Joshua Waugh, I-EN WU, Tuan Lam Minh

Zumo Robot Project Documentation

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Project Documentation

14 December 2018

|  |  |  |
| --- | --- | --- |
| Author  Title  Number of Pages  Date | Joshua Waugh, I-EN WU, Tuan Lam Minh  Zumo Robot Project Documentation  13 pages  14 December 2018 | |
| Degree | Bachelor of Engineering | |
| Degree Programme | Information Technology | |
| Professional Major | Smart Systems | |
| Instructors | Keijo Länsikunnas  Joseph Hotchkiss | |
|  | | |
| Keywords | |  |

Contents

[1 Introduction 1](#_Toc532581800)

[2 Materials and Software 1](#_Toc532581801)

[2.1 PSoC® Creator™ Integrated Design Environment (IDE) 1](#_Toc532581802)

[2.2 CY8CKIT-059 PSoC® 5LP Prototyping Kit 2](#_Toc532581803)

[2.3 Pololu Zumo Robot 2](#_Toc532581804)

[2.4 Message Queuing Telemetry Transport (MQTT) 3](#_Toc532581805)

[3 Implementation 4](#_Toc532581806)

[3.1 Sumo Wrestling 4](#_Toc532581807)

[3.2 Line Following Race 5](#_Toc532581808)

[3.3 The Maze 7](#_Toc532581809)

[3.3.1 The First Program 8](#_Toc532581810)

[3.3.2 The Second Program 9](#_Toc532581811)

[4 Results 10](#_Toc532581812)

[4.1 Line Following Race 10](#_Toc532581813)

[4.2 The Maze 10](#_Toc532581814)

[4.3 Sumo Wrestling 10](#_Toc532581815)

[5 Discussion 10](#_Toc532581816)

[5.1 Line Following Race 10](#_Toc532581817)

[5.2 The Maze 11](#_Toc532581818)

[5.3 Sumo Wrestling 12](#_Toc532581819)

[6 Conclusion 12](#_Toc532581820)

[References 13](#_Toc532581821)

Table of Figures

[Figure 1 1](#_Toc532581364)

[Figure 2 2](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581365)

[Figure 3 3](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581366)

[Figure 4 4](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581367)

[Figure 5 4](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581368)

[Figure 6 5](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581369)

[Figure 7 6](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581370)

[Figure 8 6](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581371)

[Figure 9 7](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581372)

[Figure 10 8](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581373)

[Figure 11 8](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581374)

[Figure 12 9](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581375)

[Figure 13 9](file:///C:\Users\User\Documents\GitHub\gittest\new%20Zumo%20documentation.docx#_Toc532581376)

# Introduction

This report details the Smart Systems course project for first year IT students at Metropolia University or Applied Science. The project involved programming a robot to complete three tasks: sumo wrestling, line following and maze navigation. Each member of the group working on this task had no experience with programming prior to this project, so it was approached largely as a learning exercise. This report also serves as a component of our final grade, and it too is a learning exercise for us who are new to academia.

# Materials and Software

## PSoC® Creator™ Integrated Design Environment (IDE)

PSoC (Programmable System-on-Chip) Creator is a program created by Cypress Semiconductor for writing programs for and compiling to PSoC hardware. This software was used for all code written during this project.

