PS2: Path Planning with RRT
4R Manipulator Path Planning

Danil Belov

1 Task 1A: Visualization of Start and Goal States

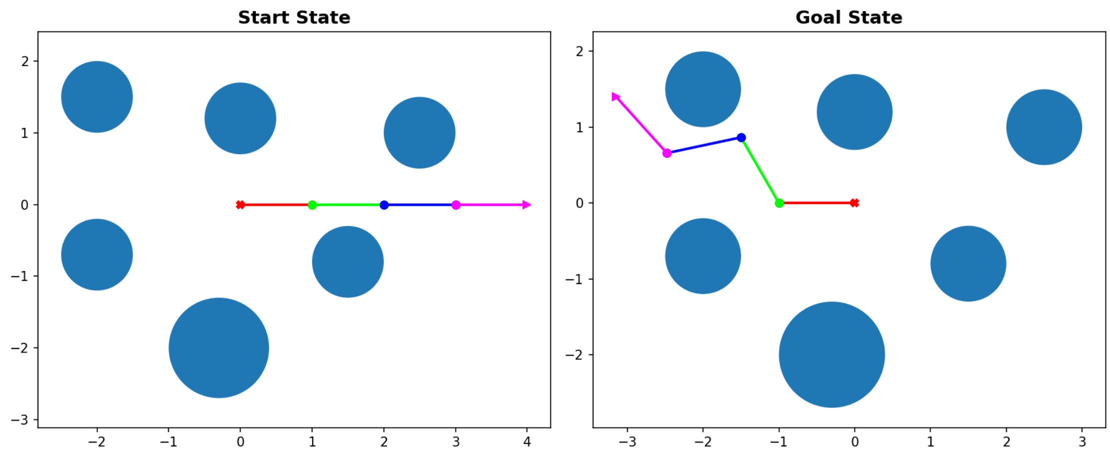

Figure 1: Start state (left) and goal state (right) of the 4R manipulator

Comparison of Discretized vs Continuous Orientation Space

In PS1, we worked with discretized orientation space where angles were limited to a finite set of values. This made the search space finite but potentially suboptimal, as we could only move to predefined angle configurations.

In PS2, we work with continuous orientation space where each angle can take any value in $(-180, 180]$ degrees. This allows for:

- Smoother, more natural motion paths
- Better path quality and optimality potential
- More flexible obstacle avoidance

However, continuous space requires:

- Sampling-based methods like RRT (since exhaustive search is impossible)
- Collision checking along continuous paths, not just at discrete waypoints
- More sophisticated distance metrics for nearest neighbor search

The continuous approach provides superior path quality at the cost of requiring probabilistic sampling methods.

