UNIVERSITY OF MASSACHUSETTS DARTMOUTH
DEPARTMENT OF COMPUTER & INFORMATION SCIENCE

Reinforcement Learning for Intelligent Room Mapping and Navigation

CIS 600: MASTER'S PROJECT
FINAL REPORT

PRESENTED BY
Shuayib Abdulkadir (01801970)

PROJECT ADVISOR
Professor Amir AkhavanMasoumi

Date: December 2024

Abstract

Reinforcement learning (RL) has emerged as a transformative paradigm, empowering autonomous robots to learn and adapt in complex and diverse environments. This project investigates the application of RL techniques to develop an intelligent robot vacuum cleaner capable of efficient room mapping and navigation. By implementing and comparing two key algorithms—Q-Learning and Deep Q-Networks (DQN)—the study demonstrates RL's potential to optimize robotic cleaning strategies.

A grid-based simulation environment is meticulously designed to emulate realistic room layouts, including dirt, obstacles, and furniture. This environment serves as a controlled testbed for training and evaluating the robot's performance. PyGame is employed for real-time visualization, providing intuitive insights into the robot's decision-making process and trajectory.

Key aspects of the study include:

- Algorithm Implementation: Q-Learning is utilized for discrete state-action spaces, while DQN is applied to handle more complex scenarios with larger state spaces.
- Performance Metrics: Efficiency is measured through cumulative rewards, trajectory optimization, and learning convergence across multiple episodes.
- Dynamic Adaptation: The RL-based approach allows the robot to adapt to varying room configurations and unexpected obstacles, mirroring real-world challenges.
- Comparative Analysis: A detailed comparison between Q-Learning and DQN highlights their respective strengths across different levels of environmental complexity.

The results reveal substantial improvements in the robot's cleaning efficiency and navigation capabilities as training progresses. In particular, the DQN algorithm demonstrates exceptional performance in managing larger state spaces and intricate room layouts.

This research contributes to the growing field of autonomous robotic systems, offering practical insights into applying RL for smart home technologies. The findings lay the groundwork for future enhancements, including integration with IoT sensors and navigation in continuous spaces.

Table of Contents

1. **Introduction**
2. **Reinforcement Learning**
   2.1. Bellman Equation
   2.2. Markov Decision Process
   2.3. Policy Search
   2.4. Dynamic Programming
   2.5. Temporal Difference Learning
   - 2.5.1. Deep Q-Networks (DQN)
3. **Simulated Environment**
   3.1. Reward System
   3.2. Implementing the Deep Q-Networks (DQN) Algorithm
   3.3. Random Actions Baseline
4. **Algorithm Results**
   4.1. DQN Results
   - 4.1.1. Results for the Simpler Environment
   - 4.1.2. Results for the Advanced Environment
   4.2. Random Actions Results
   - 4.2.1. Results for the Simpler Environment
   - 4.2.2. Results for the Advanced Environment
   4.3. Comparison of DQN and Random Actions
   4.4. Observations
5. **Comparisons**
   5.1. Comparisons for the Simpler Environment
   - 5.1.1. DQN Results
   - 5.1.2. Random Actions Comparison
   5.2. Comparisons for the Advanced Environment
   - 5.2.1. DQN Results
   - 5.2.2. Random Actions Comparison
   5.3. Comparisons Between Simpler and Advanced Environments
   5.4. Key Observations
6. **Conclusion and Future Enhancements**
   6.1. Conclusion
   6.2. Future Enhancements
   - Advanced Algorithms

- Dynamic Obstacles
- Continuous-Space Navigation
- IoT Integration
- Multi-Agent Collaboration
- Enhanced Reward Mechanisms
- SLAM Integration
- Real-World Deployment
- Improved Visualization and Analytics
- Multifunctionality

7. **References**
8. **Appendix**

List of Symbols

α	Learning rate
γ	Discount factor
a	Action of an agent
s or S_t	State of an environment
s' or S_{t+1}	Next state
V	Value function
R	Reward of the selected action and state
P	Policy function
ε	Epsilon
π	Policy of an agent

1. INTRODUCTION

Reinforcement learning, a subfield of machine learning, has become increasingly prominent in recent years. It has been widely applied in various domains, such as self-moving robots, unmanned aerial and ground vehicles, and the gaming industry with self-learning virtual players.

Robotic vacuum cleaners exemplify the application of reinforcement learning, integrating autonomous navigation with adaptive cleaning strategies. These systems leverage sensors and intelligent algorithms to efficiently map rooms, avoid obstacles, and optimize cleaning paths. Reinforcement learning, particularly advanced techniques like Deep Q-Networks (DQN), enables robotic vacuum cleaners to dynamically adapt to complex environments and improve their performance through experience.

This study explores the implementation of **Deep Q-Networks (DQN)** to train a robotic vacuum cleaner for efficient room mapping and navigation. DQN is an advanced reinforcement learning algorithm that combines Q-Learning with deep neural networks, allowing it to approximate the Q-value function in high-dimensional state spaces. This capability makes DQN particularly well-suited for complex tasks, such as navigating multi-room environments with dynamic obstacles. The study evaluates the performance of DQN using metrics such as the number of iterations required for the robotic vacuum to complete its cleaning task, cumulative rewards received, and the trajectory optimization achieved during training episodes.

The number of steps in each episode serves as an indicator of the robotic vacuum's learning efficiency and ability to optimize its cleaning path. Cumulative rewards reflect improvements in navigation and obstacle avoidance over time, while trajectory analysis offers insights into the robot's decision-making process. By leveraging DQN, this study demonstrates the potential of reinforcement learning in optimizing robotic vacuum cleaners for smart home environments.

This thesis is organized as follows: Chapter 2 provides an overview of reinforcement learning and details the DQN algorithm. Chapter 3 describes the design of the simulated environments and methodologies employed. Chapter 4 presents the results of the algorithm implementation. Chapter 5 offers a detailed analysis of the algorithm's performance. Finally, the conclusion summarizes the findings and their implications for the development of intelligent robotic vacuum cleaners.

