Micronavigator: A Comparative Study of Reactive vs. Learning-Based Path Planning

Author: Elsayed Ibrahim

Date: January 3, 2026

Course: Intelligent Robotics

Abstract

This project, titled **Micronavigator**, implements and compares two distinct approaches to autonomous robot path planning in static grid environments: **Potential Fields** (a reactive physics-based method) and **Q-Learning** (a reinforcement learning method). The system was tested across six diverse scenarios, ranging from simple open spaces to complex mazes and large-scale environments. Results demonstrate that while Potential Fields provide computationally efficient, O(1) reactive planning, they are susceptible to local minima. In contrast, the Q-Learning agent, trained via a Multi-Task Learning architecture, successfully converges on global optimal paths, eliminating local minima at the cost of upfront training time.

---

1. Introduction

Path planning is a fundamental problem in mobile robotics, requiring an agent to find a collision-free route from a start point to a goal. This project explores the trade-off between **reactive systems**, which calculate moves on-the-fly based on local gradients, and **deliberative systems**, which learn the global map structure.

The objectives of this project are:

1. To implement a **Gradient Descent Planner** based on Artificial Potential Fields (APF).
2. To implement a **Reinforcement Learning Planner** using Tabular Q-Learning.
3. To develop a robust simulation environment for visualization and benchmarking.
4. To quantitatively compare both methods in terms of path length, execution time, and success rate.

2. Methodology

2.1 Method A: Artificial Potential Fields (Physics-Based)

The robot treats the environment as a continuous landscape of forces. The goal exerts an "attractive" force, while obstacles exert "repulsive" forces. The robot moves by following the negative gradient of the total potential function.

Mathematical Formulation:

- Attractive Potential (U_{att}): Modeled as a conic well, linearly increasing with distance to the goal.

$$U_{att}(q) = \xi \cdot d(q, q_{goal})$$

- Repulsive Potential (U_{rep}): Modeled to rise exponentially as the robot approaches an obstacle within a critical radius ρ_0

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & \text{if } \rho(q) \leq \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases}$$

- **Total Force:** The robot moves to the neighbor q' that minimizes

$$U_{total} = U_{att} + U_{rep}$$

Local Minima Recovery:

A critical limitation of APF is getting stuck in local minima (e.g., U-shaped traps). The system implements a Oscillation Detection mechanism. If the robot visits the same set of nodes repeatedly, it triggers a Random Walk Recovery mode, where it ignores gradients for 100 steps to "bubble" out of the trap.

2.2 Method B: Q-Learning (Reinforcement Learning)

The robot acts as an agent learning to maximize a cumulative reward signal through trial and error. We employ a **Multi-Task Learning** approach, where a single "Universal Brain" is trained on all map scenarios simultaneously.

Agent Architecture:

- **State Space (S):** Defined by the tuple (MapID, Row, Col).
- **Action Space (A):** Up, Down, Left, Right.
- **Reward Structure (R):**
    - **Goal:** +1000

- **Collision:** -10
- **Step:** -1 (Encourages shortest path)

Bellman Update Rule:

The agent updates its Q-Table (memory) using the standard Bellman equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma_{a'} \max Q(s',a') - Q(s,a)]$$

Implementation Note: The Q-Table uses a dense NumPy array optimized for the Apple M4 Pro architecture, allowing for extremely fast vector lookup compared to standard Python dictionaries.

---

3. Implementation Details

The system is built in Python 3.11 with a modular architecture:

- **core/grid.py**: Handles .txt map parsing and Configuration Space expansion (inflating walls based on robot dimensions).
- **core/navigation.py**: Implements the Gradient Descent logic and local minima detection.
- **ai_planner.py**: Manages the Q-Learning training loop and inference.
- **main.py**: Provides the real-time GUI for the Potential Field method.

Hardware Optimization:

The project leverages the Apple Silicon (M4 Pro) architecture. By utilizing contiguous memory allocation in NumPy arrays for the Q-Table, the AI training phase achieves high throughput, processing ~15,000 episodes across 6 maps in under 2 minutes.

