1 Requirements

This lab involved doing basic obstacle avoidance to prepare for the future labs in which the obstacles are not known beforehand. The obstacle is a small cone, whose location is known beforehand and is given by its GPS co-ordinates. The cart is required to maneuver around the obstacle without touching it and leaving sufficient distance between the cart and the obstacle.

2 Introduction

3 Implementation

Since the location of the obstacle is known beforehand, this lab involved detecting if the cart is close to an obstacle. If the cart is within a range of the obstacle, we assume that the cart will collide with the obstacle and take remedial action by steering it away from its path. Once the obstacle is out of the range, we assume it to be safe to steer back and head towards the next waypoint.

The Obstacle Avoidance driver, thus, does not create any synthetic waypoints but instead just veers off from its path and then heads back again towards the waypoint. This approach was simpler than having to add a synthetic waypoint to the waypoints list maintained by the WaypointController. A new class called ObstacleDetector was written to achieve this. Depending on the necessary parameters, it determines when to start veering off from the obstacle. In this lab, the cart always turns right to steer away from the obstacle.

3.1 Obstacle Detector

The Obstacle Detector detects for an obstacle in the cart’s path. When the obstacle is static and known beforehand, this is done simply by getting the next obstacle from the obstacle list and then computing the distance and heading to this obstacle using the Haversine class. The Obstacle Detector has three different states on which it operates. These are IN_SIGHT, IMMINENT and CLEAR. If the obstacle is within 9 meters of the current position and the desired heading is less than 15 degrees, the obstacle is marked as IN_SIGHT. If the obstacle is within 4 meters of the cart’s current position and the error in heading to the obstacle is less than 10 degrees, the obstacle is marked as IMMINENT. Depending on these two states, we set the desired heading of the cart to steer it off from the obstacle and also set the throttle and braking accordingly.
If the cart is imminent to collide with the obstacle, it is slowed down while taking a sharp turn to turn it away from the obstacle. The methods exposed by this class are given below.

• detect\_obstacle()
  Detect for an obstacle given the current position and desired heading.

• avoid\_obstacle()
  Set the desired heading depending on the location of the impending obstacle.

In the process function, at each time step, the avoid\_obstacle() method is invoked to make sure the cart detects the obstacle and avoids running into it.

```python
# Obstacle detection and avoidance.
(status, self.desired_heading, dist) = \
    self.obdtc.avoid_obstacle(self.desired_heading)
if not status == self.obdtc.CLEAR:
    approaching_target = dist
```

The pseudo-algorithm to detect if the waypoint should be marked as covered is changed from the previous lab. We now read the LBO from the RDDF format and accordingly compute the midpoint of the waypoint. If the remaining distance is less than half the waypoint’s LBO, that waypoint is marked as “covered” and the cart moves on to the next waypoint.

```python
if remdist <= (self.nextwaypoint[3]/2):
    self.wpctl.waypoint_covered()
    self.odmtr.reset_odometer()
```

4 Results

As we see from the graph 1, the cart starts steering away from its path when it detects the obstacle. The points in the graph are plotted as a function of the cart’s speed at that instant. The cart decreases it’s speed when it is close to an obstacle or when it’s near a waypoint. We have used the “stop-and-go” approach to almost completely stop the cart for a smooth run of the cart when traversing the course.
Figure 1: Graph showing the route of the cart in avoiding a single obstacle