1 Introduction

This project looked at implementing the RRT-connect algorithm for the robot with three links. The smaller components of the algorithm were to generate a random configuration, check if the configuration was in collision and use the local planner to check a straight path from one configuration to another. Generating a random configuration was straightforward, as each angle was randomly generated from the range $0 \rightarrow 2\pi$.

Checking if a configuration is in collision was also fairly straightforward, since it just consists of checking if each line segment was in collision with one of the obstacles. Once these fairly simple pieces were in place, the next step was to implement the RRT Connect algorithm, which requires RRT Extend Simple and RRT Extend Multiple. The RRT Connect algorithm requires two trees, one starting at the initial configuration and one starting at the goal configuration. A random configuration is generated, and one of the trees grows towards that configuration using the RRT Simple algorithm. The other tree will attempt to grow towards the new node that was just added to the first tree using the RTT Extend Multiple. If the trees connect, then the algorithm has successfully found a path from the start configuration to the goal configuration. Otherwise, the trees are swapped and the process is repeated.

For the implementation, I used the Graph data structure in Matlab to track the trees, combined with a hash map to maps the names of the nodes in the graph ($s\#$ *or* $g\#$) to the configuration represented by the node in the graph ($\theta_1, \theta_2, \theta_3$).

2 Results

First and foremost, Figure 1 shows the results of the algorithm with a step length of .75. This step size is quite large, and may not be suitable for more complicated maps, but worked well for this map. I used the color bar to show the progression of time, as the red links demonstrate the earliest movements of the robot, and the color progresses to orange and yellow in subsequent movements. This path required 12 steps. I have uploaded a .AVI file that shows a video of the robot following a found path. **This video can also be found on YouTube**. This path is not the same path as the one shown below.

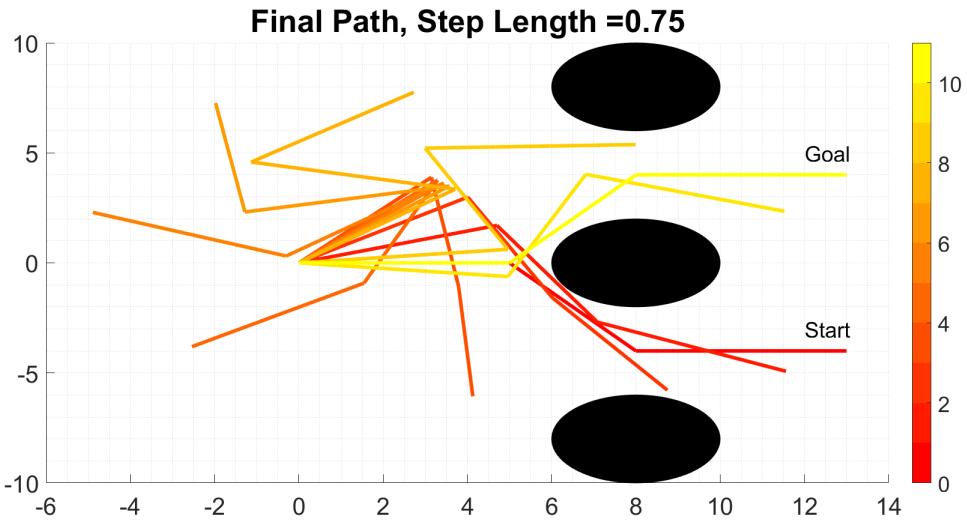

Figure 1: Solution

Figure 2 shows the graph structure of connected trees for the solution shown in Figure 1. The nodes in the tree with the goal state are labeled $g\#$ while the nodes in the tree with the start node are labeled $s\#$. Therefore, the two separate trees connected at $s13$ and $g10$, which means that the start tree has 13 nodes (or configurations) while the goal tree has 10 nodes. Note that this plot is simply a visualization of the graph structure and does not contain any information about the configurations represented in the graph. In other words, the distance between nodes does not represent the distance between the configurations represented by those nodes. This tree is fairly simple, which is a function of the large step length.

