CS F317 Reinforcement Learning Assignment Report

Abstract

This report studies Monte Carlo-based Reinforcement Learning (RL) methods for robot navigation in a grid-based environment, focusing on the performance of On-Policy and Off-Policy Monte Carlo algorithms. The robot navigates through a complex grid world containing obstacles, target locations, dynamic hazards, etc. employing stochastic transitions to simulate realistic movement.

On-Policy Monte Carlo optimises the policy based on the agent's experience, using an epsilon-greedy approach to balance exploration and exploitation. Off-Policy Monte Carlo, on the other hand, employs importance sampling to learn a target policy while following a more exploratory behaviour policy.

We explored various evaluation metrics such as obtained average rewards, convergence rate, and Q-value estimates to analyse differences between On-Policy & Off-Policy methods. The code developed for formulating the problem, implementing RL-based on-policy and off-policy Monte Carlo methods along with obtained results mentioned in the report can be found at bPratyush/CS-F317-Reinforcement-Learning-Project (github.com).

Problem Formulation: RL-Based Robot Vaccum Cleaning

We focused on developing a reinforcement learning (RL) agent capable of operating within a dynamic and complex grid-based environment. The agent's objective is to clean the room which will lead to maximisation of the cumulative rewards while navigating through a grid, interacting with various static and dynamic elements, recharging stations, time tunnelling and stochastic transitions, all while the agent operates under constraints like limited battery life.

Environment Setup

The environment is represented as a two-dimensional grid of fixed size, with each cell representing a state that the agent can interact with. There could be single or multiple agents which are essentially Cleaners.

The Cleaner can travel in 8 distinct ways at each time step: up, down, left, right, up-left, up-right, down-left, and down-right, with also having the option to remain in its current cell. Each action has multiple outcomes. Also, the transitions are probabilistic and it is possible that the agent may not always land in the expected adjacent cell but may deviate to neighbouring states based on probabilistic transitions. The chance of occurrence of primary transition on choosing a certain action is 80% and the chance of deviation, i.e., secondary transition is 10%. The associated table for transitions is given below.

Action	Primary Transition (0.8)	Secondary Transition (0.1)
Up	Up	Up Right, Up Left
Down	Down	Down Right, Down Left
Left	Left	Up Left, Down Left
Right	Right	Up Right, Down Right

The chance of Staying in the current cell is 100% with no possibility of deviation. Furthermore, different types of elements populate the grid, each introducing unique interactions and consequences for the Cleaner:

1. **Clean Spaces**: Basic navigable spaces with no specific reward or penalty.
2. **Static Obstacles**: These are immovable barriers that block the agent's movement such as walls and other objects that could be present in a room. Colliding with an obstacle results in penalties and potential episode termination. (Penalty = -10)
3. **Dynamic Obstacles**: Unlike static obstacles, which remain in fixed locations throughout the episode, dynamic obstacles change their position, which influences Cleaner's path and decision-making. Colliding with them also results in penalties and potential episode termination. (Penalty = -20)
4. **Reward Spots**: These are the dirty cells that provide positive rewards when the agent cleans them. (Reward = +10)
5. **Time Tunnels**: Multiple time tunnels at various locations in which when the agent enters, it is teleported to a different corresponding tunnel exit located elsewhere.
6. **Charging Stations**: Locations where the agent can recharge its battery, extending its operational time. If the agent reaches a charging station before its battery falls below a minimum threshold (Battery Level = 0), it is recharged and awarded a positive reward (Reward = +5) for timely arrival. Otherwise, a penalty is incurred (Penalty = -10) if the battery level falls below a minimum threshold before reaching the station.
7. **Hazardous Areas**: These are the danger zones where entering will result in severe penalties or end the episode for the Cleaner. (Penalty = -30)

