PRM Path Planning Simulation Project

Okan Berhoğlu
2739050

December 14, 2025

1 Abstract

This project presents the implementation and evaluation of the Probabilistic Roadmap (PRM) path planning algorithm for a PRRR (Prismatic-Revolute-Revolute-Revolute) planar robot arm operating in environments with polygonal obstacles. The PRM algorithm is implemented in this project. The implementation includes a custom configuration distance metric that accounts for the hybrid nature of the configuration space, a randomized inverse kinematics solver for goal configuration generation, and comprehensive collision detection for irregular polygonal obstacles. The system is tested with various goal points and obstacle configurations, demonstrating successful path planning capabilities. Results show that the PRM approach effectively handles the high-dimensional configuration space and produces collision-free paths in cluttered environments.

2 Introduction

Path planning is a fundamental problem in robotics, where the goal is to find a collision-free path for a robot to move from an initial configuration to a goal configuration while avoiding obstacles in the environment. The Probabilistic Roadmap (PRM) algorithm is a widely used sampling-based motion planning method that efficiently solves this problem for robots with multiple degrees of freedom operating in complex environments.

This project implements the PRM path planning algorithm for a PRRR (Prismatic-Revolute-Revolute-Revolute) planar robot arm with 4 degrees of freedom. The robot consists of one prismatic joint that allows vertical translation along the x axis and three revolute joints that provide rotational motion.

The PRM algorithm operates in two distinct phases. In the learning phase, a roadmap of collision-free configurations is constructed by randomly sampling the configuration space and connecting nearby configurations with collision-free paths. In the query phase, this precomputed roadmap is used to efficiently solve individual path planning queries by connecting the start and goal configurations to the roadmap and finding the shortest path through the graph structure.

This document is organized as follows. Section 2 describes the implementation method, including the configuration space representation, roadmap construction algorithm, sampling strategy, configuration distance metric, and the query phase with Dijkstra's shortest path algorithm. Section 3 presents the results and visualizations of the implemented PRM planner. Finally, conclusion and references are provided.

3 Implementation Method

In this project, Python programming language is used to implement PRM path planning algorithm. And by using matplotlib library, the results are visualized. Besides numpy library is used for mathematical operations. The PRM planner is implemented to a PRRR planar robot arm. The robot has 4 degrees of freedom. The configuration space of the robot is defined as $\mathbb{R} \times S^1 \times S^1 \times S^1$.

The configuration space is implemented in the `Planner` class with the following bounds:

- $q_1 \in [-1.0, 1.0]$ meters: The prismatic joint (P) that allows linear translation along the vertical axis
- $q_2 \in [-\pi, \pi]$ radians: The first revolute joint (R) rotation angle
- $q_3 \in [-\pi, \pi]$ radians: The second revolute joint (R) rotation angle
- $q_4 \in [-\pi, \pi]$ radians: The third revolute joint (R) rotation angle

The prismatic joint q_1 represents a linear space ($\mathbb{R}$), while the three revolute joints q_2, q_3, q_4 each represent a circular space (S^1) due to their periodic nature. This configuration space structure is crucial for proper distance calculations and interpolation between configurations, as angular differences must account for the wraparound at $\pm\pi$.

PRM divides planning into two phases: the learing phase, during each a roadmap in free space which does not contain any obstacles is built; and the query phase, during which user-defined query configurations are connected with the precomputed roadmap. The nodes of the roadmap are configurations in the free space and the edges are collision-free paths between the nodes. In the implementation phase firstly the roadmap is constructed. Secondly the query phase is implemented.

3.1 Roadmap Construction

The roadmap is represented as a graph where nodes are configurations in the free space and edges are collision-free paths between the nodes. In the roadmap construction phase, a number of random configurations are sampled and checked for collisions. If a configuration is collision-free, it is added to the roadmap as a node. The given algorithm [1] in the below is used for roadmap construction.

Algorithm 1: Roadmap Construction Algorithm

```
Input: n: number of nodes to put in the roadmap
Input: k: number of closest neighbors to examine for each configuration
Output: A roadmap G = (V, E)
V ← ∅;
E ← ∅;
while |V| < n do
    repeat
        q ← a random configuration in configuration space;
    until q is collision-free;
    V ← V ∪ {q};
for all q ∈ V do
    N_q ← the k closest neighbors of q chosen from V according to dist;
    for all q' ∈ N_q do
        if (q, q') ∉ E then
            E ← E ∪ {(q, q')};
```

The G is the undirected graph and the V and E are the set of nodes and edges, respectively. Initially, the graph G is empty. Then, for each configuration q in V, the algorithm finds the k closest neighbors of q in V according to the distance function dist. For each neighbor q' of q, if the edge (q, q') is not in E and the path between q and q' is collision-free, then the edge (q, q') is added to E. The q, q' are the start and end states of the path, respectively. The dist is the distance function between two configurations. The process is repeated until n collision-free configurations have been sampled.