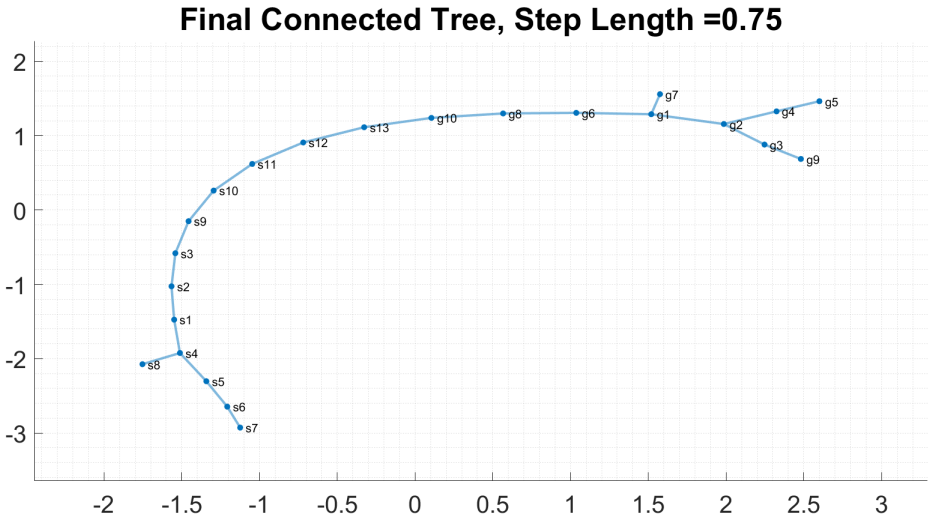

Figure 2: Tree of Solution

2.1 Hyper-Parameter Experiments

I wanted to play around with the algorithm a bit and get a feel for some of the hyper parameters. There are two hyper parameters, step length and step size. The step size just needs to be small enough that an obstacle is not missed by the local planner. I set this value to a constant .001. The step length is more interesting, as it will determine how far the robot will move towards a random configuration. I found that larger values created simpler paths and ran much quicker. The downside is that for more complex domains, a value that is too large may have a difficult time moving towards a random point without being trapped. In other words, the step length must be decreased when the obstacles are closer together. As shown above, I started with a step length of .75. I ran some experiments with higher and lower values to see how the algorithm performed.

The largest step length I tried was 2.0, which was quite large. The algorithm ran quite fast for this experiment, and had a very simple tree (i.e fewer intermediate nodes). The path is shown in Figure 3. As you can see, the robot takes large moves and finds the goal within 6 steps.

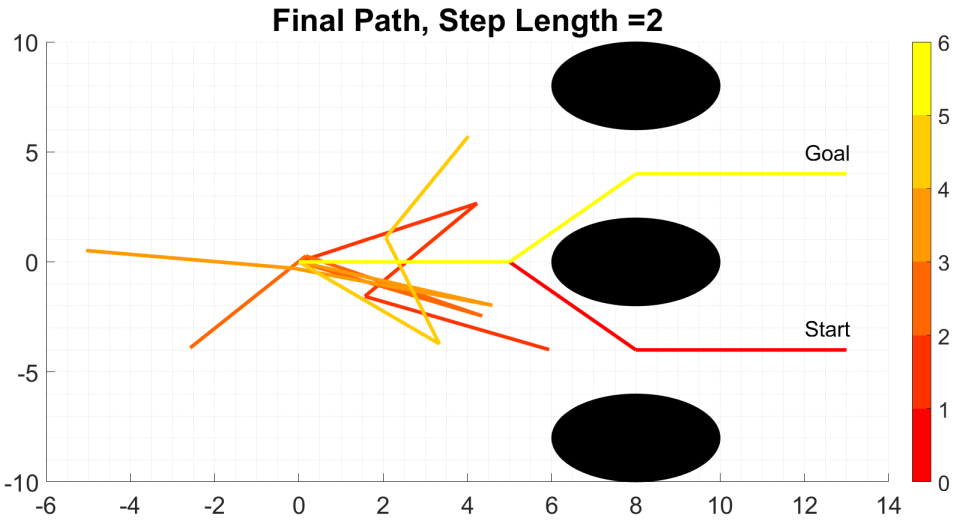

Figure 3: Solution with step length of 2.0

The tree, shown in Figure 4, which shows that the tree is fairly simple. Note that *s1* is the start point (and *g1* is the goal), so there is a fairly significant branch that is not used in the final path.

