Motion Planning RBE 550

Project 3: Sampling-Based Planners

DUE: Thursday, Oct 2nd at (8:30 am) before class starts.

All the written components should be typesetted with LaTeX. A template is provided on Overleaf. However, feel free to use your own format as long as it is in LaTeX. The written report should be submitted as a PDF file.

This assignment can be completed (optionally) in pairs. Present yours and your partner work only. In the written report include both your names in the title. You must *explain* all of your answers. Answers without explanation will be given no credit.

Submit two files **a)** your code as a zip file **b)** the written report as a pdf file. If you work in pairs, only **one** of you should submit the assignment, and write as a comment the name of your partner in Canvas. **5 points will be deducted if these instructions are not followed.**

Theoretical Questions (40 points)

1. **(10 points)** Recall the visibility graph method. Similar to the PRM, the visibility graph also captures the continuous space using a graph structure. Compare these two methods. For each method, provide at least one scenario in which it would work well while the other would not. Justify your answer.

2. **(10 points)**

   For each of the three manipulators shown in Figure 1, determine the topology and dimension of the manipulator's configuration space.

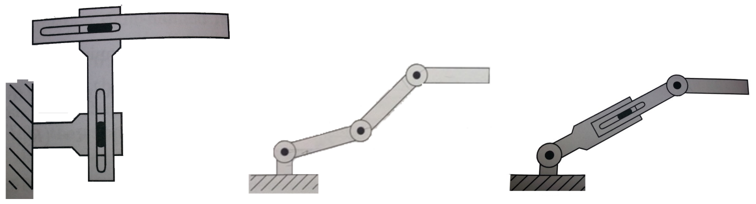

Figure 1: From left to right: a manipulator with two prismatic joints, a manipulator with three revolute joints, and a manipulator with two revolute joints and a prismatic joint.

3. **(10 points)** Consider workspace obstacles A and B. If $A \cap B \neq \emptyset$, do the configuration space obstacles QA and QB always overlap? If $A \cap B = \emptyset$, is it possible for the configuration space obstacles QA and QB to overlap? Justify your claims for each question.

4. **(10 points)** Suppose five polyhedral bodies float freely in a 3D world. They are each capable of rotating and translating. If these are treated as "one" composite robot, what is the topology of the resulting

configuration space (assume that the bodies are not attached to each other)? What is the dimension of the composite configuration space?

Programming Component (60 points + 20 bonus)

In this assignment, you are going to implement *PRM* and *RRT* algorithms. For *PRM*, you are required to implement 4 different sampling methods - *uniform sampling*, *gaussian sampling* and *bridge sampling*. These are three basic sampling strategies for PRM-based methods.

The following template is provided to you as a start point. After you finish coding, you would be able to run these algorithms to find a path in a map, and visualize the result.

- **PRM.py** is the file where you will implement your algorithms. As we will use a grader code to help us grade your assignments more efficiently. Please do not change the existing functions names, or the docstring tests.
- **RRT.py** is the scrip that provides helper functions that load the map from csv files and visualize the map and path. You are not required to modify anything but you are encouraged to read and understand the code.
- **main.py** This file includes auxiliary structures that you could use to help you with your implementation. Using these structures is optional but potentially very helpfull. You are not required to modify anything but you are encouraged to read and understand the code.
- **WPI_map.csv** is the map file you could modify to create your own map.

Please finish reading all the instructions below before starting actual coding.

Installation Instructions (Optional)

This assignment was tested with

- `python 3.8.10`
- `matplotlib 3.4.2`
- `numpy 1.17.4`
- `networkx 3.1`
- `Pillow 7.0.0`

Your default setup will probably work but if it does not, you can install these specific versions by creating a virtual environment with the following commands in the terminal:

```
# Creating the virtual environment
python3.8 -m venv venv search.py
# Activating the environment
source venv/bin/activate
# Installing the requirments
pip install -r requirements.txt
```

Now the virtual environment should be activated, and if you run any python commands it should be using the specified versions for the required packages. If you exit/change the terminal you should re-activate the environment.

Get Started

Before starting any coding, please run the code first:

```
python main.py
```

The **main.py** loads the map image **WPI_map.jpg** and calls classes and functions to run planning tasks. As you haven't written anything yet, there should be no path shown in the graph, but only the map image, start point and end point.