While calculating the random configurations in the configuration space, uniform sampling method is used.

Algorithm 2: Uniform Random Configuration Sampling

Input: q_1^{min}, q_1^{max}: bounds for prismatic joint
Input: $q_2^{min}, q_2^{max}, q_3^{min}, q_3^{max}, q_4^{min}, q_4^{max}$: bounds for revolute joints
Output: A random configuration $q = [q_1, q_2, q_3, q_4]$
$q_1 \leftarrow$ uniform_random(q_1^{min}, q_1^{max});
$q_2 \leftarrow$ uniform_random(q_2^{min}, q_2^{max});
$q_3 \leftarrow$ uniform_random(q_3^{min}, q_3^{max});
$q_4 \leftarrow$ uniform_random(q_4^{min}, q_4^{max});
return $q = [q_1, q_2, q_3, q_4]$;

In this implementation, the configuration space bounds are:

- $q_1 \in [-1.0, 1.0]$ meters (prismatic joint)
- $q_2, q_3, q_4 \in [-\pi, \pi]$ radians (revolute joints)

Each joint parameter is sampled independently using a uniform distribution over its valid range. This ensures unbiased exploration of the configuration space.

While calculating the closest neighbors of a configuration, the distance function is used. The distance metric must account for the hybrid nature of the configuration space, where q_1 is a linear coordinate and q_2, q_3, q_4 are angular coordinates on a circle.

For two configurations $q = [q_1, q_2, q_3, q_4]$ and $q' = [q'_1, q'_2, q'_3, q'_4]$, the distance is computed as:

$$d(q, q') = \sqrt{d_1^2 + d_2^2 + d_3^2 + d_4^2} \tag{1}$$

where:

$$d_1 = |q_1 - q'_1| \tag{2}$$

$$d_i = \min(|q_i - q'_i|, 2\pi - |q_i - q'_i|) \quad \text{for } i \in \{2, 3, 4\} \tag{3}$$

The key difference is that for revolute joints, the angular distance accounts for the wraparound at $\pm\pi$. For example, the distance between $-\pi$ and π is 0, not 2π, since they represent the same angular position.

This distance metric ensures that the planner correctly identifies nearby configurations in the configuration space, which is essential for building meaningful edges in the roadmap.

After the distance metric is calculated, the closest neighbors of each nodes are found. To determine the edges of the roadmap, the algorithm checks if the path between the nodes is collision-free. If it is, the edge is added to the roadmap.

3.2 Query Phase

In the query phase, the roadmap is used to solve individual path planning problems. Given initial and goal configurations, the algorithm finds a collision-free path between them using the roadmap. If successful, the path is returned as the solution.

The main question is how to connect the initial and goal configurations to the roadmap. The solution involves two steps; first, connecting the start and goal configurations to the roadmap, and second, finding the shortest path through the roadmap using Dijkstra's algorithm.

Both the initial configuration q_{start} and goal configuration q_{goal} are connected to the roadmap by finding their k nearest neighbors and attempting to create collision-free edges. The start configuration is added to the roadmap during initialization, while the goal configuration is added during the query phase.

Algorithm 3: Connect Configuration to Roadmap

Input: q: configuration to connect
Input: V: set of roadmap nodes
Input: E: set of roadmap edges
Input: k: number of neighbors to consider
Output: Updated roadmap with q connected

```
Add q to V;
neighbors ← empty list;
for each node v ∈ V (excluding q) do
    d ← distance(q, v);
    Add (d, v) to neighbors;
Sort neighbors by distance;
for each of the k closest neighbors (d, v) do
    if edge between q and v is collision-free then
        Add edge (q, v) to E;
```

Once both start and goal configurations are connected to the roadmap, Dijkstra's algorithm is used to find the shortest collision-free path through the roadmap graph.

The algorithm maintains a priority queue ordered by cumulative distance from the start node. It explores nodes in order of increasing distance, guaranteeing that the first path found to the goal is the shortest.

