P545 EMBEDDED REAL TIME SYSTEMS

FINAL PROJECT

Indiana University Bloomington

School of Informatics and Computing

Semester: Fall 2011

Instructor: Prof. Steve Johnson

Group Members:

Doga Tuncay

Hasan Kurban

Kurt Weisman

Submission Date : December 9, 2011

Doga Tuncay
Kurt Weisman
Hasan Kurban

OUTLINE

CONTENTS:

Doga Tuncay
Kurt Weisman
Hasan Kurban

1.INTRODUCTION

There is an embedded computer system that is embedded in a golf cart and with that a specific testing field is given to users in order to complete lab assignments and a final project by using python programming. In this paper we explain some of the approaches for the lab assignments and final project for the Embedded and Real Time Systems course. Throughout this course we developed several approaches for the lab assignments and final project objectives. This document consists of the descriptions for the framework, the approaches that we developed for the objectives, important parts of python coding, test field and simulator screenshots.

We used a real golf cart on which a computer is embedded and an ERTS simulator in Ubuntu image to solve the problems. The lab assignments, we started with the simple tuning projects, such as handling the steering, then we continued with different concepts, such as GPS Following, Path Planning via doing tactical maneuvers, designing a better path and so on, and we tried to do Obstacle Avoidance. Then we combined all these ideas with a one final system by adding some tuning such as Speed Control, better Path Planning strategies and more intelligent turns.

2. ARCHITECTURE

There are several component systems in ERTS architecture. By mounting the CartFs, we used the SyncFS server system. Also, we used SyncFs to test our driver programs in an environment which is similar to a real world implementation. By using the SyncFs file system we used CartFS to communicate with the vehicle. We used both sick_laser and syn_laser components for the obstacle avoidance objective, as these two laser components are designed to detect obstacles. We first used these laser components for real time obstacle detection, and later used them to test the python codes in computer environments for obstacle detection in the ERTS simulator as well.

Also, there are some sensors that have compass, odometer and GPS functions. CartFS also has some components that help developers to set the throttle and breaking values, steering control and direction.

There is a simulator of CartFS in the Ubuntu image. Ubuntu image makes it possible to code without being in the actual cart. The simulator of the CartFS denotes the performance of the cart, but these results are not exactly the same when we drive the actual golf cart in the test field. Furthermore, in the test field we need to take it into consideration obstacles of the course, slope of the field… etc. And finally, in order to do the lab assignment, square.py python file is provided**.**

3. REQUIREMENTS:

In this section, we explain the purpose of the lab assignments and the final project without giving excessively technical information.

The purpose of the first assignment is to make certain corrections in square_raw.py and square_sensor.py files to make a golf cart run a square course with 3 m rounded corners. The purpose of the second assignment is to modify the code in order to make it follow the GPS positions, which are defined in RDDF file in CartFs. In the first lab we design a heading controller that adjusts the turn radius of the steering on each clock cycle to minimize the difference between the current compass heading and the desired heading. In the second lab we use the same controller to follow a course that is defined by the list of points in the Earth coordinate system according to longitude and latitude. In this report we will mainly explain the last two lab assignments which are the most importance among the others and the final project.

The purpose of the third assignment is to make the golf cart move through the predefined waypoints. As we move the golf cart through the predefined waypoints, we need a well-defined path plan to ensure the cart remains within the corridor.

We need to introduce new synthetic points in order to succeed in this goal. The behavior of the cart will be changed with these newly introduced synthetic points. We designed our approach by swinging to the opposite direction of the turn while entering a turn. We also needed to make some adjustments for throttle, braking, and P value in order to make the turn easier with the newly created points. Finally, the purpose of the last assignment is for us to make the cart avoid a series of cones (obstacles) that are placed throughout the course. We use Sick laser to get the obstacle list from the laser in the test field. Also, we use Synlaser to create predefined obstacles in the simulator. The direction of the golf cart is set based upon the location of the obstacles in order to achieve avoidance of the obstacles.

We divided the design part into two different parts. In the first part we mention path planning and obstacle avoidance. We don't separate path planning and obstacle avoidance since we complete obstacle avoidance by inserting the capability of avoiding obstacles into the code of the path planning code. We first state path planning, after which we will explain obstacle avoidance. In the second part we will explain our final project

4. **Labs and Final Project Ideas**:

In this section, all significant calculations and our approaches will be explained for the lab assignments and the final project. After that, we will explain path planning. Once we show the significant points for path planning, we will explain our ideas for obstacle avoidance. Finally, we will explain our ideas for our final project.

The initial problem we faced with the cart was keeping it on the course while moving from one waypoint to the other. By adjusting the throttle percentage value and the braking percentage value however we were able to overcome this problem. After testing our code in the test field and running our code in visualizer, we realized that we could find an efficient way for keeping the cart on the course.

A. SQUARE DRIVER

The first lab's task was simply to tune the steering control of ERTS to drive in a square without swerving back and forth after turning a corner. This was achieved by adjusting the turn_p_term to approximately .003.

Doga Tuncay
Kurt Weisman
Hasan Kurban

B. GPS FOLLOWER

The second lab was to use the GPS capabilities of ERTS to follow a predefined set of waypoints around the test track while staying inside the corridors between waypoints.

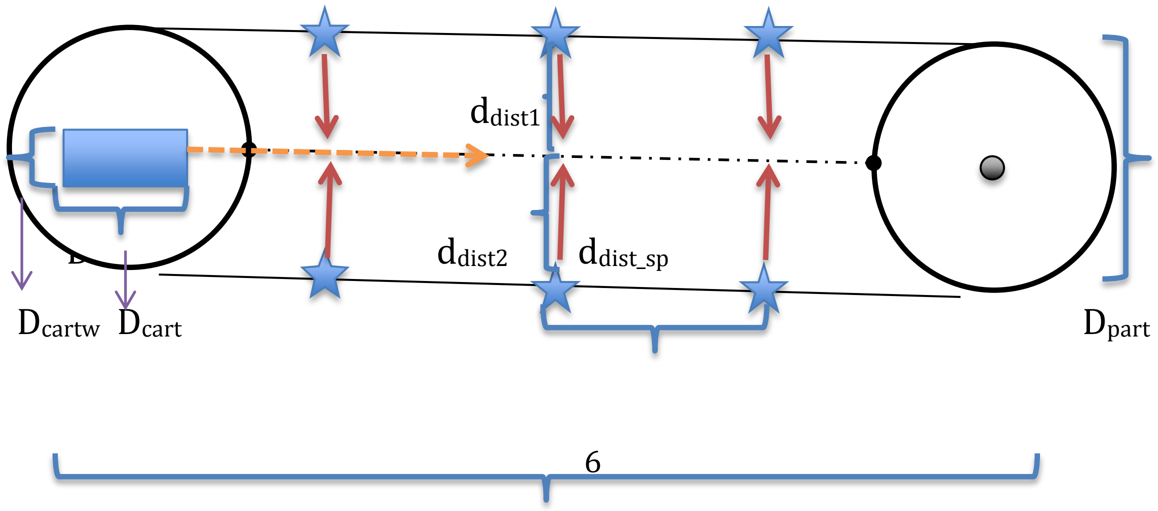
C. PATH PLANNING

1. Horizontal Control:

One of the ideas we had for path planning was Horizontal Control. The initial aim was to create synthetic points all along the corridor by dividing the path into some smaller parts which cannot be smaller than the length of the cart (We approximated it to 3 meters) and we were going to put these points on the both sides of the corridor and try to have the same distance from them.

Prof. Johnson gave us an example from the last years' projects, which involves in the same problem. In the project, they used those points as a dominating force from the both sides of the cart to make it stay in the line.

However, we found a better way to make this path planning better. We will explain it afterwards.



D_{wl}

Figure 4.1

- D_{cart} : Length of the cart
- D_{dist_sp} : Length of check points
- D_{wl} : Length of the road between two waypoints
- D_{cartw} is the width of the cart
- $D_{dist1,2}$ is the perpendicular difference from cart to the borders of the path and they have the same distance
- D_{path} is the width of the path
- We try to check the distance from the cart to the borders of the way in every 3 meters (the approximated length of the cart) by using GPS system. We will keep the same distance from the left and right side of the cart by using check points and the calculation of distances.

$$D_{dist1,2} = \frac{Dpath - Dcartw}{2}$$

Synthetic Points will be defined in every Path/2, Path/4 .. which won't have the length more than 3 meters. We will be explaining the formulation of the synthetic points in the next strategy.

2. <u>Vertical Control:</u>

In order to avoid the problem mentioned above, we figured out that adding certain synthetic points between the two consecutive waypoints is an effective solution. The case is where to put those points? Along the corridor or on the way you are going?

We think that, putting the synthetic points on the middle of the path will help us to get a better solution, stay in the middle and go in a straight line and in this way we will be doing this control by using the half number of synthetic points that we used in Vertical Control. Therefore, we will have less calculations and more performance.