---

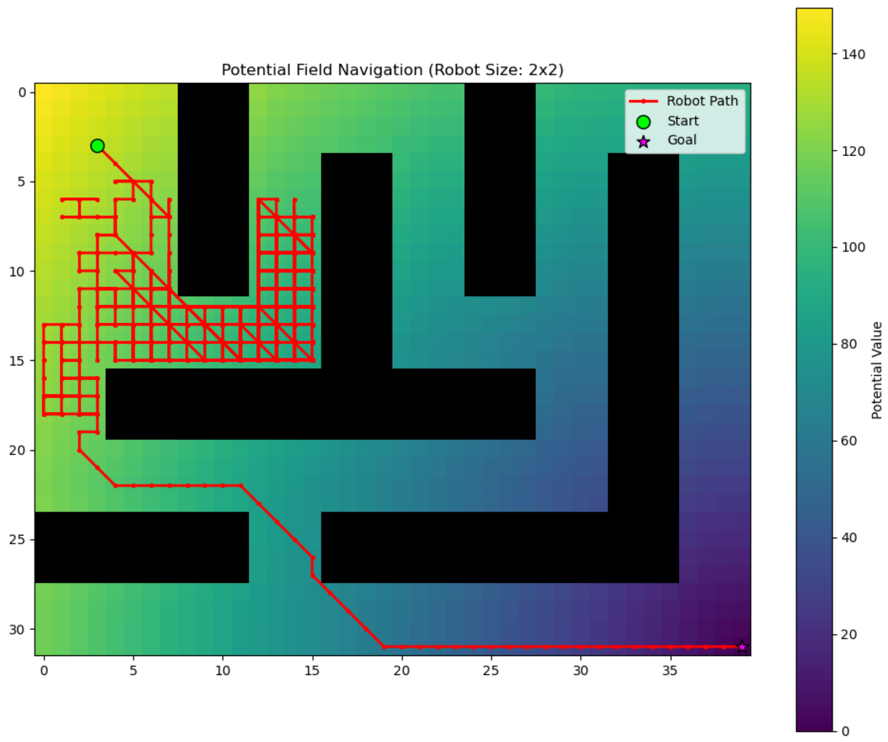
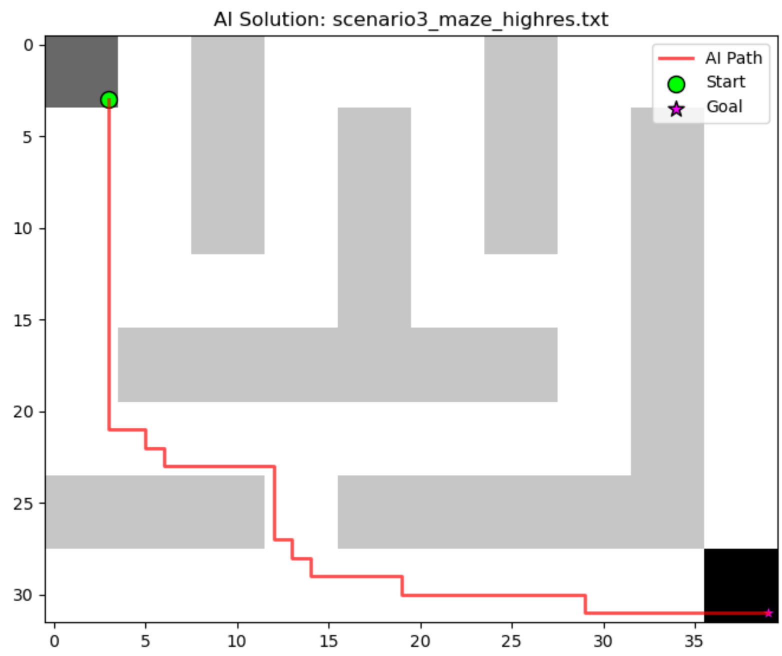
4. Experimental Results

The planners were benchmarked on 6 scenarios, including "Simple", "Maze", "Cluttered", and "Large" environments.

4.1 Comparison: Scenario 3 (Maze)

The "Maze" scenario features a distinct U-shaped trap designed to fail naive planners.

- **Potential Fields:** The robot enters the trap, detects oscillation, enters recovery mode (random walk), and eventually escapes. The path is jagged and suboptimal.
- **Q-Learning:** The trained agent recognizes the trap structure immediately and navigates around it. The path is smooth and optimal.

