COMP 138 RL: Programming Assignment 2

Shivprasad Shivratri

The **Racetrack** problem is a reinforcement learning task where an agent must navigate a track optimally while obeying acceleration constraints and avoiding track boundaries. This problem applies an **Off-Policy Monte Carlo Method** to find an optimal driving policy.

Problem Formulation:

The racetrack is represented as a discrete grid where the agent (race car) starts from a designated starting region and must reach the finish line while following predefined movement rules:

- The agent's position and velocity are discrete.
- The agent can accelerate in 9 ways:
  (-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1).
- The velocity is constrained between [0,4] in x & y direction. The velocity cannot be (0,0) unless on the start line.
- Crashing into a boundary resets the agent to the start line with zero velocity.
- A small probability (0.1) of no acceleration is introduced to simulate imperfect controls.

Environment and Implementation:

The environment is implemented using a custom class `Racetrack` that extends OpenAI Gym's `Env`. The environment handles:

- **Track Generation:** Using predefined track layouts stored in `.npy` files.
- **State Representation:** A tuple (`x, y, vx, vy`) where `x, y` are position coordinates and `vx, vy` are velocity components.
- **Actions and Transitions:** Actions modify velocity, which in turn updates the position. If a move results in crossing the finish line, the episode ends.
- **Boundary Handling:** If a move crosses a boundary, the agent is reset to the start.
- **Rendering:** A visualization is provided using `Pygame`.

Monte Carlo Control for Policy Optimization:

We employ an **off-policy Monte Carlo control** method to find the optimal driving policy:

1. **Initialize Q-values:** Randomly initialize $Q(s, a)$ for all state-action pairs.
2. **Behavior Policy:** Define an epsilon-greedy policy to explore different actions.
3. **Generate Trajectories:** Run episodes where the agent follows the behavior policy and records state-action-reward sequences.
4. **Update Action-Value Estimates:** Reverse traverse recorded trajectories and update Q-values using weighted importance sampling:

$$Q(s, a) \leftarrow Q(s, a) + \frac{W}{C(s, a)}(G - Q(s, a)) \tag{1}$$

   where W is the importance sampling ratio, G is the return, and $C(s, a)$ is a cumulative weight sum.

5. **Policy Improvement:** Extract the greedy policy from Q-values.

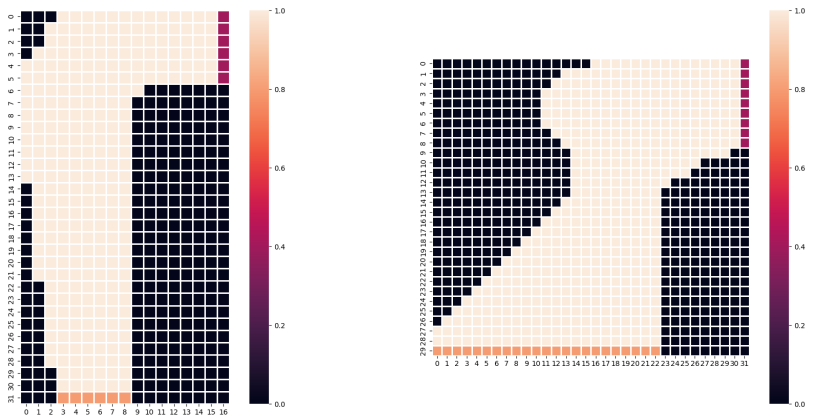
Off-policy MC control, for estimating $\pi \approx \pi_*$

Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
  $Q(s, a) \in \mathbb{R}$ (arbitrarily)
  $C(s, a) \leftarrow 0$
  $\pi(s) \leftarrow \arg\max_a Q(s, a)$ (with ties broken consistently)

Loop forever (for each episode):
  $b \leftarrow$ any soft policy
  Generate an episode using b: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  $W \leftarrow 1$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    $C(S_t, A_t) \leftarrow C(S_t, A_t) + W$
    $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \frac{W}{C(S_t, A_t)} [G - Q(S_t, A_t)]$
    $\pi(S_t) \leftarrow \arg\max_a Q(S_t, a)$ (with ties broken consistently)
    If $A_t \neq \pi(S_t)$ then exit inner Loop (proceed to next episode)
    $W \leftarrow W \frac{1}{b(A_t|S_t)}$

Experiments:

The algorithm was tested on two racetrack layouts (Track A and Track B) as shown:



(a) Track A (b) Track B

Figure 1: Racetrack Layouts

The agent was trained using **1,000,000 episodes** on each track.

Reward Convergence:

Training plots show that rewards stabilize as the agent learns the optimal policy.