3.3 Obstacle Generation

To test the PRM planner in realistic scenarios, random irregular polygonal obstacles are generated in the workspace. The obstacle generation process creates non-overlapping polygons that avoid both the robot's starting configuration and user-specified goal points.

Each obstacle is an irregular polygon with a random number of vertices (typically 3 to 6) positioned around a randomly selected center point (c_x, c_y). The polygon vertices are generated using polar coordinates by first generating n random angles uniformly distributed in $[0, 2\pi]$ and sorting them, then for each angle θ_i, generating a random radius $r_i \in [r_{min}, r_{max}]$, and finally computing the vertex coordinates as $x_i = c_x + r_i \cos(\theta_i)$ and $y_i = c_y + r_i \sin(\theta_i)$.

This approach creates irregular, star-like polygons with varying edge lengths and angles, providing a diverse set of obstacle shapes for testing.

The obstacle generation process includes several collision checks to ensure valid obstacle placement. Obstacles must not collide with the robot's initial configuration, must not contain user-specified goal points, and must not overlap with previously generated obstacles. Collision detection is performed using three geometric algorithms: segment intersection checks if two line segments intersect using cross products, point-in-polygon test uses ray casting algorithm to determine if a point lies inside a polygon, and polygon-polygon collision combines edge intersection tests and point containment checks. The generation process uses rejection sampling with a maximum number of attempts to ensure termination even in crowded workspaces. If a randomly generated polygon fails any collision check, it is discarded and a new polygon is generated.

3.4 Inverse Kinematics for Goal Configuration

To connect the goal point in workspace to a configuration in the configuration space, an inverse kinematics (IK) solution is required. Since analytical IK solutions for the PRRR robot can be complex, a randomized sampling-based approach is used. The algorithm randomly samples configurations from the configuration space, computes the forward kinematics to determine the end-effector position, and selects the collision-free configuration that minimizes the distance to the target goal point. The process continues for a maximum number of attempts or until a configuration is found within a specified tolerance.

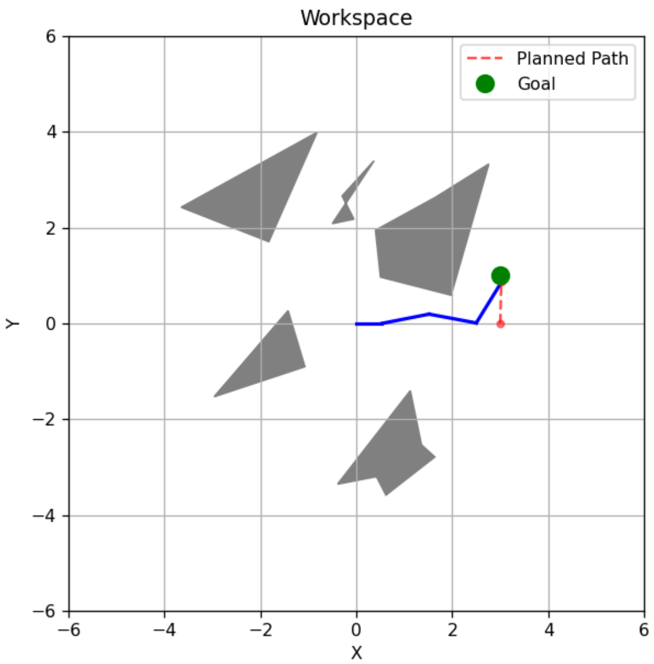

Figure 1: The workspace with the robot and the goal point.

