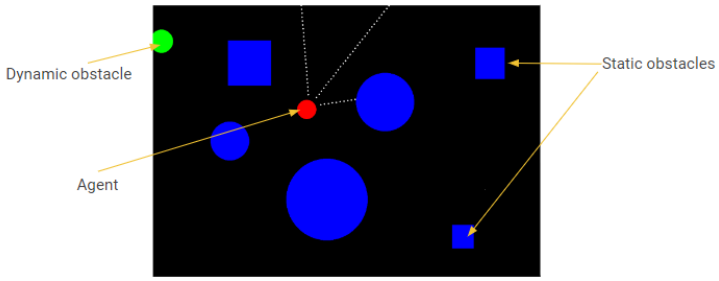

Fig. 1: *Simulated Environment*

II. Methods Used & Results

We started small by implementing Q-Learning, a classical Reinforcement algorithm. It formed the basis for our approach to autonomous obstacle avoidance. However, we soon realized that Q-learning would not give us optimal results. It's because many classical algorithms such as Q-learning work with discrete state spaces and the state space in our implementation has potentially infinite number of states. For our project, we implemented the following reinforcement learning algorithms which would helped us understand the performance of classical vs Deep learning based reinforcement learning methods:

1) **Q-Learning** is a model-free reinforcement learning algorithm suitable for discrete state and action spaces. We aimed to train the agent to make decisions in an environment by learning the quality of different actions, denoted by Q-values. The algorithm built a Q-table that stores the values for state-action pairs. Through an iterative process of exploration and exploitation, the agent learns to take actions that maximize its expected cumulative reward over time. However, Q-Learning faced limitations in handling complex environments due to its reliance on a tabular representation, which became infeasible for large state spaces.
2) **Deep Q-Network** addressed the limitations of Q-Learning by employing a neural network, allowing for more complex and continuous state spaces. This approach leveraged a deep neural network to approximate

Q-values, enabling the agent to learn a representation of the Q-function. The network took states as inputs and predicted Q-values for each possible action. Additionally, experience replay networks were employed to stabilize learning and improve convergence.

3) **Double Deep Q-network** built upon the foundation of DQN by mitigating overestimation bias in Q-value estimates. It did so by decoupling action selection from value evaluation, using two separate networks—an online network for action selection and a target network for value estimation. By employing a different network to determine the action (target network), DDQN reduced overestimation errors encountered in the original DQN, resulting in more stable and accurate Q-value estimations.

III. Q-Learning

Q-Learning, a fundamental classical reinforcement learning algorithm, initiates by initializing Q-values for state-action pairs. In our implementation, we intialzed the Q values as a dictionary where each state is a key and value is a list of size 3, corresponding to the Q value associated with each of the three actions. This process starts with exploration and exploitation, balancing the agent's choices between trying new actions (exploration) and selecting the best-known actions (exploitation). We selected an epsilon value of 0.1. Through agent-environment interaction, the agent selects actions based on the current state and receives feedback in the form of rewards, transitioning to new states. Q-values are then updated using the Bellman equation, considering immediate rewards and estimating future rewards:

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot (R + \gamma \cdot \max_{a} Q(s',a'))$$

Where:

- (s,a): Represents the state-action pair.
- $Q(s,a)$: Represents the Q-value for the state-action pair.
- α: Learning rate.
- r: Immediate reward obtained after taking action a in state s.
- γ: Discount factor representing future rewards.
- $\max_{a'} Q(s',a')$: Represents the maximum Q-value for the next state s' considering all possible actions a'.

Thus, it iteratively refines the Q-values to approximate an optimal policy. This iterative process continues until convergence, aiming to capture the best policy for the given environment. Upon completion, the agent executes its learned policy for decision-making, marking the end of the learning phase. This cycle defines Q-Learning, wherein the agent learns from experiences to make optimal decisions in unknown or changing environments.

In our exploration of Q-learning for obstacle avoidance, several challenges arose, hindering its anticipated performance. The algorithm struggled to strike a balance between exploration and exploitation, often getting stuck in suboptimal policies, which limited its ability to discover optimal actions efficiently. The epsilon value was fixed for the whole training period which hindered performance. Despite iterative adjustments, Q-learning frequently plateaued at suboptimal solutions, showcasing limitations in adapting to complex environments. The discretization of the state space was another big hindrance that affected its ability to generalize. These limitations led us to explore alternative approaches like DQN and DDQN to improve the algorithm's performance in navigating dynamic environments.

