CS590 Socially Cognizant Robotics

Jonathan Wang (jw1303), Neeraj Pise (np903), Lilian Wang (lw709)

March 10, 2023

Implementation Details

1 Vehicle

The vehicle is given a set of goal points to reach during the simulation in order. The vehicle has a few states that it alternates between depending on the situation: Stopped, Wait, Pursue, AtGoal, and Finished. The stopped state is called when we need the car to stop for non situation reasons (e.g. in part C the car isn't meant to start until the human finishes so it remains in the Stopped state). The wait state is part of the obstacle avoidance flow. We detect obstacles collisions using velocity obstacles calculations. If the car senses it will collide with a human or NPC, it will stop and wait for the human/NPC to pass (similar to real cars). The Pursue state is the main state used for car navigation. The car steers to its goals waypoints using the Pure Pursuit algorithm described here: https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Control/PurePursuit.html. As it moves, it looks for collisions via velocity obstacles and changes state accordingly. The AtGoal state is for handling when the car reaches its current waypoint. When the car reaches a goal it will either set its next goal and pursue or, in the case that there are no more goals, go to the finished state. Finally the finished state is simply when the car has achieved all its goals, in this case control of the car is complete and there is nothing to do.

2 NPCs

NPCs are given two waypoints, the top and bottom of the enclosing space, and oscillate between those two waypoints. At any point in time they calculate the unit direction vector needed to reach the next waypoint and travels along that vector. As it moves, it calculates if its current velocity will result in a collision with the human or the car via velocity obstacles. If its velocity will result in a collision, it looks for a new velocity via fanning out. It will rotate its current direction vector clockwise and counterclockwise until it finds the direction vector closest to its current direction vector that doesn't result in a collision. Aside from some issues wit the human we will discuss later, after tweeking threshold numbers and the velocity obstacle calculation time frame, we found that the NPCs avoided both the human and car with little error. Coupling the NPCs' avoidance algorithm with the car's stop and wait protocol, the interactions between them function similar to how pedestrians and cars interact in real life, resuling in a low chance of collision.

3 Part B:

The car is given 3 waypoints: the beginning of the tunnel, the end of the tunnel, and the top right corner. It navigates through those waypoints, stopping and waiting if about to collide with the

human. We put a few goal sites in the scene to represent the human and car's goals. The green waypoints are the car's goals and the blue waypoint is the human's goal.

4 Part C:

The NPCs move up and down between their waypoints (labelled as blue sites) and avoid the human and car as they pass. Once the human reaches the goal (represented by the green finish line), its trajectory will be compressed by removing adjacent duplicates to get a list of unique points in time. Then we split this trajectory into 20 individual goal points for the car to seek out. The car will then follow this trajectory, stopping and waiting for NPCs to pass if it senses a collision.

5 Issues:

Since the human does not move smooth, we set the program to remember the humans last directional input and use that as its velocity vector at any point in time. This means that the NPCs expect the human to continue moving in its last input direction. Since the human can act sporatically, this may result in collisions if the human were to intentionally seek out collisions the NPCs.

6 Submission Files:

- B
  - mushr_test.py: Main program for part B simulation
  - my_helpers.py: All helper functions for math and calculations done in part B
  - PartB.mkv: Video demonstration of part B and C
- C
  - mushr_follow.py: Main program for part C simulation
  - my_helpers.py: All helper functions for math and calculations done in part B and C
  - PartB.mkv: Video demonstration of part C