EIE3105 Report

LIU Tianyi 15102892d

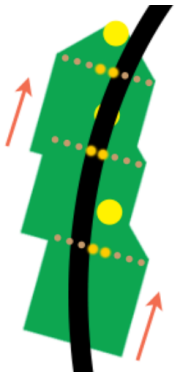
This report gives a brief overview of the work I have done in EIE3105 Demo 2-4. Technical details are also included. The codes and implementations have been uploaded to Blackboard.

Demo 2: Line Tracking

In Demonstration 2, only P (proportional) control is adopted, as the performance is already stable and the steady-state error does not affect the performance of the robot car too much. The robot car uses the SPI protocol to obtain the data from the photo-resisters soldered on the floorboard. Each bit of the received data represents the state of the photo-resister where '0' indicates there is no infra-red detected (absorbed or no reflection, i.e., black region) and '1' represents reflected light is detected (i.e., white region). While, the robot car chooses to follow the left edge or right edge which based on the number of black line segments detected, to track the inner one or outer line.

Technical Details

Definition of Error



The robot simply considers the difference between the current position of the tracking line and a pre-defined position as the error to be eliminated. When the robot car follows the line alongside the left side, the error is defined as: $e = K - 3$. While for the right side: $e = K - 4.$, where K is the first zero bit and $K \in [0,7]$. This means that when the robot car on the track, the left edge should lie between the 3^{rd} and 4^{th} photo-resistor and the right edge should lie between the 5^{th} and 6^{th} photo-resistor.

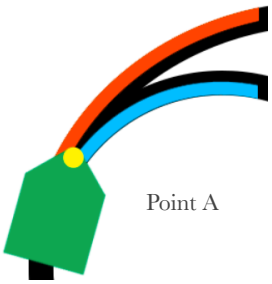
PID Control

Only P control is used in this demonstration. The control scheme uses a differential control scheme that continuously changes the value of PWMs inversely. Left wheel PWM: $PWM_L = v_{nominal} - k_P \times e_P$. Right wheel PWM: $PWM_R = v_{nominal} + k_P \times e_P$.

Detection of Black Line

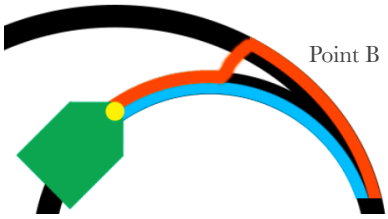
As the orthogonal black line is the key to know the exact location of the robot car, the control scheme adopts a rising-edge-triggered detection. The program stores the previous SPI detection and counts the black line only if the present SPI detection is 0 (all black) while the previous SPI detection is not 0 (not all black), i.e., a transition from white to black. This guarantees that the robot car will not count multiple times at one black line.

Change of Track and U-turn



One of the case that the robot car needs to handle properly is the changing of the track. Instead of using hardcoded program that lets the car turn right with a time delay. The control scheme changes the side of line the robot car is following when it is needed to change the track. As shown on the right, at point A, when the robot car follows the left

side (orange), the robot car will enter the outer track. While following the right side (blue) will let the robot car follow the inner track. The changing of the track is based on the count of the black line using the above-mentioned method.



Moreover, tracking following the right also solves the discontinuity problem when exiting the inner track at point B, as shown on the right. Experiments show that the discontinuity may cause certain disturbances of the robot car while the right-tracking method actually makes the robot car more stable.

It is also required to perform a U-turn when the robot car finishes the clockwise tracking to perform counter-clockwise tracking. This is also accomplished by utilizing the count of the black line. When the car meets the 6^{th} black line at Point B, it will stop tracking and perform a U-turn for a certain period of time with a hardcoded delay. When the U-turn is completed, it will resume tracking to finish the rest of the course.

