ARTIFICIAL INTELLIGENCE AND ROBOTICS CS 236006

# HOMEWORK #2

DATE: JANUARY 16, 2025

LECTURER: SARAH KEREN

SUBMITTED BY: ALMOG ANSCHEL 316353531
SHAHAR BEN YEHUDA 212723225

## 1 RRT implementation

### 1.

This method is stochastic so we will provide statistical results (averages over 10 runs). Noticing that the extend function and potentially $\eta$ influence the results. Therefore we decided to check the results based on E1 extend function where $\eta$ does not influence the results:

- **when the bias is 5%**:
  - **average path length:** The results we got after 10 runs are - 429.85, 630.64, 753.84, 527.69, 608.04, 553.04 , 515.14, 600.12, 620.12, 608.56. Therefore the average is - **584.2 [atu]**
  - **average planning time:** The results we got after 10 runs are - 1.69, 0.66, 1.36, 3.42, 2.67, 5.2, 2.36, 6.39, 0.82, 0.61. Therefore the average is- **2.518 [seconds]**
  - **Final Tree:**

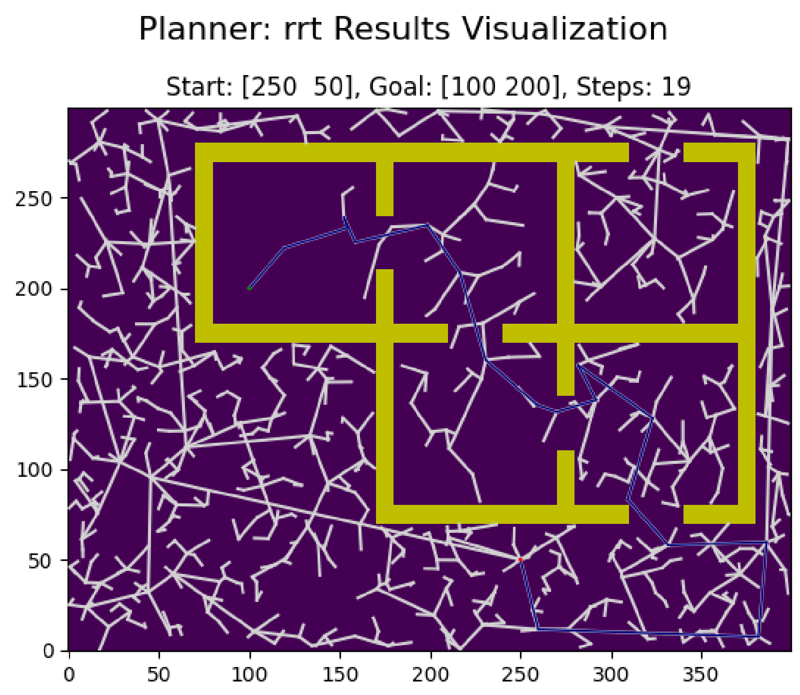


Figure 1: RRT tree with 5% goal bias

- **when the bias is 40%**:

- **average path length:** The results we got after 10 runs are- 703.37, 529.87, 522.07, 454.02, 483.64, 649.26, 479.11, 522.28, 520.93, 786.81. Therefore the average is - **565.136 [atu]**
- **average planning time:** The results we got after 10 runs are - 2.56 , 0.3, 0.24, 0.39, 0.67, 1.94, 2.35, 3.25, 1.23, 1.74. Therefore the average is - **1.467 [seconds]**
- **final tree:**

