Final Report

Shenshen Han
P545 School of Informatics and Computing
Indiana University Bloomington
Dec 2011

1) Introduction

In this report, an automatic driving controller is designed and implemented for ERTS vehicle in order to follow GPS waypoints, drive inside corridor boundaries, and avoid synthesized obstacles.

2) System Requirements

As the lab manual stated for trial run:

1. Three laps must be completed, after which the vehicle rolls to a stop.
2. Missing a waypoint is penalized.
3. Traveling outside the course boundary is penalized.
4. Smooth steering is rewarded.
5. Faster lap times are rewarded.
6. Smooth and appropriate speed control is rewarded.

In other words, our driving controller must be able to:

1. Run the pre-defined course in multiple laps
2. Enter the LOB of every waypoint
3. Drive inside the corridors
4. Reduce the magnitude of over-steering and under-steering
5. Reach maximum possible speed
6. Reduce sudden dramatic changes of throttle and break

3) Design and Implementation

3.1) Steering Correction

The basic controller implementation of ERTS is given by our course in the file of “squre_sensor.py”.

The “process()” function is called every control-loop at rate of 10 Hz (sometimes 9 Hz), so the steering error-correction is computed each loop and set to the “jdriver”. Similarly, other computation and controls are also carried out in each loop.

3.2) Waypoints List and LOB(Lateral Boundary Offset)

We keep the course as a waypoint list. In every control-loop, the vehicle is made steering to the first waypoint of the list. When the vehicle arrives a waypoint, we pop first the first waypoint and append it to the end of the list, in this way we can achieve re-lapping.

As the waypoint is given as a pair of specific coordinates, they are precise enough to millimeter, it is then not sensible to say that the vehicle only arrive the waypoint when it is on top of the specific coordinates. Instead, we say that a vehicle arrives a waypoint as soon as the vehicle entered the LOB of the waypoint. In figure 1, we say that the two cars inside the LOB arrive this waypoint, the car outside does not arrive this waypoint yet. This could help choosing a better turning point within the LOB later, as the center of LOB is not always necessary a good point for making the turn for next waypoint.

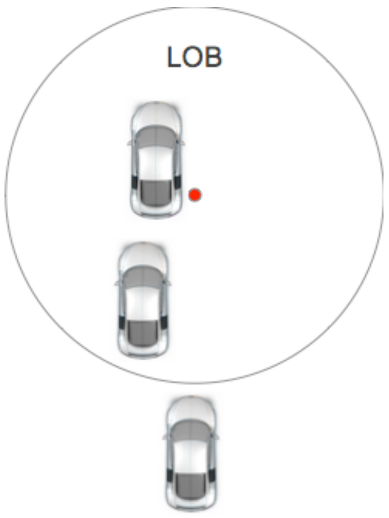


figure 1: the vehicle entered the LOB means that it arrived the waypoint

Therefore, we pop the first item of the waypoint list as soon as we detect that the vehicle (actually it is the GPS device) is within the LOB area, the pseudo code is like:

```
heading_point = waypoint_list.pop();
for control-loop {
        current_point = gps.getCurrentCoordinates();
        distance_to_heading_point = distance(current_point, heading_point);
        if (distance_to_heading_point <= LOB) {
                heading_point = waypoint_list.pop();
        }
        calculate and get turn_radius_inverse;
        set turn_radius_inverse to jdriver component;
}
```

3.3) Coping the Responsiveness of Control

From a field testing result of Lab 2 (see figure 2), we see that under a constant steering sensitivity, the vehicle over-steers when reach a high speed, but it steer quite stably when reach a low speed. Moreover, we see that the control on vehicle is less responsive at higher speed. For example, assume within a 10 meters distance, there are 10 control-loops carried out at a high speed, in contrast, there might be 20 control-loops carried out at a low speed. If the 10 meters distance is "critical", we would like to be able to make 20 decisions other than 10 decisions. However, we are required to reach high speed as much as possible, so there is a trade-off between speed (less travel time) and control responsiveness, thus we decide to category the path into two types of path components, "Turning Area" and "Straight Area". The Turning Area requires low speed and higher steering sensitivity, as it is more critical. The Straight Area can have high speed and lower steering sensitivity. The Turning Area is defined as the LOB for now. Rest of the areas are defined as Straight Area.

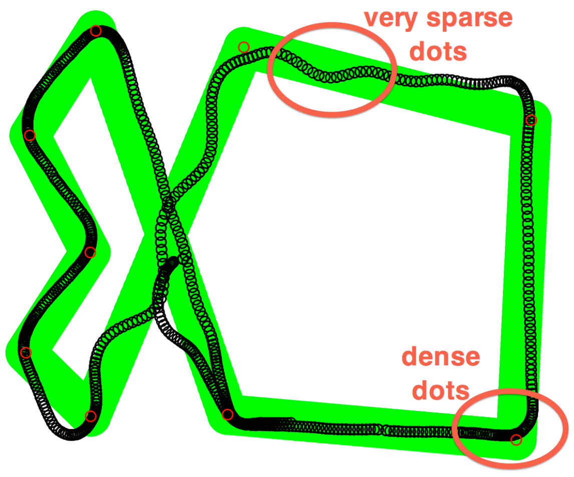


figure 2: one lap with high speed and a static steering sensitivity

3.4) Adding State Control

As section 3.3 discussed, we are defining Turning Area for controlling and adjusting vehicle's speed and steering sensitivity. However, according to a field testing result of Lab 3, the vehicle cannot start turning in time for several waypoints (see figure 3) even though we pop the first waypoint as soon as the vehicle enter the LOB.

