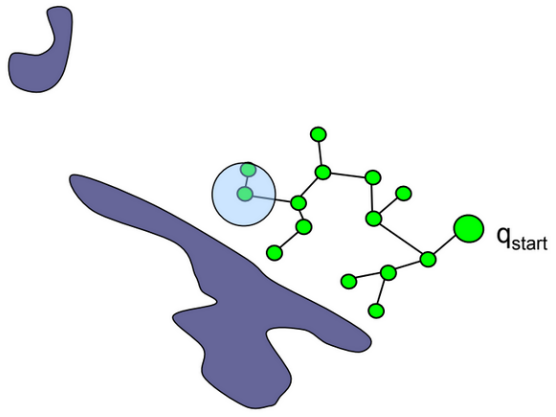
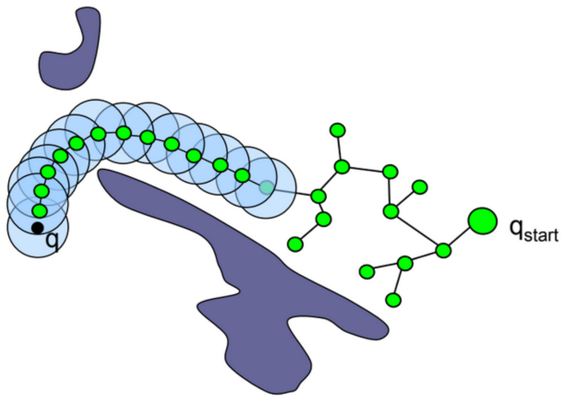
HW3

Questions

1. Prove that the naive random tree algorithm shown below is probabilistically complete.
Suppose we have a tree generated so far, like the picture used in lecture slides:



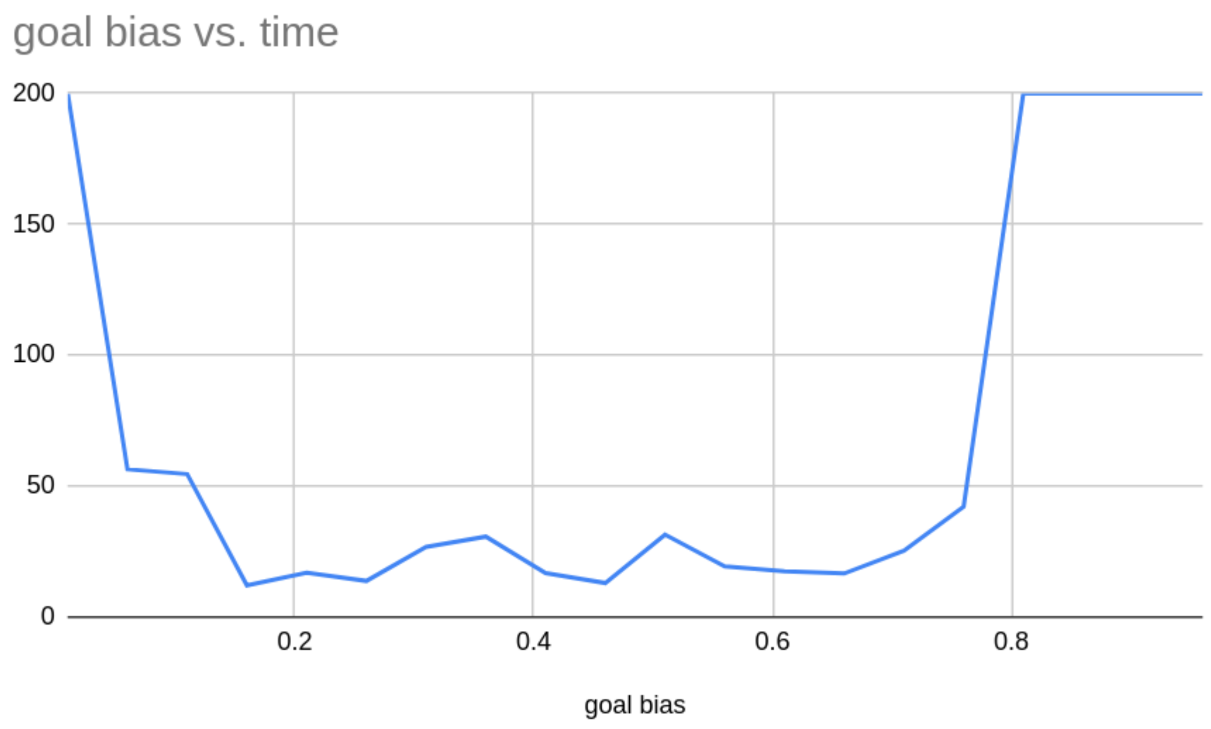
There is a small radius around the tree node. If we sample a node in that radius, and the node is collision free, then we can add it to the tree.
The probability of generating this node within the small ball is greater than 0. So if we generate infinite times, it is guaranteed that there could be a sample node inside this ball.
If there is a goal far from the tree, the goal node is in the reachable connected free space, there could also be a small ball around the tree. In fact we can create a series of balls from this node to the goal, even if there is a very narrow path, the radius of the ball could be very small, too.