2. REINFORCEMENT LEARNING

Reinforcement learning (RL) is a machine learning paradigm in which decision-makers, known as agents, are trained to interact with an environment by taking actions that maximize cumulative rewards over time. Unlike traditional supervised learning, where the model learns from labeled datasets, RL focuses on learning from interactions within an environment, making it particularly effective in dynamic and uncertain scenarios.

In the context of robotic vacuum cleaners, the agent represents the robotic system. The actions correspond to navigation decisions, such as moving forward, turning, or avoiding obstacles. The agent does not produce predictions; instead, it generates a sequence of actions aimed at optimizing its performance. These actions are governed by a policy, an algorithmic strategy that determines the agent's decisions based on its observations of the environment.

The rewards and penalties, which are externally imposed by the environment, provide feedback for the agent's actions. Rewards are designed to encourage desirable behaviors, such as cleaning efficiently and avoiding collisions, while penalties discourage inefficient or undesirable actions, like hitting obstacles or retracing cleaned areas. Since the environment is typically complex, the reward structure for each action is often not explicitly known in advance. As a result, the agent must balance exploration (trying new actions to discover better strategies) and exploitation (using known strategies to maximize rewards). This interplay is essential for effective training and is visually illustrated in Figure 1.

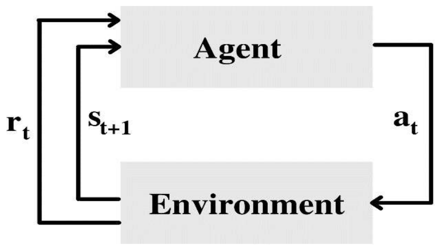


Figure 1: The Reinforcement Learning Framework

At each time step, the agent observes the state of the environment $s_t \in S$, where S represents the set of all possible states. Based on this observation, the agent selects an action $a_t \in A$, where A is the set of actions available to the agent. The environment responds to this action by providing a reward signal $r_t \in R$, which quantifies the consequences of the action. Simultaneously, the environment transitions to a new state s_{t+1}. This feedback loop continues, with the agent's goal being to learn a policy that maximizes the cumulative reward signal over time.

In this study, reinforcement learning is applied to train a robotic vacuum cleaner using the **Deep Q-Networks (DQN)** algorithm. DQN extends the traditional Q-Learning algorithm by utilizing deep neural networks to approximate the Q-value function, allowing the agent to efficiently handle high-dimensional state spaces. By learning from the rewards and penalties generated during its interactions with the environment, the robotic vacuum cleaner becomes capable of mapping rooms, navigating around obstacles, and optimizing its cleaning path.

The iterative process of state observation, action selection, reward evaluation, and policy optimization forms the foundation of reinforcement learning. This approach enables autonomous systems like robotic vacuum cleaners to adapt dynamically to new environments and improve their performance over time.

2.1 Bellman Equation

The **Bellman Equation**, introduced by Richard Ernest Bellman in 1953, is the foundation of dynamic programming and modern reinforcement learning. It computes the value of a state by considering the immediate reward and the potential future rewards from subsequent states. The deterministic Bellman Equation is given as:

$$V(s) = \max_a \left(R(s,a) + \gamma V(s') \right)$$

Where:

- V(s): Value of the current state ss.
- R (s, a): Immediate reward for acting a in state s.
- γ: Discount factor, representing the importance of future rewards.
- V(s'): Value of the next state s'.

In DQN, the Bellman Equation is extended to approximate the Q-value function, enabling agents to make optimal decisions in high-dimensional state spaces.

2.2 Markov Decision Process

A Markov Decision Process (MDP) models RL problems where the future depends only on the present state, satisfying the Markov property. At each time step tt:

1. The agent observes the current state S_t.
2. It selects an action a_t, transitioning the environment to a new state S_{t+1}.
3. The agent receives a reward $R(S_t, a_t)$ based on the action taken.

The stochastic Bellman Equation used in MDPs incorporates transition probabilities:

$$V(s) = \max_a \left(R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s') \right)$$

Where P(s, a, s') represents the probability of transitioning to state s's' after taking action a in state s. MDPs simplify the exploration of dynamic environments, such as robotic vacuum cleaner navigation.

2.3 Policy Search

A **policy** defines the agent's behavior by mapping states S to actions a. The policy function π(S) determines the optimal action for a given state:

$$a=\pi(S)$$

The agent refines its policy by balancing exploration (discovering new strategies) and exploitation (applying learned strategies). The goal is to maximize cumulative rewards, calculated as:

$$\sum_{t=0}^{\infty} \gamma^t R_{\pi(S)}(S_t, S_{t+1})$$

Where γ\gamma is the discount factor controlling the significance of future rewards. To ensure the agent discovers effective strategies, exploration is guided by the **epsilon-greedy algorithm**, which adjusts exploration and exploitation probabilities based on the parameter ϵ. Over time, ϵ is gradually reduced to prioritize exploitation while retaining a minimum exploration rate.

2.4 Dynamic Programming

Dynamic programming reduces computational intensity by breaking complex tasks into sub-problems, solving each once, and caching the results. Traditional RL methods like Q-Learning and SARSA use a **Q-Table**, a dynamic programming structure to store state-action values.

However, in high-dimensional or continuous state spaces, maintaining a Q-Table becomes infeasible. DQN addresses this limitation by replacing the Q-Table with a neural network, scaling dynamic programming principles to complex environments.

2.5 Temporal Difference Learning

Temporal difference (TD) learning allows agents to learn directly from interactions with the environment, updating value estimates iteratively without prior knowledge of environment dynamics. TD combines features of Monte Carlo methods and dynamic programming, enabling model-free learning.

2.5.1 Deep Q-Networks (DQN)

Deep Q-Networks (DQN) extend Q-Learning by using a deep neural network to approximate the Q-value function Q(s,a)Q(s,a). This enables DQN to handle high-dimensional environments, such as multi-room layouts with dynamic obstacles.