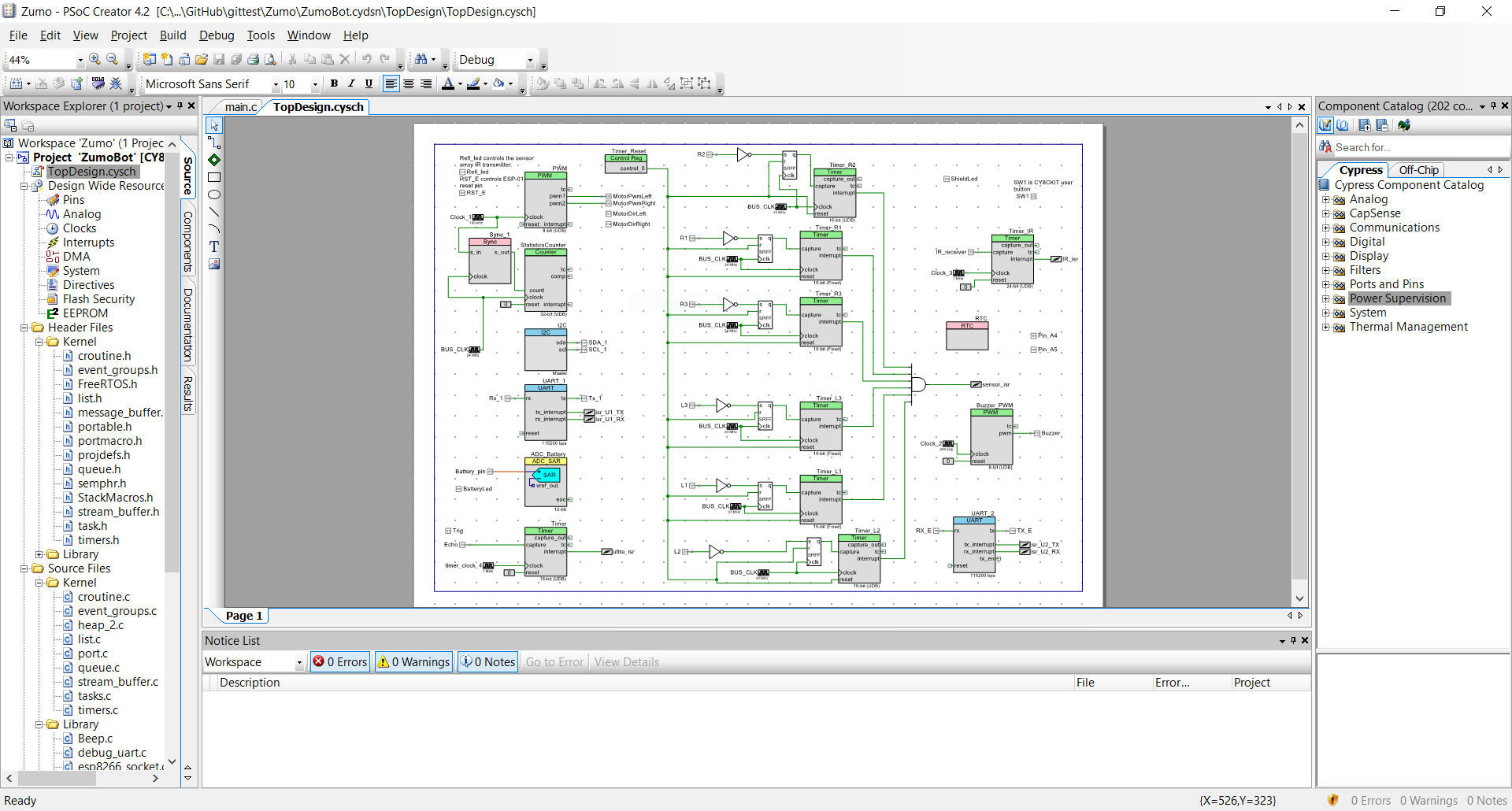


Figure 1

## CY8CKIT-059 PSoC® 5LP Prototyping Kit

The CY8CKIT-059 PSoC 5LP prototyping kit from Cypress Semiconductor was used to

control the Zumo robot. Located on the PSoC is a USB connector, a LED, a button to

reset the program and a button that can execute programmed commands. This programmable button was used extensively throughout the project and will be referred to as *the* button for the remainder of this document. Code was compiled and written to the PSoC by connecting it to a computer via USB and using PSoC Creator 4.2.