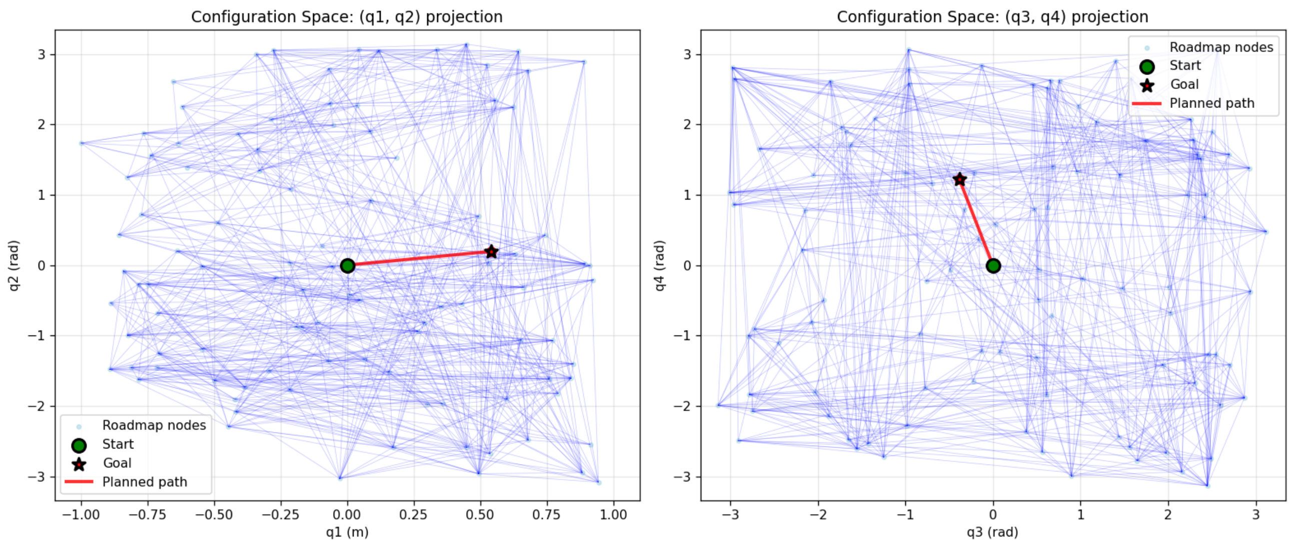

Figure 2: The roadmap generated by the algorithm in the configuration space.

4 Results

After the PRM algorithm is implemented, it is tested with different parameters. The results are given in this section.

First, the algorithm is tested with an easily reachable goal point for the robot. The figures given as figure 1 and figure 2 shows the result of this test. The first figure shows the workspace with the robot and the goal point. The second figure shows the roadmap generated by the algorithm in the configuration space.

Since the configuration space is 4 dimensional, the visualization of the roadmap is seperated as q1, q2 and q3, q4 links visualization.

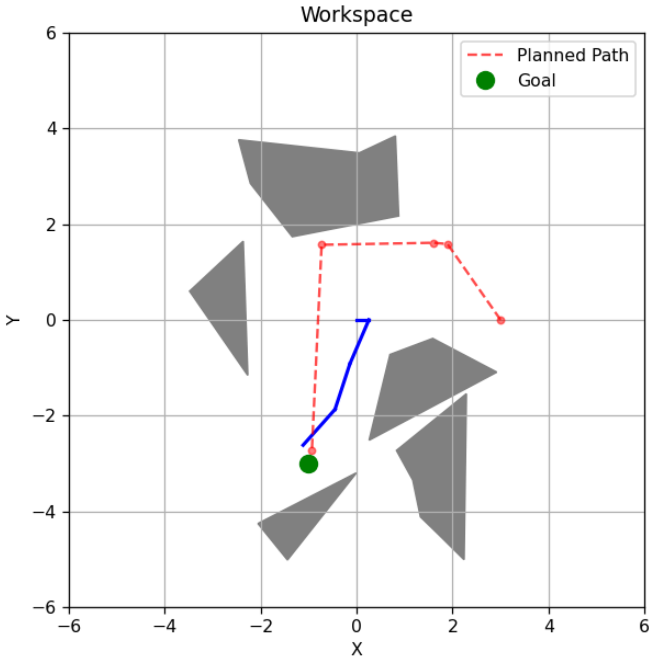

Figure 3: The workspace with the robot and the goal point.

