 *Final mark awarded*

FACULTY OF COMPUTING, ENGINEERING and SCIENCE

Assessment Cover Sheet and Feedback Form 2017/18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Module Code:**  NG4D900 | **Module Title:**  MEng Major Group Project | | **Lecturer:**  Guoping Liu | |
| **Assignment No:**  1 | **No. of pages in total including this page:** | | **Maximum Word Count:** 15000 | |
| **Assignment Title: Report Tasks:** | | | | |
| **Date Set:**  27/09/2017 | | **Submission Date:**  25/04/2018 | | **Feedback Date:**  09/05/2018 |

|  |
| --- |
| **Section A: Record of Submission** |
| **Record of Submission and Plagiarism Declaration**  I declare that this assignment is my own work and that the sources of information and material I have used (including the internet) have been fully identified and properly  acknowledged as required in the referencing guidelines provided.  **Fit to Sit Policy**  The University operates a fit to sit policy whereby you, in submitting or presenting yourself for any assessments, are declaring that you are fit to sit the assessment. You cannot subsequently claim that your performance in that assessment was affected by extenuating circumstances.  **Student Number:**  You are required to acknowledge that you have read the above statements by writing your student number(s) above.  (If this is a group assignment, please provide the student numbers of **ALL** group members)  **Details of Submission**   * IT IS YOUR RESPONSIBILITY TO KEEP A RECORD OF ALL WORK SUBMITTED. * Work should be submitted as detailed in your student handbook. You are responsible for checking the method of submission. * **Late Submission** – Work must be submitted by the submission date. If you fail to do this, you will be allowed a further five working days to submit the work but the work will be awarded a maximum mark of 40%. If you fail to submit work within five working days of the submission date, you will be deemed to have failed this assessment which will be given a mark of 0%. However see extenuating circumstances below.   **Extenuating Circumstances:** if there are any exceptional circumstances that may have affected your ability to undertake or submit this assignment, make sure you contact your Advice Shop and also see either<http://cesstudents.southwales.ac.uk/Ext_circs/>(Trefforest) or <http://glyntaffcampus.southwales.ac.uk/advice_shop/Extenuating_Circumstances/>(Glyntaff) |

|  |  |
| --- | --- |
| **Section B : Marking and Assessment** | |
| This assignment will be marked out of 100%  This assignment contributes to 100% of the total module marks. This assignment is bonded / non- bonded. Details : | It is estimated that you should spend approximately  20 hours on this assignment. |

|  |  |  |
| --- | --- | --- |
| **Learning Outcomes** |  | |
| **This assignment addresses the following learning outcome(s) of the module:**   1. Work effectively as part of a multi-disciplinary team in developing and implementing a solution to a complex, industrially relevant, major technical problem, communicating effectively with a team members and, if appropriate, personnel from the sponsoring industry. 2. Demonstrate a critical and conceptual understanding of all aspects of the work, and contribute to both the production of a detailed technical report, and a concise technical presentation. 3. Demonstrate a critical and conceptual understanding of all aspects of the work, and contribute to both the production of a detailed technical report, and a concise technical presentation. | | |
| **Marking Scheme** | **Marks**  **Available** | **Marks**  **Awarded** |
| **1.** See page 5 |  |  |
| **2.** |  |  |
| **3.** |  |  |
| etc. |  |  |
|  |  |  |

|  |  |
| --- | --- |
| **Assessment Criteria** | |
| **Performance Level** | **Criteria** |
| Fail (<40%) |  |
| 3rd Class / PASS (40%-49%) |  |
| Lower 2nd Class / PASS (50%-59%) |  |
| Upper 2nd Class / MERIT (60%-69%) |  |
| 1st Class / DISTINCTION (70% +) |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Section C : Marker’s Feedback** | | | |
| **Lecturer’s Comments:** | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| ……………………………………………………………………………………………………………… | | | |
| **Areas to concentrate on next time:** | | | |
| Report structure | Research | Content | Team work |
| Referencing | Presentation |  |  |
| **Lecturer’s signature:** | | **Date:** | **Mark awarded:** |
| **All marks are subject to confirmation by the Assessment Boards** | | | |

|  |  |  |
| --- | --- | --- |
| **Task distribution per student** | | |
| **Student ID** | **Name** | **Section/sub-section** |
| **13013521** | **Thomas Brown** |  |
| **15021106** | **Kayin Templar** |  |
| **14031329** | **Miguel Santos** |  |

Table of contents

[1. Introduction 4](#_Toc513208030)

[1.1. Overview 4](#_Toc513208031)

[1.2. Project goals 4](#_Toc513208032)

[1.3. The competition 4](#_Toc513208033)

[1.3.1. Regulations 4](#_Toc513208034)

[1.3.2. The race track 5](#_Toc513208035)

[1.3.3. Results from 2017/18 8](#_Toc513208036)

[2. Features 9](#_Toc513208037)

[3. Design 10](#_Toc513208038)

[3.1. Mechanical 10](#_Toc513208039)

[3.2. Electrical 11](#_Toc513208040)

[3.2.1. Provided Designs 11](#_Toc513208041)

[3.2.2. Sensor Bar 11](#_Toc513208042)

[3.2.3. Power Board 13](#_Toc513208043)

[3.3. Speed Control and Feedback 16](#_Toc513208044)

[3.3.1. Hall Effect Sensor 17](#_Toc513208045)

[3.3.2. Accelerometer 19](#_Toc513208046)

[3.4. Overall Electronic Design 20](#_Toc513208047)

[3.5. Control theory 21](#_Toc513208048)

[3.6. Software 22](#_Toc513208049)

[3.6.1. Program structure 22](#_Toc513208050)

[3.6.2. Backend 22](#_Toc513208051)

[3.6.3. Frontend 24](#_Toc513208052)

[4. Implementation 25](#_Toc513208053)

[4.1. Mechanical 25](#_Toc513208054)

[4.2. Electrical 26](#_Toc513208055)

[4.2.1. Provided Circuits 26](#_Toc513208056)

[4.2.1. Hall Effect circuits 27](#_Toc513208057)

[4.2.1. Peripheral Board 27](#_Toc513208058)

[4.3. Software 28](#_Toc513208059)

[4.3.1. Backend 28](#_Toc513208060)

[4.3.1.1. RTOS 28](#_Toc513208061)

[4.3.1.2. Peripheral drivers 28](#_Toc513208062)

[4.3.1.3. Communication protocol 34](#_Toc513208063)

[4.3.2. Frontend 36](#_Toc513208064)

[4.3.2.1. User application – car controller 36](#_Toc513208065)

[4.3.2.2. User interactivity – command shell and Android App 39](#_Toc513208066)

[5. Testing 44](#_Toc513208067)

[5.1. Mechanical 44](#_Toc513208068)

[5.2. Electrical 44](#_Toc513208069)

[5.2.1. Sensor board 44](#_Toc513208070)

[5.2.2. Power board 45](#_Toc513208071)

[5.2.3. Hall Effect Sensors 46](#_Toc513208072)

[5.2.4. Peripheral board 47](#_Toc513208073)

[5.3. Software 48](#_Toc513208074)

[6. Performance 49](#_Toc513208075)

[7. Problems and solutions 50](#_Toc513208076)

[8. Next year proposals 51](#_Toc513208077)

[9. Discussions 53](#_Toc513208078)

[10. Conclusion 54](#_Toc513208079)

[11. References 55](#_Toc513208080)

[Appendices 1](#_Toc513208081)

[Appendix A: 1](#_Toc513208082)

[Appendix B: 2](#_Toc513208083)

[Appendix C: 3](#_Toc513208084)

List of tables

[Table 3‑1 - Software: the backend structure 23](#_Toc513208142)

[Table 3‑2 - Software: the frontend structure 24](#_Toc513208143)

[Table 4‑1 - List of supported commands 40](#_Toc513208144)

[Table 5‑1 - Software testing methodology 48](#_Toc513208145)

List of figures

[Figure 1‑1 - Track piece: Right angle curve 5](#_Toc513208146)

[Figure 1‑2 - Track piece: T600 5](#_Toc513208147)

[Figure 1‑3 - Track piece: T300 5](#_Toc513208148)

[Figure 1‑4 - Track piece: T250 6](#_Toc513208149)

[Figure 1‑5 - Track piece: T150 6](#_Toc513208150)

[Figure 1‑6 - Track piece: R600 6](#_Toc513208151)

[Figure 1‑7 - Track piece: R450 6](#_Toc513208152)

[Figure 1‑8 - Track piece: Right lane change 7](#_Toc513208153)

[Figure 1‑9 - Track piece: Left lane change 7](#_Toc513208154)

[Figure 1‑10 - Track piece: Slope 7](#_Toc513208155)

[Figure 3‑1 - Credit www.electronics-tutorials.ws 17](#_Toc513208156)

[Figure 4‑1 - Implementation of the controller for the car 32](#_Toc513208157)

[Figure 6‑1 - Performance: memory usage 49](file:///E:\Users\Miguel\Dropbox2\Dropbox\My-Uni-Assignments\Year%204\NG4D900%20-%20MEng%20Group%20Project\Renesas-MCU-Car-Rally-2018-USW\Project%20Management\Report\NG4D900-MajorGroupProject-2018-Report.docx#_Toc513208158)

List of abbreviations

|  |  |
| --- | --- |
| m | Metre |
| mm | Millimetre |
| g | Gram |
| s | Second |
| PID | Proportional – Integral – Derivative |
| RTOS | Real Time Operating System |
| MIDI | Musical Instrument Digital Interface |
| TCP | Transmission Control Protocol |
| IP | Internet Protocol |
| UART | Universal Asynchronous Receiver-Transmitter |
| IR | Infrared |
| MCU | Microcontroller Unit |
| LED | Light Emitting Diode |
| DC | Direct Current |
| PWM | Pulse Width Modulation |
| FET | Field Effect Transistor |
| IC | Integrated Circuit |
| I/O | Input / Output |
| RC | Remote Control |
| RAM | Random Access Memory |
| ROM | Read Only Memory |
| GPIO | General Purpose Input / Output |
| IDE | Integrated Development Environment |
| API | Application Programming Interface |
| FSM | Finite State Machine |

# Introduction

## Overview

Every year the University of South Wales participates in an organised competition for students all across Europe by Renesas through partnership with the university.

In this competition, each team is tasked to build a racing car controlled with a microcontroller made from either an official kit provided by Renesas, or a custom built car. The winning team receives a certain prize, including the teams in second and third place.

This report will cover the progress of the work from the perspective of a regular academic group project, as well as a project with the competition as the end goal.

## Project goals

From the academic standpoint, this project is an attempt and execution of developing a car that is:

1. Better than last year’s (2016/2017), in terms of software and most importantly hardware;
2. Smarter in regards to its program algorithm through the use of “pseudo” machine learning

In addition to building a new car, another project goal was to actually improve last year’s code on the same old hardware. This was to allow the development of the software before the kit’s hardware could arrive. This way, it would be possible to just upload the code into the new car once it was built.

## The competition

### Regulations

There are many competition regulations that must be followed in order to prevent team disqualification. The main regulations that are not straightforward/obvious are:

1. The autonomous car must be controlled using a Renesas microcontroller, specifically the RX62G/RX62T;
2. Adhesive materials are not allowed to be used on the wheels – would provide grip, thus why some students would prefer it;
3. The maximum dimensions of the car are 300 mm wide and 150 mm tall;
4. Any material being used in the car’s hardware that might affect and/or destroy the track is strictly prohibited;
5. All cars must be subjected to inspection prior to the race. If the car does not pass this test the team is immediately disqualified even before they get the chance to race;
6. After inspection, the team must not alter in any way the shape or form of the car, including the batteries and tyres. Software modification is excluded from this rule;
7. Adding “hard-coded” data regarding the track’s specifications into the microcontroller’s memory prior to the competition IS allowed;
8. Starting the race before the signal results in disqualification;
9. Obstruction of other machines will also lead to complete disqualification of the team.

### The race track

The track is composed of lustreless acrylic pieces that are 300 mm wide and 30 mm thick. Each piece is supported by plywood which provides 27mm of extra height. In addition, each track component fits with the next which might have a different length, in fact, there are 10 types of track pieces, as shown below (all dimensions are in mm - millimetres):

* **Track type 1 – Right angle curve**

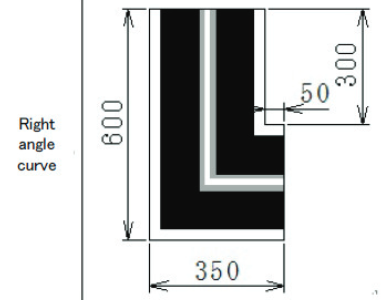


Figure 1‑1 - Track piece: Right angle curve

* **Track type 2 – T600**

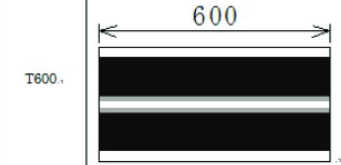


Figure 1‑2 - Track piece: T600

* **Track type 3 – T300**

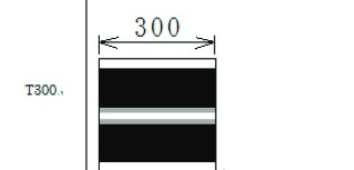


Figure 1‑3 - Track piece: T300

* **Track type 4 – T250**

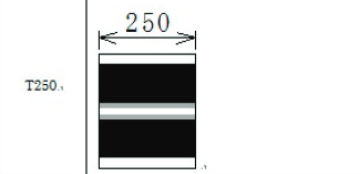


Figure 1‑4 - Track piece: T250

* **Track type 5 – T150**

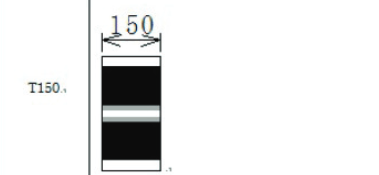


Figure 1‑5 - Track piece: T150

* **Track type 6 – R600**

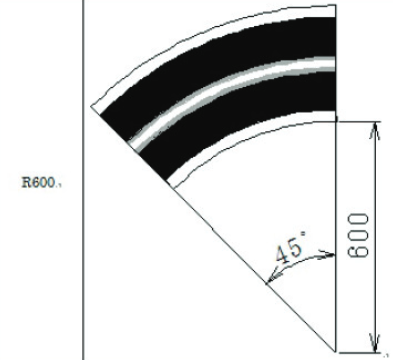


Figure 1‑6 - Track piece: R600

* **Track type 7 – R450**

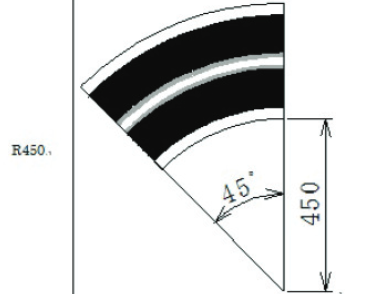


Figure 1‑7 - Track piece: R450

* **Track type 8 – Right lane change**



Figure 1‑8 - Track piece: Right lane change

* **Track type 9 – Left lane change**

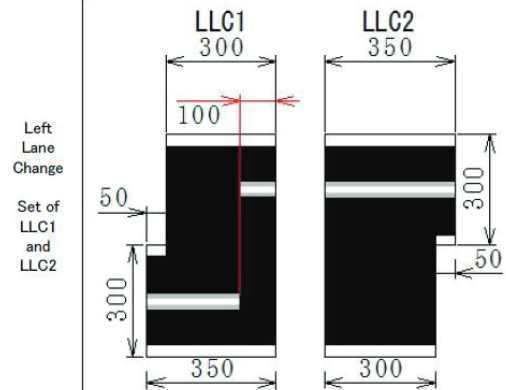


Figure 1‑9 - Track piece: Left lane change

* **Track type 10 – Slope**

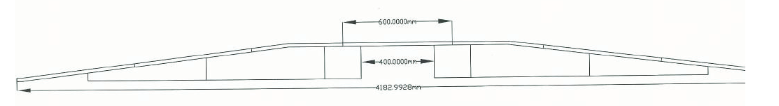


Figure 1‑10 - Track piece: Slope

It was fortunate to have a copy of these track pieces manufactured last year. The development process wouldn’t have gone as far as it did if there was no track to test the car on. Besides, having an improvised track might negatively impact the end result since the real track is completely different in terms of its specifications in comparison with the improvised version.

### Results from 2017/18

Despite all of the promises and potentials for this project, it was extremely unfortunate for the USW team in this year of 2017/18 to have entirely missed the competition. This was due to complications in the transportation (flight between countries Netherlands (Amsterdam) – Germany (Nuremberg)). Thankfully, a USW professor managed to make the recordings of the competition which allowed us to benchmark the winner’s performance against our test data.

(The results were also published after the competition, however, the short demonstration videos provided by Renesas do not favour our intent of measuring an approximated value of the velocity and acceleration of the winner’s car.)

The results were (in order):

1. Romania
   1. University: Gheorge Asachi Technical University os Iasi
   2. Penalties: 0
   3. Time: 01 minutes 35.25 seconds
2. Romania
   1. University: University of Craiova
   2. Penalties: 0
   3. Time: 01 minutes 55.3 seconds
3. Germany
   1. University: Harz University
   2. Penalties: 0
   3. Time: 02 minutes 01 seconds

# Features

Several features were implemented in the car in order to reach the goals of the project. In order to achieve the first goal – build a better car from the hardware and software perspective, the features were implemented:

1. **Mechanical**
   1. Lower the height of the car to provide stability;
   2. Redistribute the centre of gravity by laying the batteries flat at the back of the car and by using only one structural deck for accommodating the electronics instead of one;
   3. Increase the length of the sensor bar to allow the car to detect the line at a greater distance.
2. **Electrical**
   1. In addition to the speed sensor (Hall-Effect sensor), use an accelerometer to provide additional data regarding the motion of the car, specifically the momentum vector.
3. **Software**
   1. Use RTOS in order to have accurate control of the tasks being run in the program;
   2. Properly implement PID controller with Integral coefficient and Integral Windup.

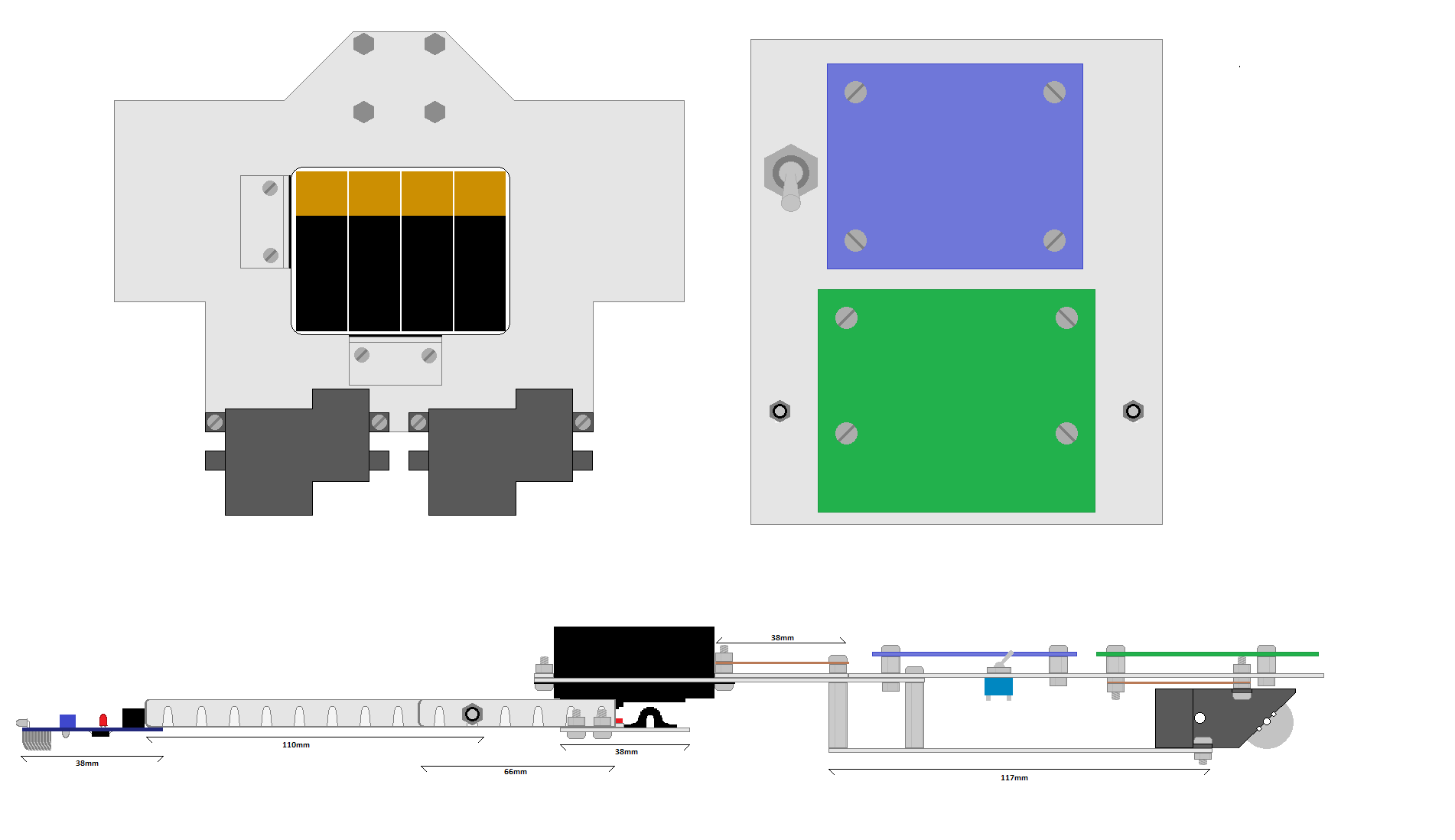
The two last project goals – improved/smarter hardware and software, were achieved through the implementation of the features:

1. **Electrical**
   1. Designed small board which accommodates a resistor network for measuring the voltage of the batteries, the accelerometer and a Bluetooth module (HC06) in order to communicate wirelessly with the car in real time;
   2. Hall Effect speed sensor (as opposed to an IR Encoder speed sensor);
2. **Software**
   1. Full-duplex communication through UART;
   2. Layered communication protocol that uses UART as the Data Link layer. Robustness and transparency (seamless) were also features designed into this protocol;
   3. Interactive real-time communication through a command line prompt and/or Android application;
   4. Intelligent and dynamic car control algorithms. These modes allow the car to use a set of different rules during the race. In short, the modes are: Basic, Advanced and Smart.