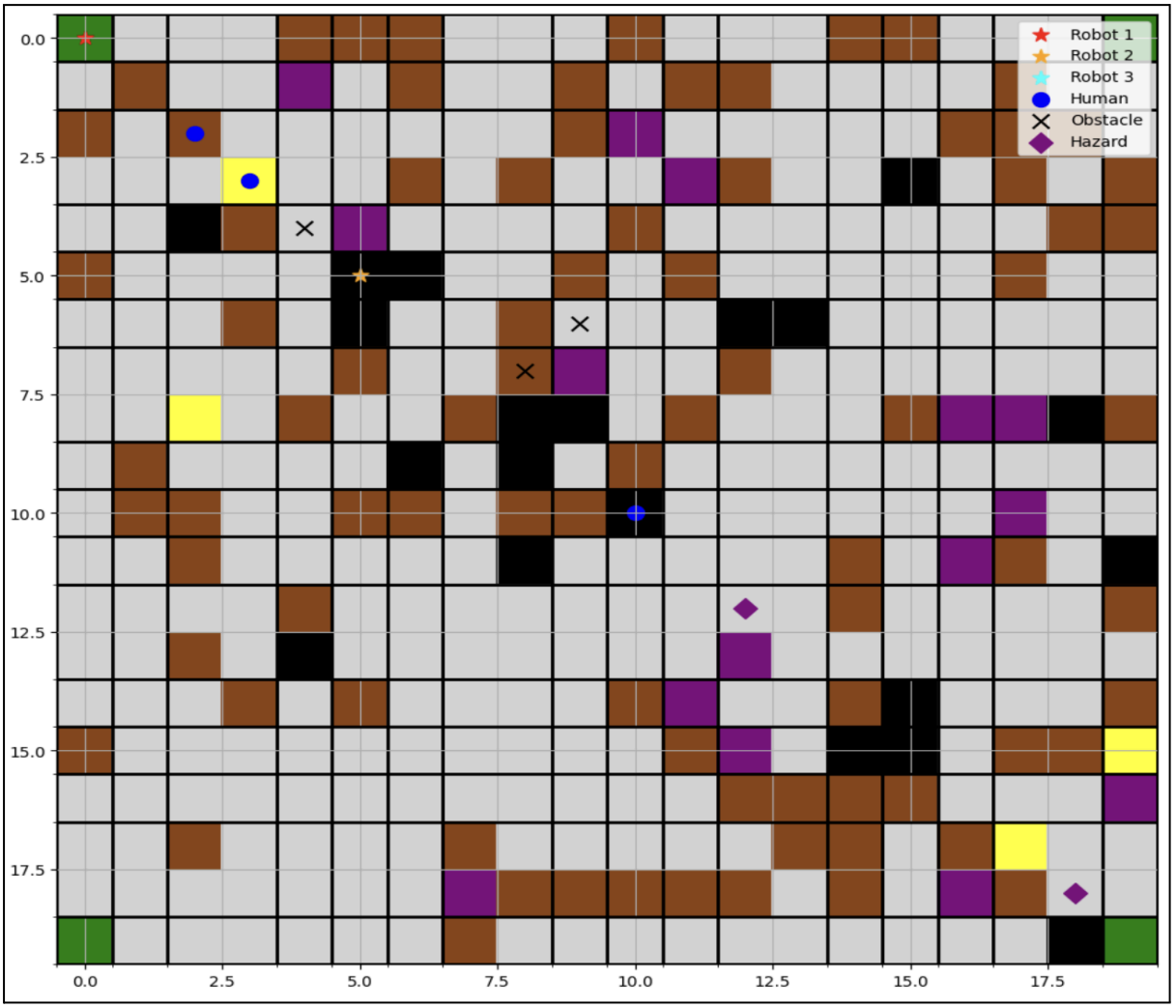
Agent's Objectives and Constraints

The RL agent must navigate through the environment to achieve the following:

1. **Maximize cumulative reward**: The Cleaner should maximise its rewards by cleaning dirty spaces.
2. **Avoid penalties**: Throughout the cleaning process, the Cleaner must also be careful to avoid penalties such as incurring step penalty without potential gain, colliding with static or dynamic obstacles, or landing in a hazardous zone.
3. **Battery management**: The agent has limited battery life, requiring it to manage its energy usage efficiently and recharge when necessary.

