CSCI 4302/5302: Advanced Robotics
Homework 3: due 15. February, 11:59 p.m. to GitHub and Moodle.

Learning objective: Using a rapidly-exporing random tree (RRT) for motion planning.

You will be implementing a Rapidly-Exploring Random Tree (RRT) for a 2D world. RRTs have a single parameter to choose—the length by which to grow the tree in each iteration. By default this is 10 for the provided code.

Things to implement:

```
rrt.m  -- RRT motion planner for the 2D grid world
nonholonomicrrt.m  -- RRT motion planner for a non-holonomic system (car)
distance.m  -- Computes distance between two "car" poses. Used in nonholonomicrrt.m
```

Tools:

```
runtest.m -- Run script for the 2D grid world - for all planners
collcheckstline.m -- Collision checks states on a straight line between two
2D points for the current map
checkLimits.m -- Checks if the 2D point robot is within map limits
bresenham.m -- Implements Bresenham's line algorithm. You do not have to use this directly
map1.txt -- Simple environment for testing the 2D planners
map2.txt -- Bigger environment for testing the 2D planners
You will find these functions useful for the non-holonomic RRT. All these (except
the first two) are defined inside nonholonomicrrt.m:
nonholrrtdata.mat -- Map and Robot data for the non-holonomic system
angdiff.m -- Computes angle differences. Returns values in -180 to +180 degrees
simulate_carBot -- Simulate the non-holonomic robot forward with given state & control
checkCollision_carBot -- Collision checker for the car
checkLimitViolation_carBot -- Checks for limit violations
plotCarBot -- Plot the car bot at a given pose
plotcircle -- Plots a circle with given center and radius
```

You may end up not using every single file. Some are utilities for other files, and you don't really need to bother with them. Some have useful utilities, so you won't have to reinvent the wheel. Most have fuller descriptions in the files themselves.

- You can extend the graph along the straight line from nearest graph state to the chosen random sample. The function `collcheckstline` can help you for this.
- The parameter `deltastep` sets the length of expansion (in terms of number of cells on the map). Play with different values to see how the RRT changes.
- You can end the graph generation once you reach a state that can be rounded off to get the goal state.
- The cost of the path can be the euclidean distance between the nodes on the path (again, `collcheckstline` can help you here).
- You are expected to display the RRT graph intermittently using the provided figures.
- Please read the comments/todo sections in the file `rrt.m` before starting your implementation.

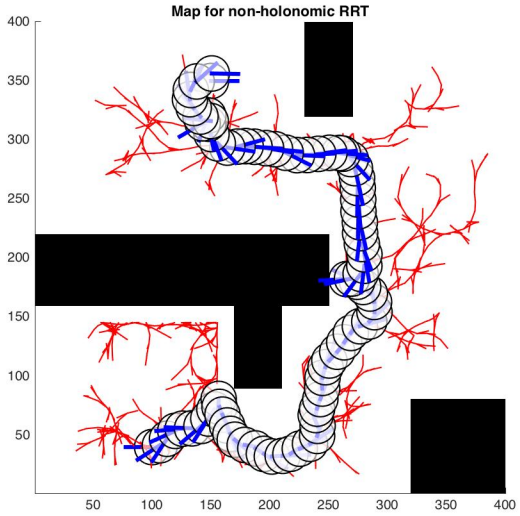
Assume we have a car-like system with non-holonomic constraints (i.e. the system cannot instantaneously move to any state from any other state). The state of the car is `[x,y,theta]` where `x, y` are the coordinates of the center of the car and `theta` is the heading. The controls that the system can apply are `v,omega` where `v` is the linear velocity of the car and `omega` the steering angle. Note that the car can move both forward & backward as well as turn right and left.

We will be sampling controls to generate possible motion "rollouts" from the graph. Given a randomly sampled state, we will find the closest graph node. We then simulate forward a sequence of randomly sampled controls from this graph node and find the one that gets us close to the randomly sampled state. The end-point of this "best" rollout is the next state we add to the graph. The code for this problem is in the function `nonholonomicrrt.m` which you will have to edit. A few points to note:

- You will have to come up with a way to sample states (5% biasing towards goal helps significantly).
- You need to implement the function `distance` which measures the distance between two car poses. This function has to be used for all distance computations for this problem. Be wary of angular distances!
- You are provided with a function for checking limit violations and collisions.
- You are also provided with a function that forward simulates controls given the current state.
- As before, you can set the length of the tree extension using the parameter `deltastep`. By default, this is 5 (which leads to simulating the motion for 5 timesteps in the future with fixed controls)
- You will have to sample controls to generate the forward motion. A simple sampling scheme is to sample uniformly between `[-maxlinearvel, +maxlinearvel]` and `[-maxsteerangl, +maxsteerangl]`, though other sampling schemes can also be used.
- Your system has to run till it gets "near" the goal. A reasonable termination criterion is to have a `x, y` distance of 7.5 cells & angular distance of 2.5 degrees. You can check to see if any intermediate states on your path (rather than just the final state) are close to the goal.

More details and pseudocode for this are in the file `nonholonomicrrt.m`. There is a single world file `nonholrrtdata.mat` for this problem. Testing the system with a `deltastep` of 5 gives the following result (which you need not match):

```
nonholonomicrrt('nonholrrtdata.mat', [100,40,pi], [150,350,0], 100, 5)
State: (150.666226, 356.398499, 6.265385) is near goal (150.000000, 350.000000, 0.000000)
```



What you turn in:

- Turn in your code project to the course GitHub. Using the **private** repository named after your IdentiKey login, create a new directory for `Homework3` and add the files for this assignment to the repo, commit them, and push your files and your repo to the server.
- Turn in a short report to the course Moodle. In the report, provide some figures of iteratively growing RRTs with the final results. Show three different restarts of the method and the solutions that result in them. Also show timing results in a histogram for 100 restarts of the algorithm.