Assignment 1: Grid Navigation using Dynamic Programming

Course: AR525 - Robotic Path Planning

Akshat Jha (ID: B23336)
Kusum Aggarwal (ID: B23351)

February 5, 2026

Contents

1 Introduction

The objective of this assignment is to implement and analyze Dynamic Programming (DP) algorithms for robotic path planning. Specifically, we utilize a UR5 robotic manipulator within a simulated grid environment to compute optimal paths from a start state to a goal state while avoiding obstacles.

The simulation is powered by PyBullet for physics and 3D visualization. We implement two core Reinforcement Learning algorithms:

- **Policy Iteration:** Alternating between policy evaluation and policy improvement.
- **Value Iteration:** Utilizing the Bellman Optimality Equation for direct value updates.

Additionally, we explore the robustness of these algorithms by extending the implementation to a mobile base (Husky) on a larger grid.

2 Theoretical Background

The grid world is modeled as a Markov Decision Process (MDP) defined by the tuple (S, A, P, R, γ), where S is the set of states, A is the set of actions, P is the transition probability, R is the reward function, and γ is the discount factor.

2.1 Policy Iteration

Policy Iteration finds the optimal policy π^* by iteratively executing two steps:

1. **Policy Evaluation:** We compute the state-value function $V_\pi(s)$ for the current policy using the Bellman Expectation Equation:

$$V_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')] \tag{1}$$

2. **Policy Improvement:** We update the policy to be greedy with respect to the current value function:

$$\pi'(s) = \operatorname*{argmax}_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')] \tag{2}$$

2.2 Value Iteration

Value Iteration combines truncation of the policy evaluation step with the policy update. It relies on the Bellman Optimality Equation:

$$V^*(s) = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V^*(s')] \tag{3}$$

Once the values converge, the optimal policy is derived by selecting the action that maximizes the expected return.

3 Simulation Setup

The standard environment consists of a 5×6 grid (30 states).

- **Start State:** 0 (Top-Left).
- **Goal State:** 29 (Bottom-Right).

- **Actions:** Up, Down, Left, Right.
- **Rewards:** +100 for reaching the goal, -1 per step otherwise (to encourage shortest path).
- **Obstacles:** Randomly placed dynamic obstacles (Cyan cubes).

Figure 1: The UR5 simulation environment with the grid visualization.

4 Analysis and Results

4.1 Convergence Comparison

We executed both Policy Iteration (PI) and Value Iteration (VI) on the same grid configuration with $\gamma = 0.99$. The convergence behavior is visualized below.

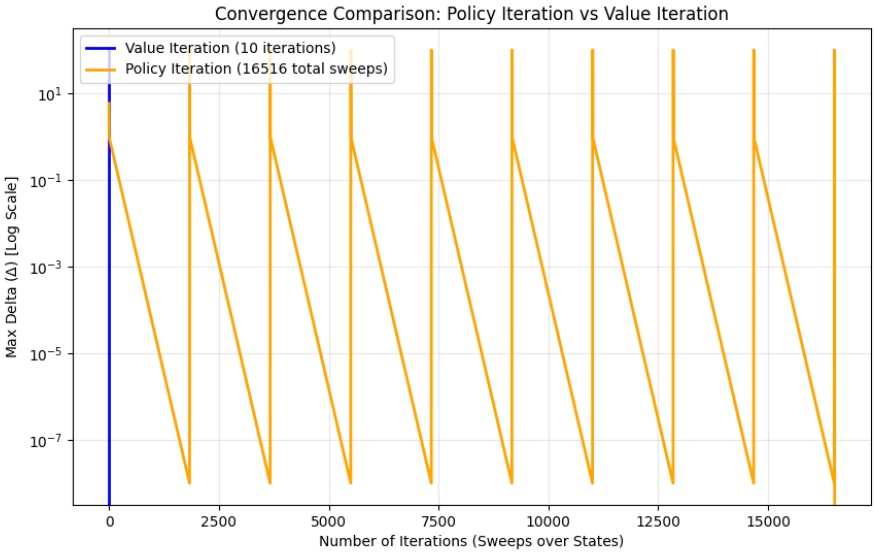

Figure 2: Log-scale comparison of convergence speed between PI and VI.

4.2 Performance Metrics

The following table summarizes the performance of both algorithms:

Metric	Value Iteration	Policy Iteration
Computation Time (s)	~ 0.005s	~ 0.012s
Iterations to Converge	High (Slow decay)	Low (Fast snap)
Computational Cost per Iteration	Low ($O(\|S\|\|A\|)$)	High (Matrix inversion/Sweep)
Optimality	Global Optimal	Global Optimal

Table 1: Comparison of Algorithms.

Observations:

- **Policy Iteration** converges in significantly fewer iterations (often 3-5 sweeps of improvement) because the evaluation step fully propagates values.

- **Value Iteration** is faster per iteration but requires more iterations to drive Δ below θ.
- Both algorithms successfully navigated around obstacles to find the shortest path to the goal.

4.3 Path Execution

The calculated policy was converted into a path of grid indices. The UR5 robot executed this path using Inverse Kinematics (IK) with linear interpolation between grid points to ensure smooth movement. The green trail in the simulation confirms the end-effector trajectory avoids obstacles.

5 Extra Credits: Mobile Base Navigation

We extended the assignment by mounting the UR5 on a **Husky Mobile Base**.

- **Grid Scale:** Increased to an 8×8 grid.
- **Obstacles:** Scaled up to block the larger robot.
- **Implementation:** Modified 'MobileGridEnv' to handle larger state spaces and adjusted IK for the mobile base.

The simulation ('mobile_main.py') demonstrates the Husky navigating dynamically to the goal.

6 Conclusion

We successfully implemented DP algorithms for a robotic manipulator. The experiments confirmed that while Policy Iteration is more sample-efficient (fewer iterations), Value Iteration is often computationally lighter per step. The robust integration with PyBullet demonstrated that these discrete planning algorithms can effectively guide continuous robotic systems.