4. **Stochastic Transitions and Outcomes**: The environment introduces uncertainty in the agent's movement. Also, each movement action may not have a deterministic outcome.

The figure represents the initial 20 X 20 grid which was developed in Google Collab File for problem formulation. The labelling for robots as * (3), humans as ● (3), static obstacles as X (3) and hazards as ◆ (2) has been shown in the grid. Also, Light Gray corresponds to Clean Cells, Brown to Dirty Cells, Black to Static Obstacles, Green to Charging Stations & Yellow to time tunnels.



Learning Algorithms

The Cleaner employs Reinforcement Learning based algorithms with discount factor (γ=0.8), specifically focusing on:

1. **On-Policy Monte Carlo Control**: In this method, the agent updates its policy using episodes generated by following its current policy.
2. **Off-Policy Monte Carlo Control**: The agent learns an optimal target policy using episodes generated from a more exploratory behaviour policy, often employing importance sampling to correct for the difference between the two policies.

Goal

The goal of this problem is to develop and evaluate reinforcement learning agents that can efficiently learn optimal policies in complex environments with dynamic obstacles, stochastic transitions, and constraints. The performance of the agents will be measured based on their ability to maximize cumulative rewards and successfully navigate through the environment while minimizing penalties and avoiding termination conditions.

Implemented RL Algorithms

On-Policy Monte Carlo Control: ϵ-Greedy First Visit

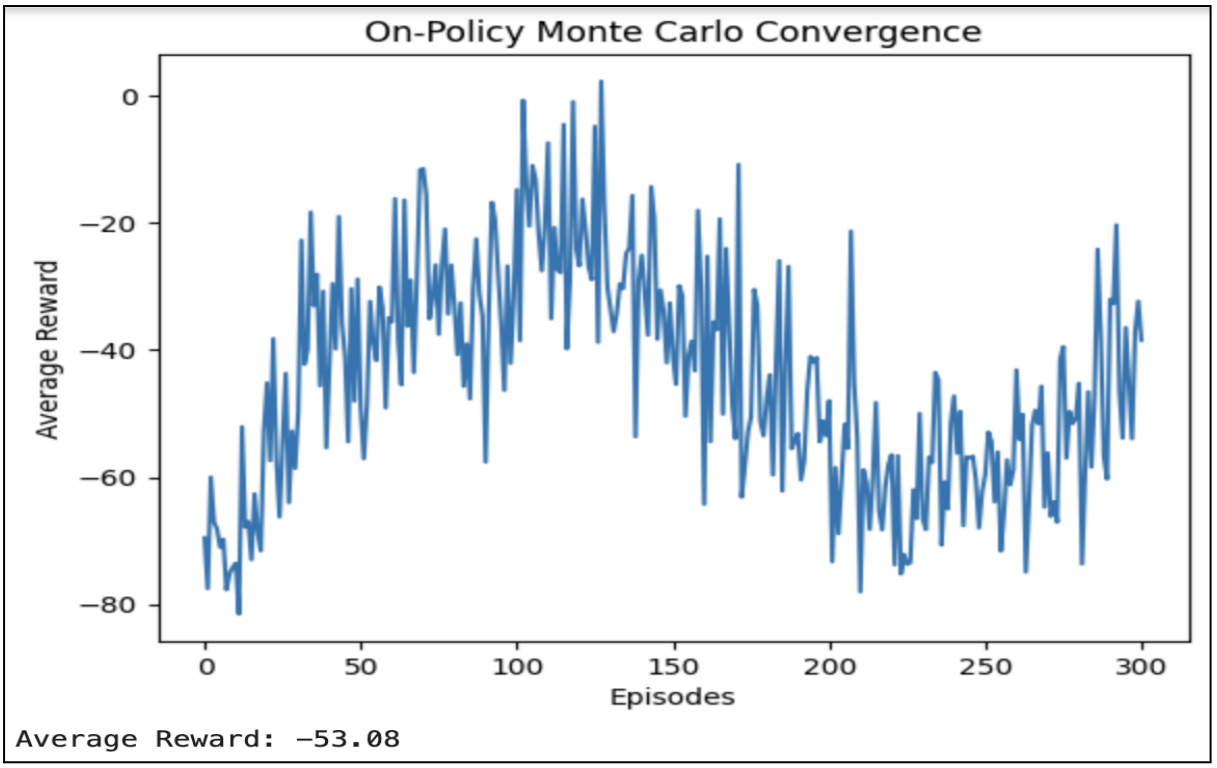
The On-Policy Monte Carlo algorithm used in the Google Collab File utilizes the **First-Visit Monte Carlo method** to iteratively evaluate and improve a policy based on simulations. In this approach, only the first occurrence of a state-action pair within each episode is used for updating the value estimates. The algorithm begins with initializing the environment and setting key parameters such as the discount factor (γ) and the exploration rate (ϵ), employing an **ϵ-Greedy strategy** for action selection, balancing between following the current policy and exploring new actions. With probability ϵ, the agent selects a random action from the available actions otherwise the agent chooses the action that has the highest estimated value.