These features will be discussed in detail in the following report sections **3. Design** and **4. Implementation**.

# Design

## Mechanical

In order to maximise the performance of the car, the mechanical design must be changed to give this car an edge over the default design. A problem that other cars have is a high centre of mass, which means that they have to take corners at a slower speed to avoid going out. Below is the mechanical design of this car. There are no wheels shown in this design to make it easier to see everything.

This design has the MCU board and Power board on the same level (top right), as opposed to the provided design that has them on two separate levels. This requires extending the length of the car by 30mm but allows the car to be much less tall. This top-level plate is supported by the front pillars and the gearboxes. Having the power board on the top level should help to sort out some of the cooling issues present in other cars.

The servo board is attached to two arms extending from the front axle much like the original design, but in this design the middle of the arms are secured by nylocs, which allow free movement of the arm without it becoming loose.

The top left of this diagram shows the bottom level. The battery pack (containing eight AA batteries) is supported by two small plates with Velcro strips. This should sufficiently stop it becoming loose while the car moves, while still making it easy to remove the battery pack for charging

The switch has been repositioned from the area behind the servo to make room for the peripheral Veroboard. The makes all circuits easily accessible except for the Hall Effect circuits, which have to be positioned above the gearboxes.

The gearboxes are both positioned with half of the box hanging off the back of the car. This helps two-fold; to reduce the size of the bottom level, and allowing more air to reach the motors to cool them.

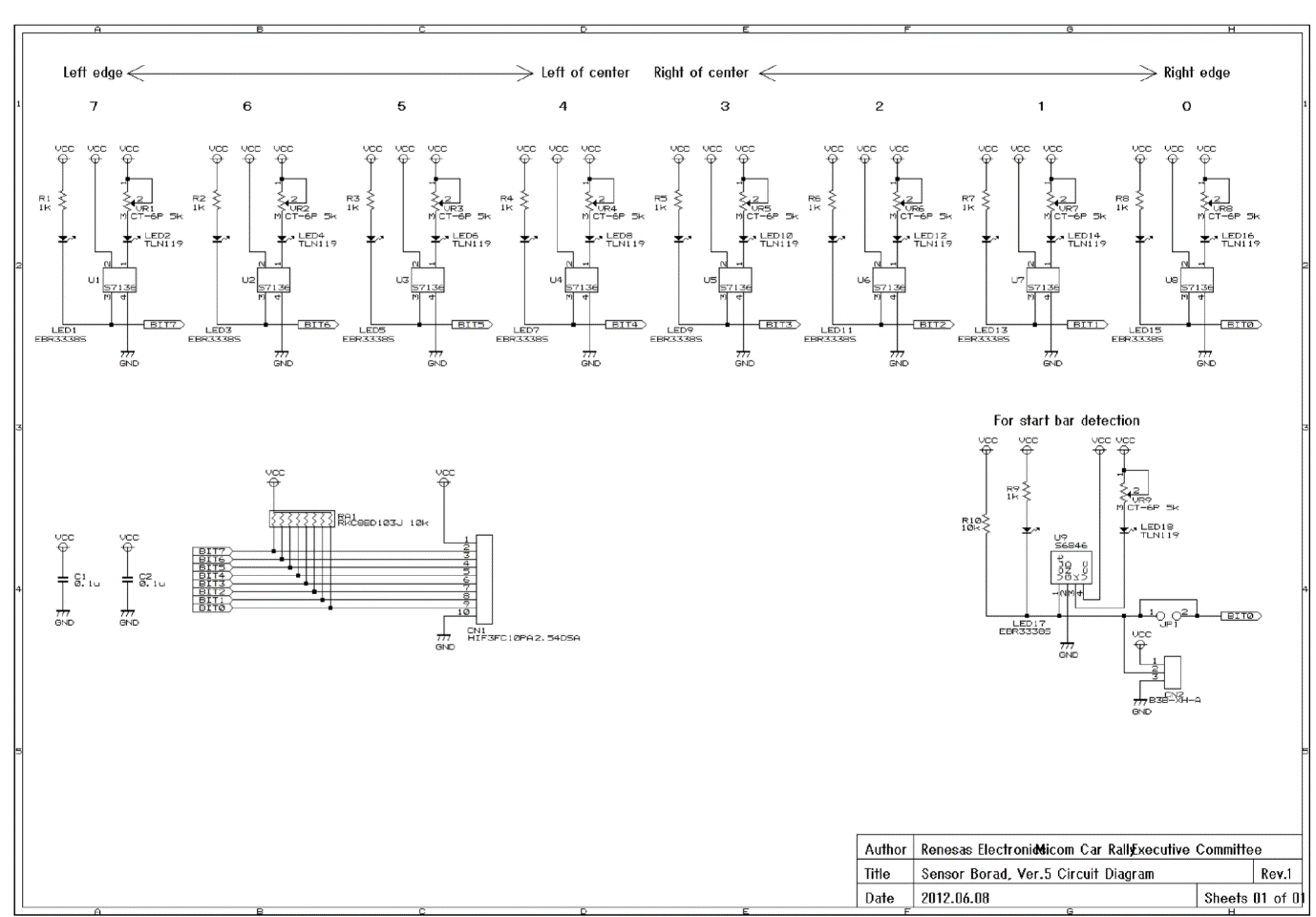
## Electrical

### Provided Designs

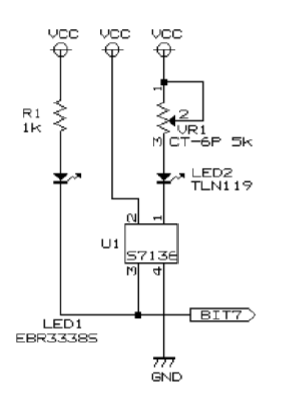
Three boards are included in the initial project package, the IR LED sensor bar, the Power board, and the MCU board. The MCU board arrives soldered however the others must be soldered. This section will explain the function of these boards, as they must be understood in case they develop a fault, and for the designing of added peripherals.

### Sensor Bar

The first of the provided circuit designs to be discussed here is the sensor bar. The purpose of this circuit is to detect the white line on the track, and send this data to the MCU board so that the car can be controlled.



Above is the schematic for the sensor bar. As each of the LED-IR sensor sections are identical, one of them can be looked at as an example.

The left most LED is a simple unicolour red LED in series with a 1k resistor. This LED has a forward voltage of 1.7V which means the current flow when on is. This LED will only be on when pin 3 of U1 goes low.

LED2 is the infrared LED which is in series with a potentiometer. As the resistance of the potentiometer is varied, the current supply, and thus intensity of light emitted varies.

Pin 1 on U1 is a cathode that sinks the current from the IR LED. Pin 2 is its power supply and Pin 4 is the ground. When U1 detects light, pin 3 goes low, turning LED1 on.

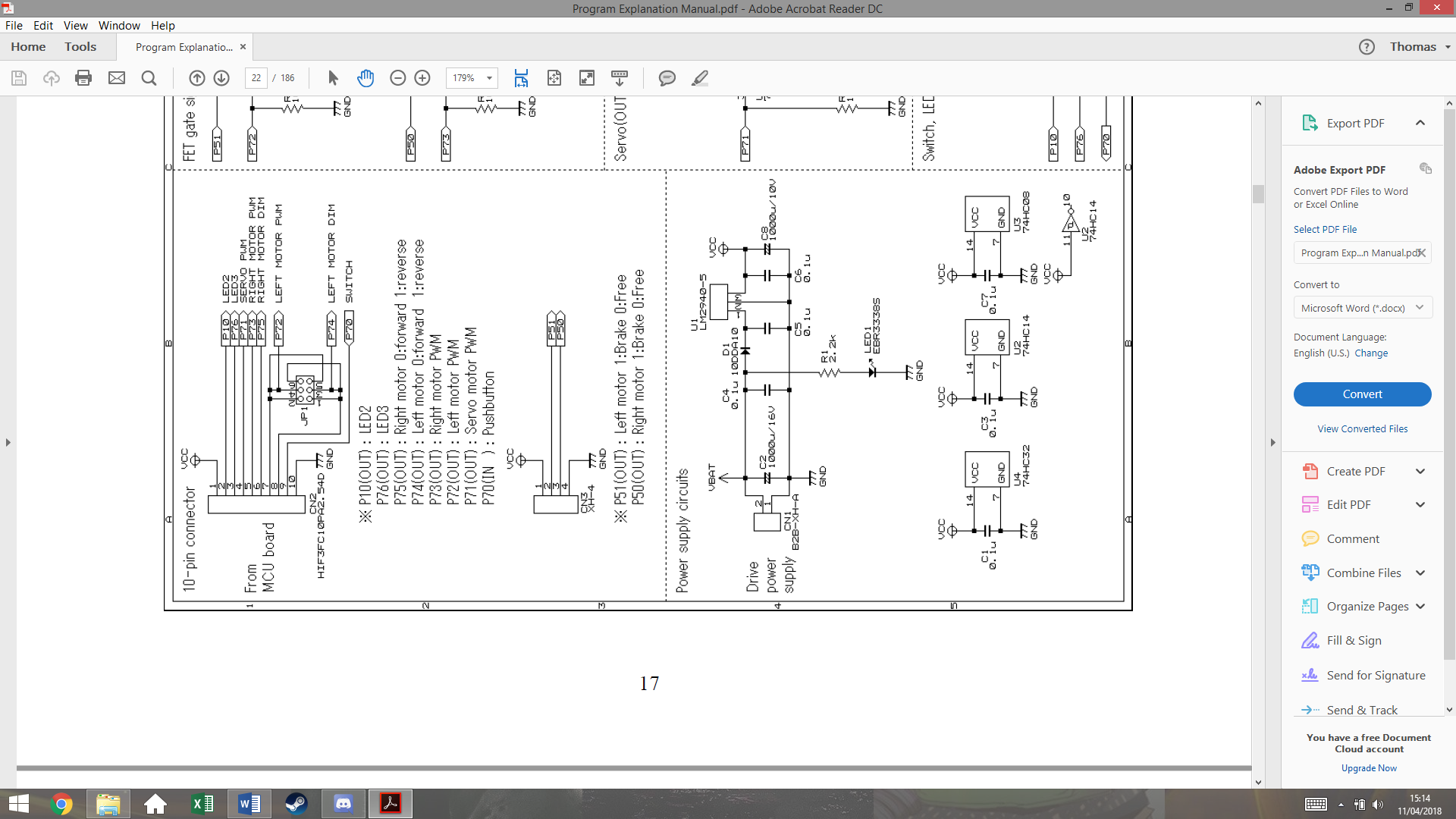
The start bar sensor works in almost the same way but shares the output Bit 0 with the right edge sensor. This means that during start up, the Bit 0 signal is used to detect the presence of the start bar, but then switches to detecting the right edge of the track.

This board is connected to the MCU via a 10 wire ribbon cord, with 8 signal wires, and 2 for Vcc and ground.

### Power Board

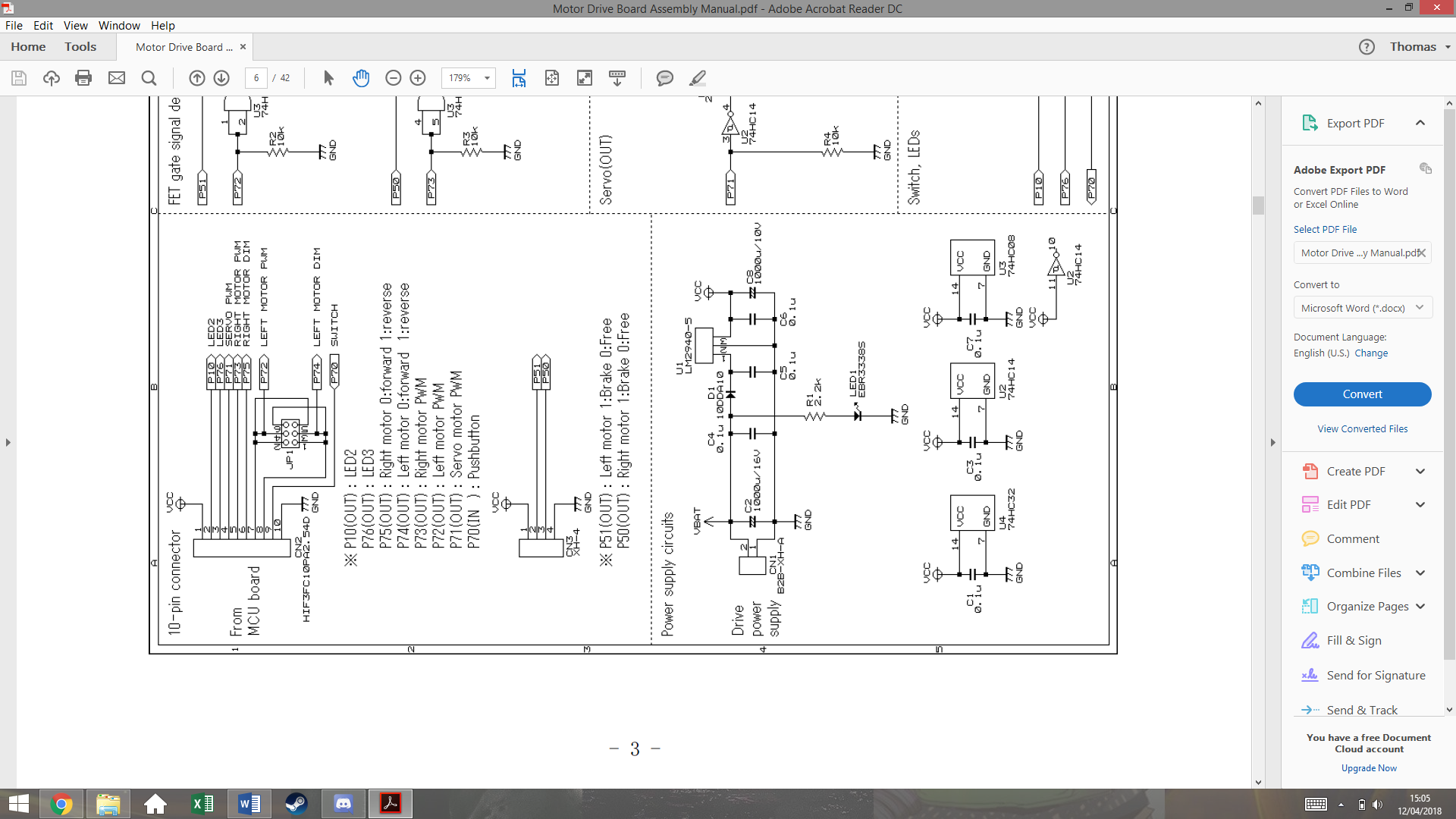
The power board’s function is to control and power the rear DC motors, as well as provide power to the servo and MCU. In its default state is can handle a power supply of 4.5 -5.5V, which is applied directly to the motors, the servo and the MCU board. However, with the addition of two linear regulators, it can be supplied with up to 7-15V. This increases the maximum speed of the DC motors and allows the use of more batteries in parallel, which in turn gives the car a greater on-board charge and battery life.

The schematic of this board can be divided up into sections.

This section shows the connections to the MCU board. It’s important to be aware that the Vcc is connected **from** the power board **to** the MCU board.

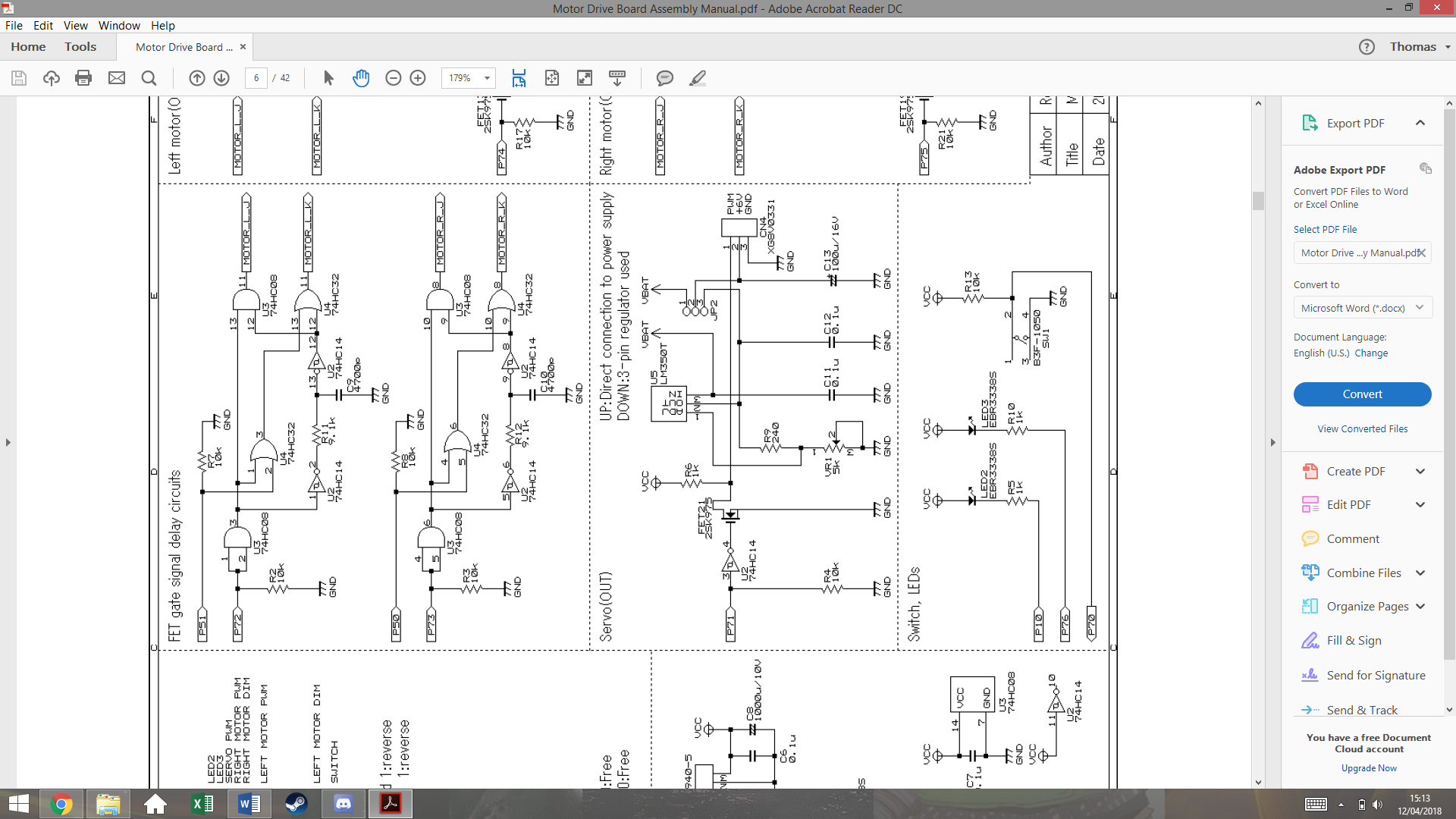
The ribbon cord from the MCU contains the PWM signals for both motors, and the direction signal for each motor (0 forward, 1 reverse). The only purpose for JP1 is to switch the Left Motor PWM and LEFT Motor DIM signal for other boards. This also contains the pushbutton signal that will be used to run the car.

CN3 is for optional use, and can be used for braking the motors. This is specifically useful for the 90° turns. This would not allow for any dynamic braking as it would completely stop any signal sent to the motors.