Figure 2

## Pololu Zumo Robot

The robot used for this project is the Pololu Zumo robot, a small chassis which can be connected to other hardware (e.g. the PSoC prototyping kit). Some of the components used throughout this project came pre-installed with the chassis, such as the two motors, six reflectance sensors and the accelerometer. The libraries for these components were provided to us by Pololu. Other components, such as the wi-fi module, IR receiver and ultrasonic sensors, were installed by our lecturers with libraries also provided by them. As the name suggests, this robot is intended to fight in a sumo ring of sorts, and as such also comes pre-installed with a large metallic shield.



Figure 3

## Message Queuing Telemetry Transport (MQTT)

MQTT was used to publish data from the robot to be collected by the lecturers for evaluation. A router acted as a broker between our robot and a computer set up to collect the data. The data sent was mostly timestamps, with the following being necessary for all projects:

* The time at which the robot reaches the starting line and waits for an infrared signal to begin (ready)
* The time at which the robot receives the IR signal and begins (start)
* The time at which the robot reaches its goal (stop)
* The difference between stop and start to find the time taken (time)

# Implementation

## Sumo Wrestling

Figure 4

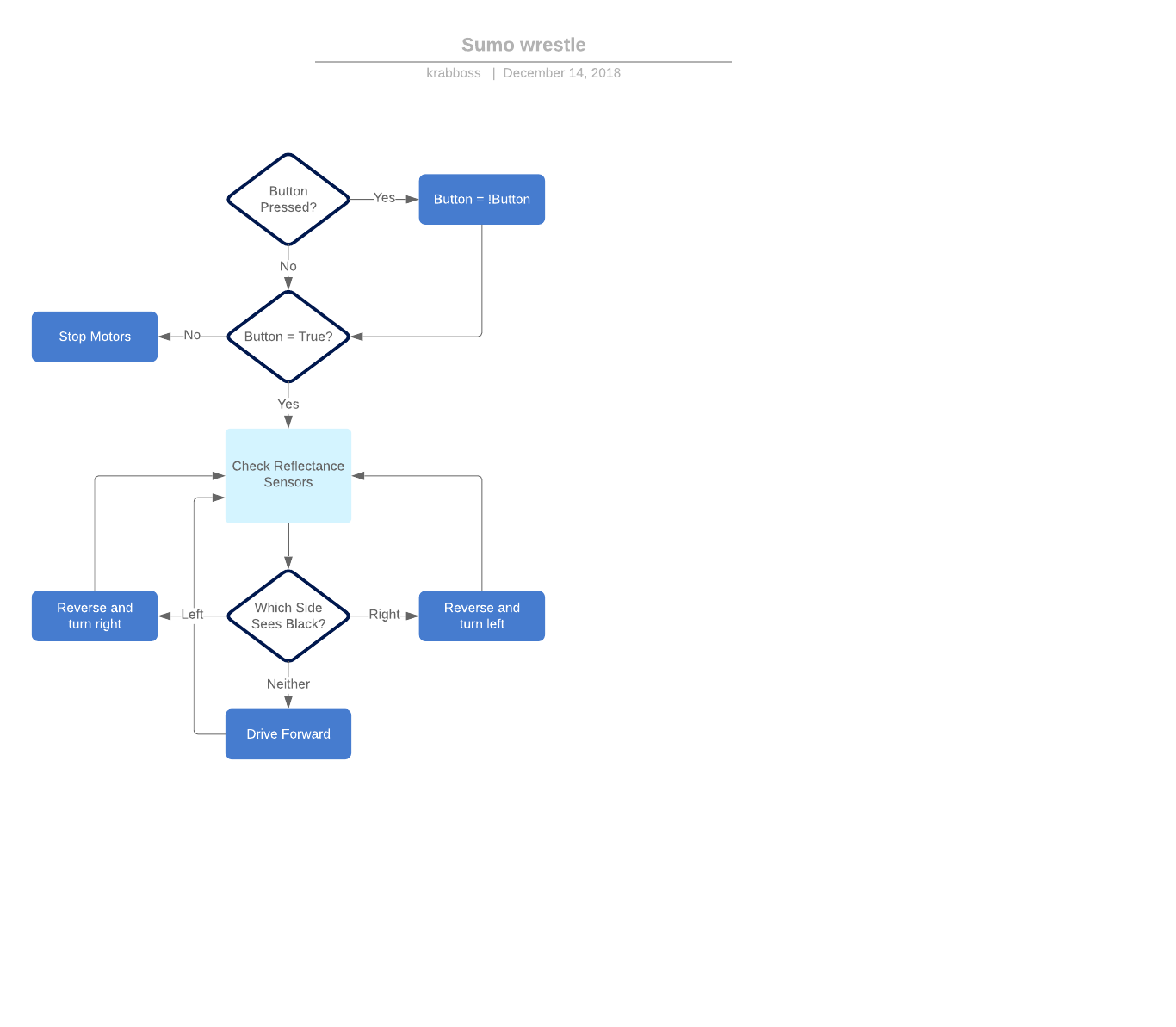
The arena for this task can be seen in Figure 4. The Sumo wrestling competition involves robots being placed at three of the lines perpendicular to the circular arena. Each robot is to drive to the edge of the arena after being powered on by the button on the PSoC. Here they wait until they are given an IR signal, at which point the battle begins. The approach used for this task is outlined in Figure 5.