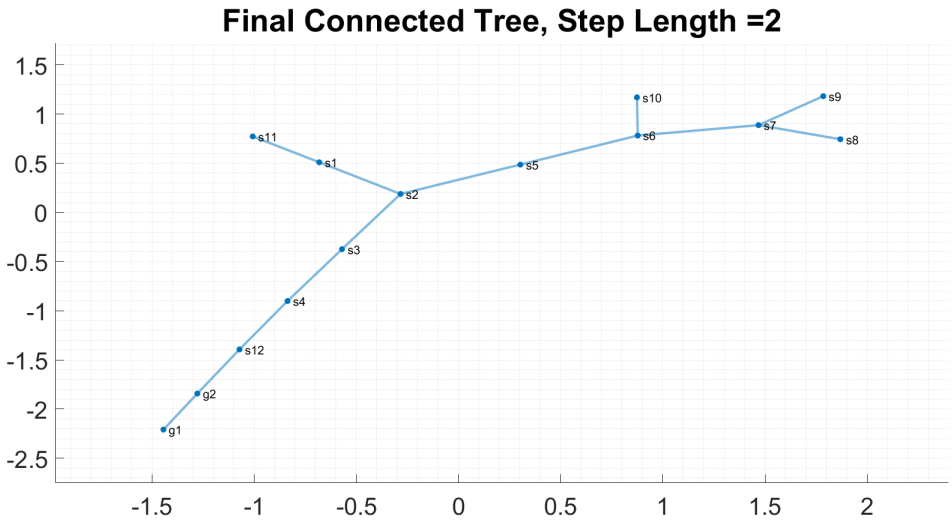

Figure 4: Tree of solution with step length of 2.0

I also experimented with smaller values of step length. Using the value of .25, the tree gets a lot larger and the solution has many more steps. This makes sense, as the robot will visit more intermediate configurations due to the smaller step size. The solution is shown in Figure 5.

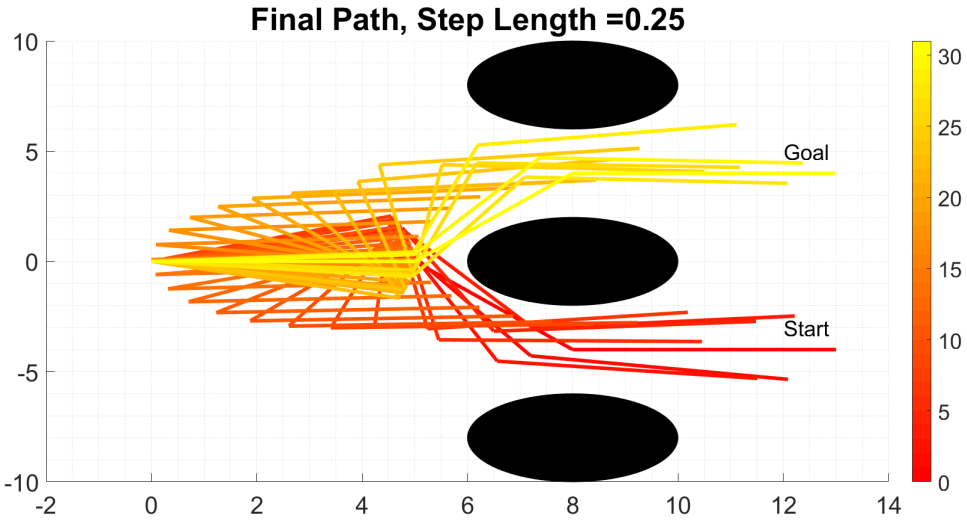

Figure 5: Solution with step length of .25

The tree is much more complex with the smaller step length, as shown in Figure 6. This tree has 93 nodes, and the robot visits over 30 configurations on the path (as compared to 6 when step length is 2.0).

