# EECS 598 Motion Planning HW #3

Madhav Achar

3/11/19

## 1 Implementation

1. (10 points) Create a header file with the classes `NodeTree` and `RRTNode`

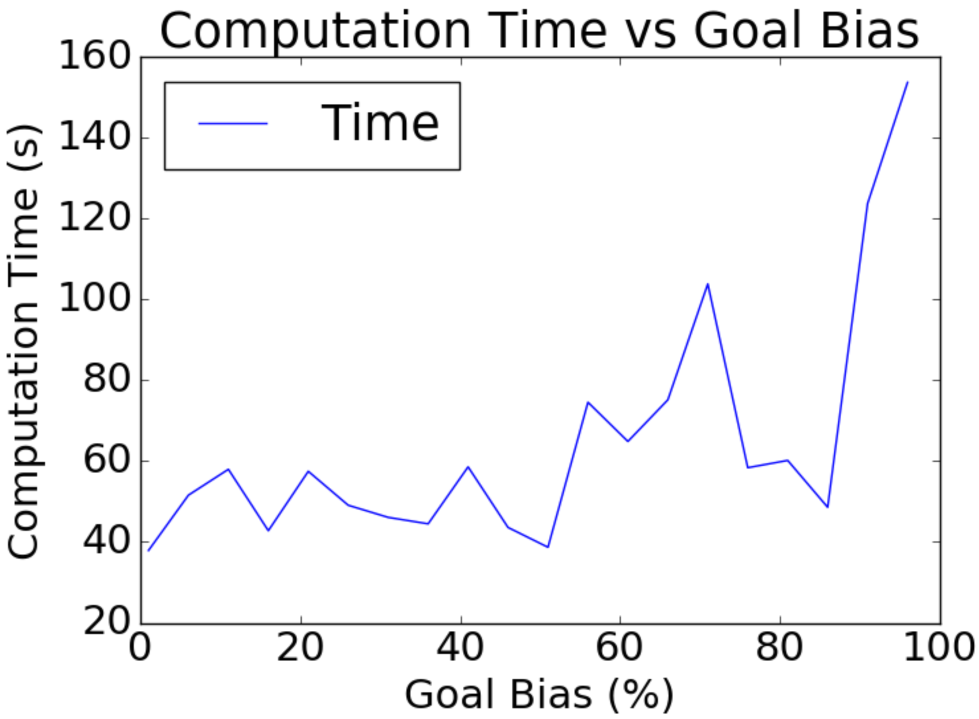
See implementation.

2. (10 points) Implement a nearest-neighbor function.

See implementation.

3. (50 points) Implement the RRT-Connect algorithm in C++

a. Plot the average computation time vs. goal bias value.



The plot shows the computation time taken for the RRT-Connect algorithm with varying percentages of goal bias. On the x-axis, the numbers represent the percentage of samples where the configuration sampled was the goal configuration. At each bias level, the RRT-Connect algorithm was run 10 times and the average computation time was plotted. While running the simulation, it was observed that there was a high degree of variability of when the algorithm was able to find a path to the goal configuration. The plot does suggest that there is a positive relationship between computation time and goal biasing especially after a goal biasing of greater than 50%. This could be due to oversampling of the goal configuration which leads to the algorithm attempting to expand the tree when no new nodes have been added since the previous attempt.

b. See Figure 1. for the unsmoothed trajectory of the RRT-Connect algorithm with a goal bias of 1% and a step-size of 0.05.