- **Policy Initialisation**: Each state's policy is set to a random action chosen from the set of possible actions. This is done to ensure that all actions are initially explored.
- **Episode Generation:** It starts from a random state and follows an epsilon-greedy strategy selecting the best action according to the current policy with probability $1-\epsilon$, and choosing a random action with probability ϵ. The agent performs the action, moves to the next state, and records the state, action, and reward. This process continues until the episode ends or the battery depletes (by -1 in each time step).
- **Policy Updation:** We first generate an episode using the current policy. It then calculates the return (G) for each state-action pair in the episode, updating the sum and count of returns for the first visits only. The Q-values are updated based on these returns. Finally, the policy is improved by setting it to the action with the highest Q-value for each state. This process is repeated to iteratively enhance the policy.
- **Policy Evaluation:** We assess the performance of the learned policy by running several test episodes. Each episode starts from a random state and follows the learned policy to determine actions. It accumulates the total reward received in each episode and updates the state and battery level until the episode ends. The average reward across all test episodes is then computed and returned.
- **Convergence:** It is monitored by tracking changes in average rewards over time, assessed when the average reward over the last 100 episodes stabilizes compared to the previous 100 episodes.

Sample results with the 1000 number of episodes, 100 number of test episodes, $\gamma = 0.8$, convergence criteria ($\Delta R100 < 0.1$) and highly exploratory policy with $\epsilon = 0.9$ have been

shown below. The convergence was reached in around 300 episodes with the average reward obtained of -53.08.

The on-policy methods are generally considered a safer strategy, often converging faster and having better online performance. However, these methods are susceptible to getting stuck in a local optima, and hence, less likely to find a global optimal policy.



Off-Policy Monte Carlo Control: ϵ-Greedy First Visit

The Off-Policy Monte Carlo algorithm implemented in the Google Collab File utilises a **First-Visit Monte Carlo method** to evaluate and improve a policy through simulations, with a focus on separating the behaviour policy from the target policy.