Then, we could add as many sampled nodes as possible inside the small ball, to reach the goal, which means we could reach anywhere in the free space. That's proofed the naive random tree algorithm is probabilistically complete.

Implementation
1&2 see HW3/rrtplugin/rrtplugin.h and HW3/rrtplugin/rrtplugin.cpp

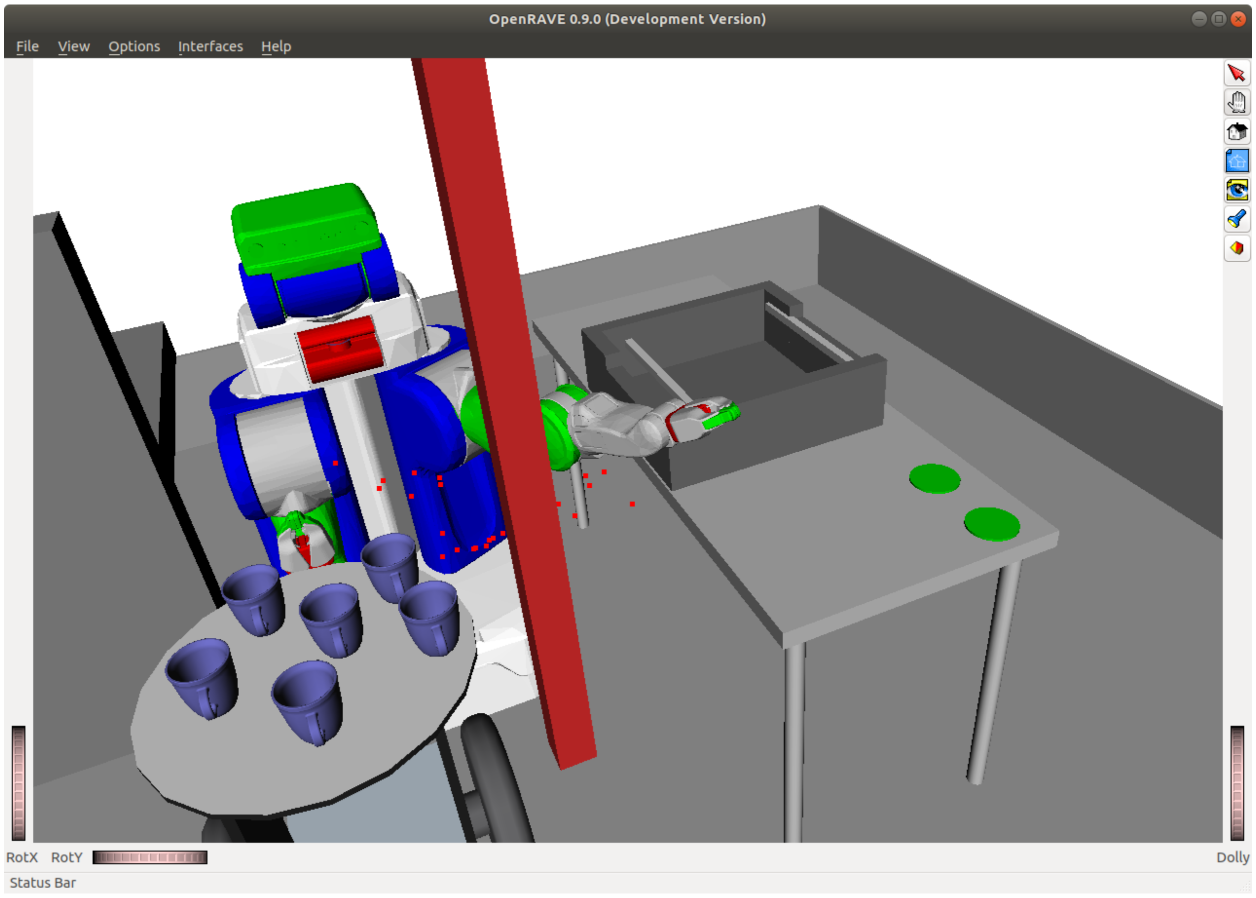
3.
(a) Run the RRT with goal bias ranging from 1% to 96% in increments of 5%, running the algorithm 10 times for each value of goal bias. Plot the average computation time vs. goal bias value. Explain why the plot looks the way it does and include the plot in your pdf. Select the goal bias that yields the lowest computation time and use that for the remainder of this problem.