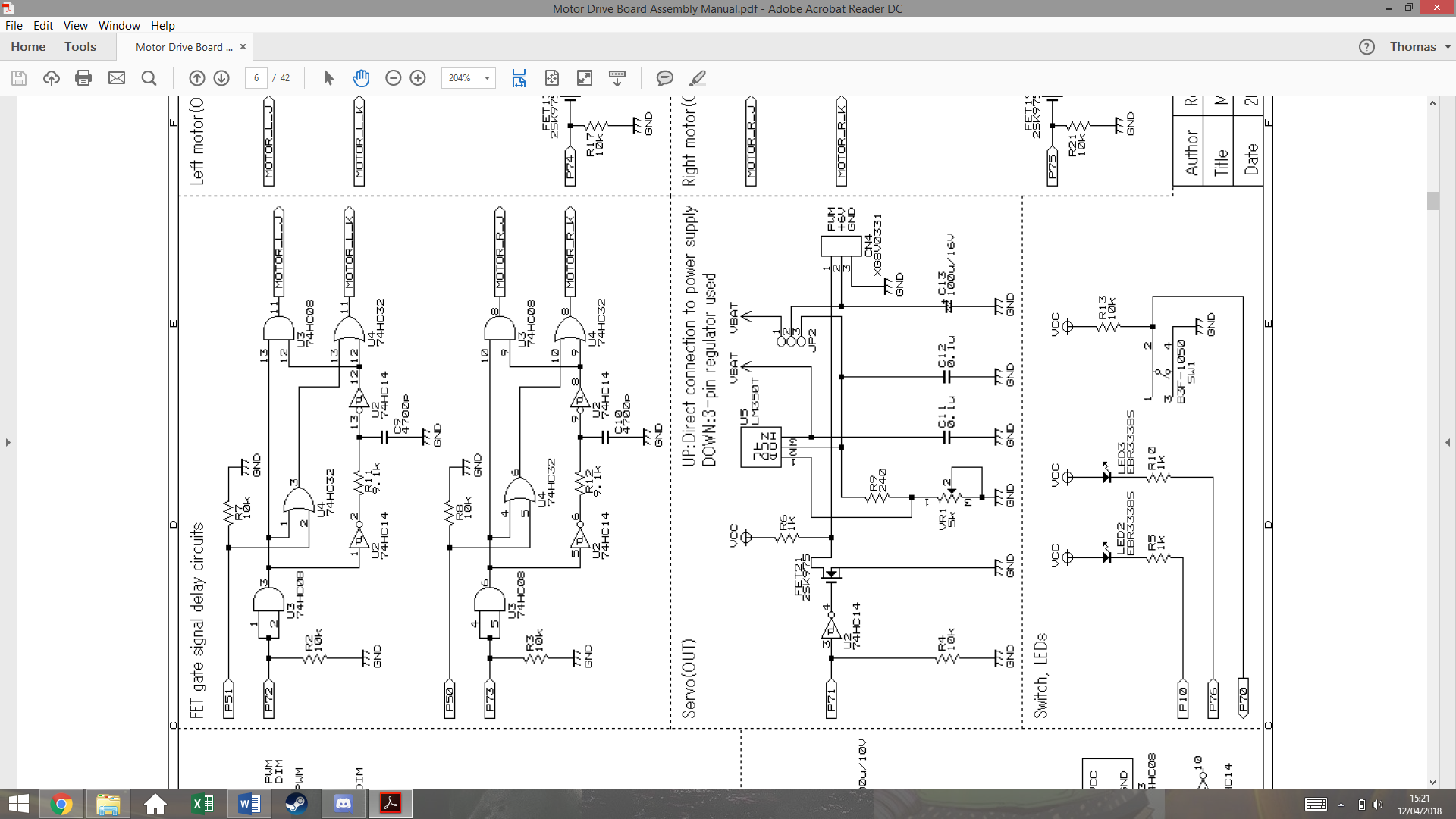


This section shows the power supply of the board. The top circuit regulates the voltage supply Vcc for the MCU board. The LM2940-5 is a 5V regulator that is not included in the initial package. C5 and C6 are also not included so these must be purchased. LED 1 is simply used to indicate that the batteries are connected and that there is power.

The bottom circuit shows the Vcc connected to the logic chips U4, U2 and U3. The capacitors C1, C3 and C7 are decoupling capacitors that remove high frequency components on the supply, often cause by the power supply switching on. This stops the chips being damaged by inrush current.

This section shows the servo circuit. U5 is another linear regulator not included. This regulator is adjustable via the potentiometer VR1, which can be varied to give a 6V output. Jumper JP2 will have a short between pins 3 and 2, so connect the 6V output of U5 to the servo, instead of the battery voltage on pin 1.

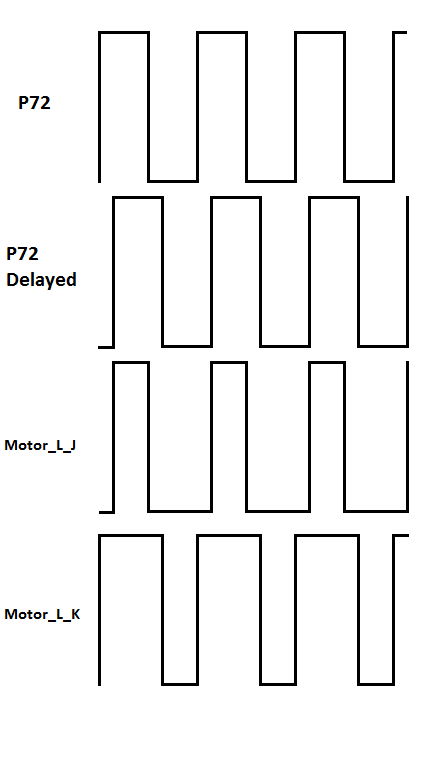
U2 is an inverting Schmitt trigger that both isolated the MCU, stopping current being drawn from the port, and inverts the PWM signal.

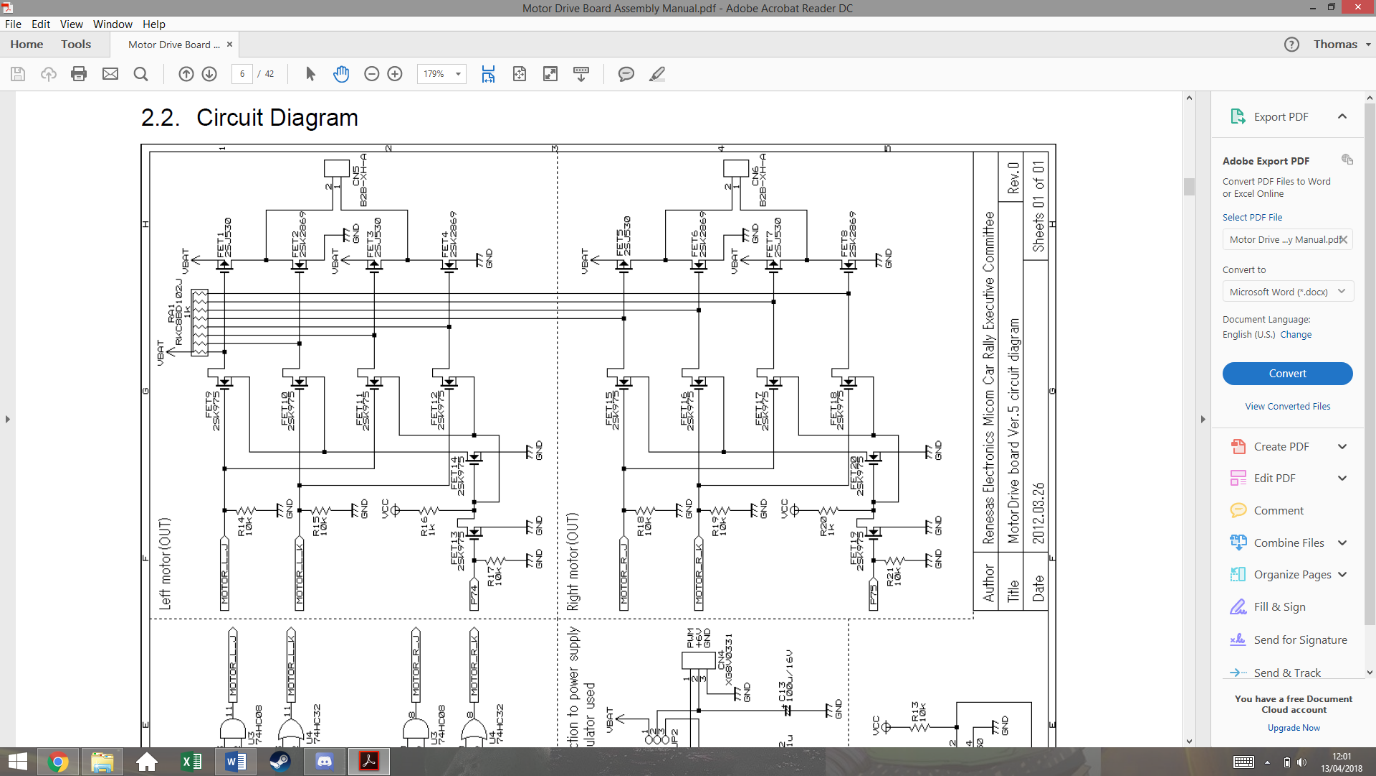


This section shows the logic circuits that control motor direction and speed. Both of these circuits are identical. The first AND gate acts as a buffer that isolates P72/73 on the MCU.

The output Motor\_L\_J is essentially just P72 but distorted with a delay, as it is the AND function of P72 before and after being filtered by R11 and C9. As it is filtered and then passed through a Schmitt trigger, a delay is created.

The purpose of this delay is to stop a short in the circuit that could be created later. For now, it’s easier to understand with the removal of P51.

This trace shows the output signals of this section of the circuit, without P51. The output Motor\_L\_K gets inverted by K-FETs in a later circuit. This explains how the short is prevented.



1

2

3

This final section of the power board is the transistor circuit that controls the motors. The bottom and top circuits are the same, but for the two different wheels. The circuit is made of p-channel and n-channel mosfets. It’s important to remember that n-channel allow current flow when the gate is high, whereas p-channel allow current flow when the gate is low.

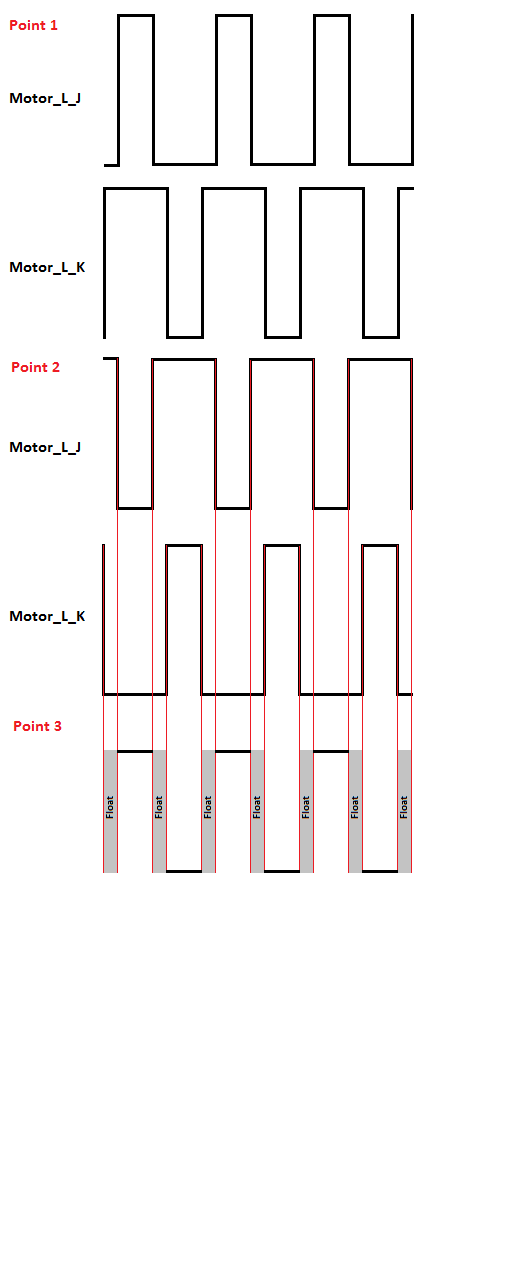
FET 1, 3, 5 and 7 are p-channel mosfets, the rest are n-channel

The resistor array functions as pull up resistors for the circuit. This means that if all of the FETs on the left and side are in an off state, the gates of the FETs on the right will be high. This will turn the p-channel FETs off and the n-channel FETs on.

If P74 is low, FET13 is off which turns FET14 on. This connects the source of FET9 and 10 to ground. This also connects the source of FET 11 and 12 to Vcc.

This means they are not functioning and keep the signal at their drains high, which hold pin 1 of the motor to ground.

The signal traces for 50% PWM earlier can now be used to analyse this design at points 1, 2, 3 on the schematic. This example has P75 as a value of 0.



The first trace shows the previously derived waves for input Motor\_L\_J and Motor\_L\_K. These signals go to the gates of FETSs 9, 10, 11, 12.

Point 2 shows the signals on the drain pins of FETs 9 and 10, and on the gate pin of FETs 1 and 2. Due to the pull up resistor, when FET 9 or 10 are low, the signal is high, and when the FETs are allowing current flow, the signal goes low, creating this inverted wave.

Point 3 shows how these signals effect this side of the motor input. Due to FET1 being a p-channel FET, when is gate signal is low, it conducts. This produces the wave form at pin 2 of the motor. The delay has created a period when the pin is not connected to Vcc or ground.

This is for safety as if they were ever both connected, a short would be created.

If P74 is on, these same signals will be produced but with FETs 11 and 12 instead, while the outputs of FET 9 and 10 would hold pin 2 of the motor at ground. This effectively applies the same signal but with the pins switched around, reversing the motor.

## Speed Control and Feedback

In order for the car to handle optimally, forms of speed control need to be implemented, so that the car can brake and accelerate dynamically. With one or two feedback methods, a closed loop PID system can be created.

After some brief research, three methods of speed feedback were found, each with their own disadvantages and advantages.

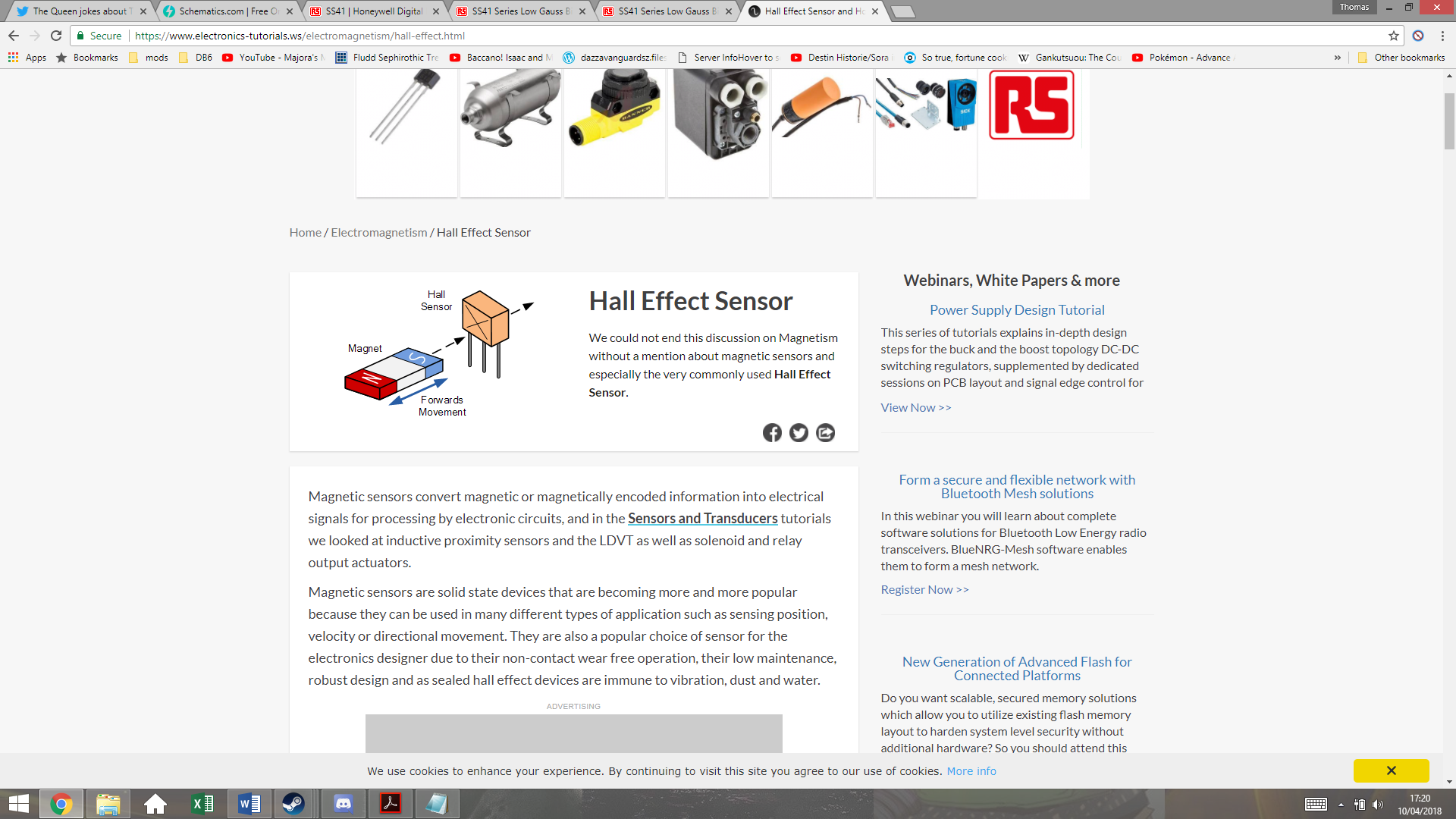
**Optocoupler**- An optocoupler is made from an IR LED and an IR sensor, which will generate a pulse every time light is able to travel between them. This would be implemented by cutting holes in one of the gears and placing the optocoupler across them. The advantage of this is its low cost but would be larger than the alternatives. It would also have to be placed inconveniently above the gearbox, making it hard to get to for fault finding. The decision was made not to implement this method however it would be kept as a backup plan.

**Hall Effect Sensor**- A hall effect sensor is a small IC that will turn on or off depending on the magnetic flux present. This would require magnets to be attached to the real wheels so that the sensor would produce a square wave, with a frequency matching the wheels rotational speed in revolutions per second. The advantage of this is that the sensor would be placed right next to the wheel, making it much easier to access that the optocoupler. The main disadvantage is the need to attach magnets to the rear wheels, slightly increasing the load. This method was decided to be our main form of speed sensing.

**Accelerometer**- While normal used to measure acceleration, and accelerometer could be used with software integration to indirectly measure the speed. The method has the advantage of not having to change the mechanical design in any way, but as it is not a direct speed measurement, it may not be very accurate. This was decided to be a secondary measurement for speed, as well as helping dynamic braking and acceleration by giving acceleration feedback.

### Hall Effect Sensor

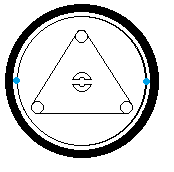
A hall effect sensor is and IC that can detect the presence of a magnetic field. They’re often used in motor control to measure rotating magnetic fields. The image below shows and example.



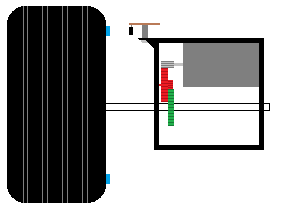
As a southern pole passes in front of the sensor, the output voltage goes high, and as a northern pole passes in front of the sensor, the output will return low. If the magnetic field is removed, it will maintain the same voltage. Meaning that it requires both poles to create an alternating signal.

Figure 3‑1 - Credit www.electronics-tutorials.ws

Both of the rear wheels will have two magnets attached on opposite sides to create a rotating magnetic field. These magnets will be attached with superglue, which should be enough to hold them even when the car is traveling at full speed. The magnets have to be placed exactly opposite each other, or this will create an unbalanced load on the DC motor. The magnets that will be used are Eclipse 5mm Neodymium Magnetic Discs, which are small but powerful. The diagram below shows how they will be positioned.

The magnets will be placed on the rim of the wheels, next to but not touching the tire. The roll bar connector in the centre of the wheel can be used to make sure they are aligned with each other. These two magnets must be polar opposite so that they switch the sensor on and off with each rotation.

