LAB 5: PATH PLANNING

Due: Friday, April 12th 11:59 pm EST

The objective of this lab is to implement and test robot path planning capabilities, specifically the RRT algorithm. You will first implement the RRT algorithm, then apply it to drive a 2D robot in the WeBots simulation environment from start to goal configurations. In this lab, we assume that the obstacles and map are known ahead of time.

To help you out, the TAs have built a few mazes as WeBots worlds, and added code to move the robot to follow the high-level path plan your code will generate. (Movement is implemented by having the robot alternate between rotating clockwise in place when it reaches a node and driving forward along each edge to the next node.)

All the code you have to implement is in the `controllers/rrt_controller` subfolder. Specifically, look for the files `controllers/rrt_controller/map.py` and `controllers/rrt_controller/rrt.py`.

For high-level RRT pseudocode, and some useful implementation details, please see the lecture slides (specifically L20 and L21).

Part 1 – Basic RRT

Please complete the following helper functions in `controllers/rrt_controller/map.py`.

- is_inbound()
- is_inside_obstacle()
- node_generator()
- step_from_to()

Each of these helper functions is designed to solve a specific step of the basic RRT algorithm laid out in lecture. Some helper functions require using other helper functions to solve. See the docstrings for details on the expected behavior and return types, and see the lecture slides for a more detailed explanation of the algorithm.

To test these helper functions, we have given you a local autograder. This should be much faster to run locally than on Gradescope. To run the autograder, navigate to the `controllers/rrt_controller` subfolder, then run `python3 autograder.py helpers`

Afterwards, complete the `RRT` method in `rrt.py`, which handles the main loop of the algorithm. We provide code to limit the maximum number of nodes generated and terminate the loop once a path to goal has been found. Use your helper functions to implement the rest of RRT. Again, you can test this locally with `python3 autograder.py rrt`

Once the above steps are complete, you can validate that your RRT algorithm works with the WeBots simulator. (More detailed steps are on the next page.)

1. Open rrt_controller.py and **ensure that the MAP_NAME constant at the top is set to the map you would like to test on**. The maps are named maze1, maze2, and maze3 (e.g. "`./maps/maze2.json`"). A working implementation should work on all three maps.
2. Launch WeBots. Click File > Open World, then navigate to the "worlds" subfolder. Open the `.wbt` file which matches the MAP_NAME from step 1.
3. A GUI window should appear which shows your RRT code running in a simple 2D approximation of the world to find a path. Then, the robot in WeBots should start moving along this path.
    a. We recommend selecting the "DEF e-puck Robot" object via the sidebar in the top-left of the WeBots screen. This will allow you to see the coordinate frame of the robot while it moves through the maze.
    b. Note that the 2D GUI window should automatically close after the robot is done moving.

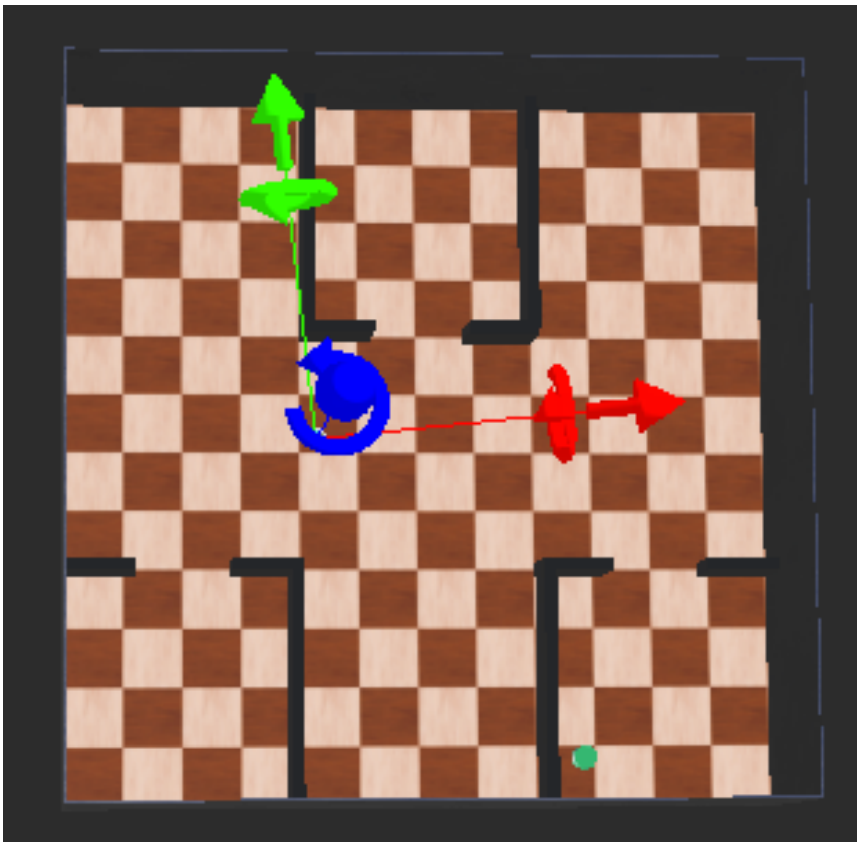

Figure 1. The WeBots world maze2 with the coordinate frame of the e-puck robot selected.

