LAB 4: PATH PLANNING

Due: Thursday, October 26th 11:59 am 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 simulation environment from start to goal configurations, while accounting for obstacles in real-time.

Part 1 – RRT Implementation

We have provided the following files to assist in development and debugging:

`utils.py`: helper functions, as well as the definition of the Node class.

`robot_sim.py`: simulator of a differential drive robot.

`map.py`: representation of the map and c-space. Features include start location, goal location, obstacles, and path storage. Map configuration is loaded from JSON file (located in `maps` folder) supplied at object creation.

`gui.py`: the visualizer.

`rrt.py`: implementation of RRT path planning.

`autograder.py`: used to grade the RRT solution in simulation.

Step 1: Complete the `node_generator` method in `map.py`. This method should return a randomly generated node, uniformly distributed within the map boundaries. Make sure to check that the node is in free-space using the provided methods `is_inbound` and `is_inside_obstacle`. Additionally, to drive the RRT towards the goal, the goal location is returned instead of a random sample with 5% chance.

Step 2: Complete the `step_from_to` method in `map.py`. This method takes as input two nodes and a limit parameter. If the distance between the nodes is less than the limit, return the second node. Else, return a new node along the same line but limit distance away from the first node.

Step 3: Complete the `RRT` method in `rrt.py`. Complete the main loop of the algorithm by generating random nodes and assembling them into a tree in accordance with the algorithm. Code for goal detection, tracking parents, and generating the path between the start and end nodes is already provided within `map.py`.

Step 4: Once the above steps are complete, validate that your RRT algorithm works. We have provided several maps for testing purposes, located in the `maps` folder. You can run the algorithm on one map with a graphical visualizer by executing `python rrt.py` (you can change the map in the main method at the bottom of the file).

You can test your implementation by executing `python3 autograder.py test1`.

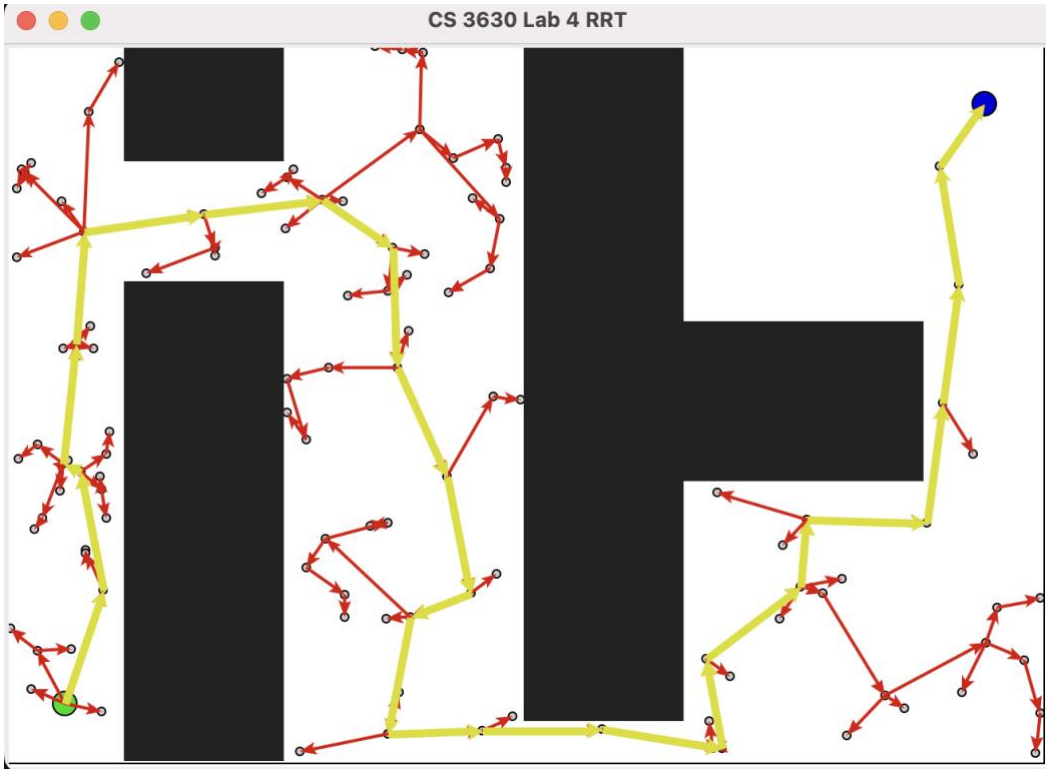

Figure 1. An example path generated using the RRT algorithm (without path smoothing).