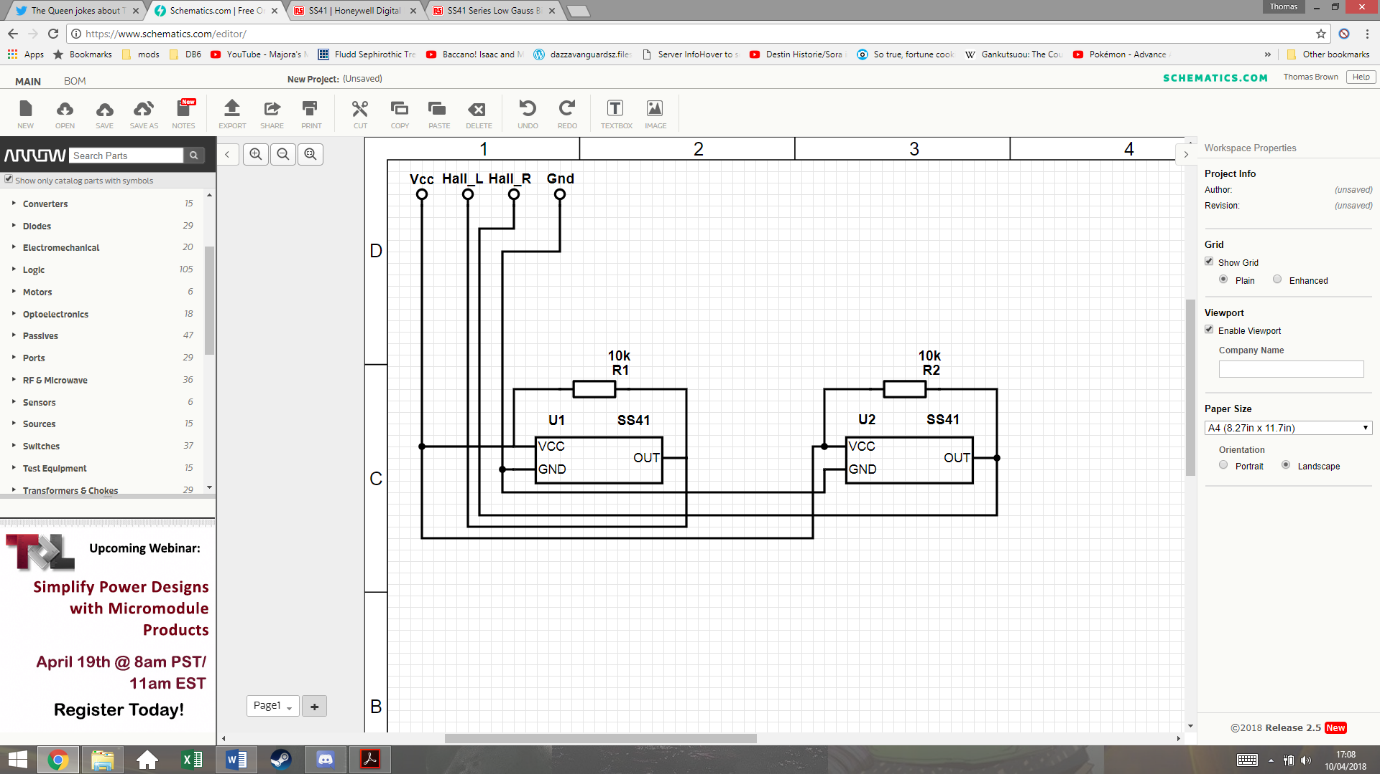
The Hall Effect sensor chosen is the SS41 Honeywell Digital Hall Effect Sensor. It has an input voltage range of 4.5-24V, meaning that it’s suitable to be run of the Vcc of the MCU board. Its maximum current draw when switching with a voltage supply of 5V is 8.7mA, which shouldn’t impact battery life significantly.



Hall effect sensor

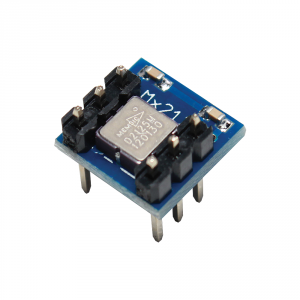
Magnet

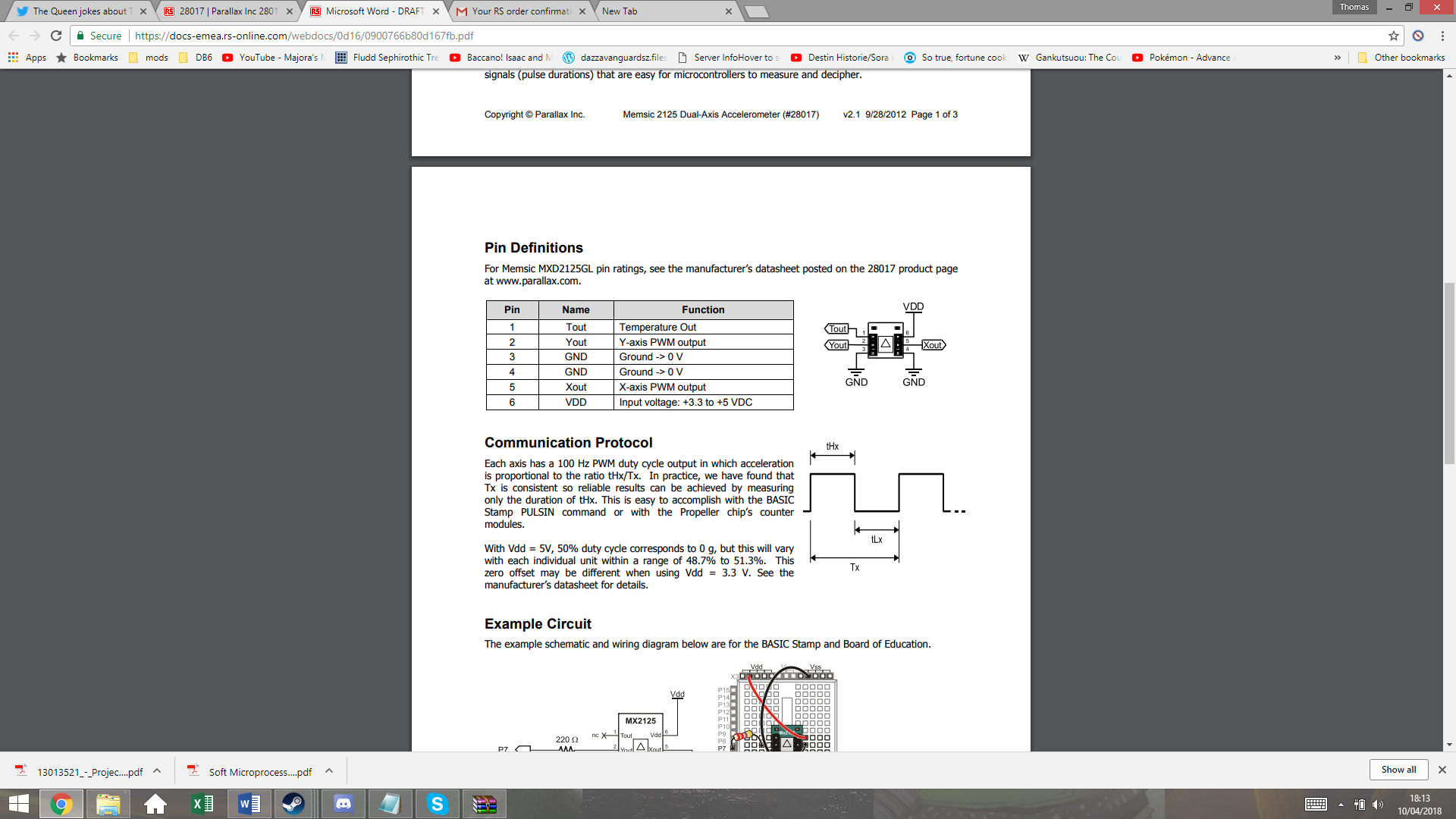
This image shows how the sensor must be positioned. The sensor is soldered onto a small piece of Veroboard, which is attached to the gear box using a screw and a spacer. The spacer is to keep it at the same level as the magnet. There is an air gap between the sensor and the magnet, but this should not be large enough to present a problem, given the strength of the magnets used.

This schematic shows how the system will be connected. The two Hall Effect sensors have pull-up resistors that prevent any floating values. They both share the Vcc and ground from the MCU board. The two outputs are then connected to I/O pins on the MCU board.

### Accelerometer

The use of an accelerometer in the car control allows the program to know the forces acting on the car. This can help prevent the car coming out of the track due to inertia and can also help breaking for the corners. For this application, only a two-axis accelerometer is required, as there should be no forces acting on the car perpendicular to the ground.

The accelerometer chosen is the MESMIC 2125 two axis accelerometer. This can easily be inserted and removed from a standard 6 pin IC socket.



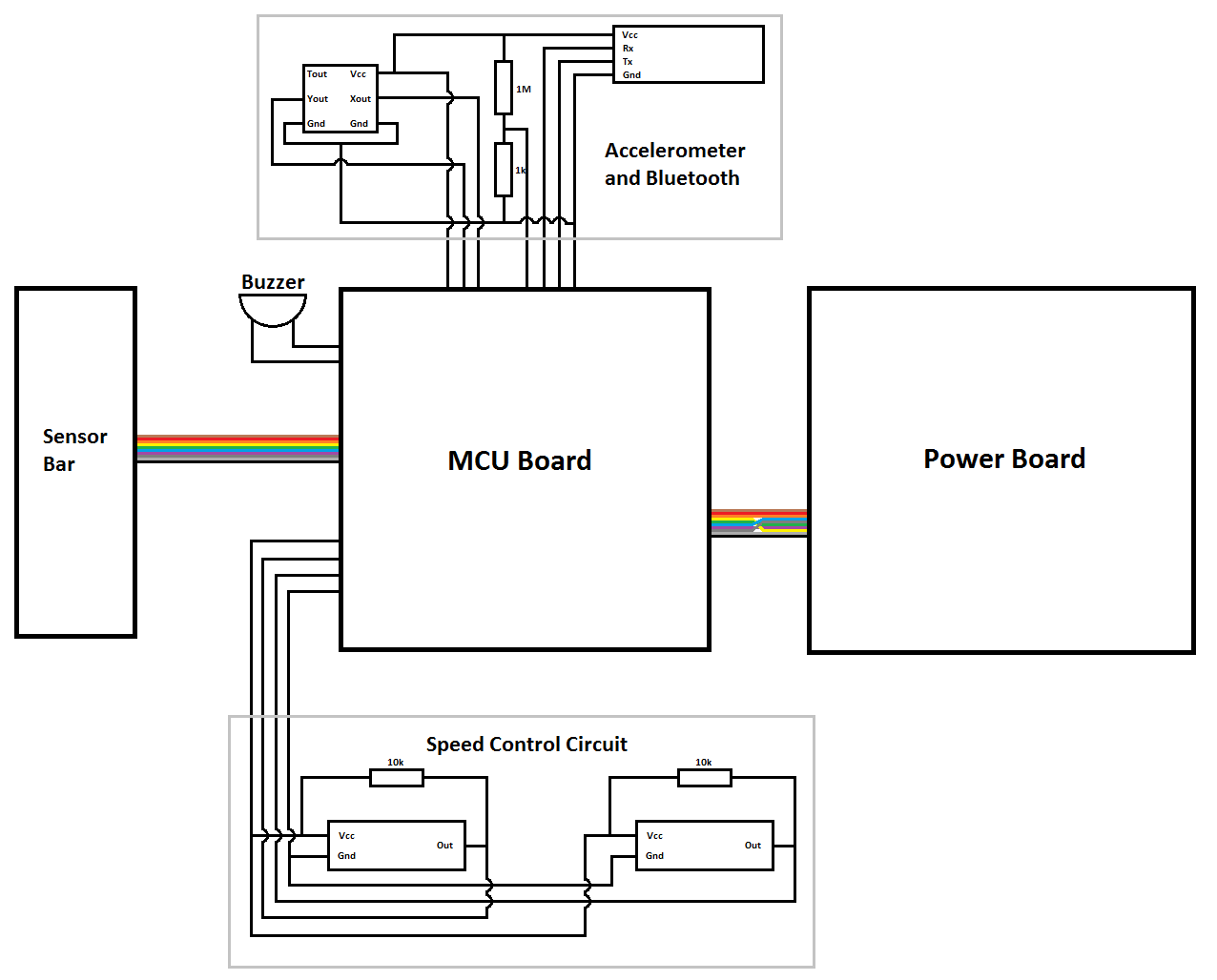
The supply voltage range for this device is 3.3-5Vdc, so this can also be run from the MCU boards Vcc. The output Xout and Yout will be used to give the X and Y coordinates of the vector of acceleration. These output give PWM signals, with duty cycles that vary in proportion to acceleration.

Tout is the temperature output from a thermistor inside the IC, but this will not be used in this system.

The datasheet states a sensitivity of 12.5% duty cycle per g on both axis, with 50% duty cycle representing 0 gg. This gives a range of +/-3 g. Taking g to be 9.81m/s^2, this gives a maximum measurable acceleration of 29.43m/s^2, which the car should not be able to exceed.

## Overall Electronic Design

Below is the overall electronic design, which shows all external connections to the MCU board.

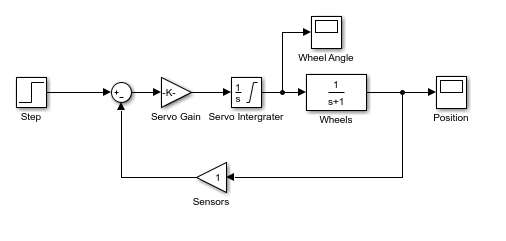


The speed control circuit requires 4 connections, for Vcc, left motor speed, right motor speed, and ground. The buzzer is attached between an MCU GPIO pin and ground, and is separate to the other systems so that it can be physically positioned anywhere. The peripheral board at the top has 7 connections. Vcc and ground for power, Xout and Yout from the accelerometer, Rx and Tx from the Bluetooth, and the scaled down battery voltage reading.

The power and sensor boards are connected via ribbon cord as per the original design.

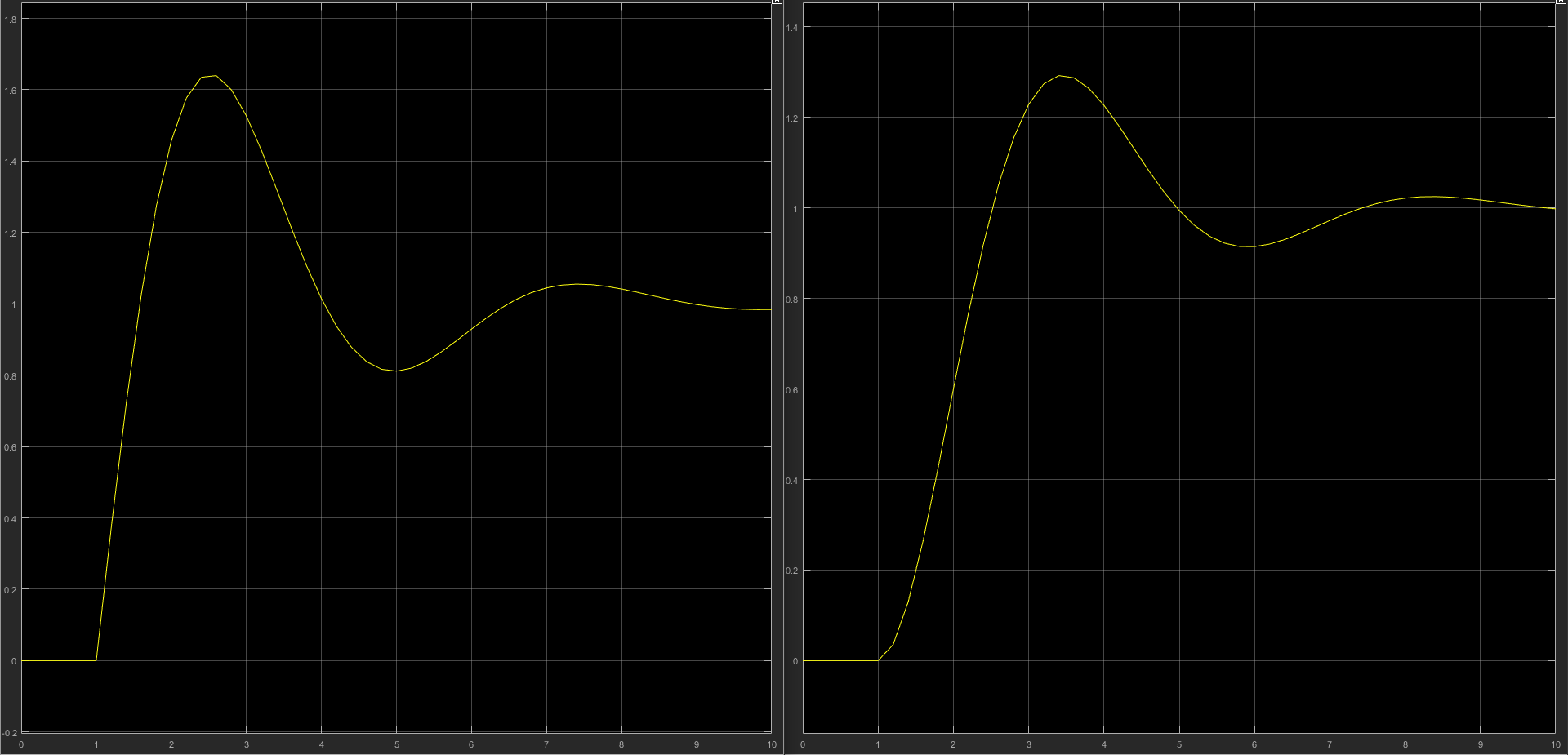
## Control theory

Before the control system can be implemented, it is a good idea to model the system on its own to judge what system it will need. Below shows a Simulink model for the position tracking system.



This shows a step response for position target being used as the input. The servo takes this voltage and in effect converts that to an angle. The angle of the servo will dictate the rate of change in position from the line. This means that an integrator can be used in this model. A low pass filter is then used to model the reaction of the wheels.

The sensors don’t have a gain and it was uncertain how the sensor data would be processed. In this model, we have it just feeding the position data back. The reaction of the system can be seen below.



This shows that with just closed loop feedback, the system will overshoot and oscillate, meaning that this will require a PID control system.

## Software

The software is built in a modular fashion with each component being entirely independent. The drivers are designed separately and are controlled via a handle that is allocated in memory. This handle identifies a single hardware component and allows the application code to control the physical device without having to know the implementation of the driver at the lower levels. In addition, errors can be isolated to single points of failure, which is ideal for debugging.

Finally, the code was developed on the e2studio IDE instead of HEW, given that it is the most recent IDE provided by Renesas.

### Program structure

In detail, the program can be split into two major designs:

1. Backend – system tasks unrelated to the control of the racing car;
2. Frontend – the application that actually controls the logic of the car.

### Backend

The backend describes the code that executes the background tasks that are not visible to the user application. In this project, these tasks/features are:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Component name | C file/folder name | Description |
| 1 | **Micrium uCOS III - RTOS** | lib/rtos/ | The Real Time Operating System |
| 2 | **Motor driver** | drivers/actuators/motor\_driver/ | Controls the two DC motors at the back |
| 3 | **Servo driver** | drivers/actuators/servo/ | Moves the sensor bar sideways by swinging the servo from +- 90º |
| 4 | **Bluetooth driver** | drivers/communications/bluetooth/ | Provides pairing with a computer or smartphone |
| 5 | **Software UART**  **(bit banged UART)** | drivers/communications/protocols/suart | Uses software-defined UART on the Bluetooth module with baud rate 9600. (Hardware UART was not used due to GPIO shortage.) |
| 6 | **Packet manager**  **(layered communication protocol)** | drivers/communications/protocols/packetman/ | Abstracts the extremely simple UART protocol and “packetizes” every single transaction that is made through UART. In essence, it adds a Transport and Application layer on top of the Data link layer (just like TCP/IP). |
| 7 | **Software PWM**  **(bit banged PWM)** | drivers/libs/spwm/ | Software-defined PWM which provides a greater level of accuracy of duty cycles through the use of floating point values rather than a 16-bit value that ranges from 0 to 216-1.  This component is used under the implementation of the motor, servo, piezo buzzer, LEDs and accelerometer. |
| 8 | **LED driver** | drivers/onchip/led.c | Controls the on-board LEDs, as well as the LEDs on the power board. |
| 9 | **Switch driver** | drivers/onchip/switch.h | Reads the push-button switch and returns 1 if the user presses it. |
| 10 | **Accelerometer driver** | drivers/sensors/accel | Measures the momentum of the car in the X and Y axis. |
| 11 | **IR Line tracker driver** | drivers/sensors/ltracker | Reads and returns the measured line in byte format. This pattern is then converted into an equivalent angle later on by the frontend side of the program. |
| 12 | **Piezo buzzer driver / MIDI player** | drivers/sound/piezo.c | Drives the piezo buzzer in order to listen for debugging messages in real-time just as the race is happening. |

Table 3‑1 - Software: the backend structure

Each of these components is then used by the frontend application. If something fails, for instance, the speed of the wheels is not the correct value, then there are two possibilities: either the user application is misusing the driver and generating erroneous calculations for the speed, or the motor driver code is poorly implemented.