Part 2 – Path Smoothing

You may notice that the RRT algorithm can return very convoluted paths which are not suitable for smooth motion on a real mobile robot. Improve the performance by implementing path smoothing in `compute_smooth_path` method in `map.py`. You should use what you've learned in class to optimize the path by reducing the number of the nodes in the path. (See L21.)

Once again, you can test your implementation with the WeBots simulator, which will follow the same procedure detailed above.

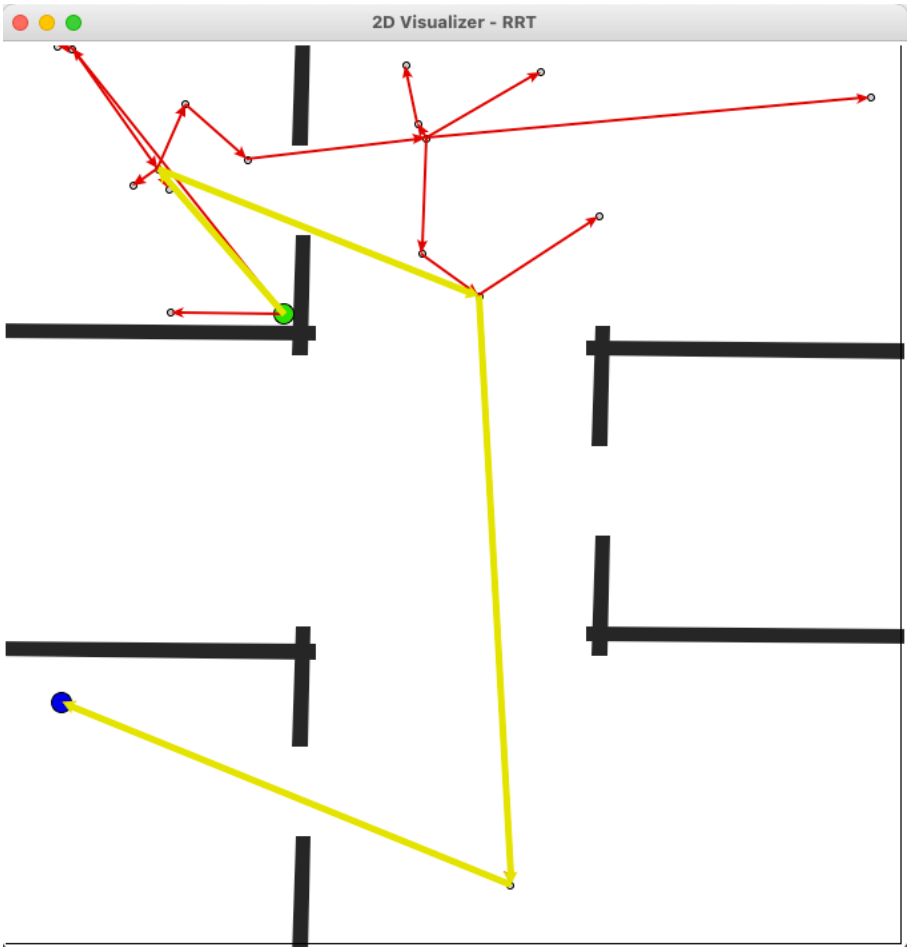

Figure 2. An example path generated using the RRT algorithm and smoothed with the path-smoothing algorithm outlined in lecture, on the world maze2.

To test this part, you can use the same local autograder. This should be much faster to run locally than on Gradescope. To run the autograder, navigate to the `controllers/rrt_controller` subfolder, then run `python3 autograder.py smoothing`

Grading: Your grade will come from the autograder running your code on the three maps you are given, plus three **hidden** maps which are different than the hidden ones.

The exact point breakdown is as follows:

Helpers	
`is_inbound()`	5 pts
`is_in_obstacle()`	5 pts
`step_from_to()`	10 pts
Full RRT implementation	
RRT, autograded with 6 maps (3 hidden), 10 points per solved map	60 pts
Path Smoothing	
Path smoothing, autograded with unsmoothed trajectories given	20 pts

Submission: Submit the files `rrt.py` and `map.py` to Gradescope. If you relied significantly on any external resources to complete the lab, please reference these in the submission comments.