The University Of Sheffield.

Department of Electronic and Electrical Engineering

NXP Cup

Autonomous Vehicle Racing

Group 9

Electronic and Electrical Engineering, MEng

Students: Owen Slack, Ben Trevett, Ahmad Zainual

Date: 06/05/2016

Supervisor: Dr. Daniel Gladwin

Second Marker: Prof. David Stone

Abstract

The NXP Cup is an international competition between university students with the goal to build an autonomous 1/18th scale vehicle that will follow a track within the fastest time. This is the University of Sheffield's third attempt at the competition, with the team aiming to improve on last year's performance and qualify for the EMEA finals. The group was able to achieve this by placing third in the UK qualifiers.

Vast enhancements were made over last year's progress: a custom PCB was designed and fabricated, improving on the previously used FRDM board and TFC-Shield; torque control was implemented; a new camera mounting system, offering improved stability, was implemented; the servo mounting was improved; visual feedback was upgraded by the addition of an LCD display; two-way telemetry was added for improved tuning; an accelerometer was used for bump detection as well as well as numerous mechanical and code improvements.

Contents

1 Introduction

Since the creation of the automobile in the late 1800's, manufactures and engineers have made huge advances in the way people conceptually think of a vehicle and how it works. In more recent years the latest of these advances has been self-sufficient driving cars, with the likes of Google's self-driving car project being at the fore-front of this movement. Human error contributes to 90% of all traffic accidents worldwide, therefore self-driving cars would remove the human error aspect from these accidents. This revolutionary step in vehicle technology could result in everyone being able to safely travel, despite the driver's ability to drive.

Autonomous design can be implemented in a variety of different applications. One example of this is Unmanned Aerial Vehicles (UAVs) which are used in situations where human life may be at risk, such as war zones or forest fires. Ideas and concepts developed for UAVs are not strictly limited to just UAVs, as UAVs use range sensors such as radar, sonar etc, for locating and mapping areas. These sensors can be directly used on an autonomous vehicle for detecting other vehicles passing by and used alert a driver to their presence if changing lanes on a motorway, therefore reducing the likelihood of a collision. Therefore, rapid developments in one field can lead to advances in another.

NXP Semiconductors, previously known as Freescale Semiconductors, were motivated by these developments and created the NXP Cup in 2004 [1]. This competition challenges students from across the world to design and build a 1/18th scale autonomous vehicle. This vehicle has to navigate a track that has been randomly-assembled in the fastest time possible. The vehicle navigates the track using feedback from optical cameras and various sensors which relays information back to the vehicle's MCU, which in turn drives a steering servo and two motors. All parts of the vehicle have to comply with the 2016 EMEA rules and regulations.

2016 will mark the University of Sheffield's 3rd attempt at the NXP cup. This report will discuss the changes and improvements made from last year's vehicle, Freescalextric, to this year's vehicle, Hotrod, as well as future recommendations and improvements for next year's team.

1.1 Competition Overview

The NXP Cup races have the following format; team vehicles were selected in a random order to attempt the race track, each team had 3 attempts at navigating the track successfully. The first successful lap stands and that lap time is recorded. Any remaining attempts are forfeited. The NXP Cup is split into 3 rounds: national qualifiers, regional finals and worldwide finals. The 1st, 2nd and 3rd place from each qualifier progresses to the next round. There were more UK teams competing in the NXP Cup compared to previous years, therefore the national qualifier was held in London, UK, at Imperial College London.

The track consisted of several sections including 90 degree corners, straight lines, chicanes, hills, bump sections, cross sections and a start/stop line. The track itself is constructed out of 1/16" ABS white plastic sheets with a 1" thick black line on both edges for the car to track [2]. The qualifying competition track at each round is not revealed until the actual race day.

NXP enforces strict rules over the vehicles in order to make the competition fair [3]. Each team was provided with the same standard vehicle kit, with only certain modifications permitted. The kit includes a vehicle chassis that can be remodelled provided it stays within the specified dimensions of 250mm/9.85in (W) x 400mm/15.75in (L) x 305mm/12in (H) and that the wheel base spacing does not change. The kit also provided two standard DC brushless motors and a set of tyres which could be changed as long as the replacements have the same specification and are approved by NXP officials.

During the race, the vehicle has to be powered by a single 7.2V, <=3000mAh NiCd, NiMH or Li-Ion battery. The vehicle was also not allowed a DC-DC boost that exceeds the voltage of the battery. The vehicle can only have one processor, which has to be a standard NXP 32-bit MCU. Each vehicle was allowed up to 16 sensors and finally the total capacity of all capacitors on the vehicle cannot exceed 2000uF.

2 Project Aims

The project aim was to further the developments made by last year's team by taking their recommendations into consideration. Last year's vehicle did not perform well at the qualifiers in Germany last year and did not successfully complete a lap during qualifiers. Therefore, they did not qualify for the EMEA finals. Unfortunately, the vehicle also did not work when initially tested at the start of the project. Therefore, the main aim of the team this year was to get the vehicle working consistently, to improve upon current existing code and to address the problems faced by the previous team. This should result in a more robust vehicle and allow Team Hotrod to produce a competitive lap time at the national qualifiers and qualify for the EMEA finals.

2.1 Starting Points

The team was provided with group 2's Freescalextric vehicle and code from last year. Group 1's code was also provided. As the previous year were the first teams to race on the double black line track, there was already an existing code for the majority of the algorithms on which this year's team could build on. However, as the vehicle provided was not as successful as the vehicle provided to the other team this year, work was needed to bring it up to the same level of performance. Therefore, an important task was to review group 1's code and implement this code on group 2's vehicle.

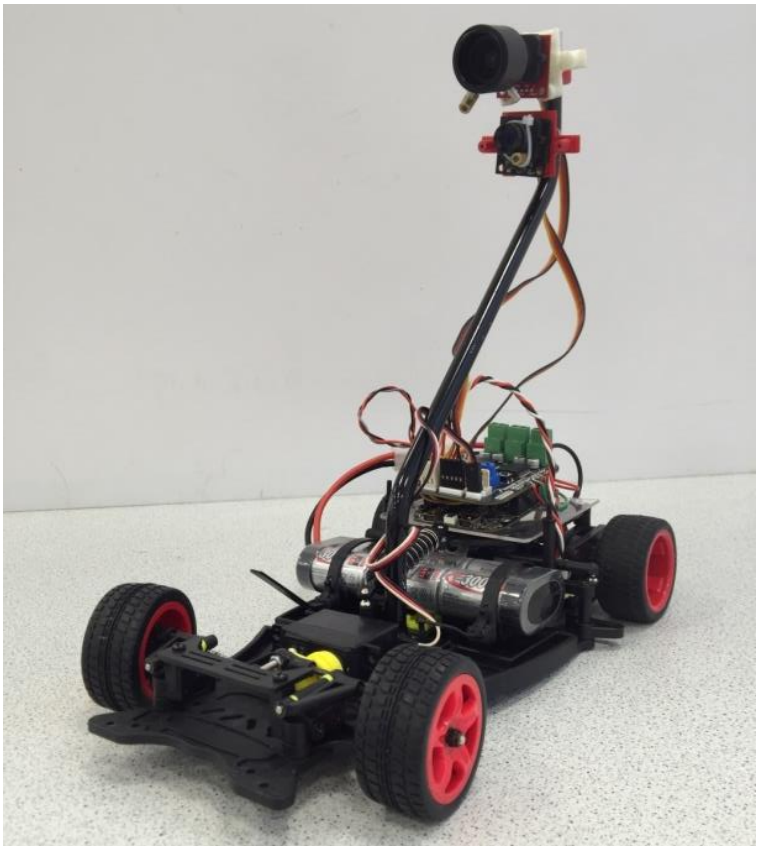


Figure 2.1.1 – Freescalextric, Group 2's Vehicle

From a review of both group's reports from last year, there were numerous recommendations for issues faced and ways to improve the vehicle's performance. Firstly, chassis design and camera mounting, both teams identified the significance in reducing the overall weight of the chassis to increase the vehicles overall performance. They also expressed the importance of a secure camera mount that reduces vibrations and movements as well as being able to allow easy adjustments to camera height. They stressed that changes to the weight significantly affects the overall race outcome.

Both groups suggested re-designing and creating a custom TFC-Shield. This was because the TFC-Shield wasn't very flexible when it came to pin configuration and created problems when connecting multiple modules to the same pins. They also stated that a custom board would allow additional hardware peripherals, such as LCD screen, to be added for debugging and tuning the vehicle. Another issue they found with the TFC-Shield was that it was very difficult to get accurate, noise-free current readings from the TFC-Shield, due to noise present on the current feedback line from the motors. Therefore, they were unable to obtain accurate current readings useful for implementing torque control.

Another recommendation was to implement 2-way telemetry using a Bluetooth module. Both teams had already implemented a Bluetooth communication that allowed them to receive data from the car for tuning. They stated how two-way telemetry could speed up testing, as the vehicle would not need to be connected to a PC to be reprogrammed.

2.2 Work Plan

The work load was shared out between the team members. Ahmad concentrated on the mechanics and speed control, Ben on the system architecture, torque control and the telemetry and Owen on the hardware design of a custom PCB. Each team member helped each other out, therefore the majority of the tasks were completed as a team effort.

Tasks	October	November	December	January	February	March	April	May
Initial Research								
Review Previous Reports								
Hardware & Software Review								
Initial Development								
Troubleshooting Vehicle								
PCB Schematic Creation								
PCB Layout Creation								
Verifying & Ordering PCB								
Advanced Development								
Implementing LCD screen								
Improving Telemetry								
Implementing Live Camera Feed								
PCB Fabrication								
PCB Testing								
Custom PCB Mounting								
Improving Camera Mounting								
Improving Line Detection								
Improving Stop Line Detection								
Implementing Accelerometer								
Implementing Torque Control								
Improving Speed Control								
Improving Servo Mapping								
Tuning								
Races								
Qualifying								
EMEA Finals								

Table 2.2.1 – Project Work Plan

3 Car Chassis and Mechanics

Figure 3.1 – Hotrod Final Design

In order for the vehicle to perform better than the previous year, mechanical changes were needed to be made to increase both the stability and the performance of the car during cornering, braking and accelerating.

One of the major changes was to the servo positioning and arms. Last year's team used the stock servo arm provided by NXP, which was not rigid as it was made from a plastic material and limits the servo linkage arm's length. Therefore, an aluminium servo arm was fitted to increase rigidity. A strong servo arm was needed as the servo operates at high speed during sharp corners. The new servo position allowed for longer servo linkage arms to be fitted to increase front wheel turning speeds. A custom aluminium strut was made from a 1mm thick aluminium plate to increase the length of the servo linkage arms. This increased the length of the servo linkage arms by 2cm.

The previous group used a heavy steel camera mount mounted directly to the chassis. The camera end of the mount sat above the rear of the vehicle to provide a wider field of view, enabling the camera to detect both lines on the track. It was determined that the weight of the camera mount was not evenly distributed and the base of the camera mount was not mounted on the centre of the chassis. This resulted in an unstable camera mount which led to inaccurate images from the linescan cameras. Hence, a custom camera mount was made. To be able to use the custom camera mount, two extra holes were drilled on the chassis to make room for the base of the camera mount. The purpose of these holes were to maximize the base area of the camera mount without affecting the servo setup and thus, increasing the stability of the camera mounting.

