LAB 5: PATH PLANNING

Due: Thursday, November 5th 11:59 am EST

The objective of this lab is to implement and test path planning capabilities for Cozmo/Vector, specifically the RRT algorithm. To facilitate easier debugging when coding your RRT, we have broken this lab into two parts, a simulated portion that will have an autograder, and a lab demonstration with Cozmo/Vector. Cozmo/Vector's goal during this demonstration will be to navigate to a target point, while accounting for obstacles in real-time.

Part 1 – Simulation

For the simulation component, we have provided the following files to assist in development and debugging:

`utils.py`: contains general methods you may find of use, as well as the definition of the Node class.

`cmap.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

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

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

Step 1: Complete the `node_generator` method in `rrt.py/rrt_vector.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. Additionally, implement your code such that with a 5% chance the goal location is returned instead of a random sample.

Step 2: Complete the `step_from_to` method in `rrt.py/rrt_vector.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 `get_global_node` method in `rrt.py/rrt_vector.py`, which is a helper function used by `detect_cube_and_update_cmap` function. This method takes as input the angle (in radians) and origin of the local coordinate frame in terms of the global frame, as well as the a node (x,y) coordinate in this local frame, and returns the node in the global frame. Remember from previous lectures that a point transformed from one frame to another follows the formula below, where theta and tx and ty would correspond to the angle and origin of the local

coordinate frame in terms of global coordinate frame, and px and py are the node (x,y) coordinate in this local frame:

$$ {}^{3}p = {}^{3}T_{1}\,{}^{1}p = \begin{bmatrix} {}^{3}R_{1} & {}^{3}t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{1}p \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -\sin\theta & {}^{3}t_{x} \\ \sin\theta & cos\theta & {}^{3}t_{y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{1}p_{x} \\ {}^{1}p_{y} \\ 1 \end{bmatrix} $$

Step 4: Complete the `RRT` method in `rrt.py/rrt_vector.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 cmap.py.

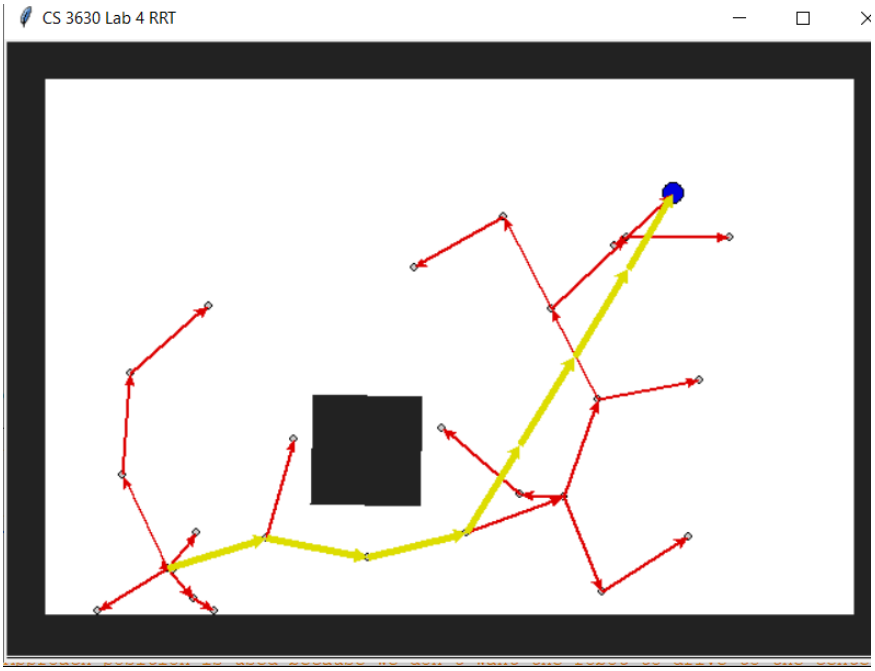
Step 5: Once the above steps are complete, validate that your RRT algorithm works. We have provided several maps for testing purposes, located in the `maps` folder. You can run the algorithm on one map with a graphical visualizer by executing `python rrt.py/rrt_vector.py` (you can change the map in the main method at the bottom of the file), or you can run the algorithm on multiple maps at once without the visualizer by executing `python autograder.py gradecase2.json`.