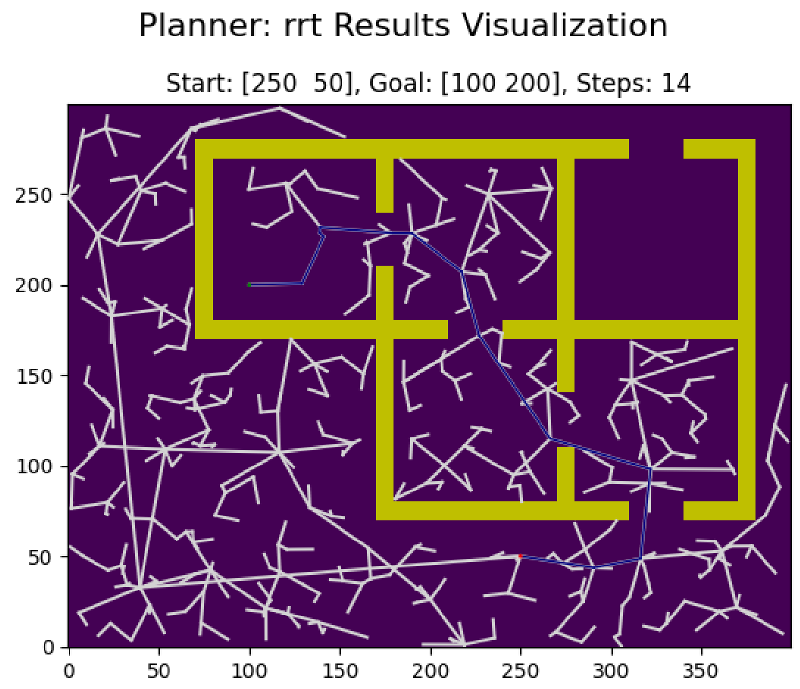


Figure 2: RRT tree with 40% goal bias

The goal bias controls the probability of sampling the goal directly. We notice that a low goal bias helps to ”explore”, which we think could help in a cluttered environment, while a higher bias leading to faster convergence to the goal.

## 2.b

We will check which step size (and bias to goal) works best for us based the average results of 10 runs. Note that we also checked step size 5 but it took a long time to find solution even for 1 run so we wont choose it:

- **step size 10, goal bias 0.05:** cost of path in 10 runs is- 645.86, 480.54, 555.32, 484.06, 533.88, 489.92, 537.02, 585.04, 544.44, 510.46. The average is **526.954 [atu]**

  time to find the path is - 6.74, 8.40, 5.11, 1.53, 5.89, 8.85, 5.84, 12.06, 4.28, 8.37. The average is **6.707 [seconds]**

- **step size 10, goal bias 0.4:** cost of path in 10 runs is- 546.11, 552.14, 497.63, 510.97, 469.86, 477.60, 514.88, 621.80, 532.82, 539.30. The average is **510.311 [atu]**

  time to find the path is- 19.00, 11.79, 4.31, 10.27, 13.43, 11.44, 9.53, 17.45, 13.11, 20.60. The average is **13.088 [seconds]**

- **step size 15, goal bias 0.05:** cost of path in 10 runs is- 532.58, 470.23, 522.89, 531.87, 620.28, 486.39, 453.66, 584.58, 454.32, 564.28. The average is **522.644 [atu]** time to find the path is- 0.99, 3.9, 3.29, 4.67, 6.36, 1.95, 1.21, 7.34, 2.84, 6.33. The average is **3.88 [seconds]**

- **step size 15, goal bias 0.4:** cost of path in 10 runs is- 520.38, 476.93, 536.92, 529.10, 498.37, 608.67, 488.74, 482.76, 470.39, 556.02. The average is **516.828 [atu]**

  time to find the path is- 6.65, 10.11, 4.43, 2.71, 2.23, 2.9, 6.07, 8.9, 5.45, 4.53. The average is **5.398 [seconds]**

As you can see in our results, the lowest average path cost was achieved with step size 10 and goal bias 0.4 ( 510.311 atu), while the lowest average time to find a path was achieved with step size 15 and goal bias 0.05 ( 3.88 seconds). However, the option that provides the best balance between path cost and planning time is step size 15 and goal bias 0.4. This configuration achieves a near-optimal path cost of 516.828 atu, only 6.517 atu higher than the lowest cost, while significantly reducing the planning time to 5.398 seconds, making it much faster than the option with the lowest cost. This balance between efficiency and quality makes **step size 15 and goal bias 0.4 the most practical choice for our scenario**.

## 2.c

As mentioned in previous sections we chose E1 with goal bias 0.05 and E2 with step size 15 and goal bias 0.4. As before, this method is stochastic so we will provide statistical results (averages over 10 runs):

- **E1 and goal bias 5%**:
  - **average path length:** The results we got after 10 runs are - 429.85, 630.64, 753.84, 527.69, 608.04, 553.04 , 515.14, 600.12, 620.12, 608.56. Therefore the average is - **584.2 [atu]**
  - **average planning time:** The results we got after 10 runs are - 1.69, 0.66, 1.36, 3.42, 2.67, 5.2, 2.36, 6.39, 0.82, 0.61. Therefore the average is- **2.518 [seconds]**
  - **Final Tree:**