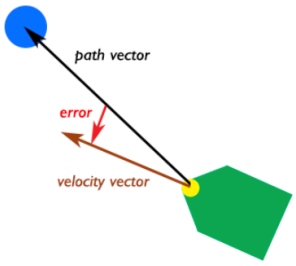
Demo 3: Hit Three Balls

P (proportional) and I (integral) control are used in this demonstration as hitting a ball requires certain accuracy. Integral control is incorporated for eliminating the steady-state error. One cap is mounted on the car to serve as an identifier. Different from the two-cap design, the one-cap design is much more robust that only requires one coordinate for navigation. Comparing to the two-cap design, the one-cap design is more challenging as well.

Technical Details

Definition of Error

Instead of defining error as the difference of x and y axis values, the program defined the angle difference between the path vector and velocity vector as the PID error. For consistency, $0°$ is defined as the direction of positive x axis, and the angle increases in the clockwise direction. This yields the angle of the path vector: $\theta_p = deg(loc_{ball} - loc_{car})$ and $\theta_v = deg(loc_{car} - loc_{car\_previous})$ for velocity vector. $deg(\cdot)$ is a defined function for converting slope into degree. Therefore, the error to be controlled: $e = \theta_v - \theta_p$.

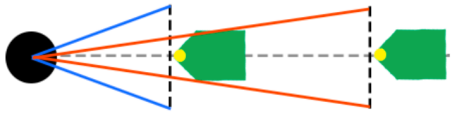


PID Control

PI control scheme is used in Demonstration 3. In addition, there is a dedicated design for the PID control parameters. As illustrated below, when the robot car approaches the ball, a same orthogonal offset will create a large error (blue angle) comparing to the scenario when the robot car is far from the ball (orange angle). Therefore, the car should prevent from having rapid changes when the distance between the car and ball are relatively small. As a result, the program linearly maps the weighted parameter according to the distance as:

$$k_P = \frac{dist(car, ball)}{1250} \times u(500 - dist) + 0.4 \times u(dist - 500),$$

where $u(\cdot)$ is a unit step function. This lets the robot car adjust itself when it is far away from



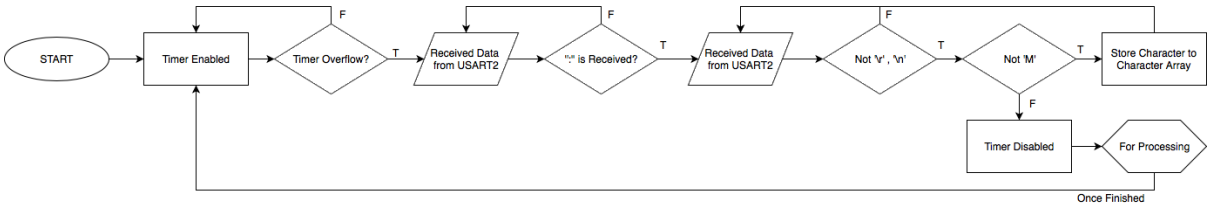
the ball and prevents the robot car from having large adjustments when the distance is small. For PID control, the formulae are the same. For integral control, a constant weights parameter $k_I = 0.1$ is used in Demo 3. This is further improved in Demo 4.

Left wheel PWM: $PWM_L = v_{nominal} - (k_P \times e_P + k_I \times e_I)$.

Right wheel PWM: $PWM_R = v_{nominal} + (k_P \times e_P + k_I \times e_I)$.