The curve is like a “U”, for small goal biases like 0.01 and 0.06, the possibility towards the goal direction is small, so it would spend more than on random sampling, which could cause much more time.
As for large goal biases like 0.91 and 0.96, the possibility towards the goal direction is big, so the random node has the large possibility to be the goal. However, there is an obstacle if the robot directly moves towards the goal, so it is also hard for the robot to find the nodes to avoid the obstacle, which also costs a lot of time. If there is no obstacle in the direction to the goal, I think the time for large goal bias could be small. This scene is very tricky.

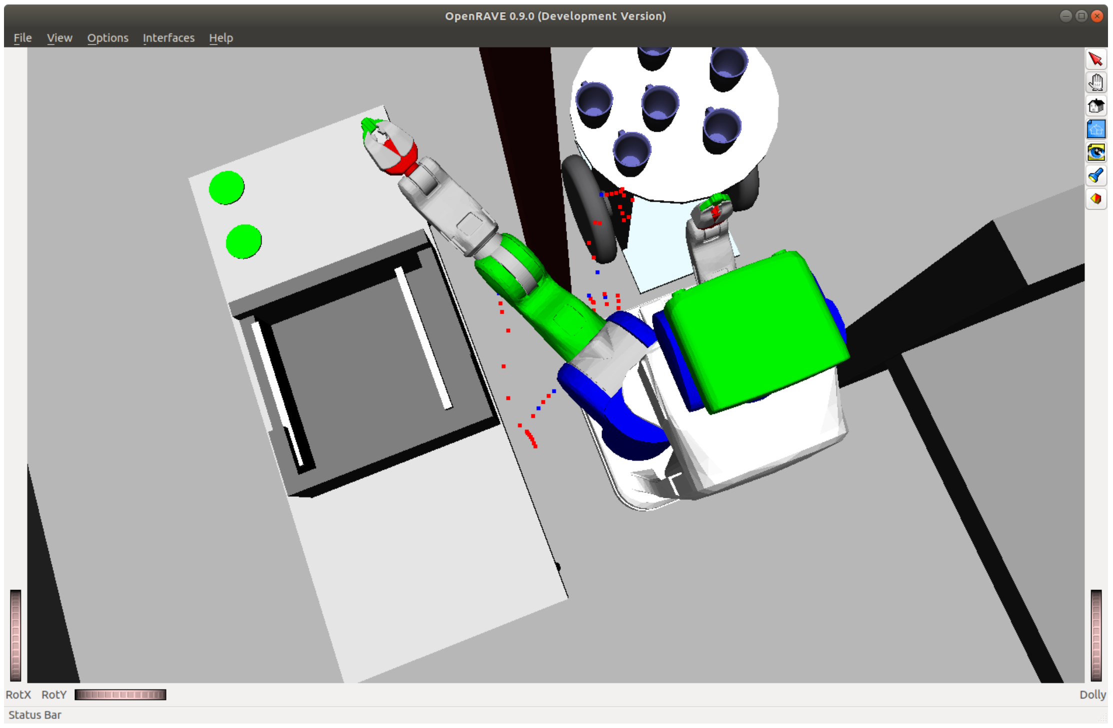
So goal biases such as 0.16 to 0.46, make a good balance between moving towards goal and random sampling to avoid obstacles, of which the computation time is smallest.

