Pybullet Simulated Kinodynamic RRT Path planning

Shrey Shah

***Abstract*—This report presents an efficient implementation of the Kinodynamic Rapidly-exploring Random Tree (RRT) algorithm and its bidirectional variant for motion planning in dynamic environments. The proposed planners are evaluated using a planar hovercraft robot navigating a complex maze environment with non-trivial obstacle configurations. The robot's motion dynamics, including acceleration, deceleration, and orientation constraints, are incorporated into the planning framework. Comparative performance analysis is conducted to assess the impact of varying motion primitives on key metrics such as path quality, computation time, and solution feasibility. Results demonstrate the trade-offs between computational efficiency and trajectory smoothness as the number of motion primitives increases.**

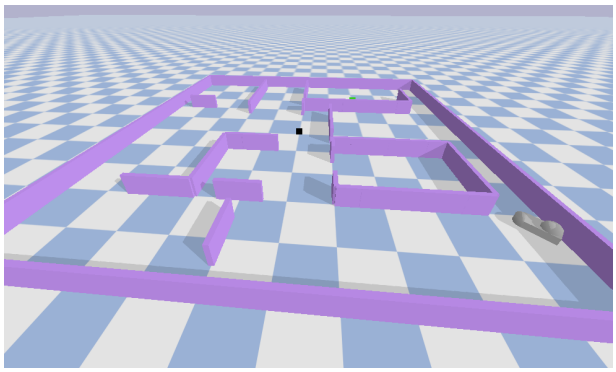

Fig. 1: Maze 1 layout for planning

I. Introduction

Robots need to move smartly through spaces without hitting obstacles, but it gets harder when they have to follow rules like smooth acceleration, deceleration, and turning. This is common for systems like hovercrafts, cars, or drones, which can't change direction instantly. To solve this, we use the Kinodynamic RRT algorithm, which plans paths that respect the robot's motion limits.

This project focuses on using Kinodynamic RRT and its bidirectional version to guide a hovercraft robot through a maze full of obstacles. The challenge is to find a smooth and collision-free path while keeping the planning time reasonable. We test how different numbers of motion primitives (basic movement options) affect the performance, such as the quality of the path and how long it takes to compute.

This kind of planning is important for real-world systems like self-driving cars, drones, or robots working in tricky environments, where paths must not only avoid obstacles but also follow natural motion constraints.

II. Implementation

This section explains the step-by-step implementation of the Kinodynamic RRT algorithm [1]. It covers the robot dynamics, distance metrics, nearest neighbor search, state propagation, and control selection, with an overall algorithmic framework.

A. Robot Dynamics Model

The robot is modeled as a planar hovercraft moving in the x-y plane. Its configuration space is:

$$\mathbf{q} = [x, y, \theta]^T,$$

and the state space is:

$$\mathbf{x} = [\mathbf{q}, \dot{\mathbf{q}}]^T = [x, y, \theta, \dot{x}, \dot{y}, \dot{\theta}]^T.$$

The control input includes forces and torque:

$$\mathbf{u} = [f_x, f_y, \tau_z]^T.$$

The continuous state-space dynamics are given by:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}, \tag{1}$$

where:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}_3 & \mathbf{I}_3 \\ \mathbf{0}_3 & \mathbf{0}_3 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{0}_3 \\ \text{diag}\left(\frac{1}{m}, \frac{1}{m}, \frac{1}{I}\right) \end{bmatrix}.$$

Here, m is the mass of the robot, I is the moment of inertia, and Δt is the simulation time step. Using forward Euler integration, the discrete dynamics become:

$$\mathbf{x}_{k+1} = \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k, \tag{2}$$

where:

$$\mathbf{A}_d = \mathbf{I} + \mathbf{A}\Delta t, \quad \mathbf{B}_d = \mathbf{B}\Delta t.$$

B. Distance Metrics

The distance metric measures the closeness between states. To handle different state components, a weighted Euclidean distance is used:

$$\text{dist}(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{W}(\mathbf{x}_1 - \mathbf{x}_2)\|_2, \tag{3}$$

where $\mathbf{W}$ is a diagonal weight matrix to balance position, velocity, and orientation contributions.

C. Nearest Neighbor Search

The nearest neighbor search is a critical step in the RRT algorithm to find the closest node in the tree to a randomly sampled state $\mathbf{x}_{\text{rand}}$. To make this search efficient, a k-d tree is used.

The k-d tree maintains all nodes of the RRT tree and enables fast querying of the nearest node in logarithmic time complexity, $O(\log N)$. As new nodes are added, the k-d tree is updated. To avoid rebuilding the entire tree at every iteration, we implement a lazy-rebuild strategy:

- New vertices are inserted into an auxiliary list to temporarily buffer nodes.
- The k-d tree is rebuilt only when the buffer size exceeds a predefined threshold.

