Report on Rapidly-exploring random tree

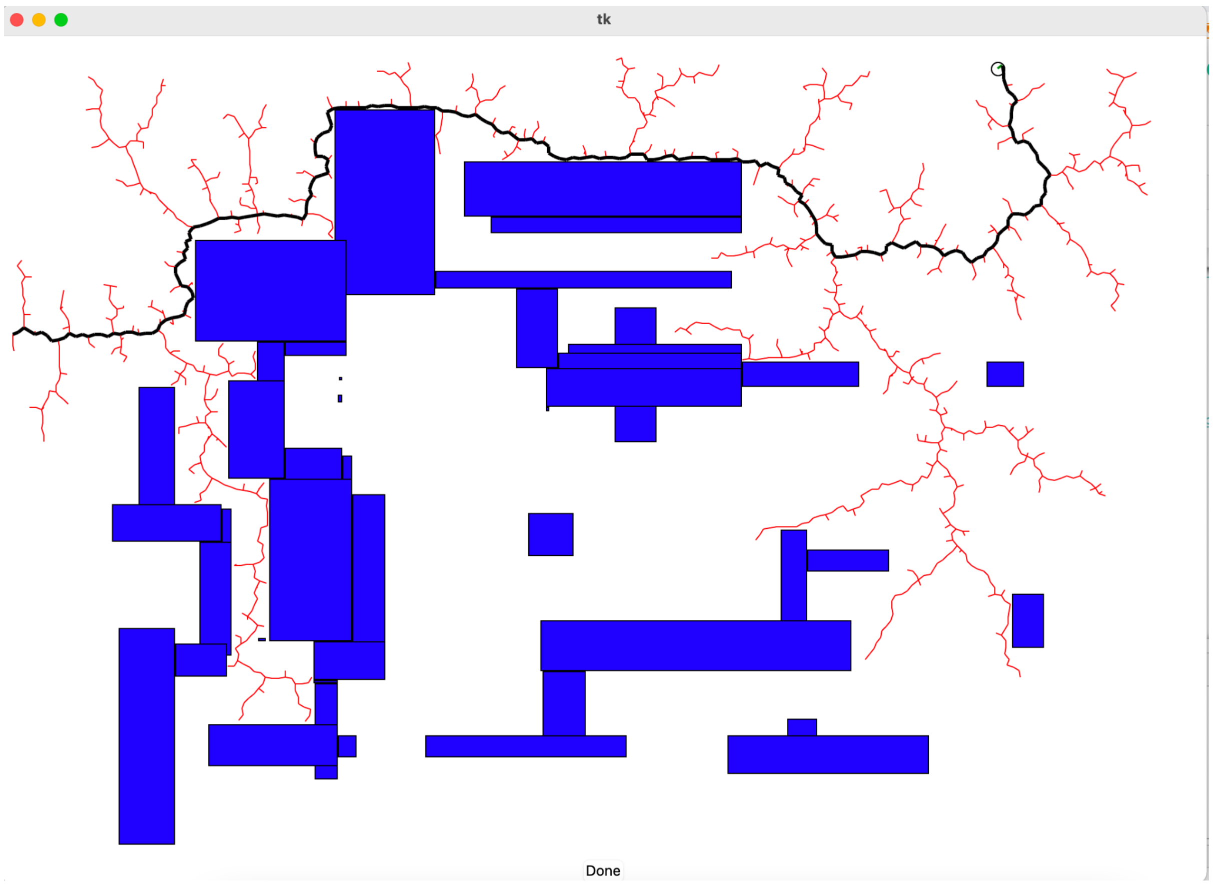
Part 1

A: Implement RRT for omnidirectional point robot

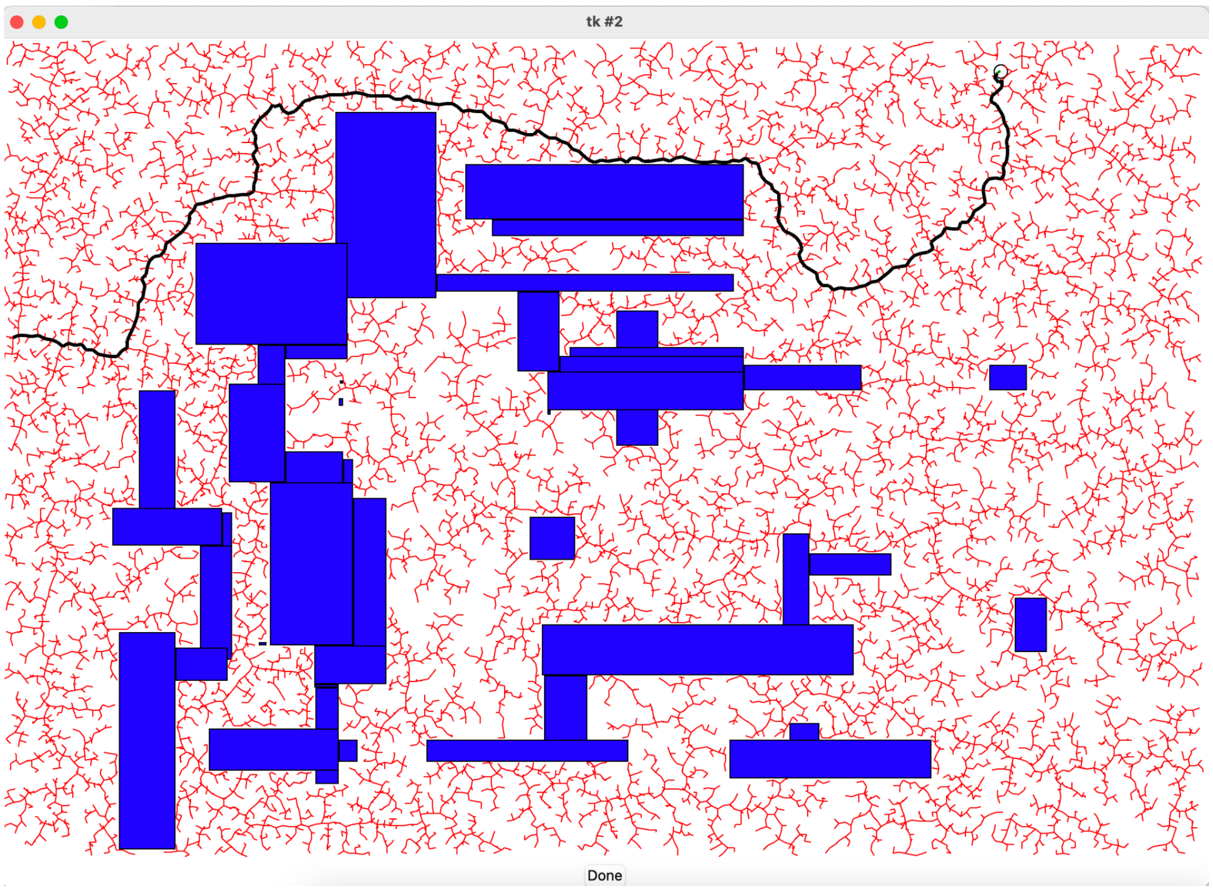
The Rapidly Exploring Random Tree (RRT) is a search algorithm in order to compute a path from a start point to a goal region using a graph. Randomly selected points are sampled in order to search and fill the space with attainable points. The RRT allows for the nearest node to the randomly generated point X to be selected and builds a new node that is closer to X. My first difficulty was to find a way to execute the program using Python2.7, which I had to resort to using a Jupyter Notebook for the handling of all of the dependencies, so changes were made in order to accommodate this environment.

The first obstacle of the exercise is to build an RRT algorithm such that the graph expands in search of the targeted zone. We must then build the obstacle collision avoidance such that the new nodes built in the tree cannot enter the zone of the obstacles.