In brief,

1. Vertical Control is easier and logical

2. Managed by creating synthetic points between two waypoints
3. Targeted the synthetic points
4. Done by dividing path by 2, 4 , 8 , 16 ..
5. There is no need to use synthetic points for the distances between two waypoints < certain distance

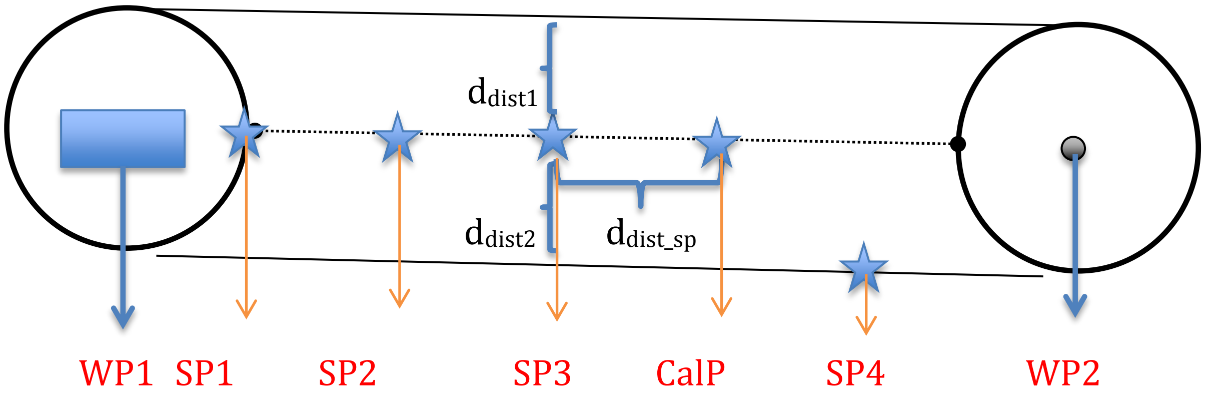


Figure 4.2

To be able to achieve this control we needed to know which waypoint we are departing from and which way point we are targeting. So we need to know the coordinates of these two waypoints to be able to do necessary calculations.

WP1 (x1,y1) = Coordinates of the waypoint that the cart moves from

WP2 (x2,y2) = Coordinates of the waypoint that the cart is targeting

In the visual we have three synthetic points that are defined in between two waypoints which have the coordinates that are shown in the table.

SP2	**SP3**	**SP1**
$\frac{3x1+x2}{4}, \frac{3y1+y2}{4}$	$\frac{x1+x2}{2}, \frac{y1+y2}{2}$	$\frac{7x1+x2}{8}, \frac{7y1+y2}{8}$

Table 4.1

We place the last synthetic point wider on the turn from the waypoint that we are targeting and from the waypoint 2 to the 3 to be able to add our tactical maneuvers strategy.

3. Tactical Maneuvers (Part of the Final Project)

As we explained before, we wanted to give the cart have some space to be able to lean to the corridor while it is turning to waypoint with a wider turn. To be able to make an wide turn we had several ideas that are shown in the graphics:

3.1

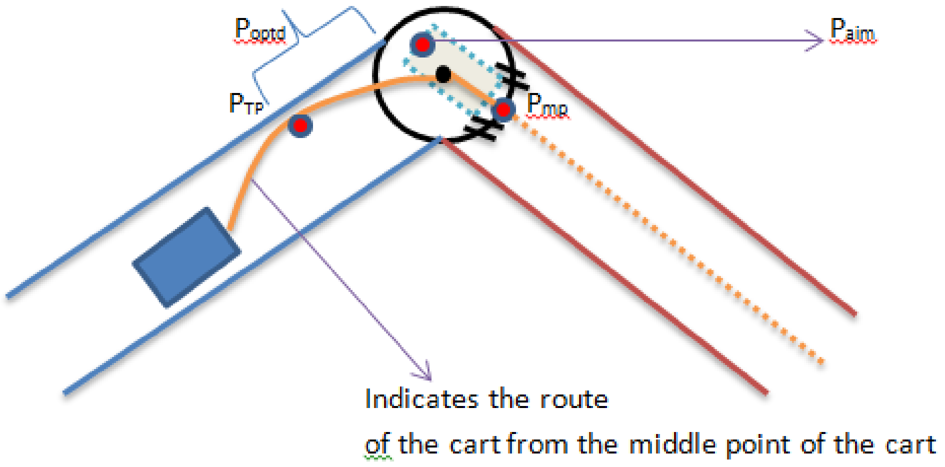


Figure 4.3

In this strategy we tried to define a leaning point to the corridor and create another synthetic point which divides the arc into two. P_{opt} shows the optimal distance for us to make the optimal turn. However, according to the discussions with Prof. Johnson we decided to not to use this strategy, because this method is physically is not possible.

3.2

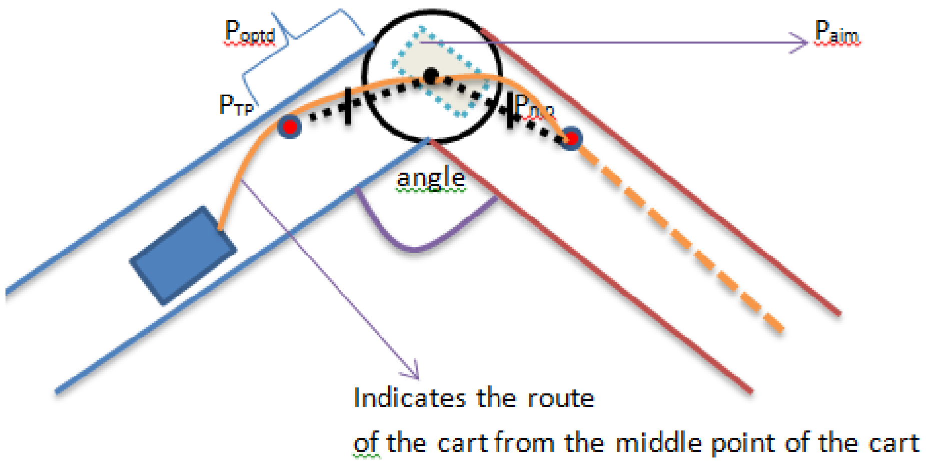


Figure 4.4

In our second strategy we had two leaning points (These points only target the middle of the cart in the visual, but the actual points are leans to the corridor). These two points have the same distances from the corner. We were calculating these points by using the turning angle, but we had problems with it and we agreed to use another calculation method.

The calculation of these leaning points which is the fourth point is in our case, the synthetic point must be different. We assume there is a calculation point (CalP) hypothetically. The calculation of the CalP is demonstrated below:

We calculate a new waypoint on the line between two waypoints. This new point is close to the second waypoint that the cart is passing through.

CalP
$\frac{x1+7x2}{8}, \frac{y1+7y2}{8}$

Table 4.2

Here is a zoomed out version of the graphic above.

We have some geometrical calculations here to demonstrate how we find the point.

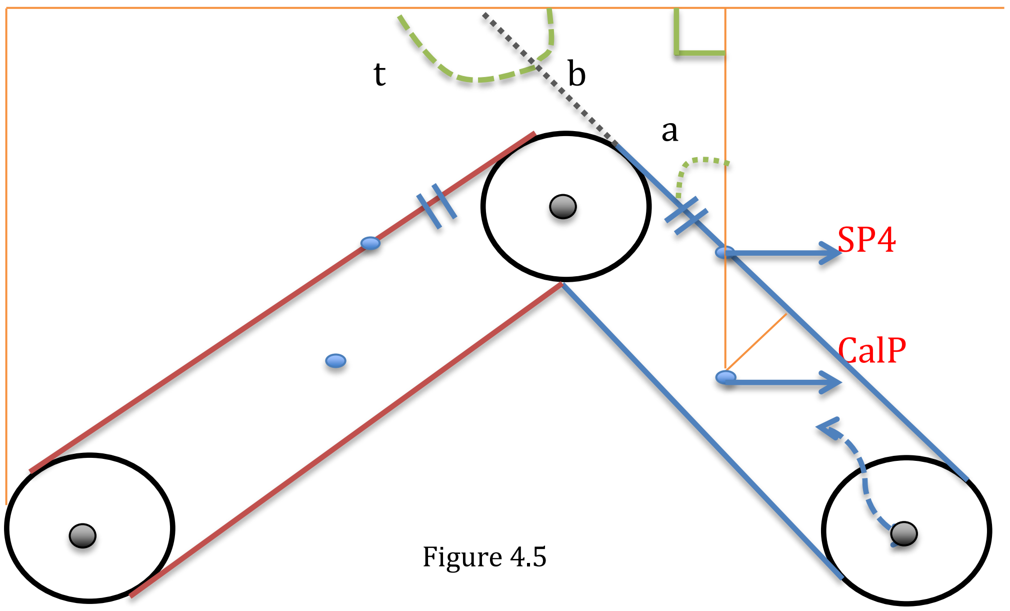


Figure 4.5

- **a=alpha, b=beta, t=theta**
- **b=180-t**
- **b=90-a**
- **t=arctan(b)**
- **t=arctan(y coordinates/x coordinates)**
- **t=arctan((y2-y1)/(x2-x1)**

As we calculate the calculation point on the line we can calculate the leaning point SP4. We calculated the angle of SP4 by using these lengths.

