Lab Report in ERTS for GPS follower

Kiran N Kumar, Weiran Li, Embedded Systems lab, Indiana University

May 1, 2011

Abstract

The ERTS navigational system at the Indiana university embedded systems lab comprises of a golf cart with sensors that enable us to study embedded and real time behavior in the practical world. This document will talk about implementing a GPS follower for the ERTS system which will also need to handle any obstacles that come in its path while following the GPS waypoints. The ERTS navigational system is accompanied with a simulator which can be used to gauge results before actually testing on the cart.

1 Introduction

The ERTS is a autonomous robot in development at the embedded systems lab, Indiana university. The goal of the experiment is to navigate the cart so that it follows a defined set of waypoints on a course. The waypoints are specified in terms of the latitude longitude 2–D format. The waypoints are also specified with an LBO - lateral boundary offset, which gives us something like a tunnel on the 2–D space within which to navigate our vehicle. Each corridor formed with the waypoints is also associated with a speed limit. All these parameters are represented in the RDDF format as specified by the DARPA challenge to be used for experimentation.

1, 39.181917, -86.5221208333, 1.5, 5.0
2, 39.1818975, -86.521724, 1.5, 5.0
3, 39.182143, -86.5217033333, 1.5, 5.0

Figure 1: RDDF DARPA format to specify waypoints
The ERTS navigational system has the following objectives to be met:

1. Follow GPS waypoints
2. Stay within the corridor boundaries
3. Cover the course with efficient speed
4. Avoid obstacles
5. Improve performance over runs

This document will cover the design and experimental procedures to meet the various objectives for the ERTS navigational system.

2 The simulator

The simulator is built with a filesystem interface to simulate the various sensors and actuators. The sensors in the system include: compass, odometer, GPS and laser range finder. The actuators in the system are the throttle, steering and brakes. The simulator is started with the following command.

$ ./cartsim

The cart simulator will write the files in CartFS to simulate the cart behavior. To visualize this behavior, we use the visualizer. The visualizer is started with the following command.

$ ./visualizer

The obstacles to be avoided are simulated through the synlaser - laser range finder, which reads all the obstacles in its viewing range of 145°. To start the laser range finder we use the following command.

$ ./synlaser

Although the simulator gives us an approximation of the results in the real world, there are a few deviations from the actual behavior of the ERTS system. The best results can be obtained through active testing in the real world with a combination of rapid prototyping with the help of the simulator.
3 CartFS - Cart File System

The filesystem interface is loaded into the memory and provides a medium for the different sensors and actuators to communicate with the program. The filesystem is updated once every 1/10 seconds. This forms the clock cycle for the cart. At every clock tick, the files for sensors and actuators are written. The files contain strings in the JSON format for communication between the different modules on the system.

The files and the JSON formats used in the system are:

- **clock** - provides the clock and the interval for update
  
  \{"clock":793668,"interval":100000000\}

- **driver_c** - describes the status of the cart - needs to be set to enabled for the cart to start
  
  \{"enable": "True/False - stops reads on jdriver device",
     "clock": "The clock value on which this data was written."\}

- **driver_s** - describes the command for the cart state - is used to calculate the turn
  
  \{"direction": "The target heading of the driver",
     "enable": "True/False - stops reads on driver device",
     "clock": "The clock value on which this data was written.

- **jdriver_c** - describes the status of the cart
  
  \{"enable": "True/False - stops reads on jdriver device",
     "clock": "The clock value on which this data was written."\}

- **jdriver_s** - describes the values to set the actuator.
  
  \{"direction": "forward", "enable": true, "clock": 743879,
     "percent_throttle": 80, "turn_radius_inverse": -0.454545454545,
     "percent_braking": 0, "mode": "auto"\}

- **compass_s** - gives the status of the compass
  
  \{"enable": true, "heading": 120.617260358, "clock": 743984\}

- **gps_s** - gives the status of the GPS
  
  \{"lat": 39.1822067112, "speed": 1.41449856283, "lon": -86.5223022219, 
     "heading": 120.617260358, "clock": 743984\}
- `vcs`s - gives the status of the odometer

```json
{"handpull_sw": true, "distance": 612.485766641, "speed": 1.41449856283, "clock": 743984}
```

All the fields are not mandatory and we only use the fields necessary to tune the performance of the vehicle.