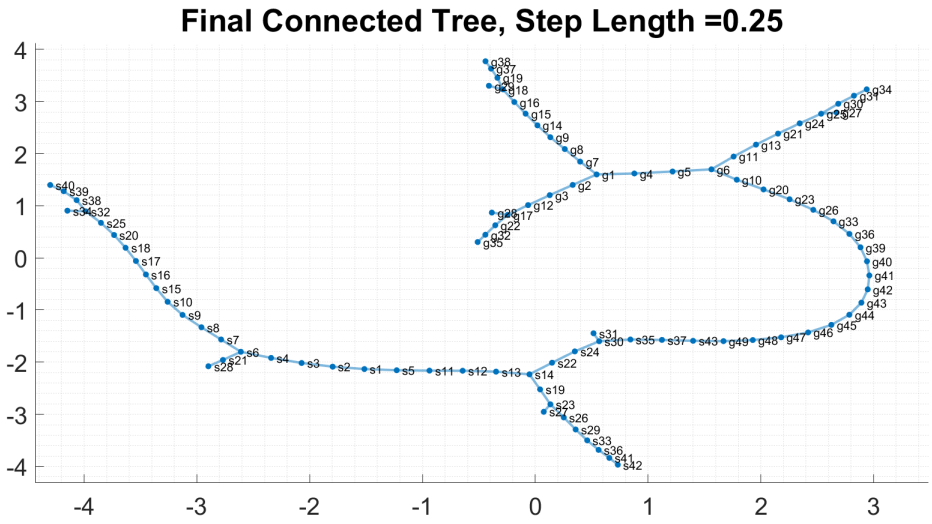

Figure 6: Tree of solution with step length of .25

As the step length gets lower, the tree gets much more complex and the algorithm takes much longer to run. For example, when step length is .025, the tree has nearly 1000 nodes, as shown in Figure 7. The tree contains too many nodes to display the names, but the path contains over 450 configurations, as shown in Figure 8. Note that the slight apparent overlap of the obstacle is due to the weight of the line in the plot and not due to a collision with the obstacle.