This explains why developing an independent C driver for each component is extremely important.

### Frontend

On the other side, we have the application code that “glues” the backend code together and actually uses the developed services in order to control the physical devices for the final purpose – driving the car autonomously.

The logic for that goes into this frontend section. Specifically, we may find the components:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Component name | C file/folder name | Description |
| 1 | **Main thread** | src/app\_main.c | RTOS runs this task/thread first. This task then initialises the drivers for the car and waits for the user to press the push-button switch in order to kick-start the car. |
| 2 | **Car controller** | src/app\_car\_control.c | The previous component triggers this component when the user presses the button.  All of the car control logic goes in this component. |
| 3 | **Car configuration** | src/app\_config.h | Stores configuration about the car, for instance, whether the sound is enabled or not, the number of RTOS tasks, the PID coefficients, the integral windup period, etc. |
| 4 | **Track data** | src/app\_track\_data.h | Stores data regarding the track itself, specifically how many corners exist and the direction of the curve. It also contains a map that converts the measured sensor pattern into an equivalent (approximated) value in angle (degree) format. |
| 5 | **Template generator** | src/app\_template\_generator.c | Allows the car to read the sensor data while it’s running and sending this same data to the computer in a burst in order to allow “playback” of the measured track at a later time. This feature is used in the ‘Smart mode’ described on section **4.3.2.1.2 – Levels of intelligence**. |
| 6 | **Shell command line** | lib/shell | Provides an interface for the user to type commands in order to control the car. (This connection is done via Serial COM port through the Bluetooth module HC06.) |

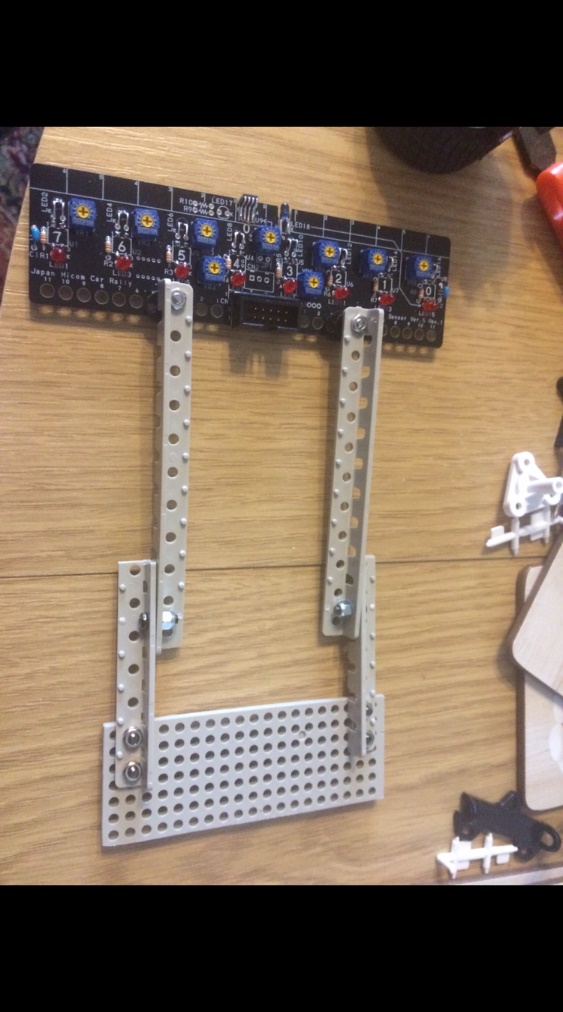
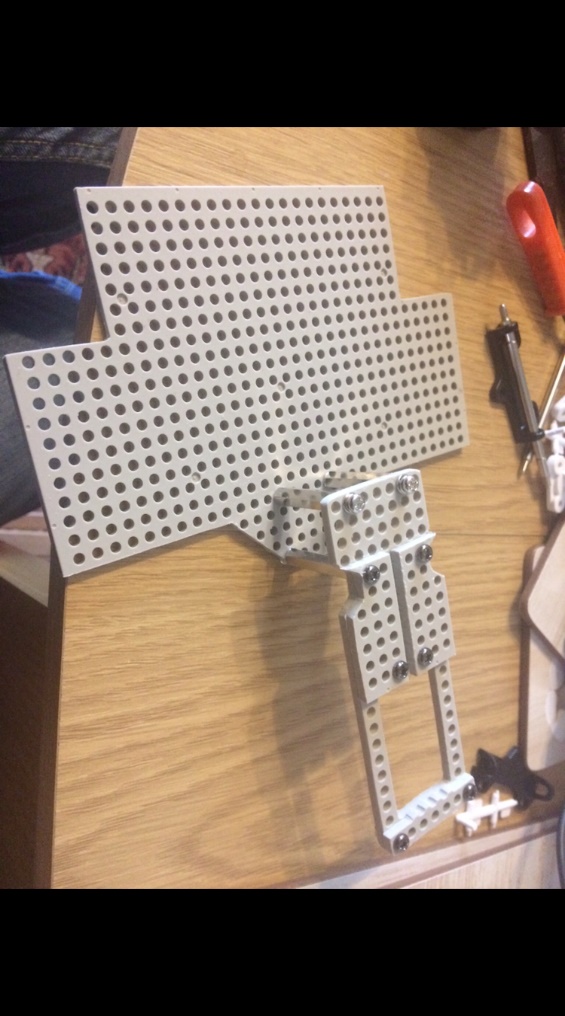
Table 3‑2 - Software: the frontend structure

# Implementation

## Mechanical

The mechanical build started by using the provided plastic board to cut out and shape the bottom level board (point 1). This was cut out using a hacksaw and then a file and sandpaper were used to shape it and smooth down any rough edges.

This process was then repeated for the servo bridge and servo reinforcing plates (point 2). The servo reinforcing plates where attached to the bridge with screws and nuts so that they could be filed down together, however they would have to be removed later to attach the servo. The bridge was then attached to the bottom level board with the four pillars.



1

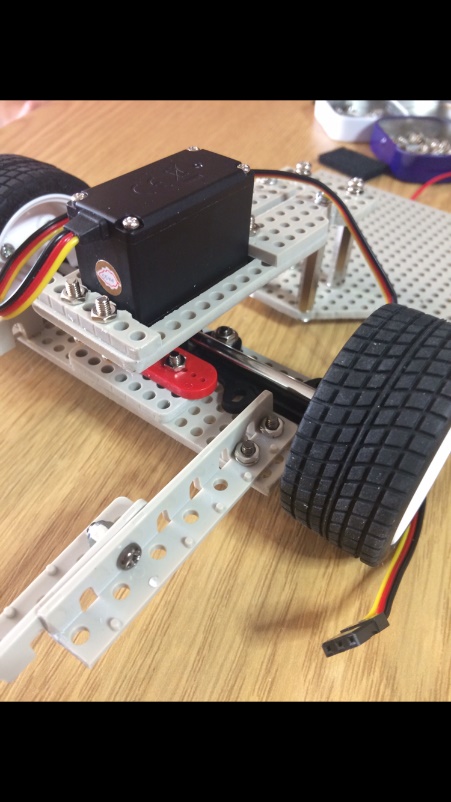
2

3

4

The axel is then cut out and shaped (point 3) but this is relatively easy as it’s just a rectangle. The arms are then assembled. The top part of the arms are already the correct length and shape, but the lower arms must be cut and smoothed.

The nylocs can be seen (point 4) one hole in to the lower part of the arm. This extra hole could be used to extend the length of the arms later on. The theory was that while longer arms may put more load on the servo, it would allow the car to detect changes in the line earlier.



The servo is then attached to the front axle via the red servo horn, and then mounted on the servo bridge using the reinforcing plates. A spacer is placed between the servo horn and the axle plate so that the wheels are the correct distance down from the servo.



The gearboxes are then assembled ready to be mounted on the back. The motor is inserted into the housing before the capacitors and wires are soldered on, so that the contacts can slot through the back of the housing. The purpose of these capacitors are to filter noise and high frequency voltages from the PWM signals.

Once the wheel has been attached to the shaft and inserted into the housing, any extra shaft is cut away so that the gearboxes can be placed closely together. The shaft is held in place by a hexagonal gear joint and set screw. The gears are then positioned to complete the gearbox.

## Electrical

### Provided Circuits

The servo board was a simple build, however did require some accuracy in aligning the IR LEDs so that they all gave out an equal amount of light. After the build the potentiometers was adjusted to each give an equal voltage to the sensors, however this would be adjusted later to give the best sensitivity.

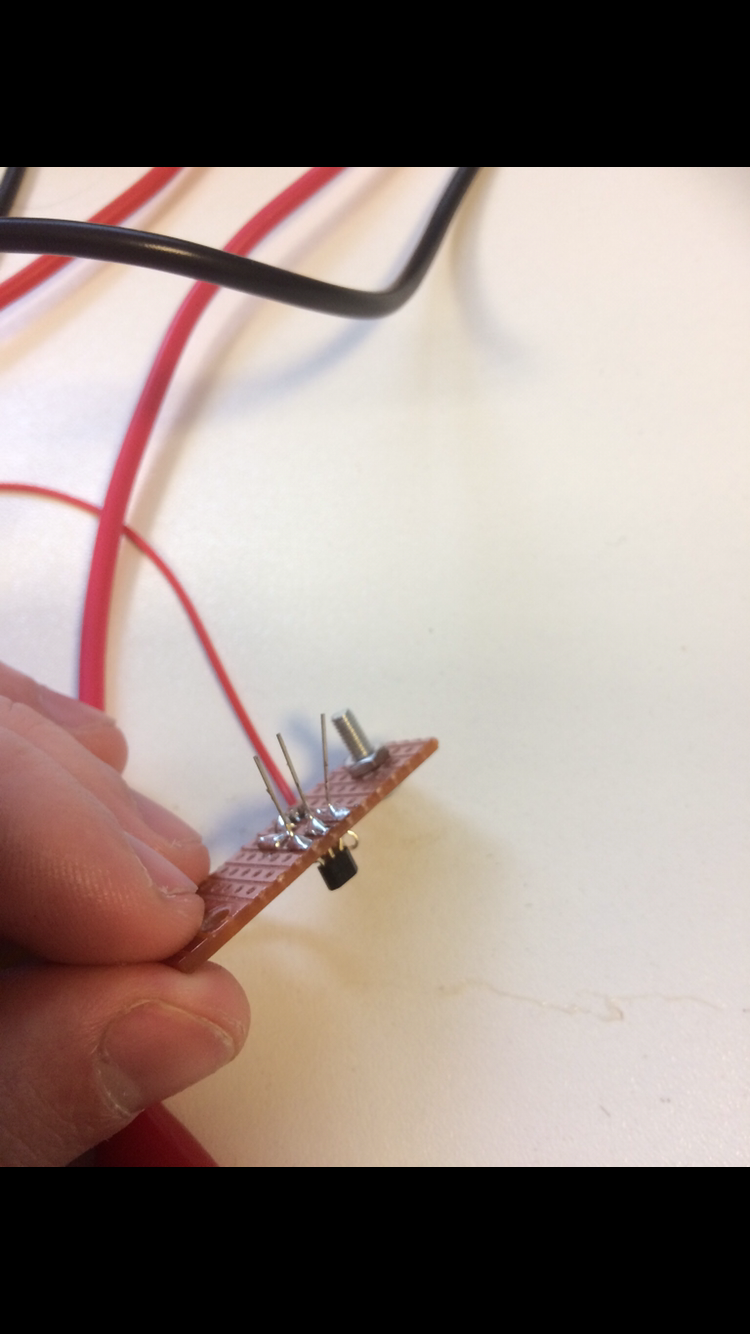
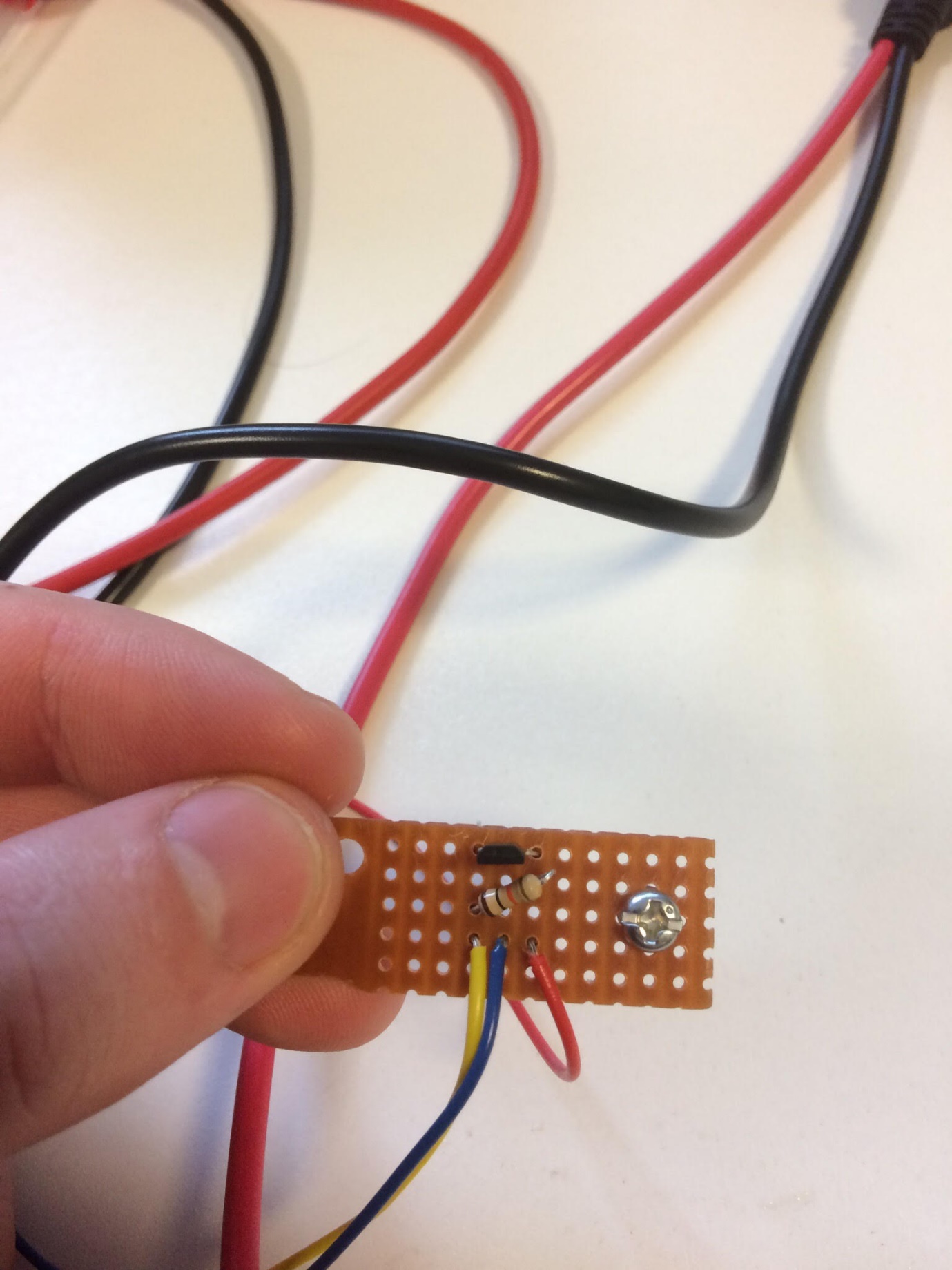
The power board was a much larger build, and to reduce the amount of time that may have been spent on fault finding, a great deal of care was put into the soldering of each component. The logic gate chips were soldered on a lower heat so as to be sure that they wouldn’t break.

The extra regulator components were added after testing the power board with a 5V supply.

### Hall Effect circuits

The image below shows one of the Hall Effect sensor board.

This was created by using a hacksaw to cut two small rectangles out of a piece of Veroboard, and then drilling the appropriate holes. The components were then soldered according to the schematic, making sure that there is no electrical path to the screws.



The legs of the Hall Effect sensors are not cut off just yet, as it makes testing easier if the legs are big enough to have crocodile clips attached.

### Peripheral Board

The peripheral board was an easy build. IC sockets were cut to the right sizes so that the accelerometer and Bluetooth systems could be removed is needed and didn’t have to be soldered directly. This is important because the accelerometer relies of temperature to function and exposing it to high temperatures could make it in accurate. Any extra copper on the Veroboard has been scratched away so as to reduce the risk of there ever being a short circuit.

Holes were drilled in the Veroboard so that it could be mounted behind the servo, as per the mechanical design, with spacers so that the bottom of the board does not touch the structure. The wires from the board are trimmed so that they are only as long as they need be, as they could otherwise get caught on something.

## Software

This section covers the implementation of the software from the two perspectives Backend – Frontend. It explains how the main components function internally and how they provide their own service/output data to the application code.

### Backend

#### RTOS

For this project, a Real Time Operating System (RTOS) was necessary to deal with the many components accurately and safely. This provides fine control on each task, allowing us to execute a certain operation periodically with guaranteed certainty that it will deliver. If an issue occurs during the execution of a task, RTOS makes sure the entire system either halts or pauses the faulty task in order to prevent the rest of the system from entering a “buggy” state.

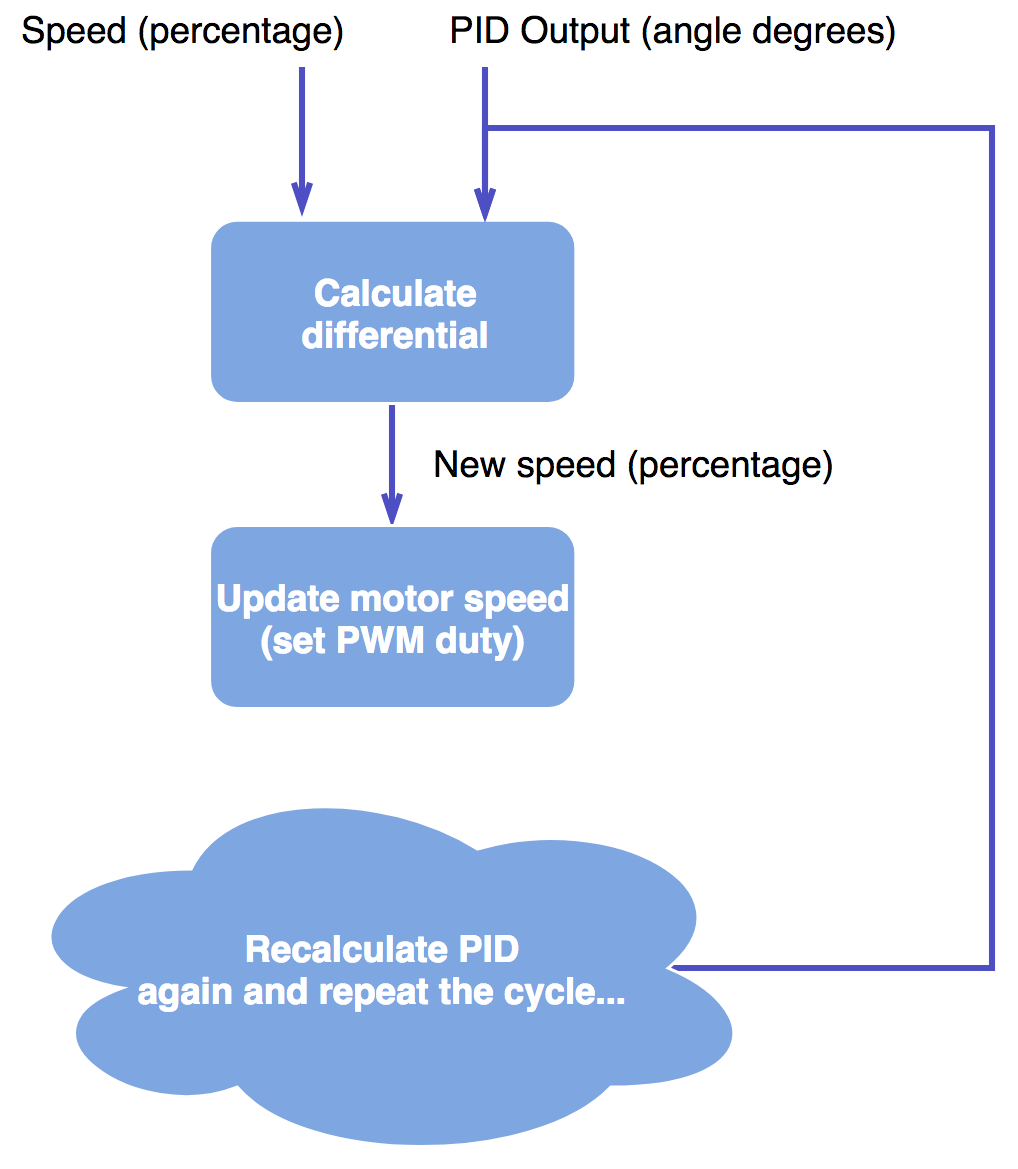
The RTOS being used is the open source uCOS III from Micrium, which runs on a variety of platforms, from ARM and RISC-V to Renesas’ set of architectures, such as the RX62G/RX62T.