Please keep in mind that, the coordinate system used here is **[row, col]**, which is different from [x, y] in Cartesian coordinates. In README and the code comment, when the word '**point**' is used, it refers to a simple list [row, col]. When the word '**node**' or '**vertex**' is used, it refers to either the Node class in RRT ,or a node/vertex in a graph in PRM.

```
 python main.py
```

There should be 3 maps shown representing the results of 3 algorithms. As said before, there would be no path shown in the graph as you haven't implemented anything yet. The **main.py** loads the map file **map.csv**, which you are free to modify to create your own map.

PRM

The two main phases of PRM are **Learning Phase** and **Query Phase**.

Learning Phase

You would code **Learning Phase** in the function *sample*, where it samples points in the map according to different strategy, and connect these points to build a graph. In this template, the graph library Networkx is used to store the result graph.

There are three different sampling methods to be implemented - *uniform_sample*, *gaussian_sample* and *bridge_sample*. Please refer to the lectures and make sure you understand the ideas behind these sampling methods before coding.

After sampling, you would need to connect these sampling points to their k-nearest neighbors. To find their neighbors, you should NOT just use brutal force algorithm (This will take way too long), but use K-D tree as mentioned in the class. Here is an example of how to use scipy K-D tree structure.

Finally, you will need to use all the sampled points and their connection with neighbors as nodes and edges to build a Networkx graph.

Query Phase

You would code **Query Phase** in the function *search*, where it searches for a path in the constructed graph given a start and goal point.

As start and goal points are not connected to the graph, you will first need to add the start and goal node, find their nearest neighbors in the graph and connect them to these two nodes. Practically, as some of the graphs don't have a good connectivity, we will not only connect the start and goal node to their nearest node, but all the nodes within a certain distance, in order to increase the chance of finding a path.

Having connected start and goal node in the graph, we could use Dijkstra algorithm or any other algorithms we learn before to search for a valid path. This part is similar to the first assignment, so is already done by using the Dijkstra function Networkx provided.

Finally, as PRM is a multi-query planning algorithms, one could call ‘search‘ with other start and goal point. So the previous start and goal nodes and their edges need to be removed in the end of each query phase. This part is also implemented already.

Read the description of the functions for more details before implementing.

RRT

For simplicity, this template uses a class *Node* and a list *vertices* in class *RRT* as a tree structure. If you prefer to use other tree structure, please feel free to do so.

You would code RRT in the function *RRT*. In each step, get a new point, get its nearest node, extend the node and check collision to decide whether to add or drop this node. When you add a new node to the tree, remember to set the cost and parent of the new node, and add the new node to the list *vertices*. You will also need to check if it reaches the neighbor region of the goal. If so, connect to the goal directly and set the found flag to be true.

Read the description of the functions for more details before implementing.

This template is only provided as a start point, feel free to make any modification of the codes or code structures if needed. After you finish coding, your algorithms should produce similar results as the images in **demo** folder.

Run your code with `python main.py` and **save your results as png images**.

Grading

1. **(30 points)** Correct *PRM* implementation.

   - **(20 points)** The three sampling methods produce the correct sampling points. In your report include the produced png pictures and explain how different sampling leads to different sample sets in the graph?
   - **(5 points)** Connect the sampling points, start and goal into a graph using a proper method.
   - **(5 points)** Given start and goal, find a path if feasible.

2. **(30 points)** Correct *RRT* implementation.

   - **(15 points)** Get proper new nodes in each step.
   - **(10 points)** Find a path if feasible.

3. **(20 bonus points)** Implement a new sampling strategy that outperforms, Uniform, Bridge, and Gaussian sampling for PRM.

   - You can use a weighted combination of the existing samplers, or design your own custom sampling distribution.
   - The performance metric you should use is the number of nodes needed until a solution is found.
   - As these planners are probabilistic you should run each planner at least 10 times, and report your results as boxplots in your report.
   - In your report explain the thinking behind the design of your sampling-distribution.
   - You would need to modify the existing structure of the code, and create an incremental PRM, that searches for a solution after one or more nodes are added to the graph, instead of adding all the nodes first and then searching the roadmap.
   - You can use the **WPI_map.jpg** or another map of your design.