Figure 5

Not shown in the figure are the accelerometer readings used to determine if the robot has been struck by an opponent. The angle of each hit is determined by taking the inverse tangent of the changes in acceleration recorded on the X and Y axes. Initially it was presumed that the robot should react when it has been struck by an opponent; however, it was decided this would not reliably improve the likelihood of winning. As such, the robot simply drives in a straight line until it sees a black line, where it then turns around and drives forward again. Timestamps are taken when the button is pressed at the beginning, when the IR signal is given, when a hit is registered, and when the button is pressed at the end. The difference between the end time and the time of the IR signal is taken to find the duration of our time in the ring.

## Line Following Race

The course for this task is shown in Figure 6. The starting point is the single horizontal line seen at the bottom right corner and the end is the lowest horizontal line on the bottom left corner. The line following race is a tournament between all teams to see whose robot is the fastest. Each race is between two robots on two identical 10-meter tracks. The race begins after the robots have approached the starting line and been given the infrared signal. Timestamps are to be recorded at the starting line and the finish line and the difference taken to find the completion time.

Figure 6

The line following algorithm used for the race, and every other project involving following a line, uses both analogue and digital readings from the robot’s six reflectance sensors. Analogue readings range from around 5000 while on white, to above 23000 while on black. Digital readings are 1 or 0, depending on whether the sensor is over a certain analogue threshold of black. The layout of the sensors can be seen in Figure 7. Firstly, a ratio is found by dividing the analogue value for the L1 sensor by the value for the R1 sensor. This ratio is applied to the motor that needs to be slowed down in any given situation. Secondly, digital readings for sensors 2 and 3 on both sides are checked for black. If they return a reading of 1, that side’s motor will be slowed further.