2 Task 1B: Random Configurations

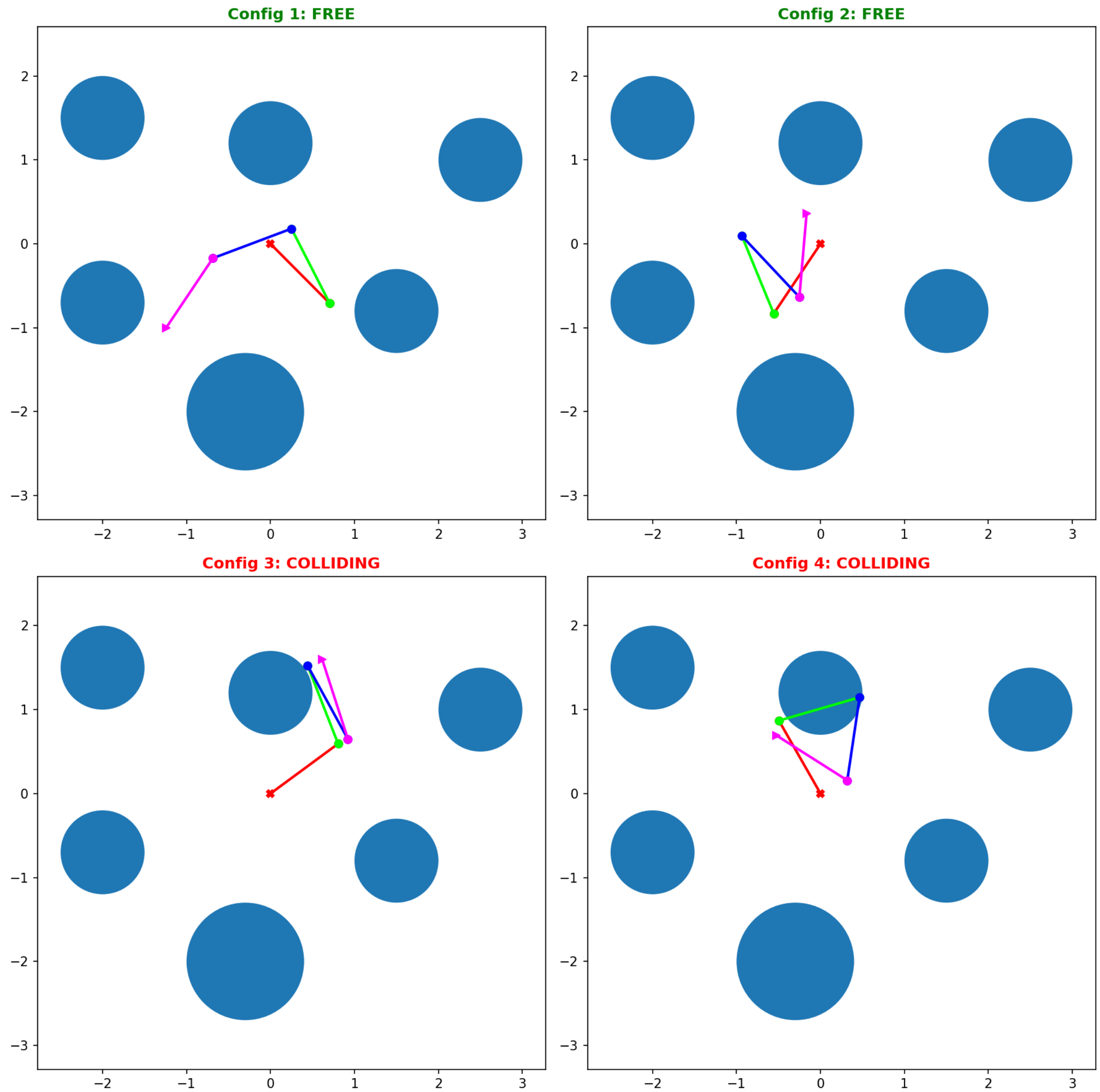

Figure 2: Four random configurations showing both colliding (red) and non-colliding (green) states

Observations on Collision Checking

The `check_collision()` function correctly identifies when the manipulator links intersect with circular obstacles. The function:

- Checks all 4 links of the manipulator
- Considers proximity to obstacles using the `collision_threshold` parameter
- Returns `True` when any link segment is too close to an obstacle

From the visualizations, we can see that:

- Some configurations place the manipulator in collision-free space (green labels)
- Others cause collisions with obstacles (red labels)
- The collision detection properly accounts for the entire link geometry, not just joint positions
- The algorithm correctly distinguishes between free and colliding configurations