WP1	WP2	lenofx	lenofy
(x1,y1)	(x2,y2)	x2-x1	y2- y1

Table 4.3

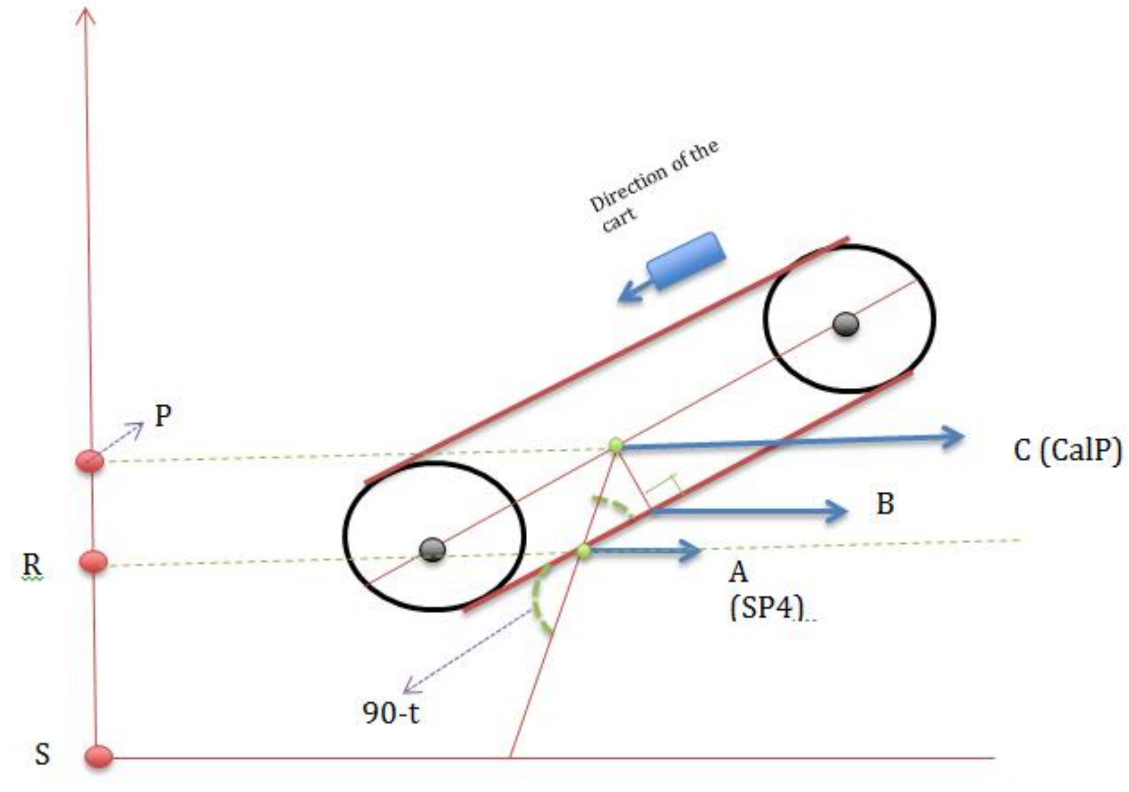


Figure 4.6

- **Angle A of triangle ABC is also 90-t**
- **|BC| = 2d/2 = d**
- **sin(90-t) = d/|AC| ➔ |AC| = d/sin(90-t)**

The X-axis of SP4 will be the same with CalP, which is x0. However, we need to calculate the y axis. Let (x_3,y_3) be the aimed point's coordinates.

In our figure:

- **|KM| =y_3**
- **|KL| = |AC| = d/sin(90-t)**
- **|LM| = y_3 – (d/sin(90-t))**
- **|LM| will provide us with the y axis of SP4.**
- **SP4 = (x_3, y_3-(d/sin(90-t)))**

If we calculate the equation above it will give us the exact synthetic points' locations.

One question arises after calculating the points above however. The question is how we should determine the place of the last synthetic point? To be able to answer this question we need to check which way the cart is going now and which way it is going to go. To be able to find out which direction the cart is moving we use a predefined forward_azimuth function. We first calculate which direction the cart is going while moving WP1 to WP2 with this predefined forward_azimuth function.

4. Speed Control by setting throttle, braking and P value (Part of the Final Project)

After we set new synthetic point, we need to improve one more point. In path planning one of the most important things is keeping the cart inside the corridor while we are reaching the maximum speed as possible. While we are moving from waypoint to waypoint, the throttle percentage and braking percentage is controlled by how far the cart is from the next waypoint. The throttle percentage is getting smaller and the breaking percentage is bigger while we are getting closer to the next waypoint by cart. We can make the car better turns by decreasing the throttle percentage. Also, It is not a good idea that setting the throttle and the breaking percentage at the same time for the mechanical equipment of the cart.

In order to identify the correction during the turns, we use P term. When we make a sharp turn, we will have big error, so the cart will need big correction. This is what is needed for the steering correction. In order to find out whether a turn is sharp or not, the angle between where the cart is coming and to where it is going is calculated. The figure below shows the calculation of the angle.

D. Obstacle Avoidance

So far we have navigated the test track with GPS and with a path planning algorithm. Finally, in this lab we used ERTS' front-mounted laser range finder to avoid obstacles placed around the test track.

Our first strategy is finding the distances from left and right of the obstacles and decides whether you go left or right by comparing the distances.

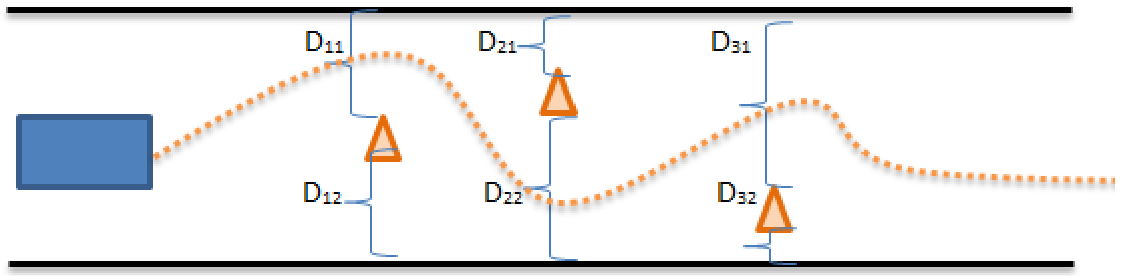


Figure 4.7

Pseude Code:

```
If obstacle_lat >= wp_lat:
    If obstacle_lon <= wp_lon:
        new_sp = [obstacle_lat + c, obstacle_lon +c]
    else:
        new_sp = [obstacle_lat + c, obstacle_lon - c]
else:
If obstacle_lon <= wp_lon:
        new_sp = [obstacle_lat - c, obstacle_lon + c]
```

The c is a constant that we add to define our new synthetic point. We should have relatively small constant which won't cause us get out of the track but also not hit the obstacle too. The (-,+) defines the direction' –' stands for left and ' +' for right.

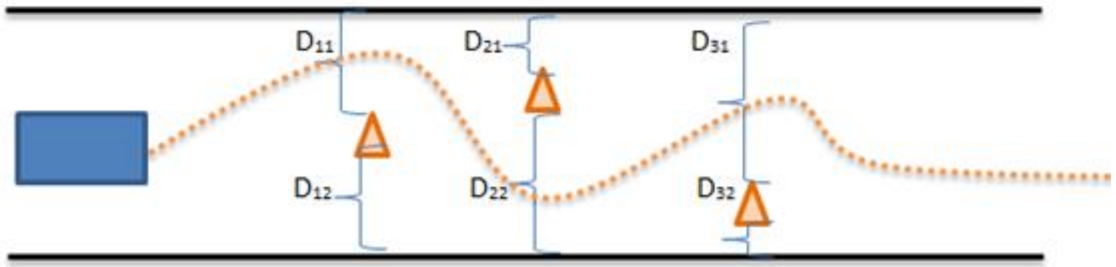


Figure 4.8

This method is pretty useful works for every case. We have another approach which is implemented.

We find the angle between from cart to obstacle and from the cart to the next waypoint, is checked with using forward_azimuth function. If the angle is smaller or equal to some angle degrees avoid from the obstacles.

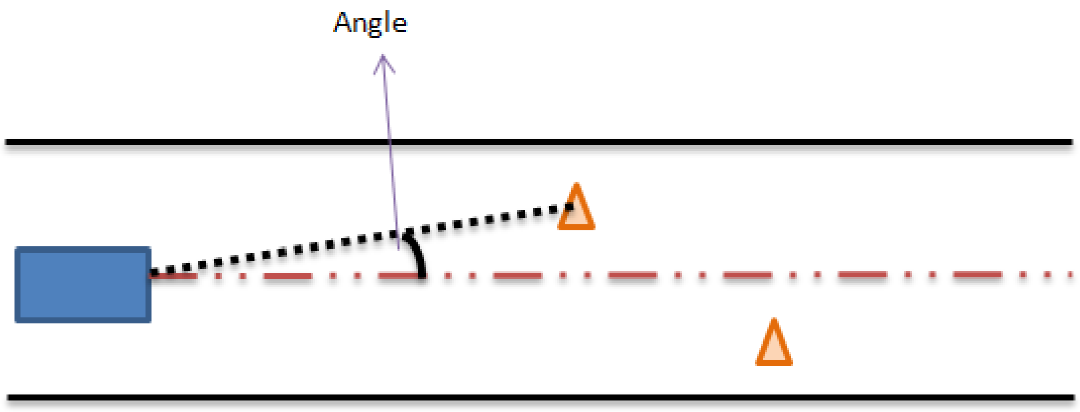


Figure 4.9

Why we implemented this idea is :

- it is easier
- it eases number of calculations

5.TEST RESULTS

Lab Report 1: Steering Control

In the first lab we tested the square_raw.py and square_sensor.py by tuning steering_sensitivity and turn_p_term.

Doga Tuncay
Kurt Weisman
Hasan Kurban

The square course that we used to test our cart had a straight length of 10 meters with a corner radius of 3 meters. There was an upwards slope from west to east. There were no weather related difficulties during the test. The square course values were assigned in the .py files under the navigation parameters that are, 'sq_side': 10.0, 'sq_corner': 3.0. The target_heading helped us to determine turning direction with some calculations. It calculated the distance the cart travelled and compared it with the length of the straight line (10 meters), when it travelled the defined distance it turned 90 degrees from the corners.

```
heading_error = float(compass['heading']) -
float(nav['target_heading'])
```

In this equation we compared target_heading with the cart's current heading to find which direction the cart should go.

Our smallest turn had a 3 meter radius. No matter what the steering correction was, we could not turn sharper than 3 meters radius. The sharpest turn was 3 meters according to some calculations of the Cart's length etc. If we tried a smaller radius than 3 meters, it would harm the Cart, so we stuck with the 3 meter radius.

```
return steering_correction, heading_error
```

We returned two values, that is what any embedded systems should return.

We had a control system called 'control' that controls our steering options that are turn and straight. Actual heading shows us that we close enough to target_heading by calculating the error_heading. If this error was smaller than the slope, then it could decide whether to travel 10 meters or not. Then it made its turn.

