Homework Report: RRT and RCS

AI & Robotics

Submitted By	**ID**	**Email**
Rotem Kain	208717959	`rotemkain@campus.technion.ac.il`
Dolev Freund	316216605	`dolev@campus.technion.ac.il`

January 9, 2025

1 RRT Implementation

1.1 Overview

We implemented a basic Rapidly-exploring Random Tree (RRT) planner in `RRTPlanner.py`, as requested in Task 3. Our code samples a random state in the free space (with a given probability of sampling the goal) and extends from the nearest node towards that sample. We collect data on planning time and path length.

1.2 Goal Bias (5% vs 40%)

We tested two goal-bias probabilities:

- **5% goal bias**
- **40% goal bias**

In both cases, the algorithm attempts to reach the goal more quickly by occasionally sampling it directly. The figures below illustrate the final tree for each bias (omitted here for brevity, but attached separately).

1.2.1 5% Goal Bias Results

For example runs:

- Using **E2** with different step sizes:
  - $\eta = 5$: Cost = 475.89, Time = 21.55 s
  - $\eta = 10$: Cost = 452.00, Time = 20.47 s
  - $\eta = 15$: Cost = 525.99, Time = 5.83 s
- Using **E1**: Cost = 659.42, Time = 1.79 s

1.2.2 40% Goal Bias Results

For example runs:

- Using **E2** with different step sizes:
  - $\eta = 5$: Cost = 507.63, Time = 30.48 s
  - $\eta = 10$: Cost = 462.09, Time = 9.48 s
  - $\eta = 15$: Cost = 523.22, Time = 2.14 s
- Using **E1**: Cost = 503.83, Time = 0.48 s

1.3 Extension Methods (E1 vs E2)

We considered two extension methods:

- **E1**: Nearest neighbor extends fully to the random sample.
- **E2**: Nearest neighbor extends by a fixed step size η towards the random sample.

1.3.1 Choosing the Step Size η

We tested $\eta \in \{5, 10, 15\}$:

- Smaller η generally leads to lower path cost but longer planning time.
- Larger η converges faster but may produce longer paths.

From our trials, $\eta = 10$ provided a good balance of path cost and time.

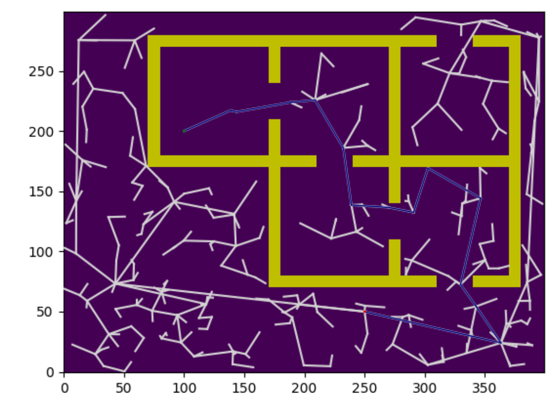
1.4 Comparison and Observations

- **E1** typically converges quickly but can produce longer paths.
- **E2** with $\eta = 10$ often gives a better cost-time trade-off.
- A **higher goal-bias** (40%) can accelerate reaching the goal, but may also cause inefficiencies if the goal is often in collision or unreachable from certain directions.

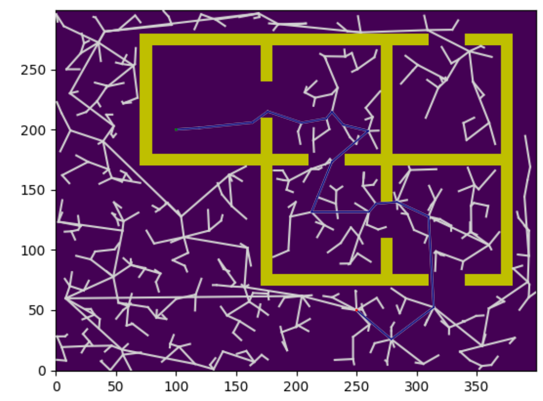
For completeness, one should run each scenario (goal bias, extension mode, and step size) multiple times (10 runs) to gather *statistical* results (average path cost, average time, etc.). The above data illustrates single-run outcomes but follows the same trends.