Step Size: 6 with a gaussian distribution



Step size: 6 with uniform distribution

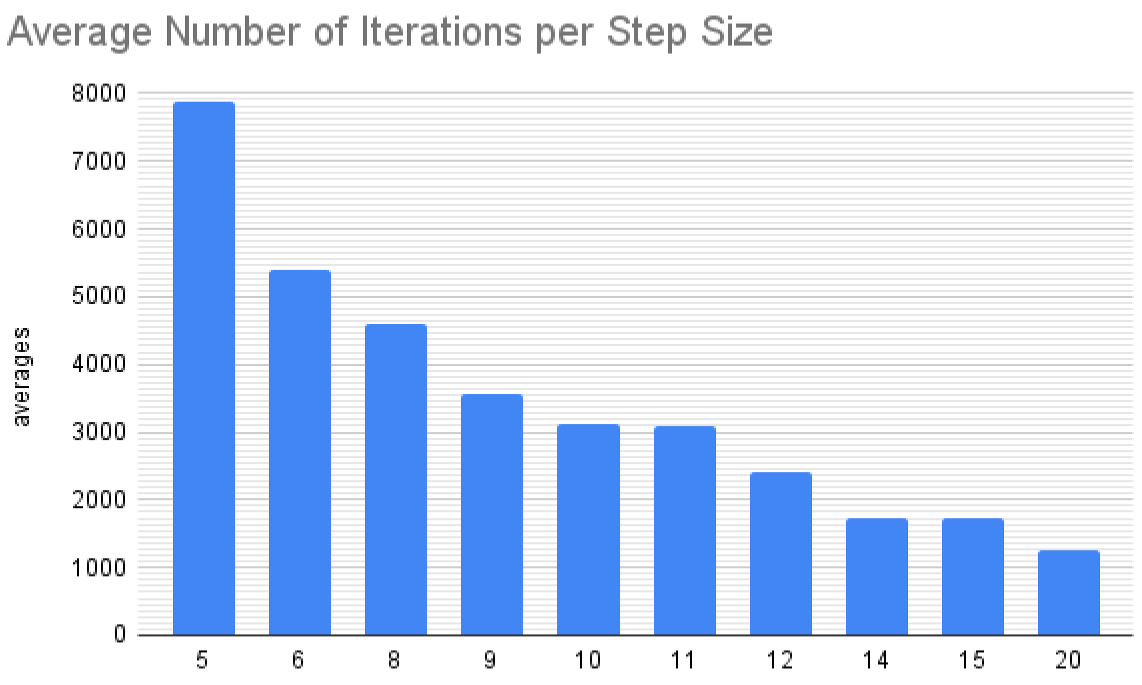


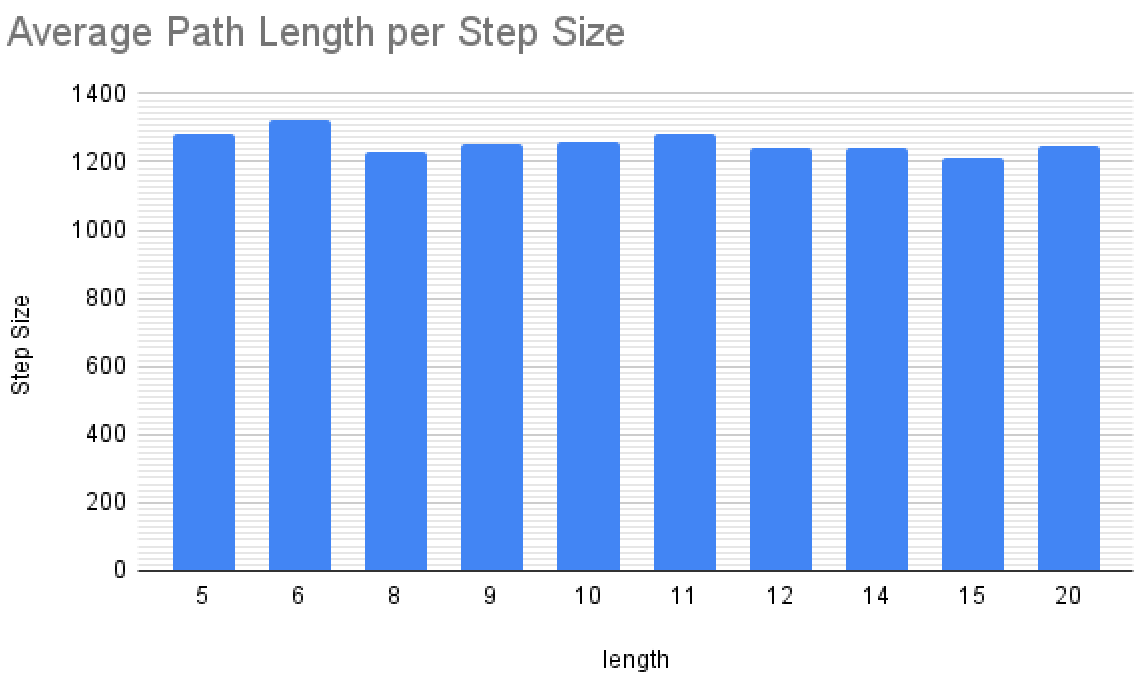
B: By implementing the RRT, using the gaussian sampling instead of the uniform sampling shows us. The gaussian sampling samples consider the position of the target and return random points which are selected closer to the target.