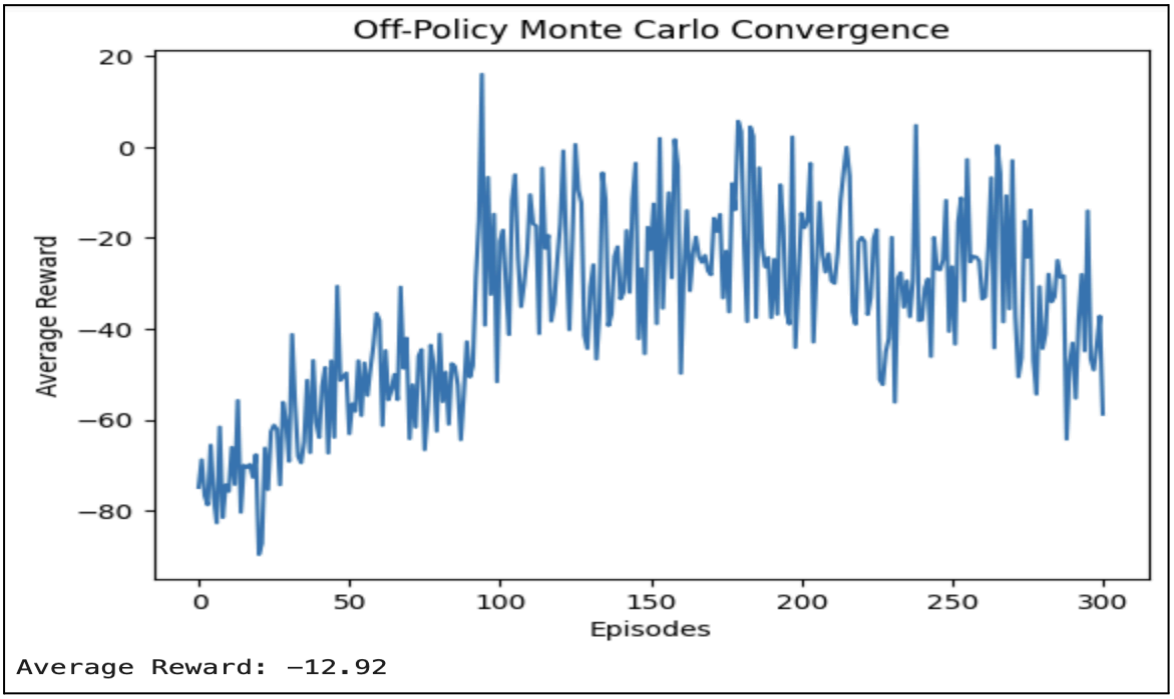
- **Policy Initialization:** The target policy and behaviour policy implement an **ϵ-Greedy strategy** where the agent explores by choosing a random action with probability ϵ, otherwise exploits by choosing the action with the highest estimated value.
    - Behaviour policy: with a higher exploration rate (behaviour ϵ) for generating diverse episodes to ensure sufficient exploration, making it stochastic.
    - Target policy: with a lower exploration rate (target ϵ) for making more greedy decisions making it deterministic and initialized to prefer actions with higher estimated values.
- **Episode Generation:** Episodes are generated by following the behaviour policy. The agent starts from a random state, and with a probability equal to the behaviour policy's ϵ, it chooses a random action. Otherwise, it selects an action based on the

behaviour policy. The agent moves through the environment, collecting state-action reward tuples until the episode ends or the battery depletes (by -1 in each time step).

- **Policy Updation:** After generating an episode, the algorithm updates the policy by calculating each state-action pair's return (G), applying **importance sampling** (W) to adjust for the difference between the behaviour and target policies. The Q-values are updated based on the weighted returns, and the target policy is improved by selecting the action with the highest Q-value for each state.
- **Policy Evaluation:** Running multiple test episodes evaluates The learned target policy. For each episode, the agent starts from a random state and follows the learned target policy to determine actions. The total reward is accumulated, and the average reward across all test episodes is computed and returned.
- **Convergence:** Convergence is monitored by tracking changes in average rewards over time. The convergence criterion is assessed when the average reward over the last 100 episodes stabilizes compared to the previous 100 episodes.

