Robot Soccer

Dyllon Dunton
12/7/2023

Robot soccer is an interesting niche in mobile robotics since it attempts to approach solutions to rapid path planning around stationary objects, in addition to unpredictable moving robots. There are methods that can quickly come up with path trajectories. However, they often sacrifice processing time in order to trim the path to the shortest, smoothest path. A good example of this is the difference between RRT and RRT*. Rapidly exploring Random Trees (RRT) are excellent at quickly finding a path in a continuous state space, but they will most always come up with a convoluted path that is much too sharp and long. RRT* improves on this by continuing to poll for points in the graph randomly, looking for a shortcut between consecutive nodes. This continues for as long as the operator desires, smoothing out the path and approaching the most optimal trajectory. However, this takes much longer than regular RRT.

In [1], a compromise is found between RRT and RRT*, which involves finding a path using RRT, and then post-processing it to make it smooth. This procedure achieves a much smoother path, while allowing for a slightly longer length than RRT* would have been found. I have implemented this algorithm in a ROS2 package. The package will talk to a robot that tracks positions of obstacles using ARUCO tags and a Logitech C270 camera. The package receives this information and builds a continuous space graph with all obstacles, in addition to the location of the ball and the goal posts. The robot then calculates the position it will need to end up to shoot the ball between the two posts. Knowing the start and end locations, a randomly exploring rapid tree (RRT) algorithm plans a jagged nonoptimal path to the goal around the obstacles. Next a smoothing algorithm is run on this path that checks every node to see if it can walk straight back to the start node. If it can, it pops off the intermediate points behind it. Once it finds a node that cannot walk back, it moves back a node and starts the algorithm over again until it reaches the goal. This reduces the amount of points in the final path from over 30 to around 3 each time while testing with my setup. This is a 90% reduction in the total amount of turns needed to complete any path. Since turning with the jetbot and other robots is often prone to error, this will help the robot drastically reduce turning mistakes on the way to the goal. All of this information is plotted using matplotlib to show the path to the operator before executing, which can be seen in the demo materials submitted along with this report. After checking the path, the node will publish messages to the robot that tell it how to navigate the course.

I was able to complete all of the objectives that were originally planned in the initial proposal. This project was a lot of work and I did stumble on some issues. The largest issue was standardizing movement for the robot. Since it turns differently on each surface, standardizing movement that would work consistently on multiple surfaces was not accomplished. Creating the standardization modules helped considerably in this effort, as I was able to tweak multipliers in the turning and forward values sent to the robot, and then copy them into the main module.

References:

[1] R. H. Abiyev, N. Akkaya, A. Çağman, I. Günsel, and E. Aytac, “Improved Path-Finding Algorithm for Robot Soccers,” *Journal of Automation and Control Engineering*, vol. 3, no. 5, Oct. 2015. doi:10.12720/joace.3.3