Project Report 2
CS 470
Eta Team
Nathan Radmall, Eric Romrell, Jon Francis
Youtube Link: https://youtu.be/jDx9OPagqjc

1.1 Description of Search Algorithms

We implemented both A* and RRT for this lab. We will describe our implementations for both in this section.

For A*, the first thing to do is discretize the search space, and then generate an accessibility mapping for each space to all other spaces that it can directly access. We decided that any adjacent square (N, NW, W, SW, S, etc) would be considered "accessible". With this mapping, you find the discrete space closest to the sphero's starting position and mark this as the "start". Our algorithm then uses an actual "distance so far" (defined as g(x)) as well as a straight line heuristic (defined as h(x)) to calculate the "best" path to take. We then use a priority queue to determine what the next best path would be.

For RRT, the basic idea is to start with the sphero's beginning location, and then randomly explore the space for possible paths to the goal. The way that we randomly explore the space is by finding a random point within the bounds, finding the point in our path nearest to it, and then adding a point to the path that is a specified distance away from that point in the direction of the random point. This follows a Voronoi pattern, and normally does a good job of exploring the rest of the space.

1.2 Description of Other Group's Code

One of the groups that we compared code with had one fundamental difference from our own code. In the code, there is a place where you have to determine whether or not something is in an april tag. The integral difference between our code and this group's code was that we only searched for whether or not the end point would be in the april tag. The group that we analyzed instead used intersection logic to determine if the full path touches the april tag anywhere. Our idea worked because our delta values was smaller than the radius of our april tag bounds, but we thought it was interesting that they considered the problem in a different way than us.

1.3 Time Spent

Eric: 8 hours
Jon: 15 hours
Nathan: 7

1.4 Suggestion to make this lab better

Nothing really seemed to out of order with this lab. Most of our time was spent out of the lab working out the problem theoretically. There was still quite a few times where either the camera or the sphero would have issues connecting, but after a few tries we would normally be able to connection just fine.

2.1 How we did on our obstacle configurations

We ran our sphero through many different configurations of mazes and paths. Through all of these test, we learned what our sphero did well with, as well as what was out of it's abilities.

First, we ran our sphero through a pattern that resembled an "H". Sphero would begin in the top half of the "H" (surrounded by walls), and would have to find the goal in the bottom half of the "H". This would require it to find its way out of the first box, around the outside of the "H", and then back into the bottom box. It actually did very well with this course!

Next, we created a large, intricate maze that went all over the world's bounds. Sadly, the calculation of the correct path took too long because we discretized the area into rather small intervals. However, after shrinking the maze, the sphero was able to solve it rather quickly and smoothly. One thing that we learned as we ran sphero through a maze-like structure was that the momentum of the sphero would often careen it into an april tag, despite the tag not being a valid point in the path. The way we overcame this was by making the sphero move a little slower, and not including any points within a certain radius from the april tag. This ended up working really well.

2.2 How we did on the other group's obstacle configurations

The other group set up a maze-like course that involved two sharp turns. Initially, our sphero wouldn't even find the path due to a discretization error, but once our sphero calculated the path correctly, we had no other issues. The sphero found it's way out of the initial box that it was in, and then turned the sharp corner and dodged it's way around some strewn april tags to the goal with ease.

2.3 How the other group did on your obstacle configurations

The other group's program worked surprisingly well. We wanted to test a few different things. The first was if it would back out of a dead end and it did. The next thing we tested is what would happen if there was no path to the target. Again, their program worked just as expected and didn't find a path to the target which meant it didn't move at all. The last thing we wanted to test was the precision of the algorithm. We created a path that at one point was about 4 inches wide. When running the program with their original grid size, the algorithm couldn't find the whole so no path was returned. Leaving the maze the same, they halved the size of the grid and ran the program again. This time, the algorithm worked perfectly. RRT works the same as A* but is quite a bit more wobbly.

3. Something Creative

For our creative section, we implemented the RRT algorithm that starts a path from both the start and the goal. This actually worked rather smoothly, and helped the RRT find the path quicker than when we only ran that path from the start. The basics of the algorithm were similar, but the implementation had a few hang-ups… We had both paths grow towards each randomly selected point, and then if ever the paths came within a certain distance from any point in the *other* path, we would join them there. Sometimes, this would cause small lurches in the sphero

when the algorithm joined paths, but we were quite pleasantly surprised that the algorithm worked so smoothly.