Motion Planning RBE 550

Project 4: Rigid Body Motion Planning

DUE: Thursday, Oct 30th 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. Please follow this formatting structure and naming:

```
project_YOUR_NAME.pdf
project_YOUR_NAME.zip
    code
        main.py
        ....
```

5 points will be deducted if these instructions are not followed.

Theoretical Questions (40 points)

1. **(10 points)** Answer the following questions about asymptotically optimal planners:

   (a) **(5 points)** What is the core idea behind RRT*? That is, explain what modifications are done to RRT in order to make it asymptotically optimal. What methods are changed, and what changes are they?

   (b) **(5 points)** How does Informed RRT* improve upon RRT*. Is Informed RRT* strictly better than RRT* ? e.g., are there any cases where RRT* can find a better solution given the same sequence of samples?

2. **(15 points)** Figure 1 shows an indoor vacuum robot has a differential drive chassis with two wheels, each with a radius of r, that are separated by a distance L.

   Each wheel is controlled independently with its own motor. Presume that you can specify the torque for each motor, and the motors can directly change the velocities (u_l and u_r) of your Roomba. Then the dynamics of your Roomba can be specified with the following differential equations:

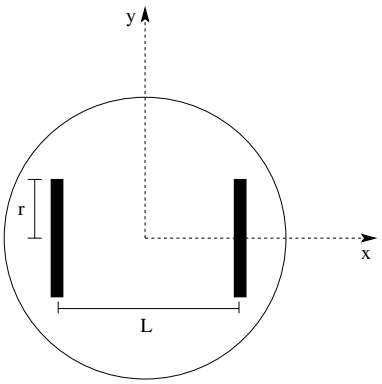

Figure 1: The Indoor Vacuum Robot

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \frac{r}{2}(u_l + u_r)\cos(\theta) \\ \frac{r}{2}(u_l + u_r)\sin(\theta) \\ \frac{r}{L}(u_r - u_l) \end{pmatrix}$$

(a) **(5 points)** What is the configuration space, the control space, and the state-space of the robot?

(b) **(10 points)** Given the above equations of motion and using the Euler approximation, compute the next state of the robot if it starts from configuration $(x, y, \theta) = (0, 0, 0)$ and controls $(u_l, u_r) = (1, 0.5)$ are applied for 1 second. Show your work, don't just write a number.

3. **(15 points)** Imagine you are having a rectangular robot that rotates in the 2D plane among axis aligned bounding boxes (AABBs) .

   The shape of the robot is assumed to be rigid, allowing us to unambiguously specify the geometry of the robot in a local coordinate frame, as shown in Figure 2(a). With this representation, it is easy to rigidly transform the geometry and place the robot at a specific configuration in the workspace for collision checking.

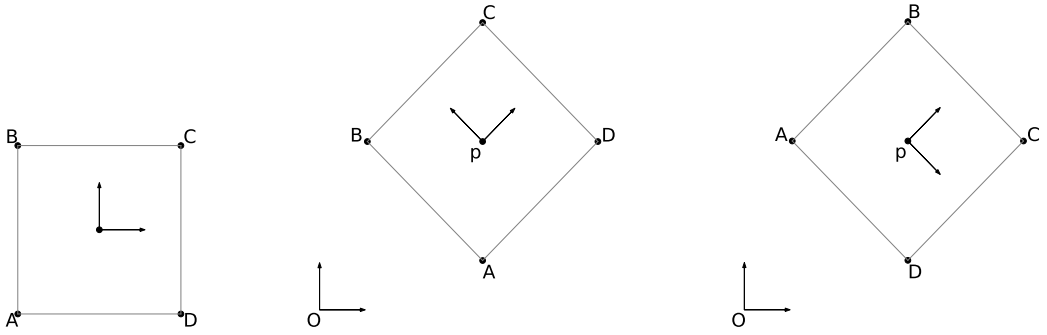

Figure 2: (a) The geometry of the square robot in its local coordinate frame. The points A-D are specified with respect to the local origin at the center of the square. The choice of local origin is arbitrary, but must remain constant. (b) The robot at $p = (x, y)$ and $\pi/4$ rotation in the workspace. (c) The robot at $p = (x, y)$ and $-\pi/4$ rotation in the workspace. Although (b) and (c) look similar, these are two geometrically distinct configurations of the robot.

For the above robot, provide a pseudo-code that checks if the robot is in collision with the obstacles. You can assume that you have access to a line_intersection algorithm e.g. what was implemented in the previous Project. You know that the obstacles are represented as Axis aligned boxes, but you are free to parameterize them as you want.

Programming Component (60 points + 20 bonus)

In this programming assignment, you are going to apply the sampling-based planners *RRT*, *PRM*, *RRT** and *Informed RRT** (**bonus**) to 3 different types of robots.

1. 2D point robot with C-space R^2.
2. 2D free-flying rectangular robot with a C-space in $S \times R^2$.
3. An N-link kinematic chain with C-space S^n.

