COMPSYS 301: Project Final Report

GROUP 15: Francis Cho Xuhan Liu Gayeon Kim Yu-gyeong Hong

1. Introduction

The group has been tasked with designing a path following a robot that will traverse a given map. The map will present the road distinguished between light and dark traces. These traces will be projected onto the floor forming a maze for the robot to navigate. The robot will operate and navigate autonomously using a path finding algorithm and light sensing. The robot must find the shortest path between 5 given locations in the "food list". This report will cover our groups' findings throughout implementing the hardware software system. Including the decisions and analysis for analogue/software design, justifying decisions made at each step. A major part of this project will rely on the functionality of the software. The software aspect is responsible for processing and reaction to the given input from the light sensing circuit. This includes the algorithm to find the shortest path from starting position to destination and wheel control.

2. Light Sensor Design

2.1 Phototransistor

The phototransistor used in the project is a TEMT6200. From the information provided by its datasheet, we decided to use $R = 47\ k\Omega$ to provide an appropriate voltage output into the filter component as well as staying below the maximum ratings.

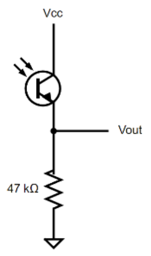

Figure A: Light sensor circuit only

2.2 Filter

For the filter design, we had to filter out ambient light, so we read the inputs from the projector light only. From analysis using the FFT spectrum analyser on the oscilloscope, we identified a major source of noise (the fluorescent lights) to be around 90 Hz, which needed to be filtered out. Thus, the cut off frequency can be set at 90 Hz.

$$R = \frac{1}{2\pi f_c C} = \frac{1}{2\pi(90)(10 \times 10^{-9})} = 176839\ \approx 180\ K\Omega$$

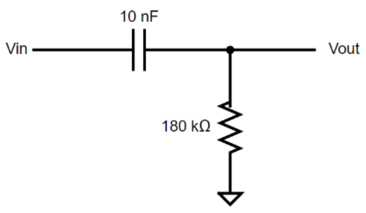

Figure B: Passive high pass filter only

We decided to use a passive high pass filter with capacitor value of $10\ nF$ resistor value of $180\ k\Omega$. The resistor value chosen was the nearest E12 value based on our calculation.

Thus, the light sensing circuit is composed of two components: (1) Light sensor (2) Filter.

After implementation of the passive high pass filter, the light sensors were consistently reading around 50 mV peak values under path in projector light, and 1100 mV peak values outside of path. To verify correct values were being read, USB UART and the PUTTY terminal were utilised.

2.3 Circuit design decisions

We decided against implementing a voltage comparator or similar, but rather processed the signals in our software to determine whether or not the light sensors were inside or outside the path. A buffer was not necessary as our signal was stable and adding one did not provide much benefit except cluttering up the circuit.

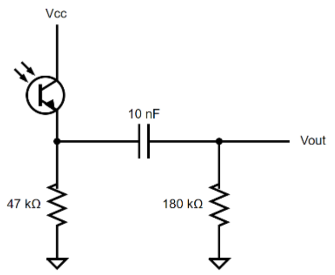

Figure C: Final light sensor design with passive high pass filter

3. PCB Design & Light Sensor Layout

3.1 PCB Design

When designing the PCB board, 6 light sensors were used in our group's design, one sensor was placed at the front of the board (Q1), two sensors were placed right below the front sensor (Q2 and Q3 respectively), two at the end of each side of the board (Q4 and Q5 respectively), and one below Q2 and Q3 that is aligned with Q1 sensor (Q6). In our code, Q1 sensor was used as 'front', Q2 sensor as 'left', Q3 as 'right', Q4 as 'far_left', Q5 as 'far_right' and Q6 as 'back'.

We decided to connect the LEDs to the PSoC instead of the power source (meaning they would be turned on the entire time) to allow more control and help with debugging. For the final demonstration, the LEDs will be configured to be turned on the entire time.

3.2 Light Sensor Placement

The 'front' and 'back' sensors are responsible for the straight motion and the 'front' sensor is also used for checking if the left or right turn has been completed. The 'left' and 'right' sensors were used to keep the robot on the track, adjusting the position of the robot left or right depending on the sensor detection when the robot is off the track. The 'far left' and 'far right' sensors are responsible for left and right turns and also for detecting the intersection. The details of the motion of the robot are explained in 4.3 section of the report.