For improved performance of the vehicle, all wheels needed to be at optimal condition. The original wheels provided for the vehicle were all of different levels of quality. This resulted in an uneven performance between left and right cornering. Brand new wheels were used for competition races, and spare wheels used for testing. Since the car was rear wheel drive, the back tyres were glued to the back rims to reduce deformation during cornering and to ensure that the tyres did not come away from the rims during acceleration. The front wheels performed better if they are unglued because glued front wheels caused the car to drift during cornering which reduced the cornering speed and increased the possibility of the vehicle leaving the track. The wheels were also cleaned as often as possible to ensure that there was no dust on the tyres as this reduced friction. It was also found that professional RC cars use Simple Green, an all-purpose spray, temporarily increased the grip of the tyres [4]. It was found that for best results, Simple Green should be applied to each tyre before the race and given time to dry.

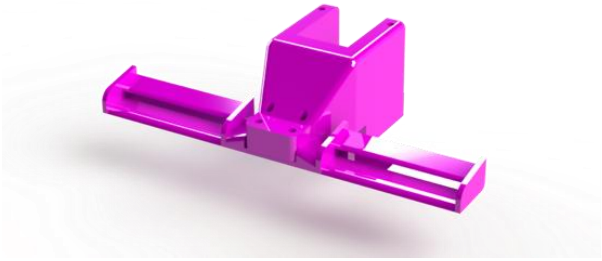


Figure 3.2 – 3D Model of Servo Protector

The servo motor housing often broke during collisions when testing. Spare servos had to be imported and took a long time to arrive. To prevent the servo casing from breaking and causing a delay in testing, a 3D printed servo protector was built. It was designed to fit over the servo and provide protection, as well as allowing a front spoiler to be fitted. A 3D printed battery case was also designed and built in order to mount the battery at the back of the car on top of the rear wheels. This was created due to the idea that adding weight at the rear of the vehicle would improve performance by increasing friction during cornering. This shift in weight required the control of the vehicle to be re-tuned. As time was limited, the vehicle was not optimally tuned for the battery at the back of the vehicle, therefore for the competition races the battery was mounted in the middle of the vehicle.

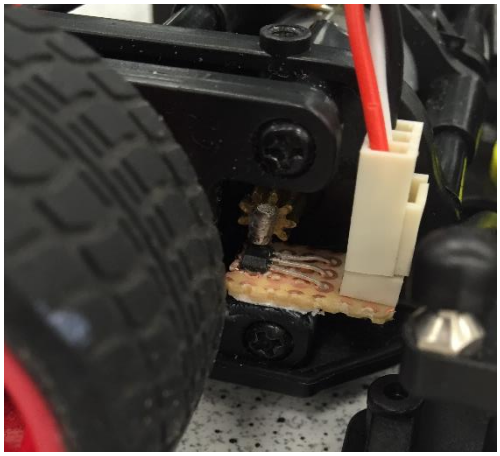


Figure 3.3 - Hall Effect Sensor on Mounting Board

To obtain more accurate speed readings, the Hall Effects sensors were refitted. The sensors were soldered to a small mounting board which was glued on the chassis. This allowed the Hall Effect sensors to be positioned closer to the magnets glued onto the gears of the motor and be positioned in the centre of the Hall Effect sensor. This provided a more accurate speed reading.

Specific grease was used to reduce frictions between gears [5]. The grease was suitable to be used on both plastic and metal, both of which were used on the vehicle. The grease was able to reduce 50% of the friction between gears which should reduce the likelihood of them wearing out after being used for long periods of testing.

4 Electronic Hardware

The main aim of the hardware design was to address the issue of the FRDM-KL25Z and TFC-Shield not being adaptable in their designs, and to also fix the issue of the noise on the current feedback from the TFC-Shield's H-Bridges. Creating a custom PCB was the most effective way to address these issues, as it gives the freedom to re-design the circuitry and to add or remove any features as necessary.

4.1 Freescale FRDM-KL25Z and TFC-Shield

The previous hardware for this project consisted of a FRDM-KL25Z board connected to a TFC-Shield. The FRDM-KL25Z was a development platform and was the fundamental control unit of the vehicle. It features an NXP KL25Z 48MHz Kinetis MCU with a high performance ARM cortex-M0+ core with 128kB of flash memory and 16KB of SRAM. It also has SPI, I2C and UART interfaces for additional peripherals to be connected. It connects to a PC via the OpenSDA debugger interface using a USB to micro-USB connector. It also has built in sensors including a capacitive touch slider and a 3-axis accelerometer [6]. This development board was used by previous groups as it was part of the standard kit and low-level drivers for the TFC-Shield were provided. It also provided enough processing capacity to process the camera data and control the boards peripherals with sufficient memory storage.

The TFC-Shield was a standard board offered to competitors in the NXP cup. The TFC-Shield mates with the FRDM board to offer additional features such as: 2 motor driver IC's, 2 servo outputs, 2 speed sensor inputs, 2 linescan camera inputs and multiple switches and trimmers for real time input. It also has four LEDs which can be programmed to display information.

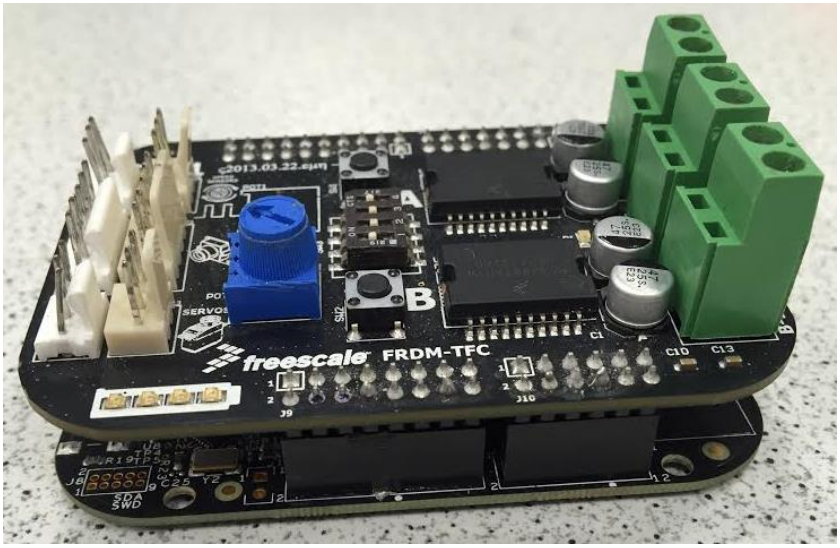


Figure 4.1.1 – FRDM-KL25Z (bottom), TFC-Shield (top) combination

4.2 Camera

The two linescan cameras provided were implemented on the vehicle. The linescan cameras consist of an array of 128 horizontally aligned photodiodes which provide a 1x128 resolution [7]. One of the cameras was positioned horizontally and used to detect the two black lines at the edges of the track. An adjustable lens was fitted to the top camera to enable the focus to be changed in order to improve detection of both lines. The lens must be regularly calibrated as the focus of the camera heavily depended on the lighting conditions. The specifications of the additional camera lens are as below: -

- 3 megapixels
- 2.8mm - 12mm focal length
- f1.4 aperture
- 1/3” image format

A second camera was implemented vertically in order to detect corners further ahead on the track. The second camera was used to control the desired target speed of the vehicle. The specifications of both of the linescan cameras are [8]:

- 1x128 pixels
- Sensor pitch of 400 DPI
- 3V to 5.5 V supply voltage
- 8Mhz maximum clock speed
- -25°C to 85°C temperature range
- $8\mu m$ spacing between CMOS linear sensors

To optimize image capture of the cameras, their exposure time was set depending on the lighting condition of the room. The vehicle has built in functions to control the exposure time.

4.3 Camera Mounting

To achieve optimal stability and allow for precise repositioning of the line scan cameras, a custom camera mount was built and implemented onto the vehicle. Stability of the camera was important as it reduces errors in the camera image feedback. Vibrations and reverse pendulum effects when going around the track needed to be minimized to obtain consistent track readings and improve lap times. Maximizing the height of the camera allows for a wider field of view, allowing both outside lines to be tracked.

Figure 4.3.1 – Camera Mount

4.3.1 Camera Base

The base of the camera mount was designed to increase stability by maximizing the surface area of the camera mount on the chassis. The best position for the camera base was found to be in the centre of the chassis behind the servo as this is the most static part of the chassis and the safest from any impact during collisions. To maximize the base area and strength, aluminium plates were used as the base of the camera mount. The aluminium plate has an area of $25cm^2$, greater than the $15cm^2$ of the previous camera mount. To further reduce the vibrations from the chassis, damping balls were sandwiched between the two identical aluminium plates. The damping balls act as a cushion or vibration damper to reduce vibrations at the top of the camera mount. To hold the pole of the camera mount, a 3D printed orthogonal shaped part was used and was fixed to the aluminium plates and chassis. The orthogonal shape of the pole holder helped in evenly distributing the weight of the cameras to prevent any swing.

4.3.2 Camera Pole

In order to further reduce the vibrations at the top of the camera mount, a carbon fibre pole was used to hold the camera in place. Carbon fibre material is well known for its shock absorbing, lightweight and durable properties which makes this material the most suitable to be used for the camera mount [9]. The carbon fibre pole consists of a $15mm^2$ cross sectional area and weighed only 51g. A lightweight pole was desired to prevent any reverse pendulum effects on the camera mount which would cause the vehicle to incorrectly read the track and cause the vehicle to begin oscillating. The length of the carbon fibre pole was 27cm to achieve the maximum allowed car height of 30.5cm from the floor to the top of the camera mount [10]. This provided the widest field of vision for the linescan camera.

4.3.3 Camera Housing

Improved camera housings were custom designed and 3D printed. A camera housing was also designed to cover the back of the line scan cameras in order to remove any light interference going through the back of the PCB of each linescan camera. This ‘light noise’ will also cause errors in the readings from the linescan cameras. The camera holders are mounted to the carbon fibre pole by using a square shaped 3D printed part that fits perfectly around the pole, designed to maximise grip. Furthermore, by having square shaped parts to connect the camera holders to the carbon fibre pole, this provides a straight camera position easily. This also prevents the camera from tilting during collisions. The camera holder was also designed in a way so that the housing can be moved up and down the pole easily and can also be fixed by tightening a screw on the arm of the camera holder.

4.4 DC Motors

The vehicle was driven by two brushed DC motors that were provided as part of the standard NXP Cup kit. These motors had to be used to comply with NXP’s rules and regulations. No alternatives with the same specification could be found, and finding new motors was difficult. Each motor was independent of the other, which allowed separate control for each wheel. Each motor connected to an individual 7:5:1 gearbox integrated into the main chassis of the vehicle located at the rear, which in turn connected to a wheel.

Specification of the Standard brushed DC motor (RN260-CN-18130) [11]:

- Drive Voltage: 7.2 V
- No Load Speed: 16000 + 3200 rpm
- No Load Current: 220 mA
- Stall Current: 3.8 A
- Stall Torque: 7.8 x 10-3 Nm
- Back EMF Constant: 0.025 Vs

One of the main issues found with these motors were that they overheat very quickly. This overheating could damage the motors over time due to the magnets becoming de-magnetized. This would result in a decreased torque output which was undesirable.

Spare motors were left over from previous years and some brand new motors were obtained from NXP officials. All of the motors had different characteristics, most likely due to the amount of usage. The motors were tested using a tachometer to measure the rotational speed of the exposed cog and from this the back EMF constant was found.

