Assignment 1
Multi-Agent Reinforcement Learning

Rudra Baunk 22268

Note: The manual derivation for the first three iterations of Question 1 (parts b and c) was performed on a 3×3 grid to verify correctness. The handwritten derivations are included in the submitted ZIP file.

QUESTION 1

(a) Formulate the problem explicitly as an MDP.

The navigation problem is modeled as a Markov Decision Process $\mathcal{M} = (S, A, P, R, \gamma)$.

State Space: $s = (x, y, \theta)$, where x, y are grid coordinates and $\theta \in \{0, 1, 2, 3\}$ denotes orientation (East, North, West, South). For an $N \times N$ grid with k obstacles:

$$|S| = 4(N^2 - k).$$

For the 10×10 grid with 5 obstacles, $|S| = 4(100 - 5) = 380$.

Action Space: $A = \{\text{Forward}, \text{TurnLeft}, \text{TurnRight}\}$.

Transition Function: Forward moves with probability 0.8, slips left/right with probability 0.1 each. Turning is deterministic with $\theta' = (\theta \pm 1) \mod 4$. Boundary or obstacle contact results in a terminal collision state.

Reward Function: -1 per step, -100 for collision (terminal), $+50$ for reaching the goal (terminal).

Note: A 3×3 grid was used in parts (b) and (c) to manually verify the first few iterations before extending the same formulation to the full 10×10 grid.

(b) Value Iteration

Implementation Note To clearly demonstrate the Bellman update process and manually verify intermediate computations, a smaller 3×3 grid was first implemented. The reduced grid allows step-by-step validation of the first few iterations of Value Iteration. After verification, the same implementation was extended without modification to the full 10×10 grid for final experiments and performance analysis.

Value Iteration was implemented using the Bellman optimality update:

$$V_{k+1}(s) = \max_a \sum_{s'} P(s'|s, a)\big(R(s, a, s') + \gamma V_k(s')\big).$$

Verification on 3×3 Grid A reduced 3×3 grid was used to manually verify the first three updates.

Iteration	$V(0,0,0)$	$V(1,2,0)$
1	-1.00	20.00
2	-1.90	20.00
3	-2.71	20.00

Table 1: First three Value Iteration updates (3×3 grid)

The state $(1, 2, 0)$ quickly captures the goal reward, while $(0, 0, 0)$ reflects gradual propagation of step cost under discounting.

Full 10×10 Grid The same implementation was applied to the 10×10 grid.

Iteration	$V(0,0,0)$	$V(1,2,0)$
1	-1.00	-1.00
2	-1.90	-1.90
3	-2.71	-2.71

Table 2: First three Value Iteration updates (10×10 grid)

Convergence required 89 iterations (runtime 0.1437 sec, memory 51968 bytes).

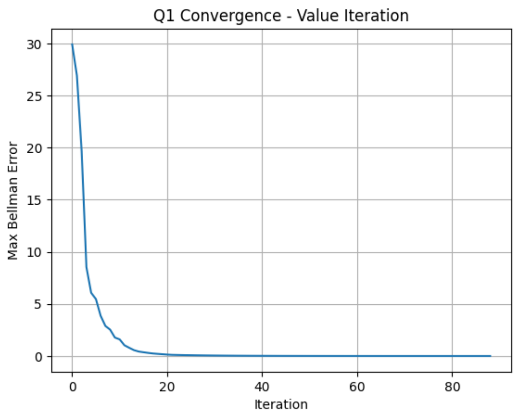

Figure 1: Bellman error convergence (10×10 grid)

(c) Policy Iteration

Implementation Note Similar to Value Iteration, Policy Iteration was first tested on the reduced 3×3 grid to manually verify policy evaluation and improvement steps. Once correctness was confirmed, the identical algorithm was applied to the full 10×10 environment for convergence comparison and performance evaluation.

Policy Iteration alternates between policy evaluation and policy improvement.

