Micro-Navigator: Potential Field Path Planning System

Technical Documentation

Author: Path Planning for Autonomous Navigation **Date:** December 2024

Abstract

This document presents the design, implementation, and evaluation of Micro-Navigator, a path planning system for rectangular robots navigating grid-based environments. The system employs potential field theory combined with A* search to generate collision-free trajectories while accounting for robot geometry through obstacle inflation. Comprehensive performance evaluation across multiple scenarios demonstrates the effectiveness and reliability of the approach.

---

1. Introduction

1.1 Problem Statement

Autonomous navigation in constrained environments requires efficient path planning algorithms that can handle static obstacles while accounting for robot dimensions. The Micro-Navigator system addresses this challenge by implementing a hybrid approach combining potential field methods with informed search algorithms.

1.2 System Overview

The system comprises the following core components:

- **Grid Map Representation**: Static occupancy grid encoding free space and obstacles
- **Potential Field Generator**: Creates navigation functions using attractive and repulsive forces
- **Path Extraction Module**: A* search guided by potential field heuristics
- **Robot Geometry Handler**: Obstacle inflation for rectangular robot footprints
- **Visualization Engine**: Path and map rendering capabilities
- **Performance Evaluator**: Statistical analysis across multiple scenarios

1.3 Key Features

- Variable robot size support (configurable width and height)
- Potential field-based navigation with configurable parameters
- A* pathfinding with gradient descent fallback
- Comprehensive planning statistics (time, nodes explored, path cost)
- Multi-scenario performance evaluation
- Visualization of paths and occupancy grids

---

2. Methodology

2.1 Potential Field Theory

The navigation potential field combines two components to guide the robot from start to goal while avoiding obstacles.

2.1.1 Attractive Potential

Guides the robot toward the goal position:

```
U_att(q) = γ_att × d(q, q_goal)
```

where:

- `γ_att` is the attractive gain coefficient
- `d(q, q_goal)` represents the Euclidean distance from position q to the goal

2.1.2 Repulsive Potential

Pushes the robot away from obstacles:

```
U_rep(q) = {
    γ_rep × (1/d(q, q_obs) - 1/ρ₀)²   if d(q, q_obs) ≤ ρ₀
    0                                    otherwise
}
```

where:

- `γ_rep` is the repulsive gain
- `d(q, q_obs)` is the distance to the nearest obstacle
- `ρ₀` is the obstacle influence radius

2.1.3 Combined Potential

The total navigation potential is the superposition:

```
U(q) = U_att(q) + U_rep(q)
```

2.2 Robot Geometry Handling

To accommodate rectangular robots in point-mass planning, the system implements obstacle inflation.

Algorithm: Obstacle Inflation for Robot Footprint

```
Input: Grid G, robot dimensions (w, h)
Output: Expanded grid G'

1. G' ← copy(G)
2. m_v ← (h−1) ÷ 2          // Vertical margin
3. m_h ← (w−1) ÷ 2          // Horizontal margin
4. For each obstacle (r_o, c_o) in G:
5.     For r_offset in [−m_v, m_v]:
6.         For c_offset in [−m_h, m_h]:
7.             (r_t, c_t) ← (r_o + r_offset, c_o + c_offset)
8.             If (r_t, c_t) is within bounds and free:
9.                 G'[r_t][c_t] ← OBSTACLE
10. Return G'
```

This ensures that any path found in the inflated grid is collision-free for the actual robot dimensions.

2.3 Path Extraction Algorithm

The system employs a two-stage path extraction approach:

2.3.1 Primary Method: A* Search

A* algorithm uses the potential field as a heuristic function:

- **Cost function**: `f(n) = g(n) + h(n)`
- **Accumulated cost**: `g(n)` is the path cost from start to node n
- **Heuristic**: `h(n) = U(n)` (potential field value)
- **Edge costs**: Euclidean distance (1.0 for cardinal moves, √2 for diagonal)

2.3.2 Fallback Method: Gradient Descent

If A* fails, steepest gradient descent on the potential field:

- Select neighbor with minimum potential value
- Cycle detection using circular buffer (size 20)
- Maximum step limit: 2 × grid_area

---

3. Implementation

3.1 System Architecture

The implementation follows a modular architecture organized into functional components:

```
micronavigator/
├── config/            # Configuration parameters
│   └── settings.py
├── map/               # Grid maps and scenarios
```

```
│     ├── grid_loader.py
│     └── scenario*.txt
├── planner/          # Core planning algorithms
│     ├── potential_field.py
│     ├── path_extractor.py
│     └── statistics.py
├── robot/            # Robot geometry handling
│     ├── shape_handler.py
│     └── exporter.py
├── visualization/    # Rendering modules
│     ├── draw_map.py
│     └── draw_path.py
└── evaluation/       # Performance assessment
      └── evaluator.py
```

3.2 Configuration Parameters

The system supports tunable parameters in `config/settings.py`:

```
# Potential Field Parameters
ATTRACTIVE_GAIN = 1.0       # Goal attraction coefficient
REPULSIVE_GAIN = 50.0       # Obstacle repulsion coefficient
OBSTACLE_INFLUENCE = 3      # Repulsive field radius

# Robot Configuration
ROBOT_WIDTH = 2             # Horizontal dimension (cells)
ROBOT_HEIGHT = 2            # Vertical dimension (cells)
```

3.3 Core Algorithm Implementation

3.3.1 Potential Field Computation

The `compute_potential_field()` function generates the navigation potential by iterating over all grid cells and computing the combined attractive and repulsive potentials. Obstacle cells are assigned infinite potential to ensure collision avoidance.