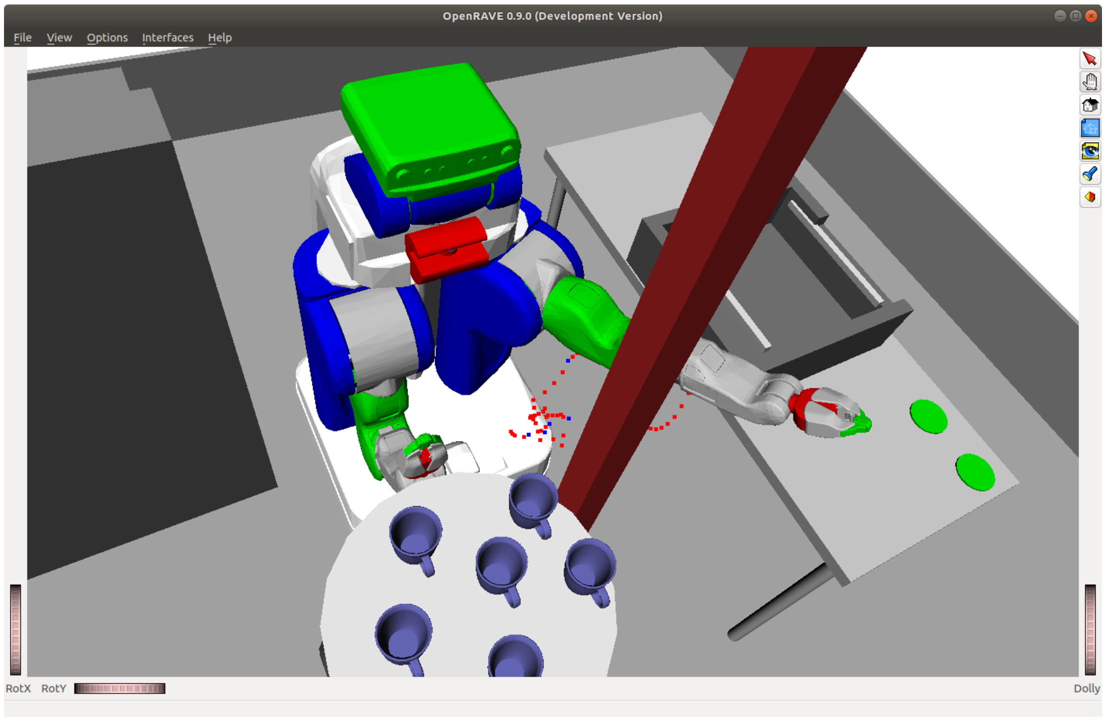
(b) Once you have computed the path from start to goal, draw the position of the left end-effector of the robot for every configuration along the path in red in the openrave viewer. You should see that the points along the path are no more than a few centimeters apart. Include a screen-shot showing the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