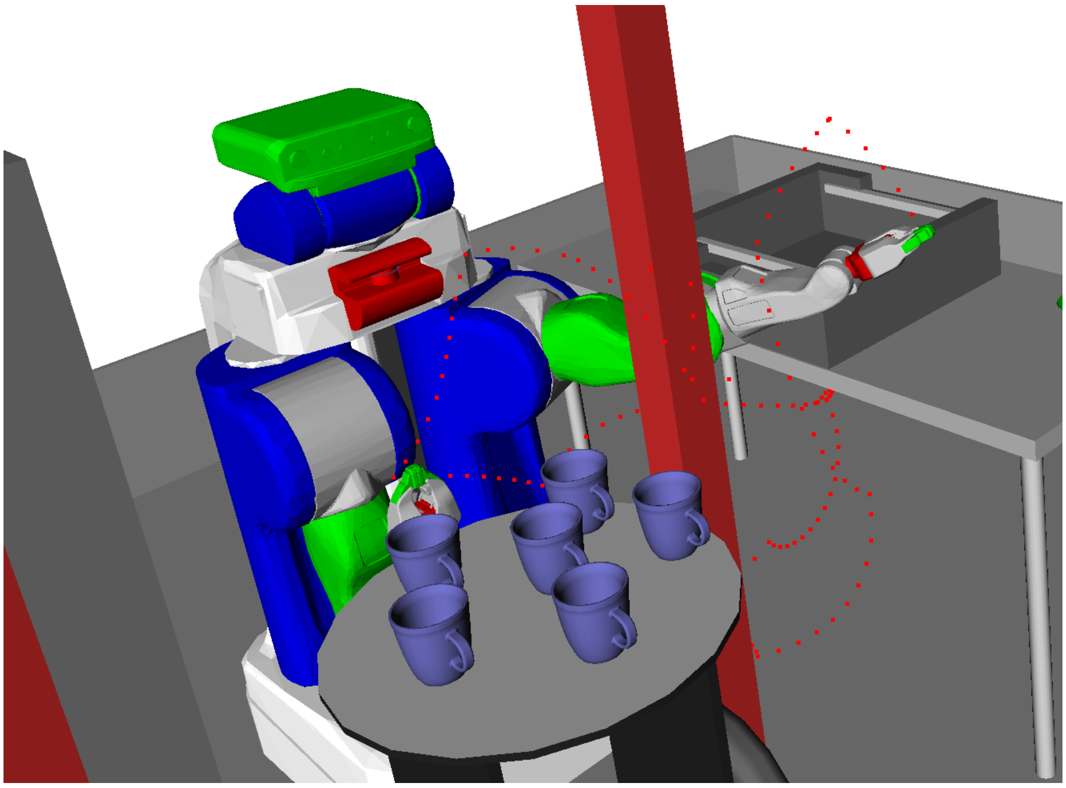


Figure 1: Unsmoothed RRT-Connect Trajectory with goal bias of 1%

c. Convert your path to a trajectory and execute it with the robots controller.

See implementation.

4. (20 points) Implement the shortcut smoothing algorithm to shorten the path. Use 200 iterations.

   a. See Figure 2.

