HW 2 – Sampling-based & search-based motion planning

1. General guidelines

This homework is to be done in pairs. Submit a single zipped folder containing:

- PDF with your typed answers and the figures needed to demonstrate your results. Note that all the figures/plots should be integrated into the PDF submitted.
- your code - including all the files of the project.

The name of the zip file needs to include the two ids of the collaborators on the assignment as follows: ID1_ID2.zip - for example: 111111111_66666666.zip

2. Code overview

The environment code is written in Python and depends on the libraries: numpy, matplotlib, imageio, and shapely.

```
pip install numpy
pip install matplotlib
pip install imageio
pip install Shapely
```

You are provided with the following files:

- run.py - Contains the main functions.
- MapEnvironment.py – Environment specific functions. Some of them must be filled by you.
- Map1/2.json – JSON files describing the maps that the code will generate. They contain map boundaries, polygonal obstacles and start and goal locations.
- RCSPlanner.py – RCS planner. Logic to be filled in by you.
- RRTPlanner.py – RRT planner. Logic to be filled in by you.
- RRTTree.py – Contains RRT tree code to be used by your implementations of RRT.

Here are examples of how to run the code using different maps and planners:

```
Python run.py -map map2.json -planner rrt -ext_mode E2 -goal_prob 0.05
Python run.py -map map1.json -planner rcs
```

The planning consists of a 2D map. You have been provided with two maps map1.json and map2.json. Make sure to report your results on the provided start and goal

positions. Note that environment-specific functions, such as collision detection and distance computation, are already implemented in the MapEnvironment.py file.

In case of search algorithms like RCS, the environment is considered to be a discrete grid while in sampling-based techniques the environment is assumed to be continuous. However, in this case since the underlying world is represented as a grid, you can snap any continuous sample points onto the grid. Note: During the whole assignment, use < x, y > coordinate convention (i.e.,) to describe 2D points. The plotting function will work properly as long you follow this convention.

3. RRT implementation [search for Task 3 in the code]

You will be implementing RRT for 2D world. The main algorithm to be implemented is in RRTPlanner.py file. Note that since this method is stochastic, you'd need to provide statistical results (averages over 10 runs). **Use map2 in your report for RRT.**

1. Bias the sampling to pick the goal with 5% and 40% probability. **Report** the performance (path length, planning time) and include figures showing the final tree for both values.
2. You can assume the robot is able to move in arbitrarily any direction. You will implement two versions of the extend function:
    a. E1 – the nearest neighbor tries to extend all the way to the sampled point.
    b. E2 – the nearest neighbor tries to extend to the sampled point only by a step size η Try step size of $\eta = \{5, 10, 15\}$ and pick which one works best for you, state it, and base your results on it. Explain why you chose that specific step size.
    c. As before, for each version of extend, report the performance and include a figure. Which extend strategy worked better for you?

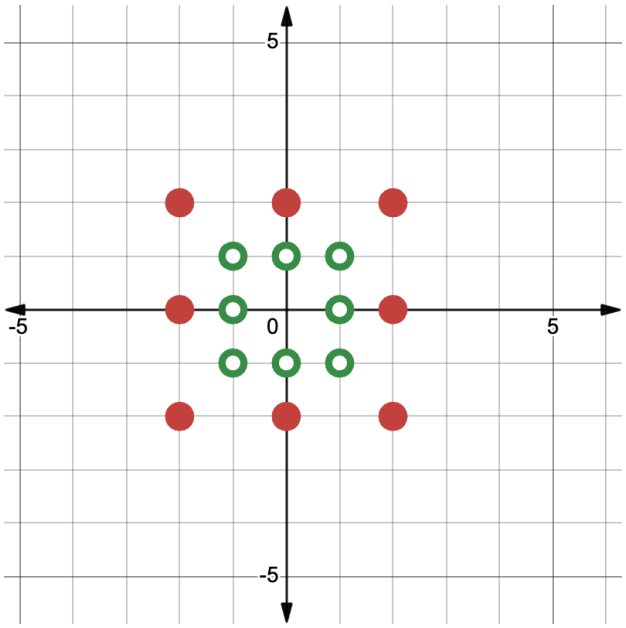
4. RCS implementation [search for Task 4 in the code]

You will be implementing RCS for 2D world. The main algorithm to be implemented is in RCSPlanner.py file. Note that this method is deterministic and therefore you need to report 1 run. **Use map1 in your report for RCS.**

1. Use an 8-connected neighborhood structure so that diagonal actions are also allowed. Report how many steps did you take in the solution path.
   The coarse movement will be defined as taking "big" steps to positions shown in red circles. A refinement process will take the "small" step from a given position.
   The OPEN list will be sorted by minimal rank.
   The rank should be updated as such: $rank(v) = rank(v.parent) + 1$

2. **Report** the path length (sum of edge length/distance) and path step count (how many moves the robot takes), as well as how many “big” and “fine” steps were taken (in percentages).
3. After running the algorithm, you should get a map.png. Include it in the PDF report.

See graph below and pseudocode:




```
coarseSet  {(2,2), (2,0), …(0,-2),…,(-2,2)}
fineSet  {(1,1),…,(-1,-1)}
rootNode  (startState,0) #Use nodes
[state,rank,resolution,parent_node]
OPENlist  {root}, CLOSEDlist <- {}
Plan = []

While not OPEN.empty() do
      V  OPEN.extract()
      If(Valid(v)):
            If(not CLOSED.existDuplicate(v)):
                  If(goal_reached):
                        Build path and return path
                  For action in coarseSet:
                        newNode  propagate(v,action)
                        OPEN.insert(newNode)
                  CLOSED.append(v)

      If v != root and v.resolution = coarse:
            For action in fineSet:
                  newNode  propagate(v.parent, action)
                  OPEN.insert(newNode)

Return np.array(plan)
```

5. BONUS [Theoretical questions]

As mentioned in class, the case we used for the exercise is a simple case of resolution search. On the course Github, under Tutorials, you can find the steerable needle paper, where resolution search is more complex.

You must answer all the questions for this part to receive the bonus.
All questions are in the context of the paper!

1. What are the robot's control inputs?
2. What information does a node hold? Specify all details.
3. What would the Node structure look like if it were for you to implement?
4. Look over the OPEN list data structure in the paper. What is it ordered by?
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.
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.
7. Is the heuristic used in the paper admissible? Explain/prove.
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?
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.
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?