Motion Planning and Trajectory Generation for Autonomous Robots

Arjun Subhedar (IMT2021069)
Nitheezkant R (IMT2021508)
Pandey Shourya Prasad (IMT2021535)

CS 819 Robotics
International Institute of Information Technology Bangalore

May 9, 2025

Contents

1 Introduction

Motion planning is a fundamental aspect of autonomous robotics, enabling robots to navigate from a start position to a goal position while avoiding obstacles. This project leverages the `python_motion_planning` library, which provides implementations of various motion planning algorithms, to develop a complete pipeline from path planning to trajectory generation.

2 Library Overview: `python_motion_planning`

The `python_motion_planning` library, developed by ai-winter, offers a comprehensive suite of motion planning algorithms, including:

- **Global Planners**: Dijkstra, A*, JPS, D*, LPA*, D* Lite, (Lazy)Theta*, RRT, RRT*, RRT-Connect, Informed RRT*, Voronoi.
- **Local Planners**: PID, DWA, APF, LQR, MPC, RPP.
- **Curve Generators**: Bezier, Dubins, B-Spline, Cubic Spline, Polynomial, Reeds-Shepp.

The library is structured into modules for global planning, local planning, curve generation, and utilities, facilitating modular and extensible development of motion planning solutions.

3 Project Implementation

The project follows a structured approach:

1. **Environment Setup**: Define the map and obstacles.
2. **Path Planning**: Use RRT* to find a feasible path.
3. **Path Smoothing**: Apply B-Spline curves to smooth the path.
4. **Trajectory Generation**: Generate a time-parameterized trajectory considering kinematic constraints.
5. **Visualization**: Plot the results for analysis.

3.1 Environment Setup

An environment map of size 51x31 units is created, and obstacles are defined using rectangles and circles.

3.2 Path Planning with RRT*

The RRT* algorithm is employed to find a collision-free path from the start to the goal position.

3.3 Path Smoothing with B-Spline

The raw path obtained from RRT* is smoothed using a B-Spline curve to ensure continuity and smoothness, which is essential for trajectory generation.

3.4 Trajectory Generation

A trapezoidal velocity profile is used to generate a time-parameterized trajectory along the smoothed path, considering maximum velocity and acceleration constraints.

4 Pseudocode

The following pseudocode outlines the key components of the implementation.

4.1 RRT* Path Planning

Algorithm 1 RRT* Path Planning

```
procedure RRTStarPlan(start, goal, map)
    Initialize tree with start node
    while not reached goal do
        Sample random point
        Find nearest node in tree
        Steer towards random point
        if path is collision-free then
            Add new node to tree
            Rewire tree to optimize path
    return path from start to goal
```

4.2 B-Spline Path Smoothing

Algorithm 2 B-Spline Path Smoothing

```
procedure BSplineSmooth(path, degree, step)
    Generate B-Spline curve of given degree
    Sample points along the curve at specified step size
    return smoothed path
```

4.3 Trajectory Generation

Algorithm 3 Trajectory Generation with Trapezoidal Velocity Profile

procedure GenerateTrajectory(path, v_{max}, a_{max})
    Compute total arc length L of the path
    Compute acceleration time $t_{accel} = \frac{v_{max}}{a_{max}}$
    Compute distance during acceleration $s_{accel} = 0.5 \cdot a_{max} \cdot t_{accel}^2$
    Compute cruising distance $s_{cruise} = L - 2 \cdot s_{accel}$
    Compute cruising time $t_{cruise} = \frac{s_{cruise}}{v_{max}}$
    Compute total time $T_{total} = 2 \cdot t_{accel} + t_{cruise}$
    for each time t in $[0, T_{total}]$ **do**
        if $t < t_{accel}$ **then**
            $s(t) = 0.5 \cdot a_{max} \cdot t^2$
        else if $t < t_{accel} + t_{cruise}$ **then**
            $s(t) = s_{accel} + v_{max} \cdot (t - t_{accel})$
        else
            $t_{decel} = t - t_{accel} - t_{cruise}$
            $s(t) = s_{accel} + s_{cruise} + v_{max} \cdot t_{decel} - 0.5 \cdot a_{max} \cdot t_{decel}^2$
        Compute position at arc length $s(t)$
    return trajectory

5 Results

The implementation successfully generates a smooth and feasible trajectory for the robot. The RRT* algorithm efficiently finds a path, which is then smoothed using B-Spline curves. The trajectory generation considers kinematic constraints, resulting in a realistic motion profile.