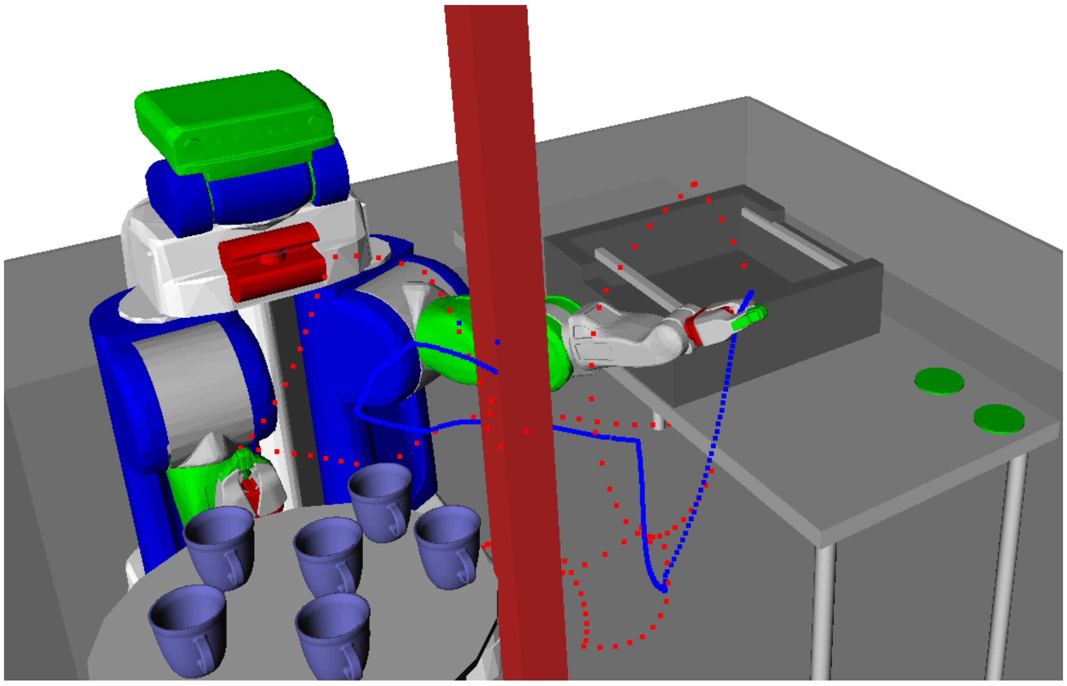


Figure 2: Red represents the unsmoothed trajectory and blue represents the smoothed trajectory

b. Create a plot showing the length of the path vs. the number of smoothing iterations executed.

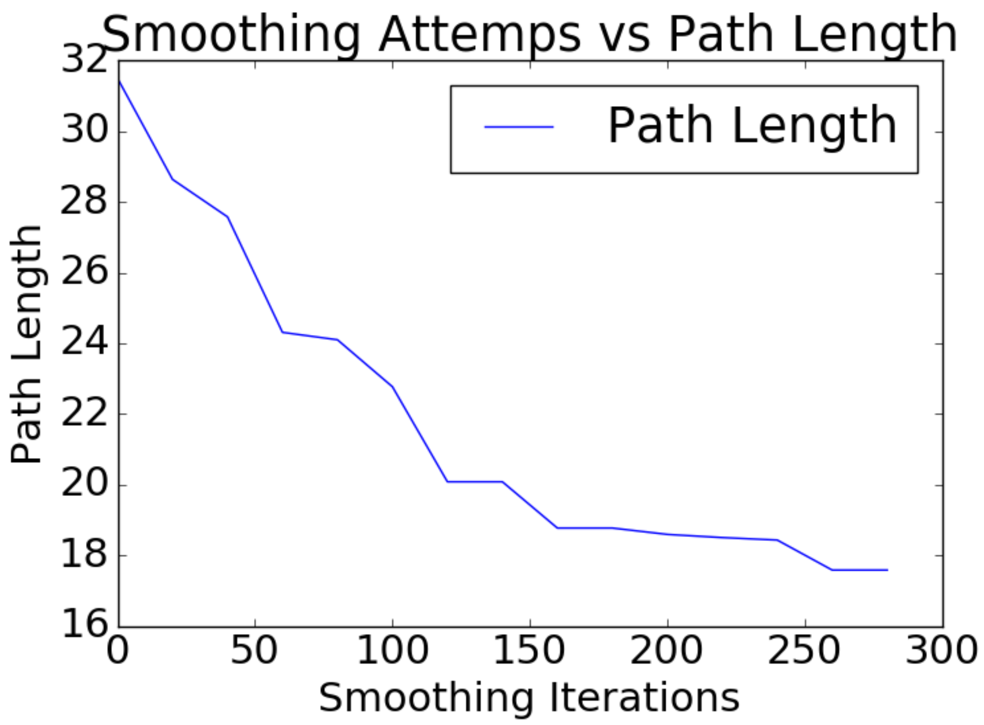


Figure 3: Length of path after successive attempts of smoothing

In Figure 3. we can see the length of the path go down as more time is spent smoothing the output path of the RRT-Connect algorithm. To generate the plot, a constant random seed was used so that calling the RRT-Connect algorithm wouldn't generate a different path.