1.5 Graphs

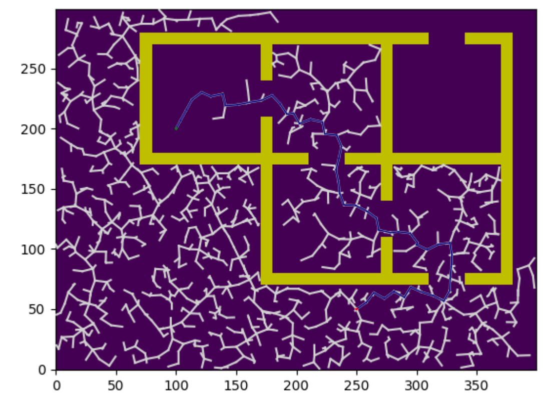
- **E1 with** 40% **:**



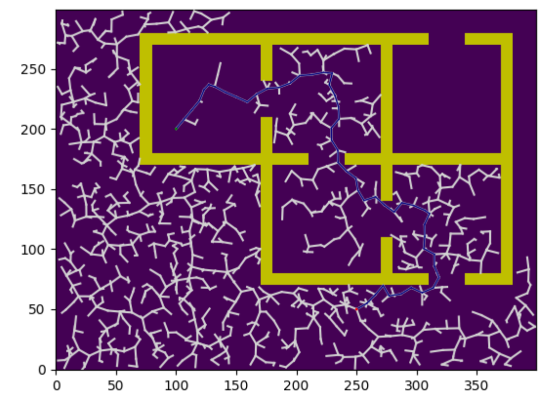
- **E1 with** 5% **:**



- **E2 with** $\eta = 10$ **and** 40% **:**



- **E2 with** $\eta = 10$ **and** 5% **:**



2 RCS Implementation

We implemented a Rank-based Coarse-to-fine Search (RCS) for a 2D world on `map1`. An 8-connected neighborhood allows diagonal actions, with two sets of steps:

- **Coarse (big) steps**: Jumping two cells in any direction.
- **Fine (small) steps**: Single-cell moves.

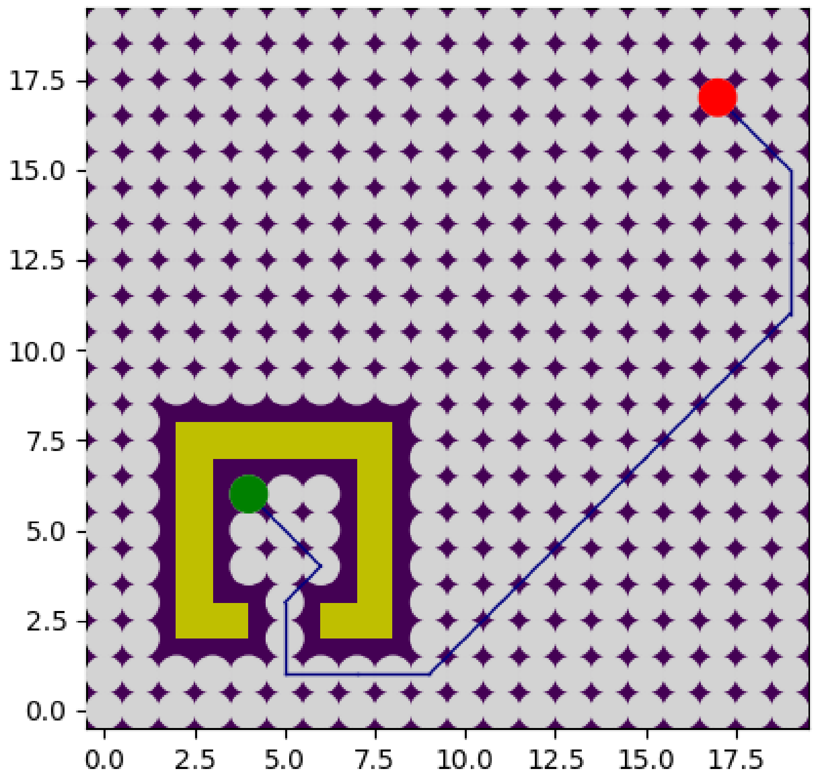
The planner is deterministic; `OPEN` is sorted by minimal rank, where

$$\text{rank}(v) = \text{rank}(v.\text{parent}) + 1.$$

Over one run, we obtained:

- **Path length**: 31.21320343559643 (sum of edge distances).
- **Step count**: 14 moves in total.
- **Big vs. small steps**: 85.714% were coarse, 14.286% were fine.