#### Peripheral drivers

As was already described on the previous section, the drivers are built entirely separately. Each driver is implemented in a way that allows the application to run only the necessary functions, while keeping the private inner functions hidden away from the application. A physical device is abstracted into what is called a handle, or a pointer, which is dynamically allocated on the heap with the use of the malloc function.

##### Motor driver

The most important driver is what makes the car move. This module uses software defined PWM to output the signal and drive the motor. It also features differential calculations, which is achieved through the PID output. The diagram below shows the inner functionality of the code for the motor.



The supported API (Application Programming Interface) for this driver may be found below:

|  |
| --- |
| **typedef** **struct** {  ***spwm\_t*** \* *dev\_handle*;  **float** *speed*; /\* Expressed in % from 0 to 100. Value can be negative (for reverse rotation) \*/  **float** *speed\_old*;  **float** *speed\_safe\_old*;  **float** *max\_speed*;  **float** *deceleration*;  **float** *rpm\_measured*;  ***uint32\_t*** *rpm\_timestamp*;  ***uint32\_t*** *rpm\_timestamp\_old*;  bool *rpm\_timestamp\_triggered*;  bool *is\_braking*;  bool *is\_safemode*;  bool *is\_stopped*;  **enum** **MOTOR\_CHANNEL** *side*;  } ***motor\_t***;  ***motor\_t*** \* motor\_init(**enum** **MOTOR\_CHANNEL** channel);  ***motor\_t*** \* motor\_init\_safe(**enum** **MOTOR\_CHANNEL** channel, bool enable\_safemode);  **enum** **MOTOR\_RETCODE** motor\_reset(***motor\_t*** \* handle);  **enum** **MOTOR\_RETCODE** motor\_stop(***motor\_t*** \* handle);  **enum** **MOTOR\_RETCODE** motor\_stop2(***motor\_t*** \* left\_motor, ***motor\_t*** \* right\_motor);  **enum** **MOTOR\_RETCODE** motor\_resume(***motor\_t*** \* handle);  **enum** **MOTOR\_RETCODE** motor\_ctrl(***motor\_t*** \* handle, **float** speed\_percentage);  **enum** **MOTOR\_RETCODE** motor\_ctrl2(***motor\_t*** \* left\_motor, ***motor\_t*** \* right\_motor, **float** speed\_percentage);  **enum** **MOTOR\_RETCODE** motor\_ctrl\_with\_differential(***motor\_t*** \* handle, **float** speed\_percentage, **float** pid\_output);  **enum** **MOTOR\_RETCODE** motor\_ctrl\_with\_differential2(***motor\_t*** \* left\_motor, ***motor\_t*** \* right\_motor, **float** speed\_percentage, **float** pid\_output);  **enum** **MOTOR\_RETCODE** motor\_refresh(***motor\_t*** \* handle);  **enum** **MOTOR\_RETCODE** motor\_refresh\_with\_differential(***motor\_t*** \* handle, **float** pid\_output);  **enum** **MOTOR\_RETCODE** motor\_refresh\_with\_differential2(***motor\_t*** \* left\_motor, ***motor\_t*** \* right\_motor, **float** pid\_output);  **enum** **MOTOR\_RETCODE** motor\_set\_safe\_mode(***motor\_t*** \* handle, bool enable, bool update\_speed);  **enum** **MOTOR\_RETCODE** motor\_set\_speed(***motor\_t*** \* handle, **float** speed\_percentage);  **enum** **MOTOR\_RETCODE** motor\_set\_speed2(***motor\_t*** \* left\_motor, ***motor\_t*** \* right\_motor, **float** speed\_percentage);  **enum** **MOTOR\_RETCODE** motor\_set\_braking(***motor\_t*** \* handle, bool is\_braking);  **enum** **MOTOR\_RETCODE** motor\_set\_braking2(***motor\_t*** \* left\_motor, ***motor\_t*** \* right\_motor, bool is\_braking);  **enum** **MOTOR\_RETCODE** motor\_set\_max\_speed(***motor\_t*** \* handle, **float** new\_max\_speed); |

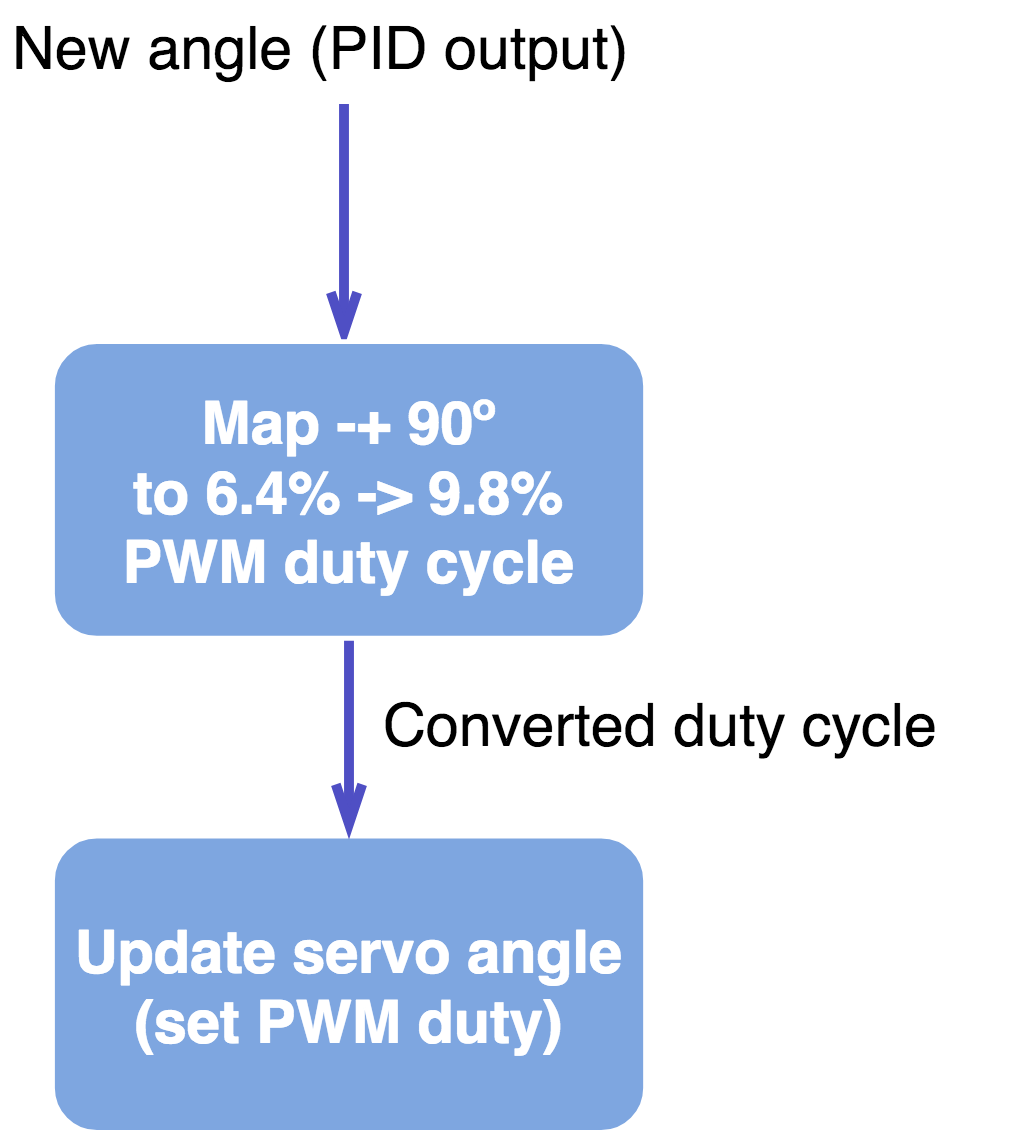
These functions are used by the application code. As can be seen, the driver allows the application to:

1. Initialise the motor in two different ways (normal and safe mode, which is just normal mode with the speed capped to a safe level);
2. Allows resetting the module variables (see the motor\_t struct for reference)
3. Supports stop and resume functions;
4. Control of the motor with new speed (ctrl) with or without differential enabled;
5. Control of the motor with just differential (using the PID output) but with the same baseline speed;
6. Disable of enable safe mode while the car is running;
7. Change the speed variable without refreshing the actual speed of the motor (Useful if there are still calculations pending and we might not want to commit the calculated speed just yet.);
8. Brake the motors to a fixed global speed already defined in the app\_config.h header;
9. Change the maximum allowed speed (both in normal and safe mode).

Finally, in addition to this, it is possible to reverse the motors by providing a negative percentage speed.

##### Servo driver

The servo is implemented with lower complexity, since it is a much simpler device compared to the motors (from the programming perspective).



This is the only process being executed by the servo driver. It receives an angle that ranges from -90 degrees to +90 degrees to a duty cycle supported by the servo, which is from 6.4% to 9.8% with a PWM frequency of 50.08 Hz.

The API is simply:

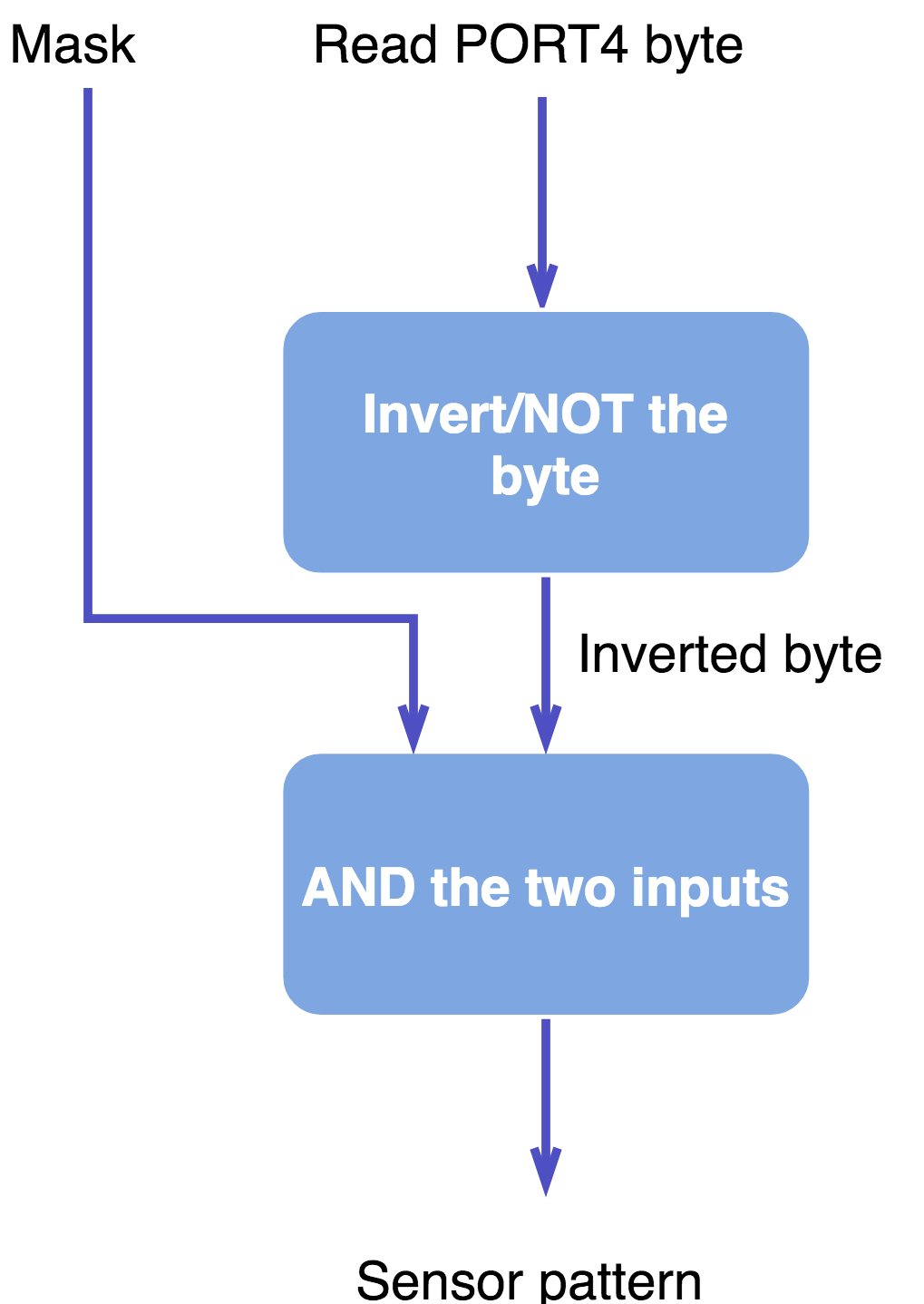
|  |
| --- |
| **typedef** **struct** {  ***spwm\_t*** \* *dev\_handle*;  ***int8\_t*** *angle*;  ***int8\_t*** *angle\_old*;  bool *is\_locked*;  bool *is\_sweeping*;  } ***servo\_t***;  ***servo\_t*** \* servo\_init(**void**);  **enum** **SERVO\_RETCODE** servo\_reset(***servo\_t*** \* handle);  **enum** **SERVO\_RETCODE** servo\_ctrl(***servo\_t*** \* handle, ***int8\_t*** angle);  **enum** **SERVO\_RETCODE** servo\_accum\_ctrl(***servo\_t*** \* handle, ***int8\_t*** sum\_angle);  **enum** **SERVO\_RETCODE** servo\_center(***servo\_t*** \* handle);  **enum** **SERVO\_RETCODE** servo\_lock(***servo\_t*** \* handle);  **enum** **SERVO\_RETCODE** servo\_unlock(***servo\_t*** \* handle);  **enum** **SERVO\_RETCODE** servo\_sweep(***servo\_t*** \* handle, ***int8\_t*** min\_angle, ***int8\_t*** max\_angle, ***uint32\_t*** delay, ***int8\_t*** increment, bool from\_cur\_angle); |

It features:

1. Initialisation and reset of the servo variables;
2. Control of the servo angle;
3. Control of the servo angle by the process of angle accumulation (example: control the angle by setting it 1 degree at a time. It will move 1 degree and this will be accumulated with the angle variable);
4. Centre the servo on angle 0 degrees;
5. Lock the servo in its position - constantly outputs the PWM signal;
6. Unlock the servo - the PWM signal is not persistent. When the angle is changed, it is changed once and the PWM signal is disabled once the servo is in the desired place;
7. Sweep the servo – moves the servo from -90 degrees to +90 degrees. Useful for checking if both the servo and the code is working.

##### IR Sensor driver

The line tracker driver is possibly the simplest piece of code in this project. It reads the PORT4 of the microcontroller, inverts its bits and ‘ANDs’ the result with a mask value.



The output sensor pattern is extremely important, however. It is the only real indication the car has to determine where the line of the track is located at. There are no other visual mechanisms to detect the line, other than predictive calculations and machine learning.

There is one problem, however. We needed to convert a pattern in byte format, which has no meaning and is unusable (in its raw format) into a compatible angle to be used by the servo and motor drivers. The process by which this problem is solved is through a lookup table combined with a PID controller.

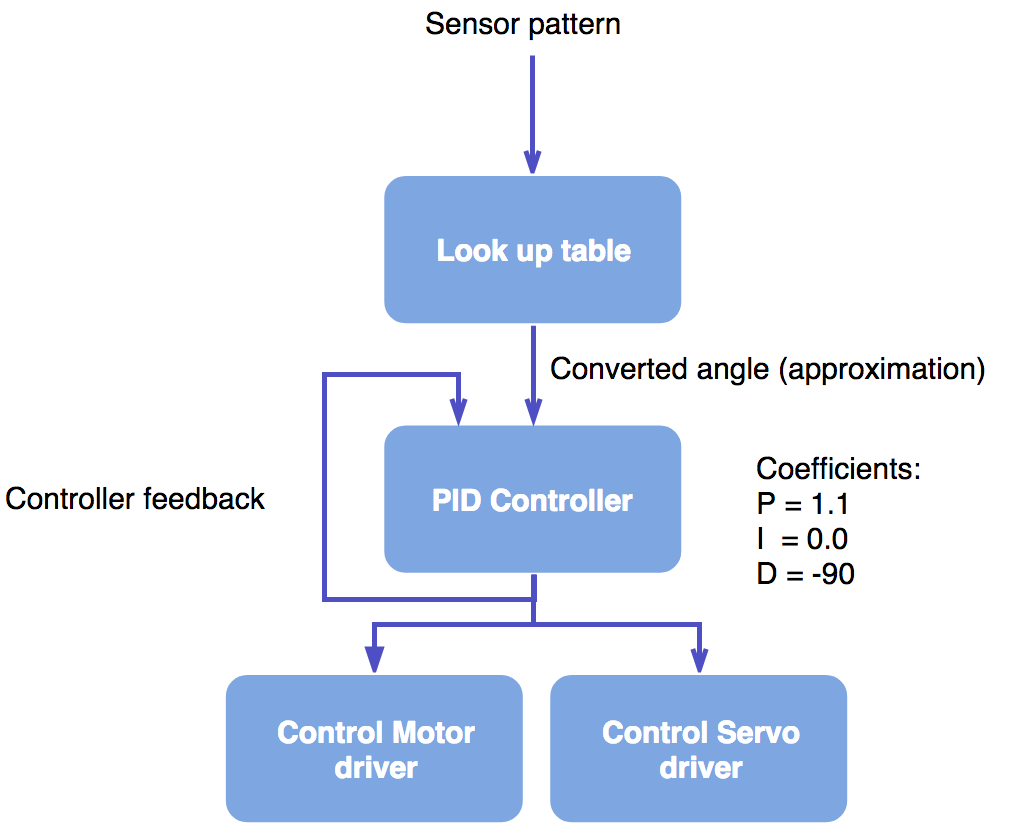


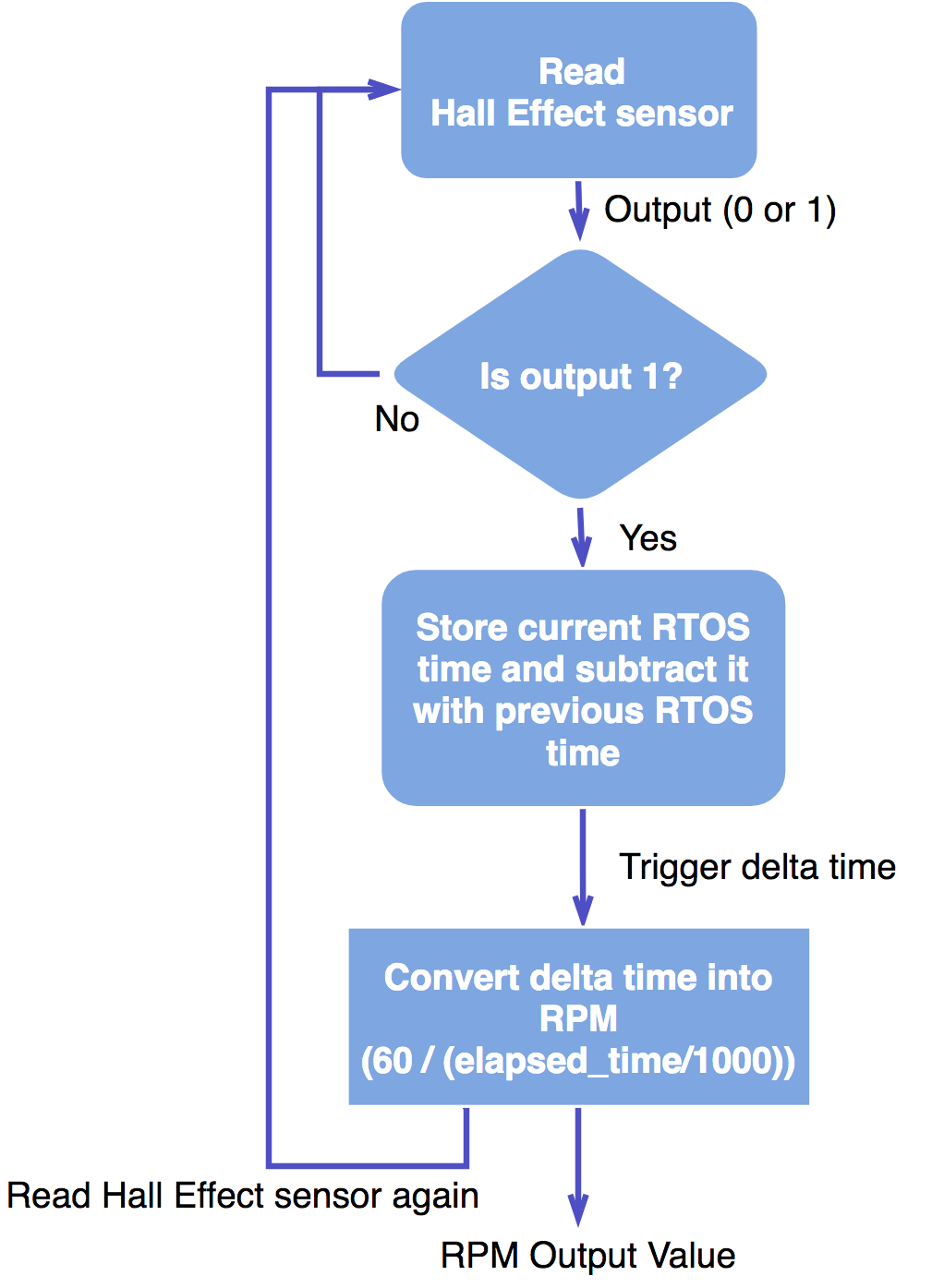
Figure 4‑1 - Implementation of the controller for the car

This is in fact how the core control of the car is implemented. All of the components implemented are used together in this way in the frontend application.