Policy evaluation computes:

$$V^{\pi}(s) = \sum_{s'} P(s'|s, \pi(s))\big(R(s, \pi(s), s') + \gamma V^{\pi}(s')\big).$$

Verification on 3×3 Grid The reduced 3×3 grid was used to verify early evaluation updates.

Iteration	$V(0,0,0)$	$V(1,2,0)$
1	-1.00	20.00
2	-1.90	20.00
3	-2.71	20.00

Table 3: First three Policy Evaluation updates (3×3 grid)

The values match those from Value Iteration, confirming correctness.

Full 10×10 Grid For the 10×10 grid:

- Policy improvement steps: 9
- Runtime: 0.3429 sec
- Memory usage: 113486 bytes

The policy became stable after 9 improvement steps.

(d) Comparison of Value Iteration and Policy Iteration

Method	Iterations	Runtime (sec)	Memory (bytes)
Value Iteration	89	0.1556	51968
Policy Iteration	9	0.3429	113486

Table 4: Performance comparison (10×10 grid)

Observation:

- Value Iteration required more iterations but converged faster in runtime.
- Policy Iteration required fewer improvement steps but had higher runtime.
-
- Policy Iteration required more memory since it stores both the value function and the policy table, whereas Value Iteration stores only the value table.

(e) Orientation-Aware Policy Visualization

The final optimal policy was visualized using orientation-dependent arrows. Since orientation $\theta \in \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}\}$ is part of the state, a separate policy map is shown for each orientation.

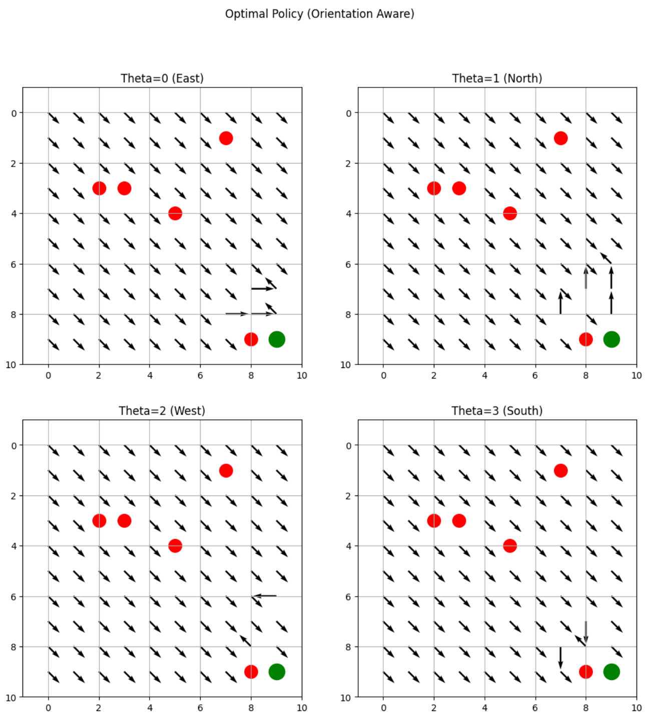

Figure 2: Optimal orientation-aware policy for the 10×10 grid

Each arrow represents the optimal action (Forward, TurnLeft, TurnRight) for a given (x, y, θ) state.

Observations:

- The policy guides the robot toward the goal while avoiding obstacles.
- Near boundaries and obstacles, turning actions appear frequently to correct orientation.
- The optimal strategy is not purely shortest-path; it accounts for orientation dynamics and slip probability.
- Different orientations at the same grid cell can produce different optimal actions.

This confirms that incorporating orientation into the state space significantly affects the structure of the optimal policy.

(f) Reward Structure Comparison

Reward Structure	VI Iter	PI Iter	VI Time	PI Time
(-1, -100, 50)	89	9	0.2438	0.6966
(-0.1, -100, 50)	89	8	0.2537	0.4219
(-1, -100, 200)	89	9	0.1410	0.3564

Table 5: Comparison under different reward structures

Observation: Increasing goal reward or reducing step penalty reduces policy iteration steps, while value iteration count remains unchanged.