## 4 Taking first steps - navigate a square

We have used a simulator function to test the code before actually deploying it on the original machine. The ERTS system runs on a control loop to control the steering. The PID variable was optimally set to drive in a approximated square at the value of 0.006 with the simulator.

![Figure 2](image)

Figure 2: above, left: Visual Telemetry for run with P=5 and slop=5, right: P=2 and slop=5; below, left: Visual Telemetry for run with P=2/3 and slop=5 (changes in code to turn earlier caused a rectangle), right, P=2/3 and slop=3

To drive the ERTS system in a square, we uses state changes. The vehicle starts going towards a initial set direction "North", as we move along the edge of a square we measure the distance covered with the odometer. As the distance of the edge is covered, we make a turn to move along the next edge of a square.
The turn is handled through a PID loop. The turn is computed as a function of the $P_{term}$ and the difference in heading. The maximum turn we are allowing on the steering column is pegged at $1/2$. This allows us to control the turn on the steering column to set the steering at a max value initially and as the difference between the heading and target comes down, the steering eases and eventually straightens.

The odometer measures the distance we have traveled including the turn and any deviations we take, so if the $P_{term}$ is not set appropriately this strategy will not converge to form a square.

The results in Figure 2 show the variation of steering with a change in $P_{term}$ for the same speed and turn angle. If the $P_{term}$ is set at a high value, the cart will oversteer and swerve around the turn before settling to the correct heading. If the $P_{term}$ is small on the other hand, there will be understeer and the square will not be completed.

5 GPS analysis

The GPS waypoints give us the latitude and the longitude on the earth’s surface. The distance between two waypoints is computed using the Vincenty distance formula built into the geopy module. This formula accounts for the ellipsoidal shape of the Earth. The Earth does not have a flat surface, and hence the distance between the 2 waypoints forms an arc which fits the surface of an ellipsoid sphere, Earth [1]. The geopy module has an destination API which calculates the distance between two waypoints and the azimuth.

$$distance.distance(waypoint1, waypoint2).kilometers/1000 \quad (1)$$

$$distance.distance(waypoint1, waypoint2).forward_azimuth \quad (2)$$
The API (1) gives the distance between the two waypoints, and the API (2) gives the azimuth of the heading from waypoint1 to waypoint2. The compass sensor installed on ERTS gives us our current heading. Using these parameters, we can compute the direction and the distance to the next waypoint.

The difference between the azimuth to the next waypoint and the heading of the cart gives the error in the direction and we compute the “turn” as a Linear function of the PID loop and use this “turn” to set our heading to the next waypoint.

6 Turn Analysis

To implement the turn a more human form, we let the ERTS get into the turn at a distance of 5 meters before the turn to compensate for the delay in actuator response. This would lead to a smoother turn and be a better approximation of human behavior. This can be implemented by checking if the distance to the waypoint is 5 meters at which point we start heading to the next waypoint (turning).

![Figure 4: Turning 5m before waypoint](image)

On observation of actual vehicle steering, the vehicle sets itself in the direction before half the turn is completed, there is a slight delay before the wheels are straightened and rest of the turn is used up to align the vehicle in a lane or straight path. The p-term value gives a linear function for the derivation of the turn which is not a very good model and we might miss the turn or not straighten quickly in many occasions. In order to perform better on the turns we used a variable $P_{term}$. 

6
\[ f(\text{turn}) = \begin{cases} P_{\text{high}} & \text{diff} > 60^\circ \\ P_{\text{low}} & \text{diff} < 60^\circ \end{cases} \]

We initialized a \( P_{\text{high}} \) and a \( P_{\text{low}} \) at the start. As we start the turn, we are using the high P value to enable a sharper turn at a higher speed, and as the turn comes close to completion, we choose 60\(^\circ\) as the mark, we use the low P value to compute the turn. Tuning for the low and high P value enables us to turn at higher speeds. This is an close approximation of human driving behavior where the steering is straightened before the turn is completed.