The implementation for *RRT*, *PRM*, and the point robot is provided. You will have to implement:

1. The 2D free-flying rectangular robot
2. The N-link kinematic chain.
3. RRT* .
4. Informed RRT* (**bonus**) .

The following files are provided:

- **main.py** This file includes all the different scenarios you can run to test your code. You should comment/uncomment this code to create different images to include in your report. You should not change the size of the rectangular robot or the kinematic chain.
- **map_2d.py** A file that represents the 2D map. In contrast to the previous assignment this map includes polygonal obstacles as which are represented by their corners. You are not required to modify anything but you are encouraged to read and understand the code.
- **util.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.
- **maps/map.csv** Is the map file that will be loaded and you should use for your report. You are encouraged to create your own maps as well to test the limits of these planners.
- **planner.py**. An abstract interface for the different planners. This Class abstract the common operations between all the different planners. You **should not** modify this code since we will be using an auto-grader.

- **sampling_method.py**. Auxiliary methods to be used by different planners. You **should not** modify this code since we will be using an auto-grader.
- **demos** Folder with demos to give you an idea on how the solutions should look like.
- **Ccode.pdf**. Schematic of the code structure. What is noted with red should be implemented/filled by you.
- **PRM.py** The PRM implementation for an abstract robot. This code is fully provided and you **should not** modify it. It includes all the methods from the previous assignments.
- **RRT.py** Includes the RRT implementation, and template code for *RRT** and *Informed RRT**. You **should modify** this code to implement these two planners.
- **robot.py** Includes an abstract interface for the three different robots, and implementations for each robot. The point robot implementation is provided but you **should modify** this file to implement all the necessary functions for the 2D free-flying robot and the N-link kinematic chain.

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`
- `Shapely 2.0.2`

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 and create a solution for the point robot using a planner. If you use any of the other robots that are not implement yet it is possible that the code will get stuck, as the default collision checker always returns false.

You can either start with implementing RRT*/Informed RRT* or the 2 new robots. Take a look at the videos folder for examples on how the solutions should look like. for the different Robots and Planners.

In this project we have abstracted away the concept ofr a robot when using the planners to demonstrate the generality and abstraction of the planners we have learned in class. Thus by only changing the specifics for each robot, and without modifying any of the core code for the planners the same planner should be able to plan for all the different robots.

Omnidirectional 2D Robot

This is a rectangular robot that opearates in $S \times R^2$ For this robot you should implement the following functions in **robot.py**:

1. `forward_kinematics()`
2. `get_edges()`
3. `distance()`
4. `interpolate()`

Although this robot is implemented for a general rectangular robot, please use the dimensions provided in main, when producing your figures. Read the descriptions of the function to understand what should be implemented.

N-Link Kinematic Chain

This is a rectangular robot that opearates in S^n. For this robot you should implement the following functions in **robot.py**:

1. `forward_kinematics()`
2. `get_edges()`
3. `distance()`
4. `interpolate()`

Although this robot is implemented for a general n-link chain please use the dimensions and number of links when producing your figures. Read the descriptions of the function to understand what should be implemented.

Collision Checking

To ensure proper collision checking you should modify the following function in **robot.py**:

`check_collision_config()`

The current implementation only accounts for the Point Robot collisions.
Note: Make sure that your collision checking will work for any polygonal obstacle.

RRT*

An RRT implementation is already provided for you, and a basic implementation for RRT* is also provided. The first few steps are pretty much the same as RRT. Besides, when a new node is added, you will need to rewire the new node AND all its neighbor nodes. Even a path is found, the algorithm should not stop as adding new nodes will possibly optimize the current found path.

You will need to implement the following functions in **RRT.py**

1. `rewire()`
2. `get_neighbors()`

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

Informed RRT* (BONUS)

Most of the code structure is the same as the RRT*'s template, but this time, all the codes that are related to RRT are provided. There are three main changes you will need to have in mind compared to RRT*

1. Informed RRT* should operate exactly the same as RRT* until an initial solution is found
2. Each time a new/better solution is found, the current best cost should be saved.
3. Ellipsoid sampling should be performed for each of the different configuration spaces.

No template is provided besides the function signature, but the implementation should be similar to RRT*

For more details, please go through the lecture and the original paper Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic.

Deliverables and Grading

1. **(15 points)** Correct Omnidirectional 2D Robot. Include images for solutions from *PRM*, *RRT* and *RRT**. Choose any sampling distribution you want for PRM, you are not required to use all of them.

2. **(25 points)** Correct N-link Kinematic chain 2D Robot. Include images for solutions from *PRM*, *RRT* and *RRT**. Choose any sampling distribution you want for PRM, you are not required to use all of them.

3. **(20 points)** Correct collisions checking for Both Omnidirectional and N-link Chain. Ensure your implementation will work for any polygonal obstacle.

4. **(20 bonus points)** Correct *Informed RRT** implementation.

    - **(5 Points)** Correct implementation for 2D point robot (Include image)
    - **(5 points)** Correct implementation for 2D omnidirectional robot (Include image)
    - **(10 points)** Correct implementation for N-link Chain (Include image)