Figure 7

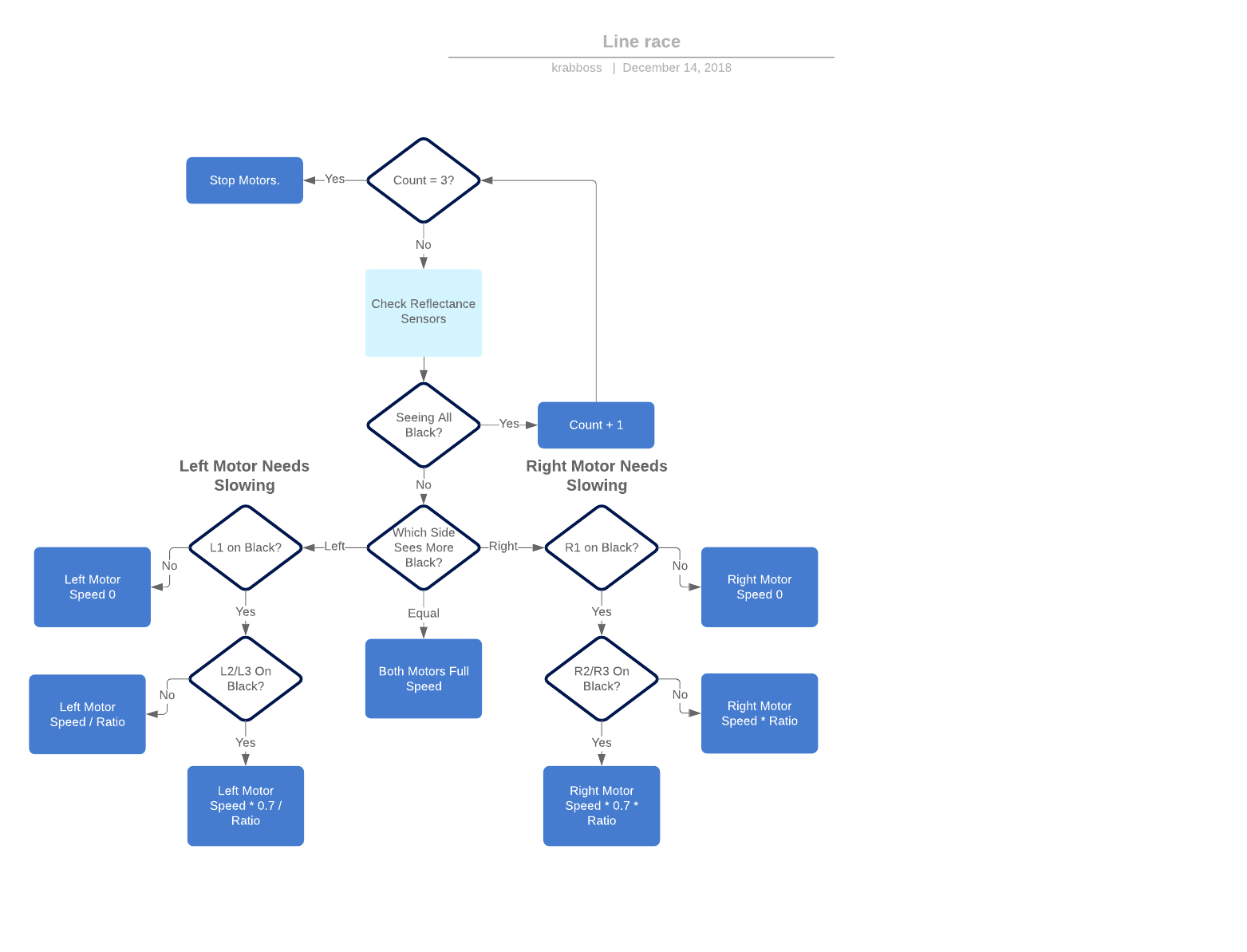
The necessary action is found through a series of if/else statements, is represented in Figure 8 below. The figure does not include the process for starting the robot with the button and IR inputs, as these are common to all projects and do not bear repeating.

Figure 8

## The Maze

Figure 9

The maze, seen in Figure 9, is a grid of 13 rows and seven columns. The starting point is the horizontal line under the grid and the end is the white space past the straight line on the other end. Placed on the maze are four wooden blocks placed randomly between rows three and ten. The obstacles are detected using the robot’s ultrasonic sensor. As with the others, this project requires taking timestamps at the beginning and the end of the maze. Two solutions were developed for this project. The first solution was abandoned after it was found to be insufficient when dealing with certain scenarios. Both programs have many similarities, however, which will be discussed here first.

The basis for both programs is deciding when the robot should turn and when it should drive ahead. In addition to turning from obstacles, the robot should also turn forwards toward the exit again after an obstacle has been avoided. If it must turn to avoid an obstacle, it should prefer turning towards the centre column as that is where the exit is. Knowing also that the grid is not perfectly rectangular and that obstacles are not placed on the final 3 rows, it is wise to turn and drive to the centre column once the robot has finished running the gauntlet.