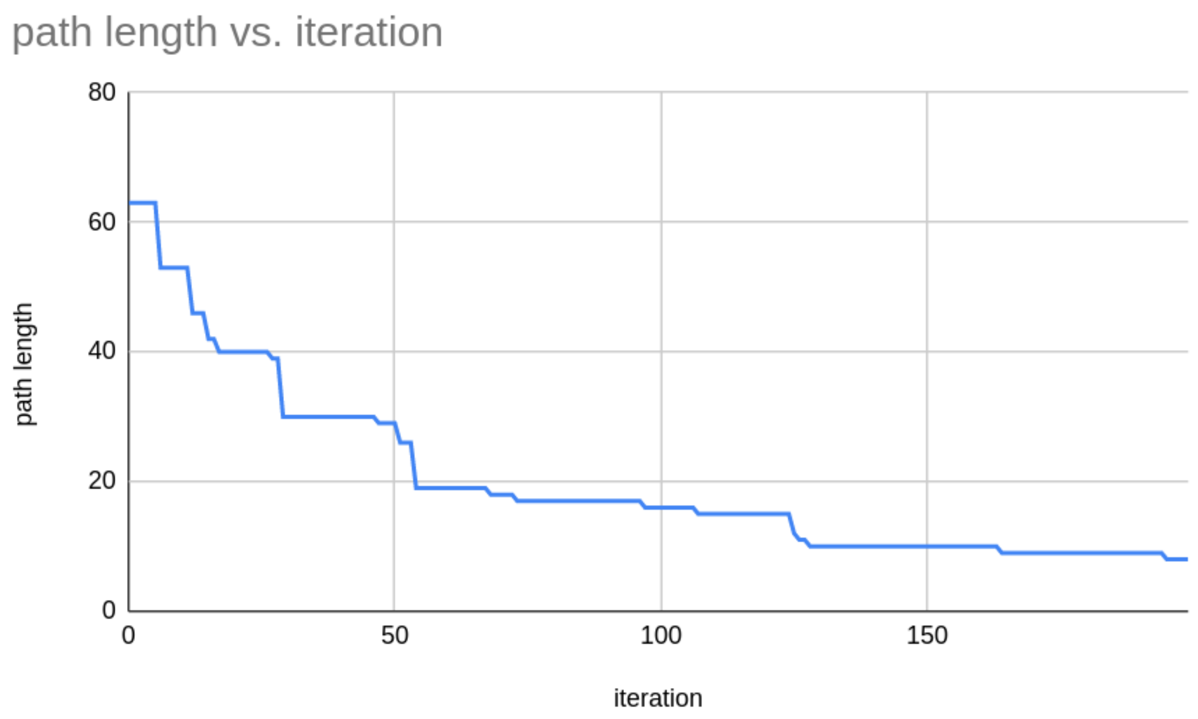
4. Smoothing

(a) Draw the original path of the end-effector computed by the RRT in red and the shortcut-smoothed path in blue in the openrave 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.