Motor	RPM	Back EMF Constant (Vs)
3	18000	1.954
5	15657	0.149
8	24300	0.244
10	16606	0.170
11	25110	0.361
New Motor 1	16700	0.173
New Motor 2	16220	0.237

Table 4.4.1 – Back EMF constant Measurements

Back EMF works against the supply voltage, therefore the lower the back EMF, the better the motor performance due to the reduced losses. However, for torque control it was just as important that the back EMF ratings for the two motors used, to be as close as possible to each other. Therefore, New Motor 1 and Motor 10 were selected to be used in races as they were close on performance with an acceptable back EMF.

4.5 Servo Motor

The steering of the vehicle was achieved using a standard NXP cup Kit servo motor.

Specification of the standard Servo motor (Futaba S3010) [12]:

- Speed: *0.20 sec/60° at 4.8V*
0.16 sec/60° at 6V
- Torque: *0.51Nm at 4.8V*
0.64Nm at 6V
- Range: *+/-90°*
- Maximum supply voltage: 9V

From the specification, it was seen that the higher the voltage, the better the servo would perform. The servo is powered from the battery; hence the higher the charge of the battery, the better the servo will operate. Therefore, it was important that during competition races there was a well charged battery at hand to get the best performance out of the vehicle.

The previous design featured the servo at the front of the vehicle laid flat, with different length struts that connected to the wheel bearings on each side of the chassis. For this year's design, the servo remained in the same location but was moved into an upright position and followed the design of group 1 from last year.

Group 1 established that the servo motor was the limiting factor in the vehicles ability to set a fast lap time. They calculated that the time taken to do a full lock was 100ms. This was compared to the camera capture time of 5-10ms and the 0.5ms to update the motor speed. They then concluded that the maximum angle to do a full lock was inversely proportional to the length of the servo arm and therefore if longer servo arms were used the result would be a reduced time taken to do a full turn. By implementing this, they were able to reduce their full lock time down from 100ms to 58ms. Therefore, this set up was implemented on this year's design as it offered the best possible turning lock time.

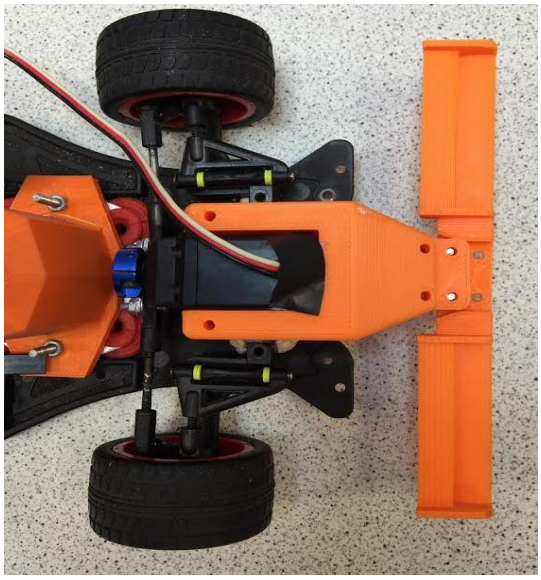


Figure 4.5.1 – Servo Motor Position

4.6 Hall Effect Sensors

Hall Effect sensors provide a crucial role in providing real-time measurements of speed of each wheel for the speed control algorithm. Hall Effect sensors were used due to their advantages of being cheap, small, reliable and accurate. There were other alternatives to Hall Effects available such as rotary encoders, however as Hall Effects were easy to implement and the previous teams used them it seemed sensible to use them again on this year's vehicle.

The previous group 2 used Hall Effect sensors on the front and rear wheels. On the rear they built removable cog caps with magnetic dots glued to them. The Hall Effect sensor was positioned above this cap which would record the rotations of the wheel. They also glued magnetic dots around the rim of the front two wheels and positioned Hall Effects to record the rotations. Problems found with this set up were that the magnetic dots on both the front wheels and the rear wheel cog caps fell off repeatedly. There were also problems with accuracy of the readings as the dots need to be perfectly spaced to give reliable readings.

The rear Hall Effect sensors were changed to match the previous year group 1 and the front Hall Effect sensors were removed. This design featured a small bar magnet glued onto the exposed motor cog of each rear wheel. When the motor ran, it spun the cog and therefore rotated the magnet. Hall Effect sensors were positioned directly under each magnet, taking great care to make sure the Hall Effects were central with respect to the magnet. The Hall Effect sensor itself was soldered onto a small breadboard that featured a header so that a wire connecter could be connected and removed with ease. The reason for this was that if the control mode changed i.e. using torque control instead of speed control, there would no longer be a need for Hall Effect sensors, therefore they could be removed conveniently without potentially damaging the connections or Hall Effect sensor.

Unipolar Hall Effect sensors were used. These sensors gave a logic high when the south pole of the bar magnet was detected. Therefore, every rotation had a clear rising and falling edge which the MCU could then use to calculate the rotational speed of each motors by counting the number of these edges. This design was a more robust method of measuring the speed of each motor.

4.7 Battery

As stated in the rules the vehicle could only have one power source and it must be a 7.2V, <=3000mAh NiCd, NiMH or Li-Ion battery. The choice of battery was limited, therefore the team considered different ways of maximising performance with the batteries available.

Battery	Cell type	Nominal Voltage (V)	Current Rating (mAh)	Weight
iMah	NiMH	7.2	3000	347g
Ansmann	NiMH	7.2	2000	269g

Table 4.7.1 – Available Batteries

During practising the iMah battery was used, as this offered a higher battery capacity and therefore a longer battery life which came in useful for long periods of testing. For the qualifier race the Ansmann battery was used, this was because the battery was lighter and still offered a reasonably high battery capacity. Unfortunately, a 7.2V, 1000mAh battery was not found that complied with the NXP Cup rules specification. A 1000mAh battery would have been most preferable as this would have offered a sufficient battery capacity for the short race and would have been lighter than the Ansmann, therefore potentially it would have increased the performance of the vehicle.

4.8 Lights

As the lighting conditions of the room for the competition races are unknown, an array of LED lights were fitted to the vehicle. This was to provide improved camera readings during low-light conditions where, even with increased exposure time, the track would be difficult to read. Having longer exposure time also reduces the performance of the vehicle. The LED lights were used to obtain consistent lighting by having them angled at the black lines. To obtain the highest light intensity, the LED were connected to 5V supply from the PCB. The same LEDs used by the previous group 2 were used. A 3D printed LED housing was built in order to mount the LEDs to the camera pole. The 3D printed LED case was also designed to be detachable, therefore can be removed to reduce weight when lighting conditions are acceptable.

The 3D printed housing for the LEDs was designed with two U-joints to make it double-axis angle adjustable, meaning that the direction of both LEDs can be adjusted vertically and horizontally. This was done to ensure that each edge of the track will be illuminated by the LEDs. The problems encountered with the LEDs was that if a large amount of LEDs were used they pull a high level of current which drastically reduces battery life. This can be fixed by connecting the LEDs to a suitable resistor to reduce the current intake, however comes at the cost of reducing the brightness of the LEDs.

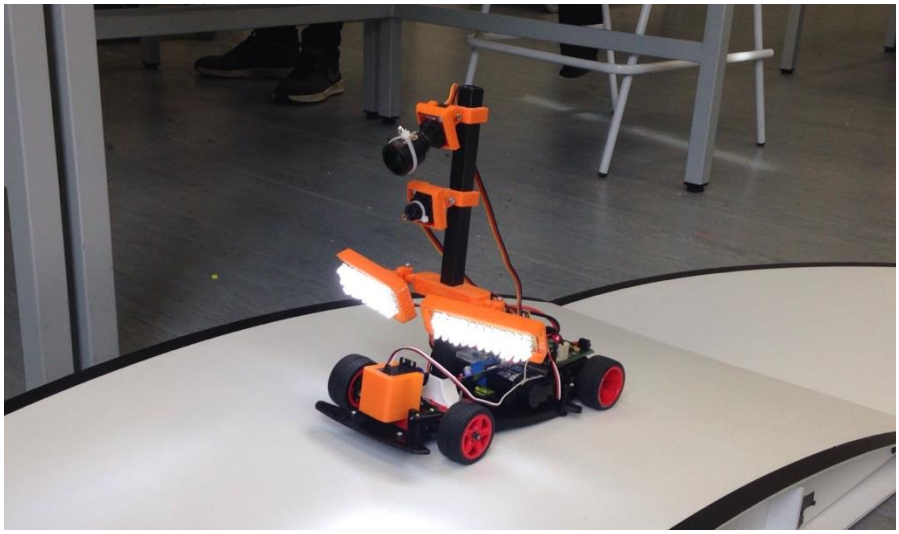


Figure 4.8.1 – Implementation of LEDs

4.9 Custom PCB

4.9.1 Software

The PCB was designed in Proteus. This was chosen over alternative PCB design software, such as EAGLE, as the University owns the license for the software. Proteus provides a large component library and various design tools in order to create a design schematic as well as complimentary software called ARES that allows the actual layout of the PCB to be designed.

4.9.2 Design Process

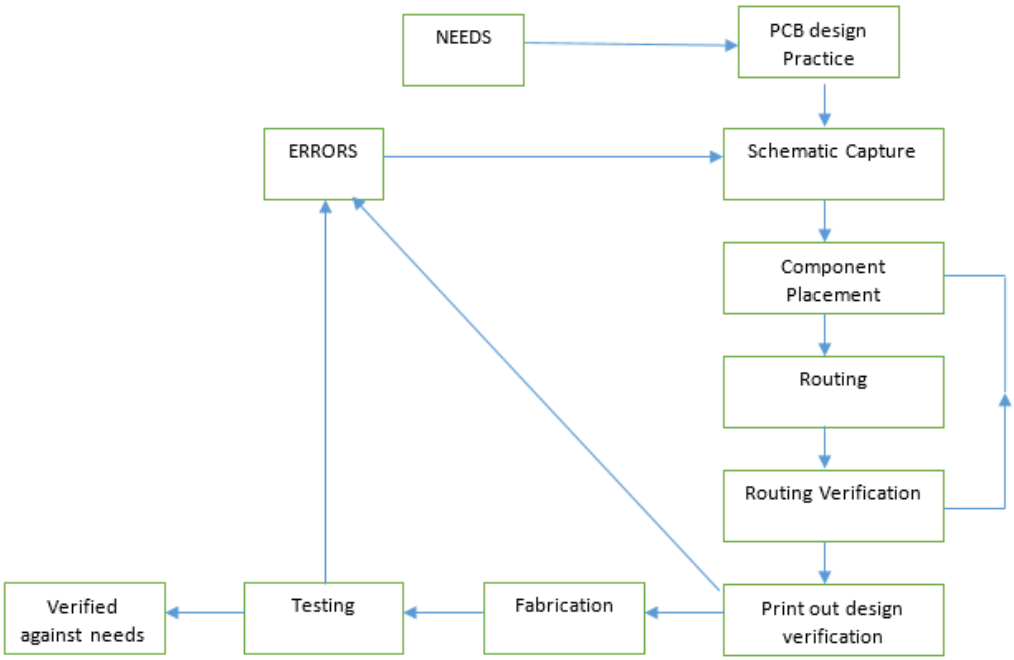


Figure 4.9.2.1 – Design Methodology Followed

The design process for the PCB was outlined in figure 4.9.2.1. The first main stage of the design process was to assess the needs of the project and then to consider an initial concept of the design. The 'needs' were what was required from the new design. Initial concept was looking at how would the needs fit into the new design. The next stage was the main part of the design process. This involved the creation of a schematic of the new design, followed by a prototype PCB layout and routing. The last stage was to get the PCB built, assembled and then tested.