In service of this, the programs apply a co-ordinate system to the grid to track the position of the robot. The direction of the robot is stored in a variable and is used to update the position of the robot whenever an intersection is encountered. For example, if the robot is driving sideways to the left, then that means the robot’s X position should decrement and its Y position should remain unchanged. The direction is initialized as 0 and decrements and increments on left and right turns respectively. It was assumed that going backwards would not be necessary, so the range for direction is only -1 to 1.

Between intersections, both programs use the same line following code shown previously in Figure 8.

### The First Program

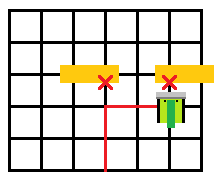
The first program would maneuver around the blocks by travelling two intersections away. The idea was this would ensure the block was avoided, as each block is slightly less than two intersections wide. However, if a second block was encountered after this, there was no easy solution. The robot could be made to travel one intersection to the left or right and hope to find a path through but doing so would defeat the purpose of moving by two in the first place. Alternatively, the robot could travel four columns to the opposite side of the grid, but by then the robot has travelled six lines when it could have travelled over just one. Figure 10 illustrates the problem of this movement system.

Figure 10

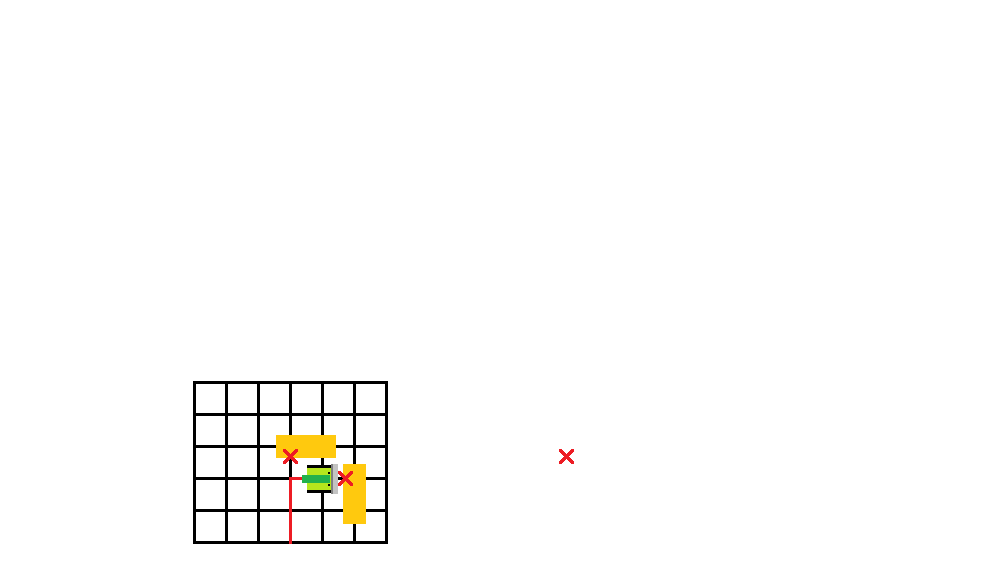


Figure 11

Moving by two also made encounters with vertical blocks awkward, as it could force the robot into an odd column. It became impractical to continuously program around exceptional situations which needed not be exceptional in the first place.

### The Second Program

Figure 12

{1, 1, 1, 0, 0, 0, 1, 1, 1},

{1, 1, 0, 0, 0, 0, 0, 1, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 0, 0, 0, 0, 0, 0, 1},

{1, 1, 1, 1, 0, 1, 1, 1, 1},

For the second program, an array was created to act as a map of the grid. The array is two rows and two columns larger than the actual grid, with the extra numbers acting as boundaries (see Figure 12). Each intersection is initialized as the number ‘0’, indicating that it is free to move through. Whenever an obstacle is detected, the position ahead of the robot is flipped to ‘1’.

