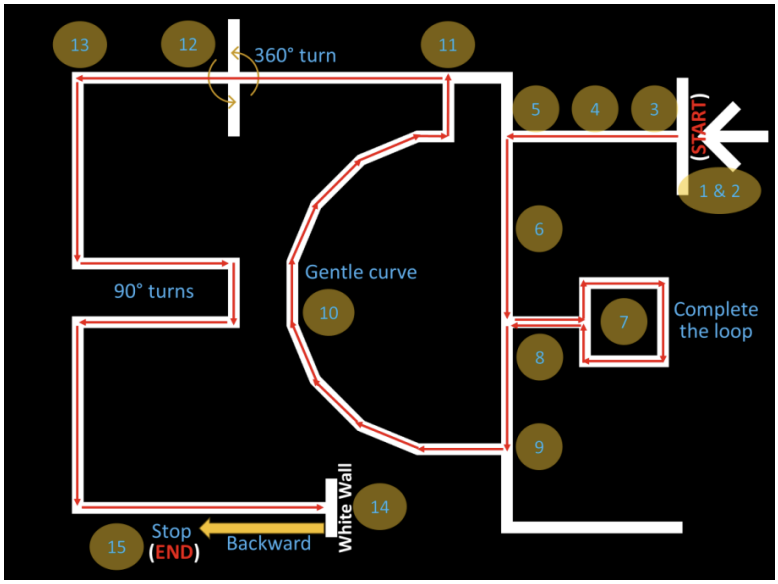
ELEC 1100 Final Project Report
Group_095 Chow Yuet Hey Samantha

Introduction

This final project required the robot car to move forward along the white line, complete specific turns and loops following the route directed in red arrows to the white wall and move back by at least 30 cm before stop, while ideally completing all tasks under 25 seconds. The map is shown below:



To finish this task, a minimum of 3 sensors (left sensor, bumper sensor, right sensor) and 2 motors with changing PWM values should be installed in the robot car to control the direction, speed and moving angle of the car, in which they are controlled by logic gates and counters in the implemented code with proper connection between the integrated circuit.

Logic Design

i) System Input and Output Relationship

Main system

Left Sensor	Right Sensor	Left_DIR	Right_DIR
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	1

go_straight() function

Left Sensor	Right Sensor	Left_DIR	Right_DIR
0	1	0	1
1	0	1	0
1	1	1	1

The car is triggered when bumper sensor sensed 0 and it followed the main system truth table throughout the entire track. Car would turn left after approaching each T-junction 0 0, turn left after sensing a left route 0 1 and turn right after sensing a right route 1 0. If the car followed along the white line with sensors exceeding white area, the car moved forward.

In our self-defined function go_straight(), we used this control logic as an angle fine-tuning function to make minor adjustments with the direction of the moving car. In some occasions, especially in sharp left and right turns, we would want to first make the car turning slightly towards left or right where both sensors could sense 0 0 as if the junctions are Y-splits, then make a full turn.

When the car reached the white wall, bumper sensor sensed 0 again and move backwards. The car would stop when the number of counters were fulfilled.