### 7 Path Analysis

On observing the course the vehicle navigates, it is prudent to add more waypoints to give the path a well defined shape and tighter control.

We achieve this using simple principles of analytical geometry, where we divide the line segment two waypoints into 3 segments, each of which are exactly one-third the length of the original. This places emphasis on smoothing the curve or path followed by the vehicle while navigating through the waypoints. Let the segment between the two waypoints \( P_1 \) and \( P_2 \) be of length \( l \), then the two intermediate waypoints are added at \( l/3 \) and \( 2l/3 \) as shown in the figure.

![Figure 5: Determining the intermediate waypoints](image)

We use two intermediate waypoints, one towards the entry and one towards the exit. We track the actual waypoint using \( \text{waypoint\_latlon} \) and the distance to the waypoint as \( \text{distance\_to\_target} \). The intermediate waypoints or the current waypoint are tracked through \( \text{curr\_latlon} \) and the distance is tracked through \( \text{distance\_to\_current} \). If we reach the first intermediate waypoint along the path, we move towards the next intermediate waypoint by setting the \( \text{curr\_latlon} \) as the next \( \text{int\_latlon} \).

\[
\text{curr\_wp}' = \begin{cases} \text{int\_wp}_{\text{next}} & \text{dist}_{\text{curr\_wp}} < 5 \\ \text{curr\_wp} & \text{otherwise} \end{cases}
\]

As we reach the last intermediate waypoint along the path we set the \( \text{curr\_latlon} \) to the \( \text{waypoint\_latlon} \) and begin to compute the intermediate waypoints for
the next path.

\[
\text{curr}_{\text{wp}}' = \begin{cases}
\text{target}_{\text{wp}}' = \text{target}_{\text{wp}}\text{next}, \text{int}_{\text{wp}}' = f(\text{target}_{\text{wp}}') & \text{if dist}_{\text{target}_{\text{wp}}} < 5 \\
\text{int}_{\text{wp}}\text{next} & \text{if dist}_{\text{curr}_{\text{wp}}} < 5 \\
\text{curr}_{\text{wp}} & \text{otherwise}
\end{cases}
\]

We also include a small shift of the intermediate waypoint towards the outer edge of the waypoint it has to enter and include a small shift of the next intermediate waypoint towards the inner edge of the waypoint the vehicle exits. This provides for an accurate level of balance towards the entry and exit of the turn. This could be improved further by specifying a number of waypoints to manage the turn based on the turn angle, speed of entry and speed of exit.

8 speed control

The speed is controlled in ERTS with the help of braking to improve the average speed and time to complete to course.

Initially we had set the throttle percentage \( T' \) at 60 which proved to be slow especially in the uphill part of the course. We then increased the throttle percentage to 80 at the start and clipped it to a speed \( S \) of 6m/s.

\[
T' = \begin{cases}
80 & S < 6 \\
60 & S > 6
\end{cases}
\]

When the ERTS system is within a distance \( D \) of 10 meters from the waypoint to turn, we set the braking percentage \( B \) at 50 and the throttle percentage at 0,
Figure 8: Speed variation over the course which enables the vehicle to slow down and the turn is completed in a smooth manner. Initially, we had the braking percentage $B$ set at a constant value of 50, which performed well on the fast corners but slowed down inappreciably in the slow corners, sometimes coming to a stop.

After studying the braking attributes for a couple of tests, we modeled the behavior of the brake $B$ as a linear function of the speed $S$, which improved the performance, but it could not adapt itself to brake efficiently for the faster and slower turns.

\[
B', T' = \begin{cases} 
50, & D < 10 \text{ and } S > 2.5 \\
0, & D > 10 
\end{cases}
\]

We applied a lower bound on the speed $S$ to disable braking $B$ for the slower turns. If the speed was below a threshold 2.5, we did not change braking percentage $B'$ and set the throttle $T'$ to speed up. This causes a small ping-pong effect but is efficiently managed by the system.

\[
B', T' = \begin{cases} 
30 \cdot S, & D < 10 \text{ and } S > 2.5 \\
0, & D > 10 
\end{cases}
\]
After further observations, we fitted a quadratic function to the braking data. This achieved a good approximation of braking behavior on the test field which can be seen in the results.

\[ B', T' = \begin{cases} 
10 \cdot S^2, & D < 10 \text{ and } S > 2.5 \\
B, & D > 10 \text{ or } S < 2.5
\end{cases} \]

9 Obstacle Avoidance

The heading of the vehicle is guided by more dense waypoints. We add an additional point every 3 meters between two waypoints.