4.9.3 Hardware Adaptions

Creating a custom PCB allowed the freedom to pick and choose features from the FRDM-KL25Z and TFC-Shield which were necessary to the operation of the vehicle and should be kept and what features should be removed. The vast majority of the features from both boards were kept for the new PCB as it was needed to be able to perform the same operations.

One of the main features of the FRDM-KL25Z was its OpenSDA interface which was used to connect a PC to the board for debugging purposes. For the new PCB this feature was removed and replaced with a JTAG connector. This was due to the OpenSDA taking up a lot of space and requiring a lot of components. One downside to this adaption was that the JTAG required an adapter to be able to be plugged into the PC, which was relatively expensive.

Another addition to the hardware which was featured on the new board was a low-pass filter on the current feedback from the H-bridges to the MCU. The reason for this was that a low-pass filter will attenuate high frequency noise interference present on the current feedback. This therefore should allow for accurate current readings for a torque control. This could have been implemented on the previous board combination, however it would have required cutting tracks and physically soldering on a capacitor and resistor in place, this process creates a high chance of permanently damaging the board.

A permanent header for Bluetooth and LCD connections was also a new feature on the custom PCB. Previously groups had to solder on wires to certain pins which caused issues if solder joints became lose and therefore loss of connection occurred. Therefore, adding somewhere on the board where they can be connected was more convenient, tidy and more reliable. The convenience of being able to attach a LCD or Bluetooth module was useful for testing and tuning.

The capacitive touch slider which featured on the FRDM-KL25Z was removed from the new PCB design. The previous team didn't use this feature as it didn't really have any practical use in this project. The slider also took up the largest amount of area on the FRDM-KL25Z, therefore removing this created a lot of room for other components on the new PCB.

One consideration that was made was to add additional switches and LEDs as it was recommended in last years' report, however the team saw no additional benefit in doing this as an LCD display was being implemented. The display would be able to display a lot more information than any additional LEDs would. It was also decided that the current number of dip switches offered a sufficient number of mode options.

Another part that was removed from the original design were the R3 Arduino connectors that connected both of the previous boards together. As the custom PCB featured both the FRDM-KL25Z and TFC-Shield components, all the physical wiring was integrated on to a single board, thus removing the need for the connectors.

A power switch was also implemented on the battery cord for ease of use.

4.9.4 Schematic Capture

After the adaptations and initial concept were considered, a schematic capture was created. The schematic was an accurate representation of the circuit that was going to be made. The design of the schematic for this project followed the design of the FRDM-KL25Z and TFC-Shield schematics closely as the new board was essentially a combination of the two. The schematic for the FRDM-KL25Z was located on the NXP website that provided a datasheet containing the circuitry design with component values and advised ratings [13]. The TFC-Shield schematic was found on the NXP Cup community page, which provided circuitry designs and a Bill of Materials (BOM) which had precise component values and ratings on it [14]. The final PCB schematic can be found in Appendix A.

The vast majority of the components needed to implement the schematic were available in Proteus's libraries, however some were not and required creating. In order to create a component, the desired component was first found online, making sure it was available for purchase. The datasheet accompanying that component was then used as reference to design it in ARES. This process was done for every component that wasn't available. Once the components were created and placed, they were simply connected up and then verified using the *'highlight net schematic'* feature that showed exactly what was connected to what. If any errors were found at any stage of the design process, they were corrected by changing the schematic first before proceeding.

4.9.5 Layout

Once a schematic for the board was created and verified, it was then laid out and routed using Proteus 7's ARES tool. For this project a number of layouts were created using different amounts of layers, each of which had advantages and disadvantages.

4.9.5.1 First Attempt

Initially when creating a PCB layout in ARES a 4-layer design was proposed. This initial layout can be found in Appendix B. This first design was very basic and not very professional. ARES's auto-place tool was used, this tool placed components sporadically without any real logic behind it. Once the components were placed, an auto-router tool was used to connect the components. The result was a very messy and unsatisfactory design, it also showed that packing in components as tight as possible on a board made it incredibly hard to route. Therefore, from this initial design it was clear to see that auto-placer and auto-router should be avoided when possible.

4.9.5.2 Four Layer Design

The most recent 4-layer design can be found in Appendix C. This designed featured first attempts at laying out the board and routing by hand. The 4-layer board had the advantage of being smaller and more compact. However, in the end the 4-layer designs were scrapped for a number of reasons. First, a 4-layer board costs triple the amount to fabricate than a 2-layer board. Considering that the majority of initial prototypes do not work correctly first time, any mistakes would become very costly. Considering the lack of experience in PCB designing, it was an inevitable that a mistake was going to be made, therefore 2-layers would be a lot cheaper in the long run.

Another reason for not using 4-layers was the potential difficulties in troubleshooting. A 4-layer board was sectioned into signals and components on the top and bottom layers and ground and power planes in the middle. If anything happened to the internal layers or tracks within those layers, nothing could be done to fix them. On a 2-layer board, all the components and tracks are on the surface, therefore everything is easy to access and can be easily repaired.

The final reason for not using 4 layers was simply because it wasn't necessary. A ground plane could be simply implemented on the top and bottom of the board and therefore removing the need for an internal ground plane. There were not many power tracks required in this design, therefore there was not a need for a dedicated power layer.

4.9.5.3 Two Layer Design

From the experience gained from earlier versions and continuous research into PCB design [15], a final 2-layer PCB was created as shown in figure 4.9.5.3.1. This was the third redesign of the board.

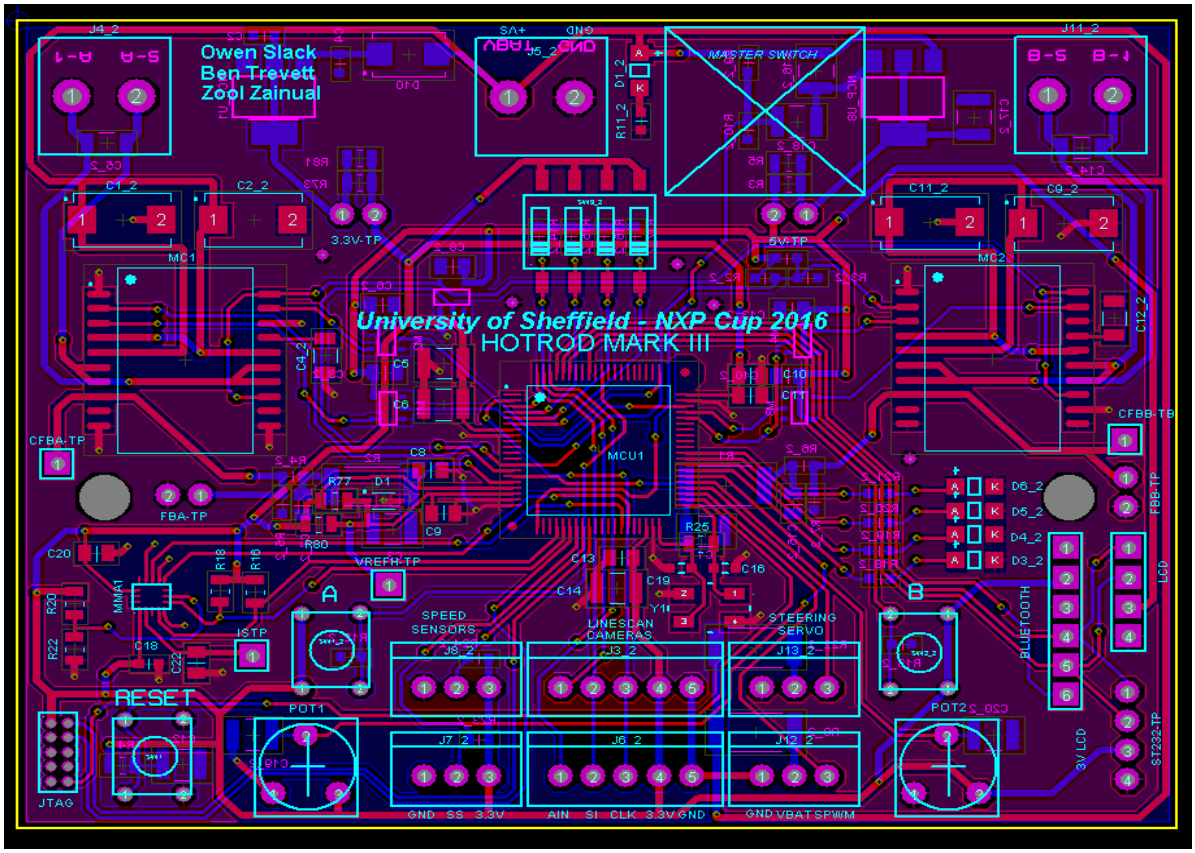


Figure 4.9.5.3.1 – Final PCB Design, Hotrod Mark 3, 2-Layer Layout.

4.9.5.4 Component Placement

In this PCB design, the majority of the components were placed on the surface of the board. This was intuitive in terms of components that will have to connect to external modules, for example, it would be poor design to position the Bluetooth connecter on the top of the board and the LCD connector on the bottom as it would make using both at the same time difficult. Another reason for putting the majority of components on the top side was for ease of fabrication, as only one side can be baked on. Therefore, all key components that could not be soldered on by hand had to feature on the same side of the board. The main components were the MCU, H-bridges, crystal oscillator and the accelerometer.

The MCU was placed on the layout first and was the focal point for this design. This was placed in the middle of the board as it connected to the majority of the components. As a result, the required trace length between the MCU and the majority of the components were minimized and the overall trace impedance was reduced.

Another strategically placed component was the crystal oscillator located just below and to the right of the MCU. There are lots of considerations when it came to placing the crystal oscillator [16]. For this design however, it was not crucial to place the crystal as close to the MCU as possible as it was only an 8MHz crystal oscillator. Typically for high speed designs, i.e. >300MHz, it is important to

place the crystal as close as possible otherwise the board would potentially suffer from clock skewing or noise reflections on the clock traces [17]. Therefore, for this design the crystal was put close to the MCU for good practice reasons, it was also isolated from other signalling traces to avoid any crosstalk interference.

Another important component positioning was the decoupling capacitors. Decoupling capacitors featured heavily in this design. For example, the MCU had five 3.3V inputs, each of these inputs featured decoupling capacitors. They were placed on the inputs as close as possible to the MCU. The purpose of those capacitors was to smooth out any voltage fluctuations that could damage the MCU. Decoupling capacitors were also used for the same purpose to protect the h-bridges, accelerometer, IC buffers, voltage regulators, crystal etc. Essentially anything connected up to a voltage track on the board had decoupling capacitors close to it to protect it.

4.9.5.5 Tracks and Signals

Another design practice that was implemented on this design was to keep high current, noisy traces away from signal traces. This was mainly to do with the current feedback trace from the h-bridge to the MCU. As this trace was connected to the MCU it was pretty much impossible to have the current feedback trace not come into contact with a signalling trace or another pad on the MCU. Therefore, it was important to keep capacitive and inductive crosstalk interference at a minimum [18]. As stated in the hardware adaptions one of the new additions to the design was a low pass filter on this feedback track. This was the only way of tackling this issue as adding a low pass filter removed the high frequency noise component of the feedback trace and with clever routing the trace only came into contact with other signalling traces for a short duration. The overall result was a reduced likelihood of the feedback trace causing interference. The power tracks were also isolated from nearby signal tracks as much as possible and routed round the edges of the board when possible to avoid the majority of the traces. Power tracks can cause noise interference, which could degrade or weaken signalling signals.