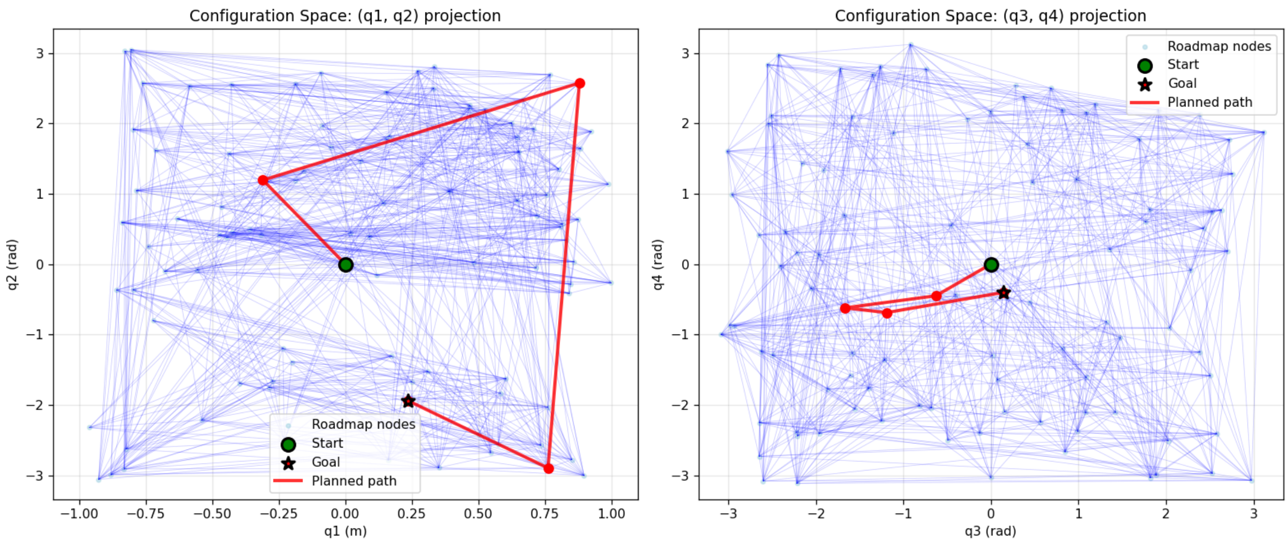

Figure 4: The roadmap generated by the algorithm in the configuration space.

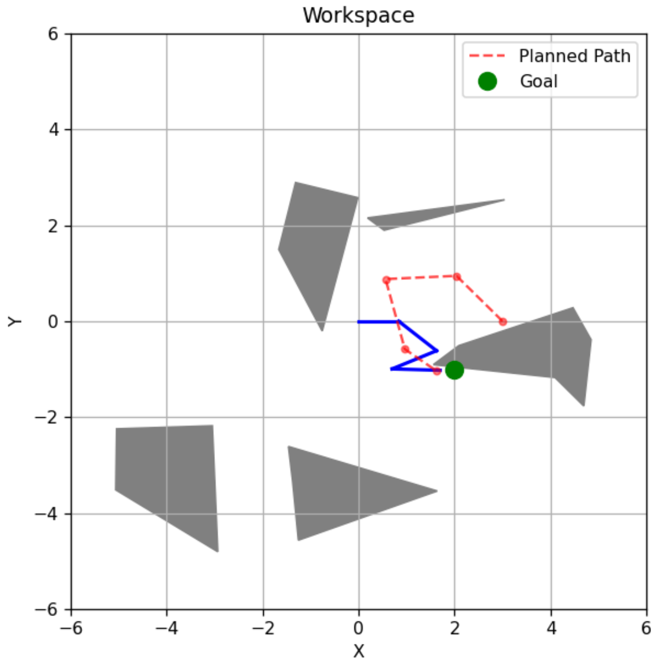

Figure 5: The workspace with the robot and the goal point.