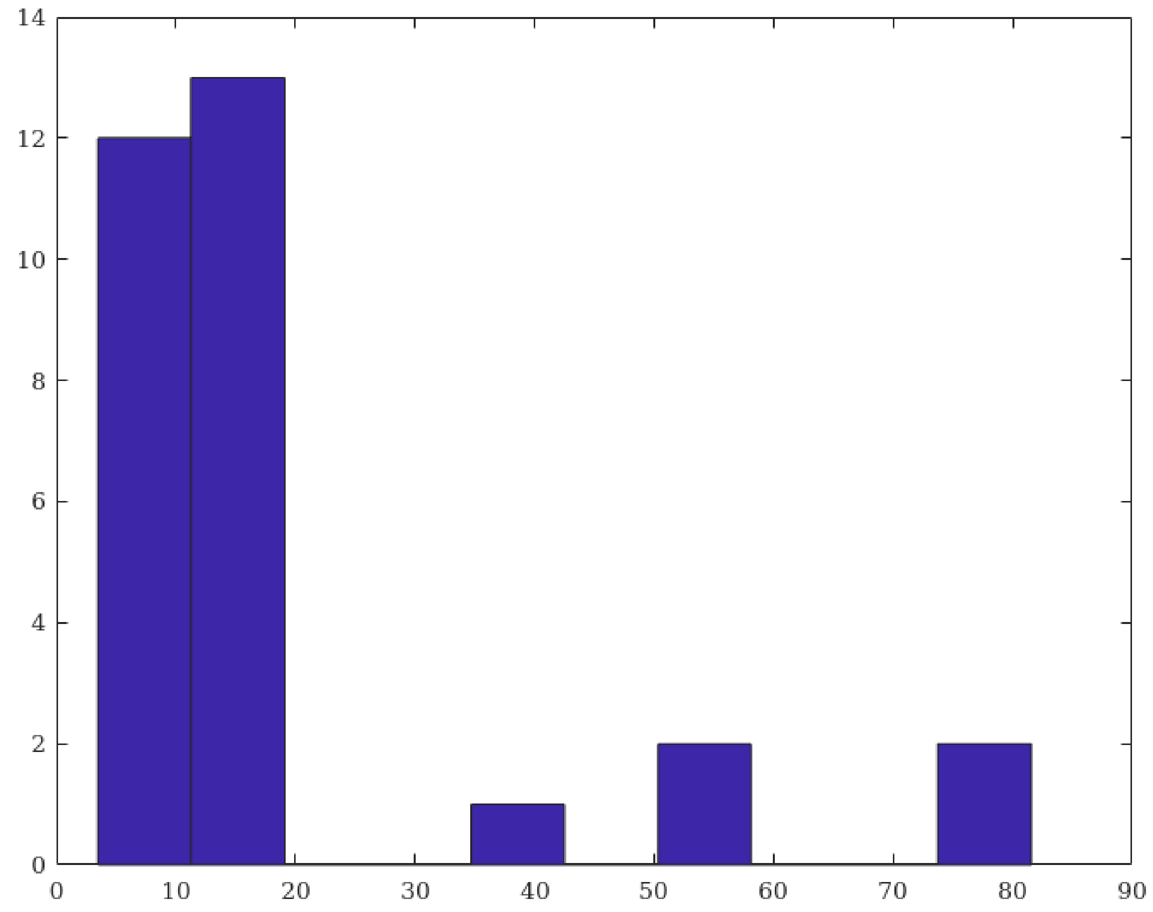


5. Run the RRT and smoothing 30 times and record
• The computation time for the RRT
• The computation time for smoothing
• The number of nodes sampled
• The length of the path (unsmoothed)
• The length of the path (smoothed)

(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.

See table.xlsx file.
The sampling part is a normal distribution, so the running time would concentrate on the first two columns. But there could be some tricky cases, if the sampling nodes are not so good for the robot to find the path, but the frequence of these cases is low.

(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.



6. BiDirectional RRT-Connect algorithm.
(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.

The results I got are:

Run time: 56.1499 seconds
Number of nodes sampled: 2042
Length of path(unsmoothed): 25.3719

The run time is larger than RRT, I think it is because I used RRC-Connect for both Tree, so it would cost much more time than RRT. It is also the reason why the number of nodes sampled is more than RRT. The length of the unsmoothed path is a bit larger than RRT, I think it is because for both Tree, after expanding a new node, another Tree would take this new node as a new target. It is good for them to meet, but for each Tree, the goal is changing compared to RRT, so the unsmoothed path would "swing", i.e. longer than the path of RRT.

(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.

The two histogram are similar because they both follow the same distribution when sampling.