Part 2 – Path Smoothing

You may notice that the RRT algorithm can return very convoluted paths which are not suitable for execution. Improve the performance by implementing path smoothing in `compute_smooth_path` method in `map.py`. You will use what you've learned in class to optimize the path by reducing the number of the nodes in the path.

You can test your implementation by executing `python3 autograder.py test2`.

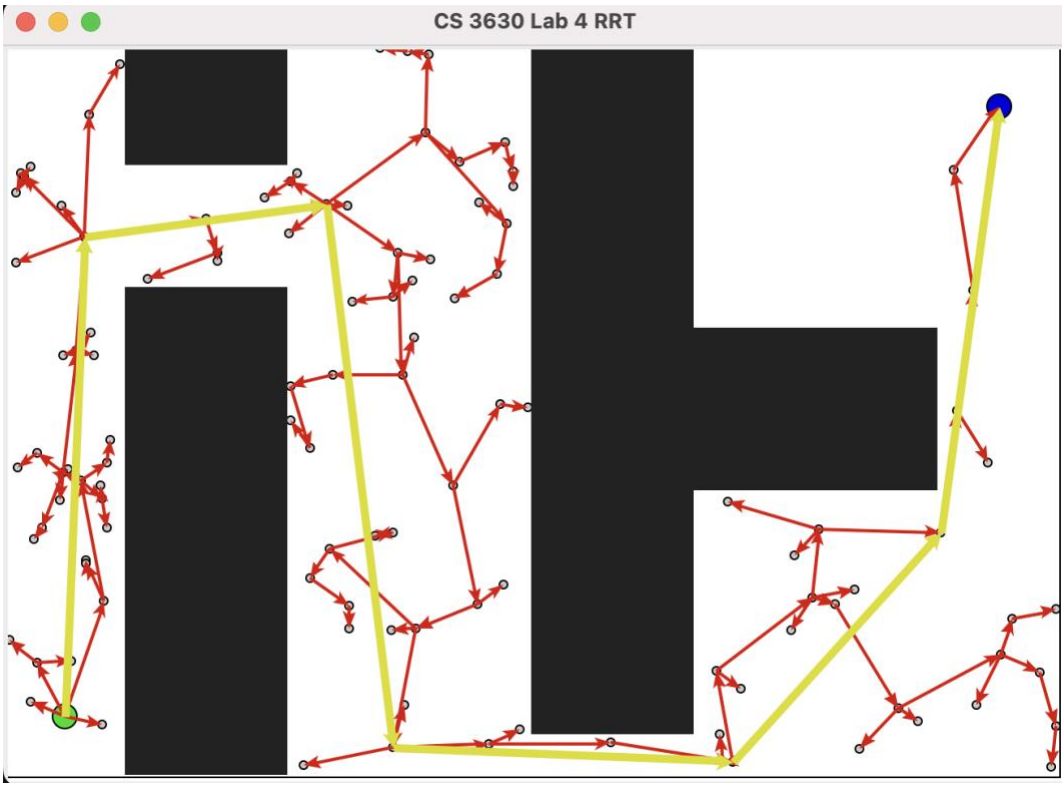

Figure 2. An example path generated using the RRT algorithm (with path smoothing).

Part 3 – RRT Planning in Simulation with Real-Time Exploration

Now, you can use the RRT algorithm to drive the robot to the goal point in exploration mode. You will complete the `robot_planning_with_exploration` method in `rrt.py`. This method is executed by the `RobotThread` and should contain all your robot behavior.