The schematic and PCB design screenshots are attached in Appendix A as Fig.3. schematic of sensor design and Fig.4. PCB design.

4. Algorithm Design

For the robot to navigate the maze, a pathfinding algorithm was needed. To implement the algorithm, we tried 2 different algorithms, Dijkstra and A* using MATLAB and plotted to visualise the shortest path. Dijkstra's algorithm finds the shortest path between two nodes. This uses the weights of the edges to find the path that minimises the total distance (weight) between the source node and all other nodes. The only difference between is that A* tries to look for a better path by using a heuristic function, which gives priority to nodes that are supposed to be better than others while Dijkstra's explores all possible ways. Fig.1. Dijkstra contour displays the contours which separate the waves in Dijkstra which grow uniformly in all the directions. While Fig.2. A* contour shows the heuristic stretches the contours in the direction of the goal state in A*.

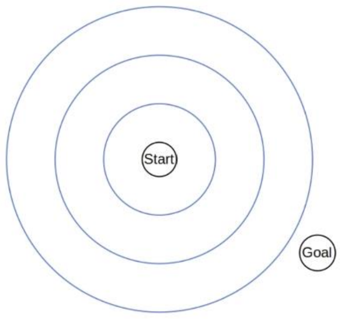

Fig.1. Dijkstra contour

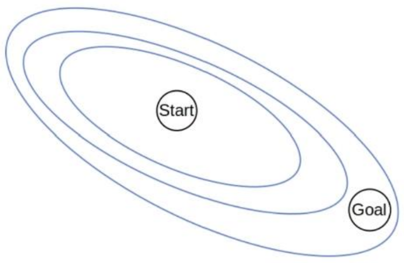

Fig.2. A* contour

For the final implementation, we used the Dijkstra algorithm. Since we decided to pre-process the pathfinding algorithm and store values for all turn actions required for the shortest path inside an array, the speed of the algorithm does not affect our robot performance. Additionally, the A* algorithm was implemented in MATLAB using various MATLAB functions which C does not support; therefore, it was quite difficult to convert to C.

5. Embedded Software & Integration

5.1 Design choice & Implementation

6 light sensors were used in our design for the projected path detection. Software was written in C and implemented using PSoC 5LP with the PSoC Creator IDE.

5.2 Dealing with Analogue Data from Light Sensors

Every time there is a timer interrupt, it initiates the start of a conversion. After the conversion has finished, it automatically triggers our *CY_ISR* interrupt which takes 10 readings, performs conversions to mV using helper functions and stores it inside a data array.

Each light sensor is connected to a channel of ADC_SAR_Seq. Therefore, we read ADC values using the function *ADC_SAR_Seq_1_GetResult16* which can read 16 bits of ADC values when ISR operates. When the robot is under the projector light, we read ADC values which were converted to a mV reading using *ADC_SAR_Seq_1_CountsTo_mVolts* function. We found 10 values of each channel and saved them in an array. To determine whether or not each light sensor was under the path, the maximum value of each set of 10 readings was calculated which would later be compared to the 'compare value'. We set our compare value to 500 mV because from testing, we discovered the peak voltage output from a light sensor inside the path was around 50 mV, while outside the path was around 1100 mV. Therefore, to differentiate between the two values of 50 mV and 1100 mV, our compare value of 500 mV was chosen. For example, if the far_right light sensor is inside the path, the converted ADC value is always smaller than the compare value.

5.3 Motor Control

In our design, PWM (Pulse Width Modulation) motor control which works by deriving the motor with a series of pulses and varying the duty cycle is used. A quadrature encoder which measures the speed and direction of a rotating shaft is utilised. The schematic for PWM and quadrature encoder is attached in the appendix A as Fig.5. Schematic of PWM and QuadDec.

To control the motor of the robot for each motion effectively, we set up the functions to control the motor speed. The functions: *PWM_leftmotor_Start()* and *PWM_rightmotor_Start()* to start the motors.

We used the compare values to control the speed of the motors. We set it up so if difference between *WriteCompare1* and *WriteCompare2* function values were larger, the motor speed increased.

We described each motor speed at each case of FSM. Due to the friction, we set the left motor speed to be faster than the right motor speed when it is going straight. Our group tried with some values and found out the best speed so that the turns are smooth and sensor detection is successful. The following motor control is written in the table in the Appendix A as Fig.6. Table for motor control .

