Q-Learning in Dynamic Enviroment with random movement of Obstacles

Adwait Kulkarni
Robotics and Autonomous Systems
Boston University
adk1361@bu.edu

***Abstract*—Traditional reinforcement learning methods, while effective in static settings, struggle in dynamic and unpredictable scenarios. This paper explores the application of Q-learning, in dynamically changing environments characterized by random obstacle movements. A novel approach is presented where an autonomous robot navigates a 10x10 grid environment, learning to avoid randomly moving obstacles and efficiently reach a designated goal. The project utilizes a modified Q-learning algorithm integrated with an epsilon-greedy strategy, balancing exploration and exploitation. Simulations conducted demonstrate the robot's improved navigation and decision-making abilities. Results indicate the effectiveness of Q-learning in dynamic settings, providing insights into reinforcement learning's potential in unpredictable environments. This work showcases practical applications of Q-learning and opens avenues for future research in complex, real-world scenarios.**

I. Introduction

In the rapidly evolving field of artificial intelligence, the development of autonomous agents capable of navigating complex and dynamic environments remains a significant challenge. This project, titled "Q Learning in a Dynamic Environment with Random Movement of Obstacles," presents an innovative approach to this challenge using reinforcement learning, specifically Q learning. Set in a grid world of a 10x10 environment, the project introduces an autonomous robot tasked with the objective of navigating from a designated start point to a goal position. Complicating this task are twenty obstacles, each exhibiting random movement upon any step taken by the robot, thereby dynamically altering the environment and the path to the goal.

The cornerstone of this project is the application of Q learning [10], a model-free reinforcement learning algorithm. This technique enables the robot to learn an optimal action-selection policy to navigate efficiently through the environment while avoiding collisions with the moving obstacles. The learning process is guided by rewards and penalties, steering the robot towards the goal and away from potential hazards.

This project not only demonstrates the practical application of Q learning in a dynamic setting, particularly in scenarios where environmental unpredictability is a significant factor. This paper begins with explaining the problem in the problem formulation section, followed by a detailed description of the prior work. The subsequent sections discuss the methodology and, simulation. Finally, the paper concludes with an analysis of the results, and future work.

A. Problem Formulation

This problem consists of a 10x10 grid environment where the robot is located at the starting position, and the goal is located at the endpoint in the environment. However, its location is varied later to get a better analysis of the proposed method. There are 20 obstacles in the environment; these obstacles can move in any direction, and their movement is randomized, constituting a dynamic environment. Each time the robot moves or takes a step, the obstacles change their location to a new location unrelated to their previous location, and this new location is completely randomized. Here, we introduce Q learning to make the robot learn to move in this uncertain dynamic environment, teaching it to avoid obstacles and reach the goal given that the robot can only take 100 steps to reach the destination.

II. Prior Work

Reinforcement learning emerged in the late 1970s inspired by behaviorist psychology, focused on mapping situations to actions for maximizing numerical reward signals [2]. Learning is based on experience gained through trial-and-error interactions with an environment. Early RL research, including the popular Q-learning algorithm [10], formalized how to balance exploration of untried actions vs. exploitation of known rewards, a key challenge in RL. Foundational Q-learning studies in the 1990s applied RL to problems like balancing a simulated pole based on a single state input [9]. The cart-pole system demonstrated how Q-learning could independently discover control policies by learning action values (Q-values) for each state based on experienced rewards over repeated episodes. These episodic tasks allowed extensive initial exploration before focused exploitation, establishing basic Q-learning principles.

As computing advanced, RL adoption in real-world domains introduced new complexities of dynamic environments [4](Littman, 2015). Tasks like network routing [1] and robot navigation required more sophisticated balancing of exploration and exploitation. Techniques like eligibility traces became necessary to handle unpredictable state changes [8]. By demonstrating reliability despite unpredictability, these studies motivated wider RL applications. Reinforcement learning methods were first successfully applied to problems with static environments. Q-learning was initially demonstrated for navigating simple static gridworld environments and maze

tasks [10, Chapter 6, Section 4]. These were simple proofs of concepts showing that agents could learn policies without maps or modeling the environments. Biped and quadruped robots also used Q-learning for complex locomotion, tweaking exploration/exploitation to master running, jumping and climbing [3]. However, research quickly progressed to handle more dynamic situations. Ng et al. [7]demonstrated the use of Q-learning for basic maneuvers and inverted flight in remotely controlled helicopter flight. This required learning policies robust to real-world variability and turbulence during testing. Michels et al. [6] applied Q-learning in the context of high speed car racing in simulation. The tracks and driving conditions varied dynamically, requiring agents to adapt their policies online.