Track length was also a consideration. It was made sure that there were no unnecessarily long tracks, as this would result in a higher parasitic capacitance and inductance on the trace and therefore the track would be more subject to noise [19].

Another consideration was trace angles when cornering. If signals take a corner with an angle of <= 90 degrees the signal has a tendency to continue forward, therefore if there is any component near to this corner or an adjacent track there would likely be cross talk interference [19]. To combat this effect ground, vias were positioned on sharp corners so any escaping signals were grounded. Another technique that was used was to angle corners >90 degrees so that the signals can navigate the corner effectively.

Track sizing was also considered in this design. The trace sizes depend on a number of factors, these include ambient temperature, current level, copper thickness etc. [20]. From the datasheet of the FDRM-KL25Z it was found that the maximum digital supply current of a signalling trace was 120mA. Therefore, using a trace sizing chart [20] it was found that a width of 0.67thou should be used at 20 degrees C. However, the smallest pad width on the MCU was 12thou, therefore 12 thou was used for the signalling traces as it offered a higher current carrying capacity. For the power traces and feedback track, it was found that no more than 1.6A of current will flow at any given time, therefore again using the trace sizing chart, it was found that a width of 25thou should be used.

Another design practice that was employed to some extent was the use of different layered tracks for different directions. In the final design, the top side traces (red) were used for routing left to right and the bottom surface traces (blue) were used for going up and down. This design practice was not crucial to the performance of the design, it just makes it look neater and more professional. In this design it was used when possible. However, because the board was compact, it was often difficult to fully implement this.

4.9.5.6 Ground Planes

A main consideration in combating the effects of Electromagnetic Interference (EMI) was to design the PCB with sufficiently good ground plane coverage. Ground planes are a layer of copper that acts as a direct path to ground for components on the board. Ground planes are used to ensure that ground signals are grounded to a constant 0V throughout the PCB. If tracks are used to route ground signals to ground then there will be naturally some resistance in the tracks, this will create a voltage drop, hence making some ground nodes not a true ground. This would be very undesirable within the system and could result in common impedance coupling. Therefore, the copper plane coverage was maximized as much as possible to cover the majority of the components. The board featured very good coverage on both sides.

Another reason why ground planes were implemented was that a sizable area of copper covering the PCB's surface would allow for better heat dissipation.

Grounding vias were also added in areas where copper coverage was poor. They acted as a link from the top copper plane to the bottom plane. They were also used in areas where there was good ground copper coverage but a long distance to the ground terminal, therefore creating a more direct path to the ground terminal. The overall result of the grounding vias was a robust ground plane design.

4.9.6 Layout Verification

The layout was verified by first ordering desired components from online retailers such as Farnell or RS Electronics. These components were then placed on a scale print out of the PCB layout *(figure 4).* The BOM for this design can be found in *Appendix D.* This stage was very useful as it showed whether physical parts would fit next to each other or whether component pad sizes were too small or too large etc. Once the physical layout had been check for errors the team sat down and went through the layout making sure everything was right before getting the PCB made.

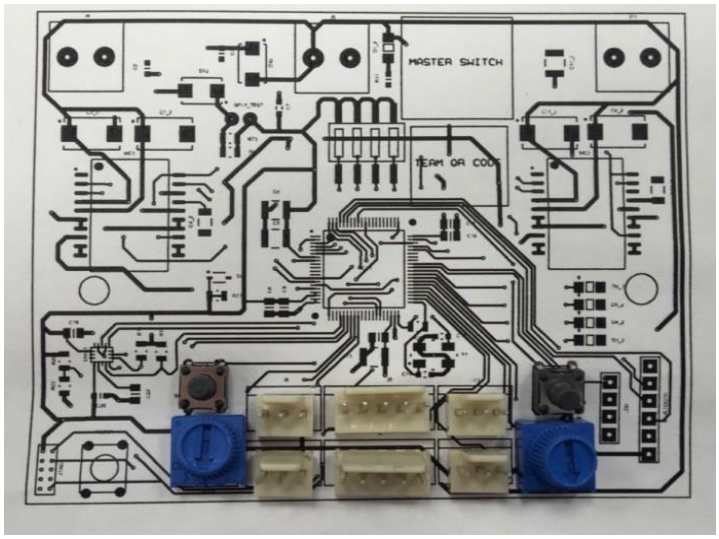


Figure 4.9.6.1 – PCB layout verification.

4.9.7 PCB Manufacture

Once the layout had been verified, the PCB was sent off to be made. The company used for this process was PCB-Pool. This company was chosen because they offered a decent turnaround of 8 days after they receive the order. They were also convenient as they were the only company to accept Proteus layout files. However, they were relatively expensive compared to alternative manufactures such as PCB-Train.

The PCB base material was FR4, 35 µm Cu, 1.6 mm. It also came with a solder mask, which considering the majority of the soldering was done by hand was very useful to have, otherwise the board would have got damaged. It also came with silkscreen on top and bottom, again useful for fabrication because the component names were shown on the board. It also featured ENIG (Electro-less Nickel Gold) for ultra-flat pads, a minimum track/gap size of >= 0.15 mm and minimum drill-end diameter of >= 0.3mm. The board also came E-tested meaning the tracks had been continuity tested. The PCB also came with a free stencil which was very useful when it came to fabrication.

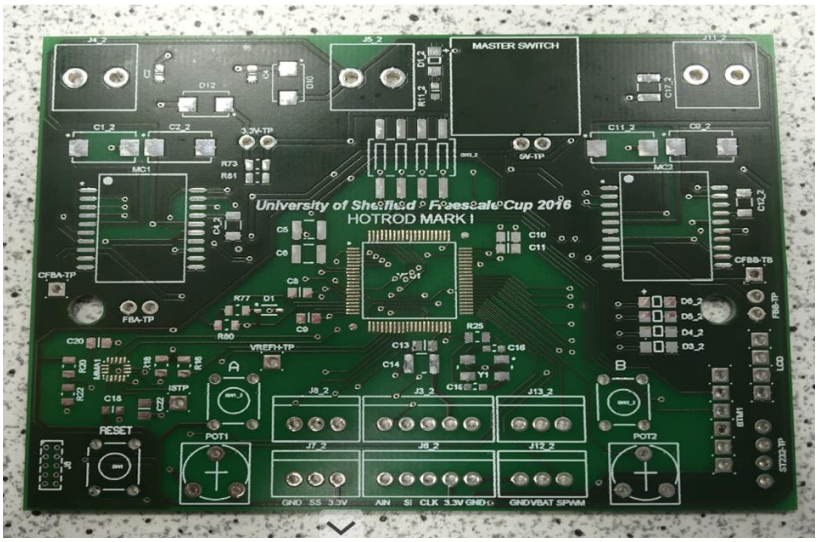


Figure 4.9.7.1 – First PCB (Hotrod Mark I).

4.9.8 Fabrication

The fabrication for this design was done by hand. This allowed the board to be built up in sections and tested so that errors in the design could be easily located.

Initially, an oven was used to solder on the components that could not be soldered on by hand. Baking on components was the most difficult part of the fabrication process. If the components did not bake on correctly first time, the board had to be discarded, as trying to remove components often resulted in the tracks and pads also being pulled off. Only one side of the board could be baked on for this design due to limitations of the oven. This was because if one side was baked on and then flipped and the process was repeated, while in the oven, the solder on the bottom would reflow and the components would fall off. Another reason the board was only baked once was that re-baking components can warp their performance or reduce their expected lifetime.

The surface mounted devices (SMD) that had to be baked on included the MCU, accelerometer, H-bridges and crystal oscillator. The MCU was a LQFP-80 type package. Therefore, in practice it can be soldered on by hand however it would be very difficult because the legs of the chip are very close to each other and the likelihood of shorting pins would have been dramatically increase. With regards to the crystal and accelerometer they are type QFN, meaning they had to be baked on because their pads featured under the component. The H-bridge was baked on as well. The rest of the SMD were mounted by hand using a hot air solder. Soldering by hand was a lengthy procedure however it also was the easiest way of finding mistakes in the design because components were individually applied.

Once the main components had been baked on, continuity testing was carried out. This was to check whether there was any shorting between adjacent tracks or pins. If this was okay, then the rest of the components that made up that particular section were then soldered on. After this they were continuity tested as well and then if that was okay the whole section was tested to see if it worked.

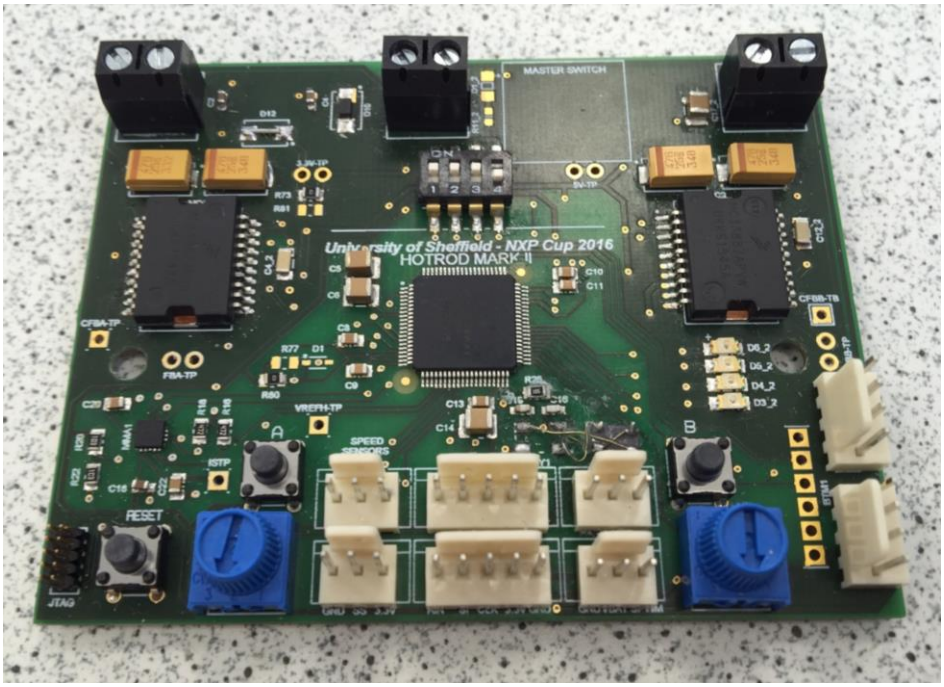


Figure 4.9.8.1 – Final Fabricated PCB

One error found during the verification stage was that pad layout of the crystal oscillator was wrong. This error was not spotted during the layout stage. This led to the pads being incorrect for the PCB. To solve this issue, the crystal oscillator was glued onto the board and fine wires were used to connect the pads to the correct pads on the oscillator. This solved the issue and allowed the PCB to function correctly and was rectified for future designs.

4.9.9 Testing

The PCB testing adopted a bottom up hierarchical approach, meaning that individual subsystems were first fabricated and then tested to check if they worked. For example, once the fabrication of the linescan camera components was finished, it was then tested and validated that it worked correctly. The next section of the design that was implemented was the motor circuitry. Again, this was individually tested to see if the motors actually worked. The motors and linescan camera were then tested together to see if they performed co-operative operations, i.e. if the camera read 'line lost' then the expected result would be for the motors to stop, which they did, therefore validating this section of the design. This hierarchical approach was repeated until the whole board was tested. The final verification that the board worked was the vehicle going round a small oval track, without failing in, both directions.