When the environment was visualized using the learned policy using Q-Learning, the performance was poor. The agent kept rotating in place until the dynamic obstacle stuck it and the episode ended. Quantitative analysis of Q learning also showed poor results.

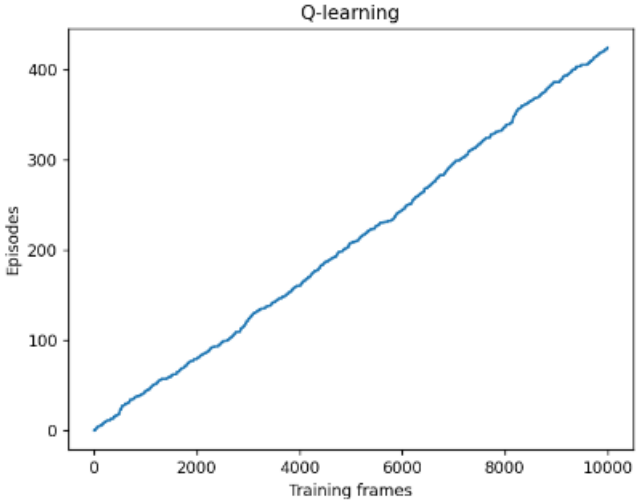

Fig. 2: *Simulated Environment*

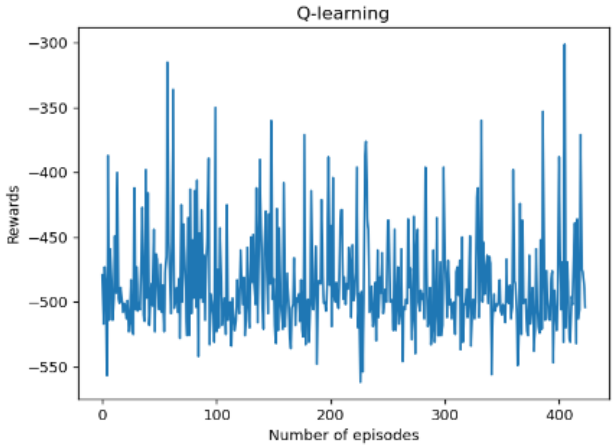

Fig. 3: *Simulated Environment*

Fig 2 shows the number episodes to training frames. Ideally, the graph should have flattened more as the number of frames increased. However, the agent did not learn anything, and the episodes got terminated quickly even after a large number of time steps. This is a result of not having every possible state-

value pair in the Q table. Similar output can be seen when we plot the rewards as obtained in each episode.

IV. Deep Q-Learning

Next, we moved on to Deep Q-Learning Network(DQN). It extends Q-Learning by leveraging deep neural networks to approximate the Q-value function. It's a powerful algorithm used in reinforcement learning, specifically in environments with high-dimensional state spaces such as raw sensor inputs in our case. Unlike traditional Q-Learning, which uses tabular methods to represent the Q-values for each state-action pair, DQN employs a neural network to estimate these values.

Deep Q Learning (DQN) is an extension of Q-learning that leverages neural networks to approximate the Q-value function. In the context of our project, the Q-value function represents the expected cumulative reward of taking action a in state s and then following the optimal policy thereafter.

The Q-value function is approximated using a neural network with parameters θ:

$$Q(s, a; \theta)$$

The loss function for DQN, known as the temporal difference error, is defined as:

$$L(\theta) = \mathbb{E}\left[\left(R + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

where: - R is the immediate reward, - γ is the discount factor, - s' is the next state, - a' is the next action, - θ^- are the parameters of the target Q-network.

The neural network is trained to minimize this loss, updating the parameters θ using an optimizer.

The training update rule for the Q-network is given by:

$$\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta)$$

where: - α is the learning rate, - $\nabla_\theta L(\theta)$ is the gradient of the loss with respect to the parameters.
At its core, the algorithm aims to learn a function Q(s, a) that estimates the expected future rewards for taking action 'a' in state 's'. The neural network is trained to minimize the difference between the predicted Q-values and the target Q-values, usually computed using the Bellman equation. This process involves iteratively updating the network's weights to improve its predictions.