III. Methodology

This project employs a reinforcement learning technique, specifically Q-learning, to train a robot to navigate through an environment and reach a designated goal. This approach integrates the epsilon-greedy strategy within the Q-learning framework to balance exploration and exploitation during the learning process. The simulation of this experiment is conducted using MATLAB. Q-learning [10], a model-free reinforcement learning algorithm, enables the robot to learn optimal actions in a stochastic environment. The state space of the environment consists of the robot's position in a grid-like structure, with the goal and various obstacles positioned within this grid. The robot's action space includes four possible actions at each step: moving up, down, left, or right. These movements are the available actions in each state of the environment. A positive reward is assigned when the robot successfully reaches the goal, signifying the completion of its objective. A negative reward is given each time the robot collides with an obstacle, indicating a penalty for undesirable actions. A smaller reward is allocated for each step taken, encouraging the robot to find efficient paths to the goal. The epsilon-greedy strategy is incorporated to balance the exploration of new paths and the exploitation of known paths. This strategy allows the robot to occasionally choose a random action (exploration) instead of the current best-known action (exploitation) based on a predefined probability (epsilon).

Algorithm 1 Modified Q-Learning Algorithm

Initialize Q-table arbitrarily (Q(s, a) for all s, a)
Choose a learning rate α (0 ¡ $\alpha \leq$ 1)
Choose a discount factor γ (0 $\leq \gamma$ ¡ 1)
Initialize an empty list Q_tables to store Q-tables after each episode
for each episode **do**
    Initialize state s
    while state s is not terminal **do**
        Choose action a from state s using policy derived from Q (ϵ-greedy)
        Take action a, observe reward R, and next state s'
        Update Q-table using:
        $Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)]$
        $s \leftarrow s'$
    end while
    Save the current Q-table to Q_tables
end for
Compute the average Q-table from all saved Q-tables in Q_tables

A significant aspect of the training process involves managing and updating the Q-table, which is central to the Q-learning algorithm. After the completion of each training episode, the Q-table, representing the learned values for each action in each state, is stored. This step is crucial as it captures the evolving understanding of the robot about the environment after each episode. Upon the conclusion of all training episodes, a comprehensive approach is adopted to refine the final Q-table. Instead of solely relying on the Q-table from the last episode, we compute the final Q-table as the average of all Q-tables obtained from each episode. This averaging process integrates the insights gained across different episodes, resulting in a more balanced and generalized representation of the learned values. It mitigates the impact of any anomalies or biases that might have occurred in individual episodes, ensuring a more robust and reliable Q-table for guiding the robot's actions in the environment.

This methodical averaging of Q-tables from each episode reinforces the stability and accuracy of the learning outcome, equipping the robot with a well-rounded strategy for navigation in diverse and dynamic settings. Through this methodology, the robot is trained to adapt to changing environments, make decisions based on learned experiences, and ultimately navigate to its goal.

IV. Simulation

The simulation commenced with a comprehensive definition of the environment, which serves as the foundational setting for the Q-learning application. In this phase, key parameters crucial to the Q-learning algorithm were initialized. These include: α: The learning rate, which determines to what extent the newly acquired information overrides the old information. γ: The discount factor used to balance immediate and future rewards. ε: A parameter for the epsilon-greedy strategy, guiding the balance between exploration and exploitation. Following the initialization, the environment was further configured by setting the initial locations of the robot and the goal.

In this dynamic environment, obstacles were placed randomly, introducing variability and complexity to each simulation episode. This setup mimics real-world scenarios where the robot must adapt to changes and uncertainties in its surroundings. The training phase is a critical component of the simulation. Here, the robot interacted with the environment, executing actions based on the Q-learning policy and updating the Q-table after each episode. This iterative process allowed the robot to learn from its experiences, gradually improving its strategy for navigating toward the goal while avoiding obstacles.

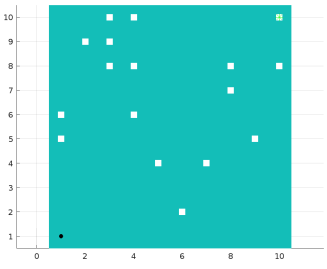

Fig. 1. Visual representation of the robot indicated by black spot in the environment.