5.4 The Finite State Machine (FSM) & Conditional Checks

The path following action of the robot was controlled using the FSM model. The states of our FSM included: *straight*, *straight_left_adjust*, *straight_right_adjust*, *left_turn* and *right_turn*. Our initialised FSM state is straight, so the robot follows a straight path once it is turned on. Furthermore, when the 'front', 'left', 'right' and 'back' sensors are on the path, the FSM state is set to 'straight' which makes the robot follow the straight path.

There were minor movements (left and right) when the robot was following a straight path, so we had to account for this by adding adjustments. The 3 light sensors (left, right, back) in the centre were used for the adjustment (FSM state = *straight_left_adjust* or *straight_right_adjust*).

The ‘far_left’ and ‘far right’ sensors are responsible for detecting turns which allows the robot to take a proper action for each case. For example, if only ‘far_left’ sensor is detected, the robot would make a left turn, if only ‘far_right’ sensor is detected, the robot would make a right turn. (FSM= *left_turn* or *right_turn*).

To determine turn completion, the ‘front’ sensor is used to detect the robot’s return to the path and the FSM is set to *straight*. Furthermore, the ‘front’, ‘far_left’ and ‘far_right’ sensors are responsible for detecting intersections. This allows the robot to decide on its location based on input from the sensors as well as the array containing relevant turn actions (output from shortest path algorithm). The full description of our system is represented in a flowchart (attached Appendix A as Fig.7. FSM flow chart).

5.5 Data structures

Data was continuously being collected and processed during operation. Simple data structures such as 1D and 2D arrays were utilised to store our data inputs and outputs from algorithm processing.

5.6 Shortest Path Algorithm Implementation

We pre-processed the algorithm and found the shortest path between all food particles and this data was stored inside an array called *turnActionArray*. Along with the path detection from the light sensors, this array kept track of the required number of turns to traverse between all the food particles.

In the array, ‘0’ indicates straight motion, ‘1’ is a left turn, ‘2’ is a right turn, ‘3’ indicates a complete turn-around. When there are intersections of a straight and left turn path, we used ‘10’ for going straight when there is an intersection of a straight path and a left turn path, ‘11’ for doing a left turn. Furthermore, ‘20’ is used for going straight for right turn and straight path intersections and ‘22’ is used for turning right in these intersections. With each completion of a turn, we incremented the current index of the *turnActionArray* array.

6. Testing and Verification

Final test cases we used. For example, test arrays for directed movement of robot, test map with food particle locations to simulate actual test. A wide variety of test cases were written to test various turns at intersections as well as overall map traversal and navigation.

Example test cases:

```
 // test case 3
int food_list[6][2] = {{1,17},
{13,13},
{8,13},
{1,7},
{3,1},
{13,1}}
```

The result was also checked and verified with MATLAB using the map function.

7. Conclusion

In this project, we have designed and implemented a line following robot integrated with a shortest path algorithm that allows it to navigate a map given a set of coordinates (food particles). An analogue light sensing circuit was designed to both filter out unwanted noise (fluorescent light) as well as providing a signal for our PSoC to interpret and use for motor control and path navigation. The PCB for the light sensing circuit was built in Altium Designer using SMT components.

This project was a very rewarding experience, with many skills gained in electrical design, computer programming with PSoC and project implementation and management skills.