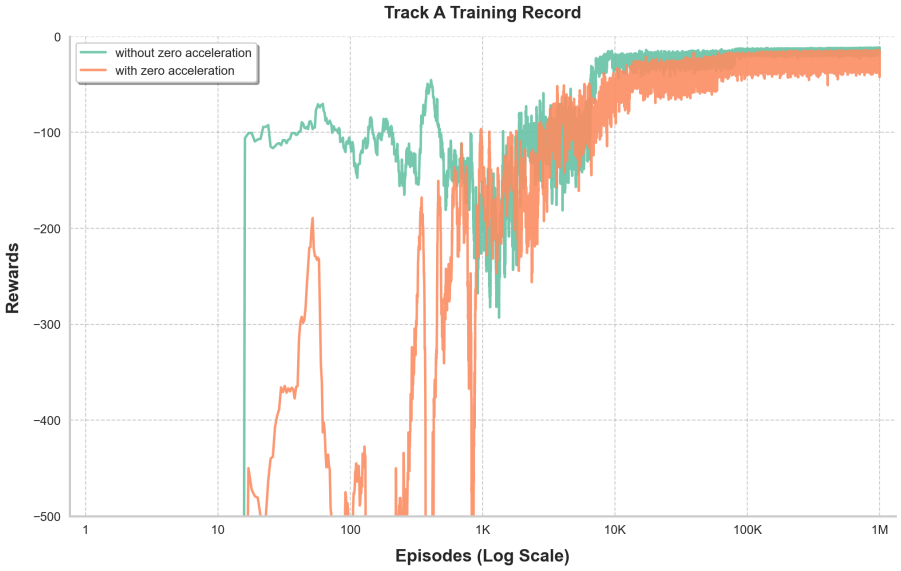


Figure 2: For Track A, Reward convergence over episodes with comparison of runs with and without acceleration.

Figure 3: For Track B, Reward convergence over episodes with comparison of runs with and without acceleration.

Optimal Trajectories:

Sample trajectories demonstrate efficient paths to the goal, avoiding unnecessary turns and crashes.

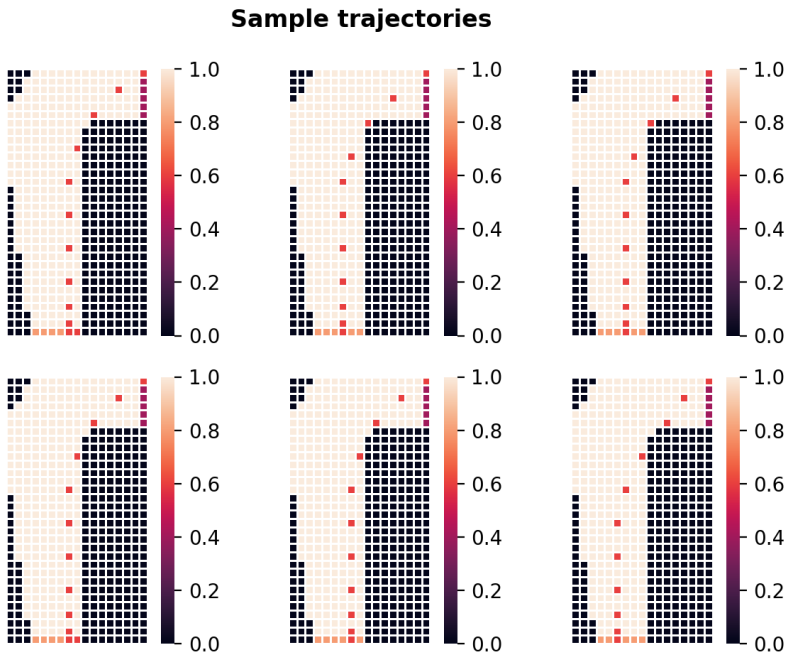


Figure 4: For Track A, Trajectory Visualization.

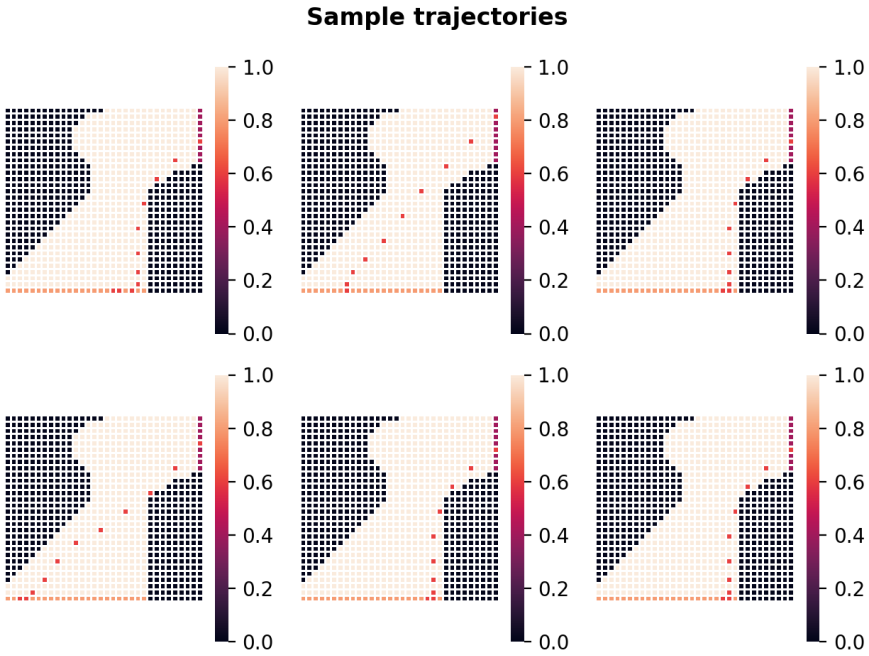


Figure 5: For Track B, Trajectory Visualization.