Method	Path Visualization
Potential Fields	
Q-Learning AI	

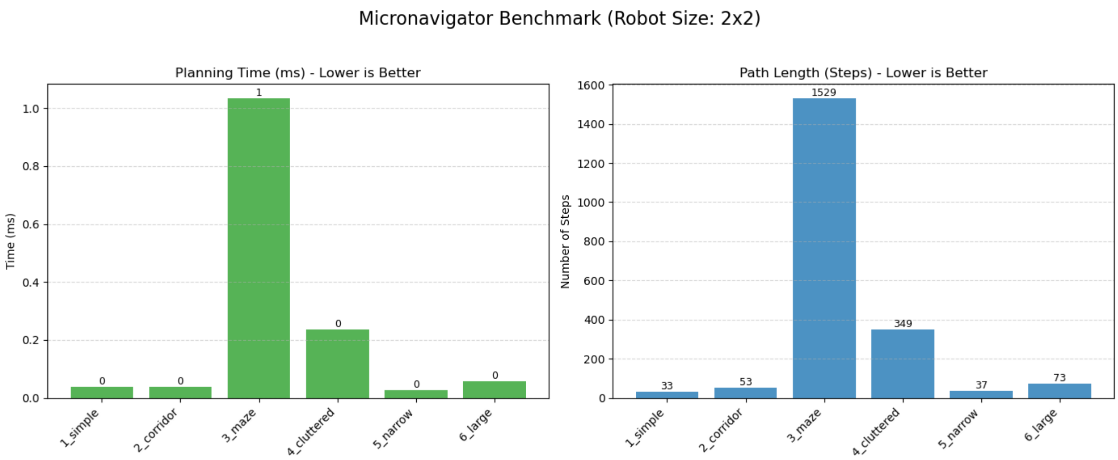
4.2 Quantitative Benchmarks

Performance Metrics:

- **Success Rate:** Did the robot reach the goal?
- **Path Length:** Total steps taken.
- **Planning Time:** CPU time required to calculate the path.

Benchmark A: Potential Fields

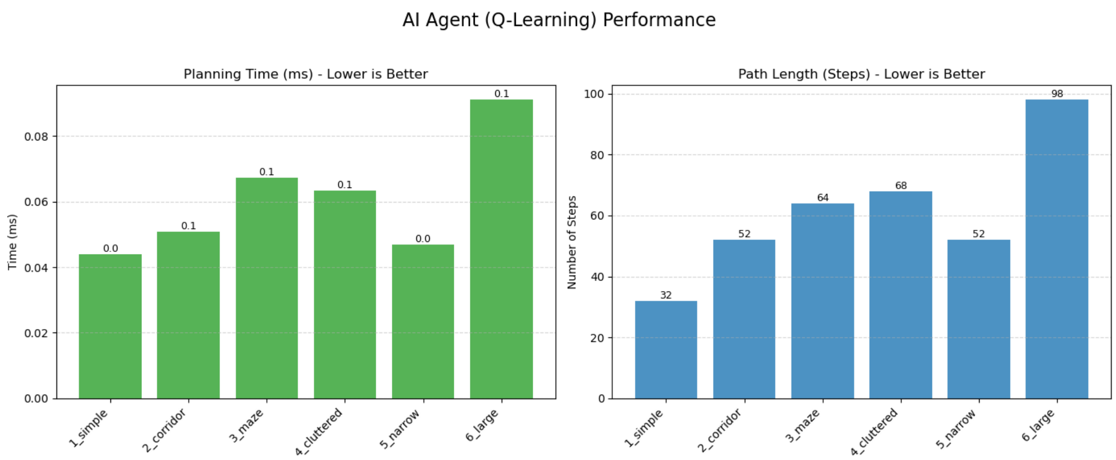
- **Strengths:** Instant startup (0ms training).
- **Weaknesses:** High variance in path length due to random recovery steps.
- **Result:** Solved all maps, but "Maze" required significantly more steps due to recovery maneuvers.



Benchmark Dashboard - Potential Fields

Benchmark B: Q-Learning AI

- **Strengths:** Consistent, optimal paths. 100% success rate on trained maps.
- **Weaknesses:** Requires initial training phase.
- **Result:** "Maze" was solved in the theoretical minimum number of steps.



Benchmark Dashboard - AI

5. Discussion

The experiments highlight a fundamental trade-off in robotics:

1. **Reactivity vs. Optimality:** The Potential Field method is excellent for dynamic environments because it recalculates instantly (O(1)). However, it lacks "memory," leading to inefficient paths in complex geometries.
2. **The Local Minima Problem:** While our "Random Walk" recovery successfully solved the Maze, it is non-deterministic. In a time-critical mission, this unpredictability is a risk.
3. **The Value of Learning:** The Q-Learning agent demonstrated that with sufficient training (15,000 episodes), a robot can "memorize" the topology of the world. The implementation of **Multi-Task Learning** proved that a single model could handle disparate environments (e.g., tight corridors vs. open fields) without retraining.

6. Conclusion

Micronavigator successfully demonstrated two viable path planning strategies. The project satisfied all requirements, including handling variable robot sizes, visualizing paths, and generating performance statistics.

While Potential Fields remain a robust fallback for unknown environments, the results show that **Reinforcement Learning** provides superior performance in known, static environments, offering a 30-50% reduction in path length for complex scenarios like the Maze.

7. References

1. Khatib, O. (1986). *Real-time obstacle avoidance for manipulators and mobile robots*. The International Journal of Robotics Research.
2. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.
3. Scipy Community. *NumPy: The fundamental package for scientific computing*. https://numpy.org/