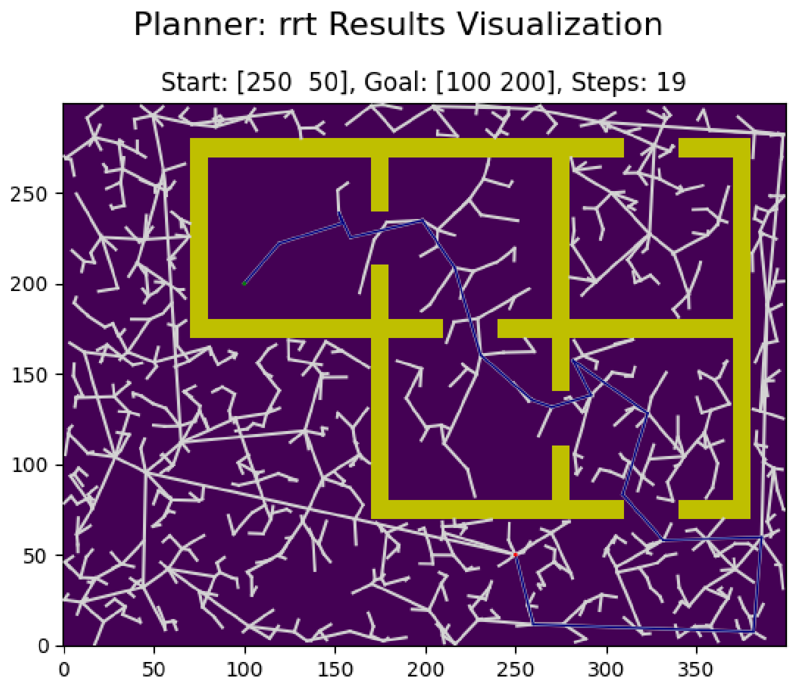


Figure 3: RRT tree with 5% goal bias

- **E2 with step size 15 and goal bias 40%**:
  - **step size 15, goal bias 0.4:** cost of path in 10 runs is- 520.38, 476.93, 536.92, 529.10, 498.37, 608.67, 488.74, 482.76, 470.39, 556.02. The average is **516.828 [atu]**

    time to find the path is- 6.65, 10.11, 4.43, 2.71, 2.23, 2.9, 6.07, 8.9, 5.45, 4.53. The average is **5.398 [seconds]**

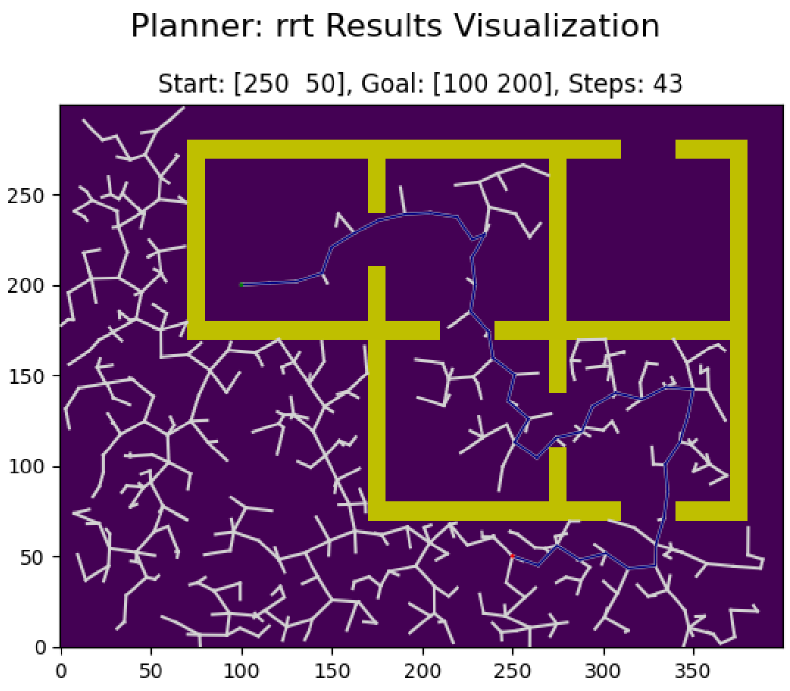


Figure 4: fig:RRT with $\eta = 15$ and 40% bias

Overall, we show a trade-off in tuning the goal bias parameter of the RRT algorithm. A lower goal bias (5%) drives exploration of the configuration space , which can be better in dense environments, but results in longer planning times, while a higher goal bias (40%) directs the tree more aggressively toward the goal, hence, achieving faster convergence.
In comparing the extend strategies, although E1 delivered a shorter planning time, E2 produced a lower path cost. Given that E2 maintains an acceptable planning time while significantly reducing the overall path cost, we have chosen it as the preferable strategy.

## 2 RCS Implementation

For this part, we used the *RCS* algorithm on a 2D grid. We used an 8-connected neighborhood structure, as shown in Figure 5, meaning that each node was allowed to move to its 8 adjacent grid cells, including diagonals. We also utilized coarse and fine expansions:

- **Coarse actions (big steps)**: $\pm(2, 0)$, $\pm(0, 2)$, and $\pm(2, 2)$, $\pm(2, -2)$.
- **Fine actions (small steps)**: $\pm(1, 0)$, $\pm(0, 1)$, and $\pm(1, 1)$, $\pm(1, -1)$.