The uniform sampling considers a sampling where the areas explored by the graph are not skewed by a selection closer to the target. Using the gaussian sampling allows us to better "aim" towards the goal, which in turn allows us to reduce the number of iterations to find a path. This bias in exploration allows for a smaller computational cost in practice. By sampling the world with different distributions, we can find that with a uniform distribution (without bias towards the goal), more exploration is completed, but the program takes longer to execute. If the random points generated to build the new nodes are too close to the target, the program may not explore enough to avoid the obstacles efficiently. A balance may be found in the distribution used, depending on the world explored, to find the target most efficiently.

C: We have picked the following step size: 5,6,8,9,10,11,12,14,15,20. The measures done below have been conducted using the Gaussian distribution. We have noticed that there are only some disparities while considering the path length, which fluctuates between step sizes 6 and 15. However, we can not conclude that there is any significant relationship between step size and path length.

We notice that for each consecutive step size, from small to large, the number of iterations seems to be reduced. While a smaller step size offers higher precision, it is much more costly in calculations for a robot. The step size ought to be considered compared to the world in which the robot evolves.





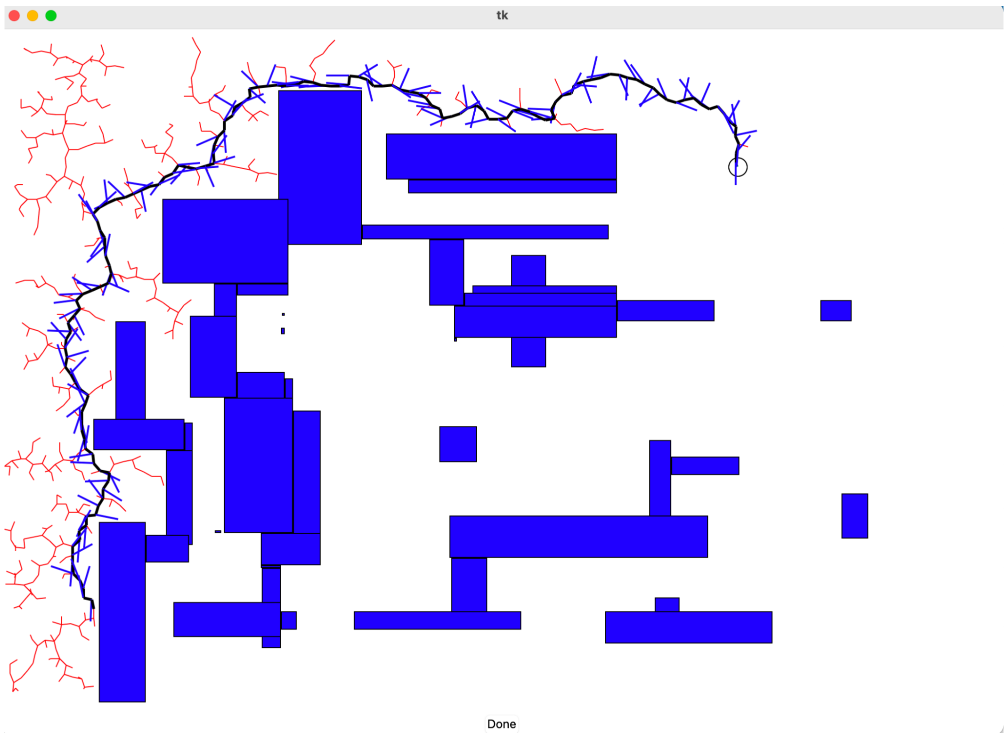
Part 2

A: We must handle the changes of the objects in a higher dimension. We must then determine the points of the graph, such as the position of x, y and Theta, which handles the angle of the robot, compared to the start position.