##### Speed sensor driver

One last component that is missing from the Figure 4-1 is the speed sensor driver. The code for this is implemented inside the motor driver, and it is implemented with the help of RTOS’ timing services.

Algorithm:



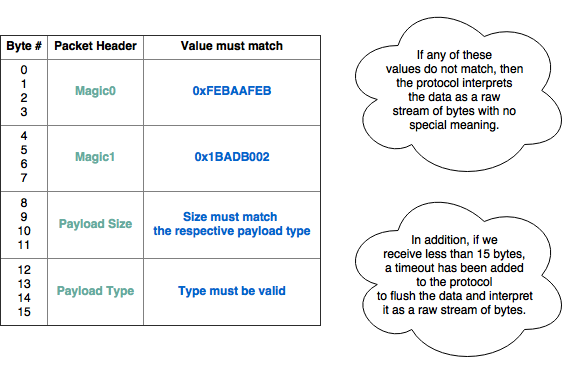
The RPM output value then contributes to the differential algorithm. If the RPM is too high, the differential effect is much greater, since the car would be at a state with large momentum. In this situation, if the car encounters a large angle (i.e. a large difference in the PID output), slowing down the car further to contain the momentum would be absolutely necessary as to prevent the car from getting out of the track.

#### Communication protocol

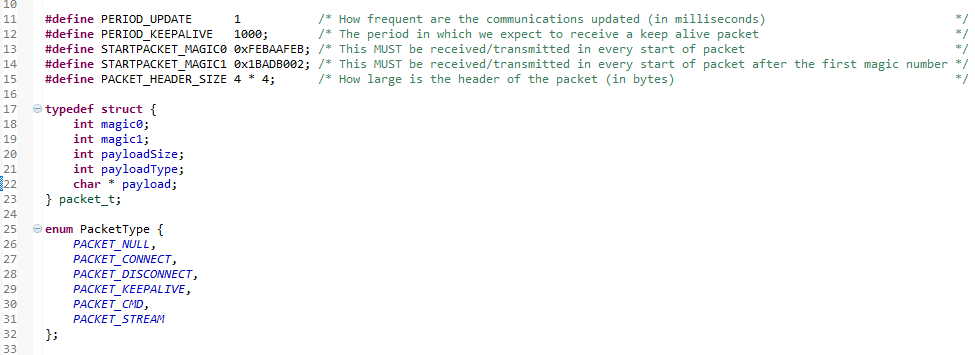
The communication protocol was necessary to be developed because it was required to send data from multiple tasks and from multiple sources. Parsing garbled raw UART strings is extremely trivial, and for that reason, a layered protocol was created.

In essence, it “packetizes” every single piece of data that is transmitted. Each packet is a carrier that holds inside a payload. The carrier contains a header regarding the size and type of the entire packet, as well as the size and type of the payload.

The header of each packet has the following structure:

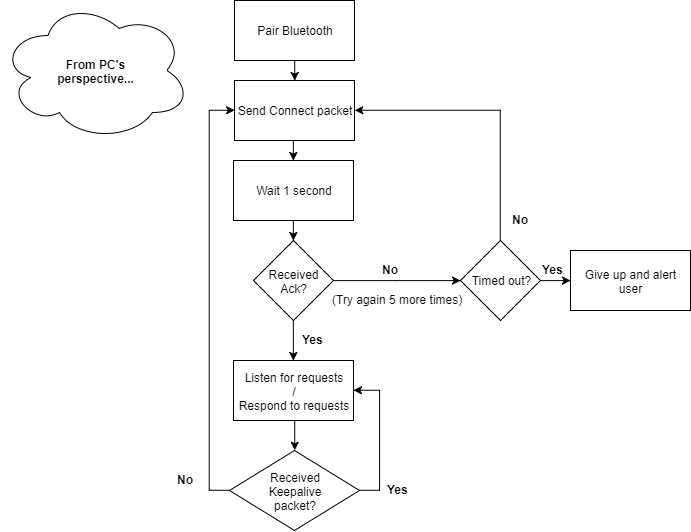


And in code format:



The payload data is consequently appended after this header.

So far this only defines the structure of each packet. It doesn’t explain the algorithm of the communication, how each packet is assembled and how does the microcontroller side keep the connection alive with the computer/smartphone.

The communication behaviour is explained below:

This way, each sensor data can be sent in bundles packet after packet, since it is possible to unpack the payload data by analysing the carrier/packet header. Packet order is not required.

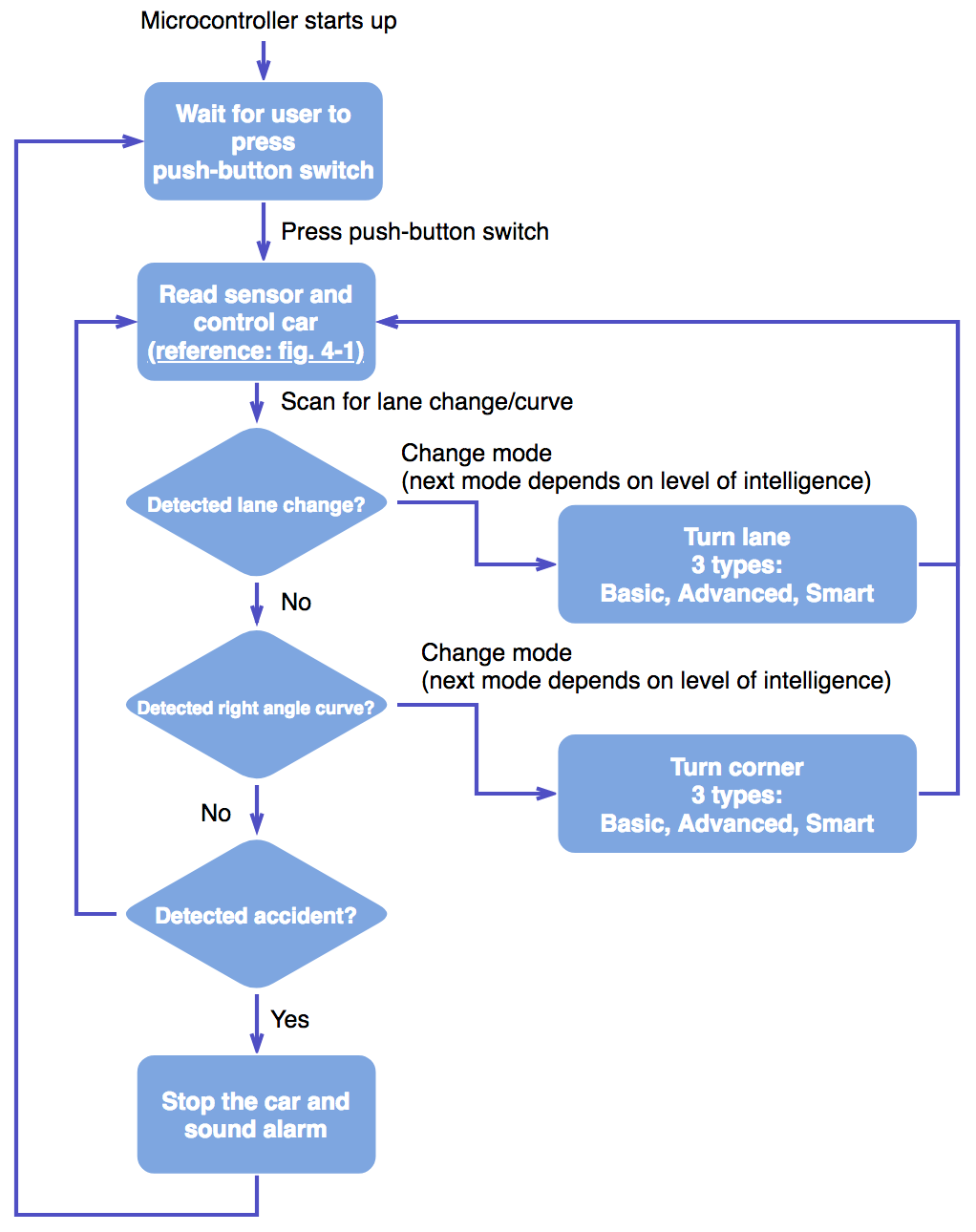
### Frontend

This subsection explains how the code for the frontend application is implemented. This is best done with algorithm flowcharts and diagrams followed by brief and concise explanations that make the diagrams much clearer to understand.

#### User application – car controller

##### Algorithm

The logic of the car controller follows the pattern below:



The ‘Turn lane’ and ‘Turn corner’ blocks vary according to the current level of intelligence. If the car is in basic mode, the car will enter a state named ‘MODE\_TURNING\_LANE’ when it sees a lane change. The full list of FSM states may be found below:

|  |
| --- |
| /\*\*\*\*\*\*\*\*\* ALGORITHM FSM DEFINITIONS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  **enum** **MODE** {  MODE\_NULL, /\* (0) An unknown state that should not be recognized by the algorithm \*/  MODE\_WAIT\_FOR\_STARTSWITCH, /\* (1) Wait for the start switch to be pressed before starting the race \*/  MODE\_FOLLOW\_NORMAL\_TRACE, /\* (2) We are tracing a straight or slightly curved line within controlled conditions \*/  MODE\_AVOID\_RIGHT\_BOUNDARY, /\* (3) The sensor detected a line at the right extreme sensors. We must check if it's a white tape, and if not, avoid it \*/  MODE\_AVOID\_LEFT\_BOUNDARY, /\* (4) The sensor detected a line at the left extreme sensors. We must check if it's a white tape, and if not, avoid it \*/  MODE\_FOUND\_LEFT\_TAPE, /\* (5) We have encountered the white tape on the left side of the track \*/  MODE\_FOUND\_RIGHT\_TAPE, /\* (6) We have encountered the white tape on the right side of the track \*/  MODE\_ALIGN\_BOUNDARY, /\* (7) We are currently aligning the car with the left/right side of the track \*/  MODE\_TURNING\_LANE, /\* (8) We are currently turning the car through a lane change in basic or advanced mode \*/  MODE\_TURNING\_CORNER, /\* (9) We are currently turning the car through a 90 degree corner in basic or advanced mode \*/  MODE\_TURNING\_CORNER\_BLIND, /\* (10) We are currently turning the car through a lane change/90 degree blindly in smart mode \*/  MODE\_ACCIDENT, /\* (11) We have gone off track. Wait until the user puts the car back on the track and presses the button \*/  MODE\_RACE\_COMPLETE, /\* (12) We have completed the race \*/  MODE\_REMOTE /\* (13) The car is being controlled/interacting with the user (not in race mode) \*/  };  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

On a lane change, the car will change to ‘MODE\_TURNING\_LANE’ if on basic mode, ‘MODE\_ALIGN\_BOUNDARY’ on advanced and ‘MODE\_TURNING\_CORNER\_BLIND’ on smart.

On a 90 degree corner, the car will change to ‘MODE\_TURNING\_CORNER’ if on basic mode, ‘MODE\_ALIGN\_BOUNDARY’ on advanced and ‘MODE\_TURNING\_CORNER\_BLIND’ on smart.

The explanation for the states ‘MODE\_ALIGN\_BOUNDARY’ and ‘MODE\_CORNER\_BLIND’ are explained on the following subsection.

##### Levels of intelligence

The level of intelligence controls how the car deals with lane changes and 90 degree corners. It is a setting predefined at compilation time and its definition may be found on the header ‘src/app\_config.h’.

It’s also important to note that the car still follows the line normally whether it is on basic, advanced or smart mode. It is only when the car encounters a lane change or corner that is switches to a new state that depends on the intelligence.

###### Basic

This is the simplest mode. On a lane change, the car slowly drives forward (still reading the sensor and using the PID controller) until it reaches the break of the line. Afterwards, it uses pre-defined data about the track that describes which way the car should turn. This means although this is a basic mode, it still uses track data manually coded on the microcontroller’s memory.

After fetching the turn direction, it drives the servo -+ 45 degrees towards the correct direction, until the sensor detects the line on the other end of the track. When this happens, the car slightly adjusts its position and eases itself in the track. Only after this the car will go back to the normal driving state.

Corners are executed by slowly driving until the end of the corner, and immediately turning the servo to the maximum allowable angle. In addition, the inner wheel will rotate backwards to add angular momentum. As soon as the correct pattern is detected (00011000), the car resumes the normal state.

###### Advanced

Advanced state is very similar to the basic mode with only one difference: the car will align itself with the edge of the track before executes the turn.

For instance, on a lane change to the right, the car will first align on the right side of the track before it even reaches the break of the line. Similarly, if the lane change is to the left, the car will on the left edge of the track.

Corners are performed the exact same way. The car will first align itself on the right edge if the corner is to the left, and will align on the left side if the corner is to the right.

This technique should mimic that of formula 1 racers. The car aligns itself on the track before reaching the apex of the curve.

Unfortunately, the size of the track and the car and the momentum do not favour this technique that well. It is extremely difficult to align the car on time when it is driving at high speeds.

###### Smart

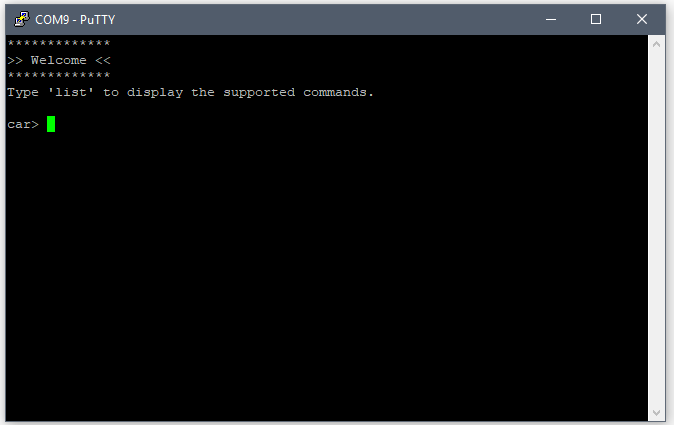
Smart mode is a very complex level of intelligence that essentially plays back computer generated (or measured otherwise) track data. On a lane change, the sensor data comes from the microcontroller’s memory rather than the physical device. This way, the car can continue driving itself easily without braking.

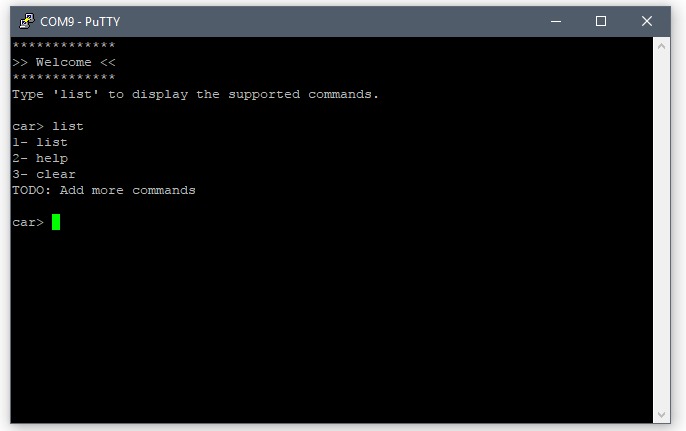
Reliability is a big issue here. There is no guarantee that the car will always end up in the same place 100% of the time, due to the nature of the mechanical parts and the track itself. For this reason, this mode has been deactivated (although the implementation is still there).

#### User interactivity – command shell and Android App

Interactivity was an extremely useful feature that had to be implemented in order to change the state of the car in real-time (i.e. change the PID constants, change the maximum allowable speed, etc…) and to analyse its status to allow debugging.

The first method of interactivity (excluding physical interactivity – pushing the start button) is through a command line, similar to a Windows/Linux command line.





It is implemented using the scanf function, which reads the user input until the enter key is pressed. The typed string is then parsed and the respective command is executed.

The screenshots above do not show the full list of commands implemented to date. The actual complete is:

|  |  |  |
| --- | --- | --- |
|  | Command name | Description |
| 1 | **help** | Shows the list of supported commands. |
| 2 | **clear** | Clears the command line’s text. |
| 3 | **reset** | Resets the microcontroller’s CPU. |
| 4 | **g** | Kick-starts the car’s algorithm. Equivalent to pressing the start button. |
| 5 | **s** | Stops the car after it was kick-started. Can resume by either running command ‘g’ or pressing the start button.  (Note: the status of the program is preserved after running this command.) |
| 6 | **speed** | Change the maximum allowable speed. |
| 7 | **motor** | Force speed on a given motor. |
| 8 | **servo** | Force angle on the servo. |
| 9 | **sweep** | Test the servo by sweeping it from -90 degrees to +90 degrees. |
| 10 | **p** | Change the P coefficient on the PID controller. |
| 11 | **i** | Change the I coefficient on the PID controller. |
| 12 | **d** | Change the D coefficient on the PID controller. |
| 13 | **iw** | Change the Integral Windup period on the PID controller. |
| 14 | **play** | Play a tone on the piezo buzzer. |
| 15 | **rcmode** | Enable/disable RC (Remote Control) mode.  (Note: this feature is not implemented.) |
| 16 | **w** | Write to memory by using the bootloader code.  (Note: this feature is extremely sensitive and for reasons that are not related to the algorithm of the car, it has been disabled on the header ‘src/app\_config.h’.) |
| 17 | **l4** | Periodically log momentum data into the console. |
| 18 | **l3** | Log any unrecognised pattern detected on the IR sensor. |
| 19 | **l2** | Periodically log the speed (RPM) of the motors. |
| 20 | **l1** | Log the state of the algorithm of the car (only when it changes). |
| 21 | **l** | Periodically log the PID values of the PID controller. |

Table 4‑1 - List of supported commands

In addition to having developed a command line, an Android application was also planned to be developed. It only stayed in its early stages, however, it was capable of exchanging packets with the microcontroller and kept its connection alive.

Data was also sent from the microcontroller to this application, however, there was not enough project time to further develop this.

The early stages of the app can be seen below:

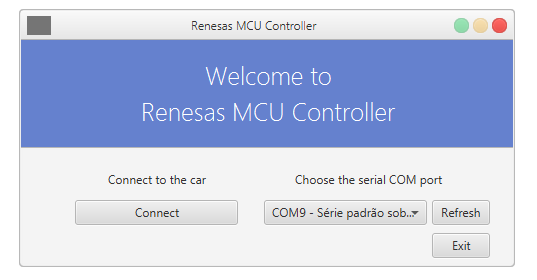


Figure 4‑2 - Android app: connecting to the car

This is the first screen the user sees. It allows the computer to pair with the Bluetooth module and will consequently perform several handshakes in order to obtain and maintain connection using keep-alive packets (which are sent every second).

