CSC477 – Introduction To Mobile Robotics

assignment 2, 15 points
due: Oct 28, 2020, at 3pm

Course Page: http://www.cs.toronto.edu/~florian/courses/csc477_fall20

Overview: In this assignment you will implement path planning algorithms, such as A* and RRT. You will also formulate an LQR problem.

1 A* implementation (5pts)

Implement the A* algorithm for an omnidirectional robot on a 2D plane. You are given starter code that implements Dijkstra's algorithm in Python at the following repository: https://github.com/florianshkurti/csc477_fall20, under the directory `path_planning_and_control_assignment/python`. You need to modify the function `plan()` in the file `astar_planner.py`. You can run this file as follows:

```
cd path/to/repo/path_planning_and_control_assignment/python/
python astar_planner.py ../worlds/map.pkl
```

What you need to submit: 3 images of paths produced by your planner. Use the same starting state that is currently provided, and 3 distinct destination states that are far from each other. Your images should be named `astar_result_[0|1|2]_firstname_lastname.png` Also, for each of the 3 destination states provided above, plot (a) an image of the visited states using Dijkstra's algorithm and (b) an image of the visited states using A*. Your images should be named `dijkstra_visited_[0|1|2]_firstname_lastname.png` and `astar_visited_[0|1|2]_firstname_lastname.png`.

2 RRT implementation (5pts)

Implement the RRT algorithm for an omnidirectional robot on a 2D plane. You are given starter code that implements some of the RRT functionality. You need to modify multiple functions which are annotated with TODOs in the file `rrt_planner.py`. Note that this version of the RRT is the simplest version to implement in the sense that we are not requiring kd-tree-based nearest neighbor queries and complicated collision queries. We use occupancy grids to simplify collision detection. Once you are done implementing the required functionality you can run this file as follows:

```
cd path/to/repo/path_planning_and_control_assignment/python/
python rrt_planner.py ../worlds/map.pkl
```

What you need to submit: 3 images of paths produced by your planner. Use the same starting state that is currently provided, and 3 distinct destination states that are far from each other. Your images should be named `rrt_result_[0|1|2]_firstname_lastname.png`

3 LQR (5pts)

Recall the example of the double integrator system with friction, or curling stone, that we saw in class:

$$m\ddot{\mathbf{p}} = \mathbf{u} - \alpha\dot{\mathbf{p}} \tag{1}$$

where α is the friction coefficient, $\mathbf{p}$ is the 2D position vector of the stone, and $\mathbf{u}$ is the external control applied to the stone. You are given two curling stones of equal mass m. They start from different starting positions, with different starting velocities. You are tasked with finding an LQR controller/policy that receives feedback on the joint state of the two curling stones and outputs command vectors $\mathbf{u}_1$ and $\mathbf{u}_2$ so

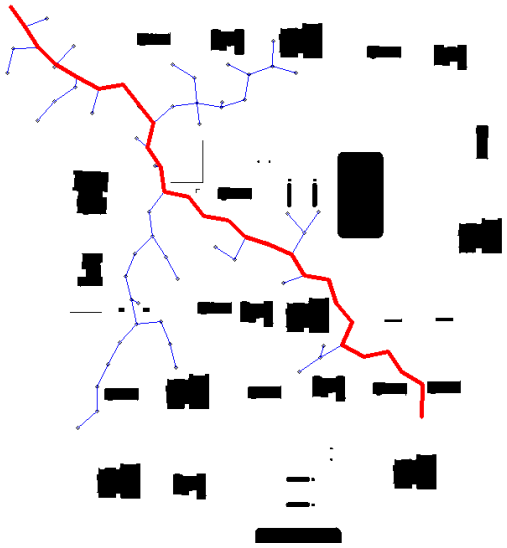

Figure 1: How the RRT planner will look like once you implement it

that the two stones end up very gently hitting each other and not bouncing away from each other. In other words, define a joint linear system

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t \tag{2}$$

and an instantaneous cost function

$$g(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_t^T Q \mathbf{x}_t + \mathbf{u}_t^T B \mathbf{u}_t \tag{3}$$

such that when the state $\mathbf{x}_t$ stabilizes around $\mathbf{0}$ the stones are touching each other and their individual velocities are also stabilized around $\mathbf{0}$.

What you need to submit: a file called lqr.pdf with the definition of the matrices A, B, Q, R, as well as the steps you took to arrive at your formulation.

4 How to submit

Similarly to assignment 1, you will submit all your work in a file called `path_planning_and_control_assignment.zip` which will contain your extensions to the provided starter code, as well as the six images and the pdf file.