In the first run, if there is an obstacle detected, the locations of the nearest waypoints will be changed accordingly. And that obstacle will be added to obstacle list for later optimization. We consider whether the turn is clockwise or not. If the turning is clockwise, the vehicle should turn right to avoid the obstacle and
make a more smooth turn. Otherwise, to turn left is better. Also the distance between the obstacle and the ideal path is considered to decide the direction of the turn. If the obstacle is far enough from the path, but seen by the vehicle, we let the vehicle turn towards the path. If the obstacle is on the path, then the direction is based on whether it is clockwise turn as said before.

![Figure 11: Added shifts to the waypoints when there is an obstacle near by](image)

In the next run, if the vehicle detects an obstacle, we first check if the obstacle is in the obstacle list. If it is not there, we continue to use the same method as that in the first run. If it is in the list, we move the waypoints near that obstacle more further as long as they stay in the boundary, to help turn smoothly. Since gps longitude and latitude may jump from time to time. We allows local displacements when comparing obstacles.

![Figure 12: Adjust the shifts in the next run](image)

In order to minimize the influence of fake obstacles, the additional waypoints is not added based on the obstacles, and we have constraints on the shifts which added to the waypoints. Moreover, we just consider the obstacles seen by the vehicle with in (-15 degree, 15 degree), and 10 meter ahead. This works well when the vehicle goes straight. But if the vehicle was just after a sharp turn and haven’t back to the path yet, the avoidance might fail because of the angle constraint. Therefore, in this case, we have plan B by considering the distance between the obstacle and vehicle first. The performance might not be the best at the beginning. But we can get better in the second run.

![Figure 13: Angle constraints when detecting obstacles](image)
10 Field Test

We continue obstacle avoidance base on the path plan project which proved quite successful. In path plan test, the ERTS managed to follow the GPS points within a high level of accuracy, there were a few more than expected deviations on certain turns which would be due to the low $P_{term}$ of 0.003. Increasing the $P_{term}$ should facilitate better turning in the corners. The results are shown below.

![Figure 14: Path Plan Field Test Result: Visual Telemetry for run with P=0.003, turn analysis, path analysis and speed control to follow GPS waypoints](image)

The improvements from the path analysis and speed control gave us a better control over the cart and we were able to improve the steering performance and the speed while navigating the course.
In the obstacle avoidance test, there are two real obstacles. One is in the first segment, and the other is in the second segment. The obstacle avoidance results are shown in figure 15 and 16. From the result, real obstacles are avoided, and fake ones are ignored as desired. There is a little oversteer when see an obstacle, this is due to the higher variable P term which causes oversteer at high speeds. We avoid the obstacles really quickly, and maintain the speed. There is definitely a trade off between speed and accuracy to measure the performance of the ERTS in obstacle avoidance.
11 Conclusions

The GPS follower works acceptably in turning towards the next waypoint based on the azimuth. The throttle and braking on the cart seems to model human behavior very closely based on the polynomial function, braking when close to a turn or on seeing an obstacle. The turning at corners has improved vastly with negligible oversteer due to the braking and the variable $P_{term}$.

The addition of waypoints to the path helps clearly define a strategy for the cart to follow, smoothing the entry and exit from a waypoint. Obstacle avoidance strategies to avoid an obstacle as soon as we see it tend to work on straight paths. Improvement of obstacle avoidance through additional waypoints and a memory based learning technique tends to improve the results.

Further work can be pursued in realizing a high level planner which reacts on seeing an obstacle and plans the path to provide smooth course following. This could be achieved by running the simulator in parallel and updating the waypoints based on the simulated outputs.
References