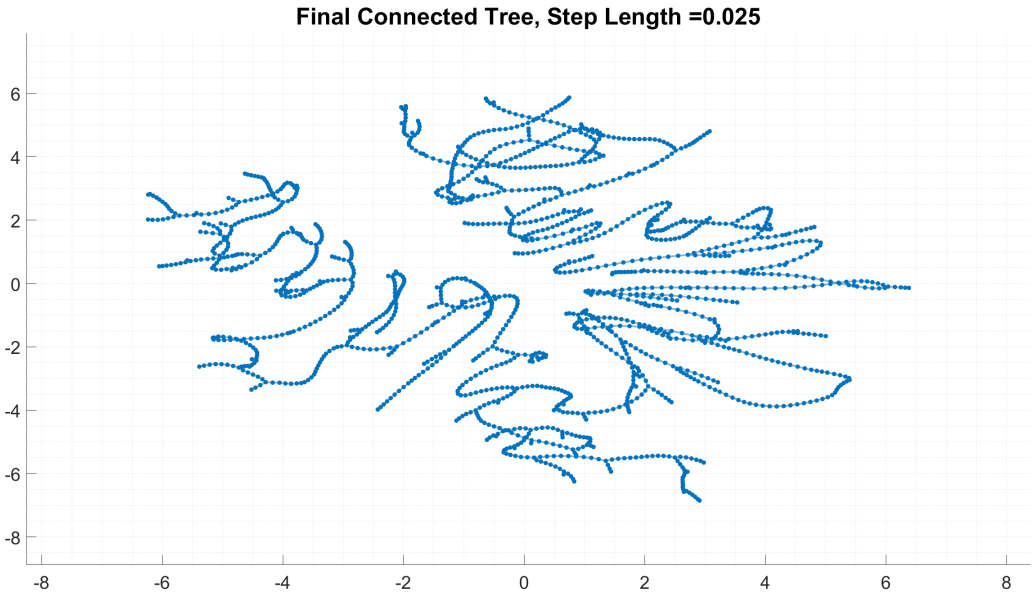

Figure 7: Tree of solution with step length of .025

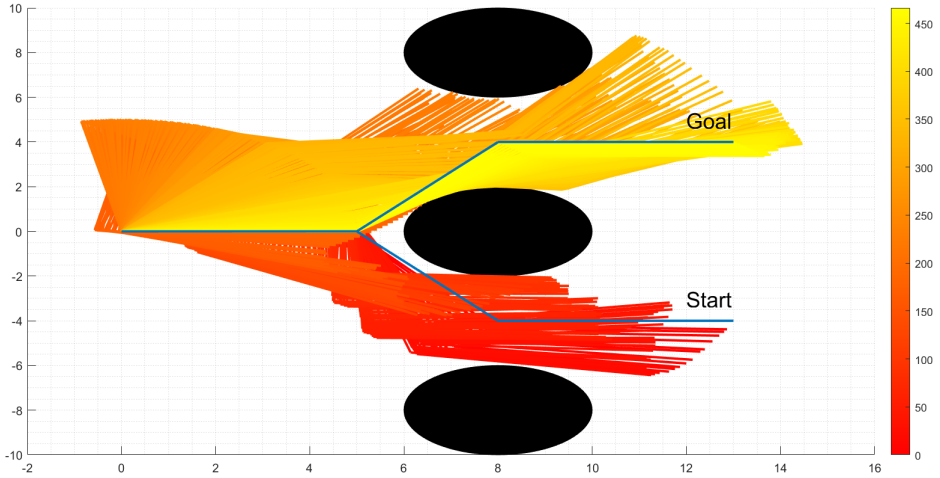

Figure 8: Solution with step length of .025

2.2 Map Experiments

When coding this algorithm, I implemented the map in a flexible manner, so that I can easily make changes to the problem, within reason. I can change the length of the robot arms, the radius of the obstacles, the initial and goal configurations of the robot and the number of obstacles.

First, I added two additional obstacles to the map to complicate the planning problem, but also constrain the robot to a certain type of motion by making certain paths more difficult. This path is shown in Figure 9. Note that this motion is fairly efficient, in that it does not take too many unnecessary movements. Without these additional obstacles, the robot can take any path. Some of these paths can be pretty efficient (for example, consider the path in Figure 10