5. (15 points) Run the RRT and smoothing 30 times.

   a. Create a table showing these results and compute the mean and variance for each type of data.

| Metric | Mean | Variance |
|---|---|---|
| Computation Time RRT (s) | 47.33 | 1023.96 |
| Computation Time Smoothing (s) | 10.73 | 4.13 |
| Samples | 2641 | 2765832 |
| Unsmoothed Path Length | 38.58 | 90.59 |
| Smoothed Path Length | 16.09 | 12.20 |

   As expected, there is a large variance in both the computation time and the number of configurations sampled. Especially considering the nature of the problem, the red pole in the environment creates a narrow passage between the start configuration and the goal configuration. Thus it highly depends on if the RRT algorithm is able sample in the narrow passage (aka a configuration where the robot can start moving around the pole). 200 iterations were used to smooth the resulting path. Smoothing consistently was able to bring down the average path length by about 22 units.

   b. Histogram

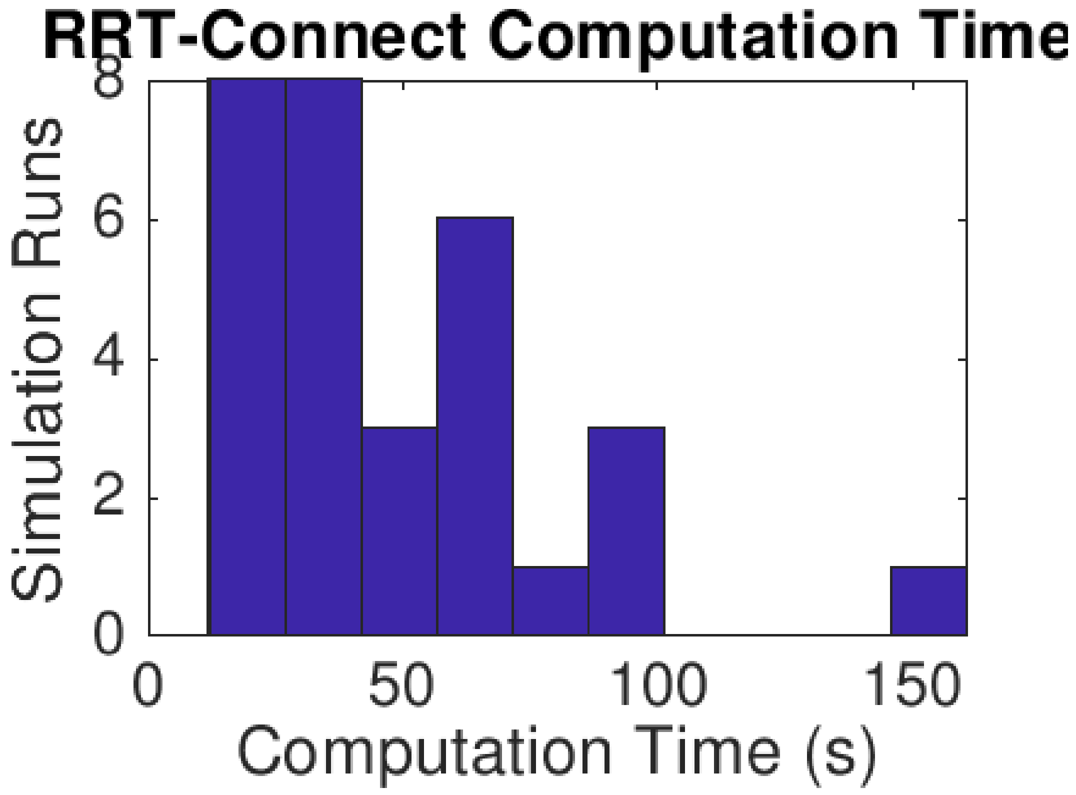


Figure 4: RRT-Connect run 30 times with a step size of 0.05 and goal bias of 10%

The histogram reflects the mean and variance calculated in the table for the RRT-Connect computation time. The RRT-Connect algorithm finds a path on average in 47 seconds. Using a smaller step size would result in increased computation time. The distribution looks similar to a chi-squared distribution.

