ERTS: Embedded Real Time System

<table>
<thead>
<tr>
<th>Version</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>5th November</td>
</tr>
<tr>
<td>Purpose</td>
<td>Lab Report for GPS waypoint follower and Obstacle Avoidance</td>
</tr>
<tr>
<td>Status</td>
<td>Complete</td>
</tr>
<tr>
<td>Author</td>
<td>Shreyes Ahir(sahir), Abhijeet kodgire(akodgire)</td>
</tr>
</tbody>
</table>
Table of Contents

1 Requirements ........................................................................................................... 3

2 Introduction .............................................................................................................. 3

3 Design and Implementation of square driver ......................................................... 3
   3.1 Design ................................................................................................................. 3
   3.2 Implementation ................................................................................................. 3
      3.2.1 Turning mode ............................................................................................ 4
      3.2.2 Straight mode ............................................................................................ 4

4 Design and Implementation of GPS follower ......................................................... 4
   4.1 Design ................................................................................................................. 4
   4.2 Implementation ................................................................................................. 5
      a) Pythagorean Theorem ..................................................................................... 5
      b) Havensine Formula ......................................................................................... 5
      c) Using Geopy Module ..................................................................................... 5
         4.2.1 Algorithm used to traverse way points. ..................................................... 6
         4.2.2 Speed Control ......................................................................................... 6

5 Results ....................................................................................................................... 7

6 Obstacle Avoidance Design and Implementation .................................................. 8
   6.1 Design ................................................................................................................. 9
   6.2 Implementation ................................................................................................. 10

7 Results for Obstacle Avoidance ............................................................................. 11

8 Conclusions ............................................................................................................. 11

9 Future Work ............................................................................................................ 11

10 Acknowledgement ................................................................................................. 11

11 References ............................................................................................................. 11
1 Requirements
The main purpose of this assignment is to design and develop driver for ERTS autonomous golf cart so that it follows given GPS (Global positioning system) waypoints. The drivers should maintain the course of golf cart well within the tunnel formed by connecting two way-points. The basic idea is to use the square driver which was designed in Lab1 assignment to adjust the turn radius of steering system to minimize the difference between current direction and desired heading.

2 Introduction
ERTS is autonomous experimental golf cart at Indiana University. Golf cart has bunch of onboard sensors which can be use for various applications like way point follower, obstacle avoidance etc. In this field run onboard GPS reader was used to continuously compute desired heading to follow the given path. Desired heading is based on cart’s current co-ordinates and co-ordinates of next way point. Once cart has come close enough to the way-point, cart has to slowly head towards next way-point. The path is given as list of way-points as specified in RDDF format. The way-points are nothing but 2D circles where centre of circle is at co-ordinates specified by latitude and longitude and radius of circle is defined by parameter called lateral boundary offset (LBO). Also each way point is numbered and includes the speed limit for that segment. LBO is given in meters where speed limit is mentioned in meters per seconds.

3 Design and Implementation of square driver

3.1 Design
The cart was designed to run on two modes, straight and turning. In the turning mode, the compass heading is moved towards the target heading. While the cart is in the straight mode, the distance travelled by the cart is continuously read while the heading is maintained as a constant. Once the distance travelled by cart in this mode is the same as the side of the square that needs to be traversed, the mode is changed to turning.

3.2 Implementation
In the square.py (Lab1 Assignment) P value which is a PID variable based on which we control the turn radius and correction of the ERTS vehicle was tuned. The value was 2. In the waypoint follower assignment P has very high we observed that though the turn was sharp we were not able to keep the vehicle within the tunnel formed by the joining of the LBO of two way points. The problem faced was that the high P value would make the vehicle to move in a zigzag order in a straight path. And small P value
was not helping in making sharp turns. We decided to use dynamic P values such that instead of computing the P value continuously we decided to use a step function in which we increment the P by certain step value and in the entire course use P value from it. While moving in the cart we observed that the P value is inversely proportional to the speed so at high speed we need low P value.