DQN operates by collecting experiences SARS (state, action, reward, next state) in an experience replay buffer. From this buffer, it samples minibatches to train the neural network, reducing correlations between consecutive experiences and improving training stability. Additionally, DQN utilizes a separate target network to stabilize training, where the target network's weights are periodically updated with the learned network's weights.

The integration of deep neural networks in Q-Learning helped us handle complex state representations, enabling it to excel in environments with high-dimensional input spaces. However, it also introduces challenges, such as instability due to overestimation bias and the need for careful exploration strategies to effectively explore large state spaces. The model's versatility was evident in its application to diverse obstacle avoidance scenarios, showcasing a capacity to learn robust strategies beyond the training scenarios.

DQN faced challenges in effectively generalizing its learning in highly complex environments with multiple obstacles or rapidly changing dynamics. The model occasionally exhibited erratic behavior, struggling to consistently apply the learned obstacle avoidance strategy.The exploration-exploitation trade-off posed challenges when exploration was crucial. DQN, in certain scenarios, tended to prioritize exploitation over exploration, potentially missing alternative, more effective actions.

In specific scenarios, DQN showcased unexpected and erratic behavior, deviating from the learned optimal policy. This unpredictability posed challenges in ensuring reliable obstacle avoidance in all possible scenarios, indicating potential limitations in the model's decision-making under certain conditions.The occasional erratic behavior was found to be sensitive to hyperparameter settings. Fine-tuning hyperparameters became crucial to achieving a balance between exploration and exploitation, highlighting the importance of parameter optimization for the model's robust performance.

We analyzed the performance of Deep Q Learning (DQN) for autonomous obstacle avoidance by plotting graph of smoothed distance versus the number of frames(Fig. 4). We were able to validate that the learning process was not only effective but surprisingly swift, surpassing initial expectations. However, a notable observation surfaced during this analysis – the presence of considerable randomness in the results.

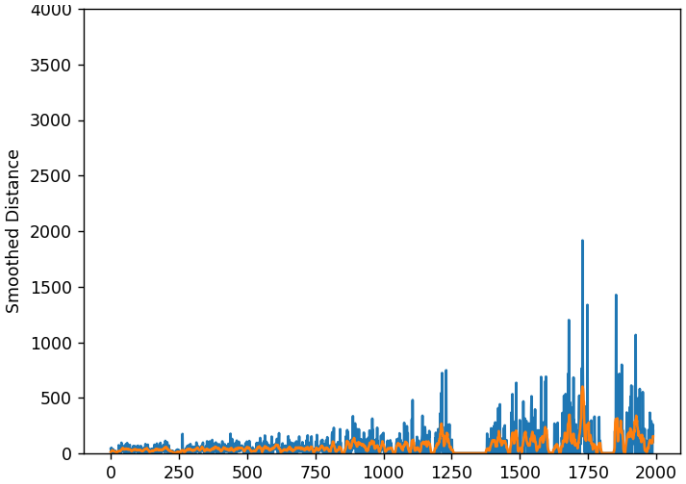

Fig. 4: *Smoothed Distance vs Number of frames*

This randomness stemmed from multiple sources. Firstly, the assortment of obstacles chosen at random and the initial position of the car introduced variability into each learning scenario. Additionally, the inherent stochastic nature of the algorithm, particularly the epsilon-greedy exploration strategy, contributed to the observed unpredictability. Even when the

model was at its least random, with a mere 10% chance of selecting a random action (controlled by epsilon), the influence of this randomness on the outcomes was still significant.

This aspect emphasizes the nuanced nature of the learning process, where external factors and stochastic elements play a substantial role in shaping the model's behavior. Acknowledging and understanding this variability is crucial for interpreting and refining the results of our autonomous obstacle avoidance system powered by Deep Q Learning.

A. *Hyperparameter Tuning Details*

With the loss reading available, the focus shifted to hyperparameter tuning. The following parameters were explored:

- **Buffer Size Impact** The smaller the buffer size, the lower the loss but with greater variance. A large 500,000 replay buffer showed minimal variance but limited learning. The optimal balance was found at 50,000, offering lower loss with manageable variance.
- **Network Size Impact** Larger networks exhibited lower loss; however, they also had lower max distance and average distance per game. This suggests that larger networks may train more slowly, but it isn't necessarily a drawback.
- **Batch Size Impact** Larger batch sizes resulted in lower loss and lower variance. Unfortunately, larger batch sizes led to slower training.
- **Gamma Influence** Consistently, a gamma value of 0.9 outperformed 0.95 across the board.