The navigation parameters are defined below. Steering_sensitivity was our value that is needed to be changed in the square_raw.py code and turn_p_term in the square_sensor.py code.

```
nav = {
      'mode': 'auto',                # autonomous mode
      'enable': True,                # Driver enabled
      'sq_side': 10.0,               # meters
      'sq_corner': 3.0,              # radius in meters
      'speed': 50.0,                 # fixed for this assignment
      'steering_slop': 5.0,          # degrees
      'steering_sensitivity': 0.0035, # TUNED THIS VALUE #
      'control': 'turn',              # {'turn', 'straight'}
      'target_heading': 0.0,          # degrees from magnetic north
      'mark': 0.0,                    # starting position in meters
      }
```

```
 driver_s = {
     'clock': 0,
     'enable': True,
     'direction': 0,
     'side_length': 10, # meters
     'corner_radius': 3, # meters
     'turn_state': 'turn',
     'turn_slop': 5, # degrees
     'turn_P_term': 0.03, # <<<!!! TUNE ME, please.
 }

 driver_s_doc = {
     'clock': "The clock value on which this data was written.",
     'enable': "True/False - stops reads on driver device",
     'direction': "The target heading of the driver",
     'side_length': "How long a side of the square should be (meters)",
     'corner_radius': "How large of a radius the corners should be (meters)",
     'turn_state': "What state of the turn we are in (turn/straight)",
     'turn_slop': "How close to straight we must be to be going straight",
     'turn_P_term': "The P term of the PID controller for steering",
 }
```

After we tuned these values we achieved better and smoother squares. We applied different sensitivities that are shown in the next part.

We observed that, the course was inclined which affected Cart's speed that of which influenced steering. Because of the incline from the west to east we have shifted squares. In other words, inclination affects speed and speed affects the turning point. As you go faster, the vehicle travels further as the steering moves to a new radius.

The current steering sensitivity (0.03) without any changes produced this square :

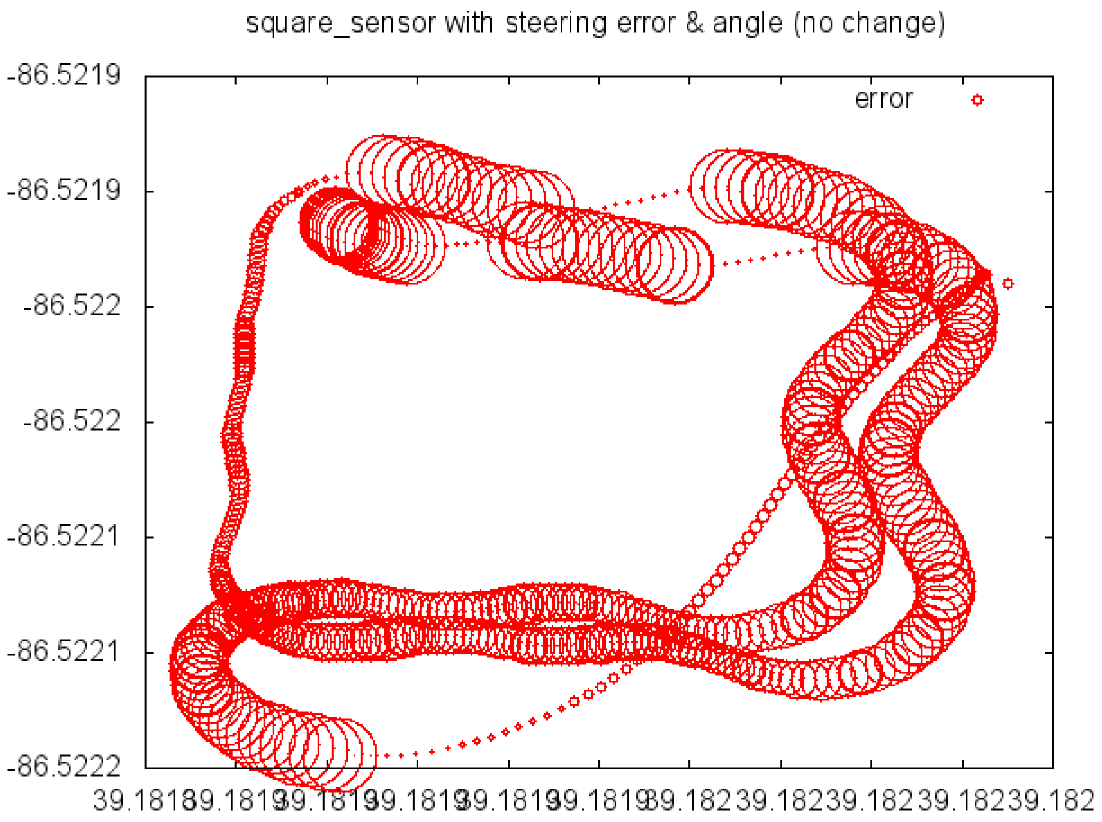


Figure 5.1 No change in the sensitivity

The square is shown above currently is not a smooth square it has more steering error with zigzags.

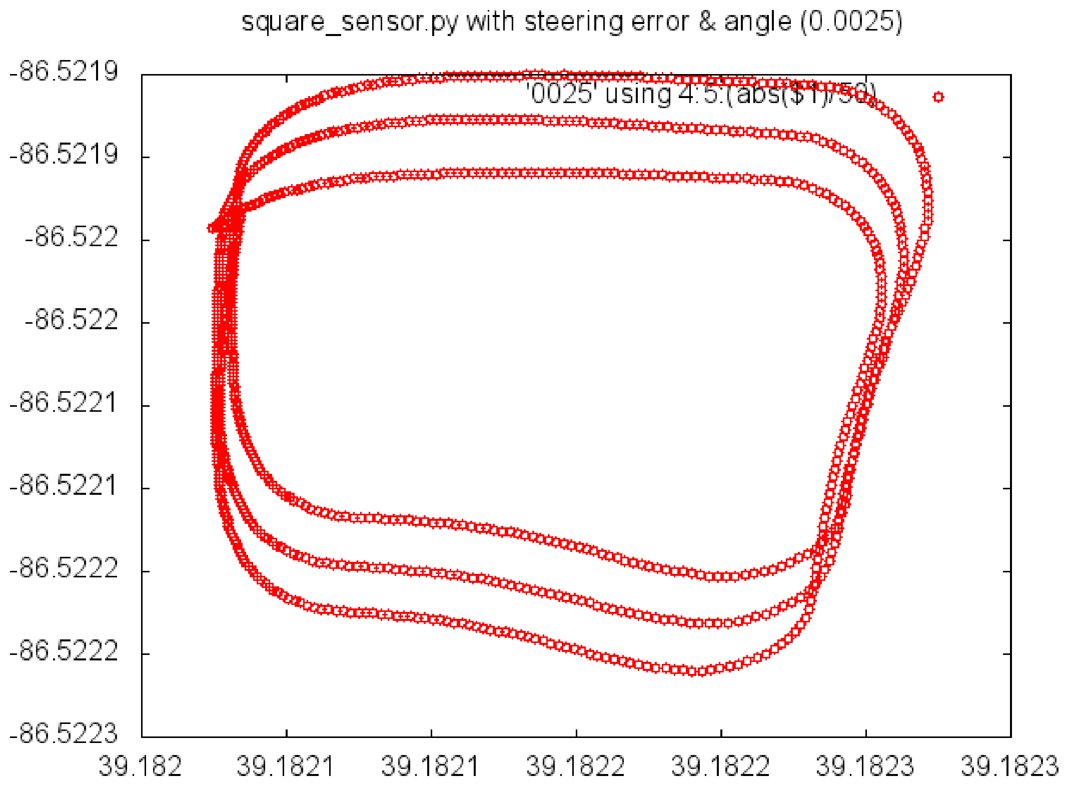


Figure 5.2 sensitivity = 0.0025

In here we had a better square, but we can still observe the shifting which is not something we can prevent.

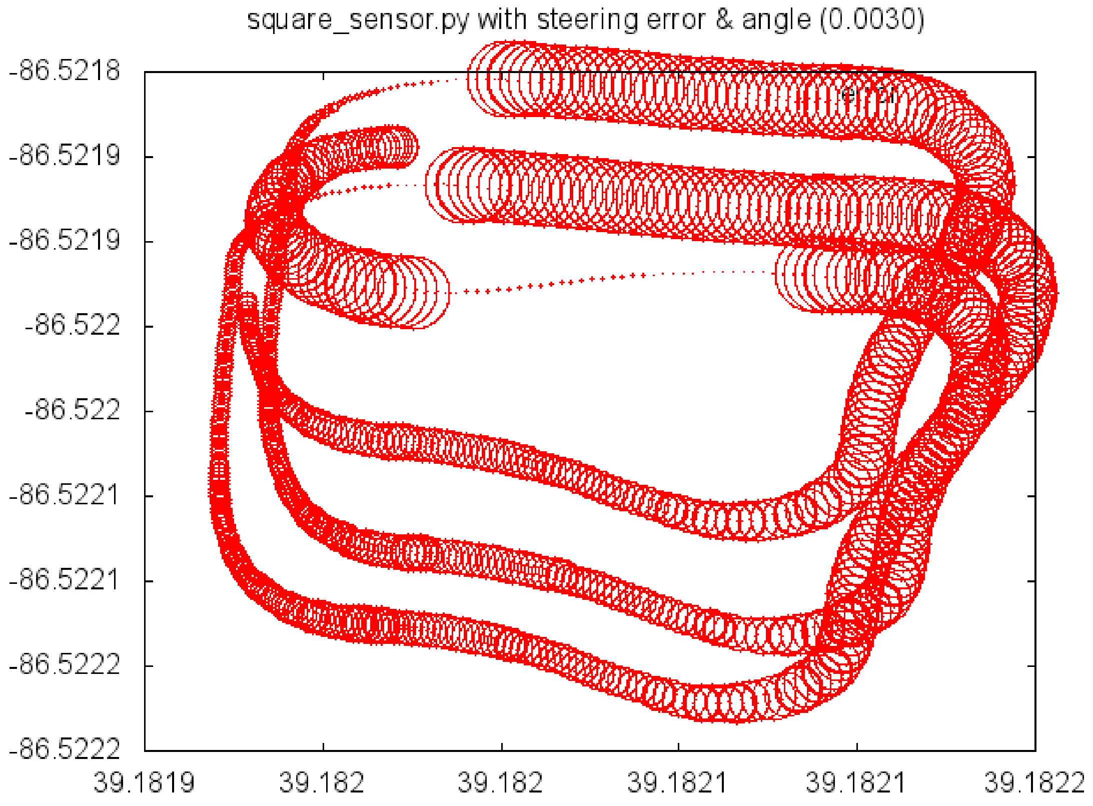


Figure 5.3 sensitivity = 0.003

Finally, the better result is shown in the following graphic which had the better steering and smoother square. As can be seen in all the graphics, we didn't have the same speed too. That is also because of the inclined course that we had. In our first lab, we observed that the best steering parameter is 0.0035.