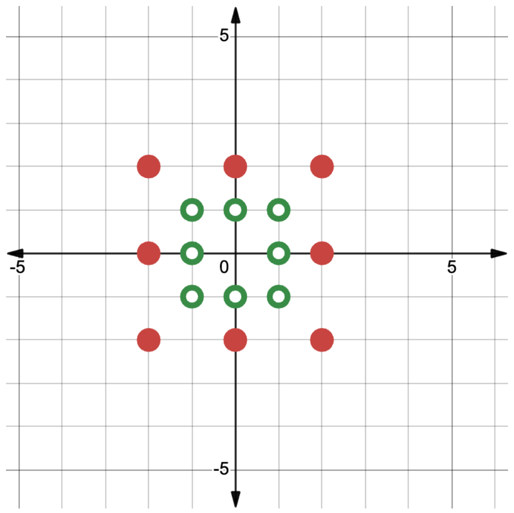


Figure 5: RCS planning 8-connected neighborhood

The Algorithm that we implemented follows the pseudo-code in the exercise, as stated:

1. Each coarse action adds 1 to the node's rank.
2. If the current node is not the root and was reached by a coarse action,
   we attempt to expand from its parent with all fine actions.
3. The priority queue is sorted by ascending rank.
4. Once we pop a node off priority queue, if it is valid and unvisited, we check whether it is the goal.
   If so, we reconstruct and return the path.

**Results** Our solution path took:

- Total Path Steps $= 14$
- Path Length = 29.5
- Big Steps (# ,%) = 13, 92.86% of the path
- Fine Steps (#, %) = 1, 7.14% of the path

The RCS algorithm results path (Figure 6) demonstrates the balance between exploration and refinement. Most of the path is composed of big steps which allows the robot to make large jumps through the free configuration space. Hence, the planning converges quickly. The occasional use of a fine step provides a tighter navigate trough narrow corridor of the obstacle.
The final path avoids obstacles successfully, but we can see that is not the optimal solution, This solution is "good" in the sense that it completes the task in a small number of steps, but "bad" if we simply asked for the shortest path.

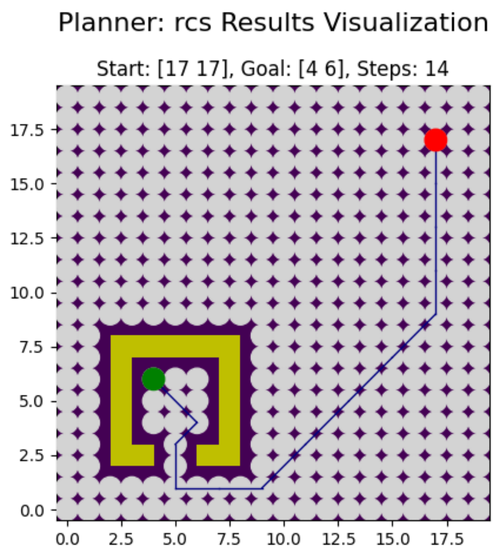


Figure 6: RCS planning result

## 3 Theoretical Questions - Bonus

We know this paper as Ido, the tutor spoke about it in general line in class.

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

The robot's control inputs are defined as:

- $\delta\ell$: The insertion length of the steerable needle.
- $\delta\theta$: The axial rotation angle of the needle at its base.
- $\kappa$: The curvature of the needle trajectory, constrained by $\kappa \leq \kappa_{\text{max}}$.

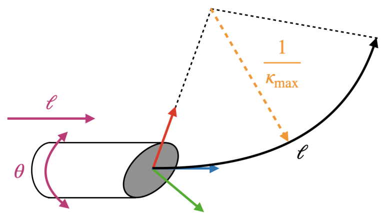


Figure 7: Input Controls, figure taken from the relevant paper

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

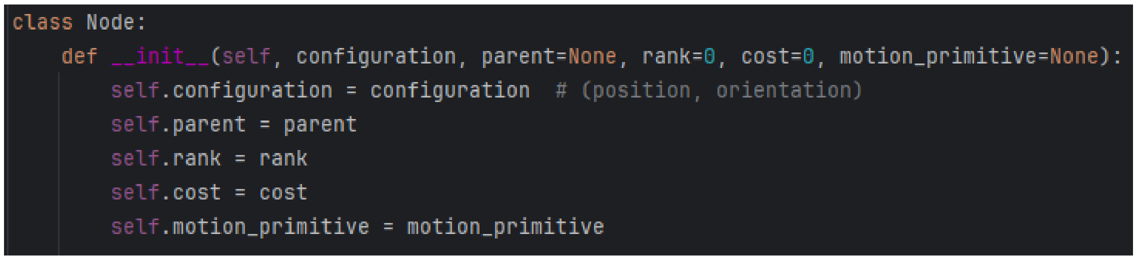
A node holds the following information, :

- **Configuration**: $x = (p, q)$, where $p \in \mathbb{R}^3$ is the position and $q \in SO(3)$ is the orientation.
- **Parent node**: A reference to the parent node in the search tree.
- **Rank**: The node rank, which is updated based on motion resolution.