QUESTION 2

(a) Formulate the problem explicitly as an MDP.

The battery-aware navigation problem is modeled as a Markov Decision Process $\mathcal{M} = (S, A, P, R, \gamma)$.
State Space: $s = (x, y, b)$, where (x, y) are grid coordinates and $b \in \{0, 1, \ldots, B\}$ denotes the battery level.
Action Space:

$$A = \{\text{MoveUp}, \text{MoveDown}, \text{MoveLeft}, \text{MoveRight}, \text{Recharge}\}.$$

Each movement action changes the robot's position and consumes one unit of battery.
Transition Function: Movement actions update (x, y) deterministically and reduce battery by 1. Recharge restores battery to full capacity B and is allowed only at designated charging stations. If $b = 0$ away from a charging station, the system transitions to a terminal failure state.
Reward Function: Move: -1, Recharge: -2, Battery depletion: -100 (terminal), Goal reached: $+100$ (terminal).
Terminal States: Goal state and battery-depletion failure state.

(b) Value Iteration

Value Iteration was applied to compute the optimal value function.

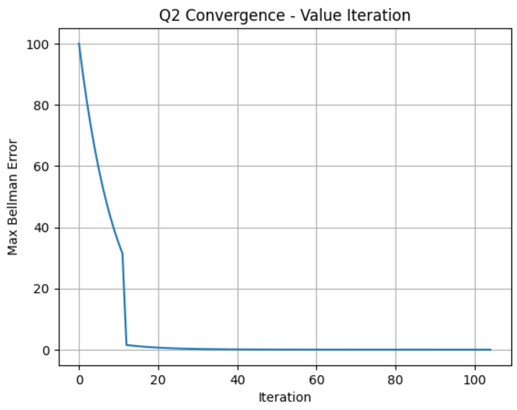

Figure 3: Bellman Error Convergence for Value Iteration

Convergence Plot The Bellman error decreases rapidly and approaches zero, confirming convergence.

Convergence Statistics

- Total iterations: **105**

(c) Effect of Discount Factor

Value Iteration was repeated for different values of γ.

γ	Iterations to Converge
0.99	27
0.9	105
0.7	33

Table 6: Effect of discount factor on convergence

Observation:

- Higher γ values consider long-term rewards more strongly.
- Moderate $\gamma = 0.9$ required the highest iterations.
- Lower γ values reduce long-term influence and converge faster.

The convergence behavior depends not only on γ but also on reward magnitude and termination structure.

(d) Policy Visualization

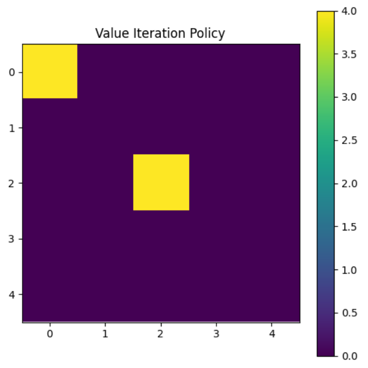

Figure 4: (a) Value Iteration

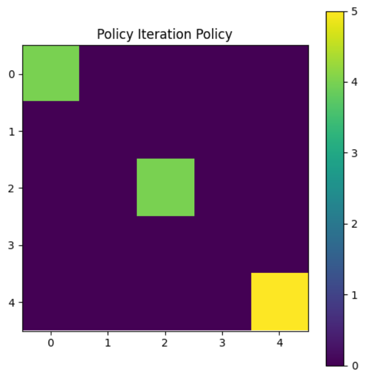

Figure 5: (b) Policy Iteration

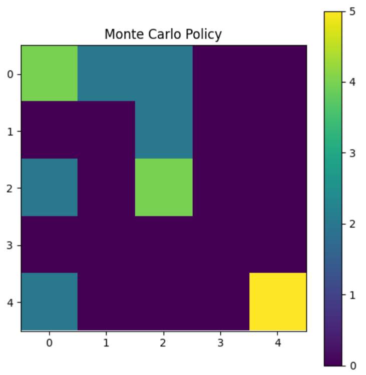

Figure 6: (c) Monte Carlo

Figure 7: Comparison of policies obtained using Value Iteration, Policy Iteration, and Monte Carlo

