ROB 520: Motion Planning Winter 2022
Homework Assignment #3, Due 3/9/2022 at 2:59pm

Rules:

1. **All homework must be done individually, but you are encouraged to post questions on Piazza.**
2. No late homework will be accepted.
3. **You must show all your work to receive credit for your solutions.**
4. The goal of this homework is to learn how to use RRTs for practical problems and to compare the performance of variants of RRT.
5. Submit your code along with a pdf of your answers in a zip file to Canvas.

Questions

1. (15 points) Prove that the naive random tree algorithm shown below is probabilistically complete. Assume your configuration space is a bounded subset of $\mathbb{R}^n$ and there are no non-holonmic constraints (only obstacles). *Hint*: you will need to use the following fact: When randomly sampling in $\mathbb{R}^n$, the probability of generating a sample inside an n-dimensional ball with radius greater than 0 is strictly positive.

Algorithm 1: Naive Tree Algorithm

```
1  Input : $q_{start}$ the start configuration; $q_{goal}$ the goal configuration; $p_{goal}$ probability of sampling the
      goal at an iteration;
2  InitializeTree($q_{start}$); $q_{node} \leftarrow q_{start}$;
3  while True do
4      $r \leftarrow$ Random$(0, 1)$;
5      if $r < p_{goal}$ then
6          SAMPLEGOAL $\leftarrow$ True;
7          $q_{rand} \leftarrow q_{goal}$;
8      else
9          SAMPLEGOAL $\leftarrow$ False;
10         $q_{rand} \leftarrow$ RandomSampleInCspace();
11     $v \leftarrow q_{rand}$; $e \leftarrow (q_{rand}, q_{node})$;
12     if e is not in collision then
13         Add $v$ and $e$ to tree;
14         if SAMPLEGOAL then
15             return path from $q_{start}$ to $v$;
16     $q_{node}$ = Pick random node of tree;
```

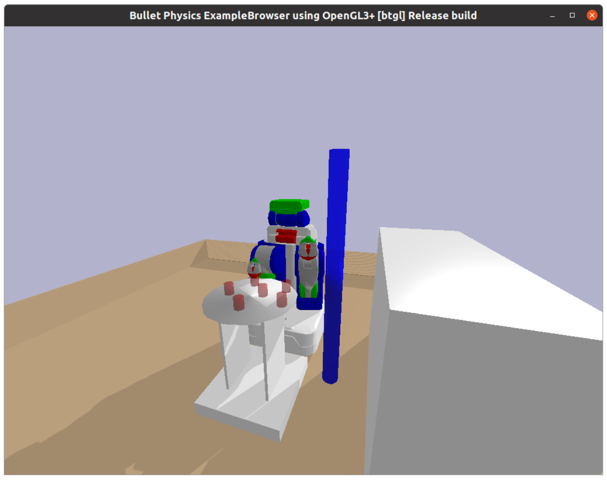
Software

1. Download and unzip `pybullet_interface.zip` (this is the same code as was used for HW1). Run `demo.py` and verify that you see the PR2 robot in a scene with a table.
2. Download the HW3 template here. Place `rrt_template.py` in the `pybullet_interface` directory and `pr2clutter.json` in the `pybullet_interface/scenes` directory.

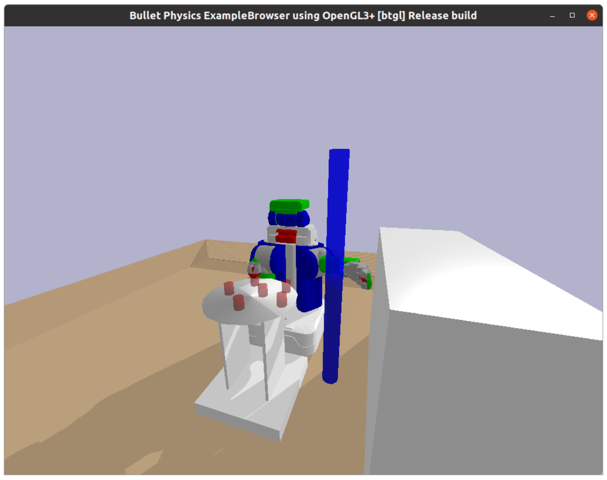
Implementation

The following implementation problems should be done starting from the python template. Only edit what is inside the `### YOUR ... HERE ###` blocks. You should implement your own code from scratch unless explicitly stated otherwise. Sharing code is not allowed.