!!!!

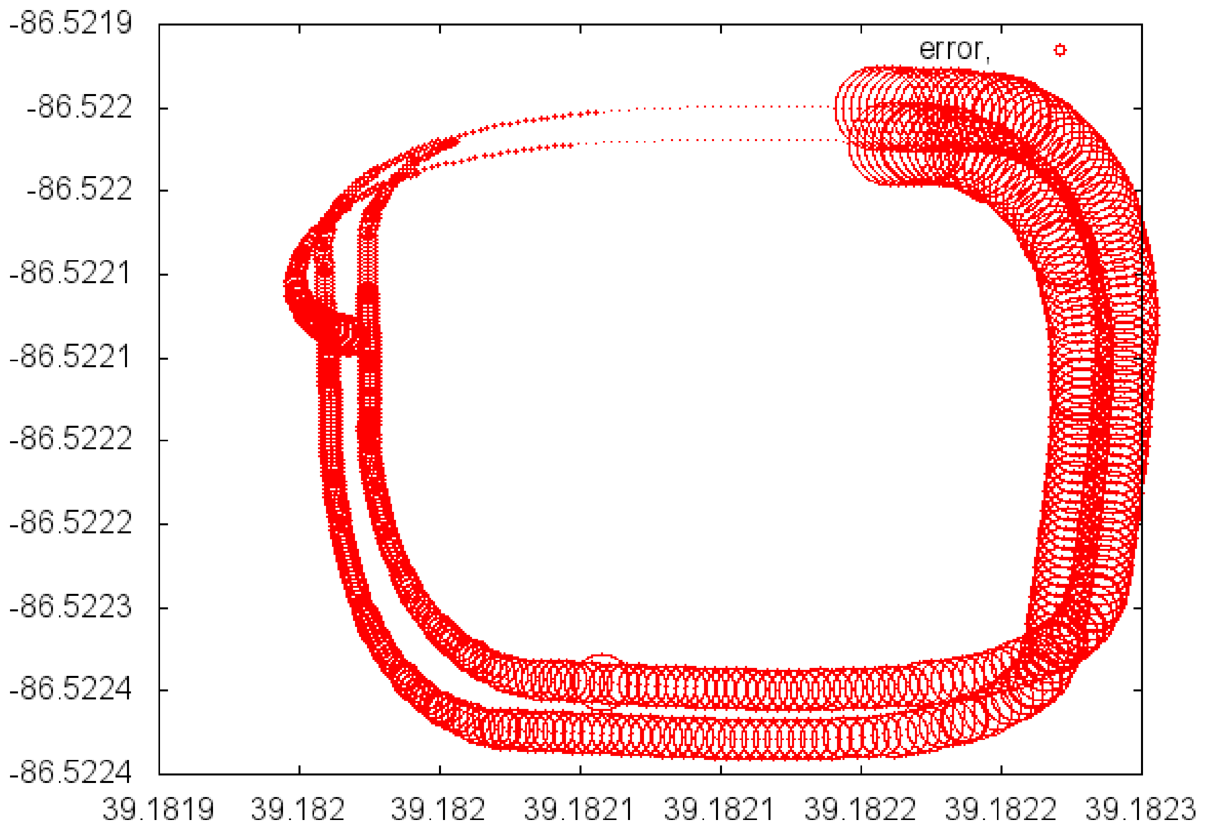


Figure 5.4 sensitivity = 0.0035

We ran square_sensor.py in the simulator by tuning the term_p_term, here are some screenshots:

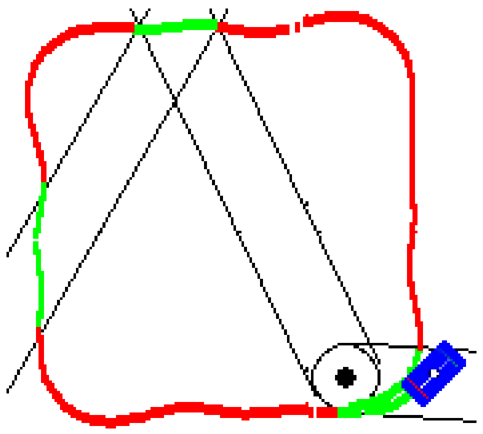


Figure 5.5 term_p_term=0.035

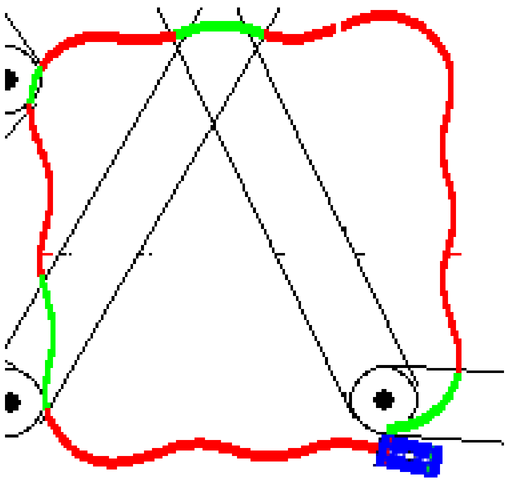


Figure 5.6 term_p_term=0.055

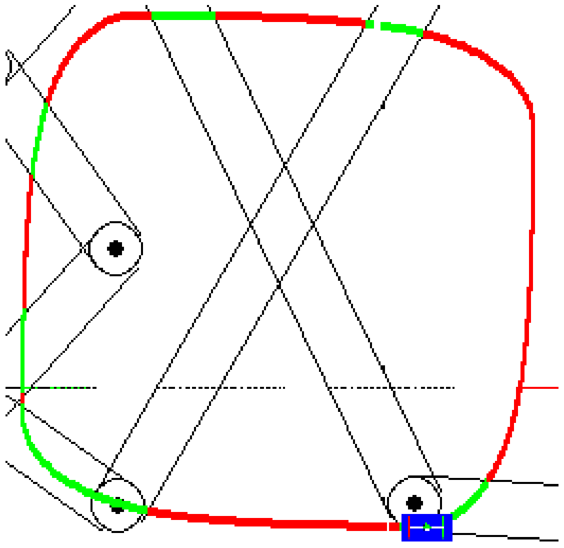


Figure 5.7 term_p_term = 0.0035

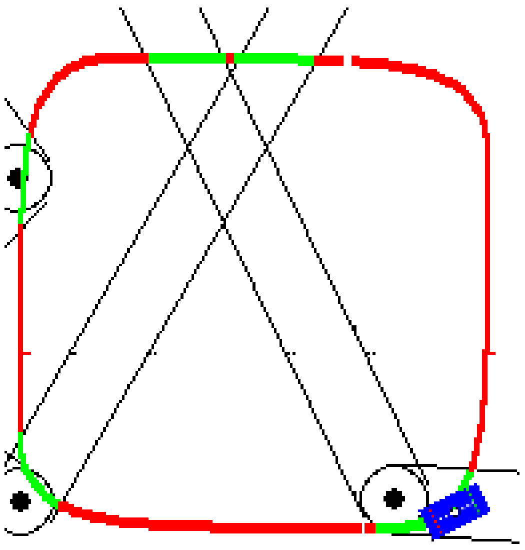


Figure 5.8 term_p_term = 0.0065

As you can see from the screenshots, lower term_p_term we got better squares.

Lab Report 2: GPS Follower

In the second lab we needed to use the same controller to follow a course that is defined by the list of pints in the Earth coordinate system with longitudes and latitudes.

In this assignment our component is given a list of GPS positions P=(Plat, Plon) , lat and lon stand for longitude and latitude.

We designed a GPS driver that follows GPS course as given :

```
course = [[1,39.181917,-86.5221208333,1.5,3.0],\
           [2,39.1818975,-86.521724,1.5,3.0],\
           [3,39.182143,-86.5217033333,1.5,3.0],\
           [4,39.182199,-86.5220985,1.5,3.0],\
           [5,39.1819156667,-86.522309,1.5,3.0],\
           [6,39.1819645,-86.522398,1.5,3.0],\
           [7,39.1820415,-86.5223095,1.5,3.0],\
           [8,39.1821313333,-86.5223926667,1.5,3.0],\
          [9,39.1822116667,-86.522302,1.5,3.0]]
```

The course waypoints are defined by the course code above. Each line shows the sequence number, lattitude(deg), longitude(deg) , LOB radius in meters, speed (m/s).

Here are the information of components :

1- Sequence number : an integer >= 0, that unifies the waypoint. The sequence number must be unique. The waypoints with the same positional info will have different sequence numbers.
2- Latitude : range of [-90, 90] that represents the degrees of latitude.
3- Longitude : range of [-180,180] that represent the degrees of longitude.
4- Lateral Boundary Offset is the number represents the radius in meters of a circle centered at the given lat/lon.
5- Speed Limit : the number specifies the maximum speed in meters per second. The speed is only allowed between the current waypoint to the next.