Appendix A

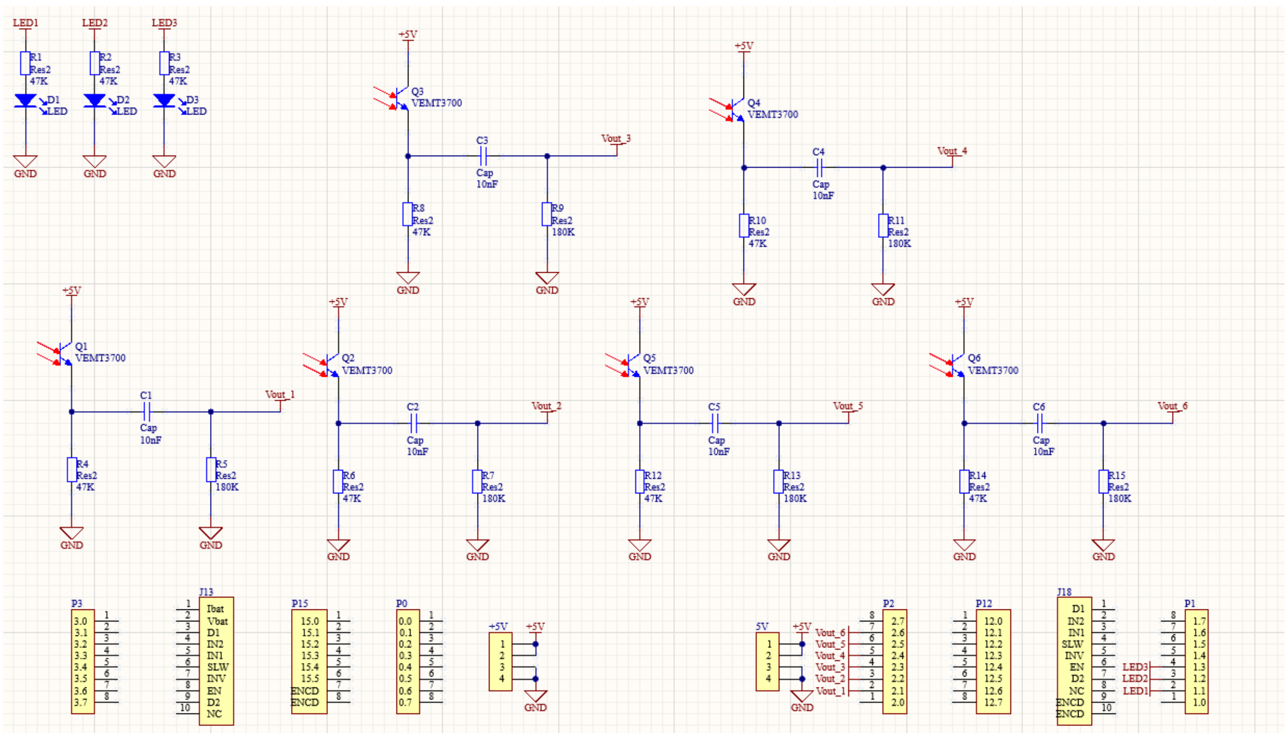

Fig.3. Schematic of Sensor Design

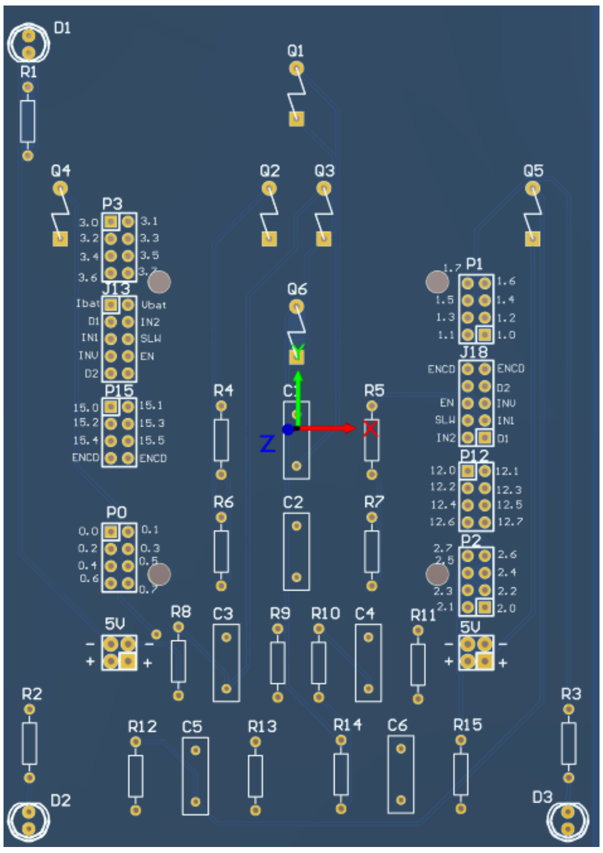

Fig.4. PCB design

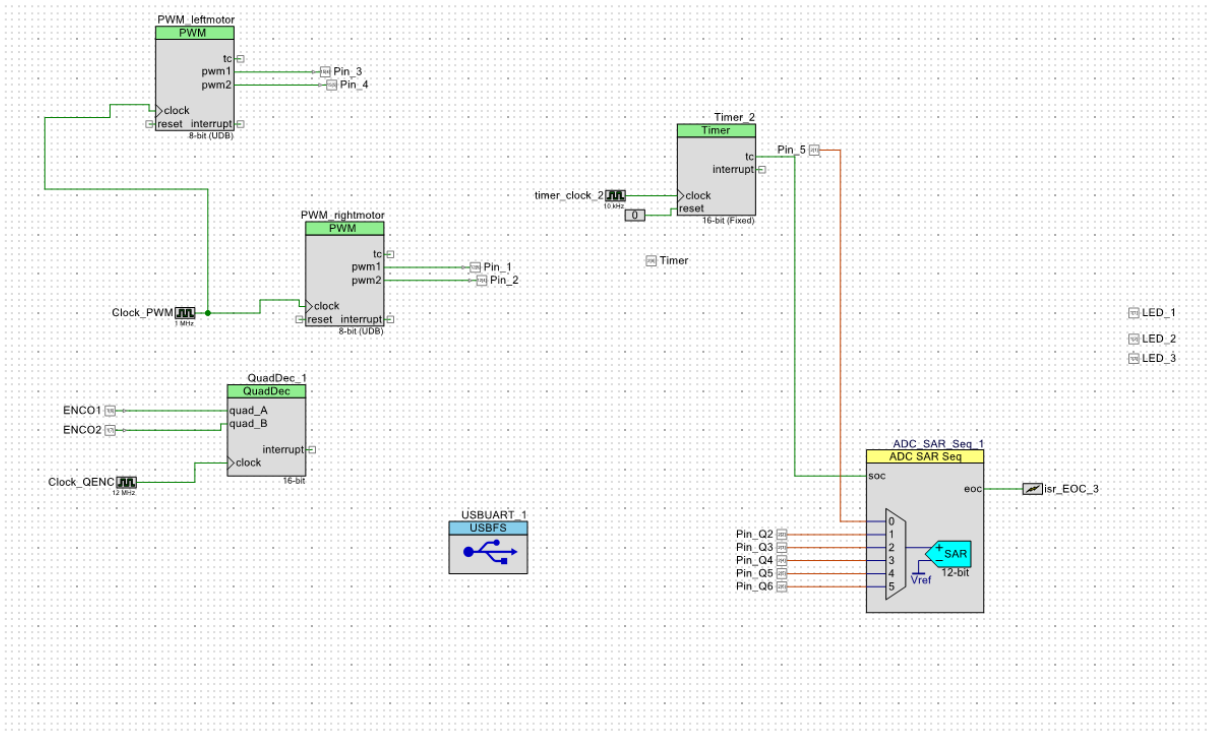

Fig.5. Schematic of PWM and QuadDec

Functions	PWM motor control

GoStraight()	PWM_leftmotor_WriteCompare1(0);
	PWM_leftmotor_WriteCompare2(97);
	PWM_rightmotor_WriteCompare1(90);
	PWM_rightmotor_WriteCompare2(0);

ArrangeToLeft()	PWM_leftmotor_WriteCompare1(0);
	PWM_leftmotor_WriteCompare2(leftDuty);
	PWM_rightmotor_WriteCompare1(startDuty);
	rightDuty = rightDuty + offset;
	PWM_rightmotor_WriteCompare2(rightDuty);
	
	(Note: offset = 15, startDuty = 90, leftDuty = 97, rightDuty = 0)

ArrangeToRight()	PWM_leftmotor_WriteCompare1(0+ offset);
	PWM_leftmotor_WriteCompare2(startDuty);
	PWM_rightmotor_WriteCompare1(startDuty + offset);
	PWM_rightmotor_WriteCompare2(rightDuty);

DoLeftTurn()	PWM_leftmotor_WriteCompare1(90);
	PWM_leftmotor_WriteCompare2(0);
	PWM_rightmotor_WriteCompare1(90);
	PWM_rightmotor_WriteCompare2(0);
	
	We are reversing the right wheel in this case for the left turn

DoRightTurn()	PWM_leftmotor_WriteCompare1(0);
	PWM_leftmotor_WriteCompare2(90);
	PWM_rightmotor_WriteCompare1(0);
	PWM_rightmotor_WriteCompare2(90);
	