Below we can see the the graph for the best performing DQN revealed that the optimal DQN configuration exhibited several distinctive characteristics. First and foremost, it showcased a relatively lower loss compared to other configurations. This lower loss indicates that the model was successful in minimizing the discrepancy between predicted and actual outcomes during training.

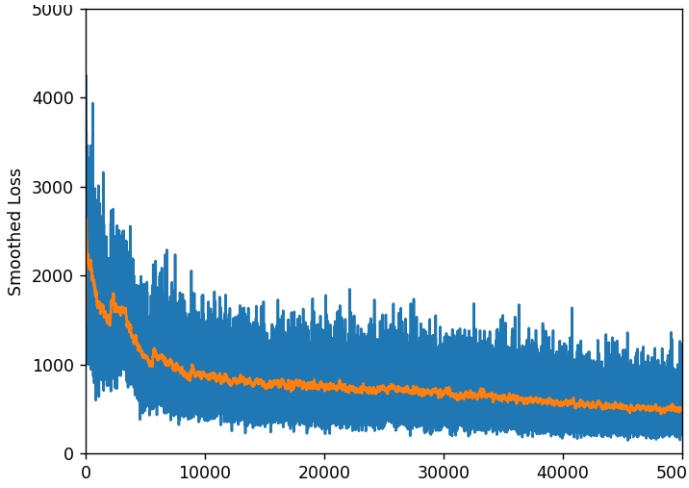

Fig. 5: *Smoothed Loss vs Number of frames*

Additionally, the best-performing DQN displayed low variance in its learning process. This is a crucial aspect, indicating that the model's performance was consistently reliable across different runs and scenarios. Low variance suggests stability in the learning process, which is highly desirable for real-world applications. Moreover, the continuous, albeit slow, learning pattern observed in the best-performing DQN configuration is noteworthy. Continuous learning signifies that the model was adapting and improving over an extended period, reinforcing its ability to handle evolving and complex obstacle avoidance scenarios.

V. Double Deep Q-Learning

To further improve the performance of our system and reduce overestimation bias, we moved on to implement Double deep Q learning with an aim to have an infinite run time of our system. Double Deep Q Learning (DDQN) is an extension of the traditional Deep Q Learning (DQN) algorithm, designed to address a common issue known as overestimation bias. This bias can occur when estimating the value of actions in a state, potentially leading to suboptimal policy learning. DDQN introduces a novel approach by decoupling the selection of the best action from the estimation of its value.

Similar to DQN, DDQN employs experience replay to store and randomly sample past experiences. This facilitates more stable and efficient learning by breaking the temporal correlation between consecutive experiences.DDQN introduces the concept of using two separate Q-value networks — one for selecting actions (the "online" network) and another for evaluating those actions (the "target" network). The online network is responsible for selecting the best action, while the target network is used to estimate the value of that action.

The Q-values are updated using a combination of the reward and the maximum Q-value from the target network, providing a more accurate estimation. This decoupling mitigates the overestimation bias observed in traditional Q-learning approaches. The overestimation bias occurs because the same set of parameters is used to select and evaluate an action. This can lead to overoptimistic Q-value estimates. The loss function for Double DQN is modified as follows:

$$L(\theta) = \mathbb{E}\left[\left(R + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta^-) - Q(s, a; \theta)\right)^2\right]$$

The target Q-network is used to select the action in the next state, mitigating the overestimation bias.

The target Q-network is updated more slowly to stabilize training:

$$\theta^- \leftarrow \tau\theta + (1 - \tau)\theta^-$$

where τ is the target network update rate.

In our analysis of autonomous obstacle avoidance, we employed Double Deep Q Learning (DDQN) and plotted a graph of smoothed distance against the number of frames. The objective was to assess the learning process and compare the performance with that of Deep Q Learning (DQN).The graph(Fig. 6) of smoothed distance versus the number of

frames demonstrated a consistent and effective learning process over time. The increasing trend in smoothed distance indicated that the DDQN algorithm successfully learned optimal policies for navigating the environment and avoiding obstacles.

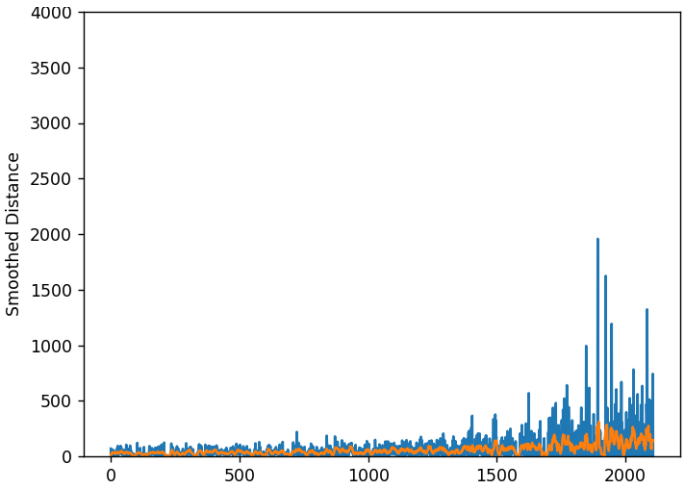

Fig. 6: *Smoothed Distance vs Number of frames*

Notably, the performance of DDQN surpassed that of Deep Q Learning (DQN). The distances achieved by DDQN were consistently higher, signifying better obstacle avoidance. This superiority could be attributed to the enhanced mechanisms of DDQN, such as the reduction of overestimation bias through the use of two Q-value networks. The validation of the learning process was evident in the gradual increase of smoothed distance over the course of training frames.The algorithm demonstrated an ability to adapt and improve its decision-making based on the feedback received from the environment, thereby achieving more proficient obstacle avoidance.

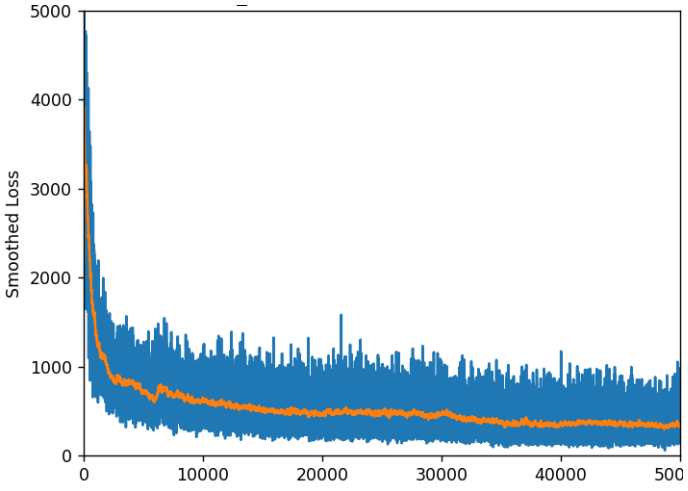

Fig. 7: *Smoothed Loss vs Number of frames*

To delve deeper into the comparative analysis of Double Deep Q Learning (DDQN) and Deep Q Learning (DQN), we carefully examined the smoothed loss versus frames graph. This analysis aimed to bring to light crucial aspects of the learning process and shed light on the distinctive features of DDQN.

DDQN consistently displayed lower loss values compared to its counterpart, DQN. This implies that DDQN was more effective in minimizing the disparity between predicted and target Q-values, resulting in more efficient learning.The observed lower loss in DDQN holds significant implications for its practical application in autonomous obstacle avoidance scenarios. Lower loss indicates improved convergence and accuracy in approximating optimal Q-values, contributing to more reliable decision-making and efficient navigation in dynamic environments.

Despite the inherent complexities of the training process, DDQN demonstrated a remarkable reduction in variance. This stability in the learning process signifies robustness, establishing DDQN as a reliable algorithm for obstacle avoidance.Unlike some learning algorithms that plateau or exhibit erratic behavior, DDQN exhibited continuous learning. The graph depicted a steady decline in loss over time, underscoring the algorithm's ability to adapt and refine its strategies consistently.

The best performing DDQN model, ran for 2 hours without colliding with an obstacle after which it was terminated by the user so we can successfully say that DDQN has an infinite run life. The Model was also able to generalize to unknown environments and dynamic obstacle without any problems.

VI. Moving Average Performance Comparison

In the comparison of the moving averages of five distinct models—P(DDQN), J(DDQN), N(DQN), H(DQN), and D(DQN)—we aimed to assess their performance in autonomous obstacle avoidance. Each model exhibited unique characteristics and configurations, contributing to a comprehensive understanding of their strengths and weaknesses.