### 3.2.1 Turning mode
The Entire track is divided in to two parts turning mode and Straight mode.

Waypoint 1 = W1
Waypoint 2 = W2
Distance between two waypoints = D1 = |W2 – W1|
Straight mode = (D1-5) meters
Turning mode = Till the abs error is less than 25.Error between heading and bearing

Turning mode starts when the cart reaches 5 meters before reaching the waypoint. At this point the change in the heading is calculated and P value is selected based on the change needed in the heading. For higher change P value should be higher which helps in achieving sharper turn.

### 3.2.2 Straight mode
After reaching the waypoint and the error is less than 25 between heading and bearing at that point the mode is changed to straight and it remains straight till the cart is just 5 meters before reaching the waypoint. P value for the straight distance is kept minimum .3 because in straight line correction required is minimum. One problem though is that the track is not level field hence when the cart moves downhill the cart do deviates from its straight path and does a zigzag turn. In downhill the cart’s speed increases at same throttle so becomes bit more sensitive to small change.

### 4 Design and Implementation of GPS follower

#### 4.1 Design
In square.py we used 4 different headings values (0, 90, 180, and 270 degrees) for desired heading based on the direction where the cart is heading. In GPS way point follower; we needed to consider the distance
and angle to next way point. Since the angle and distance to next way point is depends upon the position of next way point. We needed to calculate the distance and angle between current position and next way point continuously.

4.2 Implementation
There are different ways to calculate distance and direction between two GPS points.

a) Pythagorean Theorem
If the distance is less than about 20 km (12 mi) and the locations of the two points in Cartesian coordinates are X1,Y1 and X2,Y2 then the

Pythagorean Theorem

\[
D = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}
\]

This formula is based on assumption that earth is flat and hence formula is erroneous.

b) Harvensine Formula
Assuming the earth is sphere with radius r and location of the two points in spherical co-ordinates are (lon1, lat1) and (lon2, lat2) then the Harvensine Formula for distance is as follows.

\[
d = R \cdot \arcsin(\min(1, \sqrt{a}))
\]

This formula will give you exact results. The Harvensine formula gives erroneous results if two points are on opposite side of earth.

c) Using Geopy Module
We evaluated above two approaches and we decided to go with third party package. We used Geopy module to calculate the distance and angle between two given points. Internal implementation of Geopy module is out of scope of this document.

Here are the two formulae from distance package that we used:
meters_to_target = distance.distance(current_latlon,waypoint_latlon).kilometers * 1000
heading_to_target = distance.distance(current_latlon,waypoint_latlon).forward_azimuth

meters_to_target gives you distance between two points and heading_to_target gives you angle between two points.

4.2.1 Algorithm used to traverse way points.
The list of waypoints is read from the file and stored in array. Distance and heading to the first given way point is calculated by using Geopy Module mentioned above. Using the distance and the heading, the heading controller module computes the inverse turn radius that is required to be applied to the steering amplifier to steer the cart to the waypoint. In this way cart is driven towards first way point until it is 5 meters away from the way point. Once cart is traveled this much distance, next way point from the way point list is chosen and repeated same steps as mentioned above. This algorithm is used for all way points given in input file and eventually whole course is traversed three times.

4.2.2 Speed Control
Initially we adopted the approach of a look up table but found it very complicated for this task so we used a discrete step function where the speed of the cart is increased or decreased depending upon the position of the cart. At the beginning the throttle is set to 70 % and is reduced to 60 % when the cart is less than 5 meters away from the waypoint. When the cart leaves straight mode and enters turning mode throttle is set to 50%. This variation in speed helps us to achieve smooth turn and good vehicle control at high speed keeping the cart in the tunnel.
5 Results

Figure 1. Visual telemetry for midterm trial to follow GPS way points.
6 Obstacle Avoidance Design and Implementation

After the way point follower driver in place, next task was to design a driver to avoid obstacles. Golf Cart has to follow the course defined by a list of way points. This course will be introduced with static obstacles which Cart has to avoid carefully. The Sick_laser component is designed to dynamically produce an obstacle list based on a static list of potential obstacles. This Sick_laser component will be used to determine the obstacle position. The cart is required to traverse through the entire list of waypoints while avoiding the obstacles and when it has finished traversing the list once, it has to loop through the same way point list again.