In the previous assignment we had a two state control automaton. However, in this assignment we have only one state, which is trying to reach the target waypoint. We it reaches the target waypoint, it selects a new target iteratively.

The architecture of the GPS follower system is the same with square driver, but both the GPS component and the compass component in the square driver provide a heading value.

The important difference between GPS and square driver is GPS heading is only accurate the vehicle is moving, while the Compass is always accurate. So, if the GPS is wrong at the beginning, it is better to use it instead of the Compass.

In the GPS follower we needed to take into account the bearing and distance to target waypoint. Here are some problems in calculations :

1- Latitude and Longitude units are in degrees but we use meters for measuring distance
2- We converted the degrees to meters 1 degree = 111.122 meters. But this only for latitudes (between one latitude to the other each degree

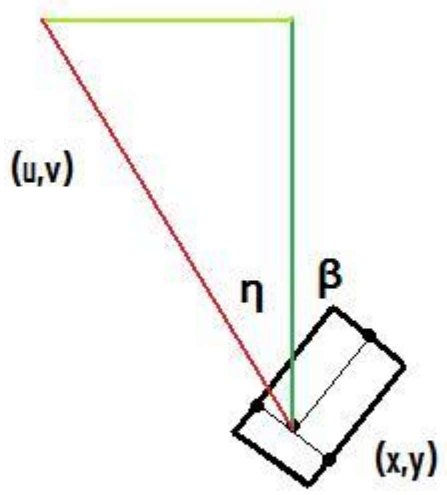
shows 111.122meters) However, for our course, due to the coordinates of the course each degree of longitude corresponds to ~86.358 meters.

Here are the formulas of Planar distance and bearing :

$$\sqrt{(u-x)^2+(v-y)^2}$$

$$\eta = \arctan\left(\frac{u-x}{v-y}\right)$$

Where you can see the parameters below :



When we ran the code on the visualizer, here are the visuals we got and the visuals from the field test.

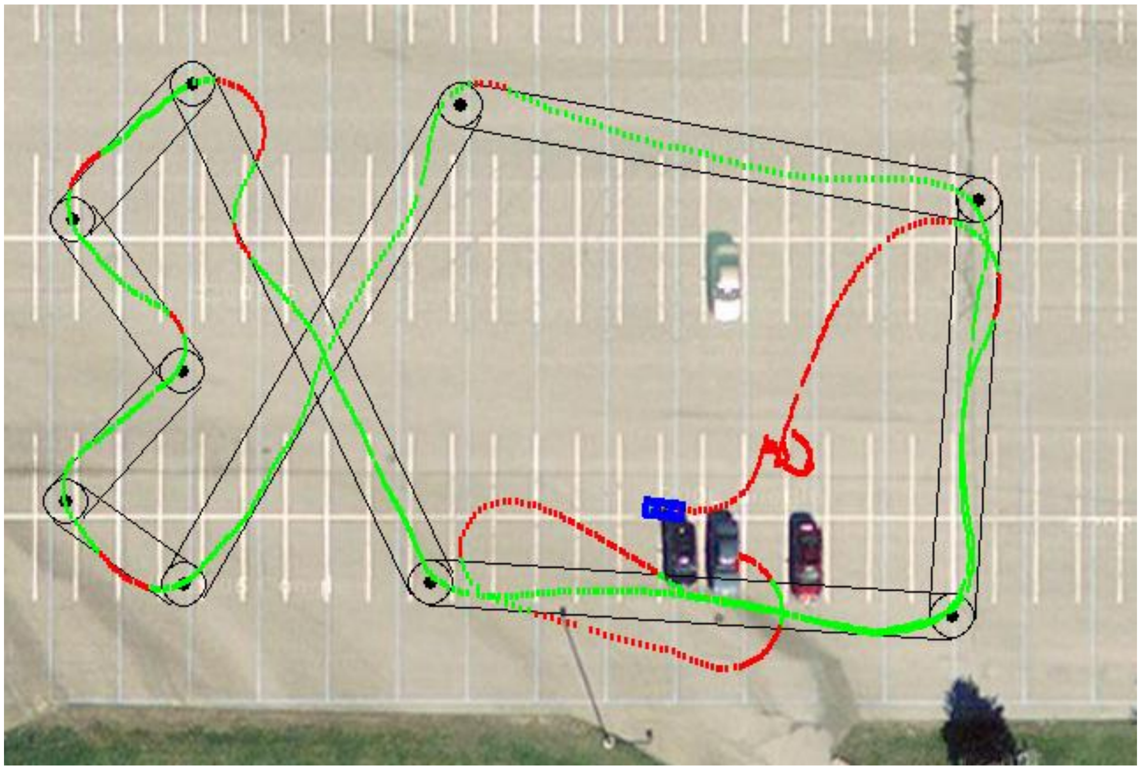


Figure 5.9

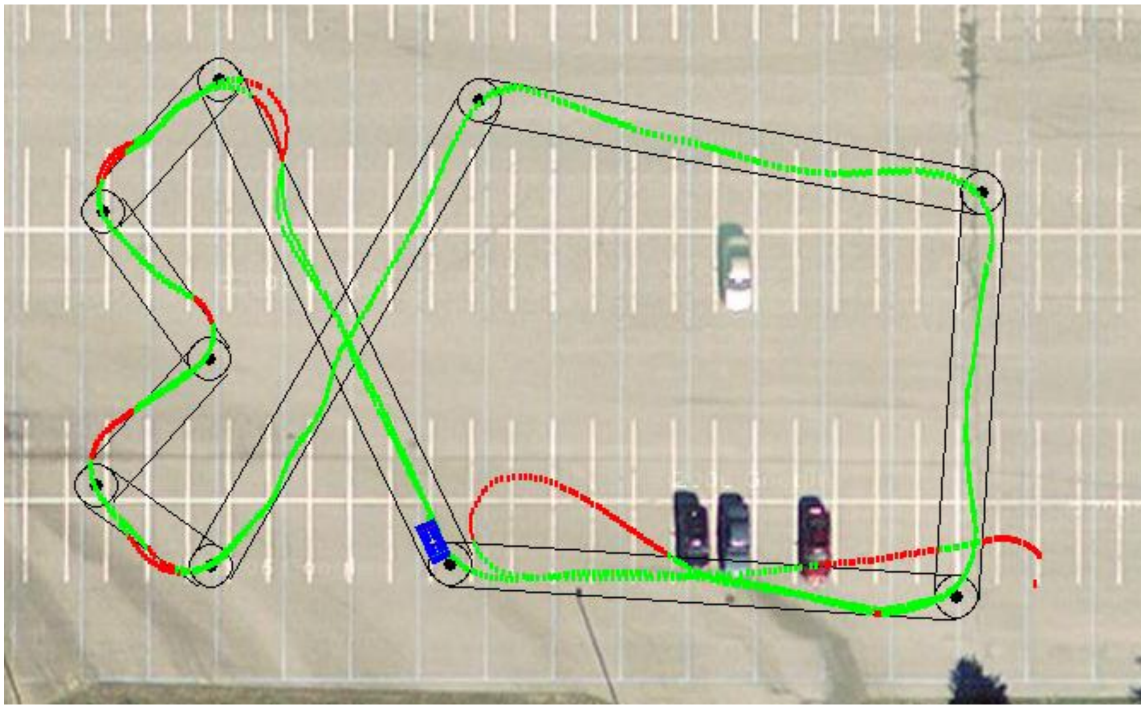


Figure 5.10

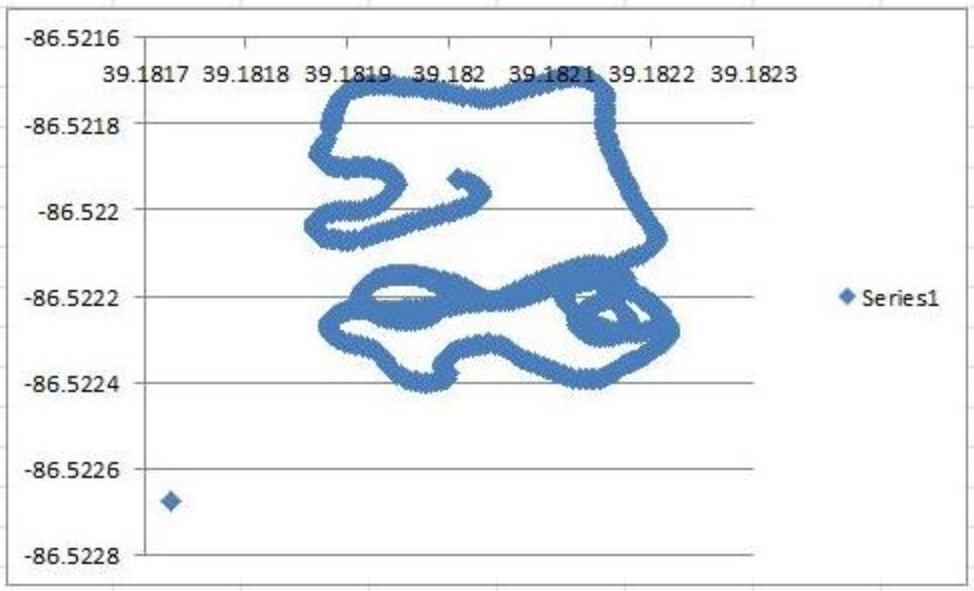


Figure 5.10 Excel Plot

As you see in the visuals, even we start from the wrong GPS, the cart is able to correct by using GPS heading. We have small steering off the road, but generally our cart successfully follows the GPS coordinates.

Lab Report 3: Path Planning (With the Final Project Strategy Tactical Maneuvers)

In the first lab, we designed a heading controller that adjusts the turn radius of the steering on each clock cycle to minimize the difference between the current compass heading and the desired heading. In the second lab we, needed to use the same controller to follow a course that was defined by the list of points of latitude and longitude.