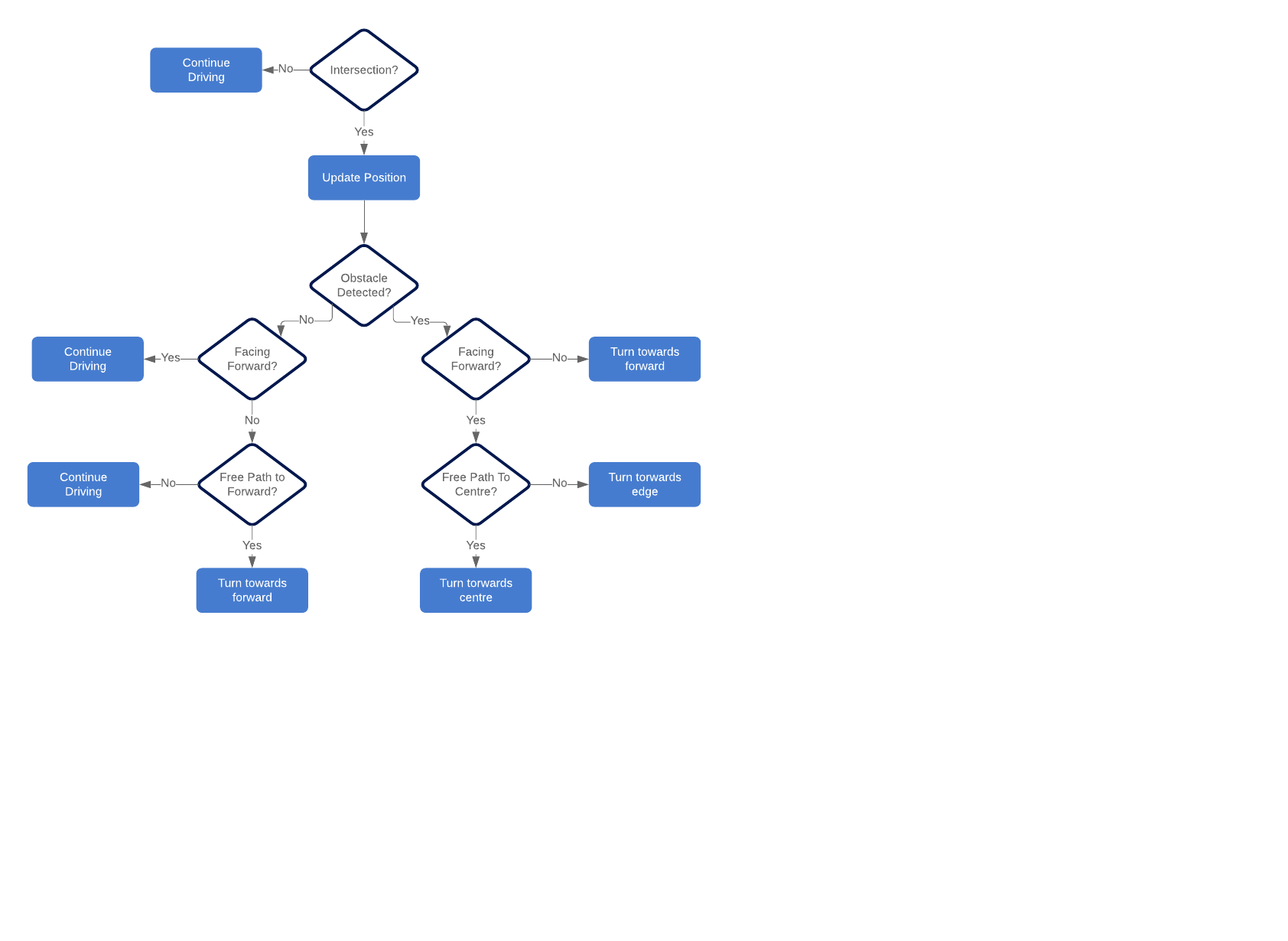
When facing any section marked as ‘1’, a path finding function checks the surrounding intersections to find one marked ‘0’. It is important to realise what it means if the robot now turns and encounters another obstacle. Refer again to Figure 11 and notice the intersection diagonal to the robot is empty but nevertheless inaccessible. This intersection is also set to ‘1’ so that the path finding function has a truer sense of its options. The process of the second program can be seen in Figure 13. This figure does not include the line following logic or button press loop already shown in earlier figures.