A. P(DDQN) - Preferred Configuration

P(DDQN) emerged as the preferred configuration, demonstrating robust and consistent performance. Key attributes are summarized in Table I.

TABLE I: Configuration of P(DDQN)

Parameter	**Value**
Gamma	0.9
Dropout	0.2

B. J(DDQN) - Unusual Behavior

J(DDQN) exhibited unusual behavior with occasional performance peaks but occasional failures. Key characteristics are summarized in Table II.

TABLE II: Configuration of J(DDQN)

Parameter	**Value**
Gamma	0.999

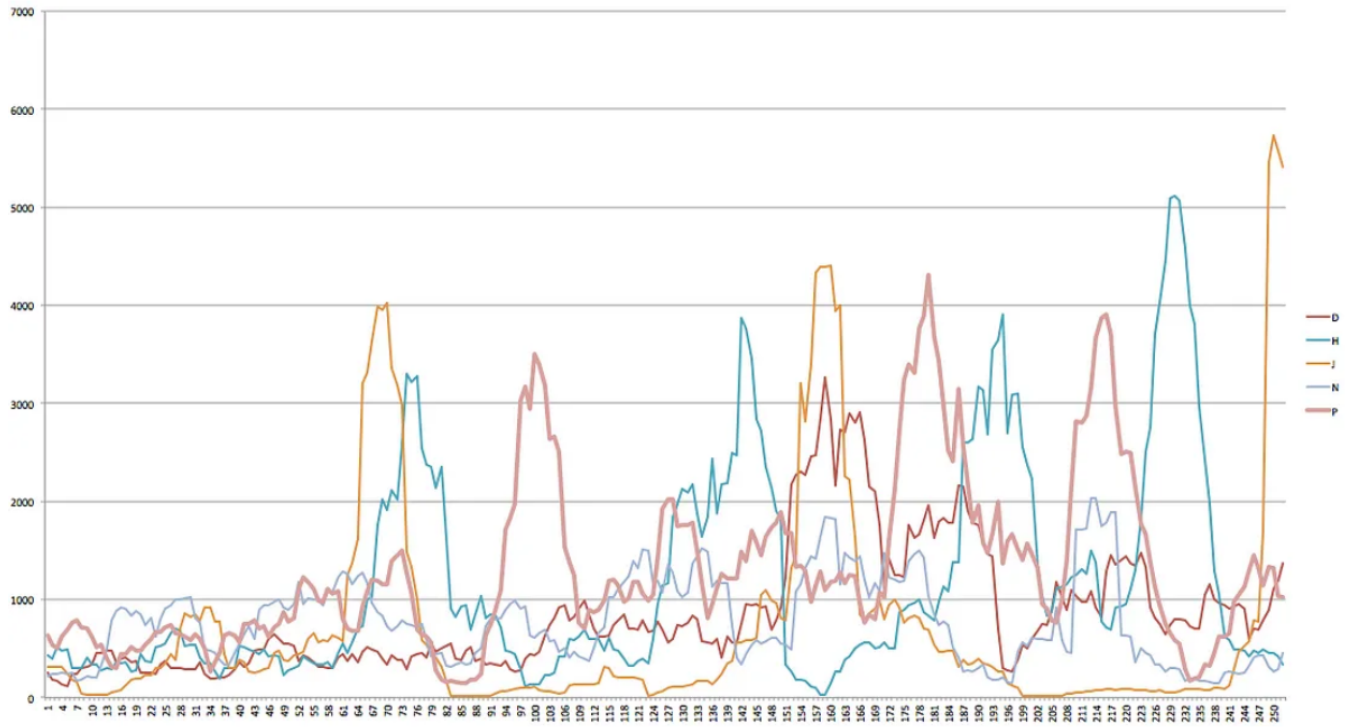

Fig. 8: *Moving Average plot for different models*

C. N(DQN) - Worst Performer

N(DQN) surfaced as the least effective performer in the comparison. Key characteristics are summarized in Table III.

TABLE III: Configuration of N(DQN)

Parameter	Value
Neural Network Structure	Extra hidden layer

D. H(DQN) - Similar to P(DDQN)

H(DQN) demonstrated performance similar to P(DDQN), suggesting a comparable effectiveness. Key attributes are summarized in Table IV.