In this lab, we refined things further by giving ERTS the ability to plan its path in a better way than just using GPS. By adding "synthetic" waypoints as ERTS travels the track and a "leaning point" before and after the actual waypoints, it is better guided around the corners by forcing it to lean toward the edge of the corridor and make a much wider turn. The synthetic points were defined as follows:

```
x1=(self.course[(self.waypoint+1)%len(self.course)][1]+7*(self.course[(
self.waypoint)%len(self.course)][1]))/8
y1=(self.course[(self.waypoint+1)%len(self.course)][2]+7*(self.course[(
self.waypoint)%len(self.course)][2]))/8

x2=(self.course[(self.waypoint+1)%len(self.course)][1]+3*(self.course[(
self.waypoint)%len(self.course)][1]))/4
```

```
y2=(self.course[(self.waypoint+1)%len(self.course)][2]+3*(self.course[(
self.waypoint)%len(self.course)][2]))/4

x3=(self.course[(self.waypoint)%len(self.course)][1]+(self.course[(self
.waypoint+1)%len(self.course)][1]))/2
y3=(self.course[(self.waypoint)%len(self.course)][2]+(self.course[(self
.waypoint+1)%len(self.course)][2]))/2
```

Self.course is the array containing the track information including latitude and longitude of waypoints, waypoint radius, and maximum speed. Self.waypoint is the most recent waypoint passed. Len(self.course) is the length of the corridor that ERTS is in at the time calculated using the distance formula from the coordinates from the array in self.course.

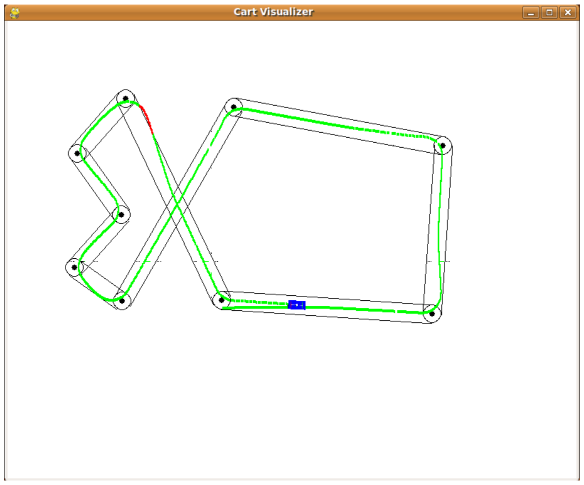


Figure 5.11

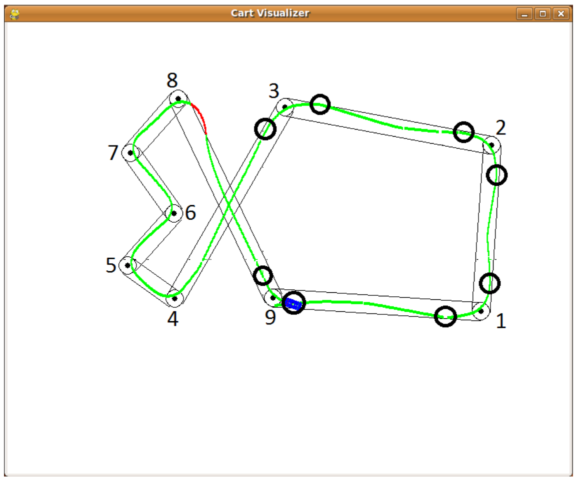


Figure 5.12

Doga Tuncay
Kurt Weisman
Hasan Kurban

Starting from ERTS starting point on the course and following that, set 1 corresponds to turn 1, set 2 to turn 2, and set 3 to turn 3. A fourth set of synthetic points (x4 and y4) are placed at turn 9 (the final turn) using the same formula as x1 and y1. Turns 1, 2, 3, and 9 were the only turns used due to these being longer corridors with a length greater than 20 meters. It would looks as if turn 8 would need a synthetic point as well, since the corridor between turns 8 and 9 is greater than 20 meters long, but after several tries, we determined the angle of turn 8 is too acute and the turn would not be sharp enough and would send ERTS too far off track.

Lab Report 4: Obstacle Avoidance

So far we have navigated the test track with GPS and with a path planning algorithm. Finally, in this lab we used ERTS' front-mounted laser range finder to avoid obstacles placed around the test track.

For the simulations, a pre-built array of obstacles containing 9 more arrays containing the name of the cone, the GPS coordinates for it, and the size of the obstacle. The obstacle_list array is fed to the synlaser_s class so that it may know where the obstacles are. A variable called obs_found is intiaialised as false and the integer obs_count is initialized to 0, but will switch to "turn_m" when an obstacle is found and will increase the count according to how many are seen.

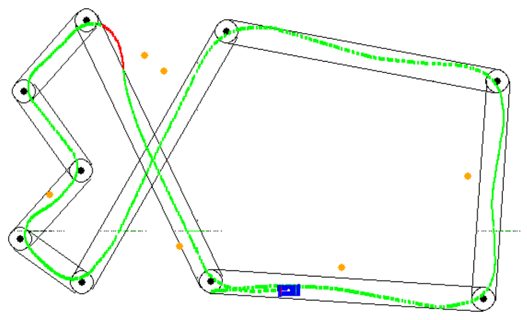


Figure 5.13

When an obstacle is spotted, first the angle and distance to the obstacle must be computed. This is achieved by using elements from another class called geopy. Geopy's gps_s is used to calculate the distance to the obstacle by taking the GPS coordinates of both the obstacle and ERTS and comparing them using the distance formula. The difference of angle is calculated in two parts: angle1 and angle2. Angle2 uses geopy's distance function to find the forward azimuth of the obstacle. Angle1 is found in a similar way, but used the forward azimuth of the next way. Angle2 is subtracted from angle1 to find the angle to the obstacle. Using all of these components, the distance is calculated. If the distance is calculated to less than or equal to .00009 and the absolute value of the difference of angles is less than or equal to 15, an obstacle is counted as detected obs_found switches from false to turn_m.

Next the best way to avoid the obstacle must be calculated. The same method of path planning using synthetic points (refer to lab 3) is used except with the added use of the compass. The current heading is calculated using the forward azimuth of ERTS and the heading of the next waypoint is calculated the same. Depending on the compass headings, the fourth synthetic point moves causing the cart to veer to the left or right. If the obstacle is no longer seen, it continues on as normal. If the obstacle is still in sight, the synthetic points will continue to shift.

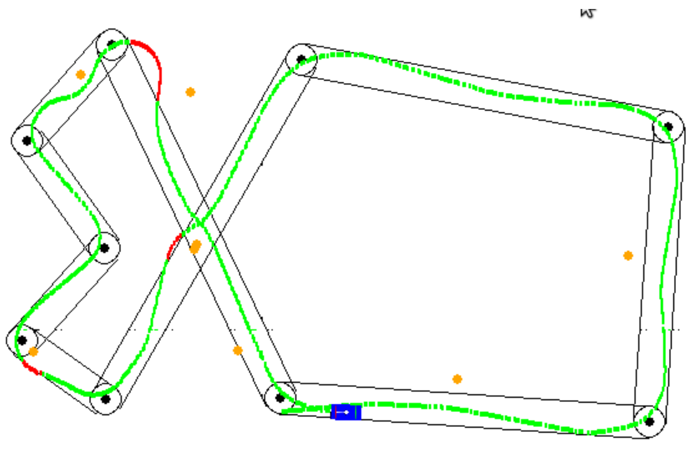


Figure 5.14 Updated Obstacle List

Just to be sure that everything worked as it should, the simulator was run again, but this time with a slightly altered list of obstacles. Everything performed as hoped.

Speed Control

In experimenting with the speed control aspect of this project, we initially started with using throttle and braking values that changed according the distance from the next major waypoint. By changing the values, we had assumed this would minimize time to travel around the test course.