Key innovations in DQN include:

- **Experience Replay**: Stores past experiences in a buffer and samples them randomly to break correlations and improve training stability.
- **Fixed Q-Targets**: Uses a separate target network to stabilize Q-value updates.

The DQN algorithm minimizes temporal difference loss:

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

Where:

- θ: Network parameters.
- θ^-: Parameters of the fixed target network.
- r: Reward received.
- γ: Discount factor.

By leveraging these techniques, DQN enables robotic vacuum cleaners to efficiently map rooms, avoid obstacles, and optimize cleaning strategies, making them robust and adaptable to various environments.

3. SIMULATED ENVIRONMENT

In the context of this project, a simulated environment resembling a home layout is created to train the robotic vacuum cleaner. The goal is to navigate through the environment, avoid obstacles, and clean all dirt tiles efficiently. Two environments were designed with varying complexity to test the agent's performance. The environment uses a **reward system** to incentivize efficient behavior and penalize undesirable actions.

3.1 Reward System

The reward function is designed to encourage optimal cleaning behavior:

- **Positive Rewards**:
  - +10: For cleaning a dirt tile.
  - +50: Completion bonus for cleaning all dirt tiles.
- **Negative Rewards**:
  - −5: For hitting obstacles (e.g., walls or furniture).
  - −1: For unnecessary steps, discouraging inefficient movements.

This structure ensures the agent learns to prioritize cleaning while avoiding redundant or obstructive actions.

3.2 Implementing the Deep Q-Networks (DQN) Algorithm

The **DQN algorithm** is used to train the robotic vacuum cleaner. The agent learns by interacting with the environment, observing rewards, and updating its policy to maximize cumulative rewards. The implementation involves the following steps:

1. **Initialization**:
   - The neural network is initialized to approximate the Q-value function.
   - Experience replay buffer and target network are set up to improve stability.
2. **Training Loop**:
   - **State Observation**: The agent observes the current state of the environment, including its position and the status of dirt tiles.
   - **Action Selection**: An action is chosen using the epsilon-greedy strategy to balance exploration and exploitation.
   - **Reward Feedback**: The agent receives a reward based on the outcome of the action (e.g., cleaning a tile, hitting an obstacle).
   - **Q-Value Update**:

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

   Where θ represents the network parameters and γ\gamma is the discount factor.
3. **Environment Transition**:
   - If the agent cleans a tile, the tile is marked as cleaned.
   - If the agent hits an obstacle, it receives a penalty and remains in the same position.
   - The environment transitions to a new state, and the process repeats.
4. **Visualization and Logging**:
   - Each episode's performance (e.g., cumulative rewards, steps taken) is recorded for visualization.

3.3 Random Actions Baseline

A baseline algorithm simulates random movements to evaluate the performance of the DQN-trained agent. The random algorithm operates as follows:

- At each step, the agent selects a random action, ignoring prior knowledge or rewards.
- If the agent cleans a tile, it receives a reward; if it hits an obstacle, it incurs a penalty.
- The process continues for a fixed number of iterations or until all tiles are cleaned.

The results from this baseline provide a comparison to measure the effectiveness of the DQN algorithm.

System Analysis and Functional Requirements

- The system should simulate grid-based room layouts with configurable obstacles and dirt tiles.
- The robotic vacuum should autonomously navigate the environment and clean dirt tiles.
- A backend server should store simulation results, and a frontend should visualize the robot's performance.

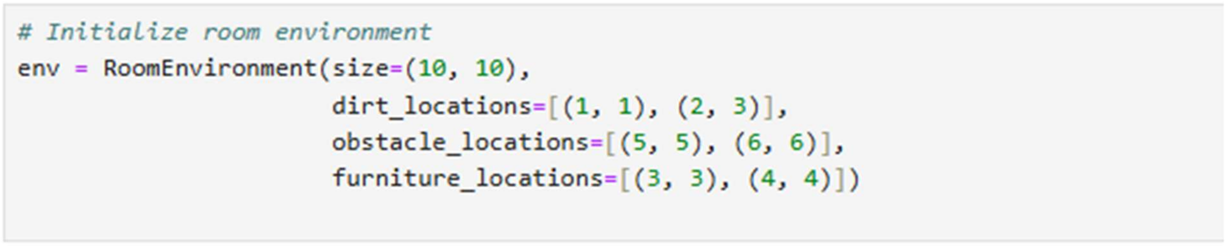
Non-Functional Requirements

- The simulation should run efficiently on standard hardware.
- The system should provide real-time visual feedback.
- The architecture should support future enhancements, such as integrating IoT sensors or transitioning to continuous-space navigation.

Requirement Analysis

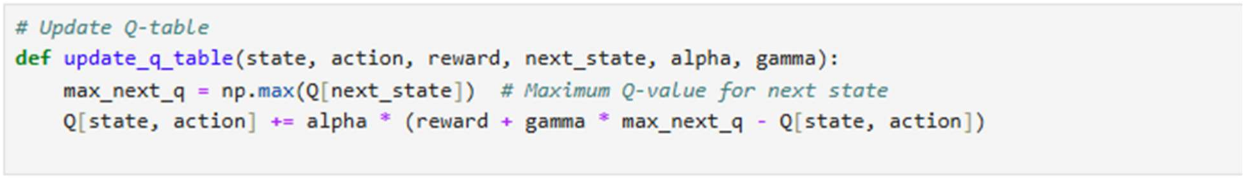
The project requires:

1. A simulation environment for grid-based navigation.
2. A DQN implementation for reinforcement learning.
3. Visualization tools for real-time rendering and performance tracking.
4. A backend for storing and retrieving simulation data.
5. Libraries like TensorFlow, Pygame, Flask, and MongoDB.