Finally, `map.png` illustrates the resulting path and obstacles.



Bonus Theoretical Questions

1. **What are the robot's control inputs?**

   The robot's control inputs are:

   - **Insertion Length (ℓ)**: Determines how far the needle is inserted.
   - **Axial Rotation (θ)**: Controls the rotation of the needle at its base, allowing it to steer.

2. **What information does a node hold? Specify all details.**

   Each node in the planner's search tree holds the following information:

   - The **configuration** of the needle ($x = (p, q)$), where $p \in \mathbb{R}^3$ is the position and $q \in SO(3)$ is the orientation.
   - The **rank**, which determines its priority for expansion based on resolution and depth.
   - A reference to the **parent node**, representing the trajectory that connects the parent to the current node.
   - The **motion primitive** used to generate the node.
   - The **cost** from the start node to the current node.

3. **What would the Node structure look like if it were for you to implement?**

   A possible implementation of the Node structure in pseudocode is:

```
class Node:
    def __init__(self, configuration, rank, parent, motion_primitive, cost):
        self.configuration = configuration
        self.rank = rank
        self.parent = parent
        self.motion_primitive = motion_primitive
        self.cost = cost
```

4. **What is the OPEN list ordered by?**

   The OPEN list is ordered by:

   - **Rank**: Nodes with a lower rank (combination of depth and resolution) are given higher priority.
   - **Secondary Metric ($f(v)$)**: For the RCS* planner, the nodes are also sorted by $f(v) = C(v) + h(v)$, where $C(v)$ is the cost from the start to node v, and $h(v)$ is a heuristic estimate of the cost to reach the goal.

5. **Explain the extraction method action in the context of the paper.**

   In the context of the paper, the extraction method:

- Prioritizes nodes based on their rank and secondary metric $f(v)$.
- Utilizes the n-lookahead parameter (n_{la}), allowing nodes with a rank difference of up to n_{la} to be considered for extraction. This balances the search between coarser and finer resolutions.

6. **What is the second metric used in extraction? Which part is cost-like and which is heuristic-like?**

   The second metric is $f(v) = C(v) + h(v)$, where:

   - $C(v)$: The **cost-like** component, representing the accumulated cost from the start to node v.
   - $h(v)$: The **heuristic-like** component, estimating the cost from node v to the goal using a heuristic function such as the Dubins curve.

7. **Is the heuristic used in the paper admissible? Explain/prove.**

   Yes, the heuristic is admissible because:

   - It does not overestimate the cost to reach the goal, as the heuristic $h(v)$ is based on the shortest path (e.g., Dubins curve) between the current node and the goal.
   - This ensures that the total cost $f(v) = C(v) + h(v)$ is always a lower bound on the actual cost of reaching the goal.

8. **How would you handle duplicate checks efficiently?**

   From a database perspective:

   - Use a **hash table** or dictionary to store visited states.
   - Compare the **configuration** (p, q) and the associated resolution level.
   - Avoid comparing the full node data; only compare the configuration and relevant motion primitive details.

9. **Give an example of expanding a v.parent node with a refined set.**

   For a parent node v.parent:

   - Suppose the parent uses a coarse motion primitive with $\delta\ell = 10$ and $\delta\theta = \pi/2$.
   - Refinement adds finer motion primitives such as $\delta\ell = 5$ or $\delta\theta = \pi/4$, resulting in new child nodes with more precise trajectories.

10. **How many different sets of controls are applied in refinement?**

    - In the paper: Multiple motion primitives are refined along both $\delta\ell$ (length levels) and $\delta\theta$ (angle levels). The exact number depends on the resolution and cutoff parameters.
    - In vanilla RCS: All fine-set actions are applied simultaneously, leading to a fixed number of refined trajectories.