One of the main aims of this design was to implement a way of reducing the noise on the current feedback from the H-bridges. The previous board configuration current feedback was tested and can be seen in figure 4.9.9.1. It shows the current feedback from both motor h-bridges to the MCU were subject to a lot of noise interference, hence it was impossible to get accurate current feedback measurements to implement an effective torque control.

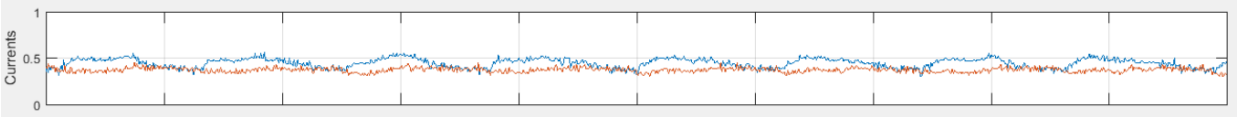


Figure 4.9.9.1 – Current feedback before low pass filter, left motor (blue), right motor (red).

As stated in the hardware adaptions section a low-pass filter was implemented on each of the current feedback lines to attenuate the high frequency noise component. The resulting effect of the low-pass filtering can be seen in figure 4.9.9.2.

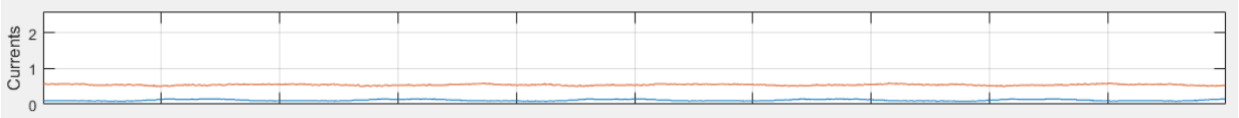


Figure 4.9.9.2 – Current feedback after low pass filter, left motor (blue), right motor (red).

Figure 4.9.9.2 shows that that the current feedback being received by the MCU was a much more smoothed out response. Therefore, the reduced noise on the current feedback means that accurate current measurements could be taken and torque control could be implemented accurately.

4.9.10 Verification

The overall implementation of the custom PCB was a success. The board was able to operate as the previous FDRM-KL25Z and TFC-Shield configuration had done. The overall area reduction was from 84.8cm^2 to 63cm^2 with a total area saved of 21.8cm^2. The compact design allows the freedom to change the positioning the PCB as it was small enough to be positioned either in the middle or on the back of the vehicle. The weight of the board was also reduced from 94g to 41g which saved 53g's therefore, over 50% reduction in weight.

The custom PCB was also able to solve the issue of a noisy current feedback as shown in the testing section. Accurate current feedback readings were able to be made and therefore, a more effective torque control could be implemented. The H-bridges also did not over heat excessively which was due to the PCB design having almost complete ground copper plane coverage on both the top and bottom of the board, resulting in sufficient heat dissipation. It also helped having grounding vias in locations where there was no direct path to the ground terminal. The result was a design that did not require heat sinks. The new PCB was also a more flexible design and the additions of a permanent Bluetooth and LCD connector make adding additional modules easier.

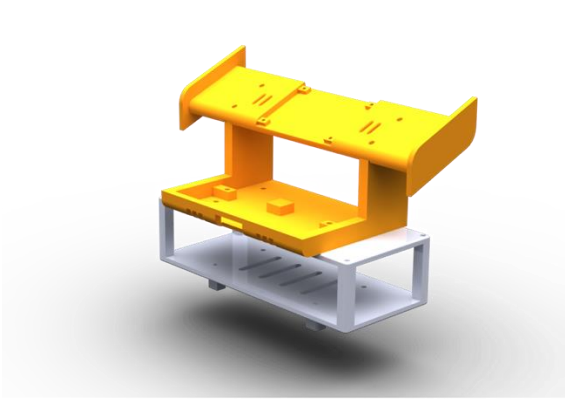
The custom PCB was also comparable on costs to FRDM-KL25Z and TFC-Shield, with the previous board combination costing around £70 from the NXP community page. The custom PCB with components cost roughly the same when fabricated by hand and if the board was purchased through a cheaper manufacturer it would be even cheaper. However, a professionally fabricated board would cost significantly more; although, it would be more reliable.

4.9.11 PCB Mounting

To securely hold the custom PCB to the vehicle, a PCB holder was designed using SolidWorks and 3D printed in the lab. The PCB holder replaces the heavy Perspex tray which had been used by last year's team. The Perspex tray from last year did not provide any protection to the PCB and the components on it. Protection was required as the custom PCB was expensive and takes a long time to fabricate if damaged.

The 3D printed PCB holder used an ABS plastic material that was robust and lightweight, which was suitable for the vehicle. To hold the custom PCB in place without any movement caused by vibrations from the chassis, the custom PCB was secured with plastic M3 screws and nuts to prevent any unwanted tilt and movement that could interfere with the accelerometer readings. The 3D PCB holder was designed to protect the sides and bottom of the custom PCB since those were the parts of the PCB which have the highest probability to damage during collisions with surroundings. Ventilation holes were also implemented in order to provide better air flow for the bottom side of the custom PCB since the H-bridges tended to get hot after a few test runs with the aim of increasing heat dissipation.

In order to maximize the usage of the PCB holder, it was designed to have a detachable spoiler that holds the LCD display, which also allowed the user to easily view the display.



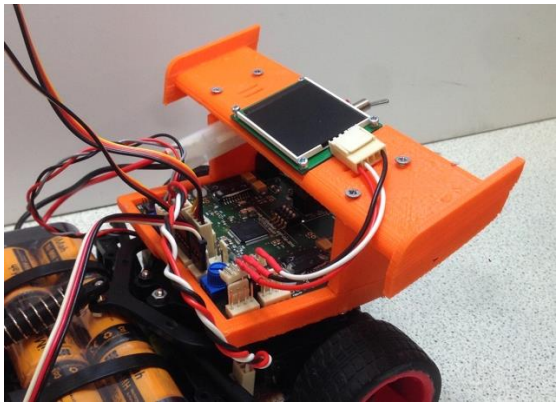


Figure 4.9.11.1 – Spoiler for LCD Display

5 Software and Control

Processing of the software and hardware was done using NXP's CodeWarrior 10.6 IDE. This was chosen due to it being NXP's IDE and thus offered the best support for the architecture. NXP's Kinetis Design Studio (KDS) was also considered, however due to time required to import the code from CodeWarrior to KDS, it was decided to use CodeWarrior.

The vehicle uses a large number of functions in order to provide improved functionality of the vehicle. This leads to the overall size of the code being large. It was decided to focus on readability and efficiency in order to generate a code structure that was easy to understand whilst being effective. As such, a separate source file was created for each main task of the vehicle, i.e. line detection, steering control, speed control, etc.

A vast amount of *#define* and *#ifdef* functionality was used allowing for a large amount of flexibility by allowing multiple different algorithms to be implemented for the same problem with an easy method to switch between them. The majority of the #define statements are contained in *settings.h* and can be enabled or disabled by uncommenting them or commenting them out.

The project is structured as follows:

- /sources/ – high level files
- /sources/TFC – low level drivers
- /project_headers – headers for the high level files
- /project_headers/TFC – headers for the low level drivers
- /project_headers/typedefs_structs – structure definitions

5.1 Sensor Data Collection

Real time information was required from various sensors on the vehicle. The majority of the low level drivers were written by NXP with some alterations made by last year's team, and contained in the *TFC* folder. Some further adjustments and additions were made to the low level drivers in order to perform required sensing and processing.

5.1.1 Camera Image Capture

The image capture functionality is contained in *TFC_LineScanCamera.c* and *TFC_ACD.c*. The ADC sets a pin high to enable the value of each pixel to be sampled. A ping-pong buffer is used to ensure that no sampled values have been overwritten before they have been read. LineScanImage points to a 1x128 array of pixels, the captured image. Once the image has been captured, the *lineScanState* is set to *'linescan image ready'* so it can be processed.

The number of FPS and the exposure time depends on how often the image was sampled and was controlled by AutoExposure.c. The auto-exposure takes the sum of the intensity values from all 128 pixels and performs proportional control on the exposure time in order to set the exposure of the image identical in different lighting conditions. The desired exposure is calculated from half of the sum of all pixels at maximum intensity.

The exposure time has the value of proportional gain and the limits of the exposure time in *settings.h*. If the exposure time goes below the limit, the vehicle is not able to fully process the image before a new one is captured, therefore causing the vehicle to fail to follow the track correctly. If the exposure time goes above the limit, the vehicle does not capture enough images and therefore does not maintain an accurate image of the current track ahead, also causing the vehicle to fail to follow the track correctly.

5.1.2 Speed Detection

Speed detection is carried out using the Hall Effect sensors by last year's team. Whenever one of the poles on the magnet on the end of the cog passes the Hall Effect sensor, a pulse is triggered. The pulse interrupts the MCU and a timer counts the time between interrupts to calculate the speed of the vehicle. This was calculated using the size of the wheels and the ratio of the gears.

The actual speed is measured in 'car speed units' (CSU) and was calculated as:

$$Speed\ (CSU) = \frac{20}{\pi} * Speed\left(\frac{m}{s}\right)$$

This unit was created by the team before last year's team in order to simplify some calculations and it was decided to continue to use it. The maximum speed the vehicle is able to achieve on long straights is approximately 30 CSU or 4.71 m/s.

Speed is calculated for each wheel individually, with the option of taking a speed reading from one wheel and mapping it to both, or to map the average of the two speeds to both wheels.

It was found through telemetry that the speed readings were very noisy, and gave the appearance of the wheels repeatedly stopping and starting, even when they were moving at a constant speed and that when a wheel had stopped spinning the speed reading stayed at the last non-zero value.

Noise on the speed readings would give an incorrect speed value and thus lead to non-optimal speed control. To solve the noise on the speed readings, a weighted average was used. This would set the new speed reading to 90% of the previous reading and 10% of the measured reading. Before and after is shown below in figure X, with red being the speed of the right wheel, yellow the speed of the left wheel and blue the target speed.

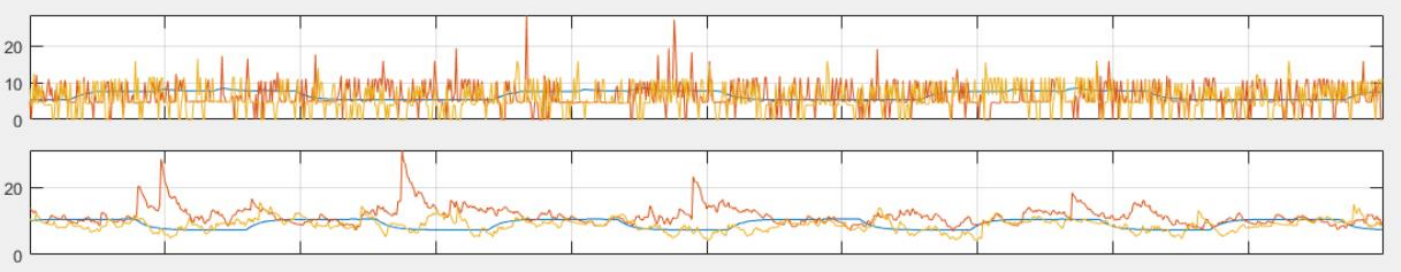


Figure 5.1.2.1 – Speed of Motors Before (top) and After (bottom) Averaging

Even though the speed reading was vastly improved, once the vehicle went above certain speeds, noise spikes began to appear again. A moving average filter was then implemented in order to further average the speed. Using the formula:

$$speed_{MA} = \frac{1}{M}\sum_{i=0}^{M-1} speed_i$$

It was found that even though this slightly smoothed the value further, large spikes still appeared. It was decided that these spikes were due to hardware issues, such as noise or interference from other components. The weighted average filter was used due to providing good speed readings without the additional processing power consumed by the moving average filter.