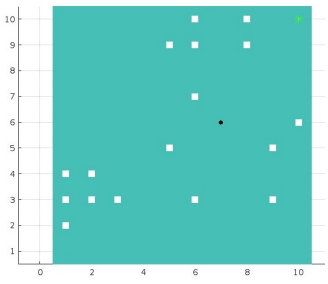

Fig. 2. Location of robot at time T=t

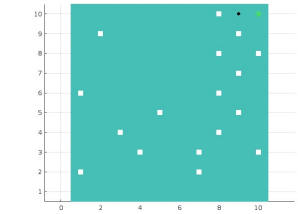

Fig. 3. Position of robot at time T=t+k

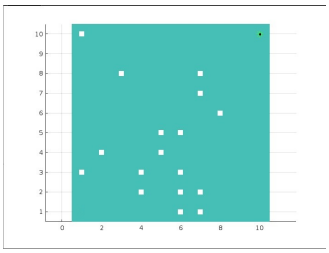

Fig. 4. Visual representation or robot reaching the goal successfully

Post-training, the robot underwent a testing phase to evaluate its learned behaviors. During this phase, no further learning occurred; the robot utilizes the final Q-table obtained from the training phase to make decisions. Figure [1],[2],[3],[4] clearly represent the robot's learned navigation paths and strategies. This simulation, encompassing the initial setup, training, testing, and parameter tuning, provided a comprehensive evaluation of the Q-learning model applied to robotic navigation. Optimization of Q learning parameters is essential for the learning model, ensuring its effectiveness and efficiency in different environmental conditions. The impact of these variations on the robot's performance is discussed in the following section of the paper. This analysis provides valuable insights into the sensitivity of the Q-learning model to its parameters and guides future adjustments for improved performance.

V. Analysis

To ascertain the impact of each parameter, we conducted simulations by varying α: ranging from 0.001 to 1, indicating the degree of learning. γ: spanning from 0.01 to 1, reflecting the value of future rewards. ε: adjusted from 0.1 to 0.9, balancing exploration and exploitation. Maximum Steps per Episode: The number of steps the robot can take in each episode was also varied to understand its influence on the learning process. A pivotal part of this analysis involved calculating the probability of the robot successfully reaching the goal for different maximum step counts. This calculation was performed by varying the goal position throughout the environment, excluding the robot's starting location. The probability of success was then assessed for each distinct maximum step value. The simulation results were compared to the obtained probability, as illustrated in the subsequent figures: At extremely high maximum steps see figure[8], probability of reaching the goal closes to 1 for most of the goal points in the environment.

Algorithm 2 Probability Algorithm

```
Initialize 10x10 grid with start point at (1,1)
for each goal point do
    Set success count to 0
    for each trial do
        Randomly place 20 obstacles on the grid
        Initialize robot position at start point
        Initialize step count to 0
        while step count != 100 do
            Move robot to a valid adjacent cell
            if robot encounters an obstacle then
                Return robot to previous position
            end if
            if robot reaches goal then
                Increment success count
                Break loop
            end if
            Increment step count
        end while
    end for
    Calculate probability for goal as success count / total trials
end for
```

This was substantiated by the simulation outcomes, where altering the max steps value affirmed the robot's consistent success in reaching the destination. For moderate maximum step counts see figure [5],[6] the probability closely aligns with actual simulation results, specifically, when the goal was placed at distant locations, the success rate was naturally lower. This was evidenced in simulations where the robot reached the goal 2-3 times in 10 episodes, with a maximum observed frequency of 5 times in an episode. As the maximum steps per episode increased, the probability distribution across the environment broadened. This indicates a higher likelihood of the robot reaching more diverse goal locations.

An additional layer of our analysis involved visualizing how different combinations of parameters influenced the robot's success in reaching the goal(see figure[9]). This was accomplished by thoroughly examining each parameter's impact while holding the others constant. It was also observed that a small change in the learning parameter did not immensely affect the outcome. To enhance the visualization, the range of variation of the learning parameter was scaled down. An average of the number of times the robot successfully reached the goal was calculated by keeping the other two parameters constant and plotted(see figure[10]). This approach offered a clearer understanding of the parameters collective impact on the robot's navigational proficiency.