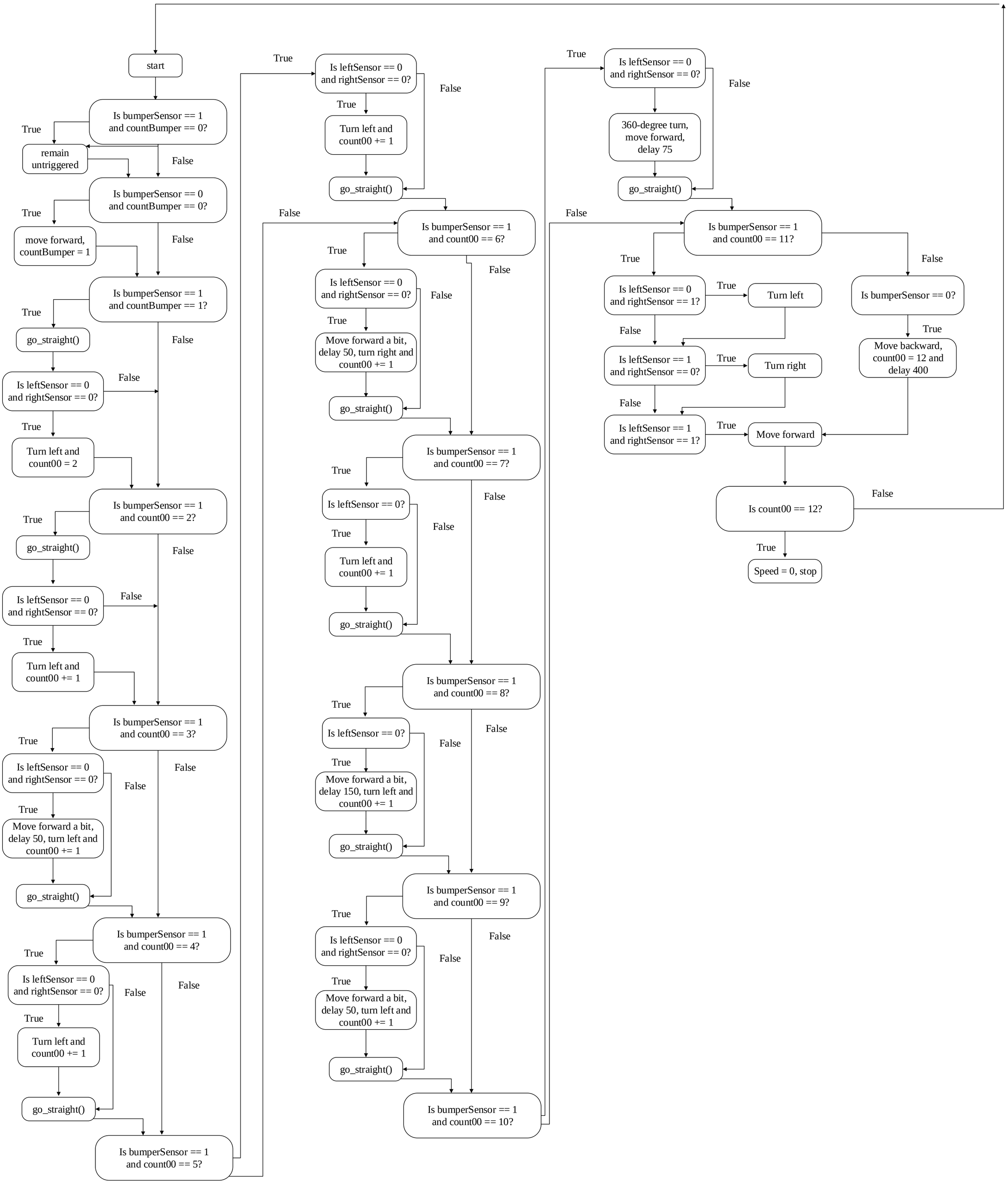
ii) Logic-flow Chart

(Please read the Logic-flow Chart on the **next page** as it does not fit in this section.)

iii) PWM values

In the analogWrite function, we used a range of PWM value of 200 - 245 for both left and right motor when the car is turning left and right during most of the tasks. However, we kept the forward motion 1 1 to be a PWM value of 100 throughout the gentle curve to leave enough time for the car to sense 1 0 and make slight right turns without rushing outside of the map due to high speed. In our self-defined function go_straight() for the use of fine-tuning moving angle and during the 90-degree sharp turns, we made PWM value difference of 50 between 2 motors when making left and right turns, which left motor was faster during left turns and right motor was faster during right turns.

Between the submitted code versions, we have adjusted the PWM values mostly by increasing its value with delay time during sharp turns and decreasing its value during short forward motion, which did help the car to perform smoother in the square loop.


start
Is bumperSensor == 1 and countBumper == 0?
True
remain untriggered
False
Is bumperSensor == 0 and countBumper == 0?
True
move forward, countBumper = 1
False
Is bumperSensor == 1 and countBumper == 1?
True
go_straight()
False
Is leftSensor == 0 and rightSensor == 0?
False
True
Turn left and count00 = 2
Is bumperSensor == 1 and count00 == 2?
True
go_straight()
False
Is leftSensor == 0 and rightSensor == 0?
False
True
Turn left and count00 += 1
Is bumperSensor == 1 and count00 == 3?
True
Is leftSensor == 0 and rightSensor == 0?
False
True
Move forward a bit, delay 50, turn left and count00 += 1
go_straight()
Is bumperSensor == 1 and count00 == 4?
True
Is leftSensor == 0 and rightSensor == 0?
False
True
Turn left and count00 += 1
go_straight()
Is bumperSensor == 1 and count00 == 5?
True
Is leftSensor == 0 and rightSensor == 0?
False
True
Turn left and count00 += 1
go_straight()
False
Is bumperSensor == 1 and count00 == 6?
True
Is leftSensor == 0 and rightSensor == 0?
False
True
Move forward a bit, delay 50, turn right and count00 += 1
go_straight()
Is bumperSensor == 1 and count00 == 7?
True
Is leftSensor == 0?
False
True
Turn left and count00 += 1
go_straight()
Is bumperSensor == 1 and count00 == 8?
True
Is leftSensor == 0?
False
True
Move forward a bit, delay 150, turn left and count00 += 1
go_straight()
Is bumperSensor == 1 and count00 == 9?
True
Is leftSensor == 0 and rightSensor == 0?
False
True
Move forward a bit, delay 50, turn left and count00 += 1
go_straight()
Is bumperSensor == 1 and count00 == 10?
True
Is leftSensor == 0 and rightSensor == 0?
False
True
360-degree turn, move forward, delay 75
go_straight()
False
Is bumperSensor == 1 and count00 == 11?
True
Is leftSensor == 0 and rightSensor == 1?
True
Turn left
False
Is leftSensor == 1 and rightSensor == 0?
True
Turn right
False
Is leftSensor == 1 and rightSensor == 1?
True
Move forward
False
Is bumperSensor == 0?
True
Move backward, count00 = 12 and delay 400
False
Is count00 == 12?
True
Speed = 0, stop