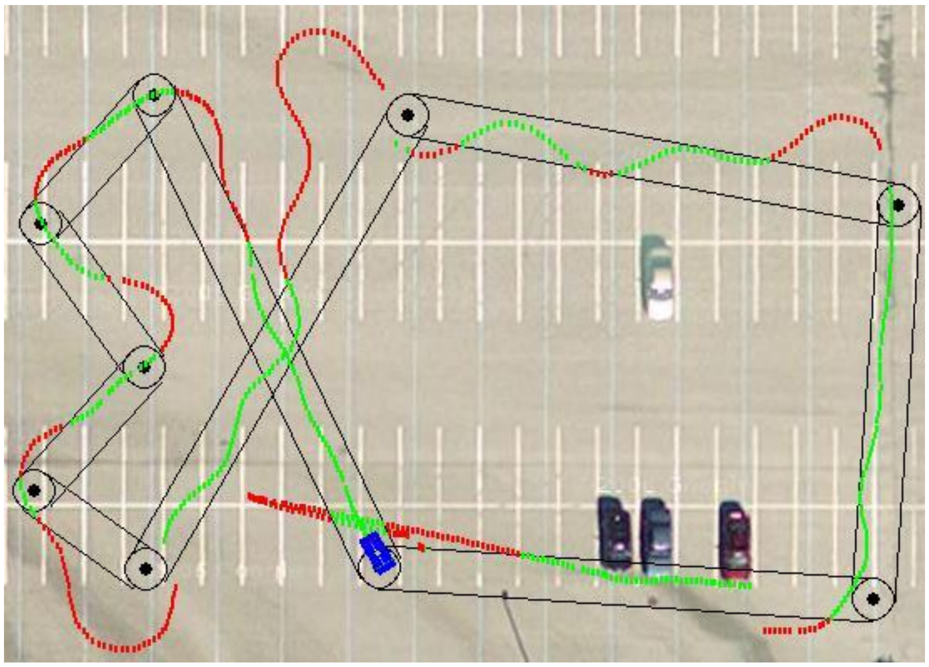


Figure 5.15 Throttle %70

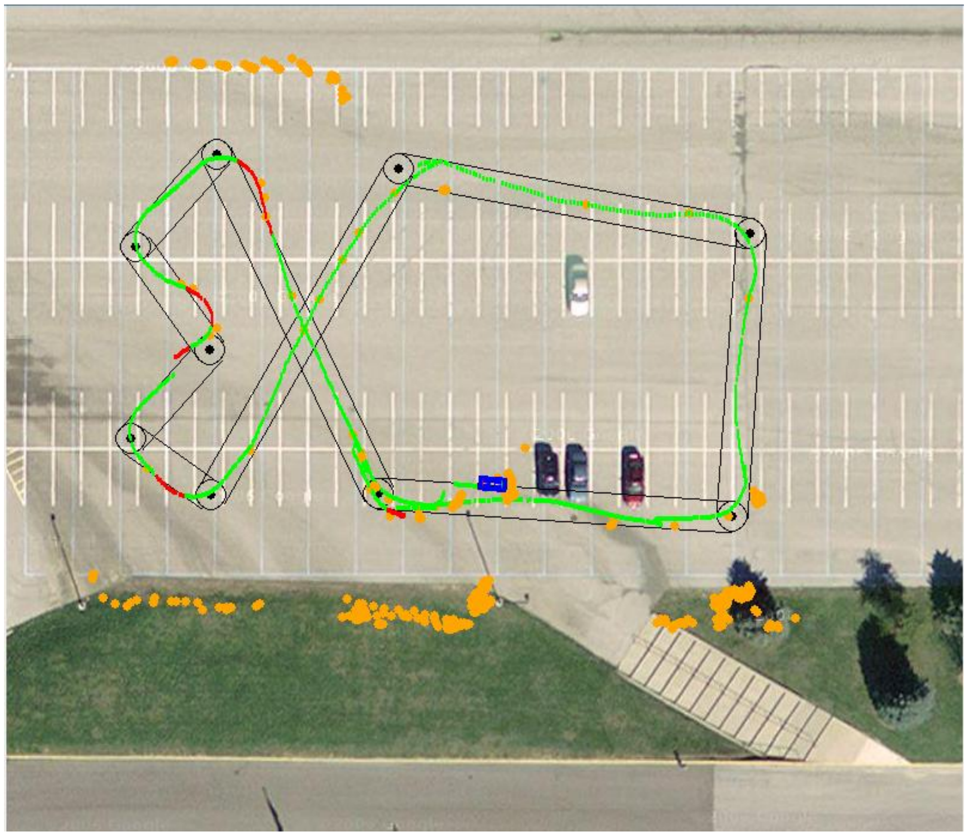
One thing that was brought to attention after the fact was that in straightaways, the cart would slow greatly. This has since been fixed, but not yet tested on the track in person. Besides the ERTS vehicle going off track due to going into turns too fast and not braking enough, over-steering was also evident in the test. This can be seen in the extreme weaving illustrated in the test image above. The latest version of the speed control code is below.

```
if(self.rounds == 4):
      self.jdriver_s['percent_throttle'] = 0.0
      self.jdriver_s['percent_braking'] = 100.0
elif distance_left < 30:
      self.jdriver_s['percent_throttle'] = 45.0
      self.jdriver_s['percent_braking'] = 55.0
elif distance_left < 20:
      self.jdriver_s['percent_throttle'] = 35.0
      self.jdriver_s['percent_braking'] = 65.0
elif distance_left < 10:
      self.jdriver_s['percent_throttle'] = 20.0
      self.jdriver_s['percent_braking'] = 70.0
elif distance_left < 5:
      self.jdriver_s['percent_throttle'] = 10.0
      self.jdriver_s['percent_braking'] = 80.0
```

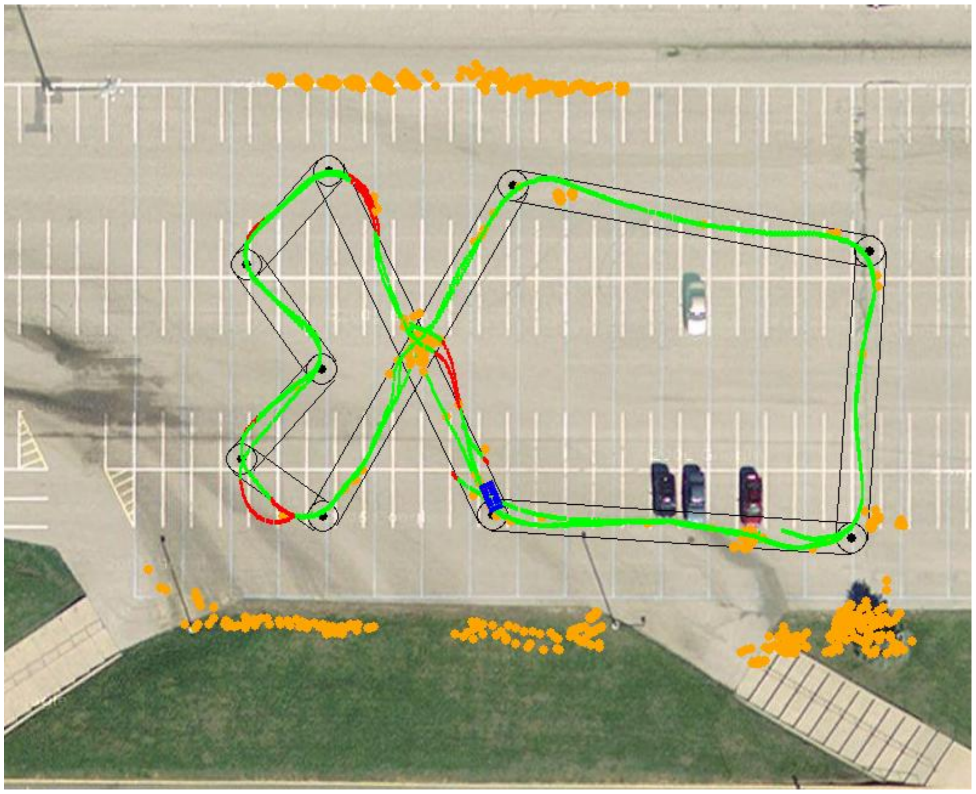
```
        else:
                self.jdriver_s['percent_throttle'] = 100.0
                .jdriver_s['percent_braking'] = 0.0
```

Lab Report 5: Final Run and Results

In the end, we were able to field test the code for the final project. Just to make sure everything worked correctly and the track was still safe for testing (it had begun to snow shortly before and there was a bit of water slicked across the southwest end) as well to drop off more obstacles, several test laps were taken. This was recorded on the visualizer as shown below. The visualizer did crash partway through, but was still able to capture most of the activity. There were a couple at GPS hiccups right before turn 6 and another just past turn 9. This seemed to be a somewhat common problem.

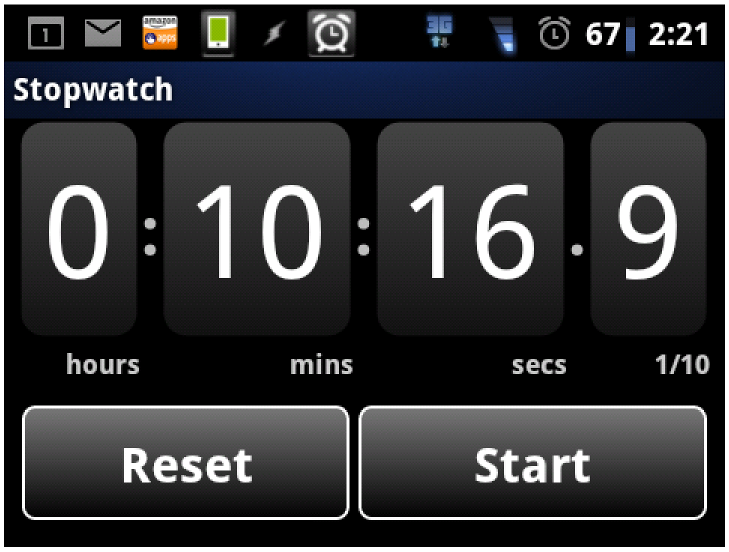


Once the actual test began, things went relatively smoothly. All visuals can be seen below. The shift in GPS can be seen in the shift of location in the cones at the north end of the track. More GPS shifts can be noticed: one before turn one and another in on turn 4 that caused the arc seen in the southwest part of the track. One final one is right before turn 9 where it seems to jump from the outside of the track to the other.



Despite these inconsistencies, the first four corridors were almost identical in their paths, even with obstacles. Strangely enough, even though the obstacle was set in the middle of the crossing of the two corridors, there seemed to be more problems between turns 8 and 9 than 3 and 4.

Speed seemed to be relatively consistent, but if we pursue the project further, it should probably accelerate out of turns a bit later than it does.



Doga Tuncay
Kurt Weisman
Hasan Kurban
In the end, the cart was able to complete a 3-lap run in approximately 10:16.9 (though the time has claimed to have started the clock 6-7 seconds early). Given the code and weather conditions, we feel this was an adequate final run of the course, despite the 3 dead cones that lie in the wake.

<u>CONCLUSION</u>:

We explained our approaches, design and testing results of the lab assignments and the final project for the class of P545 Embedded and Real Time Systems in this report. In the introduction about we show the architecture of the ERTS. In the second part, which is requirements, we give some information about the objectives for each lab assignments and our final project. In the design part we presented our approaches for each lab and improved them by adding our Final Project Ideas (such as Tactical Maneuvers, Speed Control etc.) In order to make our ideas more clear, we gave some geometrical calculations that are also shown by several figures and we compare the strategies and reasoned why we used them. In the testing part of this document we show some screenshots that we got from the test field and the simulator by using our lab reports.

We would like to thank you Prof. Johnson for your help and guidance during the project.