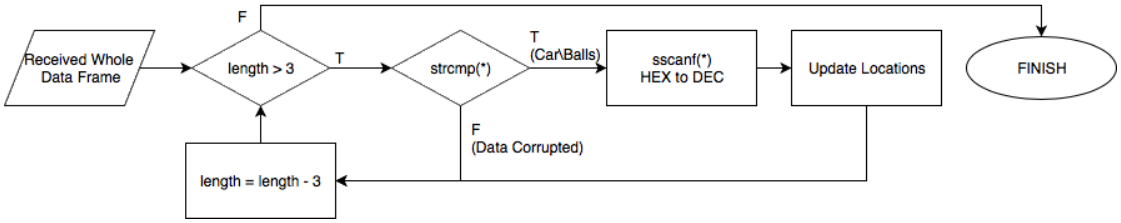
Obtaining and Processing Information from WiFi Station

The Wi-Fi station sends the locations of balls and car repeatedly. However, processing all the information received will lag the processing of other information. To counter this problem, a 0.15 s timer is set and polls for the information to update the locations. First, when the timer interrupt flag is set, the program will poll for the character ':' which serves as the starting point of meaningful location data. Second, the program will store the characters received except for '\r' and '\n' in a character array for further processing. Then, when the USART receives the character 'M', which comes from the command "CMD", it will stop reading and disable the timer.



The C library <string.h> is used to compare the identifiers and convert the hexadecimal numbers into readable decimal numbers for location updates. Once the processing is completed, the timer will be re-enabled and wait for another overflow event. Each object is represented by a C structure, which contains its x and y coordinates. There are also supportive function sets to manipulate the structures defined in the program.

```
typedef struct loc_struct
{
  volatile int X;   //  x axis
  volatile int Y;   //  y axis
}loc_struct;
```



Calculation of Direction of the Robot Car

With the locations of two caps, the program will be able to calculate the real-time direction of the car directly. However, with only one-cap, this process will become a dynamic calculation. First, the program stores the past location of the robot car. When the robot car is initialized, there is a hardcoded 200 ms forward movement for the robot car to obtain the initial direction. Then, it becomes possible to calculate the direction with the velocity vector obtained from two consecutive location measurements. However, it is also noticed that the one-cap design will decrease the resistance of delay. In the case the car is

rotating, due to the dynamic calculation and Wi-Fi delay, it will be very difficult for the robot car itself to recover from the rotation. Nevertheless, this will not affect the performance of the robot car frequently.

Hitting and Returning

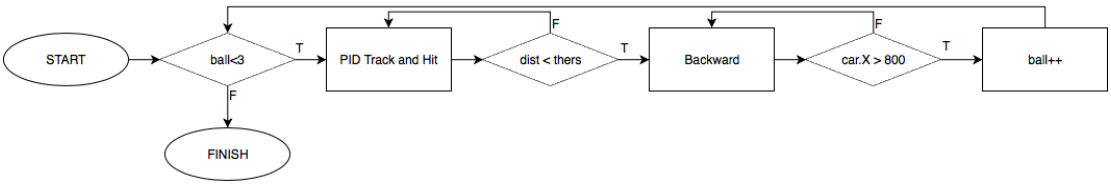
There is a certain delay between the Wi-Fi transmitted information and the real-time locations. Consequently, the control scheme defines a slightly higher distance threshold. When the distance between the ball and car below the threshold, the control scheme will consider the car has already hit the ball.

Actually, there are two different thoughts on implementing the hitting process.

1. Slowly and continuously push the ball until it enters the green region.
2. Hit the ball only once with an appropriate speed.

While the first idea guarantees the ball to enter the specified region, considering time efficiency, the control scheme adopts the second idea with certain experiments to make sure the strike force is enough.

After hitting the ball, considering both timing and practicability, the robot car simply moves backward until its x axis value reaches 800, slightly behind the edge of the green region. This gives the robot car enough space to adjust its direction to hit the next ball.



Demo 4: Hit Balls in Turns

Very similar to Demo 3, Demo 4 uses most of the idea and implementations in Demo 3. P (proportional) and I (integral) control are adopted. The major difference between Demo 3 and Demo 4 is the location of the ball is not fixed. This requires the robot car can hit the ball at all possible locations. To counter this problem, the control scheme considers the hitting-ball problem as a hitting-target problem with several modifications comparing to Demo 3.

Technical Details

Definition of Error

Same as Demo 3.