Debugging Report

i) Case 1 (counter counting system)

One of the major problems that we encountered during the programming part was setting up the correct counter counting system. At first, we used count00, countL, countR to keep track of the total numbers of times approaching white line, left turns and right turns respectively. We then tried to make use of these counters in the if-conditions to make the car do certain tasks if the car has reached certain amount of count00, countL and countR. However, this method could easily cause miscounting of sensing white line and turns due to lots of factors such as insufficient or too much delay time, speed and energy from battery. Hence, lots of code were either executed at the wrong position or the performance was entirely skipped on the demo mat.

Our solution was to only use count00 as our counter in the if-conditions so that there were less requirements to meet the conditions, less chance to miscount and easier to keep track of the car movement after each individual task was completed. After each count00 if-condition was met, the car would have to pass another if-condition of sensing (!leftsensor && !rightsensor) before making turns or moving forward.

ii) Case 2 (adjusting sensor sensivitiy)

During all the testing sections, we kept encountering sensing problems with our left and right sensor and their installation position on the car. In some cases, left and right sensors could not sense 0 0 instantly together at T-junctions, which causes the car to make unexpected turns or wiggle in the middle of the route. In some cases, the left and right sensor kept sensing 0 0 along the straight line where it should be sensing 1 1, hence the car started making repeating 360 turns.

Our solution was to separate the sensors further outward on the transceiver mount bar to ensure the sensors can clearly distinguish white line and black mat. Then, we kept adjusting the knob on the sensor using a screwdriver while making slight left, right turns and moving towards T-junction to enhance the synchronization of the sensors and their sensitivity while encountering white area or exceeding any white area. Also, it was also important to straighten the IR receiver and IR emitter because it would cause inaccurate sensing result even if they were just slightly crooked.

Results and Conclusion

The part that was well completed in this project was building the circuits with my groupmate. Our final product was built clean and tidy, and we almost did not encounter problem with the constructed circuits such as bad wire connection. Another part that was well done was the Arduino coding as me and my groupmate spent lots of time discussing how to write the individual parts corresponding to the expected tasks and how to include minor adjustments along with the car movement, which we figured through creating go_straight() function and incorporating this fine-tuning function either before or after each turn and it helped the car to sense more precisely.

The part that was challenging was the code and hardware debugging. It was very difficult to find a perfect combination of PWM values on each motor and the delay time as they made a huge impact on whether the car can make a successful turn. Even during the demo trial, we still encountered situations where the sensor can successfully sense 0 1, it only wiggled a bit and continue moving forward due to either insufficient speed for the left motor to move backward, or insufficient delay time for make a slight turn to sense 0 0 that make a confirmed right turn.

During the project period, there were unexpected situation similar to our difficulty of debugging happened such as the car kept turning right even if the car can sense 0 1 and the logic flow of the code is correct. We fixed this problem through increasing the PWM value of the left motor to quicken the left backward movement so that the car can make a sharp left turn. During the demo trial, we also tried to place the car a bit closer to the left to solve the problem.

If I can start over, I will first create the truth table and a brief logic flow chart as a guide before started Arduino coding so that it is easier to extend the logical thinking on top-of-mind map. I would also consider recreating a 1-to-1 demo mat with the same material so that me and my groupmate can have a lot more trial testing before the demo. If I have more time for the preparation, I may install one more sensor on each side of the mount bar (total of 2 left sensors and two right sensors) to give more precise sensing results.

Finally, I had learnt both techniques in software and hardware debugging and the process of completing this project also showed a lot of effective communication and work cooperation between me and my groupmate.

Word count: 1346