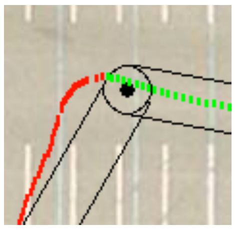


figure 3: the vehicle is still too fast to make the turn in time

Therefore, only use LOB itself to define the Turning Area is not enough, as we want to keep a high speed as possible and have a good turn as the same time. So we define the Turning Area into three different states (figure 4): Approaching state, Turning state and LOB state.

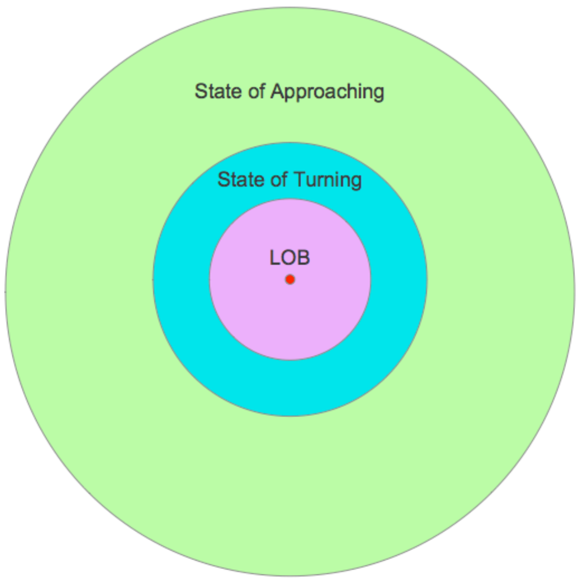


figure 4: the three states of Turning Area

Approaching state means the vehicle should start to reduce its speed, normally it should use 0% throttle, in another word, the vehicle is sliding. That is what people normally do when they drive a car and approach an intersection, as suddenly letting the throttle off will not reduce the speed dramatically, and people can prepare to hit the break also save the gas at the same time. In this state, it should keep a low steering sensitivity (say 0.0037) in order to maximize the steering stability under a high speed.

Turning state means the vehicle should start to make the turn to the next waypoint. It will be a slightly larger area than the actual LOB, as LOB is a boundary limitation and it will not be an uniformed value if we want to describe different size of corners. Therefore, considering the size of the vehicle itself, speed inertia and the control delay, vehicle should always start to make the turn just before entering the LOB, and we define that:

radius of turning state area = (1 or 2) * length of vehicle + radius of LOB

Say if our vehicle is 1.5 meters long and the LOB radius is 3 meters, then the radius of turning state area is 2*1.5+3=6 meters. From figure 5, we should see that green car is turning better than the red car in the case of they have the same corner entering point, because the green car travels less distance for the turning. The difference between the two cars is that, at the "corner entering point", green car's front wheel is already turned to right for some degrees, but red car's front wheel is still straight.

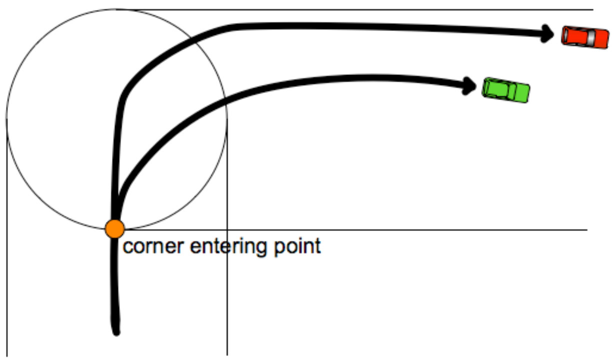


figure 5: one car makes turn earlier than the other

And in the turning state, we woud like to maximize the steering sensitivity in order to make the vehicle to turn as “quickly” as possible, as we are only setting the “turn_radius_inverse” at each control-loop.

LOB states means that the vehicle has completed this waypoint, and from now on it should head to the next waypoint. However, if we are required to touch the actual center point of the LOB then this mechanism is not suitable. But we thought it is not sensible to be made touching the center point of the LOB for making a turn if we are required to reach high speed and travel less distance.

3.5) Adding New Waypoint for Better Corner Entering
According to the normal “Racing” theory, a large turn is better than small turn, as the it can help a vehicle to keep higher speed, and a larger turn is easier to make as it needs to overcome less Torque (moment of force). Thus, we will dynamically put new additional way points for the vehicle to head to in order to make a larger and better turn. In addition, this would serve as an implementation base of obstacle avoidance.

In implementation (see figure 6), we make a straight line between two consecutive waypoints, and reversely extend that line according to the actual travelling direction, then we choose a most appropriate longest point (cannot be the edge as it may cause vehicle to go out boundary) on the extended line as the actual additional waypoint for the vehicle to follow. The new coordinates are calculated by the ratio between distance and GPS coordinates. In figure 6, the red dot is the center of the new additional waypoint, the yellow do is the center of the waypoint P2, and the green dot is where the vehicle start to turn for P2. Possibly, after the green dot the vehicle will turn to waypoint P3 very soon. And this path is better for turning than the path of always traveling in the center of the corridor.