6. (35 points) Implement the Bi-Directional RRT-Connect algorithm.

   a. Compare the results of BiDirectional RRT-Connect to your implementation of the RRT-Connect

| Metric | Bi-RRT Mean | Bi-RRT Variance | RRT Mean | RRT Variance |
|---|---|---|---|---|
| Computation Time (s) | 25.7 | 266.41 | 47.33 | 1023.96 |
| Samples | 558.83 | 143319.67 | 2641 | 2765832 |
| Unsmoothed Path Length | 27.14 | 30.11 | 38.58 | 90.59 |

   Compared to RRT-Connect with a goal biasing of 10%, Bi-Directional RRT with both trees implementing Connect is significantly faster. The results in the table are averages taken over 30 simulations. To select which tree to grow at each iteration in the algorithm, the number of nodes in each tree was compared and the tree with fewer number of nodes was selected to grow. The result is that the average computation time to arrive at a solution is approximately 22 seconds faster than the RRT-Connect algorithm.

   In addition to having an on average quicker solution time, Bi-Dirrectional RRT produces on average a shorter path as well as finds a solution with a smaller number of samples. One reason for this is that for narrow passage problems, growing two trees increases the probability of finding a solution through the narrow passage (attacking it from two directions). A possible explanation for why the path length has decreased is that because the computation time has decreased, extra nodes are not connected to the existing trees elongating future solution paths.

   b. Create a histogram showing how runs of the BiDirectional RRT-Connect are distributed in terms of computation time.

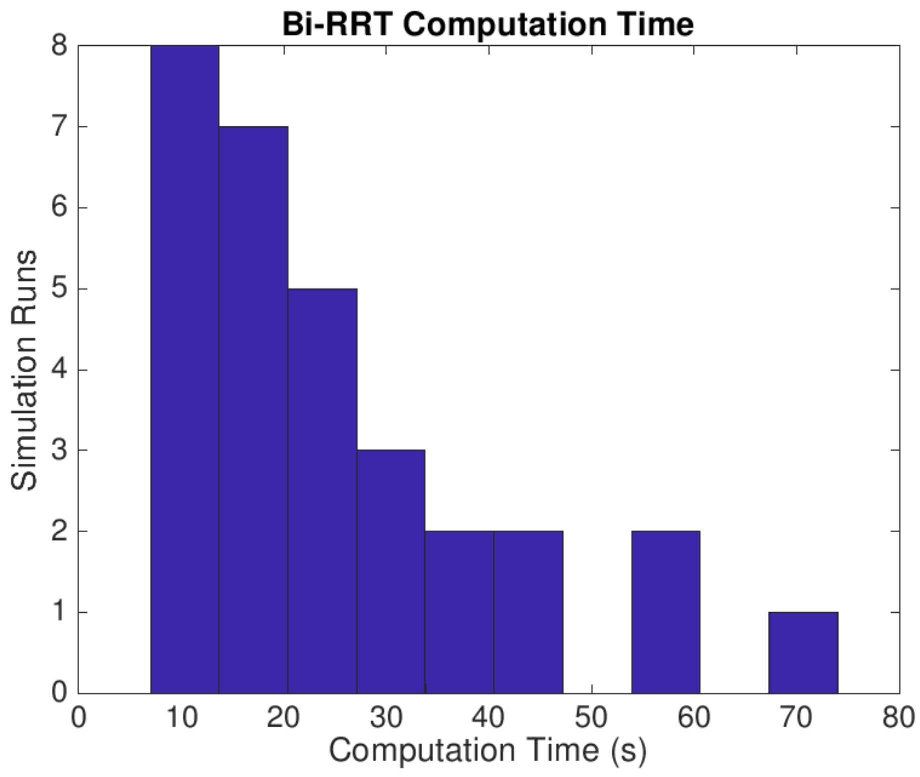


Figure 5: Bi-Directional RRT-Connect run 30 times with a step size of 0.05

The histogram shown in Figure 5 represents the computation time taken by 30 different Bi-RRT simulations. The distribution is similar to the one for the RRT algorithm, indicating that the narrow passage requires some amount of the C-space to be explored before finding the solution.