Step 6: Now RRT should be working, but will return very convoluted paths. Improve performance by implementing path smoothing in `get_smooth_path` method in `cmap.py`.

Note that you may edit provided functions such as `detect_cube_and_update_cmap` if you would like, but shouldn't need to.

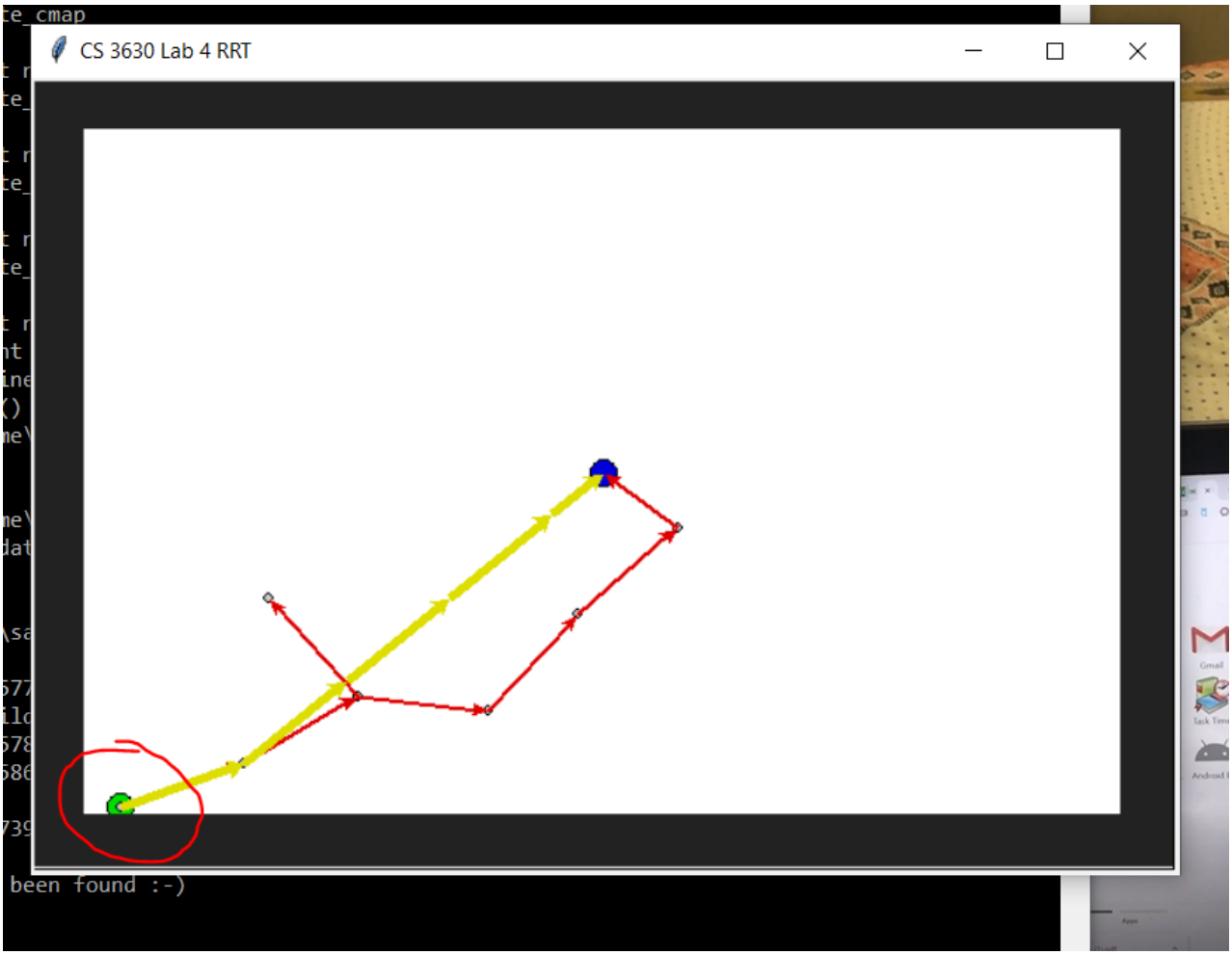
Part 2 – Cozmo

For the Cozmo/Vector portion you will complete the `CozmoPlanning` method in `rrt.py` and `VectorPlanning` method in `rrt_vector.py`. This method is executed by the `RobotThread` and should contain all your Cozmo behavior, while you can implement the Vector behavior within `VectorPlanning`. Your implementation should use RRT to find a path to a specific predetermined target point, follow the path found by RRT, and re-plan (reset and run RRT again) to avoid an obstacle cube that is added during navigation. Once a cube is placed down and becomes visible to cozmo, you can assume it will not move. Note that the `detect_cube_and_update_cmap` method has been provided in `rrt.py/ rrt_vector.py`. Also note you are **NOT** allowed to use the go_to_pose() function as it does its own path planning.



Example View of RRT path generated around a cube obstacle.

To test your code with the robot, run `python3 rrt.py -robot`. Cozmo's starting position (in mm) should be (50, 35, 0) ~ (x, y, theta) in the code/simulated arena as circled in red below, so its very close to the bottom left corner of the arena. The destination position should be the center of the arena (map_width*1/2, map_width*1/2, any) ~ (x, y, theta), and the robot should roughly reach it. You are not required to use the arena, but it you don't use it, please make sure to mark where the center of the arena would theoretically be.



As cozmo moves along the path, place a cube in the path. Note that the cube might not be detected unless the robot is directly facing the cube.

You will submit a video that meets the following requirements:

1) The video will show the GUI so we can see the rrt paths generated, as well as show the robot navigating, so we can see whether the the robot is following the rrt path, and whether the robot detects the cube and goes around it. An example video is ExampleOfVideoCS3630Lab5.mov with the center of the arena marked by a white piece of paper (https://drive.google.com/file/d/18ki6tRPWwZgq8ZfLK86zrROzwERW0pxt/view?usp=sharing).
2) The steps of the demo are:
    a. You run the `python3 rrt.py -robot/ python3 rrt_vector.py -robot` command, with no cube obstacle. The robot starts following the path.
    b. You place an obstacle cube in the robot in the robot path. When the robot encounters the cube, the robot re-plans its RRT path so that it will go around the obstacle.

c. Robot roughly reaches the center of the arena, or if you didn't use an arena, it reaches a marking indicating the center of the arena.

Grading: A portion of your grade will come from the autograder running your code on new maps in simulation. The remainder of your grade will be based on a recorded video. The exact point breakdown is as follows:

Part 1 – Simulation:	
RRT, autograded with 6 maps, 10 points per solved map	60 pts
Part 2 – Cozmo: Video submission	
The robot follows the path found by RRT roughly	10 pts
The robot roughly reaches target point	5 pts
The robot follows a smoothened path	5 pts
The robot re-plans to avoid obstacle cubes	15 pts
The map is updated to reflect newly seen cubes	5 pts

Submission: Create a zip file that includes, `rrt.py`, `cmap.py`, the video of the demo, and an image of your autograder output using `gradecase2` as input. Make sure you enter the names of both partners in a comment at the top of the files. Make sure the files remain compatible with the autograder. Name the zip file `lastname1firstname1_lastname2firstname2.zip`. Only one partner should upload the file to Gradescope. If you relied significantly on any external resources to complete the lab, please reference these in the submission comments