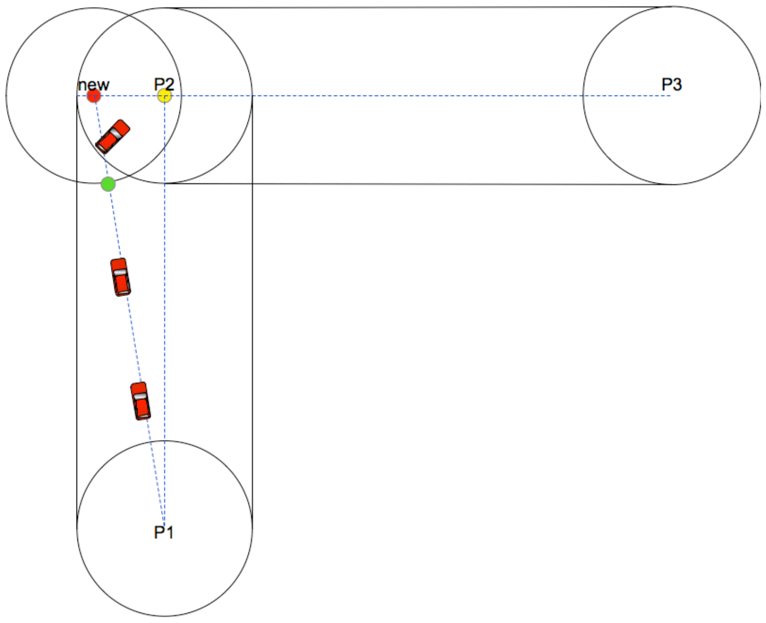


figure 6: adding a new waypoint for better turn

This is a simple approach for determining the new waypoint for better turn, it requires very small computation. However, the actual coordinates of the new waypoint are not so critical, as long as we make the vehicle move to the left-side of the corridor before we need to make a right turn (or vise versa). The critical factor is the LOB of the new waypoint, it will affect that when we make the actual turn for the original waypoint, but we are not considering this factor much in this report, so we use the same LOB for new additional waypoints as the original waypoints.

3.6) Adding New Waypoint for Avoiding Obstacle

Out principle for “adding new waypoint” is to add less waypoints as possible. More waypoints may cause the collisions between different waypoints that are close to each other. Also additional waypoints may collide with obstacles. Those may result that a lot of additional waypoints are actually ignored after added due to more complicated situation. In the example of figure 7, for such a path we actually only need 3 waypoint to describe the shape of the path, such as figure 8. Therefore, we use One additional waypoint for One obstacle.

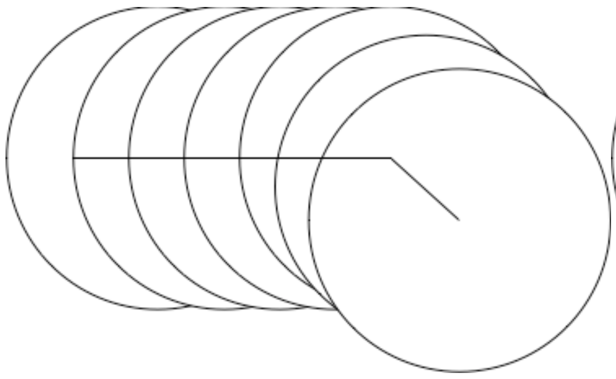


figure 7: too many waypoints

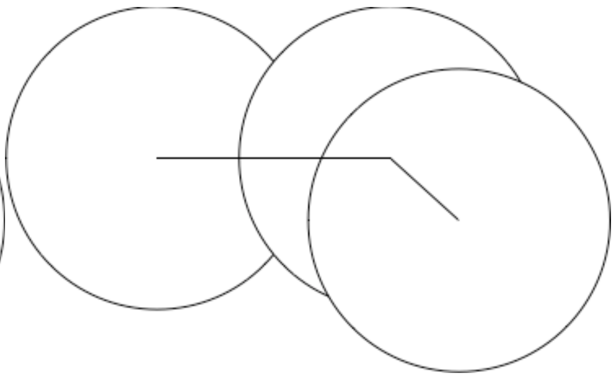


figure 8: too many waypoints

We mainly use the “destination()” function from “geopy”. Given a point as starting point, a bearing and a distance, the destination function will compute and return a point as the destination. This is very useful for calculating the latitude and longitude of additional waypoints. Although this function is also desired in Lab 3, we used a simple mechanism for determining a new additional waypoint for better corner turning in Lab 3.

In order to avoid an obstacle, we put an additional waypoint besides the obstacle for a certain range in order to allow vehicle to bypass the obstacle. So it is important to decide where the best additional waypoint is, but we have no resource to prove the best waypoint in this report.

We will make a perpendicular line to the path line made between the starting and end waypoints, then chose an appropriate length (maybe quater or third of the corridor width with considering the sizes of the vehicle and the obstacle) on the perpendicular line in order to determine the new avoidance waypoint.

For example (figure 9), considering we have a path from P1 to P2, an obstacle P3, and we want to add and locate a new waypoint P4.

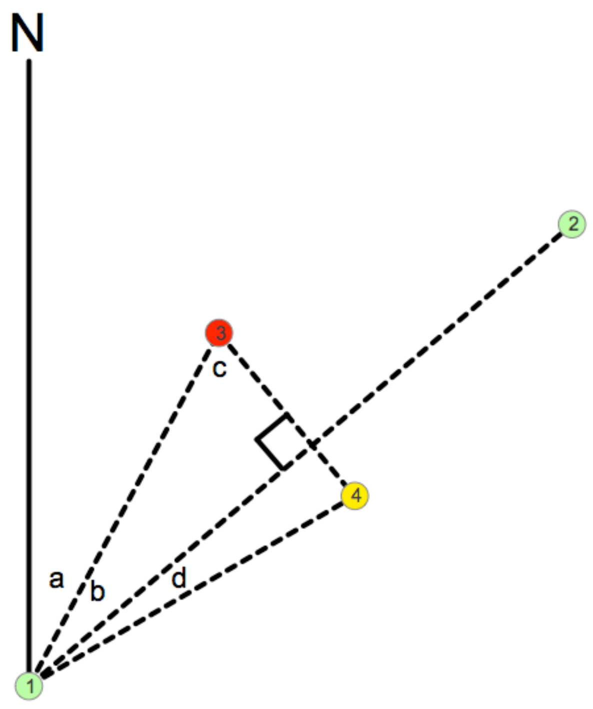


figure 9: calculating the avoidance point

From P1 and P2, we can get the angle(a+b). From P1 and P3, we can get the angle(a). Then, we know angle(b) = (a+b) - (a). Then we know angle(c) = 90 - (b). As we also can get the length of (P1_P3) and (P3_P4), therefore, we can get the length of (P1_P4) according to the “law of cosines”. Thus we can get the angle (b+d), and then we know the bearing of P4 from P1 is (a+b+d), and the distance from P1 to P4 is (P1_P4), therefore we can compute and get the coordinates of P4.

In order to decide that the vehicle circumvents an obstacle from the right-hand side or left-hand side, we calculate the different between the bearings that is from the starting point to the obstacle point and the end point. For example:
a = bearing(P1_P2) - bearing(P1_P3)
if a >=0, then circumvent by right-hand side
if a<=0, then circumvent by left-hand side
If the obstacle is at the actual center of the width of the corridor, we randomly choose a side to circumvent.

For more clear computation procedure, we also define the bearing into 4 sections (figure 10). It can help to determine that if an obstacle is behind the vehicle. Also it can help the above “avoidance point” calculation.

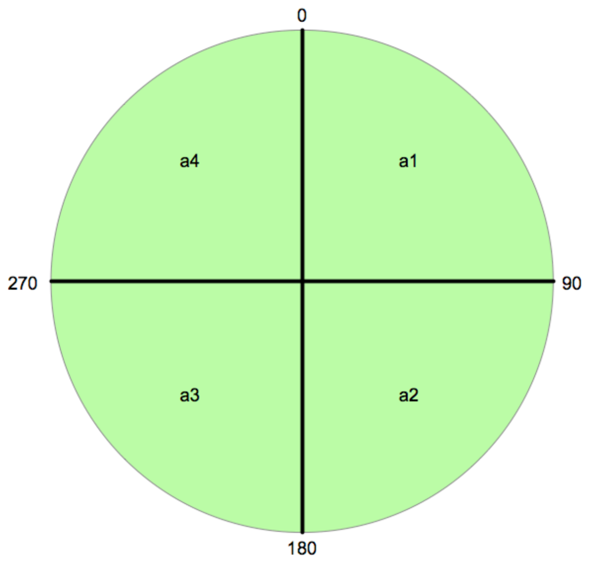


figure 10: bearing sections

In order to reduce unnecessary computation and consideration, we ignore two types of the obstacles that are given by the “obstacle detection laser”.

One type is the obstacle that is outside the corridor segment, we use the “sine rules” to calculate that if an obstacle is outside the corridor (figure 11). The corridor segment is determined by the current heading waypoint and passed waypoint. Assume P4 (yellow point) is the obstacle, we calculate the length of (P4_P3), if it is larger (figure 12) than the radius of LOB, then we say the obstacle is outside the corridor.

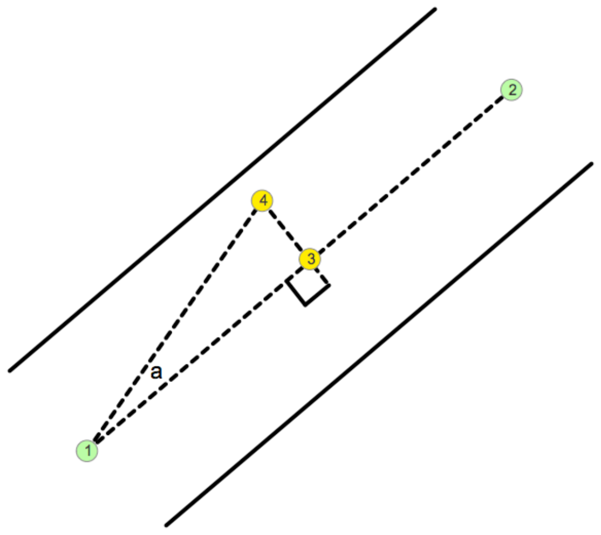


figure 11: calculating if an obstacle is within corridor

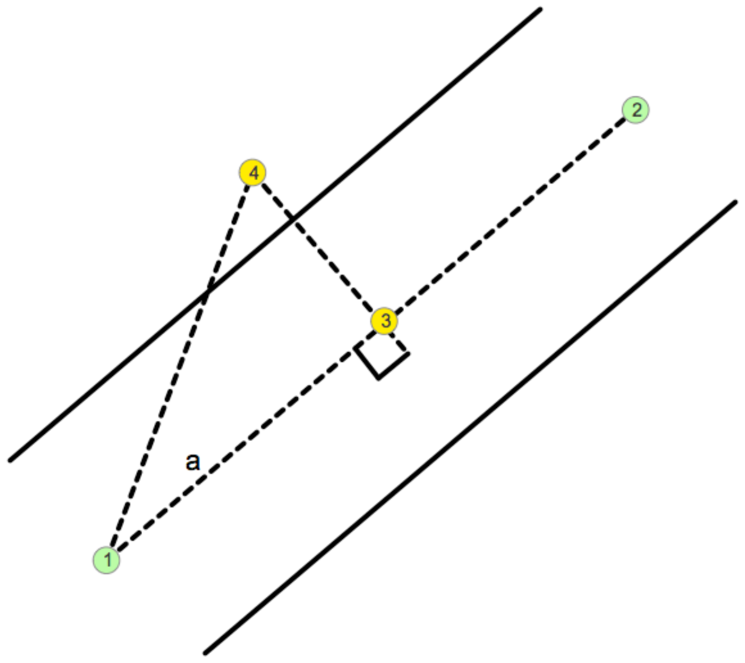


figure 12: an obstacle outside corridor

Another type is the obstacle that is outside the heading range of the vehicle, we say that an obstacle is already bypassed if the obstacle is at the range between 3 clock and 9 clock of the bearing of the vehicle. In figure 13, the red cones are considered as necessary obstacles to be avoid, the green cones are considered as be already bypassed the vehicle.