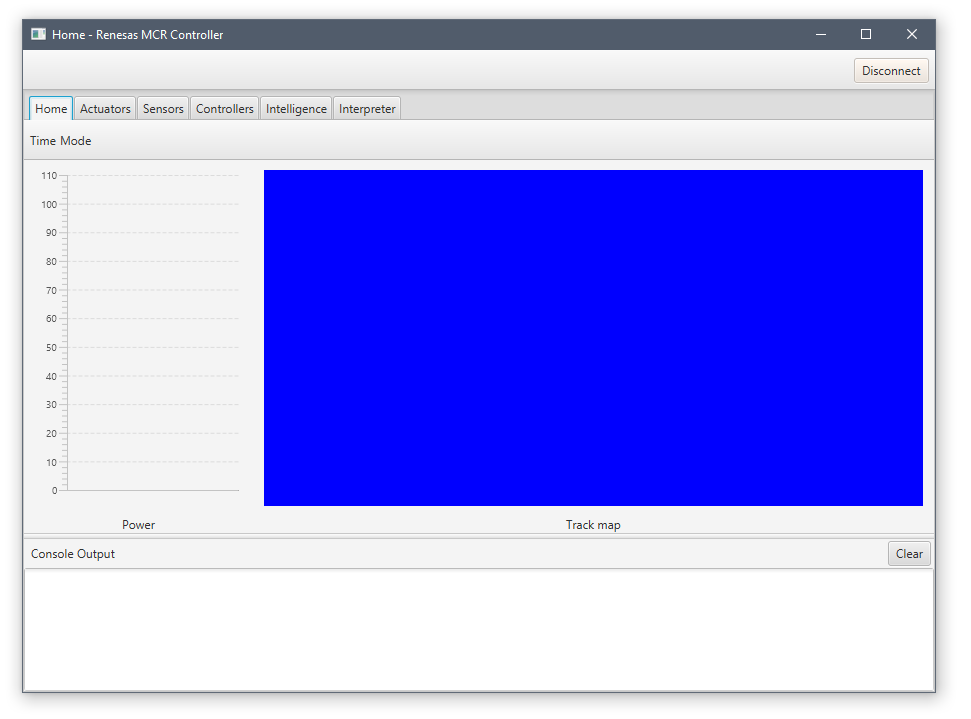
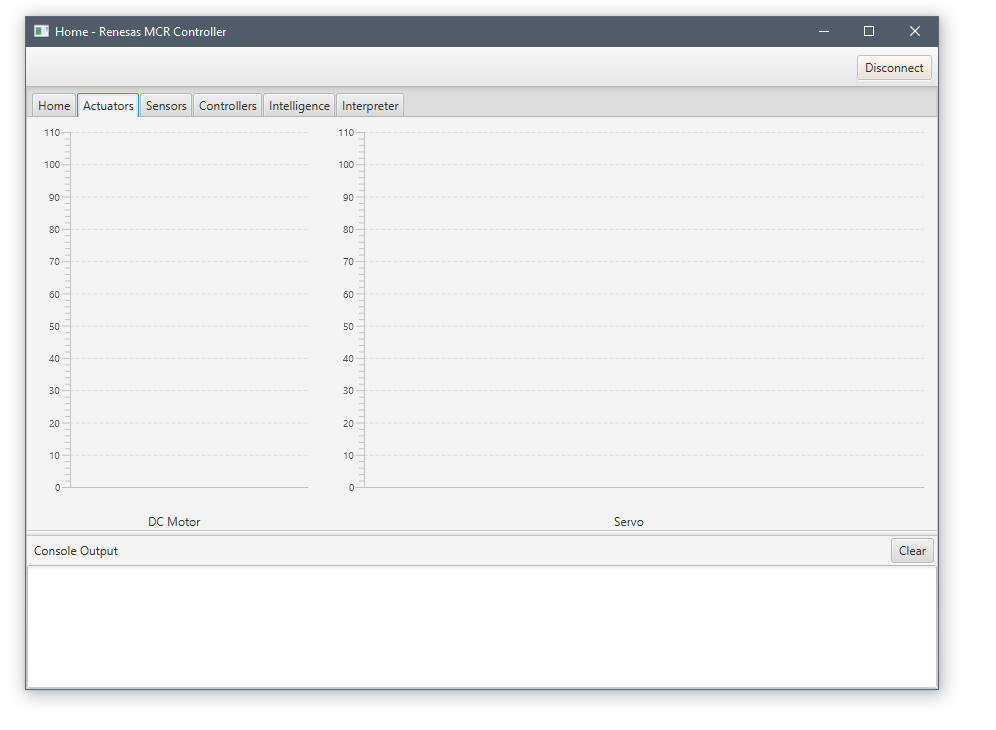
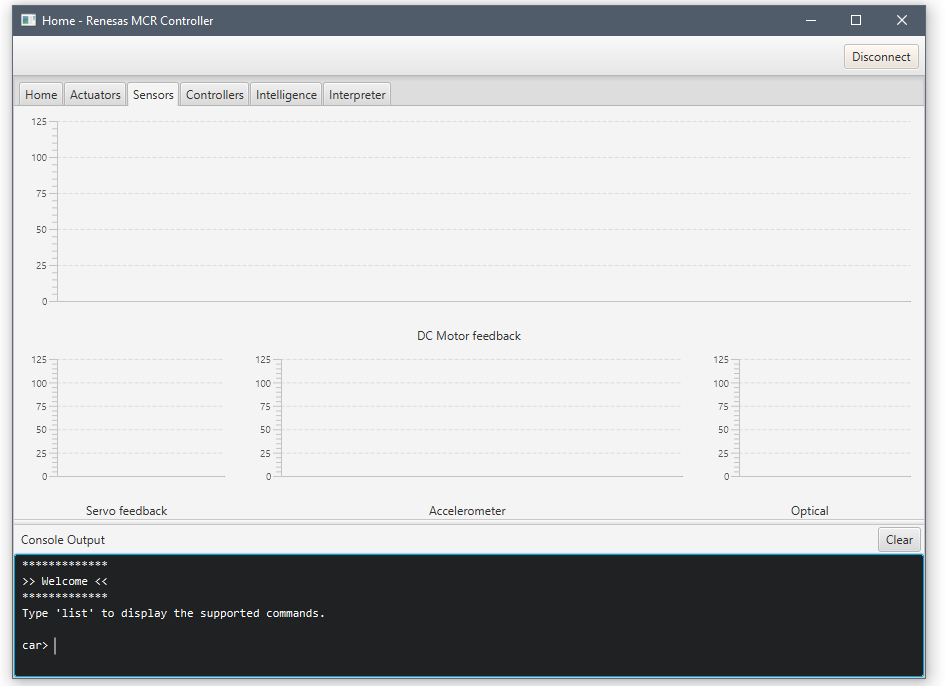


Figure 4‑3 - Android app: the home page

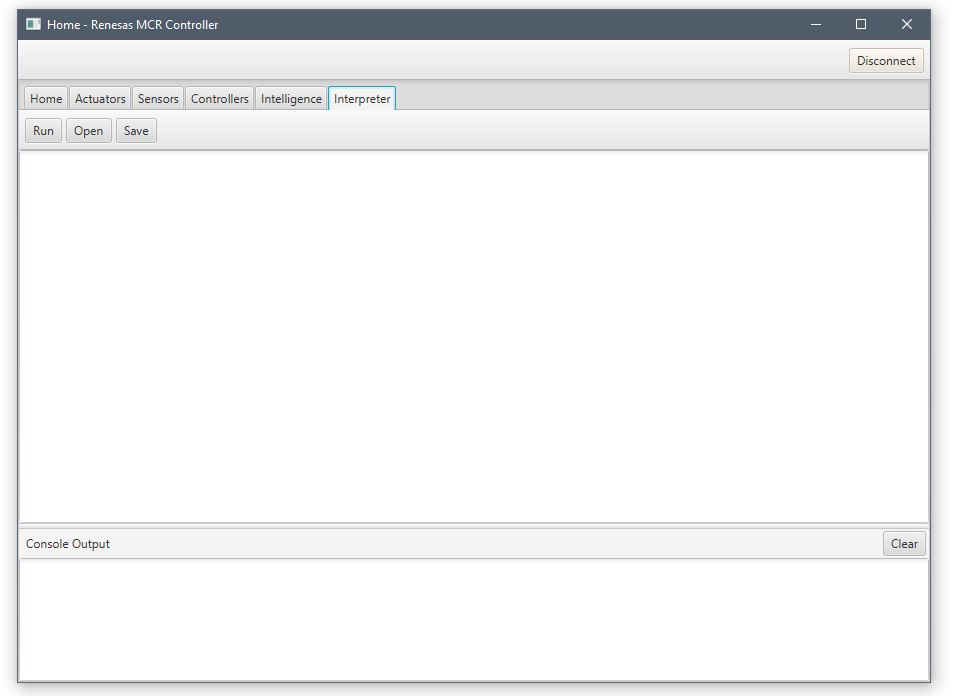
This screen shows that the VCC power was planned to be plotted as time went by. The blue image on the right is a placeholder for a canvas that would draw the track in real time.



Here we see two graphs that would plot the speed of the motors and angle of the servo.



On the sensors tab we may find a graph that would plot the measured RPM of the motors, the feedback of the servo angle (this was a desired feature on the early stages of the project), the accelerometer’s X and Y outputs and the measured pattern of the IR sensor. Also note down at the bottom we are still able to type commands just like previously.

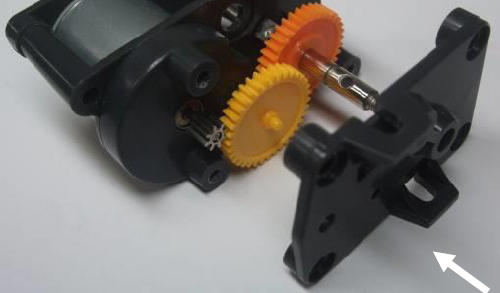


Finally, this tab would be dedicated to command automation. It would allow us to type a list of commands in a text-editor and would consequently send the commands in a burst. In a way, this would act as a macro.

# Testing

## Mechanical

Mechanically there was little to test, however the endurance of the car had to be ascertained so that the car could handle to trip to Germany. The car was set to run at full speed for five minutes to make sure that the gearboxes functioned well at highest stress. This revealed a critical problem. As the gears spin at high speeds from a prolonged time, the nib of the gear wore away the hole it sits in the housing (point A), causing the gears to fail.



This was fixed by filing down that part of the housing so that the nib pokes all the way through, stopping it from wearing away the plastic.

Other than this there were no mechanicals problems. The design was sturdy and could take minor knocks without being damaged. The Velcro functioned very well in allowing the batteries to be removed quickly for charging.

## Electrical

### Sensor board

<NOTE: KAYIN, WRITE YOUR CONTENT HERE>

### Power board

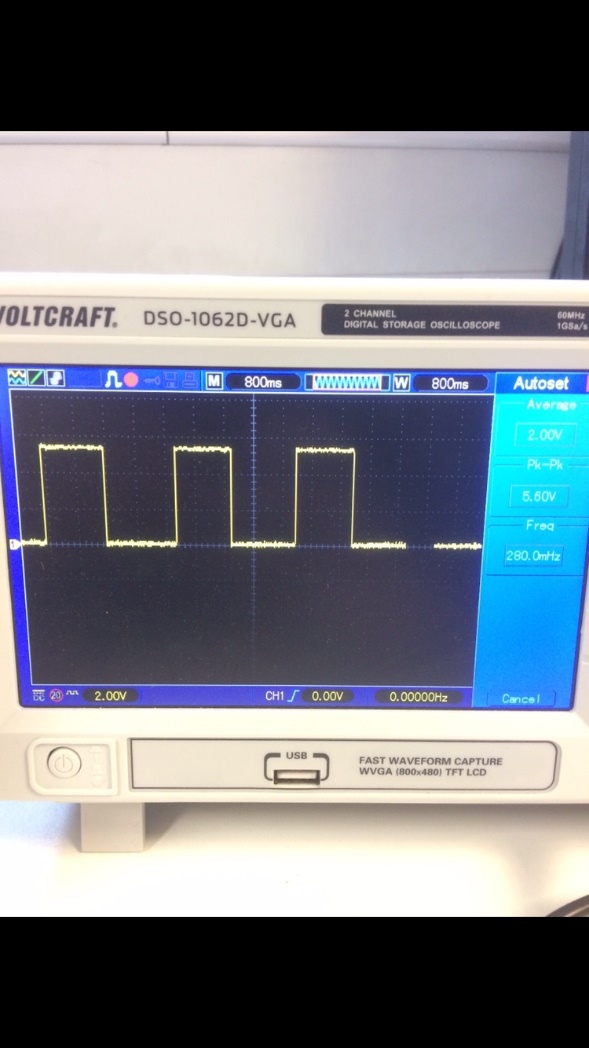
The power board was tested in a few ways. The first test involved connecting the power supply and checking the output voltages from the regulators. The 5V regulator for the MCU board maintained a constant voltage of ~4.9V, which is perfectly accurate for this purpose.

The adjustable regulator was tested with a multimeter and adjusted until the output was 6V. This was checked regularly throughout the project so as to be sure that the potentiometer hadn’t been accidentally turned.

Lastly the output signals are checked to see that they operate as intended. The signals do indeed line up with those mentioned in the design.

### Hall Effect Sensors

The first test that was conducted on the Hall Effect circuits, was to move a magnet at a distance of ~2.5cm to see that the sensors were working correctly. This was a success, and the signal switched between 5V and 0V when exposed to magnetic south and north respectively.

The next test involved monitoring the sensor signal as the wheel is rotated by the rear motors. For this test the motors where disconnected from the power board and connected to a variable power supply with a PWM output. The actual speed of the motor was read with a laser tachometer.

The results of these tests can be seen below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Motor PWM (%) | Motor Speed (RPM) | Hall Effect Frequency (Hz) | Hall Effect Speed (RPM) | Error (%) |
| 20 | 1906 | 31.4 | 1881 | 2 |
| 30 | 2894 | 47.5 | 2851 | 1 |
| 40 | 3832 | 67.2 | 4032 | 5 |
| 50 | 4813 | 81.6 | 4896 | 2 |
| 60 | 5770 | 98.9 | 5933 | 3 |
| 70 | 6721 | 106.4 | 6384 | 5 |
| 80 | 7692 | 125.4 | 7526 | 2 |
| 90 | 8637 | 145.4 | 8726 | 1 |
| 100 | 9602 | 160.8 | 9650 | 0.5 |

The Hall Effect error doesn’t seem to exceed 5% error, making it accurate enough for this system but could be made better, possibly by adding a second pair of magnets to each wheel.

### Peripheral board

Testing the accelerometer was difficult to do accurately, as it is hard to expose the car to forces while testing it. To confirm that it was working, the accelerometer’s Y axis pin was connected to an oscilloscope. Then the car would be moved in that axis with different forces to confirm that the signal did change proportionally to acceleration.

When stationary, the PWM signal had a duty cycle of 50% as expected. When the car was swiftly moved forwards this signal would increase to ~60% and when moved backwards this would decrease to ~40%. This shows the proportionality and a measurable amount of change in the signal, as it would be accelerating and decelerating with greater force when under its own speed.

The potential divider was also tested with a voltmeter and was 3V with a battery voltage of 9V, showing that the voltage feedback does indeed work.

## Software

In order to test the software effectively, the command line should be used. The only other way of testing is by observing the behaviour of the car. For this reason, the testing was solely based on the interaction through the command line interface.

First, in order to perform the testing, the Windows/Linux program ‘PuTTY’ must be downloaded. This allows us to connect to the Bluetooth module (HC06) via a serial COM (communication) port through UART.

|  |  |
| --- | --- |
| Test # | Observation/Execution steps |
| Test 1 – User can input data and receive data from the Bluetooth module. | The image below proves that the user can receive and send text through Bluetooth:    Table 5‑1 - Testing the UART reception and transmission of data |
| Test 2 – Analyse the momentum of the car. | **How to**: run the command ‘l4’ and the program will periodically output the momentum counter, the simulated momentum trigger counter and the simulated momentum enable flag. This ‘simulated’ momentum replaces the accelerometer and the explanation for this decision can be found on section **7. Problems and solutions**. |
| Test 3 – Drive the car and detect any anomalies on the sensor board. | **How to**: run the command ‘l3’ and the program will output any unrecognized sensor pattern. This will allow us to determine if there is a hazardous pattern that could be causing the car going off-track. |
| Test 4 – Analyse the current speed of the motors. | **How to**: run the command ‘l2’ and the program will periodically show the output of the PID controller, the RPM of the two back wheels and whether the hall effect sensors are on or off. It may be used to analyse how much the car is braking and with what frequency. If the car is going too slow on a straight line it may be that either 1 – the PID controller has gone unstable and the gradient calculation speeds are constantly slowing down the wheels unnecessarily, or 2 – the car is braking. |
| Test 5 – Analyse the current FSM state of the car controller. | **How to:** run the command ‘l1’. This will show the present FSM state of the car algorithm, however, it will only output this when it changes from one state to another. |
| Test 6 – Analyse and plot the PID controller’s variables. | **How to:** run the command ‘l’ and the proportional, integral and derivative variables will be constantly outputted. The PID output, feedback, error and last error are also dumped by this command.  If the user desires, this data could be plotted on a graph in real-time, however, this feature was not fully implemented on the Android app for timing reasons. |

Table 5‑2 - Software testing methodology

# Performance

Despite all of the resource limitations, it was possible to fit the RTOS, all of the drivers, the track data and all of the configurations into only 16 KB of RAM memory.

The final size of the code and program data can be seen using the IDE e2studio.



Figure 6‑1 - Performance: memory usage

We see that the memory usage is:

1. Program size: **50,820 bytes (50,82 KB)**
2. ROM usage: **58 KB / 256 KB (22 %)**
3. RAM usage: **9 KB / 16 KB (58 %)**

Considering the amount of implemented features, this is extremely positive for the performance of the car from the perspective of memory resources.

In regards to the speed of the car…

<TODO: THOMAS>

# Problems and solutions

<NOTE: KAYIN, WRITE YOUR CONTENT HERE>

# Next year proposals

Although we did not compete this year, and thus, we did not get the chance to benchmark the car on the real track, there are still some improvements that could further optimise the car.

Some of the proposals for next year’s team:

1. **Reduce the height** of the car even further, which may help with the stability. (Hint: compare the structure of the car with that of the Formula 1.)
2. **Make the car as light as possible**. This is critical, as it builds up momentum and causes the car to get off-track. The weight is a much greater problem when the low torque gears are being used.
3. **Shave the outer part of the gearbox that touches the tip of the yellow/red** gear. This will prevent the gear from drilling and destroying the gearbox. (Note: mind that this issue may only be observed if the car is set to 100% speed for a certain amount of time. In any case, it is highly recommended to perform this step.)
4. **Properly utilise the accelerometer** to slow down the car’s momentum with full accuracy.
5. **Be mindful about what gears you choose.** The yellow gears provide higher speed but less torque. This means the car will easily get out if there is a lot of weight. On the other hand, if you use the red gears, you will obtain lower maximum speed but higher torque. The stability will be considerably better, especially during braking. These are trade-offs have to be weighed against one another.
6. **Complete the Android application** and add more features to it.
7. **Implement Remote Control mode**, which allows the user to take manual control using a computer keyboard or a joystick.
8. **Develop an auto-tuning PID** controller algorithm in order to obtain the ideal coefficients.
9. **Use dynamic PID** as to allow the coefficients to change according to the detected pattern.
10. **Use neural networks** in order to allow the car to drive itself even without the IR sensor.
11. **Implement computer simulations** that approximate the behaviour of the car in order to test a certain PID setting without having to test it in real life.
12. **Add a feature on the Android app that allows us to design our own track**. The car will then follow this invisible track that we design ourselves. If this turns out to work successfully, it could be used to draw the real track, however, it would be susceptible to a lot of off-track accidents. (Perhaps a solution to this is to add an extra feature that keeps reading the sensor and prevents the car going off-track even though it is still replaying the computer generated/designed track data.)
13. **Do not change any mechanical feature more than once**. Adding a yellow gear first only to have it changed to a red gear later on is only the best way to get yourself in trouble. The reason for this is the code will most certainly have to change as you modify the hardware. New measurements have to be made in order to determine what the new best PID coefficient is, and this is extremely time consuming and inefficient.
14. **Make sure the batteries are fully charged**. Far too often the batteries were at 50% charge without us realising it. Basically, when the PID is tuned to a certain coefficient that value will only work ideally with the current battery level. As soon as the battery lowers its charge, the PID coefficients may not work as well as before, thus requiring a re-tuning of the PID controller.

# Discussions

According to the measured performance, there is a great chance that this year’s USW team could end up in second place if only it had competed in the real race.

Let’s consider that:

1. The motors were limited at 80% duty cycle during the demonstration;
2. The high torque gears were limiting the maximum achievable speed;
3. The weight certainly does not favour the stability (extra momentum = harder to control);
4. The motors brake constantly due to the impossibility of measuring momentum. Using approximated gradient values for the braking is not the best way to stabilise a system;
5. The tests and measurements were mainly done with half charged batteries. This makes an immense difference that might not be noticed at first, but will certainly affect the performance from 100% down to ~50%.

And we still achieved a satisfying result.

We can safely say that there is a great potential next year to reach second place and even possibly first place. This will definitely be possible if all of the proposals for next year are followed and all conditions are ideally met, especially on the time schedule.

# Conclusion

Looking back at the project objectives, it is fair to say that the desired set-goal has been accomplished in the end. Despite all of the issues, we still managed to:

1. **Develop a better program on the old car itself**;
2. **Build a new (better) car compared to last year’s team**;
3. **Improve and implement optimised algorithms for the algorithm of the car**. This was achieved thanks to the levels of intelligence.

The proper implementation of a PID system and the finer adjustment of the lookup table that converts patterns into angles has also show great improvements over last year’s car.

It has been a “bumpy ride” since the start of the project. Countless issues were encountered and immediately solved afterwards. This proves we had the opportunity to develop very strong problem-solving skills, which is without a doubt a much required and desired skill on the real-world.

In the end, it can be said that the project gave us the opportunity to enrich and strengthen a wide set of engineering skills. It put in practice the theoretical knowledge and made abstract/ideal concepts such as PID theory (which is at first, developed on paper) into a concrete system that can be observed and is tangible (through the car’s behaviour).

# References

Appendices

Appendix A:

Appendix B:

Appendix C: