**FP2: Final Project Specification**

**CSCI 3302: Introduction to Robotics**

# 

# Team Members:

1. Brett Denson: brett.denson@colorado.edu
2. Joseph Ledesma: ledesmaj@colorado.edu
3. Kairui Liu: kairui.liu@colorado.edu
4. Chase Engle: chen3239@colorado.edu

# a. Abstract

Our team plans to create a dynamic robot which has the capability to autonomously traverse an intersection. Additionally, our robots will be navigating a world containing multiple other vehicles and moving in optimally through a dynamic space. The main premise is to implement path finding algorithms to navigate a complex space/problem.

# b. Equipment

* One of the Webot Automobile Platforms

# c. Deliverables and Implementation Plan

* Deliverables
  + Intersection world or area that robots may collide
  + Path finding algorithm
  + Base world controller
  + ALTINO controllers
* Implementation Plan

Lead: Brett Denson Deadline: 11/25/20

* + Code a world where robots have intersecting courses

**World Creation Write Up:**

Creating the world consisted of creating a 1 km by 1 km rectangle arena so that the vehicles

Had plenty of area to carry out the algorithm with long stretches of the road leading up to and away from the intersection at the middle of the arena. To create the intersection, we utilized the “RoadIntersection” node within webots and then edited its size to be a four-way intersection of four-lane roads, extending from the middle of each side of the arena to the center of the world at (0,0). For aesthetics, we also added stop lights to designate the intersection and changed the floor appearance to green to represent grass in the areas not part of the road.

Within the algorithm implementation process, it became necessary to do conversions between coordinates and distances when it came to distance, velocity, and acceleration of the robots. Through this, we calculated that 1 meter is equivalent to 0.90000000252 coordinates, and thus were able to use this to calculate the other values to the units needed.

Lead: Kairui Liu Deadline: 11/30/20

* + Create a traffic light scenario that includes edge cases of robots being in intersection when the light changes

Lead: Joseph Ledesma Deadline: 12/03/20

* + Implement that as a controller for n numbers of robots

Lead: Chase Engle Deadline: 12/05/20

* + Come up with a path finding algorithm. The most difficult parts being coming up with a data structure, an efficient “step” format, and a heuristic

**Algorithm Write Up:**

This became the crux of the final and took me around 22 hours to create a pretty bad solution. There are three iterations of my algorithm all are based on RRT. Essentially, I use RRT to explore a state space of times and velocities. It is really similar to the one in class. My algorithm runs into problems with the size of cars and GPS. Because of the car size I need to add some padding to any given obstacle measurement but the padding I can add is limited because it might cover the cars multiple lanes over. This can quickly lead to an issue because a lot of the time if I lower the padding the cars just barely knick each other. This means my algorithm almost never works but on occasion it does navigate the intersection and it is always very close. In addition to this, I have an iteration of the algorithm where there are no obstacles and any given run is a coin-flip on whether or not it works. This it seems works more often than the other one because the cars aren’t looking for a path that is narrow at all. Another note on my RRT algorithm is that it is biased to pick faster velocities. The demonstration will be examples of success and failure of both the coin flip algorithm and the full RRT with obstacles included. I also implemented this as a controller in the end which just turned into the controller passing info to a RRT class through OS. Then RRT passing back to the regular controller through OS. Then my robots have these times associated with velocities and they run the velocities as the time comes. I think the other thing that leads my solution to not working is a bunch of small inaccuracies adding up. There is inaccuracy converting anything from GPS coordinates to world coordinates and back and there are a lot of very tiny numbers being worked with in the time section and all of this adds up to not being able to use the really narrow padding that is needed to make an interesting solution that is consistent. To make it all more interesting if my program fails and no solution is found I added a “kill switch” that tells the robot to just go full speed.

<https://www.youtube.com/watch?v=dRIPt1jYK90&feature=youtu.be>

Coin flip only works somewhat consistently with about 4 cars but obviously that is just based on luck. The deterministic algorithm with a padding of 2 or 6 works almost every time. With 10 cars it works frequently and with 11 or 12 cars I never got it to work, if I went up to 6 padding here it just crashed if I left it at 2 it never worked.

The attached world submitted can be very finicky just a heads-up so if you want to run it on your own machine it may take a few attempts, make sure to delete all the textfiles in between runs if it doesn’t happen on its own. Also the submission only includes the deterministic form. Finally, webots may crash if you just wait it will probably come back.

Lead: N/A Deadline: N/A

* + (Optional/stretch goal): If there is extra time, add a complication to the course plan like a moving boulder
* Algorithm Premise/Plan
  + Create a graph of potential ways of executing a way through the intersection
  + Initially other options will include accelerating and decelerating, if there is extra time other options will be added such breaking to a stop, or accelerating and decelerating temporarily
  + We will likely use A star to navigate this graph but coming with a heuristic will be one of the more difficult parts of the problem, so the initial implementation may be done using BFS
  + Cars will be aware of each other instead of navigating based off of live sensor information
  + Cars will initially only move along the x and y axis
  + Cars will avoid each other in the best case.
  + If there is extra time some cars will be given paths that include turning

# d. Demo

* A demonstration of some intersection solutions and fails