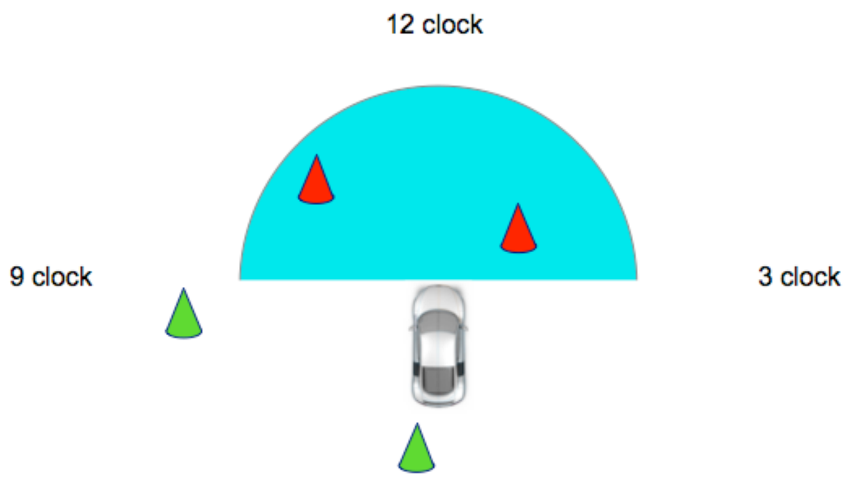


figure 13: determining if an obstacle is passed

Finally, the entire couse with original waypoints might be like figure 14.

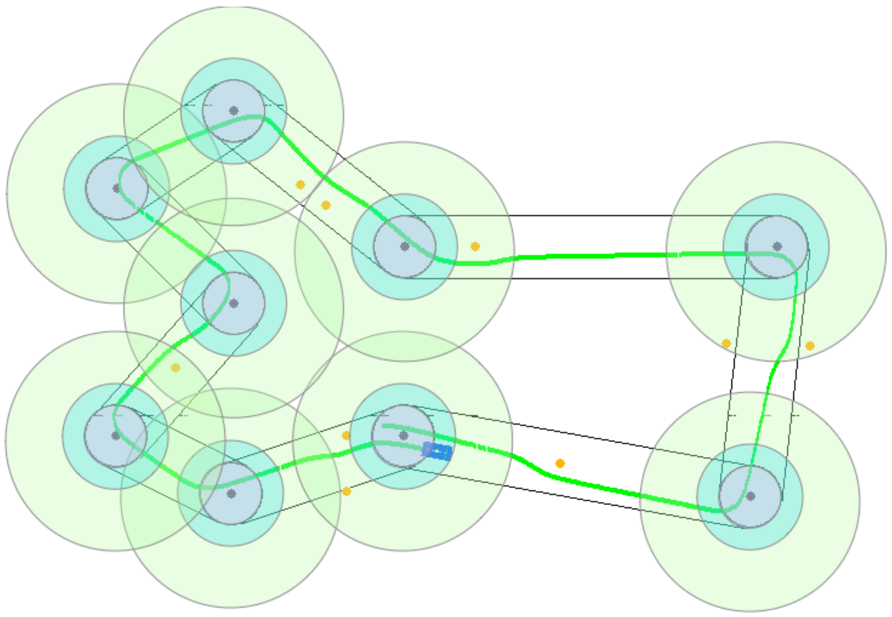


figure 14: original waypoints

The entire couse with additional waypoints might be like figure 15.

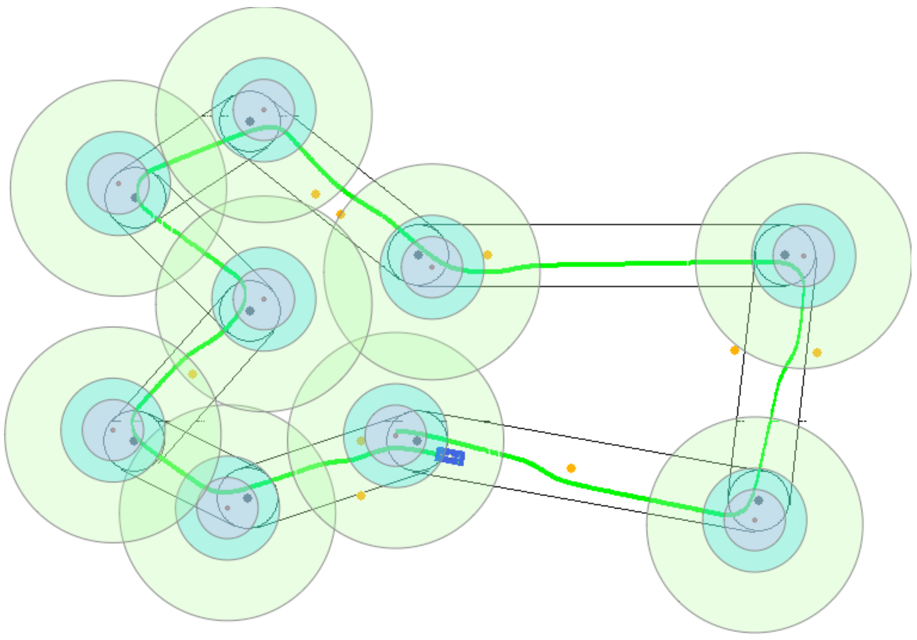


figure 15: additional waypoints for better turn

The entire couse with additional avoidance waypoints might be like figure 16.

