E&RTS – Swarm Path Planning and Obstacle Avoidance

Requirements:

This lab required the cart to traverse a known list of GPS waypoints while avoiding any obstacles seen by the mounted laser sensor and remaining in the corridor defined by the given waypoints.

Design:

The basic design of this solution is a point-to-point driver. The cart is constantly moving toward some target waypoint by manipulating values for the inverse turn radius (steering) and throttle (speed). Elements of path planning and obstacle avoidance are introduced by applying a swarm algorithm to a generated list of waypoints.

The waypoints to be followed by the cart are generated from the initial list of waypoints describing the course. These generated waypoints make up the swarm.

Swarm of waypoints with a separation of 4.0 meters.
Each swarm point is aware of the two swarm points nearest to itself, and can move in order to create a smooth path between itself and its neighbors. Movement of each point is restricted to a line perpendicular to the cart path, with end points lying inside the defined corridor.

Swarm members can also move in order to create a path around an obstacle.
During a typical process cycle, this solution will:
1. Check for any obstacles in the current path.
2. If there are obstacles, notify the appropriate swarm points.
3. Update all swarm points.
4. Check distance to target waypoint.
5. If within the LBO of the target waypoint, change target to next waypoint.
6. Set inverse turn radius to steer cart toward the target waypoint.

There are four components used to achieve this behavior: SwarmController, HeadingController, the helper class HeadingUtils, and the distance module from Python’s geopy package.

The SwarmController class is responsible for generating swarm points from the initial list of waypoints, and also for updating the position of all swarm members.

The HeadingController class is responsible for creating an inverse turn radius from a target heading that can then be written to the jdriver file in order to turn the cart toward its target waypoint.

The HeadingUtils class contains static functions for handling addition, subtraction, and averaging of heading values.

The distance module from Python’s geopy package is used to calculate headings and distances between GPS points. It is also used to calculate new GPS points in several places based on a given lat/lon, heading and distance value.

**Implementation:**

In order for this solution to work, the swarm points must move in a useful way when updated. They have two modes of movement; one to create a smooth path and the other to avoid an obstacle.

Moving to create a smooth path is the default movement behavior of a swarm point when it is being updated. Movement is calculated by finding the angle created by the lines to the next and previous swarm points. This will be a value between -180 and 180 degrees. The sine of this angle gives a value between -1 and 1 and is used to calculate a movement value for the swarm point.
An important aspect of the update function in this solution is that each swarm point not only moves itself the calculated movement value, but it also moves its neighbors a smaller distance in the opposite direction. Without this opposite movement, the swarm will only collapse toward the center of the obstacle course.

Movement values should be small enough to allow the path to smooth out (no large spikes in movement), but also large enough so the path can smooth fairly quickly.

The other mode of movement in the update function is to avoid obstacles. When an obstacle is spotted by the cart’s laser, a list containing an estimation of the obstacle’s GPS position is created. The nearest swarm points are then given a copy of the obstacle list, and also have a flag set to cause them to avoid an obstacle in their next update rather than their normal path smoothing.

On the next update, the swarm points which are set to avoid the obstacle check to see whether they are the closest two points to the obstacle. If they are, they do another check to see if they are already a safe distance from the obstacle. If the points are already the specified safe distance from they obstacle, they do not move and set themselves back to path smoothing for the next update. If the points are not a safe distance away from the obstacle, they move themselves a safe distance to the left or right of the obstacle and also move the left or right endpoint of their line of movement in order to avoid the obstacle in the future.
Coding:

Documentation for source code was created using Doxygen and included with this submission.

Testing:

The class SwarmTester was included with this submission in the package tester.py. It is a simple program to generate swarm points from a list of control points and update the position of the swarm a given number of times. Data from updates is written to a text file called swarm.txt which can be easily imported into Excel.