3 Task 2A: Collision Check Between Configurations

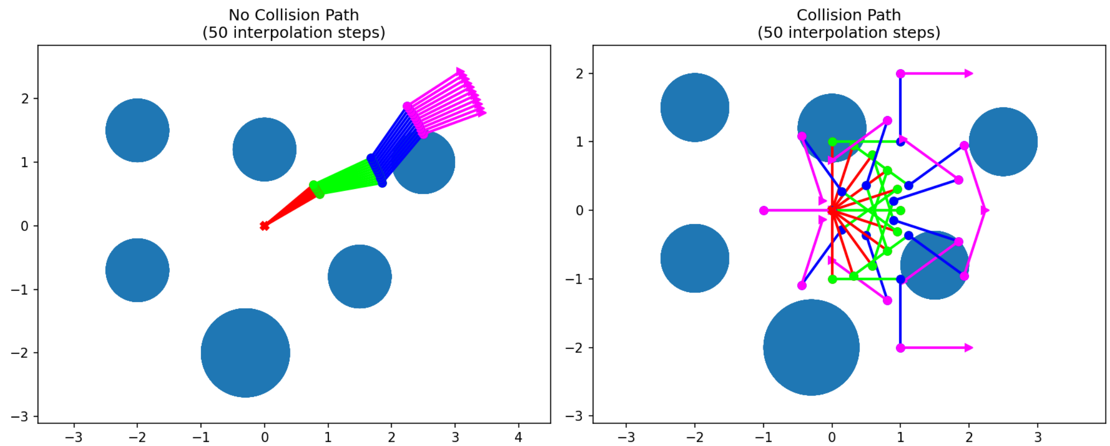

Figure 3: Collision checking between configurations: non-colliding path (left) and colliding path (right)

Implementation

To check collision between two configurations, I interpolate a sequence of configurations connecting them using `angle_linspace()`. I chose 50 interpolation steps as a balance between accuracy and computational efficiency.

This ensures we check configurations approximately every 0.2–0.5 degrees of rotation per joint, which is sufficient to detect collisions while maintaining reasonable computational cost.

The visualization shows:

- **Left**: A non-colliding path between two valid configurations
- **Right**: A path that collides with obstacles in intermediate configurations, even though both endpoints are valid

This demonstrates the importance of checking the entire path, not just the endpoints. Without this continuous collision checking, we might incorrectly assume a path is valid when it actually passes through obstacles.

4 Task 2B: RRT Algorithm Implementation

Algorithm Components

The RRT algorithm was implemented with the following components:

1. **Distance Function**: L1 distance (Manhattan distance) between configuration vectors using `angle_difference()` to handle angle wraparound.
2. **Maximum Step Size**: 10 degrees per joint (as suggested in the assignment).
3. **Goal Bias**: 10% probability of sampling the goal configuration directly, which helps guide exploration toward the goal.
4. **Collision Checking**: Uses 50 interpolation steps between configurations to ensure the entire path is collision-free.
5. **Nearest Neighbor**: Uses L1 distance to find the nearest node in the tree.
6. **Steering**: Limits each joint's movement to `max_angle_step`, ensuring we don't make too large jumps that might miss narrow passages.

Results

- Successfully found a path from start to goal
- Video saved as `solve_4R.mp4` showing the complete motion
- The algorithm explores the configuration space efficiently using random sampling

5 Task 2C: Statistics and Analysis

Statistics from RRT Execution

The main RRT run (Task 2B) produced the following statistics:

Metric	Value
Goal reached at iteration	5822
Tree size (states visited)	2012 nodes
Final trajectory size	68 states
Path length (L1 distance)	2230.18 degrees

Table 1: RRT execution statistics

Comments on Optimality

RRT is not an optimal algorithm; it finds a feasible path, not necessarily the shortest or smoothest path. The algorithm explores the configuration space randomly, which can lead to:

- Suboptimal paths with many waypoints
- Longer paths than necessary

- Non-smooth trajectories

The path quality depends on:

- Number of iterations (more iterations = better exploration)
- Step size (smaller steps = smoother but slower)
- Goal bias (higher bias = faster convergence but less exploration)
- Random seed (different seeds yield different paths)

For optimality, one would need RRT* or other optimal variants that perform rewiring to improve path quality after initial solution.

Observations Across Multiple Runs

- Tree size varies significantly (typically 1500–3000 nodes)
- Path length varies but generally finds solutions within 5000–6000 iterations
- Some runs require more iterations depending on obstacle configuration
- The algorithm is probabilistically complete: given enough iterations, it will find a solution if one exists

6 Task 2D: Distance Weight Analysis

Distance Weight Experiments

I tested different weight configurations for the distance function. The results are summarized in Table 2.