Curve and Result Figures

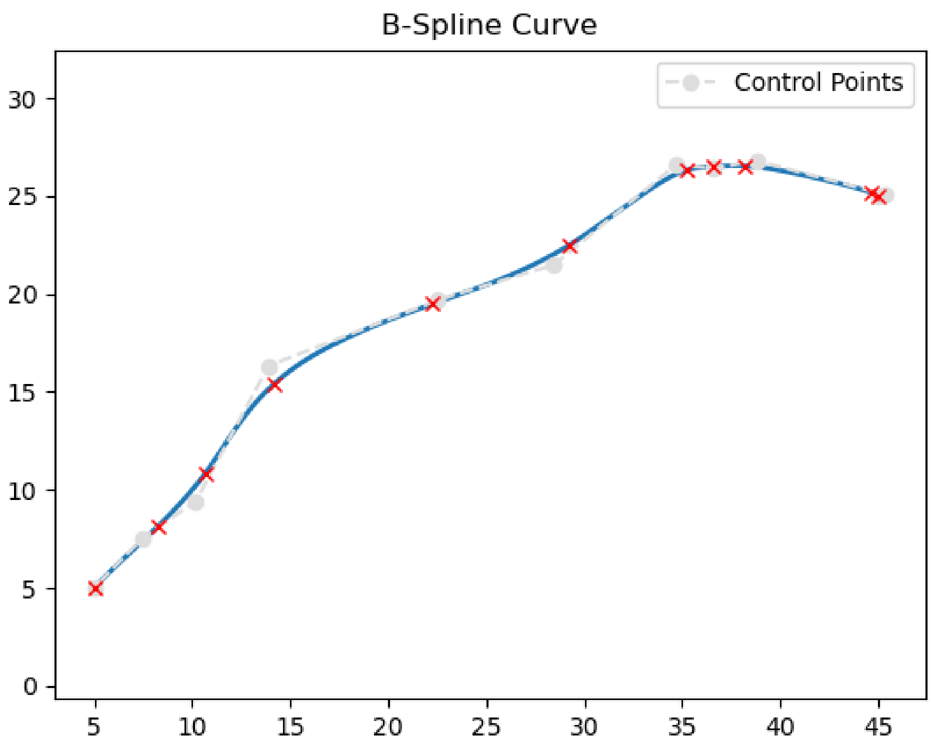

Figure 1: B-Spline Curve generated using the RRT* path.

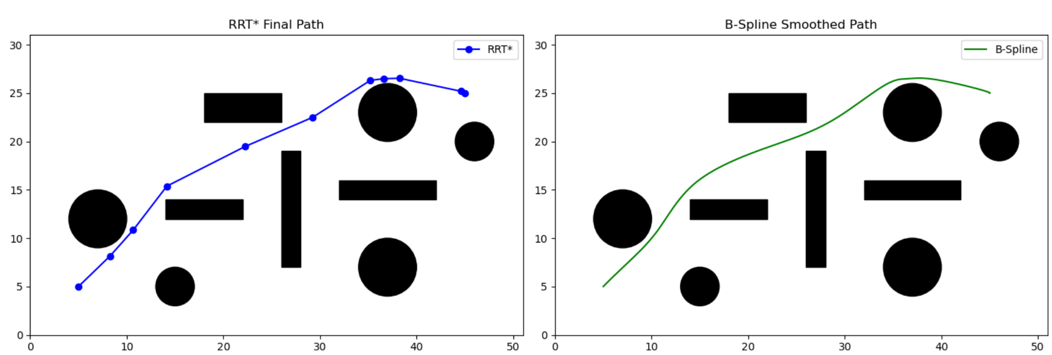

Figure 2: The left image shows RRT* path with inflated obstacles and the right image shows the B-Spline path with inflated obstacles.