Observation:

- Value Iteration and Policy Iteration produce nearly identical optimal policies.
- Monte Carlo shows minor deviations due to sampling-based estimation.
- Differences are mainly observed in states visited less frequently.

(e) Emergence of Battery-Awareness

Battery-awareness naturally emerges from the optimal policy:

- For higher γ (e.g., 0.99), the agent plans long-term and recharges early to avoid failure.
- For lower γ, the agent prioritizes immediate reward and may delay recharging.
- The penalty of -100 for depletion strongly encourages safe battery management.
- Recharge action appears strategically before battery reaches zero.

Thus, battery management behavior emerges automatically from reward optimization without explicitly programming safety rules.

(f) Monte Carlo Comparison

- Value Iteration iterations: 105
- Policy Iteration steps: 6
- Monte Carlo episodes: 800

Comparison:

- Value and Policy Iteration converge faster due to full model knowledge.
- Monte Carlo requires many episodes for accurate estimation.
- Model-based methods are more stable.
- Monte Carlo is suitable when transition model is unknown.

Question 3

(a) Formulate the problem explicitly as an MDP.

The risk-sensitive robot navigation task is modeled as a Markov Decision Process $\mathcal{M} = (S, A, P, R, \gamma)$.
State Space:

$$s = (x, y, h),$$

where (x, y) are grid coordinates and h indicates whether the robot is in proximity to a hazardous region.
Action Space:

$$A = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}\}.$$

Transition Function: In normal regions, actions succeed with high probability and move the robot to the intended neighboring cell. In hazardous regions, actions have a small slip probability that may result in a transition to a catastrophic terminal state.
Reward Function: Step cost: -1, Goal reached: $+50$ (terminal), Catastrophic failure: -200 (terminal).
Terminal States: The goal state and the catastrophic failure state.
Value Iteration was applied to compute the optimal value function under stochastic slip conditions near hazards.
The Bellman error decreases monotonically and converges after 9 iterations.

Convergence Statistics

- Iterations to converge: 9
- Runtime: 0.0012 sec

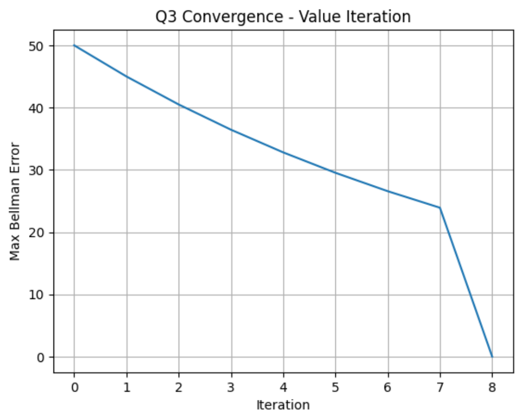

Figure 8: Bellman Error Convergence for Value Iteration (Q3)

(b) Optimal Policy (Slip Probability = 0.1)

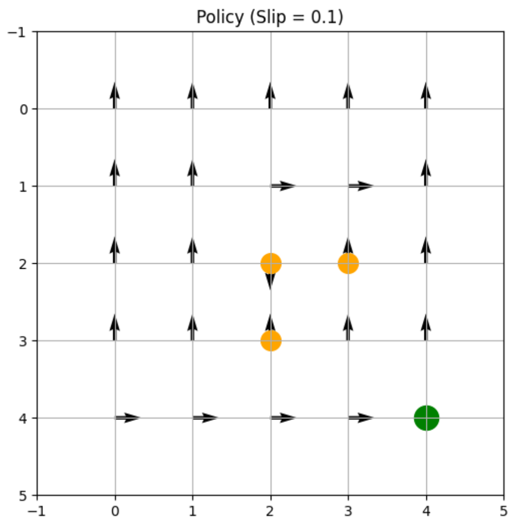

Figure 9: Optimal Policy for slip probability 0.1

Observation:

