Homework 2

Robot Autonomy
CMU 16-662, Spring 2014

TAs: Jennifer King and Matt Klingensmith

February 26, 2014

1 Introduction

In this homework you will be exploring using randomized planners for two different configuration spaces. You will implement the RRT planner and the RRTConnect planner. You will first run the planners in a simple two dimensional configuration space. Then you will move to the higher dimensional space of the WAM arm on the HERB robot.

2 Planning for the Mobile Base

First, you will implement the planners and use them to move the robot base. We will consider an omni-directional robot and ignore orientation. So the pose of the robot $q = [x, y] \in \mathbb{R}^2$.

Included in the assignment are stubs for each piece of code that you will need to implement. You should see a file **run.py**. This runs the code. The following will show you the options available:

```
python run.py --help
```

Running the following command will generate and execute the default plan for the mobile base:

```
python run.py --robot simple
```

The default plan is simply a two point plan that contains the start and goal. As can be seen, the robot drives straight through the table in the environment. Your goal in this part of the assignment is to implement a planner to create a collision free trajectory for the robot.

2.1 Simple Environment

The file **SimpleEnvironment.py** implements the 2D configuration space that you will plan in. First you will implement the following three functions in this file:

1. *GenerateRandomConfiguration* - This function generates a random collision-free configuration from $\mathbb{R}^2$. The configuration should be inside the boundaries of the space.

2. *ComputeDistance* - This function computes the distance between two configurations in $\mathbb{R}^2$.

3. *Extend* - This function attempts to extend from a start configuration to a goal configuration. The extension should be a linear interpolation between the two configurations. The function should return either None or a configuration such that the linear interpolation between the start configuration and this configuration is collision free and remains inside the environment boundaries.

2.2 RRT

Next you will implement an RRT. To do this, implement the *Plan* function in the file **RRTPlanner.py**. The function should use the three functions you implemented in the previous step.

We have provide a visualization tool for you to visualize the growth of your tree and use for debugging. To use the visualizer, call the function *PlotEdge* on the planning environment every time an edge is added to the tree. Then start the planner with the visualize option:

```
python run.py --robot simple --planner rrt --visualize
```

2.3 RRTConnect

Finally you will implement the bidirection version of the RRT algorithm - RRTConnect. To do this, implement the *Plan* function in the file **RRTConnectPlanner.py**.

3 Planning for a manipulator

Once you have a working version of the RRT and RRTConnect algorithms for the two dimensional robot, we will now use those planners to plan for a 7 degree-of-freedom manipulator. Running the following command will generate and execute the defualt plan for the 7 DOF arm:

```
python run.py --robot herb
```

Again, the default plan is simply a two point plan that contains the start and gaol. As can be seen, the robot arm plans straight through the table in the environment. Your goal in this part of the assignment is to implement a planner to create a collision free trajectory for the arm.

3.1 Herb Environment

The file **HerbEnvironment.py** implements 7DOF WAM arm configuration space. Implement the following three functions in this file:

1. *GenerateRandomConfiguration* - This function generates a random collision-free configuration. The configuration should respect joint limits for each degree of freedom.

2. *ComputeDistance* - This function computes the distance between two configurations.

3. *Extend* - This function attempts to extend from a start configuration to a goal configuration. The extension should be a linear interpolation in joint space between the two configurations. The function should return either None or a configuration such that the linear interpolation between the start configuration and this configuration is collision free and respects joint limits.

The following code can be used to obtain the defined joint limits for the robot:

```
lower_limits, upper_limits = self.robot.GetActiveDOFLimits()
```

3.2 Path Shortening

The plans generated in the previous step contain a lot of extra unecessary movements. In this part of the assignment we will attempt to smooth the plan. To do this, implement path shortening by implementing the *ShortenPath* function in **HerbEnvironment.py**. The algorithm you implement should respect the time constraint given as a parameter to the function.

4 Deliverables and Grading

Please turn in a zip file containing your code, a PDF writeup and the videos mentioned below. Only one person per group needs to submit but please make sure everyone's name and andrewid is on the pdf. The following shows the point breakdown:

1. Run the RRT planner for the 2D configuration space several times (10 or more). Report average path length, average plan time and average number of vertices in the tree. Submit a video of at least one run. What goal sampling probability did you use? (5 pts)

2. Run the bidirectional RRTConnect planner for the 2D configuration space several times (10 or more). Again report average path length, average plan time and average number of vertices in the tree. How does the bidirectional version compare to the regular RRT? (5 pts)

3. Run the RRT planner and the bidirectional RRTConnect planner for the WAM arm. Report average path length, average plan time and average number of vertices in the tree. How do the two algorithms compare? Is it more, less or equally useful to use bidirectional planning in the higher dimensional configuration space? Why? Submit a video of at least one run. (5 pts)

4. Utilize the path shortening algorithm to shorten the path. What cost function do you use in the shortening algorithm? Run the planner 10 times and use the path shortening algorithm to reduce the path. What is the average reduction in path cost? Submit a video of a path before and after shortening. (5 pts) Extra Credit: Also implement path shortening for the simple robot. Submit a video of a path before and after shortening. (+3 pts)