1. (50 points) Implement the RRT-Connect algorithm to search for collision-free paths in the 7DoF configuration space of the robot's arm. The algorithm should start from the current configuration of the robot's arm and find a path to a specified goal configuration. Use `rrt_template.py` to call your algorithm.


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

(a) Start configuration


Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

(b) Goal configuration

   You will need to determine a reasonable step size to use. Your algorithm should produce a path in under **3 minutes** on average for the best goal bias (see below). We will run this exact command from the command line to test your code: `python3 rrt_template.py`. We will not run any other commands. We will run your code three times, and the code should find a path in under 3 minutes at least one of those times.

   Complete the following:

   (a) Run the RRT with goal bias ranging from 1% to 96% in increments of 5%, running the algorithm 10 times for each value of goal bias. Plot the average computation time vs. goal bias value. Explain why the plot looks the way it does and include the plot in your `pdf`. Select the goal bias that yields the lowest computation time and use that for the remainder of this problem.

(b) Once you have computed the path from start to goal, draw the position of the left end-effector of the robot for every configuration along the path in red in the viewer. You should see that the points along the path are no more than a few centimeters apart. Include a screenshot showing the path you computed in your `pdf`.

(c) If the robot touches any obstacles during execution, adjust your step size in the RRT so that this doesn't happen (you will need to redo part (a) if you change the step size).

2. (20 points) Implement the shortcut smoothing algorithm to shorten the path. Use 200 iterations. When we run `python3 rrt_template.py`, we should see the execution of the smoothed path in the viewer.

   (a) Draw the original path of the end-effector computed by the RRT in red and the shortcut-smoothed path in blue in the viewer. Include a screenshot showing the two paths for one run of the RRT in your `pdf`.

   (b) Record the path length after each iteration of the algorithm. Create a plot showing the length of the path vs. the number of smoothing iterations executed. Include this plot in your `pdf`.

3. (15 points) Run the RRT and smoothing 30 times and record

   - The computation time for the RRT
   - The computation time for smoothing
   - The number of nodes sampled
   - The length of the path (unsmoothed)
   - The length of the path (smoothed)

   (a) Create a table showing these results and compute the mean and variance for each type of data. Include this, along with a discussion explaining why the results look the way they do in your `pdf`.

   (b) Create a histogram showing how runs of the RRT are distributed in terms of computation time. Matlab's hist function is an easy way to do this. Include this histogram, along with a discussion explaining why the results look the way they do in the `pdf`.

4. (35 points) Copy the `rrt_template.py` template to a file called `birrt_template.py`. Implement the BiDirectional RRT-Connect algorithm and call it from this python file. We will run this exact command from the command line to run your code: `python3 birrt_template.py` and no others. When we run this code we should see the algorithm generate a new path, smooth it using shortcut smoothing, and execute the corresponding trajectory on the robot. The code should find a path in under `3 minutes` and we will test it as with the RRT-Connect. You will be able to re-use much of the code you wrote for RRT-Connect in your implementation.

   (a) Compare the results of BiDirectional RRT-Connect to your implementation of the RRT-Connect in terms

      - The computation time

- The number of nodes sampled
- The length of the path (unsmoothed)

Discuss why you are seeing the results that you see and include the discussion in your `pdf`.

(b) Create a histogram showing how runs of the BiDirectional RRT-Connect are distributed in terms of computation time and compare to the histogram for the RRT-Connect. Discuss the similarities (if any) and differences (if any). Include this discussion and histogram in your `pdf`.

5. Extra Credit (40 points) Copy the `rrt_template.py` to a file called `rrtstar_template.py`. Implement the RRT* algorithm. The cost of a path is its length. We will run this exact command from the command line to run your code: `python3 rrtstar_template.py` and no others. When we run this code we should see the algorithm generate a new path and execute the corresponding trajectory on the robot. Do not smooth the path. Terminate the code after 15 minutes and output the best path found (or no solution). You will be able to re-use much of the code you wrote for RRT-Connect in your implementation.

   (a) Run RRT* 30 times and produce a plot showing average path length vs. computation time. Include the plot and discuss your results in your `pdf`.

   (b) Compare the results of RRT* to your results from the RRT-Connect and BiDirectional RRT-Connect in terms of

   - The number of nodes sampled
   - The length of the path (unsmoothed for RRT and BiRRT)
   - The length of the path (smoothed for RRT and BiRRT)

   Discuss why you are seeing the results that you see and include the discussion in your `pdf`.