- The robot follows a relatively direct path toward the goal.
- Hazard-adjacent states are avoided moderately.
- Since slip probability is low, the agent tolerates moderate risk.

(c) Increased Slip Probability (Slip = 0.3)

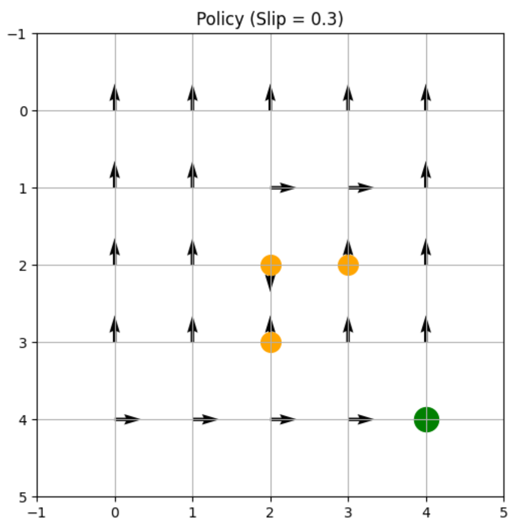

Figure 10: Optimal Policy for slip probability 0.3

Observation:

- The policy becomes more conservative.
- The robot avoids states close to hazards.
- Longer but safer paths are preferred.

(d) Policy Comparison

Increasing slip probability significantly alters the optimal strategy.

- For slip = 0.1, the agent balances risk and path length.
- For slip = 0.3, catastrophic penalty dominates decision making.
- The optimal policy shifts from shortest-path behavior to safety-first navigation.

(e) Why Shortest-Path Intuition Fails

In deterministic environments, shortest path minimizes total step cost. However, near hazardous regions:

- There is a non-zero probability of catastrophic failure.
- Expected return includes weighted catastrophic penalty (-200).
- Risk-adjusted expected value becomes lower for shorter risky paths.

Thus, even if a path is geometrically shorter, its expected return may be worse due to stochastic risk. The agent therefore selects longer but safer routes.

(f) Monte Carlo Comparison

Metric	VI	PI	MC
Iterations/Episodes	9	7	5000
Runtime (sec)	0.0012	0.0089	9.262

Table 7: Comparison of VI, PI, and Monte Carlo (Q3)

Observation:

- Model-based methods (VI, PI) converge rapidly.
- Monte Carlo requires many episodes due to sampling noise.
- PI required fewer iterations but slightly higher runtime.

- VI is computationally efficient for small grids.

Overall Comparative Analysis

The three problems illustrate different aspects of decision-making under uncertainty.

Problem	State Size	VI Iter	PI Iter	Key Insight
Q1	380	89	9	Orientation affects policy structure
Q2	Larger (x,y,b)	105	6	Resource awareness emerges
Q3	Small (x,y,h)	9	7	Risk alters shortest-path behavior

Observations:

- Increasing state complexity (orientation, battery, hazard proximity) significantly changes optimal behavior.
- Value Iteration generally requires more iterations but has simpler updates.
- Policy Iteration converges in fewer improvement steps but involves more computation per step.
- Monte Carlo requires many episodes due to sampling noise but does not require a known transition model.
- Risk and long-term penalties strongly influence expected return and alter policy structure.

Overall, the experiments demonstrate how modeling additional real-world constraints (orientation dynamics, battery limitations, and risk) fundamentally changes optimal decision-making behavior in reinforcement learning systems.