Path Planning for Point and Disk Robots

Aniruddh Kumar Sharma

April 2025

1 Introduction

This report explains an implementation of path planning algorithms for navigating through environments with polygon obstacles. The solution uses visibility graphs and Dijkstra's algorithm to find the shortest path from a start point to an end point. Two versions are presented: one for a point robot (zero radius) and another for a unit disk robot (with non-zero radius).

2 Problem Formulation

Given:

- A 2D environment with polygon obstacles
- Start point and goal point
- Robot (either a point or a disk with radius)

The task is to find the shortest collision-free path from the start to the goal.

3 Data Structures

The implementation uses the following data structures:

- `Point` = `Tuple[float, float]` - Represents a 2D point
- `Polygon` = `List[Point]` - Represents a polygon as a list of vertices
- `VisibilityGraph` = `Dict[int, Dict[int, float]]` - Adjacency list with distances
- `PathResult` = `Tuple[List[Point], float]` - The resulting path and its length

4 Visibility Graph Approach

Both implementations use the visibility graph approach:

1. Collect all vertices of obstacles along with start and goal points
2. Create a visibility graph where:
   - Nodes are the vertices of obstacles plus start and goal points
   - Edges connect vertices that can "see" each other (line of sight not blocked by any obstacle)
   - Edge weights are Euclidean distances between vertices
3. Apply Dijkstra's algorithm to find the shortest path in the visibility graph

5 Core Components

5.1 Line Intersection Detection

A critical component is determining whether two line segments intersect. The implementations use orientation-based tests:

- `check_direction`/`get_turn`: Computes the orientation of three points (clockwise, counterclockwise, or collinear)
- `is_point_on_line`/`is_between`: Checks if a point lies on a line segment
- `do_lines_cross`/`lines_cross`: Determines if two line segments intersect

The intersection test uses the principle that two line segments intersect if:

- The endpoints of one segment are on opposite sides of the other segment, and
- The endpoints of the second segment are on opposite sides of the first segment

5.2 Visibility Testing

The function `can_points_see_each_other`/`can_see` determines if two points have a clear line of sight:

- If the points are part of the same obstacle edge, they can see each other
- Otherwise, the function checks if the line segment connecting the points intersects with any obstacle edge
- Special handling is needed for points that are vertices of obstacles

5.3 Visibility Graph Construction

The visibility graph is built by:

1. Initializing an empty graph with all vertices as nodes
2. For each pair of vertices, checking if they can see each other
3. If they can see each other, adding an edge with weight equal to the Euclidean distance

5.4 Shortest Path Finding

Dijkstra's algorithm is used to find the shortest path in the visibility graph:

1. Initialize distance values: 0 for start vertex, infinity for all others
2. Use a priority queue to process vertices in order of increasing distance
3. For each vertex, update distances to its neighbors if a shorter path is found
4. Keep track of the predecessor of each vertex to reconstruct the path
5. When the goal is reached, backtrack to construct the complete path

6 Unit Disk Robot Extension

The unit disk robot implementation extends the point robot solution with one key difference: obstacle inflation.

6.1 Obstacle Inflation

Since the robot has a non-zero radius, the obstacles must be "inflated" by the radius of the robot:

1. For each edge of a polygon, calculate the outward normal vector
2. Move each vertex along the normal direction by the robot's radius
3. This creates a larger polygon that the center of the robot must avoid

The function `make_bigger` implements this inflation:

- It computes the perpendicular direction to each edge
- It moves each vertex outward along this direction by the specified distance
- The result is a larger polygon that accommodates the robot's radius

Once the obstacles are inflated, the problem reduces to finding a path for a point robot (the center of the disk) through the environment with inflated obstacles.

7 Implementation Details

7.1 Initialization

Both implementations begin by:

- Storing obstacles, start point, and end point
- Collecting all vertices (start, end, and obstacle vertices)
- Assigning unique indices to all vertices
- Identifying and storing obstacle edges

7.2 Numerical Stability

The implementations handle numerical precision issues:

- Using small epsilon values (e.g., 1e-10) for floating-point comparisons
- Special handling for collinear points and edge cases

7.3 Edge Handling

Special care is taken when handling:

- Points that are vertices of obstacles
- Line segments that are edges of obstacles
- Collinear points and other geometric edge cases

7.4 Path Reconstruction

After running Dijkstra's algorithm:

- The path is reconstructed by backtracking from the end point using the predecessor information
- The result includes both the sequence of points and the total path length
- If no path exists, an empty list and infinity are returned

8 Algorithm Complexity

8.1 Time Complexity

Let n be the total number of vertices (including start, end, and all obstacle vertices):

- Visibility graph construction: $O(n^3)$
  - $O(n^2)$ pairs of vertices to check
  - $O(n)$ time to check visibility for each pair (testing intersection with all obstacle edges)
- Dijkstra's algorithm: $O(n^2)$ without a priority queue, $O(n \log n + E)$ with a priority queue (where E is the number of edges in the visibility graph)
- Total: $O(n^3)$

8.2 Space Complexity

- Vertices and indices: $O(n)$
- Visibility graph: $O(n^2)$ in the worst case
- Dijkstra's algorithm data structures: $O(n)$
- Total: $O(n^2)$

9 Conclusion

The implementations provide efficient solutions for finding shortest paths in environments with polygon obstacles. The visibility graph approach is well-suited for this problem as it reduces the continuous space to a discrete graph where optimal solutions can be found.

Key aspects of the solution:

- Efficient geometric algorithms for visibility testing

- Systematic construction of the visibility graph
- Application of Dijkstra's algorithm for shortest path finding
- Extension to handle robots with non-zero radius through obstacle inflation

The solution could be further improved by:

- Using more efficient algorithms for visibility graph construction
- Implementing approximate solutions for very large environments
- Incorporating dynamic obstacle handling
- Adding path smoothing for more natural robot movement

10 Algorithm Complexity

10.1 Time Complexity

Let n be the total number of vertices (including start, end, and all obstacle vertices), and m be the number of obstacles:

- **Initialization**: $O(n)$ - Collecting all vertices and assigning indices
- **Visibility testing between two points**: $O(e)$ - where e is the total number of obstacle edges
  - Since each polygon with k vertices has k edges, and we have m obstacles, $e = O(n)$ in the worst case
  - Each visibility test requires checking for intersection with every obstacle edge
- **Visibility graph construction**: $O(n^2 \cdot e) = O(n^3)$
  - We check visibility for all $O(n^2)$ pairs of vertices
  - Each visibility check takes $O(e) = O(n)$ time
- **Dijkstra's algorithm**: $O(E \log V)$ with a binary heap priority queue
  - $V = n$ is the number of vertices
  - $E = O(n^2)$ is the maximum number of edges in the visibility graph
  - This gives $O(n^2 \log n)$ in the worst case
- **Path reconstruction**: $O(n)$ - Backtracking from end to start
- **Obstacle inflation** (for unit disk robot): $O(n)$ - Processing each vertex once

Overall time complexity: $O(n^3)$, dominated by the visibility graph construction.

10.2 Space Complexity

- **Vertices storage**: $O(n)$ - Storing all points and their indices
- **Obstacle edges storage**: $O(e) = O(n)$ - Storing all edges of obstacles
- **Visibility graph**: $O(n^2)$ - In the worst case, every vertex can see every other vertex
- **Dijkstra's algorithm data structures**:
  - Distance map: $O(n)$
  - Previous node map: $O(n)$
  - Priority queue: $O(n)$
- **Inflated obstacles** (for unit disk robot): $O(n)$ - Same size as original obstacles

Overall space complexity: $O(n^2)$, dominated by the visibility graph storage.