The issues with the speed readings not falling to zero once the wheel had stopped was fixed by altering the *isNewMeasurementAvailable* and *GetSpeed* functions. Originally, last year's team had implemented a timeout that would interrupt when the counter had overflowed and no Hall Effect interrupts had triggered, implying the wheels had stopped spinning. When the timeout interrupt triggered, the measurement availability flag would be set high with a speed reading set to 0. This was altered so that the measurement availability flag would be set low on a timeout. The *GetSpeed* function was altered so that it took the measurement availability flag as an argument, and if the flag was low it would return 0 speed. The speed detection is implemented in *SpeedSensor.c*.

5.1.3 Battery Voltage Measurement

The battery voltage reading was taken by reading the battery voltage using the MCU's ADC and contained in *TFC_ACD.c*. Last year's team used a multimeter to calibrate readings to ensure accuracy and this was repeated. The battery voltage was measured every 100ms and when the battery voltage was below a certain threshold, specified in *settings.h*, the LED's indicate this.

Ideally the battery voltage should be as high as possible, as this causes the motors to accelerate and decelerate faster as well as the servo turning faster, giving vastly improved vehicle performance compared to when the vehicle is operating at low battery levels.

5.1.4 Current Measurement

The H-bridge motor drivers allow for current feedback to be measured using a feedback pin. Current measurement was useful as it can be used for torque control, where each motor's torque is proportional to the motor's current. The TFC-Shield had a large amount of noise on the current feedback pin, giving readings that were too noisy to be used for a reliable torque control. The custom PCB has two low-pass filters in order to smooth out any noise, allowing for torque control to be used.

In order to smooth out current readings further, the *TFC_ReadMotorCurrent* function was modified to return an average current value by using 90% of the previous current value and 10% of the actual value. This reduces any large changes in current, reducing the effect of noise and increasing the stability of the torque control.

5.1.5 Accelerometer

An MMA8451Q accelerometer was utilised in order to allow the vehicle to manoeuvre bump and hill sections of the track. On initialisation when restarting the MCU, the accelerometer takes a number of readings in the X, Y and Z axis and averages them to find the 'zeroed' values. This allows the accelerometer to take readings relative to a zero point, allowing for more accurate determination of the vehicle's acceleration.

It was observed that the vehicle slowed down when climbing a hill section and sped up too quickly on the decline of the hill, possibly failing to make a turn directly after the hill. The accelerometer was used to increase the desired motor speed when travelling uphill, by checking when the accelerometer's X axis reading went above a certain threshold and decrease it when travelling downhill, by checking when the X axis reading went below a certain threshold.

It was found then when travelling at high speeds, the bump section of the track caused the vehicle's movements to become unpredictable, with the risk of leaving the track entirely. The accelerometer was used to reduce the desired speed of the vehicle when passing a bump section in order to reduce the risk of leaving the track. Due to the vehicle's unpredictable 'bouncing' when crossing bump sections of a track, a timer was initially used to differentiate between hill sections and bump sections, as the vehicle mistakenly thinking it was climbing an incline when on a bump section would further increase the odds of the vehicle leaving the track. If the accelerometer returned consecutive upwards acceleration (above a threshold) for a set amount of time, the vehicle was determined to be on a hill section. If the vehicle returned positive and negative acceleration within a short time of each other, the vehicle was determined to be on a bump section.

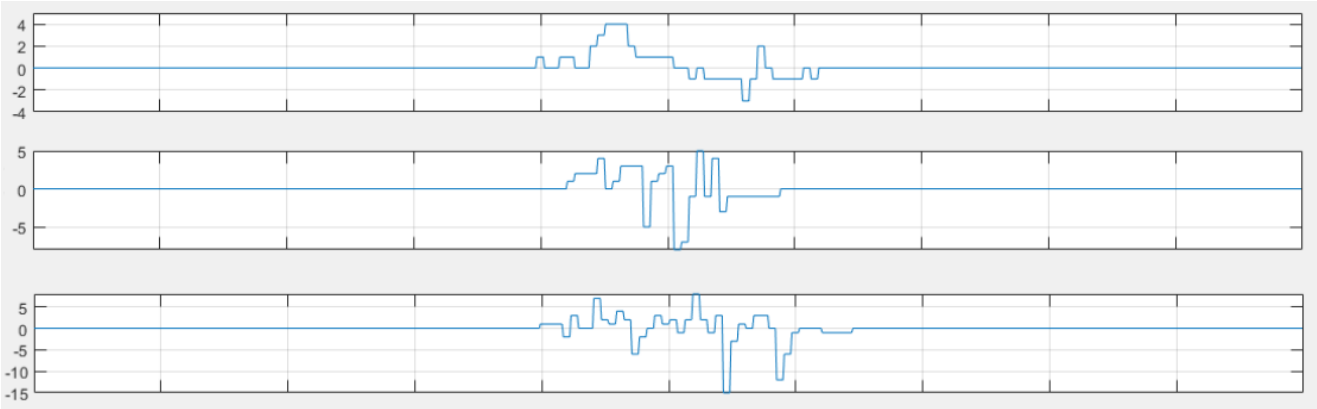


Figure 5.1.5.1 – Accelerometer Readings of hill section (top), medium speed bump section (middle) and high speed bump section (bottom)

Even with many alterations to the detection algorithm, it proved difficult to differentiate between hill and bump sections using this method, especially as the number of 'bumps' on the bump section of a track and the gap between the bumps varied, i.e. during testing a section with 8 bumps was used, whereas during the competition the bump section only has 5 bumps. The possibility of leaving the track due to incorrectly speeding up was deemed more important than loss of speed when climbing a hill section, therefore the final implementation of the accelerometer was used to detect when the value from the accelerometer had gone negative, indicating a bump section or going downhill, and then reducing the vehicles maximum speed for a set amount of time.

5.2 Image Processing

The linescan camera used only sees light intensity values in a 1x128 line. These images have to be processed by the vehicle in order to determine the layout of the track ahead of it. Previous methods used were having a threshold of light intensity values. The sections of the track with the lowest light intensity should be the black lines, and the centre point between the two black lines should give the position of the centre of the track. This is not used due to light intensity being heavily affected by poor lighting conditions, shadows on the track, unpredictable surroundings around the track, etc.

Another method used by some teams was calculating the gradient of the image intensity between each pixel, as the left hand black line should have the highest positive gradient (going from black to white) and the right hand black line should have the highest negative gradient (going from white to

black). This was less susceptible to an overall lower level of light intensity, i.e. a poorly lit room, however shadows on the track will still cause an erroneous reading as the vehicle may assume that a high derivative between a shadow and a white section of track is one of the black lines.

Last year's team implemented Canny edge detection, as it was widely used and regarded as a highly optimised method of detecting edges [21]. Although Canny edge detection uses complex calculations, and hence more processing power, as the image was only a 1x128 array this was determined not to be a problem. However, to minimise processing power used, the Gaussian kernel's length was minimised and is generated on initialisation and pre-calculated values were used for mathematical calculations.

The edge detection, implemented in *EdgeDetection.c* and used in *LineDetectionDouble.c*, takes in the light intensity values from the camera and performs the following 3 steps to return an array with detected edges:

1. Perform Gaussian Filter
2. Find Intensity of Gradients
3. Non-maximum Suppression
4. Threshold Comparison

The Gaussian filter was used to smooth the image in order to reduce the effects of noise. This was performed by convolving the captured image with a Gaussian kernel, a matrix. The kernel was created using the formula:

$$K(i) = \frac{1}{\sqrt{2\pi\sigma^2}} * e^{-\frac{i^2}{2\sigma^2}}$$

i was the position in the matrix, σ was the standard deviation. The size of the kernel dictates the image's sensitivity to noise and the standard deviation alters the intensity of the blurring. A kernel length of 9 and standard deviation of 2 was used, as this was experimentally optimised by last year's team.

Next, the intensity of the gradients between pixels in the image is calculated using the Sobel derivative.

$$S(i) = I(i+1) - I(i-1)$$

This calculates an array of gradients, with the highest absolute values indicating where a line was detected, i.e. a strong positive gradient possibly indicates white track to black outer line.

Non-maximum suppression was then performed on the array of gradients to only leave it with absolute local maximums. This was done by checking each pixel to see if it has a greater absolute derivative value than its two neighbouring pixels. Any non-local maximums are removed as they are not needed.

Threshold comparison was done to determine the most likely track edges out of all edges detected.

5.3 Track Detection

5.3.1 Position Detection

To detect the position of the vehicle on the track, Canny edge detection is used. This allows the vehicle to detect the position of the edge of the white track and the black line by calculating the derivatives of the image from the main linescan camera [22]. The Canny detection was improved for blurred images as this blurring reduces noise on the images which can usually provide an error. Data received after Canny detection will be used to determine the position of the vehicle on track from

the position of right and left black lines. Image processing was done in *LineDetectionDouble.c* and the *findLine* function was used to return the error from the centre of the track.

The raw values returned from the edge detection are stored in arrays and the data was converted into an array of transitions. A transition was how the edges are described in the software in terms of their position on the track, derivative values and also direction of the edges. Direction of an edge was determined by differentiating if the edge is black to white or white to black. Hence, the direction of the edges also determines if a detected edge is a left or right black line. Line and track objects are created by the transition which hold information about lines and are used to create track objects with information such as the track width, the centre of the track and also the certainty of the track detected.

The track and line objects created were kept separate to provide flexibility when implementing the data in the software. By having a multi-level abstraction which consisted of arrays of derivative values at the low level and line and track objects at a higher level, a simpler and more flexible algorithm can be used.

In order to ensure that the correct track was detected, each track had a certainty value calculated. This certainty value was used as the linescan camera may be detecting 3 or more lines at once and hence will have multiple possible tracks. The highest certainty track will be followed, with the other tracks assumed to be objects outside the track. In order to increase the performance of correctly calculating certainty, Gaussian probability distribution is used.

To summarise the program flow of the position detection:

- Edges detected by Canny edge detection are converted to transitions
- Line objects are built from the transitions
- Track objects are created from the line objects
- The track object with the highest certainty value is used to determine the position of the vehicle
- If a track is not detected, the vehicle will follow the single line with the highest certainty value.

The linescan camera compares the latest captured image with the previous image captured, to aid in calculating certainty, as the line that is similar to the previous has a high probability of being the correct line. Line objects contain information about the line positions compared to the line positions detected in the previous frame. Furthermore, the derivatives of the lines detected will also be compared with the derivatives of the previous image. Similar to line objects, track objects also contain information about the position of the track compared to the previous image.

If the certainty calculated was above the set threshold value, the measured track width and track position are used to accurately determine the position of the vehicle on the track. The difference between the track centre and the car centre can be determined, this is fed to the servo to steer.