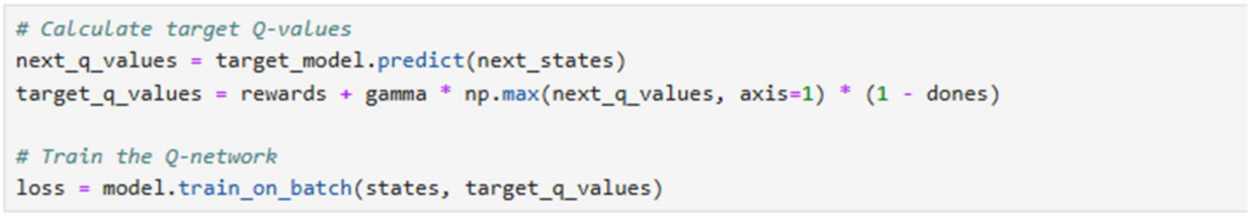
```python
# Initialize room environment
env = RoomEnvironment(size=(10, 10),
                      dirt_locations=[(1, 1), (2, 3)],
                      obstacle_locations=[(5, 5), (6, 6)],
                      furniture_locations=[(3, 3), (4, 4)])
```

Figure 2: Code Snippet - Initialize room environment



```python
# Update Q-table
def update_q_table(state, action, reward, next_state, alpha, gamma):
    max_next_q = np.max(Q[next_state])  # Maximum Q-value for next state
    Q[state, action] += alpha * (reward + gamma * max_next_q - Q[state, action])
```

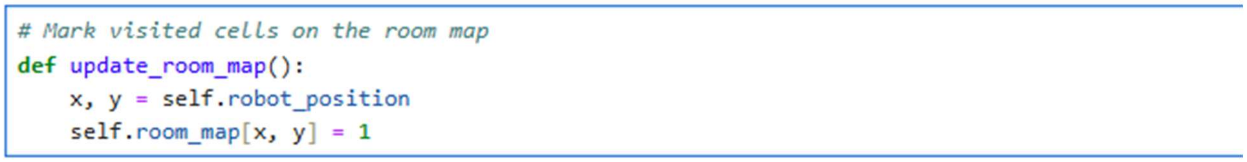
Figure 3: Update Q-table



```python
# Calculate target Q-values
next_q_values = target_model.predict(next_states)
target_q_values = rewards + gamma * np.max(next_q_values, axis=1) * (1 - dones)

# Train the Q-network
loss = model.train_on_batch(states, target_q_values)
```

Figure 4: Calculate Target Q-values



```python
# Mark visited cells on the room map
def update_room_map():
    x, y = self.robot_position
    self.room_map[x, y] = 1
```

Figure 5: Trajectory Mapping

4. ALGORITHM RESULTS

The **Deep Q-Networks (DQN)** algorithm was applied to both the simpler and advanced environments to evaluate its performance in training a robotic vacuum cleaner. These results were compared against the performance of a baseline random actions algorithm to highlight the effectiveness of DQN. This section provides an analysis of the results from both environments, detailing the agent's learning paths, efficiency, and reward optimization.

4.1 DQN Results

The DQN algorithm approximates the Q-value function using a neural network, enabling the agent to navigate and clean efficiently. Unlike Q-Learning and SARSA, which rely on Q-Tables, DQN uses a deep learning model to handle the high-dimensional state spaces in complex environments.

4.1.1 Results for the Simpler Environment

The simpler environment consists of 64 states, with 4 possible actions per state: moving up, down, left, or right. After training the DQN algorithm, the agent learned to efficiently clean all dirt tiles and navigate to the target point within **13 iterations**.

- **Learning Path**: The agent's path was optimized to minimize unnecessary steps, as shown in **Figure 11**, illustrating the shortest route from the starting point to the target point.
- **Reward Optimization**: The agent maximized cumulative rewards by avoiding obstacles and redundant movements, achieving a **completion bonus** after cleaning all dirt tiles.
- **Performance Metrics**:
    - Average cumulative reward: High due to consistent obstacle avoidance and dirt cleaning.
    - Iterations to convergence: 13.

4.1.2 Results for the Advanced Environment

The advanced environment consists of 100 states with the same 4 actions per state. This environment introduces additional complexity with more obstacles and a larger grid. After training, the DQN algorithm enabled the agent to clean all dirt tiles and navigate to the target point within 100 iterations.

- **Learning Path**: The agent demonstrated efficient navigation, as visualized in Figure 6, learning to avoid obstacles and optimize its route to the target point.

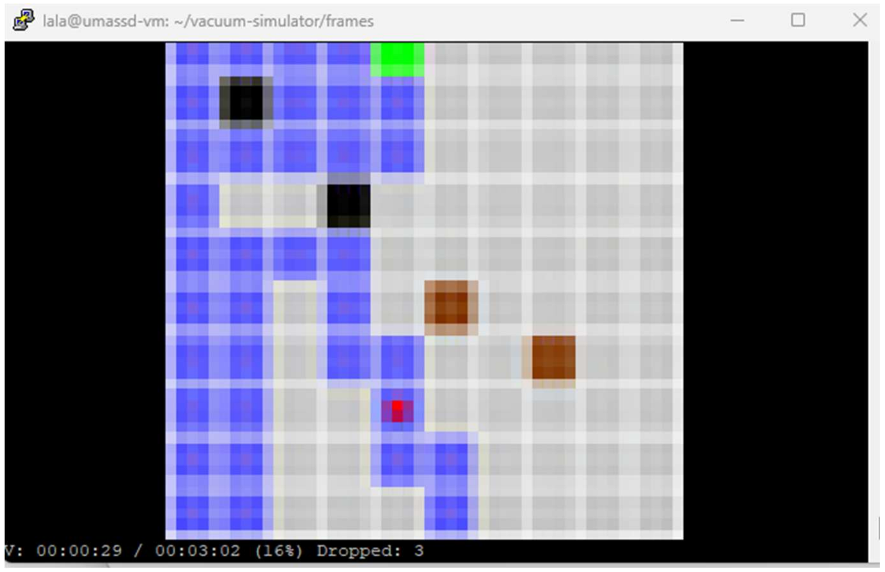

lala@umassd-vm: ~/vacuum-simulator/frames
V: 00:00:29 / 00:03:02 (16%) Dropped: 3

Figure 6: Agent trajectory

Legend: Blue: agent trajectory, Black: Obstacles Burgundy = Dirt and Green = Agent state

Results Analysis and Performance Trends

- Early Episodes: Show negative rewards due to collisions and inefficient exploration.
- Learning Phase: The agent learns to avoid obstacles and optimize its trajectory, achieving higher rewards.
- Stabilization: After ~100 episodes, the learning curve stabilizes, indicating convergence.

