Project Name: Robot Racing

Members: Jeffrey Ciferno, Chinmay Bhale, Kumal Patel

Introduction:

As our world becomes more and more automated, technology will seep into every nook and cranny of our society. This trend is particularly true in the sport of competitive racing. Companies like Nvidia and Roborace have invested in self driving robotic race cars. Our project was inspired by Roborace's robocar, which is a full autonomous, electric race car. It boasts many methods to collect environmental data, from cameras to ultrasonic sensors. This concept was intriguing, so tackling a smaller feat with rudimentary robotics skills seemed possible.

General description of Project Goals:

1. Design a track for the robot to race around.
2. Implement RTT algorithm.
3. Get multiple robots to race.

Initial Plan:

As a group we went into the project with the hopes of getting a set of robots to race between each other. We hope to design an initial track to build into Coppelia Sim. The main goal of the track is to test the robots ability to navigate turns and straights like a race car might. We then aim to integrate an RTT algorithm into the robots. This is with the intent to try to get the robot to go around faster each time.

Work Breakdown:

Chinmay Bhale:

1) Worked on adding obstacles wall to the racetrack
2) Worked on developing the RRT algorithm
3) Helped in completing the project report

Jeffrey Ciferno

1) Designed a track inside AutoDesk 3DS Max
2) Worked on scene design
3) Tested code and worked to try to integrate code onto the KukaBot

Kumal Patel

1) Kept the groups morale together during a pandemic
2) Helped with working with challenges and broke down any logic

3) Worked with project management

Design Overview:

The intended design is to create a racetrack where a vehicle would need to speed through the track as fast as possible while maneuvering around any incoming obstacles. We implemented a rapidly exploring random tree (RRT) algorithm to avoid any obstacles the vehicle might encounter. This algorithm is a tree structure and works by incrementally constructing samples from the search space and is baised to grow in the direction of any unsearched space. Since all the obstacles are known we can use that knowledge to have the vehicle take the path that is opposite to the obstacle, hence avoiding any incoming obstacles.

The idea was that we were going to set way points along the track and find the RRT algorithms in these sectors. We were going to split the track into multiple sectors and then use the waypoints from the RRT algorithm to drive the KukaBot around the racetrack.

Timeline:

3/25 - Grouped up, got initial ideas flowing.

4/7 - Decided on our idea of robot racing

4/8 - Group meeting #1

4/9 - Finished building first track in Autodesk 3DS MAX

4/12 - Setup environment and started the initial code

4/13 - Group meeting #2

4/19 - Made track edits due to errors with interaction

4/20 - RTT algorithm is finished, just need to adapt the code to work on the robot

4/26 - Group meeting #3

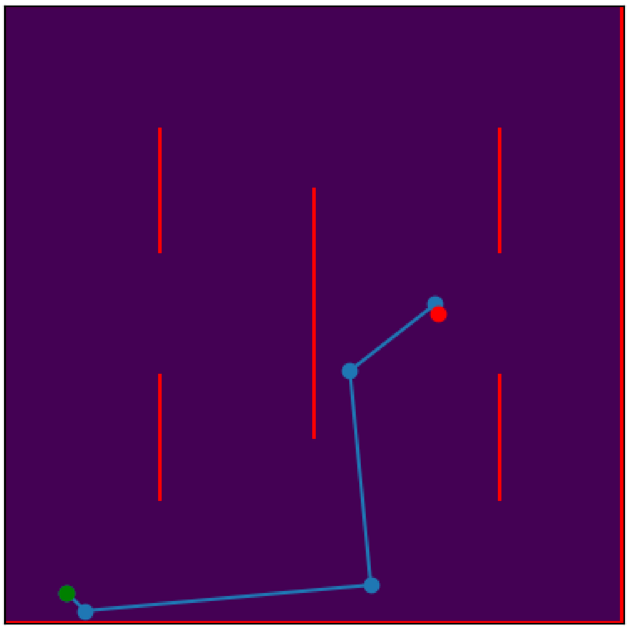
Project Significance:

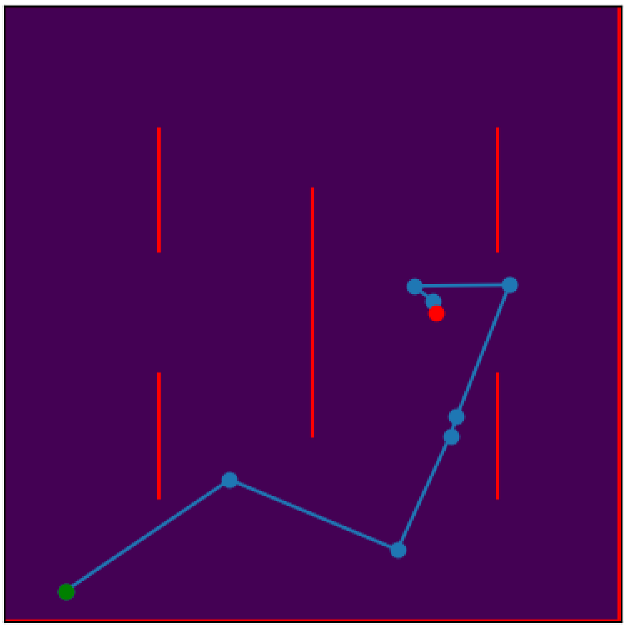
This project is significant because of the concept it shows. Pathfinding, robotics automation, and AI technology. The rrt algorithm uses the concept of pathfinding which can be used in various applications, even concepts with AI technology. Because the rrt algorithm inherently is biased we can take advantage of that and implement it into robotics.

Final results:

In the end, we were able to create a running RRT algorithm which could find a path from start to goal within a million iterations. The algorithm was able to calculate the path even with obstacles in the way and there is no appreciable hit in performance. Below are some of the results from the RRT algorithm.

We were also able to get the kukabot working within the track that we had designed to test the algorithm. The robot currently uses an inbuilt algorithm to move around the track.





Challenges:

We faced many challenges in doing this project. We first tried to build the track in blender and then import it to CoppeliaSim. Unfortunately, this did not work as expected and we traced the track with walls so that the robot would only drive within the track limits. The next challenge was trying to interface with the scene using PyRep. This proved to be the most challenging part of the project as we did not understand how to import the track walls. The problem was that we wanted the length, orientation and thickness of the wall and we were not able to figure out how to use the APIs successfully to work with our code. In the end we decided to focus solely on the RRT algorithm.

For the RRT algorithm to work, we needed to find the path from the starting point to the goal. This required a pretty big rewrite of the code as till then we were not able to find an elegant solution to find the path. We ended up using a tree-like data structure so to get the path, all we needed to do was to traverse the tree backwards up to the root node. We also found it challenging to display the graph and the obstacles in a visual manner. We ended up using matplotlib and plotting each obstacle individually on the same graph to get the maze-like image that we were looking for.

Results:

In the end, we were able to get the RRT algorithm running and also get the CoppeliaSim robot scene set up. If we could have had a little more time to understand and sort out the problems with using the PyRep APIs we could have had the robot racing around the track using the RRT algorithm. However, even if we were not able to complete everything we were set out to do, we were still able to learn a lot about creating scenes on CoppeliaSim, learning about the PyRep APIs, and learning and implementing the RRT algorithm.

In doing this assignment, we learnt a lot about using Autodesk to create the track model and about how to import it into CoppeliaSim. We also learnt the basics of calling PyRep APIs to get information about obstacles in CoppeliaSim. We also learned about how to use matplotlib to plot the maze and the final path found by the RRT algorithm.