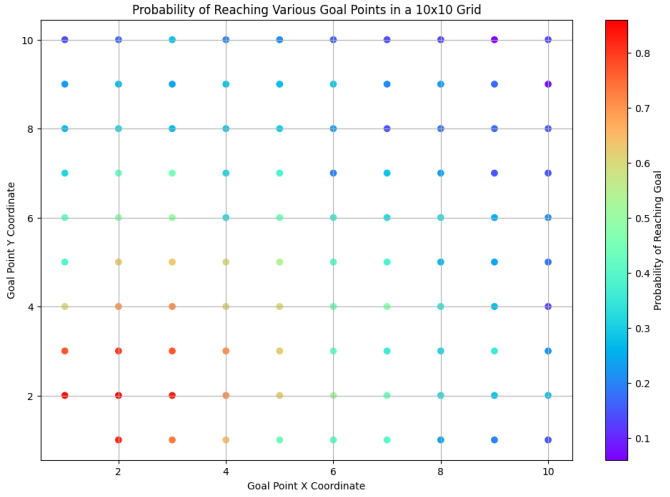

Fig. 5. Probability Distribution for Maximum number of steps = 100

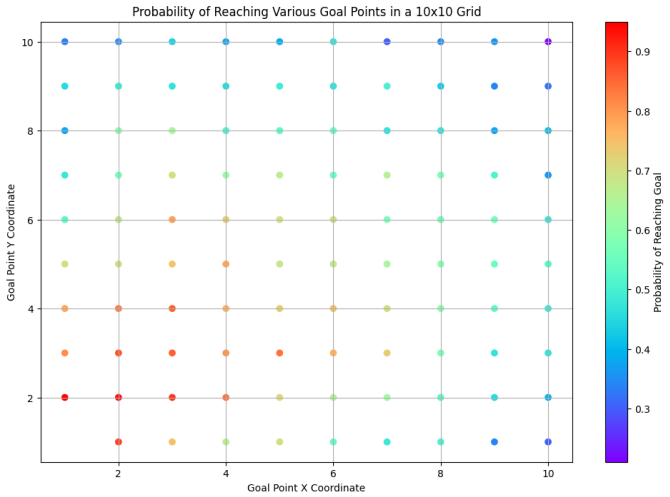

Fig. 6. Probability Distribution for Maximum number of steps = 200

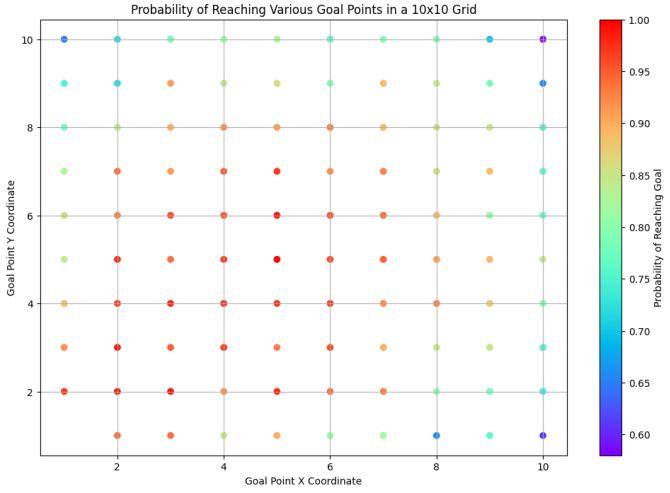

Fig. 7. Probability Distribution for Maximum number of steps = 500

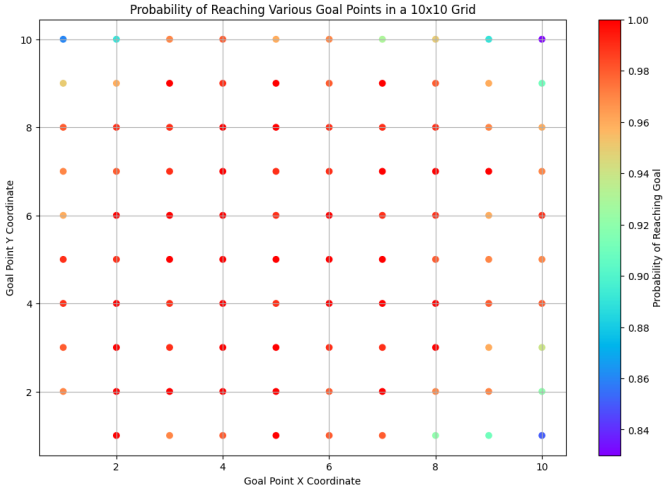

Fig. 8. Probability Distribution for Maximum number of steps = 1000

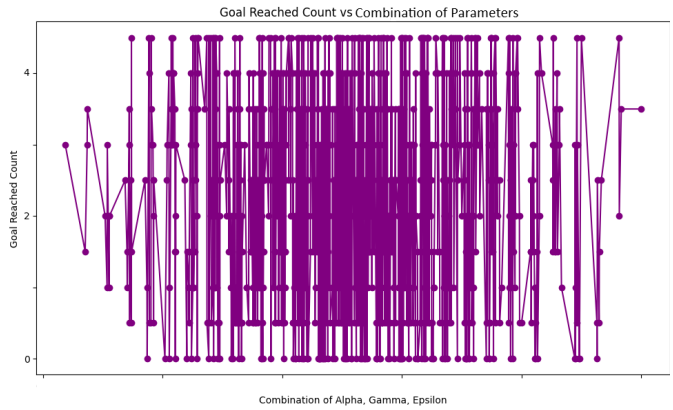

Fig. 9.

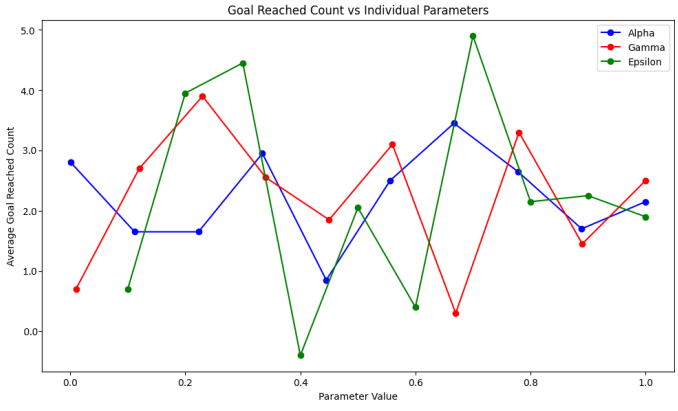

Fig. 10.

VI. Conclusion and Future Work

In this project, we successfully implemented a Q-learning algorithm to enable a robot to navigate an environment with dynamically moving obstacles. The crux of our approach was

to harness the power of reinforcement learning, particularly Q-learning, to teach the robot to avoid static obstacles and anticipate and adapt to their unpredictable movements. Throughout the training phase, the robot was exposed to various scenarios where the obstacles exhibited random movements governed by an underlying random number generator algorithm [5]. This setup was deliberately chosen to mimic real-world conditions where predictability is a luxury and adaptability is key. The primary challenge for the robot was to discern patterns or underlying algorithms in the obstacle movements, a task that stands at the intersection of machine learning and statistical analysis.

This project, while successfully demonstrating the applicability of Q-learning in a simulated robotic navigation environment, opens several avenues for future development. Advancing the model complexity, integrating neural networks, particularly Deep Q-Networks (DQNs), presents a significant enhancement. By applying DQN, the model can potentially handle higher-dimensional state spaces and more complex decision-making scenarios. This approach combines the reinforcement learning framework with the power of deep learning, enabling the system to learn more efficient policies in environments where traditional Q-learning might struggle. A natural progression from simulation is implementing the Q-learning model in