There are lots of things to be considered during driver for obstacle avoidance. The important things which we needed to consider were as follows

- Distance of the obstacle from the current position of the Golf Cart.
- The direction of obstacle from current position of Golf Cart.
- Calculation of synthetic way point in order to avoid the Obstacle.
• Determine if obstacle is appeared in path of Golf Cart.

6.1 Design

The method used for obstacle detection is based on simple co-ordinate geometry.

The slope between the cart’s current position and the destination way point is given by

\[
Slope(\text{curpos}, \text{destpos}) = \frac{(y_2-y_1)}{(x_2-x_1)}
\]

Similarly, the slope between the cart’s current position and obstacle position is given by

\[
Slope(\text{curpos}, \text{obspos}) = \frac{(y_3-y_1)}{(x_3-x_1)}
\]

\[
\tan(A-B) = \frac{\tan A - \tan B}{1 + \tan A \cdot \tan B}
\]

Using above formula we get \(\tan(\text{Slope(\text{curpos}, \text{destpos})} - \text{Slope(\text{curpos}, \text{obspos})})\) this gives us the tan of the difference between the two angles. We then calculate the inverse of tan to get the angle in radians which is then converted to degrees. Then the cart is said to be heading towards the obstacle. And if the difference between two slope is greater than threshold (± 10 degree) then obstacle is assumed to be not in the path of Cart and hence obstacle need not be avoided. Based on this formula it is possible to determine if the obstacle is appeared in the path of Golf Cart or not. The next thing is to determine orientation of obstacle according to the Cart’s current direction.

The method of determining the orientation of the obstacle with respect to the cart heading is also based on the concept of slope in co-ordinate geometry. Slope of a line determines the tangent of the angle that the line makes with the horizontal axis. The value of slope is directly proportional to the angle it makes with horizontal.
If \((\text{Slope}(\text{curpos, destpos}) - \text{Slope}(\text{curpos, obspos}))\) is positive then the obstacle is considered to be right of the path taken by Cart hence the Cart is steered to left to avoid the obstacle. Similar way, if \((\text{Slope}(\text{curpos, destpos}) - \text{Slope}(\text{curpos, obspos}))\) is negative then the obstacle is considered to be left of path taken by Cart hence Cart is steered to right to avoid the obstacle.

Next important thing is to calculate the synthetic way point. Once the direction and distance to the obstacle is known, the destination function in geopy module is used to calculate the synthetic point. The call to destination function looks something similar to following.

\[
\text{synthetic}_\text{latlon} = \text{distance}.\text{destination}(\text{current}_\text{latlon}, 0, \text{dist} + 0.002, +/- 30)
\]

### 6.2 Implementation

The obstacle avoidance driver was built on the GPS way point follower driver. The way point follower driver already had the capability to follow a course using GPS points is modified to handle hardcoded obstacles introduced in the path. The obstacles are tracked in every cycle by computing the distance and the bearing to the obstacles. When the cart comes close to the obstacle, a new synthetic way point is introduced and Cart is steered towards new synthetic way point to avoid the obstacle. Once this is done, the driver is modified to handle obstacles returned by the synthetic laser. The driver should poll for these obstacles by adding a reader for the ./sick_laser/sick_laser s file in the code. When obstacle is detected the driver calculates the slope to obstacle to determine the orientation of obstacle with respect to path of the Cart. If the obstacle is in the path then the new synthetic way point is inserted and cart is made to drive towards new synthetic way point avoiding the obstacle. Once the synthetic point is reached and obstacle has been avoided Cart returns to original GPS course driving toward next way point.
7 Results for Obstacle Avoidance

Fig 3. Visual telemetry for obstacle avoidance

8 Conclusions
GPS way point follower works correctly in almost all scenarios. Also the steering control module is well tuned results in smooth turning at waypoints with negligible over steering sometimes. Adding path planning in existing driver will definitely help for better performance and obstacle avoidance purpose.

9 Future Work
Modification to the current obstacle avoidance driver is required to keep the cart within the corridor.

10 Acknowledgement
We would like to thank Caleb Hess for helping us to test our program on Golf Cart and answer all our queries. Also we would like to thank Prof Johnson and Scott Dial for their continuous suggestion to improve our program.

11 References