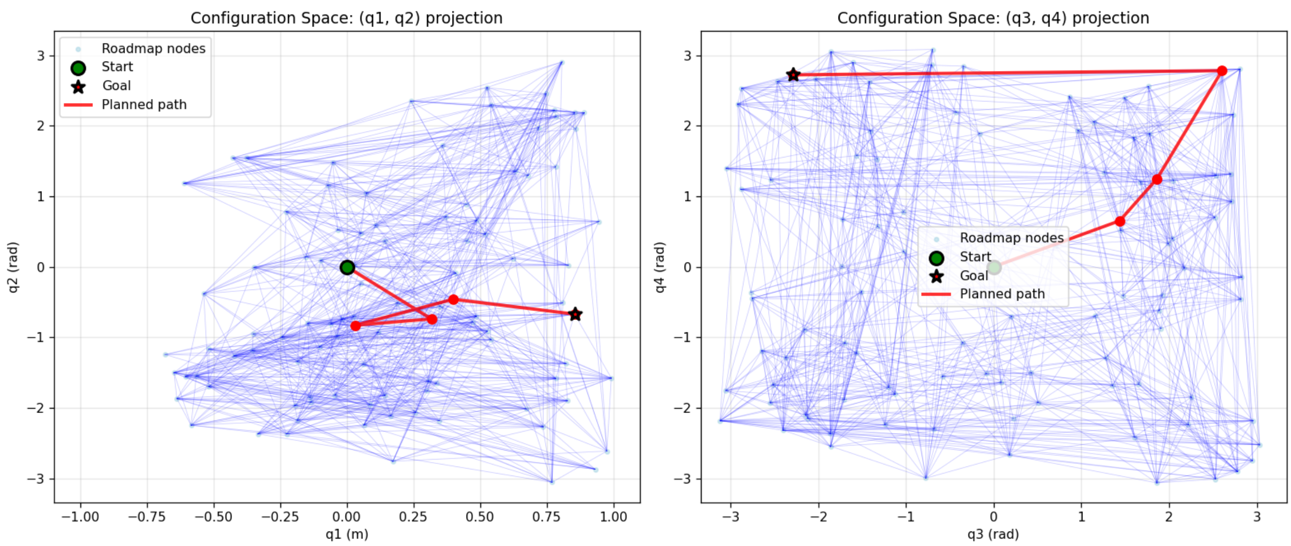

Figure 6: The roadmap generated by the algorithm in the configuration space.

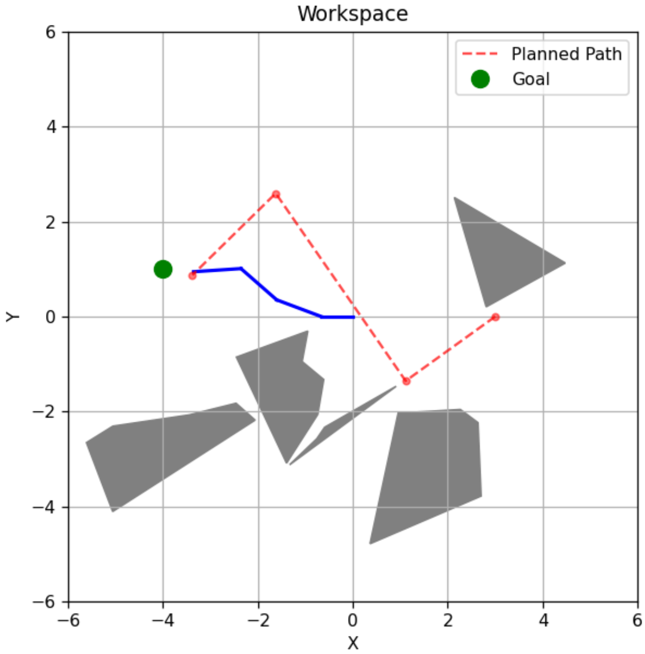

Figure 7: The workspace with the robot and the goal point.

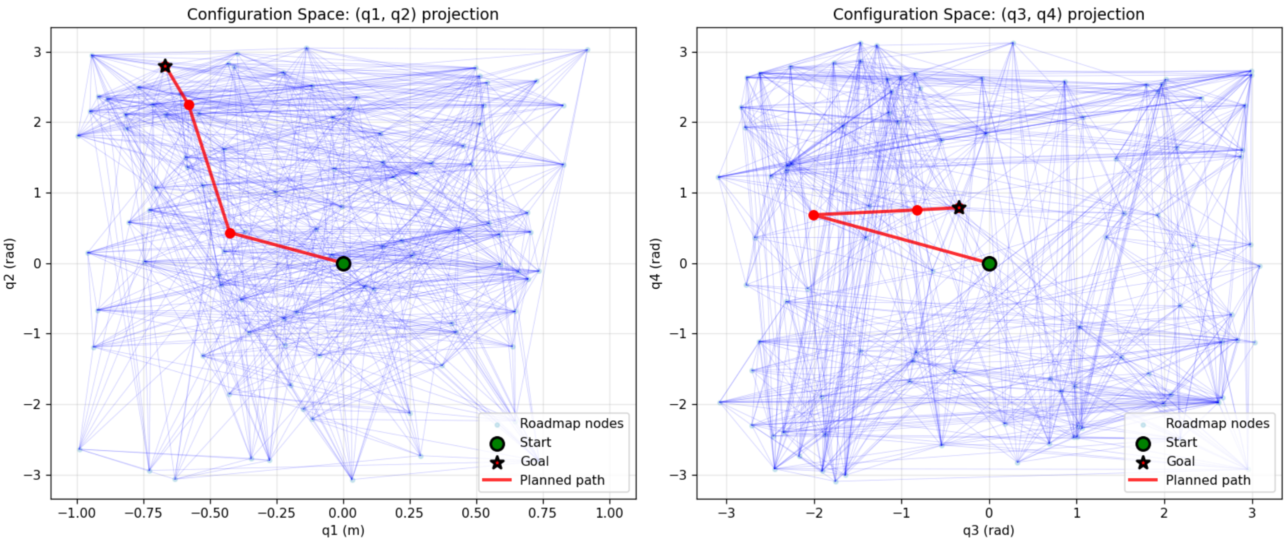

Figure 8: The roadmap generated by the algorithm in the configuration space.