- **Cost**: The accumulated cost from the start node to the current node.
- **Motion Primitive**: The motion primitive $\mathcal{M} = (\kappa, \delta\ell, \delta\theta)$ that connects the parent to the current node.

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

The same as we defined last task:



```
class Node:
    def __init__(self, configuration, parent=None, rank=0, cost=0, motion_primitive=None):
        self.configuration = configuration  # (position, orientation)
        self.parent = parent
        self.rank = rank
        self.cost = cost
        self.motion_primitive = motion_primitive
```

## 4. Look over the OPEN list data structure in the paper. What is it ordered by?

The `OPEN` list is ordered by:

1. **Rank**: Nodes are prioritized by their rank, which reflects the resolution and depth.
2. **Secondary Metric**: Within the same rank, nodes are prioritized by $f(v) = C(v) + h(v)$, where $C(v)$ is the cost and $h(v)$ is a heuristic estimate.

## 5. Extracting from the OPEN list acts differently from popping a node out of a heap. Explain what the extraction method action is in the context of the paper. Make sure not to miss the *n* look-ahead parameter.

The extraction method uses a look-ahead parameter $n_{\text{la}}$:

- Nodes are first sorted by rank.
- Nodes within a range of rank $\leq r_{\text{open}} + n_{\text{la}}$ are evaluated based on $f(v) = C(v) + h(v)$.

This approach ensures coarse resolution nodes are expanded first to find early initial solutions.

## 6. Extraction also depends on a second metric. Write down this metric, and explain which part is similar to a cost-like metric and which is a heuristic-like metric.

The secondary metric is:

$$f(v) = C(v) + h(v),$$

where:

- $C(v)$: The cost-like metric, representing the accumulated cost from the root to the node.
- $h(v)$: The heuristic-like metric, estimating the cost from the node to the goal.

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

Yes, As written in section `4.2` of the paper, the heuristic $h(\cdot)$ is admissible because it never overestimates the true cost to the goal, it based on *Dubins Curves*, which provides a lower-bound estimate of the path cost, Hence $h(v) \leq$ true cost to the goal, and it satisfies the admissibility criterion.

**8. A CLOSED list is used to identify duplicate states. Explain from a database point of view how you would handle the duplicate check to be as efficient as possible. What node values would you compare? Is it going to compare all the node data?**

To efficiently handle duplicates, We would utilize a hash table where the key is a hash of the node configuration $x = (p, q)$. following the next procedure:

- Compare only configuration$(p, q)$ and the cost $C(v)$:
  - If a duplicate is found with lower cost, discard the current node.
  - Otherwise, replace the stored node with the current one.

**9. Unlike the vanilla RCS, where all *fine set* actions are applied simultaneously, the paper uses a different approach. Give an example of expanding a $v.parent$ node with a refined set (in the context of the paper). In your answer, refer to the insertion level and the angle level.**

For a node $v$, refining its parent node $v$.parent involves generating additional child nodes using finer motion primitives $M = (\kappa, \delta\ell, \delta\theta)$

- **Insertion Length** ($\delta\ell$): Refined into smaller intervals as:

$$M_{\ell\pm} = \left(\kappa, \delta\ell \pm 2^{-(l_\ell(M)+1)} \cdot \delta\ell_{\text{max}}, \delta\theta\right),$$

- **Axial Rotation Angle** ($\delta\theta$): Refined into finer steps as:

$$M_{\theta\pm} = \left(\kappa, \delta\ell, \delta\theta \pm 2^{-(l_\theta(M)+1)} \cdot \delta\theta_{\text{max}}\right),$$

**10. Following the last question, in a refinement primitive motion, how many different sets of controls are applied to the node in the context of the paper? How many in the context of the simple vanilla RCS from part 4?**

- **In the paper**: Each refinement applies up to 4 sets of controls (refining both $\delta\ell$ and $\delta\theta$).
- **In vanilla RCS**: All fine set actions are applied simultaneously, leading to fewer iterations but potentially more computational overhead.