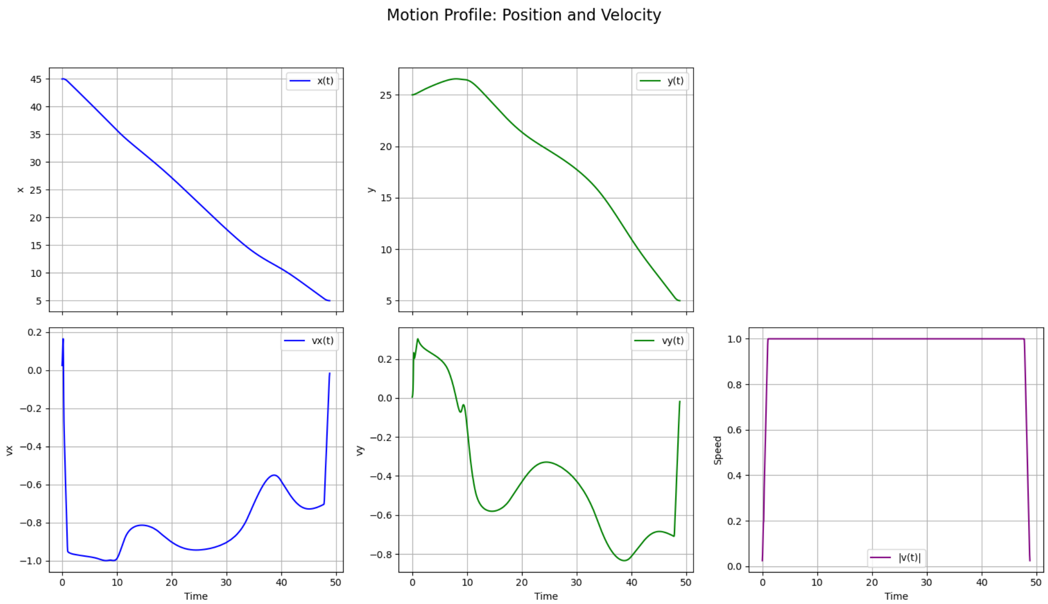

Figure 3: Motion Profile of Robot

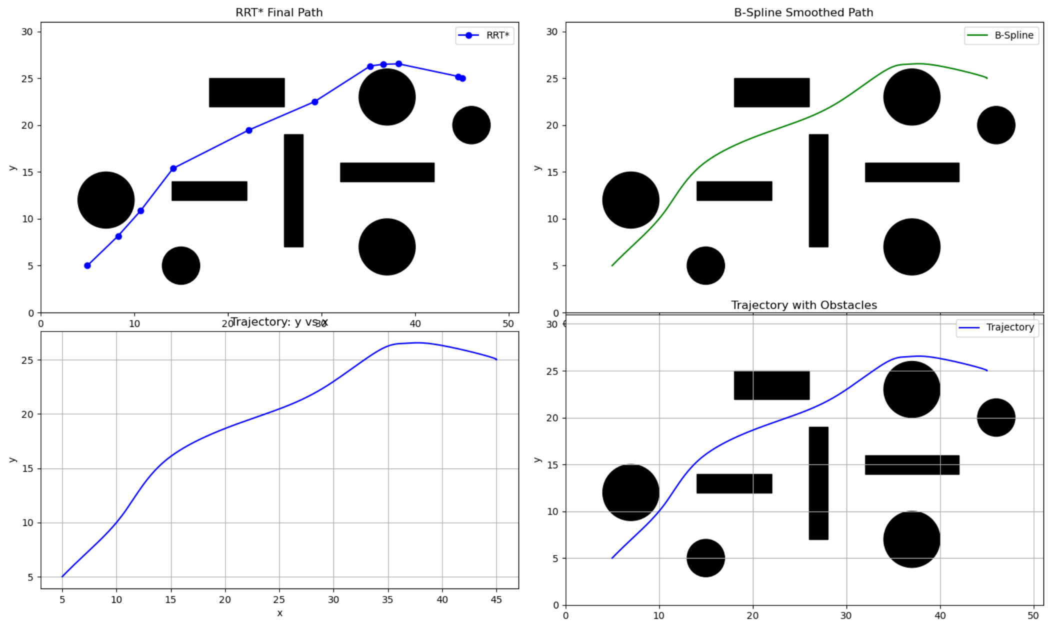

Figure 4: Bottom two images are the curve generated by plotting position vs time of robot which is derived from the motion profile of robot

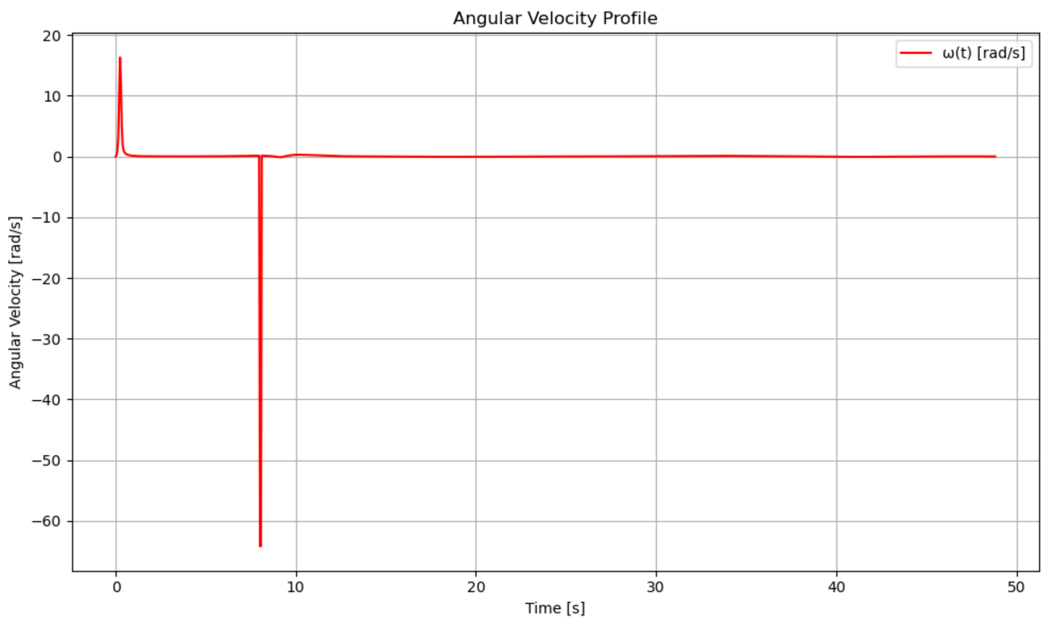

Figure 5: Angular Velocity Profile of Robot

6 Conclusion

This project demonstrates the integration of path planning, path smoothing, and trajectory generation using the `python_motion_planning` library. The modular approach allows for flexibility and can be extended to more complex scenarios, including dynamic environments and higher-dimensional spaces.

7 GitHub Repository

Source code of the project.

8 References

1. ai-winter/python_motion_planning GitHub Repository

2. LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

3. Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G., Burgard, W., Kavraki, L. E., & Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press.