Due Date: 11 NOV 11:59 PM

CS 7649 Project #1

CS 4649/7649 Robot Intelligence: Planning
Instructor: Matthew Gombolay

Notes:

- You may work with one or more classmates on this assignment. However, all work must be your own, original work (i.e., no copy+pasting code). You must list all people you worked with and sources you used on the document you submit for your homework.

Introduction:

Recall the path-planning problem from PSet 4 for the room below (with a notional RRT graph partially depicted). All dimensions are in meters. The four corners of the room are

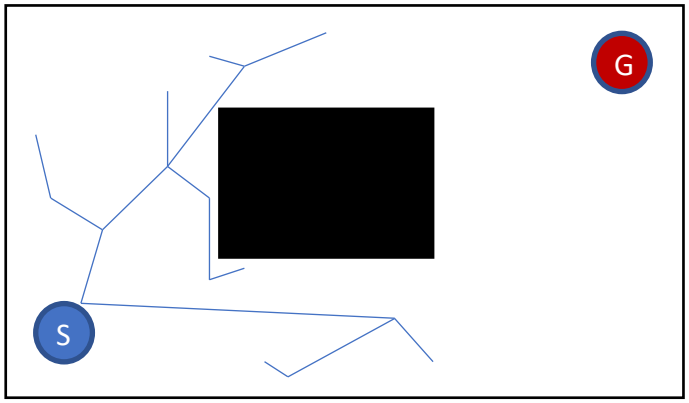
- (0,0)
- (10,10)
- (0,10)
- (10,0)

The four corners of the obstacle in the middle are:

- (3,3)
- (7,7)
- (3,7)
- (7,3)

Your robot starts at location (1,1). The goal location is (9,9).

In this problem, we asked you to implement **RRT** to find a collision-free path from the start to goal locations. You were told to ignore dynamics (i.e., your robot can turn in any direction with infinite linear and angular acceleration). Specifically, Steer($x_{nearest}, x_{rand}$) was to simply create a line segment from $x_{nearest}$ to x_{rand}. The robot could not go outside of the room, nor could it go through the obstacle. When evaluating ObstacleFree($x_{nearest}, x_{new}$), you needed to check to make sure the path does not go through an obstacle! We asked you to run the FOR loop for $N = 1{,}000$ iterations. We also said you did not need to implement an algorithm to find the shortest path. From S to G. Rather, your code should have generated a plot of the environment depicted below and overlay the graph generated by RRT.



Due Date: 11 NOV 11:59 PM

Project:

In the first graduate student project, we are going to ask you to extend this RRT and path planning environment implementation.

Part 1 – Motion Planning Environment

You should implement a motion planning environment that is a square with four corners located at:

- (0,0)
- (100,100)
- (0,100)
- (100,0)

All units are in meters.

Next, your code should read from a file, "obstacles.txt," that gives the locations of obstacles. For simplicity, we assume the obstacles are circular, and the file gives you the center and radius of each obstacle. Each obstacle is listed as a row in the file. The example file we have given has 10 obstacles.

Your code should also read in a file, "start.txt," which will have the start location and orientation, (x_o, y_o, θ_o), of your robot.

Finally, your code should read in a file, "goal.txt," which will have the goal location, (x_g, y_g).

Part 2 – Motion Planning Algorithm

Implement RRT as your motion planning algorithm. For collision checking purposes, assume the robot is circular and has a radius of $1\ m$.

Assume the robot's dynamics are car-like, and are given by:

$$\frac{dx}{dt} = v\cos(\theta)$$

$$\frac{dy}{dt} = v\cos(\theta)$$

$$\frac{d\theta}{dt} = u$$

Assume the robot drives with a constant speed, $v = 1\ m/s$. Further, assume that your control input, u, (i.e., steering) is bounded by $u \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$.

When you implement $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$ in steer.py, you should account for:

- The orientation you achieved at $x_{nearest}$, not just the x-y position, in order to make sure you can create a continuous, smooth trajectory. As such, the orientation of your robot when you arrived at $x_{nearest}$ should be the same orientation your robot has when it starts a new path from $x_{nearest}$ to x_{new}.
- Now, perform a search over u and t that minimizes the Euclidean distance between x_{new} and x_{rand}. For simplicity, assume $t \in [0,2]$. You may use any python optimizer or search algorithm

to perform this for you provided that the appropriate python package has been white listed on Piazza. If you find a package you want to be whitelisted, please contact us via Piazza.

Remember, when checking if a path is "obstacle free," you cannot just check a straight line between the start and end of the path. Rather, you have to check along the curvilinear path described by the dynamics above. To perform this collision checking, you should discretize your path and check at each intermediate location if a collision has occurred. For this project, break up your trajectory into 100 intervals steps spread uniformly over the trajectory and check for a collision at each time step. If a collision has occurred at any of those intermediate points, then the entire path is void.

Your RRT implementation should generate N=1,000 samples.

Part 3 – Motion Plan Extraction

Now that you have your tree from running RRT, you should extract a path from the star to the goal. But which one? We will define the "optimal" path as follows:

Of all the paths that allow you to get from the start to within 5 meters of the goal, pick the one that gets you there in the shortest time.

Within path.py, you should thus find all the leaves of the tree that get you within 5 meters of the goal and then determine which one has the shortest temporal cost to traverse from the start location to that leaf. Return the associated "optimal" path.

Part 4 – Motion Plan Display

You should finally implement display.py, that is called last from main.py, to display your environment, your entire RRT tree, and then the "optimal path". Obstacles should be shown in black. Your start location should be a green circle. Your goal location should be a blue circle. The RRT tree should be purple. The optimal path should be red.

Deliverables

Python files:

- main.py – this python file should be called by the graders and run everything
- obstacleFree.py – this python file should take as input a path, your environment, and return whether the path is collision free
- steer.py – this python file should contain your steering code
- path.py – this python file should extract the "optimal" path from the RRT tree from the start to the goal.
- display.py – this python file should plot the result

PDF files:

- A single PDF with your first and last name, a title "Project #1" at the top, and a nice, cropped, screen shot of the plot from display.py for running your code on the example files.