The robot starts without any knowledge about the obstacles in the map. Therefore, the first planed path is likely a straight line from the start to the goal. As the robot executes the path, it will start to see obstacles when they are within visible distance to the robot. Whenever an observed obstacle that blocks the current path, you need to re-plan (reset the map by using the current robot location as the start point and run RRT again) a new path with the updated map, and then follow the new path. You will repeat the process until you reach the goal. Note that you can use the `check_new_obstacle` method provided in `map.py` to detect new obstacles and update the map. To maintain safety execution, you should only drive the robot for its visible distance before calling `check_new_obstacle` again. You can use methods `turn_in_place` and `move_forward` in `robot_sim.py` to drive your robot, and the method `reset` in `map.py` to reset the map with a new starting location.

To run your code, execute `python3 rrt.py -explore`.

You can test your implementation by executing `python3 autograder.py test3`.

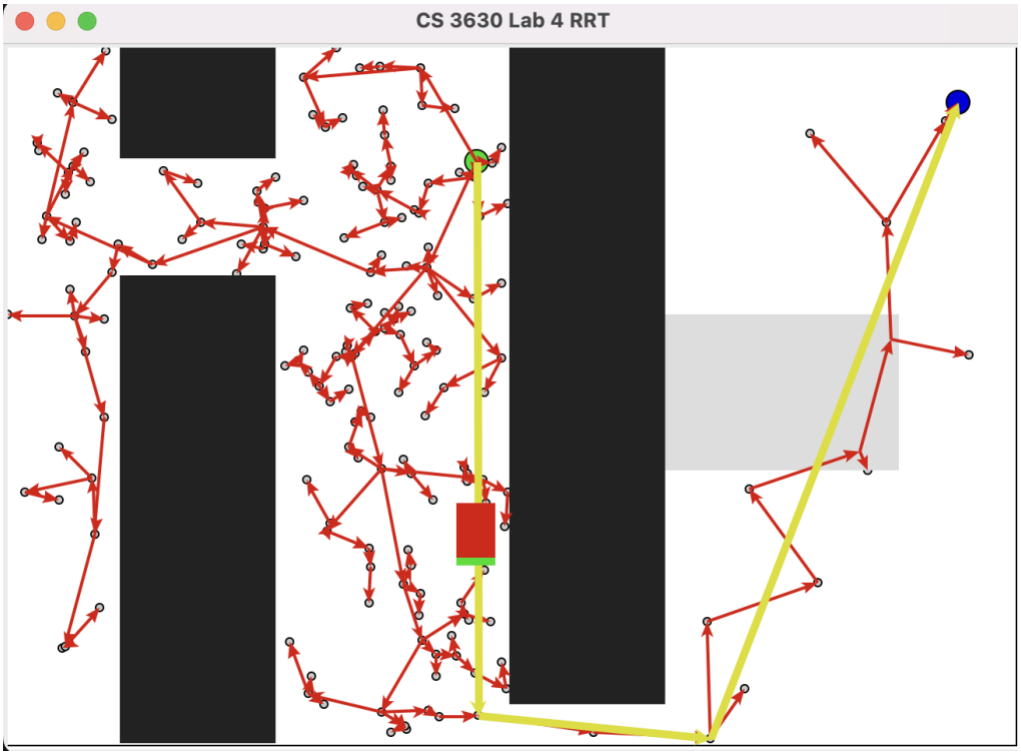

Figure 3. An example instance while the robot performs planning with exploration. The observed obstacles are shown in black, while the unobserved obstacles are shown in gray.

Grading: Your grade will come from the autograder running your code on new maps in simulation. The exact point breakdown is as follows:

Part 1 – RRT implementation:	
RRT, autograded with 6 maps, 10 points per solved map	60 pts
Part 2 – Path Smoothing:	
Path smoothing, autograded with 3 unsmoothed trajectories, 5 points per solved trajectory	15 pts
Part 3 – RRT with real-time exploration:	
RRT, autograded with 2 maps, 12.5 points per solved map	25 pts

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