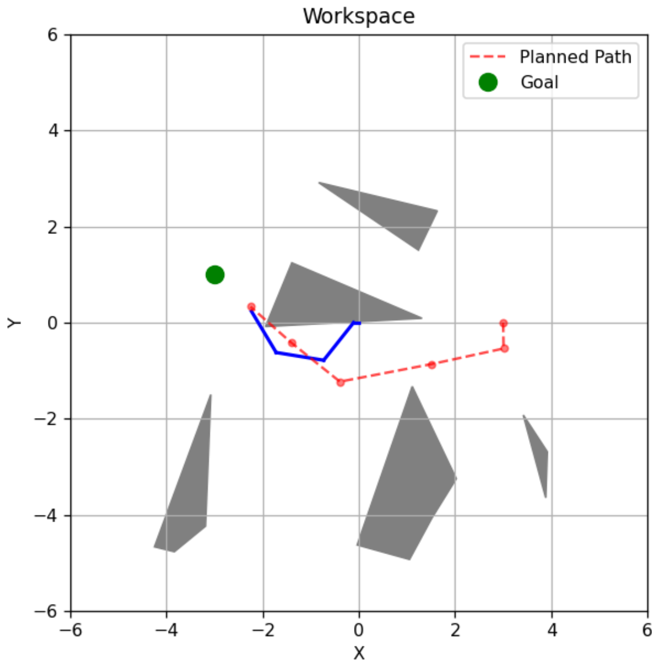

Figure 9: The workspace with the robot and the goal point.

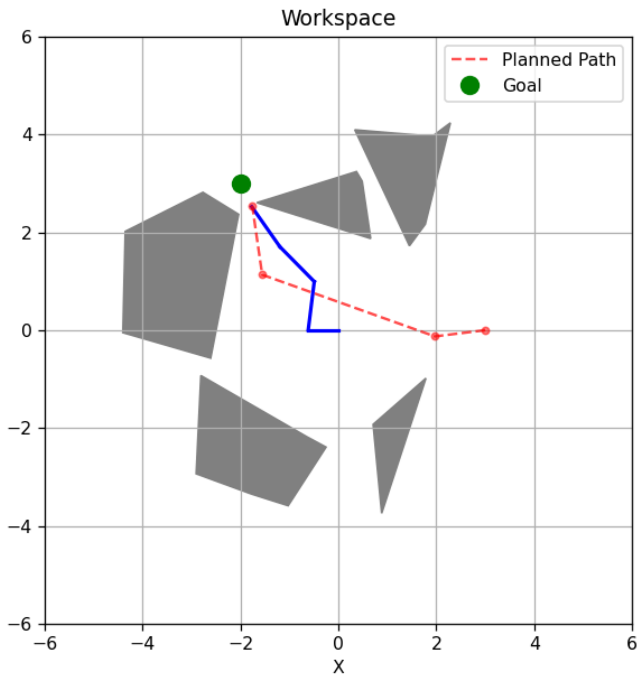

Figure 10: The workspace with the robot and the goal point.

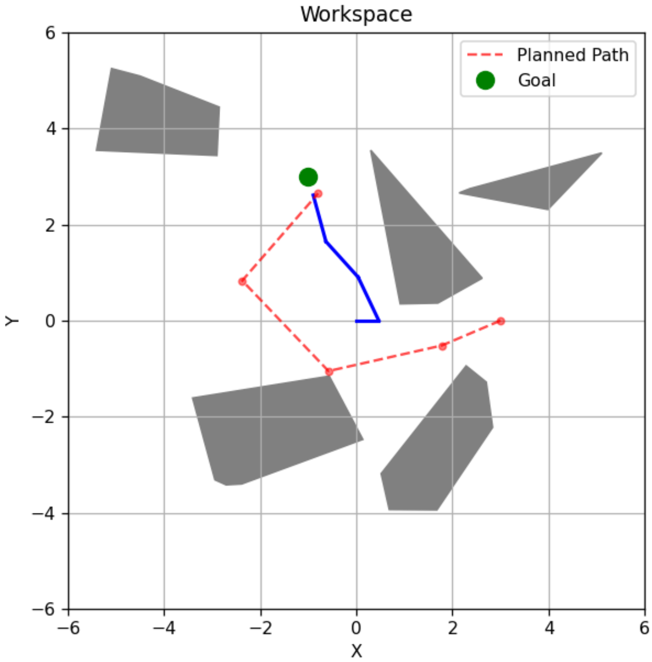

Figure 11: The workspace with the robot and the goal point.