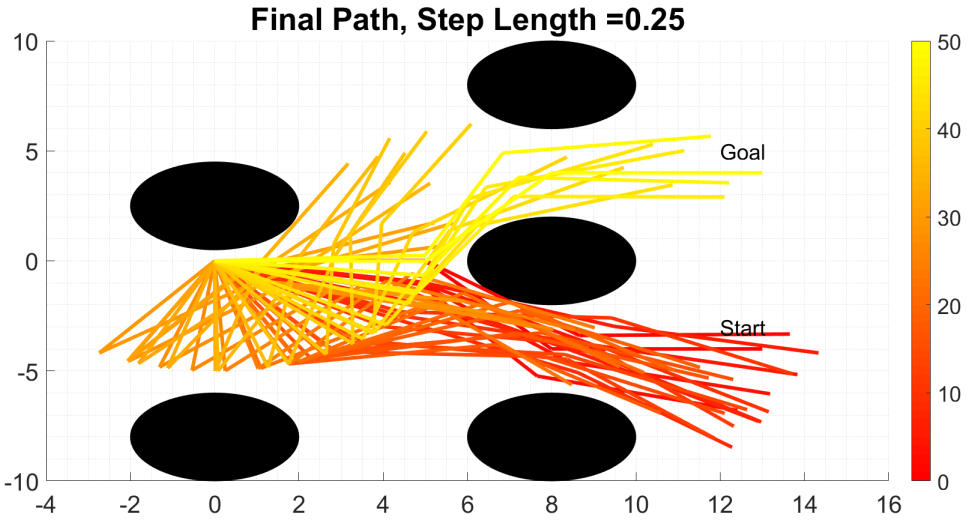

Figure 9: Solution with additional obstacles

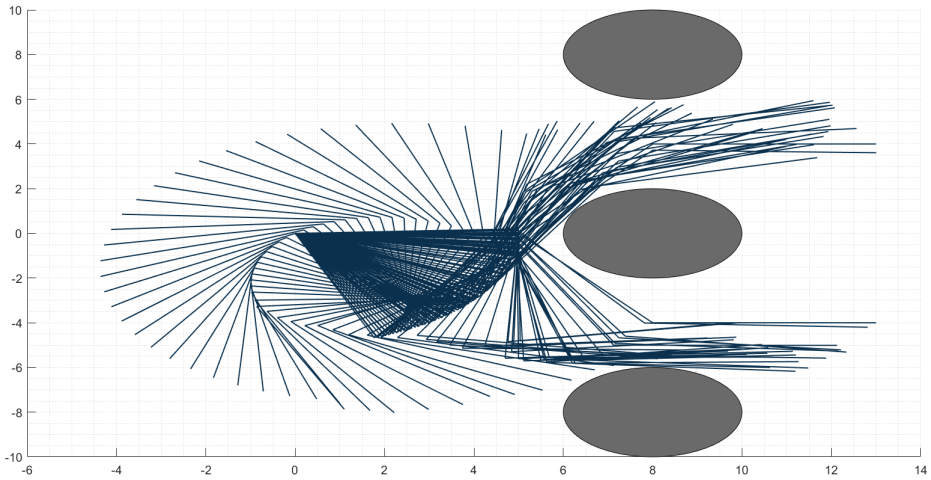

Figure 10: Inefficient Solution

I can make the problem even harder on the robot by manipulating the obstacle locations, as

shown in Figure 11. For example, I can move the obstacle in the lower left up to restrict the movement of the robot even more. This modification makes the algorithm take longer, since it is harder to connect configurations with the additional obstacles. I decreased the step length for this experiment to allow the agent to find intermediate configurations a little easier. While this problem is more difficult, the robot moves much more efficiently for this problem.

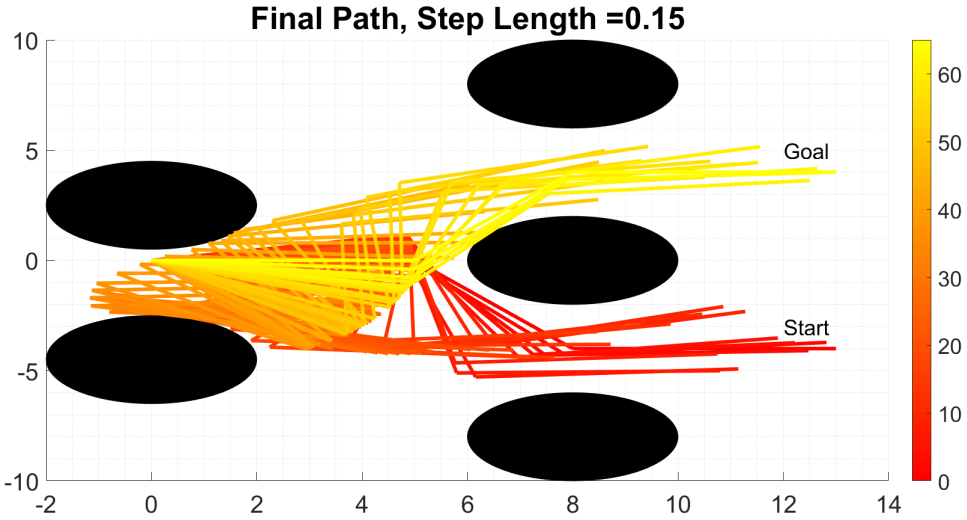

Figure 11: Solution with more difficult additional obstacles

Next, I changed the length of the robot arm to see how this affected the robots ability to find a path. Care had to be taken to make sure that an actual path existed. In other words, the length of the arm could be changed such that the starting or final configuration is in collision with an obstacle. After slightly modifying the start and end configurations, Figure 12 shows the path found by the robot when the arm length of extended to 5.75.