As told in class, we consider the robot from an "origin" point, similar to the single point of part 1, and we handle the "body" of the robot from this point; the origin point is the "head" of the robot, and the rest is the "tail." We must modify the collision avoidance by not only taking into the head but also the tail of the robot. By considering the angle taken by the robot at each new point, we then compute the tail, using the robot length multiplied by the sinus and cosines of the angle, in order to determine whether a point in the robot's tail collides with an obstacle.

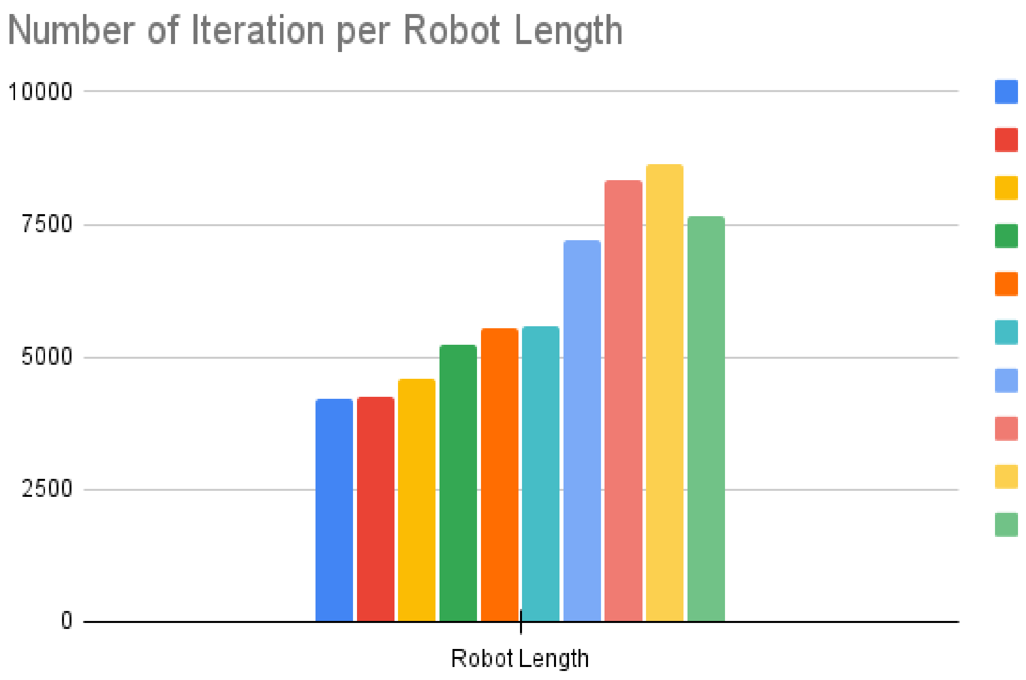
The last difficulty encountered was on how to make the robot turn one last time, such that it is of the same angle Theta = 0, at the start and at the finish. A possible solution is to "fix" the first and last position of the robot by directly setting the angles to 0 before drawing the resulting map of the robot's positions.

Step size: 10 with a uniform distribution



B: We must now consider the size of the robot, given a constant step size in the result on the number of iterations, to build the path. We choose the step size 10, and we variate the length of the robot from 5 to 50 using increments of 5.
We notice that while there are very large numbers of iterations at all robot sizes, the smaller the robot and the quicker it seems the problem is solved, with shorter best-case scenarios, i.e. lower number of iterations, with smaller sizes of robots, on the launches with the best times. Because a robot is smaller, it is easier to find new nodes that will not cause collisions between the robot and the obstacles. Compared to the general trend, the lower number of

iterations (compared to the general trend) for the robot of length 50 may be due to outliers in the computations.


Number of Iteration per Robot Length
Robot Length