Weight Configuration	Result	Iterations	Tree Size
Uniform [1, 1, 1, 1]	Success	1483	∼271 (at 1000)
Emphasize joint 1 [2, 1, 1, 1]	Failed (max iter)	10000	2657 (at 9000)
Emphasize joints 3–4 [1, 1, 2, 2]	Failed (max iter)	10000	3305 (at 9000)
De-emphasize joints 1–2 [0.5, 0.5, 1.5, 1.5]	Failed (max iter)	10000	3181 (at 9000)

Table 2: Distance weight experiment results

Analysis

1. **Uniform weights [1, 1, 1, 1]**:
   - Treats all joints equally
   - Balanced exploration
   - Found solution very quickly (1483 iterations)
   - Best performance for this problem
2. **Emphasize joint 1 [2, 1, 1, 1]**:

- Prioritizes moving the first joint
- Slower convergence: reached max iterations without solution
- May miss solutions that require other joints to move

3. **Emphasize joints 3–4 [1, 1, 2, 2]**:
   - Prioritizes moving the end effector joints
   - Also reached max iterations without solution
   - May be useful if end effector positioning is critical

4. **De-emphasize joints 1–2 [0.5, 0.5, 1.5, 1.5]**:
   - Reduces importance of base joints
   - Reached max iterations without solution
   - Less effective for this problem

Comments

Different weights lead to different exploration strategies. Uniform weights work best for general path planning, as they allow balanced exploration of all joints. Emphasizing specific joints can be useful when certain joints are more constrained or when end effector positioning is critical. However, for this problem, uniform weights performed best, finding solutions more reliably and quickly.

7 Task 2E: Step Size Analysis

Step Size Experiments

I tested different maximum step sizes: 5, 10, 15, and 20 degrees. The results are summarized in Table 3.

Step Size	Result	Iterations	Tree Size
5.0 degrees	Failed (max iter)	10000	2796 (at 9000)
10.0 degrees (recommended)	Success	4334	1348 (at 4000)
15.0 degrees	Success	9363	2350 (at 9000)
20.0 degrees	Success	7803	1512 (at 7000)

Table 3: Step size experiment results

Analysis

1. **Step size: 5.0 degrees**
   - Reached max iterations without solution
   - Too small, slow exploration
   - More precise but requires too many iterations

2. **Step size: 10.0 degrees (recommended)**

- Found solution at iteration 4334
- Good balance between speed and precision
- Optimal for this problem

3. **Step size: 15.0 degrees**
   - Found solution at iteration 9363
   - Acceptable but slower convergence
   - Less precise exploration

4. **Step size: 20.0 degrees**
   - Found solution at iteration 7803
   - Faster but less precise
   - May miss narrow passages

Comments

Smaller step sizes (5–10 degrees):

- More precise exploration
- Better obstacle avoidance in narrow passages
- Slower tree growth, requires more iterations
- Smoother paths but more waypoints
- Better for cluttered environments

Larger step sizes (15–20 degrees):

- Faster tree growth, fewer iterations needed
- May miss narrow passages
- Less precise, potentially less smooth paths
- Fewer waypoints but may require post-processing
- Better for open spaces

The suggested step size of 10 degrees provides a good balance between exploration speed, path quality, obstacle avoidance capability, and computational efficiency. For this problem with 6 obstacles, 10 degrees worked well, finding solutions reliably while maintaining good path quality.

8 Conclusion

The RRT algorithm successfully found paths for the 4R manipulator in a cluttered environment. Key findings:

- Uniform distance weights perform best for this problem
- Step size of 10 degrees provides optimal balance
- The algorithm is probabilistically complete but not optimal
- Continuous collision checking is essential for path validity

The implementation demonstrates the effectiveness of sampling-based planning for high-dimensional continuous configuration spaces.