The nearest neighbor query is performed in two steps:

1) Search the k-d tree to find the closest node.
2) Search the auxiliary buffer (brute force) for any newly added nodes.

The node with the smallest distance is returned as the nearest neighbor.

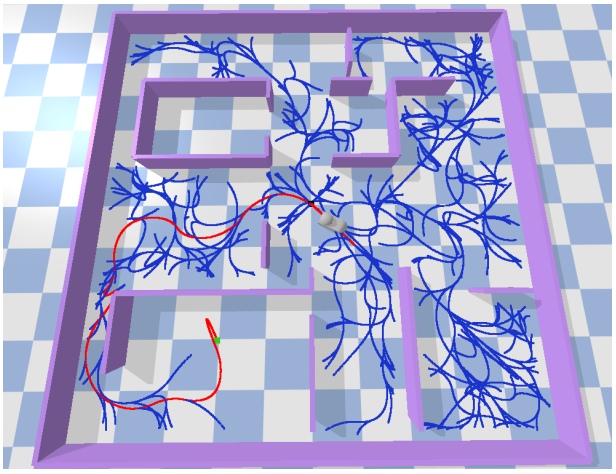

Fig. 2: Kd RRT on maze 1

D. Evaluation Metrics

The performance of the Kinodynamic RRT algorithm is measured using the following metrics:

a) 1. Path Quality:: Path quality is determined by:

- **Path Length:** The total distance between consecutive states in the final path, given by:

$$L = \sum_{i=1}^{N-1} \text{dist}(\mathbf{x}_i, \mathbf{x}_{i+1}). \tag{4}$$

- **Smoothness:** Assessed by evaluating the variation in control inputs along the path.

b) 2. Computation Time:: The total time required by the planner to compute a feasible path, including sampling, nearest neighbor search, state propagation, and collision checking.

c) 3. Solution Feasibility:: Ensures that the generated path satisfies:

- **System Dynamics:** All states are propagated using the defined motion model.
- **Collision-Free Property:** The path avoids all obstacles in the environment.

These metrics provide a clear assessment of the planner's effectiveness in terms of path efficiency, computational performance, and adherence to constraints.

E. State Propagation and Control Application

The tree expands by propagating the nearest state $\mathbf{x}_{\text{near}}$ using a set of predefined motion primitives $\mathbf{u}_i$ (control inputs). The candidate states are calculated as [2]:

$$\mathbf{x}_{\text{new}} = \mathbf{A}_d \mathbf{x}_{\text{near}} + \mathbf{B}_d \mathbf{u}_i. \tag{5}$$

The control input $\mathbf{u}^*$ that minimizes the distance to $\mathbf{x}_{\text{rand}}$ is selected:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}_i} \text{dist}(\mathbf{f}(\mathbf{x}_{\text{near}}, \mathbf{u}_i), \mathbf{x}_{\text{rand}}). \tag{6}$$

F. Switching Between States

The state $\mathbf{x}_{\text{new}}$, generated through state propagation, is added to the tree if:

- It remains within the valid state space.
- The path from $\mathbf{x}_{\text{near}}$ to $\mathbf{x}_{\text{new}}$ does not collide with obstacles.

Collision checks are performed using bounding boxes or meshes.

G. Kinodynamic RRT Algorithm

The steps of the Kinodynamic RRT planner are summarized below:

Algorithm 1 Kinodynamic RRT Planner

```
Initialize tree T with root node x_start
Define goal region x_goal and motion primitives u_i
while Solution not found do
    Sample a random state x_rand
    Find nearest node x_near using k-d tree
    for each control u_i do
        Propagate state: x_new = A_d x_near + B_d u_i
        Evaluate distance: dist(x_new, x_rand)
    Choose u* that minimizes distance
    if x_new is collision-free then
        Add x_new to tree T
        if x_new reaches goal region then
            Return path
end while
```

H. Evaluation Metrics

The performance of the Kinodynamic RRT planner is evaluated using:

- **Path Quality**: Measured by total path length and smoothness.
- **Computation Time**: Time taken to generate a valid path.
- **Solution Feasibility**: Ensures the path respects system dynamics and avoids obstacles.

I. Bidirectional Kinodynamic RRT

The bidirectional Kinodynamic RRT improves efficiency by growing two trees simultaneously:

- One tree starts at $\mathbf{x}_{\text{start}}$.
- The other tree starts at $\mathbf{x}_{\text{goal}}$.

At each step, the trees expand toward random samples. When they grow close enough, a connection attempt is made using a steering function to propagate between states within a threshold distance.