robotic systems. This step involves conducting real-world experiments where the theoretical and simulated models are applied to physical robots. Such implementation will validate the model in more complex and less predictable environments and provide insights into practical challenges and adjustments necessary for real-world application.

Another critical area of exploration is comparing the epsilon-greedy strategy used in this project with other exploration strategies like softmax exploration, Boltzmann exploration, and Thompson sampling. Such comparative studies would provide valuable insights into the strengths and limitations of various exploration techniques, guiding the selection of the most appropriate strategy for different environments and tasks. The proposed future work aims to refine and enhance the current model and explore new frontiers in the application of machine learning in robotics by transitioning from simulation to real-world implementation, integrating advanced neural network architectures, experimenting with diverse exploration strategies, and optimizing learning parameters.

REFERENCES

[1] Justin Boyan and Michael Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, 6, 1993.

[2] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[3] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *AAAI*, volume 4, pages 611–616, 2004.

[4] Michael L Littman. Reinforcement learning improves behaviour from evaluative feedback. *Nature*, 521(7553):445–451, 2015.

[5] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[6] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 593–600, 2005.

[7] Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental robotics IX: The 9th international symposium on experimental robotics*, pages 363–372. Springer, 2006.

[8] Jing Peng and Ronald J Williams. Incremental multi-step q-learning. In *Machine Learning Proceedings 1994*, pages 226–232. Elsevier, 1994.

[9] Juan C Santamaria, Richard S Sutton, and Ashwin Ram. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive behavior*, 6(2):163–217, 1997.

[10] Christopher J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.