Conclusion:

The implementation successfully learned an optimal policy for navigating the racetrack using Monte Carlo control. The off-policy approach allowed efficient learning despite disturbances in the acceleration. Overall, Monte Carlo control proved effective for optimizing navigation in the racetrack environment, demonstrating the ability of reinforcement learning for sequential decision-making problems.

Archive of the code:

Clicking here shall redirect you to the Github Repository for Programming Assignment #2

Effect of allowing negative velocity in X-direction:

We wanted to see how the learning/rewards and the final results would be affected when the agent is allowed to move in the left direction (negative X-velocity). According to the results attached below, we could not see any significant changes in the reward as opposed to the results from our earlier experiments. One change we might have observed is in the speed, but we did not track it in our experiments.

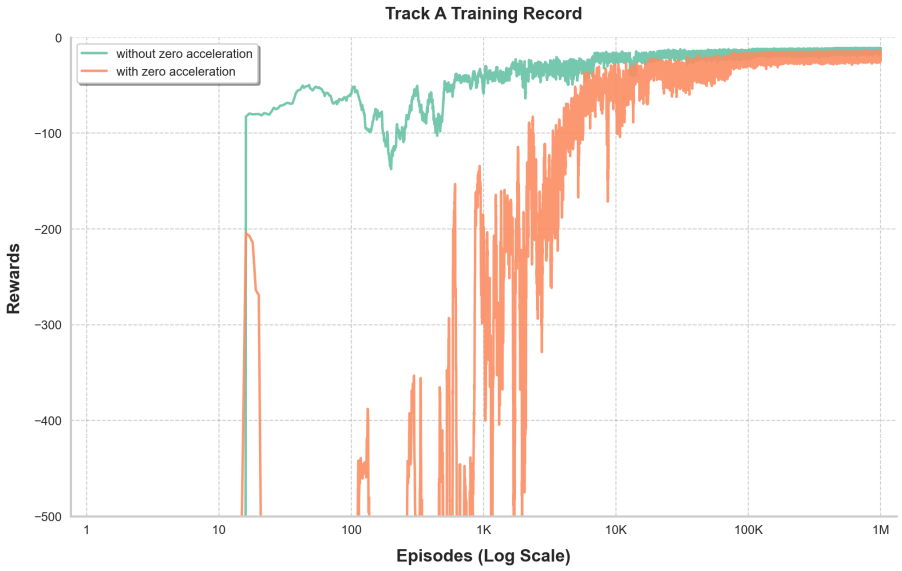


Figure 6: Reward convergence over episodes with comparison of runs with and without acceleration.

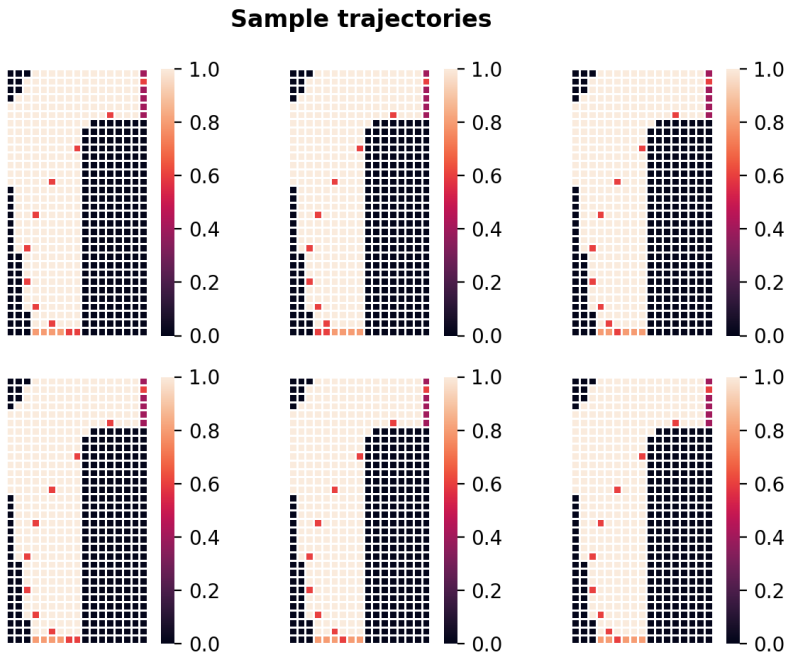


Figure 7: Trajectory Visualization.