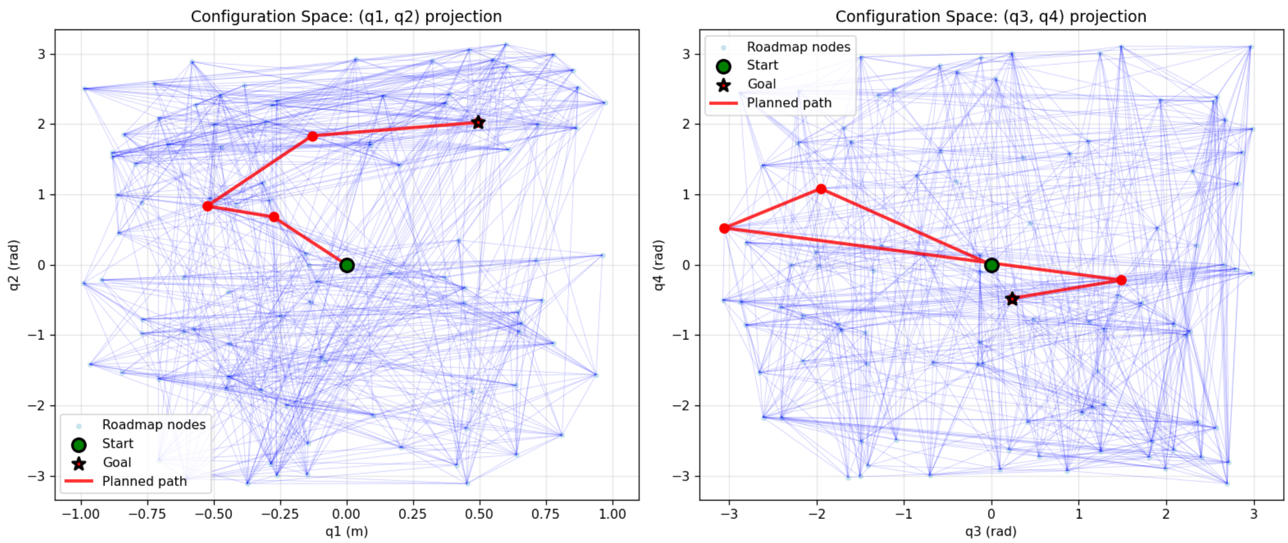

Figure 12: The roadmap generated by the algorithm in the configuration space.

5 Conclusion

This project successfully implemented and evaluated the Probabilistic Roadmap (PRM) path planning algorithm for a 4-degree-of-freedom PRRR planar robot arm. The implementation demonstrates the effectiveness of sampling-based motion planning approaches for robots with hybrid configuration spaces operating in environments with complex polygonal obstacles.

The key achievements of this project include the development of a complete PRM planning system with several critical components. A custom configuration distance metric was implemented to properly handle the hybrid nature of the configuration space, where the prismatic joint occupies a linear space and the revolute joints occupy circular spaces. This metric correctly accounts for angular wraparound at $\pm\pi$, ensuring accurate nearest-neighbor identification during roadmap construction. The randomized inverse kinematics solver provides a practical solution for converting workspace goal points to configuration space, trading computational efficiency for implementation simplicity while maintaining satisfactory performance for reachable targets.

The roadmap construction phase employs uniform random sampling to explore the configuration space, with collision checking ensuring that only valid configurations are added to the roadmap. The k-nearest neighbor connection strategy creates a well-connected graph structure that facilitates efficient path queries. The query phase successfully connects start and goal configurations to the precomputed roadmap and applies Dijkstra's algorithm to find the shortest collision-free path through the graph.

Testing with various goal points and obstacle configurations demonstrates the robustness of the implementation. The system successfully generates collision-free paths in cluttered environments, with the roadmap visualization clearly showing the distribution of sampled configurations and the connectivity of the graph structure in the 4-dimensional configuration space.

Future work could explore several enhancements to improve the planner's performance. Adaptive sampling strategies could focus computational effort on difficult regions of the configuration space, potentially reducing the number of samples required for adequate coverage. Path smoothing techniques could reduce unnecessary waypoints and produce more natural robot motions. Additionally, implementing lazy collision checking could defer expensive collision detection operations until a candidate path is found, improving query-time performance.

References

[1] Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., & Burgard, W. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations.* MIT Press.