Key Implementation Details:

- Grid traversal: O(n^2) where n is grid dimension
- Distance calculation: Euclidean norm for all cells
- Boundary handling: Infinite potential for obstacles

3.3.2 A* Pathfinding

The `compute_astar_path()` function implements priority queue-based A* search:

- Uses Python's `heapq` for efficient min-heap operations
- Maintains a cost map to track optimal paths to each position
- Explores 8-connected grid neighbors (cardinal and diagonal directions)

- Returns trajectory and node count upon goal discovery

Data Structures:

- Priority queue: `(total_cost, accumulated_cost, position, trajectory)`
- Cost map: Dictionary mapping positions to optimal costs
- Iteration limit: `grid_rows × grid_cols × 4` for safety

3.3.3 Statistics Tracking

The `PlanningStatistics` class captures:

- **Timing**: Planning time in milliseconds
- **Search metrics**: Nodes explored during search
- **Path quality**: Path length (waypoints), path cost (distance)
- **Map data**: Size, obstacles, robot dimensions
- **Status**: Success/failure with detailed reasons

---

4. Experimental Evaluation

4.1 Test Scenarios

The evaluation system tests six distinct map configurations plus robot size variations:

Scenario	Characteristics	Robot Size
Scenario 1	Simple open environment	1×1
Scenario 2	Narrow corridor navigation	1×1
Scenario 3	Maze with multiple turns	1×1, 2×2
Scenario 4	Cluttered workspace	1×1, 2×2
Scenario 5	Tight clearances	1×1
Scenario 6	Large-scale map	1×1

4.2 Performance Metrics

For each scenario, the following metrics are recorded:

- **Success Rate**: Percentage of scenarios reaching the goal
- **Planning Time**: Computational time in milliseconds
- **Nodes Explored**: Search space efficiency
- **Path Length**: Number of waypoints in trajectory
- **Path Cost**: Total Euclidean distance traveled

4.3 Results

The evaluation system generates:

- **Comparison Table**: Side-by-side performance across scenarios
- **Summary Statistics**: Aggregate metrics (averages, success rate)
- **Visualizations**: Path overlays for each test case
- **Export Files**: JSON and CSV formats for further analysis

Example Output from Successful Planning:

```
Status: SUCCESS
Planning Time: 0.13 ms
Nodes Explored: 13
Path Length: 9 steps
Path Cost: 9.66 units
Average Step Cost: 1.07 units
```

The system demonstrates consistent performance across scenarios, with planning times under 1 millisecond for typical grid sizes (5×9 to 20×30 cells).

4.4 Algorithm Analysis

Time Complexity

- **Potential Field**: $O(n^2 \times m)$ where n is grid dimension and m is obstacle count
- *A Search**: O(b^d log b^d) where b is branching factor (8) and d is path length
- **Obstacle Inflation**: $O(m \times w \times h)$ for m obstacles and robot size (w,h)

Space Complexity

- Grid storage: $O(n^2)$
- Potential field: $O(n^2)$
- A* data structures: $O(n^2)$ worst case

---

5. Usage and Deployment

5.1 Environment Setup

The system requires Python 3.11+ with matplotlib. A dedicated conda environment is recommended:

```
conda create -n micronavigator python=3.11
conda activate micronavigator
pip install -r requirements.txt
```

5.2 Running the System

Single Scenario Execution

```
python main.py
```

Outputs:

- `map_output.png` - Occupancy grid visualization
- `path_output.png` - Path overlay on grid
- `robot/path_output.csv` - Waypoint coordinates
- Console output with planning statistics

Batch Evaluation

```
python run_evaluation.py
```

Outputs:

- `evaluation/results.json` - Detailed results
- `evaluation/results.csv` - Tabular format
- `evaluation/*_path.png` - Per-scenario visualizations
- Comparison table and summary statistics in console

5.3 Configuration Customization

Modify `config/settings.py` to adjust system behavior:

- **Robot dimensions**: Change `ROBOT_WIDTH` and `ROBOT_HEIGHT`
- **Potential field tuning**: Adjust `ATTRACTIVE_GAIN`, `REPULSIVE_GAIN`, and `OBSTACLE_INFLUENCE`
- **Visualization**: Enable/disable potential field display

---

6. Conclusion

6.1 Summary

Micro-Navigator successfully implements a robust path planning system combining potential field theory with A* search. The system effectively handles:

- Static obstacle avoidance in grid environments
- Variable rectangular robot geometries
- Efficient path generation with sub-millisecond planning times
- Comprehensive performance evaluation and statistics

6.2 Key Achievements

- **Modular Architecture**: Clean separation of concerns enables maintainability
- **Hybrid Approach**: Combines strengths of potential fields and A* search
- **Geometric Awareness**: Proper handling of robot dimensions through inflation

- **Evaluation Framework**: Systematic testing across diverse scenarios
- **Professional Documentation**: Comprehensive README and inline code documentation

6.3 Potential Extensions

Future enhancements could include:

- Dynamic obstacle handling with replanning
- Multi-robot coordination and collision avoidance
- 3D environment navigation
- Real-time visualization during planning
- Integration with robot operating system (ROS)
- Path smoothing and trajectory optimization
- Alternative heuristics (Euclidean, Manhattan, Chebyshev)

6.4 Applicability

The system is suitable for:

- Educational purposes in robotics and AI courses
- Prototyping navigation algorithms for mobile robots
- Warehouse automation path planning
- Grid-based game AI navigation
- Benchmarking and algorithm comparison studies

References

The complete source code, documentation, and evaluation results are available in the project repository. The system demonstrates that hybrid approaches combining classical potential field methods with modern search algorithms can achieve both efficiency and reliability in autonomous navigation tasks.

End of Documentation