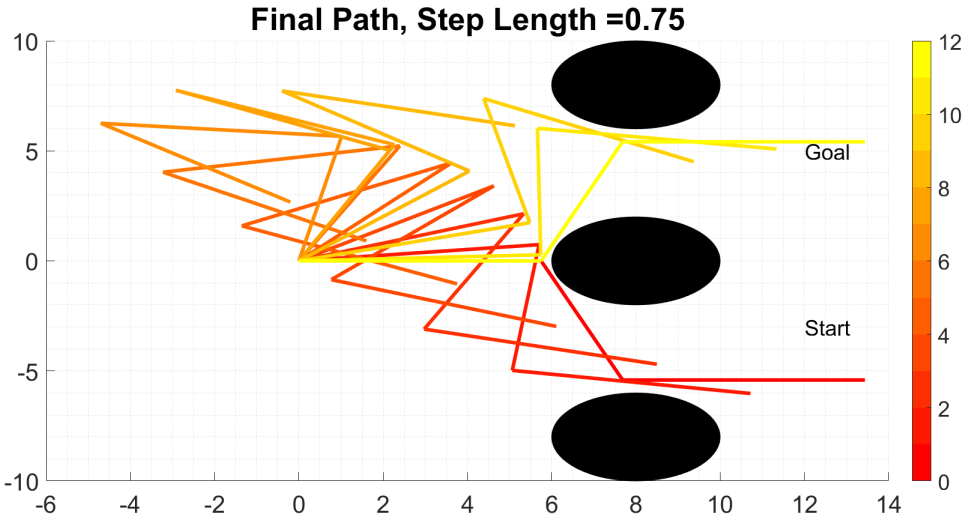

Figure 12: Solution with arm length = 5.75

Finally, without modifying the start and end configurations, increasing the arm length will cause

collisions. However, I increased the arm length length to 6.0 and reduced the radius of the obstacles to 1.5, so that the initial and final configurations did not collide with any obstacles. This makes the planning problem harder, since the robot has to maneuver an extra 3 units of length. These results are shown in Figure 13.

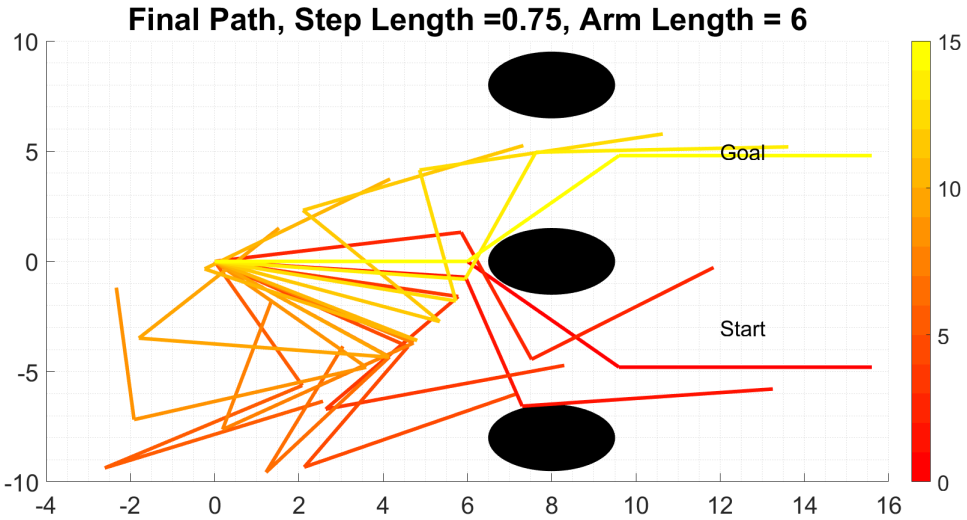

Figure 13: Solution with arm length = 6.0