Algorithm 2 Bidirectional Kinodynamic RRT

```
Input: x_start, x_goal
Output: Path from x_start to x_goal
Initialize trees T_a ← x_start, T_b ← x_goal
for k = 1 to K do
    x_rand ← Random_State()
    Expand T_a toward x_rand
    if Is_Connect(T_a, T_b) then
        Connect trees and reconstruct path
        break
    Swap T_a and T_b
Return: Path
```

a) Steering Function:: The steering function propagates states using the dynamics model to ensure smooth transitions:

$$\mathbf{x}_{\text{new}} = \mathbf{A}_d \mathbf{x}_{\text{current}} + \mathbf{B}_d \mathbf{u}.$$

b) Advantages:: The bidirectional approach reduces search time by growing trees from both the start and goal states, improving efficiency for large environments.

III. Results

This section demonstrates and analyzes the performance of the implemented Kinodynamic RRT algorithm and its bidirectional variant. We first describe the experimental setup and then discuss the results based on path quality, computation time, and control configurations.

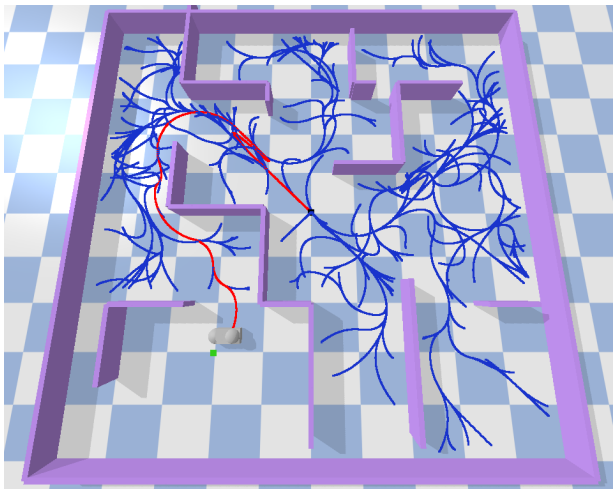

Fig. 3: KdRRT: Maze 2

A. Experimental Setup

The environment for the experiments is implemented in **PyBullet** [3], which allows real-time collision detection and physics simulation. A 2D maze environment with obstacles is created as the testing platform, and the robot model is defined as a planar hovercraft with the following characteristics:

- **Mass and Inertia:** The mass of the robot is $m = 1\,kg$, and its moment of inertia is $I = 1\,kg \cdot m^2$.
- **State Bounds:** Positions are constrained within $[-10, 10]\,m$, orientations within $[-\pi, \pi]\,rad$, and velocities within $[-1, 1]\,m/s$ or rad/s.
- **Control Bounds:** Control inputs (forces and torques) are bounded within $[-1, 1]\,N$ or $N \cdot m$.

a) Motion Primitives:: Motion primitives are generated using two methods:

1) **Heuristic Grid Sampling:** Control inputs are discretized into a grid (e.g., $[-1, 0, 1]$) for each dimension.
2) **Random Sampling:** A set of N control inputs is randomly sampled within the control bounds [4].

The number of motion primitives tested includes 6, 26, and 52 controls, representing varying levels of motion agility.

b) Environment Configuration:: A 2D maze is defined with obstacle walls and a start and goal position. The path quality is evaluated as the accumulated sum of Euclidean distances between waypoints:

$$L = \sum_{i=1}^{N-1} \|\mathbf{q}_{i+1} - \mathbf{q}_i\|_2. \tag{7}$$

B. Performance Analysis

The performance is evaluated for both the standard Kinodynamic RRT (KdRRT) and its bidirectional variant (BiKdRRT). The metrics for comparison include:

TABLE I: Performance Statistics for Various Examples

Algorithm	# Controls	Random	Path Quality	Min Time	Avg Time
KdRRT	6	False	18.23	1.45	7.25
	26	False	14.41	3.24	8.99
	6	True	29.46	2.41	10.78
	26	True	20.71	3.05	7.54
BiKdRRT	6	False	19.10	0.21	1.75
	26	False	20.90	0.48	1.85
	6	True	22.79	0.77	2.81
	26	True	21.59	0.63	2.72