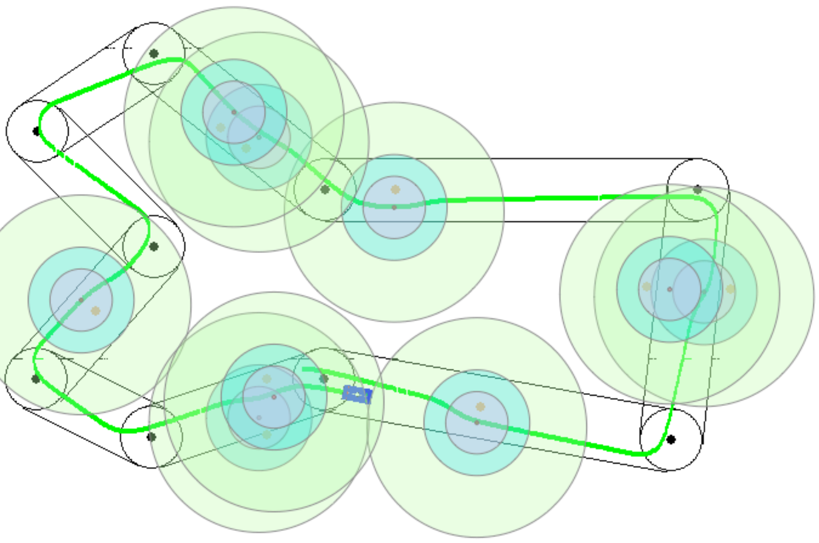


figure 16: additional waypoints for obstacle avoidance

4) Field Testing Results

4.1) Figure 17. The vehicle travels with a constant steering sensitivity of 0.007. It cannot finish several turns in time.

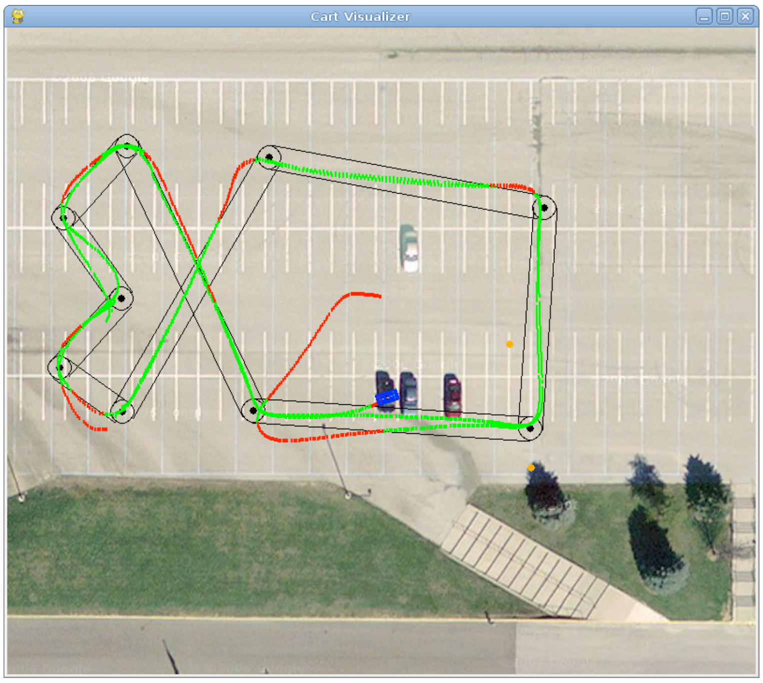


figure 17: a testing result from Lab 2

4.2) Figure 18. The vehicle travels with a constant steering sensitivity of 0.011. It turns better than when using sensitivity of 0.007. The vehicle travels too fast for the fourth waypoint. And it cannot make a good turn for the last (ninth) waypoint.

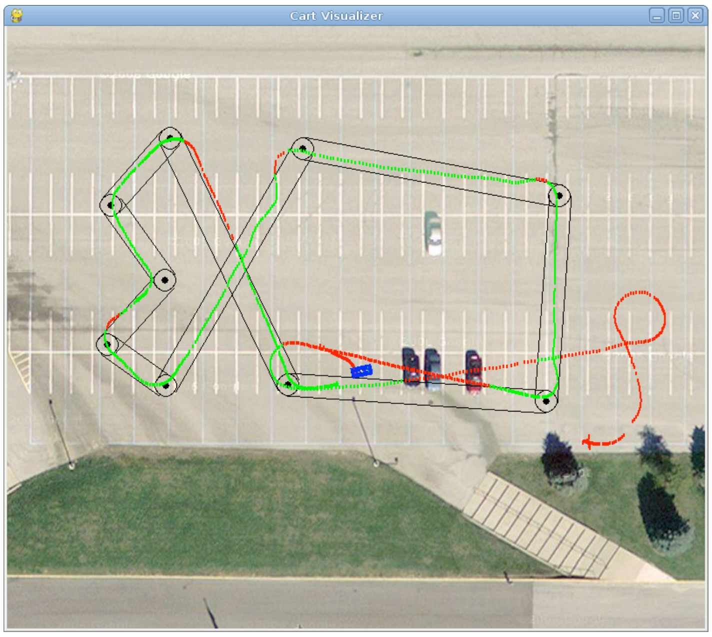


figure 18: a testing result from Lab 2

4.3) Figure 19. The vehicle travels with dynamic steering sensitivity and speed by the state control. The vehicle turns well for the last (ninth) waypoint, although we have encountered the “GPS jump” problem.