	We are reversing the left wheel in this case for the right turn

Fig.6. Table for motor control

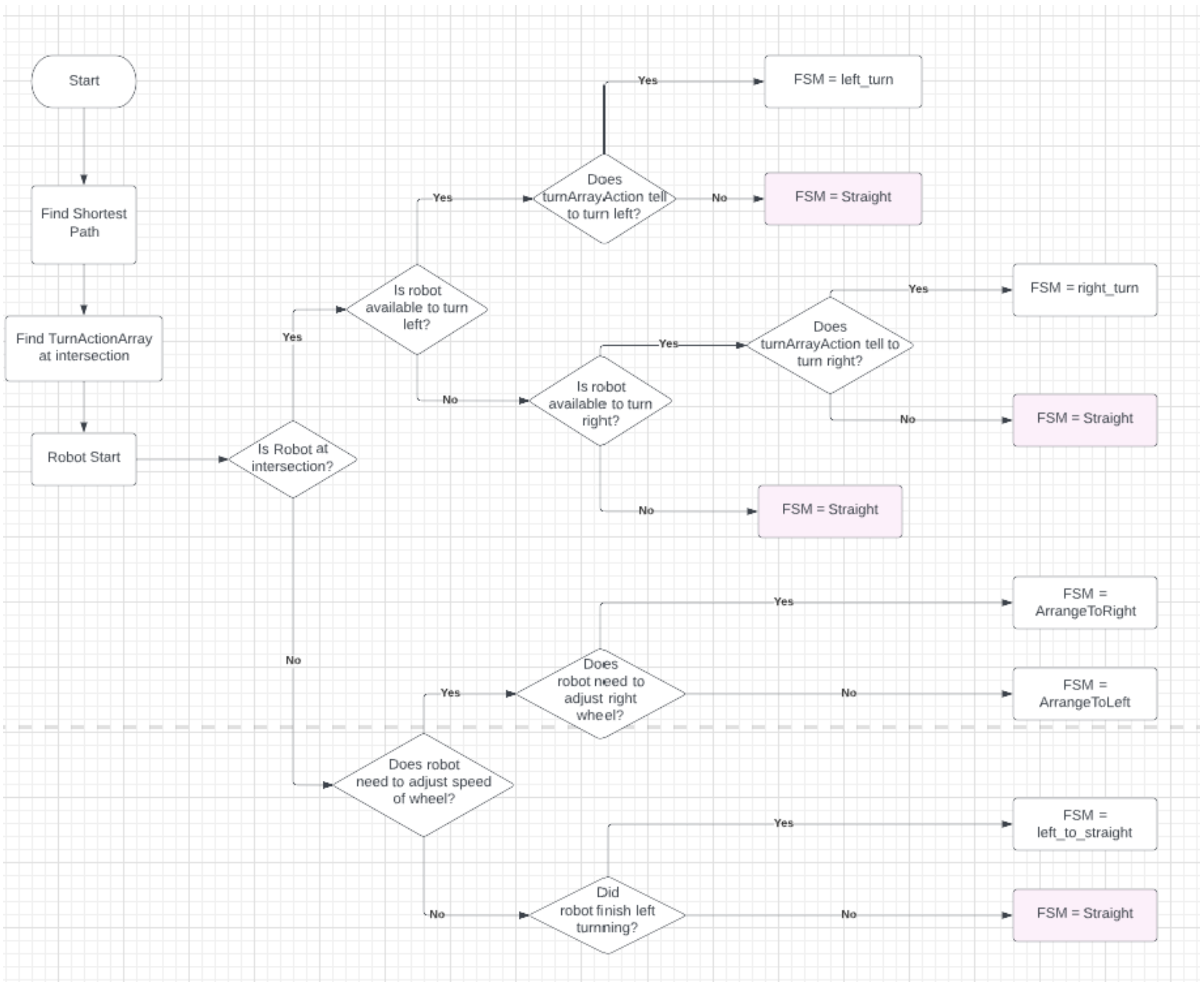
Start
Find Shortest Path
Find TurnActionArray at intersection
Robot Start
Is Robot at intersection?
Yes
No
Is robot available to turn left?
Yes
No
Does turnArrayAction tell to turn left?
Yes
No
FSM = left_turn
FSM = Straight
Is robot available to turn right?
Yes
No
Does turnArrayAction tell to turn right?
Yes
No
FSM = right_turn
FSM = Straight
FSM = Straight
Does robot need to adjust speed of wheel?
Yes
No
Does robot need to adjust right wheel?
Yes
No
FSM = ArrangeToRight
FSM = ArrangeToLeft
Did robot finish left turnning?
Yes
No
FSM = left_to_straight
FSM = Straight

Fig.7. FSM flow chart

Appendix B

Group member contributions

	Lab+ preparation	Filter and sensor design	PCB design	Shortest path finding algorithm	Robot implementation	Report
Francis Cho	✓	✓			✓	✓
Yu-gyeong Hong	✓			✓	✓	✓
Gayeon Kim	✓		✓		✓	✓
Xuhan Liu	✓			✓	✓	✓