TABLE IV: Configuration of H(DQN)

Parameter	Value
Gamma	0.95
Dropout	0.2

E. D(DQN) - Intermediate Performance

D(DQN) performed at an intermediate level between H(DQN) and N(DQN). Key characteristics are summarized in Table V.

TABLE V: Configuration of D(DQN)

Parameter	Value
Gamma	0.98
Neural Network Structure	Two extra hidden layers

The detailed comparison among the five models provides valuable insights into the impact of different configurations on autonomous obstacle avoidance. While P(DDQN) emerged as the preferred configuration, the unique characteristics of each model contribute to a nuanced understanding of the interplay between algorithmic choices and performance outcomes. Fine-tuning these configurations based on specific environmental dynamics and task requirements remains an area for further exploration.

VII. Conclusion

After a comprehensive exploration of Q-Learning, Deep Q-Learning (DQN), and Double Deep Q-Learning (DDQN) algorithms for autonomous obstacle avoidance, it's evident that each technique offers unique advantages and challenges.

Q-Learning, a classical reinforcement learning method, struggled in dynamically changing environments, often converging to suboptimal solutions due to limitations in exploration and the discretization of state spaces. The algorithm struggled to balance exploration and exploitation, frequently converging to suboptimal solutions due to a fixed exploration rate. Its visualized performance revealed poor navigation

strategies, frequently leading to collisions with obstacles. It is easier to implement, and faster to train but doesn't give the desired results in complex environments.

Transitioning to Deep Q-Learning (DQN), the integration of neural networks significantly enhanced its capability to handle high-dimensional state spaces. Despite this improvement, DQN faced occasional erratic behavior, especially in scenarios where exploration was crucial. Stability and consistent decision-making remained challenges, impacting its reliability in dynamic environments.

In contrast, Double Deep Q-Learning (DDQN) addressed overestimation bias and showcased notable improvements over both Q-Learning and DQN. By decoupling action selection from value estimation, DDQN reduced variance, exhibited stable learning, and demonstrated a remarkable ability to adapt to changing environments. The best-performing DDQN model exhibited an infinite run life, indicating robustness in handling unknown environments and dynamic obstacles.

The comparative analysis among various model configurations emphasized the importance of parameter tuning. P(DDQN) emerged as a preferable choice, displaying robust and consistent performance in obstacle avoidance scenarios. However, each model configuration unveiled unique characteristics, offering valuable insights into the interplay between algorithmic and parametric choices and how they affect the performance.

Overall, this exploration highlighted the nuanced nature of reinforcement learning algorithms in autonomous obstacle avoidance. We were able to understand the importance of neural network depth and complexity. While each technique presented strengths and challenges, fine-tuning parameters and customizing these approaches for specific environmental dynamics remain critical for their practical application. Further research and optimization efforts using advanced reinforcement learning techniques are essential for enhancing the reliability, adaptability, and robustness of these algorithms in real-world scenarios.

VIII. References

[1] Jaewan Choi, Geonhee Lee & Chibum Lee et al. (2021) Reinforcement learning-based dynamic obstacle avoidance and integration of path planning.

[2] Shumin Feng, Bijo Sebastian, Pinhas Ben-Tzvi et al. (2021) A Collision Avoidance Method Based on Deep Reinforcement Learning

[3] Razin Bin Issa, Modhumonty Das, Md Saferi Rahman, Monika Barua, Md Khalilur Rhaman, Kazi Shah Nawaz Ripon, Md Golam Rabiul Alam et al. (2021) Double Deep Q-Learning and Faster R-CNN-Based Autonomous Vehicle Navigation and Obstacle Avoidance in Dynamic Environment

[4] https://blog.coast.ai/reinforcement-learning-in-python-to-teach-an-rc-car-to-avoid-obstacles-a1d063ac962f

[5] https://medium.com/@sdeleers/autonomous-car-with-reinforcement-learning-part-1-obstacle-avoidance-7c73a2567b7b

[6] Xiaoyun Lei, Zhian Zhang, Peifang Dong et al. (2018) Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning

[7] https://medium.com/@qempsil0914/deep-q-learning-part2-double-deep-q-network-double-dqn-b8fc9212bbb2

[8] https://davidrpugh.github.io/stochastic-expatriate-descent/pytorch/deep-reinforcement-learning/deep-q-networks/2020/04/11/double-dqn.html

[9] https://chat.openai.com/

Link to Code- Google Drive