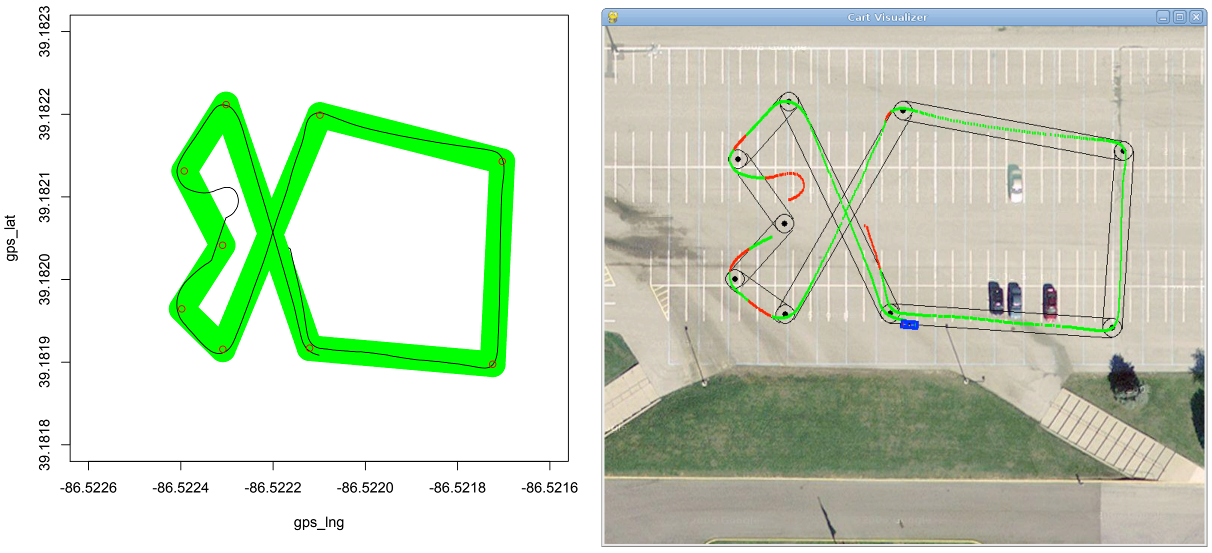


figure 19: a testing result from Lab 3

4.4) Figure 20. The travelled distance between each control-loop is captured in meters. It shows that sometimes we have to have wait for 0.3 meters to make a control decision.

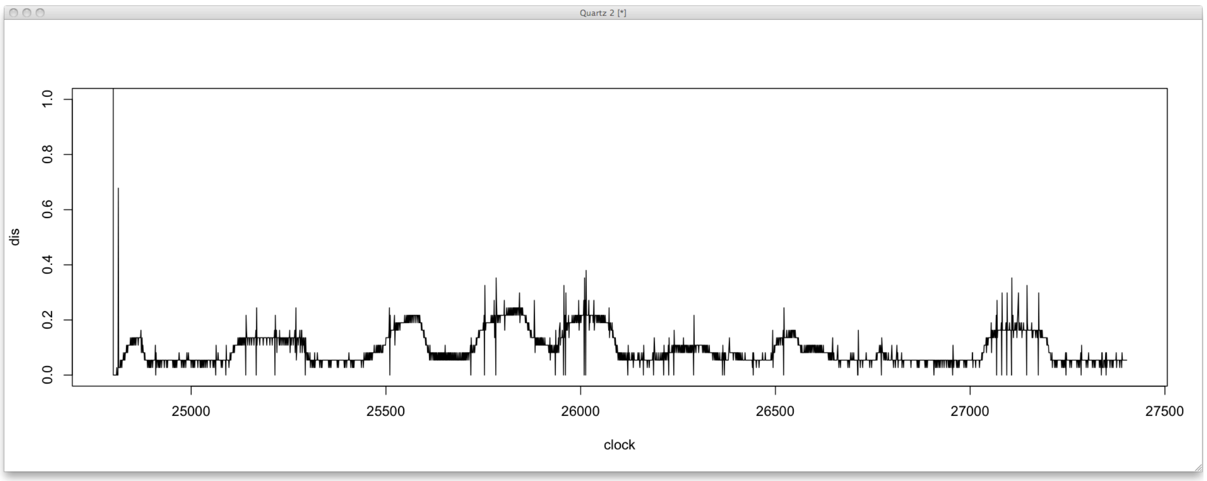
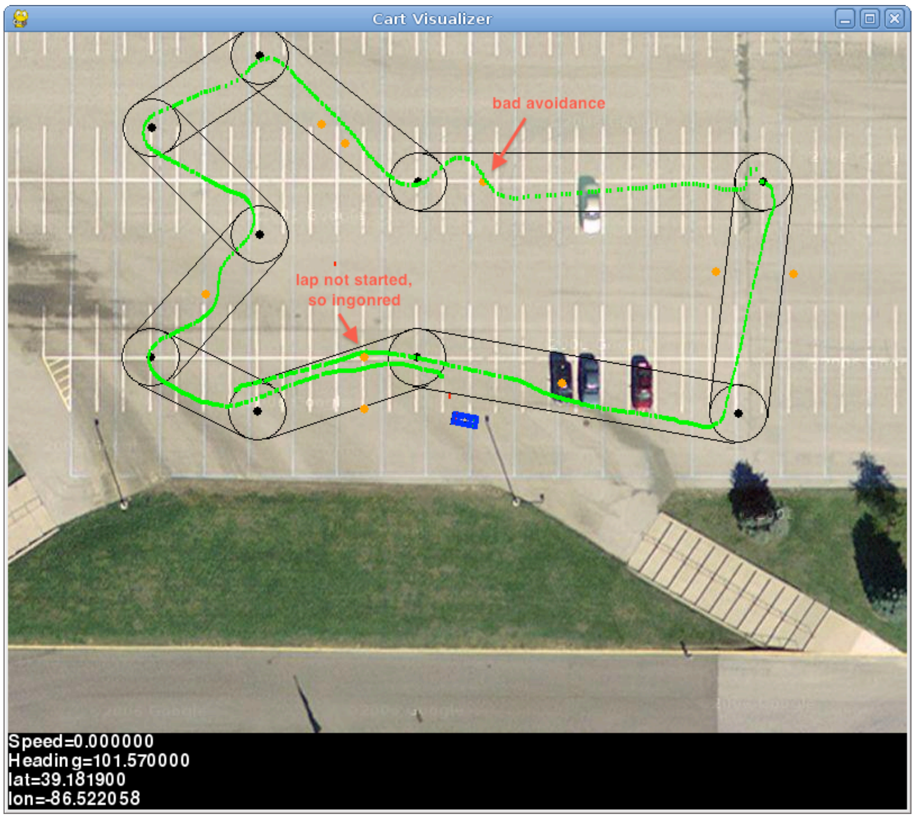