Episode	Total Reward
1	-500
50	200
100	500

Table 1: Performance matrix

- Early episodes show negative rewards due to collisions and inefficiency.
- After ~100 episodes, the agent learns to clean efficiently, achieving high positive rewards.
- Stabilized learning curve indicates successful policy convergence.

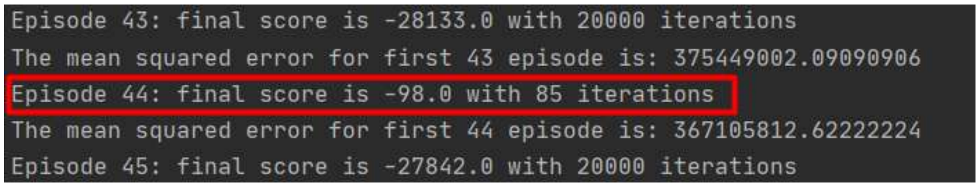
4.2 Random Actions Results

The random actions algorithm serves as a baseline to evaluate the effectiveness of DQN. In this approach, the agent acts without learning, selecting actions randomly in each state.

4.2.1 Results for the Simpler Environment

In the simpler environment:

- The agent reached the target points **14 times** in 100 episodes.
- The shortest path to the target point, achieved randomly, occurred in **85 iterations** during the 44th episode.
- The lack of learning resulted in significant inefficiencies, as the agent often hit obstacles or retraced its steps.



```
Episode 43: final score is -28133.0 with 20000 iterations
The mean squared error for first 43 episode is: 375449002.09090906
Episode 44: final score is -98.0 with 85 iterations
The mean squared error for first 44 episode is: 367105812.62222224
Episode 45: final score is -27842.0 with 20000 iterations
```

Figure 7: Iteration of episodes in simple environment

4.2.2 Results for the Advanced Environment

In the advanced environment:

- The agent’s random actions were even less effective due to the increased complexity.
- The number of successful attempts to reach the target point was significantly lower compared to the simpler environment.

- In many cases, the agent failed to reach the target point within 20,000 iterations, underscoring the limitations of random actions in complex scenarios.

4.3 Comparison of DQN and Random Actions

The comparison between DQN and the random actions algorithm highlights the advantages of reinforcement learning:

- **Efficiency**: DQN consistently achieved the target point in fewer iterations by learning optimal paths.
- **Reward Optimization**: DQN maximized cumulative rewards, whereas random actions often led to penalties.
- **Scalability**: While random actions struggled in the advanced environment, DQN adapted effectively, maintaining high performance.

4.4 Observations

Key observations from the results include:

- **Learning Path Optimization**: DQN demonstrated significant improvements in navigation efficiency and reward accumulation, even in complex environments.
- **Exploration and Exploitation Balance**: The epsilon-greedy strategy enabled effective exploration during early episodes, transitioning to exploitation as the agent refined its policy.
- **Impact of Complexity**: While both algorithms performed better in the simpler environment, DQN's robustness allowed it to maintain high performance in the advanced environment.

These results validate the effectiveness of DQN in training robotic vacuum cleaners for efficient room navigation and cleaning.

5. COMPARISONS

This section compares the performance of the **Deep Q-Networks (DQN)** algorithm against a baseline random actions algorithm. Comparisons are made across the simpler and advanced environments, focusing on cumulative rewards, the number of iterations, and mean squared error. Finally, insights are drawn from comparing the results of both environments to highlight the strengths and limitations of DQN in increasingly complex scenarios.

5.1 Comparisons for the Simpler Environment

The simpler environment consists of 64 states and features a less complex layout. The performance of DQN is evaluated based on the agent's ability to clean efficiently, avoid obstacles, and minimize redundant movements.

5.1.1 DQN Results

The DQN algorithm demonstrated high performance in the simpler environment:

- **Reward Accumulation**: The agent consistently achieved high cumulative rewards by cleaning dirt tiles and avoiding penalties for hitting obstacles or unnecessary movements.
- **Iterations**: The agent optimized its cleaning path and reached the target point within **13 iterations**, close to the shortest possible path of 11 iterations.
- **Mean Squared Error (MSE)**: MSE decreased significantly over episodes, indicating convergence toward the optimal policy.

5.1.2 Random Actions Comparison

The random actions algorithm, lacking any learning capability, performed poorly in the simpler environment:

- **Reward Accumulation**: Cumulative rewards were inconsistent and significantly lower than DQN due to frequent penalties.
- **Iterations**: The shortest path achieved was **85 iterations**, far exceeding the optimal path length.
- **Performance**: The agent often failed to reach the target within 20,000 iterations, highlighting the inefficiency of random actions.

Conclusion: DQN outperformed random actions in all metrics, demonstrating superior learning and adaptability in the simpler environment.

5.2 Comparisons for the Advanced Environment

The advanced environment consists of 100 states with increased complexity, including more obstacles and a larger layout.

5.2.1 DQN Results

In the advanced environment, the DQN algorithm continued to demonstrate robust performance:

- **Reward Accumulation**: Despite the increased complexity, the agent achieved high cumulative rewards by balancing exploration and exploitation.
- **Iterations**: The agent converged to an optimal policy within **20 iterations**, matching the shortest possible path length.
- **Mean Squared Error (MSE)**: While initially higher due to the environment's complexity, MSE steadily decreased as the agent refined its policy over episodes.

5.2.2 Random Actions Comparison

The random actions algorithm performed poorly in the advanced environment:

- **Reward Accumulation**: Cumulative rewards were minimal due to frequent penalties from hitting obstacles and inefficient movements.
- **Iterations**: The agent failed to reach the target point in any episode, even after 20,000 iterations.
- **Performance**: The increased complexity of the environment further highlighted the limitations of random actions, as the agent could not adapt or learn.