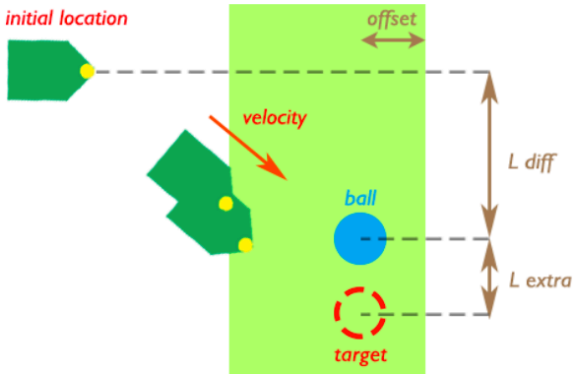
PID Control

Demo 4 is largely based on the implementation of Demo 3. While for PID control, the weight parameter for integral control is also linearly mapped as: $k_I = \frac{dist(car, ball)}{4000} \times u(500 - dist) + 0.1 \times u(dist - 500)$, where $u(\cdot)$ is a unit step function.

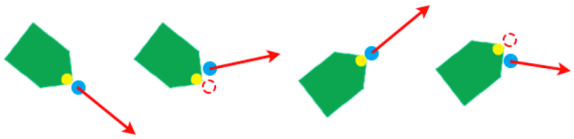
From Hitting-Ball to Hitting-Target

In Demo 4, the coordinates of the ball are varying every time. Therefore, the optimal solution may not be hitting the ball directly, as the trajectory and reflections will be hard to predict. The program shifts from hitting a ball to

hitting a target which is defined by the geometry relationships. As shown below, in an ideal case, the final velocity and angle before hitting the ball are related to the y-axis difference (L_{diff}) between the initial location and the location of the ball. Consequently, the y axis of the target offset L_{extra} is defined as $\frac{1}{10} \times L_{diff}$. Additionally, the ball may stop at a different location in the green region. Therefore, using the same speed to hit the ball will not guarantee the ball to enter the opposite green region. In this case, an offset is defined as the distance the ball entered the green region and this is positively mapped to the speed of hitting: $v = v_{nomimal} + \frac{1}{20} \times L_{diff} + \frac{1}{10} \times offset$.

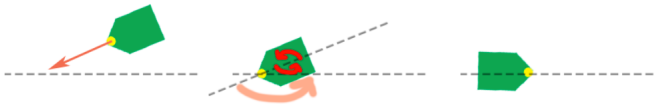


Also, this hitting-target design lets the robot car hit the ball using the diagonal side instead of the tip, which makes the ball moves more horizontally (parallel to x axis) than before. This also reduces the uncertainty of reflections to some extent.



Returning to the Base

To hit the ball next time with a relatively broad field of view, it is required the car should move back to the original location as nearly as possible. However, this involves several dedicated designs in the control scheme. First, after hitting the ball (target), the car will move towards a base location (near the starting point) using exactly the same control methodology. The base location also involves the hitting-target deigns mentioned above. When the distance below the threshold, the last velocity angle will be recorded. Then, the robot car will rotate with a time duration calculated through experiments: $t_{rotate} = 3.5 \times [\theta_v u(180° - \theta_v) + (360° - \theta_v) u(360° - \theta_v) u(\theta_v - 180°)]$. where $u(\cdot)$ is the unit step function. This roughly let the robot car heads towards 0° to get ready for the next hit. For a returned velocity larger than 180° and smaller than 180°, the robot car will rotate clockwise and counter-clockwise respectively.



Other Details in the Implementation

Although the car is designed to track the ball with the most updated location, due to the decay of weight parameters of PID controller, it becomes even more difficult for the car to track a moving ball. Therefore, to ensure the accuracy of hitting, the robot car will wait until the ball is fully stopped by obtaining two measurements in 0.2 s (5 Hz) which is slightly lower than the update frequency (0.15 s/6.67Hz).