figure 20: a sampling of distance between cycles

4.5) Figure 21. First testing result of synthesized obstacle avoidance. In this testing, most obstacles are avoided, except one. The reason is that we still travel too fast when we approach the obstacle, so the avoiding responsiveness is low. And we set the “warning range” of detecting obstacle too short. Those problems can be easily resolved. We will set “warning range” from 6 meters to 10 meters or even longer.



```
Speed=0.000000
Heading=101.570000
lat=39.181900
lon=-86.522058
```

figure 21: a testing result from Lab 4

4.6) Figure 22. Second testing result of synthesized obstacle avoidance. In this testing, we have a successful avoidance for the obstacle we missed last testing. However, as the additional “turning” waypoints we added are a little bit far to the original waypoints, as the vehicle did a circle at the fourth waypoint. This problem can be easily solved. We will reduce the distance of turning waypoints from 2 meters to 1 meter.

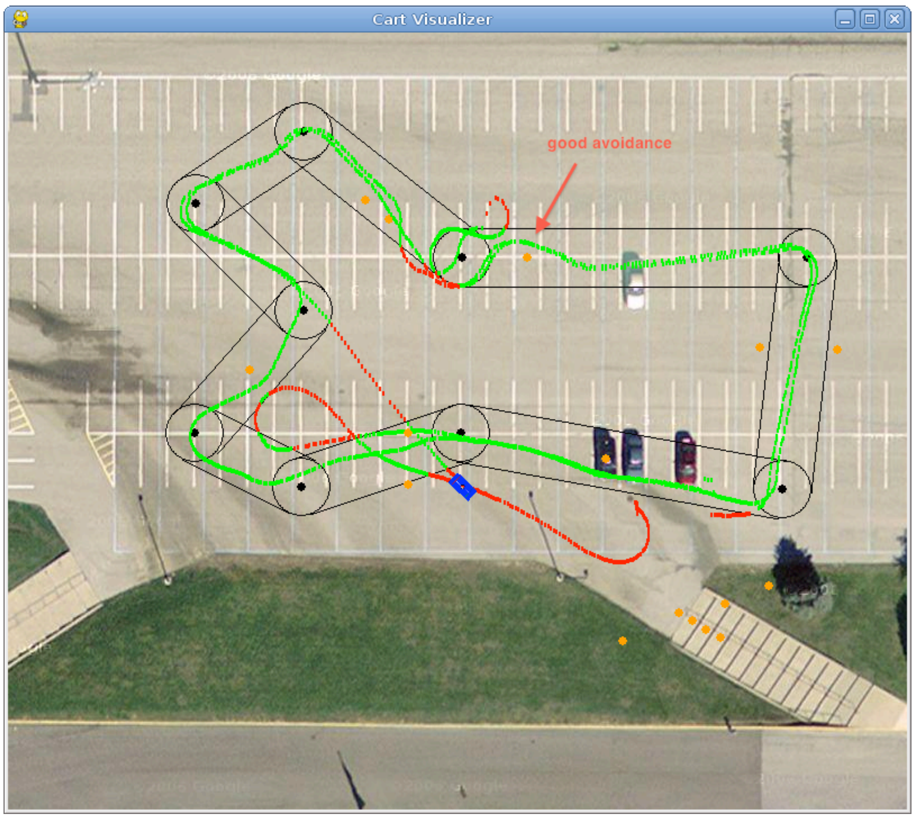
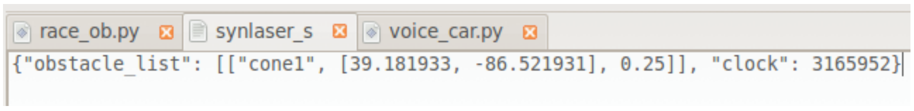


figure 22: a testing result from Lab 4

4.7) First testing result of physical obstacle avoidance.
We have no chances to have a field testing before the report deadline. However, as long as the physical laser still put the detect obstacle data into the “synlaser_s” file and with the same JSON format, our program will have no problems for avoiding the physical obstacles. But with a condition, the detection range is the longer the better, say at least 10 meters. If it is required, we look forward to have a field testing even after the report deadline.



```
{"obstacle_list": [["cone1", [39.181933, -86.521931], 0.25]], "clock": 3165952}
```

figure 23: the JSON format of file synlaser_s

5) Future Work

5.1) PD-Loop for computing throttle increment/decrement

As professor Johnson suggested, it is better to use a PD-loop to help the speed control as well as the steering sensitive. We are only make throttle settings for the lab, so that is the reason why sometimes we feel “cranky” when we are on the vehicle, because we set the throttle to 0% and 50% too often. This would be a very good further work but we have no resource for this implementation yet.

6) Personal Criticism

6.1)
As professor Johnson suggested, I did excessive simulation studies and those are not able to accurately describe the vehicle model. So this may have caused some time wasted and so I got less time for the field testing and tuning.

6.2)
Our course suggests that we work on the lab project in group, but I am more used to work alone. Although there was a classmate wanted to work with me in group, but we hardly shared any useful information. This is also a reason that I got very little time for the field testing, as I spent too much time on the designing and implementation, so there is no one can help me to run my own field testing at the same time. This might be a reasonable factor that I should lose some grades for this course.

7) References

[1] http://code.google.com/p/geopy/source/browse/trunk/geopy/distance.py