Figure 13

# Results

The results of this project are somewhat binary, as the robot could either succeed or not succeed in completing the challenges outlined. For further analysis, refer to the Discussion chapter.

## Line Following Race

The line following race was the first event of the day. The robot performed well, defeating all other teams and winning the tournament. Our fastest recorded time was 30.074 seconds.

## The Maze

The maze was the second event of the day. Unfortunately, our robot was not able to complete this challenge. It failed to avoid one of the blocks and pushed through it instead.

Fortunately, it did not lose the line and managed to continue tracking its location and printing the co-ordinates as it made its way to the exit.

## Sumo Wrestling

The final event was the sumo wrestling competition. The robot was eliminated in the first round but succeeded in passing the requirements of the project.

# Discussion

## Line Following Race

As shown in the implementation chapter, our robot used a list of 7 if/else statements to follow lines. Another possibility for this project was to use a proportional–integral–derivative (PID) controller. Using a PID controller was initially considered for this project, despite it being much more mathematics intensive and requiring much fine tuning. It was not pursued when, during testing in the weeks leading up to the final competition, our if/else algorithm outperformed those who were using the PID controller. By our reckoning, the PID controller appeared to overly slow the motors when driving on straight lines, resulting in sub-optimal completon times.

The track used for testing was about half the length of the final competition track. Our robot completed the test track at 13.5 seconds, roughly one second faster than the fastest PID controlled robot in the class. On the competition track, our robot was only 18 milliseconds faster than the same PID controlled robot. It was clear on the longer track that our robot was significantly slower on corners. The if/else algorithm used meant that the speed of one of the motors was zero on every corner, while a PID controller could find a more optimal solution.

It is likely that a hybrid solution is ideal, if the goal is just to design something that is as fast as possible. Using a PID controller at all would have likely been more fruitful than winning the race, just as a learning exercise. Either way, it is regrettable that a PID controller was not used.

## The Maze

As was mentioned in the results, the robot did not complete the maze without incident. It is not clear where the point of failure happened without further testing. It is possible that the robot became too close to the block it was meant to detect and could not see it. The ultrasonic sensor is somewhat inaccurate as distances fall below 10 centimeters. It is also possible that there was a bug in the code that prioritized the wrong action in this scenario.

The final layout of the blocks on the competition day was relatively simple. In truth, the first program written would have solved the maze without incident, as most teams were successful with a move-by-two method.

## Sumo Wrestling

It was assumed going into this competition that winning would come down to mostly chance. This was not entirely true. Some effort could be made to increase the chances of victory. Most teams chose to control their robot similarly to ours, just with additional options. The teams who found success did in fact react to their opponents somewhat, rather than relying on the eventuality of eliminations occurring.

# Conclusion

This project served as our introduction to C and embedded system programming. Each week brought new roadblocks that improved our knowledge as we solved them. The maze in particular required becoming intimately familiar with writing functions and logical operations. Our work for the entirety of this course is catalogued in one long main.c file, and our code from the initial weeks is certainly not on the same level as our work on the final three projects this report pertains to. Despite the rocky finish, we are proud of the achievements we made during this project and feel prepared to take on bigger projects in the future.

References

Pololu - Zumo Robot for Arduino, v1.2 (Assembled with 75:1 HP Motors) [WWW Document], n.d. URL <https://www.pololu.com/product/2510> (accessed 18.11.18).