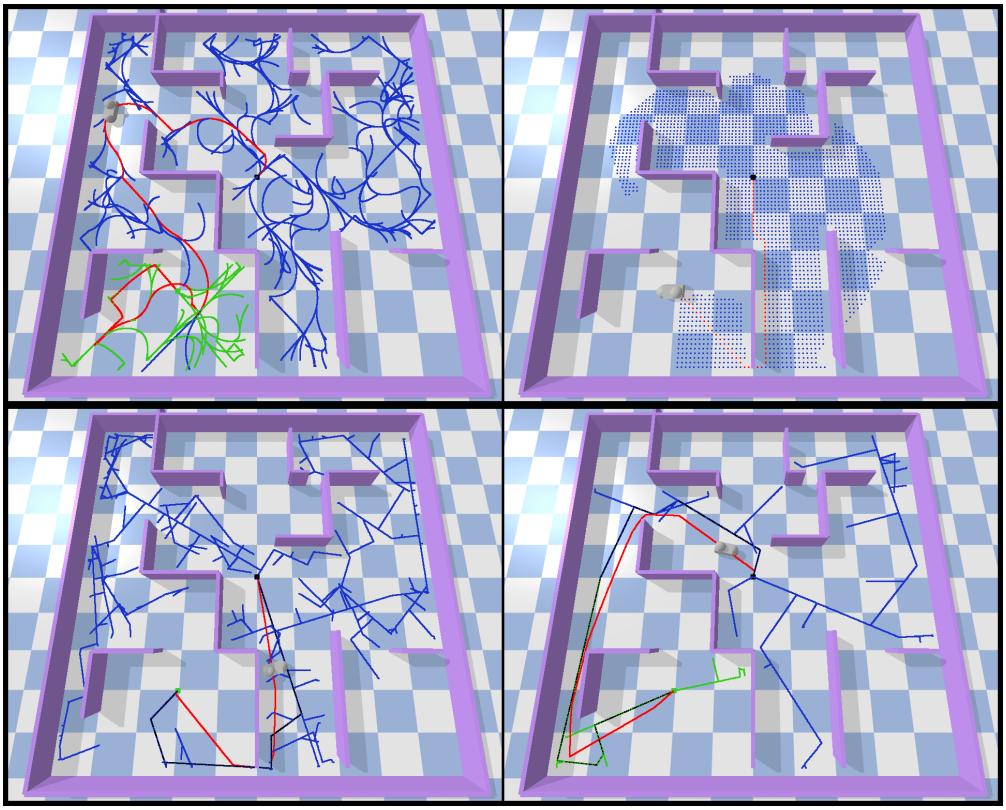

Fig. 4: Comparison of different algorithms on maze 2: a) BiKdRRT, b) A*, c) RRT, d) BiRRT

- **Path Quality:** Measured as the total length of the path.
- **Computation Time:** Total time taken to compute the path.
- **Control Configurations:** The effect of increasing the number of motion primitives on performance.

a) Results Table:: Table I summarizes the performance of both algorithms for different numbers of controls. The results are averaged over 10 executions.

C. Observations

- **Path Quality:** As the number of motion primitives increases, the path quality improves due to greater motion agility.
- **Computation Time:** The bidirectional Kinodynamic RRT (BiKdRRT) significantly reduces computation time, approximately 3 times faster than the standard KdRRT.
- **Random Sampling:** Random control sampling results in longer paths but increases the likelihood of finding a solution in complex environments.

D. Visualizations

Figures 1 and 4.a) illustrate the paths generated by the algorithms. The explored nodes are shown in blue, and the final searched path is highlighted in red and green for the start and goal trees.

Fig 4 also shows a comparison between different path planning algorithms derived from homework assignments to show how KD RRT behaves differently and is smoother compared to the other algorithms.

IV. Conclusion

In this work, we implemented and analyzed the Kinodynamic RRT algorithm and its bidirectional variant (BiKdRRT) for motion planning in a 2D environment using a planar hovercraft robot. The algorithms were tested in a maze-like environment with non-trivial obstacles, and their performance was compared in terms of path quality, computation time, and control configurations.

The following key findings were observed:

- The bidirectional Kinodynamic RRT significantly reduced computation time compared to the standard RRT, achieving up to three times faster planning while maintaining comparable path quality.
- Increasing the number of motion primitives improved the overall path quality due to higher motion agility but increased the computational cost.
- Random control sampling, while less efficient in path quality, demonstrated robustness in finding solutions for complex environments.

The experimental results and visualizations confirmed that the BiKdRRT approach is well-suited for scenarios requiring efficient motion planning under dynamic constraints. Future work can focus on extending this approach to 3D environments, optimizing the motion primitives for real-time applications, and integrating more sophisticated steering functions for smoother trajectories.

References

[1] T. Webber, T. Howard, and J. F. Canny, "Rrt-based planning for kinodynamic systems," *Robotics: Science and Systems*, 2010.

[2] A. Perez, R. Platt, G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2537–2542, 2012.

[3] P. Developers, "Pybullet: Real-time physics simulation for robotics." https://pybullet.org, 2022.

[4] L. Jaillet, J. Cortés, and T. Simeon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, 2010.