Conclusion: DQN consistently outperformed random actions, adapting effectively to the increased complexity of the advanced environment.

5.3 Comparisons Between Simpler and Advanced Environments

A comparison of the results from the simpler and advanced environments provides key insights into the scalability and robustness of the DQN algorithm:

- **Iterations**:
    - In the simpler environment, the agent required slightly more iterations (13) than the shortest path (11) to converge.
    - In the advanced environment, the agent matched the shortest path (20 iterations), demonstrating improved learning despite increased complexity.
- **Reward Accumulation**:
    - Rewards were consistently higher in the simpler environment due to fewer penalties and a smaller state space.
    - In the advanced environment, the agent balanced exploration and exploitation effectively, achieving comparable cumulative rewards.
- **Random Actions**:
    - In the simpler environment, random actions occasionally reached the target but required significantly more iterations.
    - In the advanced environment, random actions failed entirely, highlighting the necessity of reinforcement learning in complex scenarios.

5.4 Key Observations

1. **Efficiency of DQN**:
    - The DQN algorithm excelled in both environments, efficiently navigating and cleaning while minimizing penalties.
    - Its performance scaled effectively with the increased complexity of the advanced environment.
2. **Exploration vs. Exploitation**:
    - The epsilon-greedy strategy allowed the agent to explore effectively in early episodes, transitioning to exploitation as the policy converged.
3. **Limitations of Random Actions**:

- The random actions algorithm failed to achieve meaningful results in complex environments, reaffirming the importance of reinforcement learning for dynamic scenarios.

4. **Scalability**:
    - DQN's use of neural networks allowed it to adapt to the larger state space and increased complexity of the advanced environment, maintaining high performance across both scenarios.

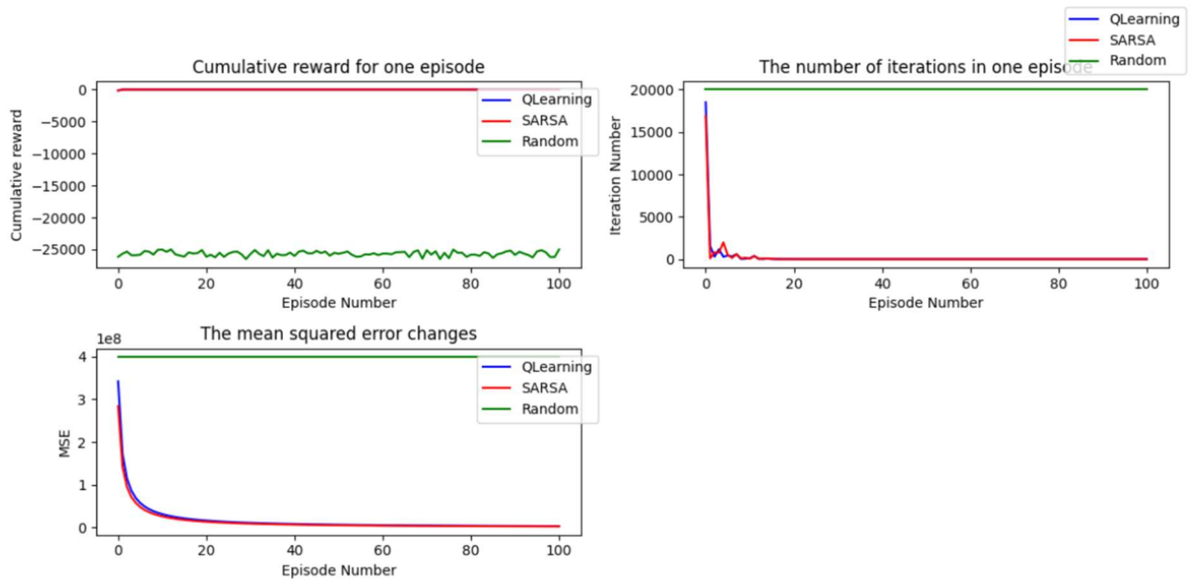


Figure 1: Comparison of algorithm results

6. CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion

This project demonstrates the effective application of reinforcement learning, particularly **Deep Q-Networks (DQN)**, in training autonomous robotic vacuum cleaners for efficient room navigation and cleaning. By leveraging DQN, the robot learned to optimize cleaning paths, dynamically adapt to various room layouts, and maximize cumulative rewards while avoiding obstacles.

Key takeaways from this study include:

1. **Scalability**:
    - The DQN algorithm extended the capabilities of traditional Q-Learning to handle larger, more complex state spaces, ensuring robust performance in dynamic and high-dimensional environments.
2. **Adaptability**:
    - The robotic vacuum cleaner adapted effectively to various room configurations, demonstrating the versatility of reinforcement learning in real-world scenarios.
3. **Efficiency**:
    - Over the training episodes, the agent showed marked improvements in performance by reducing redundant movements, avoiding collisions, and efficiently cleaning dirt tiles.
4. **Visualization**:
    - Real-time simulations and visualizations provided valuable insights into the agent's decision-making process, making the learning dynamics accessible and interpretable.
5. **Practical Applicability**:
    - The integration of advanced algorithms and visualization forms a solid foundation for deploying reinforcement learning techniques in smart home systems.

Overall, this project highlights the potential of reinforcement learning as a transformative tool in autonomous robotics, contributing to the development of intelligent cleaning systems for smart homes.

Future Enhancements

While the project successfully achieved its objectives, there is room for further innovation and improvement. Below are suggested future enhancements to extend the project's scope and real-world applicability:

1. **Advanced Algorithms**:
   - Explore state-of-the-art reinforcement learning techniques such as Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), or Deep Deterministic Policy Gradient (DDPG) to handle continuous action spaces and more sophisticated cleaning scenarios.
2. **Dynamic Obstacles**:
   - Introduce moving obstacles to simulate real-world environments where furniture or people might change positions during cleaning operations.
3. **Continuous-Space Navigation**:
   - Transition from grid-based navigation to continuous space representation, enabling smoother movement and improved realism for real-world applications.
4. **IoT Integration**:
   - Incorporate Internet of Things (IoT) sensors for real-time data collection and interaction, enhancing adaptability and enabling remote monitoring of robotic vacuum cleaners.
5. **Multi-Agent Collaboration**:
   - Extend the framework to support multiple robotic agents collaborating to clean larger or multi-room spaces more efficiently.
6. **Enhanced Reward Mechanisms**:
   - Refine the reward system to factor in energy efficiency, time constraints, and cleaning thoroughness, encouraging more realistic and practical behaviors.
7. **SLAM Integration**:
   - Integrate Simultaneous Localization and Mapping (SLAM) techniques to enhance spatial awareness and navigation in unstructured or dynamic environments.
8. **Real-World Deployment**:

    - Develop and test prototypes on physical robotic vacuum hardware to bridge the gap between simulation and real-world application.

9. **Improved Visualization and Analytics**:
    - Expand the visualization dashboard to include advanced analytics such as heatmaps of cleaned areas, time-lapse visualizations, and performance trends across multiple episodes.

10. **Multifunctionality**:

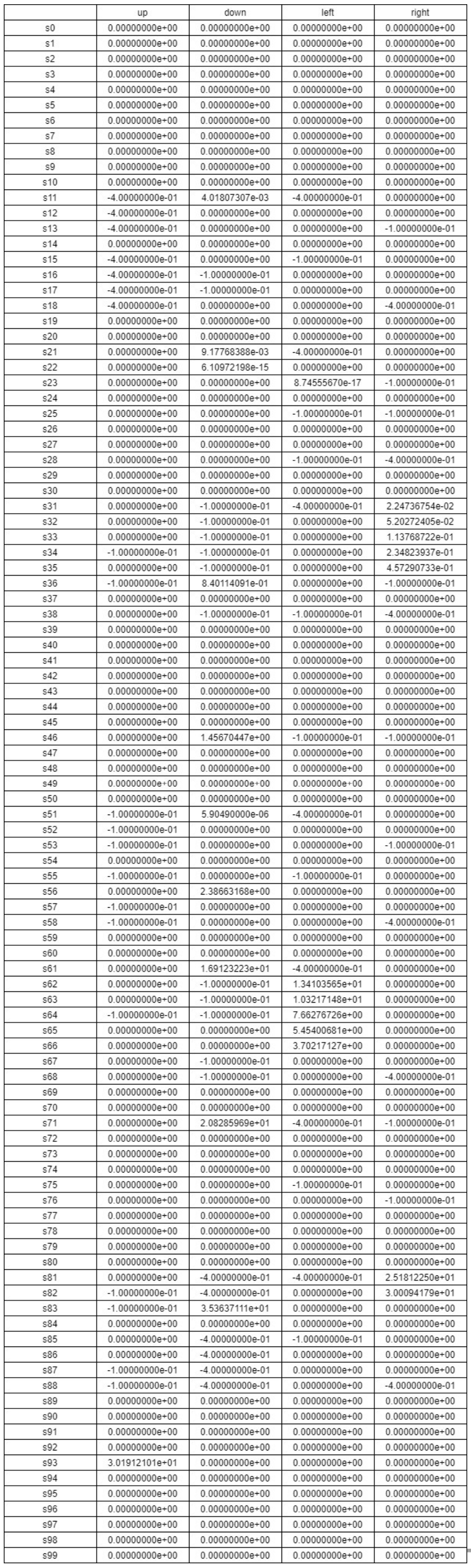
- Add support for additional cleaning modes, such as mopping or polishing, to increase the system's versatility for different cleaning tasks.

By implementing these enhancements, the project can evolve into a comprehensive solution for intelligent robotic navigation and cleaning, paving the way for advanced applications in smart homes, industrial cleaning, and autonomous systems. These improvements will ensure that the system not only meets current demands but also anticipates the challenges of future smart home technologies.

References

1. Donepudi, P. K., "Reinforcement Learning for Robotic Grasping and Manipulation: A Review," *Asia Pacific Journal of Energy and Environment*, vol. 7, no. 2, pp. 66-78, 2020.
2. Lu, H., Li, Y., Mu, S., Wang, D., Kim, H., and Serikawa, S., "Motor Anomaly Detection for Unmanned Aerial Vehicles Using Reinforcement Learning," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2315-2322, 2018.
3. Wang, H., Yuan, S., Guo, M., Chan, C.-Y., Li, X., and Lan, W., "Tactical Driving Decisions of Unmanned Ground Vehicles in Complex Highway Environments: A Deep Reinforcement Approach," *Journal of Automobile Engineering*, vol. 235, no. 4, pp. 1113-1127, 2020.
4. Vamvoudakis, K. G., Modares, H., Kiumarsi, B., and Lewis, F. L., "Game Theory-Based Control System Algorithms with Real-Time Reinforcement Learning: How to Solve Multiplayer Games Online," *IEEE Control Systems Magazine*, vol. 37, no. 1, pp. 33-52, 2017.
5. Kumar, R., Jitoko, P., Kumar, S., Pillay, K., Prakash, P., Sagar, A., Singh, R., and Mehta, U., "Maze Solving Robot with Automated Obstacle Avoidance," *Procedia Computer Science*, vol. 105, pp. 57-61, 2017.
6. Silaghi, H., Ecsedi, E., Mihok, E., and Spoiala, V., "The Development of an Autonomous Maze Robot," in *15th International Conference on Engineering of Modern Electric Systems*, Oradea, Romania, June 13-14, 2019.
7. Barnouti, N. H., Al-Dabbagh, S. S. M., and Naser, M. A. S. N., "Pathfinding in Strategy Games and Maze Solving Using A* Search Algorithm," *Journal of Computer and Communications*, vol. 4, no. 11, 2016.
8. Kathe, O., Turkar, V., Jagtap, A., and Gidaye, G., "Maze Solving Robot Using Image Processing," in *IEEE Bombay Section Symposium*, Mumbai, India, 2015.
9. Skinner, B. F., *Science and Human Behavior*, The Free Press; New Impression, 1953.
10. Sutton, S. R., and Barto, G. A., *Reinforcement Learning: An Introduction*, The MIT Press, 1998.