Sample results with 1000 episodes, 100 test episodes, $\gamma = 0.8$, explorative behaviour-$\epsilon = 0.9$, target-$\epsilon = 0.1$ and convergence criteria ($\Delta R100 < 0.1$) are shown below. Convergence was achieved in around 300 episodes, with an average reward obtained of -12.92.

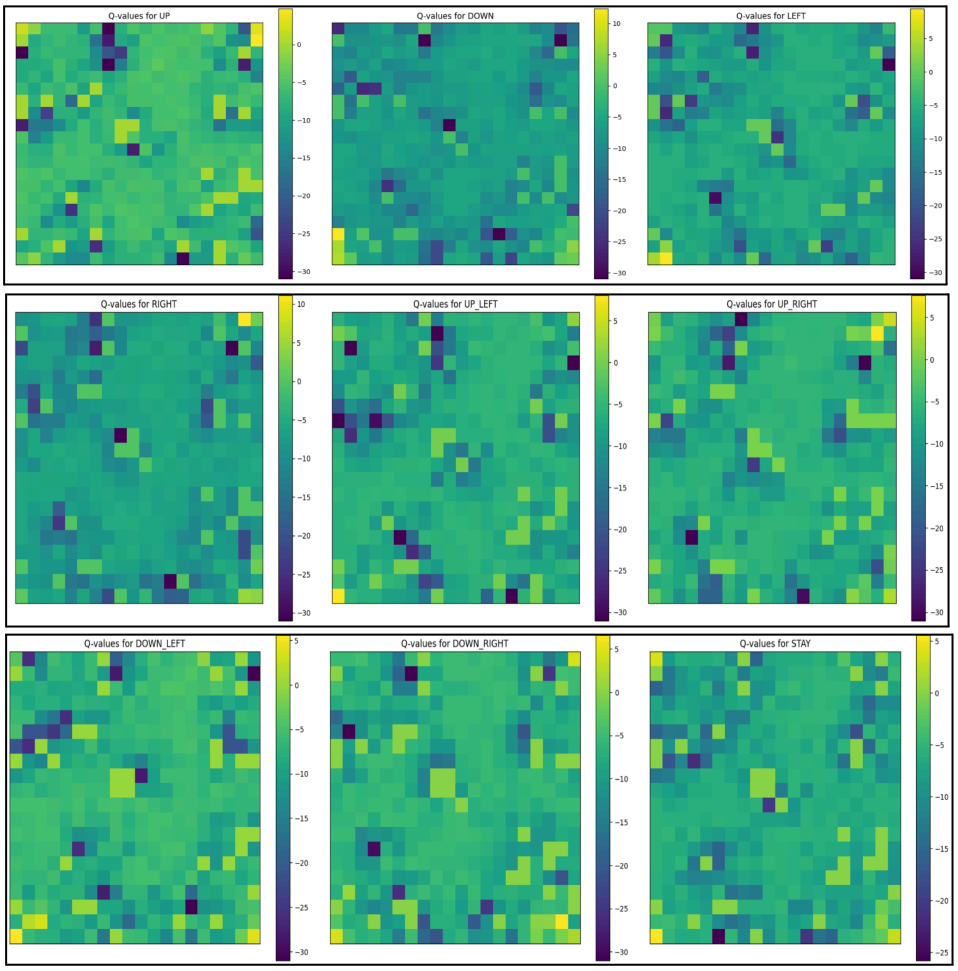
The off-policy methods are more likely to find a global optimal policy and less likely to get stuck in a local minimum. However, the policy learned may not be safe and may not have better online performance.



Visualising Q-Values

The heatmaps of the final Q-Values obtained at convergence for both on-policy and off-policy Monte Carlo methods have been show below.

On-Policy Q-Value Heatmaps: Shows concentrated Q-values for certain state-action pairs



Off-Policy Q-Value Heatmaps: Shows more exploration across various actions and a wider distribution of Q-values across state-action pairs.