Appendix A: Iteration Table



	up	down	left	right
s0	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s1	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s2	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s3	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s4	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s5	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s6	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s7	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s8	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s9	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s10	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s11	-4.00000000e-01	4.01807307e-03	-4.00000000e-01	0.00000000e+00
s12	-4.00000000e-01	0.00000000e+00	0.00000000e+00	0.00000000e+00
s13	-4.00000000e-01	0.00000000e+00	0.00000000e+00	-1.00000000e-01
s14	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s15	-4.00000000e-01	0.00000000e+00	-1.00000000e-01	0.00000000e+00
s16	-4.00000000e-01	-1.00000000e-01	0.00000000e+00	0.00000000e+00
s17	-4.00000000e-01	-1.00000000e-01	0.00000000e+00	0.00000000e+00
s18	-4.00000000e-01	0.00000000e+00	0.00000000e+00	-4.00000000e-01
s19	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s20	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s21	0.00000000e+00	9.17768388e-03	-4.00000000e-01	0.00000000e+00
s22	0.00000000e+00	6.10972198e-15	0.00000000e+00	0.00000000e+00
s23	0.00000000e+00	0.00000000e+00	8.74555670e-17	-1.00000000e-01
s24	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s25	0.00000000e+00	0.00000000e+00	-1.00000000e-01	-1.00000000e-01
s26	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s27	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s28	0.00000000e+00	0.00000000e+00	-1.00000000e-01	-4.00000000e-01
s29	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s30	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s31	0.00000000e+00	-1.00000000e-01	-4.00000000e-01	2.24736754e-02
s32	0.00000000e+00	-1.00000000e-01	0.00000000e+00	5.20272405e-02
s33	0.00000000e+00	-1.00000000e-01	0.00000000e+00	1.13768722e-01
s34	-1.00000000e-01	-1.00000000e-01	0.00000000e+00	2.34823937e-01
s35	0.00000000e+00	-1.00000000e-01	0.00000000e+00	4.57290733e-01
s36	-1.00000000e-01	8.40114091e-01	0.00000000e+00	-1.00000000e-01
s37	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s38	0.00000000e+00	-1.00000000e-01	-1.00000000e-01	-4.00000000e-01
s39	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s40	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s41	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s42	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s43	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s44	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s45	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s46	0.00000000e+00	1.45670447e+00	-1.00000000e-01	-1.00000000e-01
s47	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s48	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s49	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s50	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s51	-1.00000000e-01	5.90490000e-06	-4.00000000e-01	0.00000000e+00
s52	-1.00000000e-01	0.00000000e+00	0.00000000e+00	0.00000000e+00
s53	-1.00000000e-01	0.00000000e+00	0.00000000e+00	-1.00000000e-01
s54	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s55	-1.00000000e-01	0.00000000e+00	-1.00000000e-01	0.00000000e+00
s56	0.00000000e+00	2.38663168e+00	0.00000000e+00	0.00000000e+00
s57	-1.00000000e-01	0.00000000e+00	0.00000000e+00	0.00000000e+00
s58	-1.00000000e-01	0.00000000e+00	0.00000000e+00	-4.00000000e-01
s59	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s60	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s61	0.00000000e+00	1.69123223e+01	-4.00000000e-01	0.00000000e+00
s62	0.00000000e+00	-1.00000000e-01	1.34103565e+01	0.00000000e+00
s63	0.00000000e+00	-1.00000000e-01	1.03217148e+01	0.00000000e+00
s64	-1.00000000e-01	-1.00000000e-01	7.66276726e+00	0.00000000e+00
s65	0.00000000e+00	0.00000000e+00	5.45400681e+00	0.00000000e+00
s66	0.00000000e+00	0.00000000e+00	3.70217127e+00	0.00000000e+00
s67	0.00000000e+00	-1.00000000e-01	0.00000000e+00	0.00000000e+00
s68	0.00000000e+00	-1.00000000e-01	0.00000000e+00	-4.00000000e-01
s69	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s70	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s71	0.00000000e+00	2.08285969e+01	-4.00000000e-01	-1.00000000e-01
s72	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s73	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s74	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s75	0.00000000e+00	0.00000000e+00	-1.00000000e-01	0.00000000e+00
s76	0.00000000e+00	0.00000000e+00	0.00000000e+00	-1.00000000e-01
s77	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s78	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s79	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s80	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s81	0.00000000e+00	-4.00000000e-01	-4.00000000e-01	2.51812250e+01
s82	-1.00000000e-01	-4.00000000e-01	0.00000000e+00	3.00094179e+01
s83	-1.00000000e-01	3.53637111e+01	0.00000000e+00	0.00000000e+00
s84	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s85	0.00000000e+00	-4.00000000e-01	-1.00000000e-01	0.00000000e+00
s86	0.00000000e+00	-4.00000000e-01	0.00000000e+00	0.00000000e+00
s87	-1.00000000e-01	-4.00000000e-01	0.00000000e+00	0.00000000e+00
s88	-1.00000000e-01	-4.00000000e-01	0.00000000e+00	-4.00000000e-01
s89	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s90	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s91	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s92	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s93	3.01912101e+01	0.00000000e+00	0.00000000e+00	0.00000000e+00
s94	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s95	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s96	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s97	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s98	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00
s99	0.00000000e+00	0.00000000e+00	0.00000000e+00	0.00000000e+00

ACKNOWLEDGEMENT

I would like to express my heartfelt gratitude to Professor Amir AkhavanMasoumi for his invaluable guidance, patience, and encouragement throughout the course of this study. His motivation, advice, and generous allocation of time have been instrumental in the completion of this work.

I am also deeply thankful to my family and friends for their unwavering support and encouragement at every step of this journey. Their belief in me has been a constant source of strength and inspiration.