Due to the limitations of the camera focal length, the vehicle was not able to see the full track whilst cornering. The majority of the image captured will be outside of the track, with the track consisting of only 40% of the image. Thus, when cornering the vehicle only follows a single outside line. As the track width was consistent throughout, the vehicle can use this to calculate the centre of the track from the single line.

The vehicle enters the line temporarily lost' state once the vehicle has failed to detect any reliable lines on the track. This will cause the vehicle to continue following the previous reliable track

positions until a reliable line or track was found. If no lines can be detected within a specified time, the vehicle enters the 'line lost' state which immediately stops the vehicle in order to ensure it does not collide with any objects and cause damage to the vehicle.

The detection algorithm was reliable in eliminating errors in images received by the linescan camera such as objects outside of the track as well as random objects on the track, such as tyre marks or shadows.

5.3.2 Distance Measurement

The main linescan camera was positioned horizontally to detect the track however provides limited information about the track in front of the vehicle due to its short look-ahead distance as it was pointing downwards. Hence, to obtain useful information about what was happening in front of the vehicle, a second linescan camera was implemented. The information provided by the second line scan camera was used for speed control, detecting cross sections and s-mode.

The second line scan camera was implemented vertically and positioned as high as possible to achieve a greater look-ahead distance. Vertical orientation allows the camera to detect black lines up to 2.5m ahead of the vehicle. Further distances are unreliable due to the weakness of the derivatives obtained. The camera image of the second linescan camera was obtained in pixels and was converted to distance in metres but requires accurate mapping of pixels to distance via the *CameraFeed* MATLAB script.

Distance measurement was obtained by finding the first white to black transitions on the vertical camera, which represents the nearest black line in front of the car. All black lines detected further ahead were ignored as they are deemed outside the track. The obtained distance was used for target speed, straight mode, cross-section detection and s-mode. The vertical camera needs to be calibrated and focussed in order to provide an accurate measurement of the forward distance. Forward distance measurement was carried out in *findLineDistance* which can be found in *LineDetectionDouble.c.*

5.3.3 Stop Line Detection

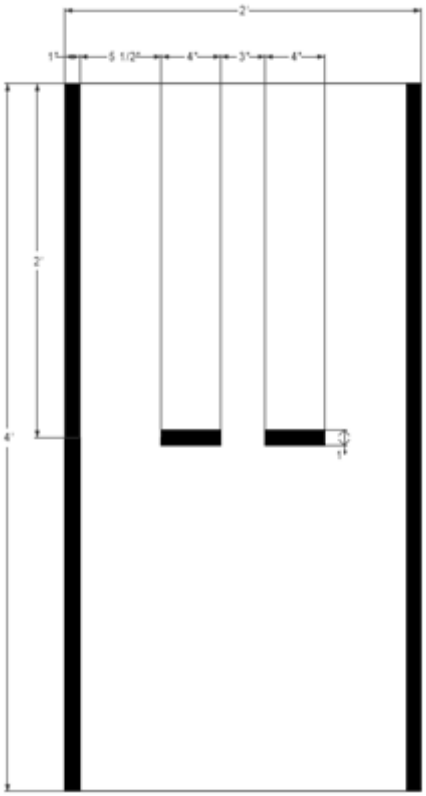


Figure 5.3.3.1 – Start/Stop Line Section

Stop Line detection was required during the competition races, as a 1 second penalty would be added to the lap time if after the vehicle crosses the start/finish line, the vehicle fails to not stop within 2 metres/6 foot or leaves the track.

The previous year's team stop line algorithm, situated in *LineDetectionDouble.c*, would create a pair of possible stop lines for every line detected and store in a stop line structure. For each pair of stop lines, it also calculated the size of the gap between the lines. It then used the standard deviation, mean and percentage of track width of both the width of the stop lines and the width of the gap between them, in pixels, to calculate the probability that a pair of stop lines were the actual stop lines of the track. If one of the stop line pairs had a calculated probability above a set minimum certainty threshold, the vehicle would enter the 'stop line detected' state.

As the stop line is detected before the vehicle crosses the stop line, a delay is added, in *main.c*, before the vehicle attempts to stop. The delay uses the look ahead distance, the distance the vehicle's top camera sees ahead of the front of the vehicle, to calculate the actual delay.

Once the delay has passed, a measurement of the speed of the vehicle is taken. If the vehicle is travelling above a threshold speed, the motors reverse direction to rapidly decelerate the vehicle until the measured speed is below the threshold, at which point the motor's PWMs are set to 0.

The reliability of the stop line detection algorithm greatly depended on how central the vehicle was to the track and the quality of the lighting conditions. Approaching the stop line after a straight section gave a higher chance the stop line was detected, approaching the stop line after a corner section reduced the accuracy. There were also issues where the vehicle thought the shadows caused by the bump sections or the beginning of a hill section were stop lines and poor lighting vastly reduced stop line detection.

Multiple different stop line detection algorithms were devised and tested in order to increase the probability of detecting a stop line while the vehicle is not perfectly straight whilst also reducing false detections. Table 5.3.3.1 lists each stop line detection algorithm and a short description, all of which only execute if at least 3 lines are found.

N	Description
0	Original Algorithm
1	Looks for 1 stop line width within mean +/- standard deviation
2	Looks for at least 2 stops line with widths within mean +/- standard deviation
3	Looks for 1 stop line gap width within mean +/- standard deviation
4	Looks for 1 stop line gap width and 1 stop line width within mean +/- standard deviation
5	Looks for 1 stop line gap width and at least 2 stop line widths within mean +/- standard deviation

Table 5.3.3.1 – Stop Line Algorithms' Specific Success and Failure Rates

The additional algorithms were found not to provide any advantages over the original algorithm. They proved to be either too broad, causing the vehicle to repeatedly incorrectly detect a stop line (algorithm 1) or too specific, causing the vehicle to never detect a stop line (algorithm 5). The probability based algorithm originally devised by last year's team was decided to be the superior algorithm and was used in competitions and testing.

5.3.4 Cross Section Detection

The cross-section proved to be the most difficult section of the track to reliably detect. This was due to the fact that the vehicle had to be as straight as possible and be at the centre of the track when entering the cross section. If the vehicle was not passing through the cross-section straight, then the vehicle may only be able to detect one side of the cross section and misinterpret it as a turning, incorrectly passing through the cross-section and failing the lap. If the vehicle was perfectly straight before entering the cross-section, it will enter the 'line temporarily lost' state and continue to use the previously detected left and right lines to drive straight through the cross section. Once leaving the cross-section it will find the lines again and begin to drive as normal.

On some occasions, the vehicle will initially begin to incorrectly detect the cross-section as a turn and then while turning will detect the lines ahead of the cross-section and correctly pass through the cross-section. This is inconsistent and reduces lap speed due to the initial incorrect turning.

To ensure that the cross-section can be completed consistently, a way to detect and pass through the cross-section was implemented. The cross section detection will straighten the vehicle when a cross section was detected allowing it to pass through successfully. Cross-section works by noticing the differences between a normal turn and a cross-section piece. The cross-section pieces consist of black lines at 90 degrees that give a large difference between the previous detected line and the current line, whereas normal corners have a smoother curve. The derivatives detected by the camera between the previous track need to be high in order to differentiate between a cross-section and a turning.

There were problems faced when implementing cross-section detection. As the horizontal linescan camera only reads a slice of the track at a time it is difficult to accurately differentiate the cross section with normal corners, even with the above algorithm implemented. The probability of the cross-section being detected directly after a corner was low, this was due to the vehicle entering the line temporarily lost state when entering the cross-section and using the previous track position of the corner to have the vehicle continue to turn, incorrectly passing through the cross-section. It was found that in order to maximize the probability of detecting the cross section, the horizontal linescan camera must be calibrated by repeatedly altering the focal length and width.

5.4 Steering Control

For this project, the 'follow-the-carrot' path following method was used instead of other advanced path following methods such as pure pursuit, Stanley method and vector pursuit [23]. This was due to the image captured from the cameras being insufficient to perform those advance path following methods. For example, pure pursuit path following mode requires extensive data, such as the curvature angle from the rear wheels to the target path which is impossible to obtain accurately using linescan cameras [23]. Other than that, these complex algorithms will require large data processing that is not suitable to be used with FRDM-KL25Z. Hence, 'follow-the-carrot' was chosen due to its simplicity and reliability.

This method has the vehicle attempt to remain at the centre of the track at all times and this will aid passing through hill sections, cross-sections and detecting stop-lines as for accurate detection these require the vehicle to be at the centre of the track. 'Follow-the-carrot' works by using the horizontal linescan camera to detect the vehicle's offset from the centre and steer the servo to reduce this error [24]. This requires accurate calibration of the horizontal camera in order to provide a reliable track reading to determine the centre.

Before the steering control PID tuning was done, all the potential sources of error needed to be eliminated or minimized, such as: reflections, shadows and marks on the track surface and vibrations of the camera. The camera focusing, focal length and exposure time needed to be calibrated properly using either the live camera feed from MATLAB or the oscilloscope.

PID constant of the algorithm can be adjusted in *SteeringControl.c* for turning and it is recommended to tune the PID of the steering control every time the horizontal linescan camera angle is changed. The proportional control of the PID determines how fast the orientation error will be corrected by multiplying the orientation error with proportional constant to determine the steering magnitude for the servo to operate, stated in the equation:

$$\varphi = K_P \times e_o$$

Where φ, is the steering magnitude, K_P, is the proportional constant and e_o, is the orientation error[24].

To tune the PID constants, trial and error was used as it was the simplest and quickest way to tune the PID controller. Tuning process of the PID can be made easier by using the telemetry and MATLAB to view how closely the vehicle follows the centre line. First, the proportional constant needs to be tuned first with zero integral and differential components. Increasing the proportional constant increased the speed of reducing the orientation error, which results to faster steering. The proportional constant was increased until oscillations in steering begin to appear.

These oscillations are then smoothed by added integral and differential components. Increasing the integral and differential errors increased the accuracy of PID controller by taking into account the past and future errors. Hence, the integral constant was increased until steering oscillation was reduced. If the integral constant was set too high, greater oscillation will occur. Thus, setting the integral constant to a correct value is important.

The servo offset can be adjusted in *Settings.h* to ensure that the servo was orientated correctly. In order for the servo offset to be corrected, the servo value was set to zero using the *TFC_SetServo* function. This should point both front wheels should point straight ahead, any offset needs to be removed.

There are a few disadvantages of using 'follow-the-carrot' path following method. The vehicle will begin to cut corners if the horizontal camera was looking too far ahead as the vehicle was reacting to the track before it has reached the corner. Having the cameras not looking far enough ahead causes the vehicle to react to the track too late, steering late on corners and reducing overall lap speed.

The previous group had also implemented an 's-mode', developed to allow to vehicle to pass through chicanes faster. It detects a chicane by noticing that the track position will change at a higher frequency. Without the s-mode, the vehicle will slow down for every corner, reducing lap speed. S-mode is implemented by low-pass filtering to remove the high frequency components of the track position, hence allowing the vehicle to steer less during chicanes and pass through them faster.

The s-mode also helps cross-section detection as the sudden change in line position from the 90 degree corners of the cross-sections emerge as high frequency changes in track position and will be filtered out by the s-mode.

S-mode is deactivated automatically when a turning is detected by the bottom camera, allowing the vehicle to accurately steer for normal corners.

5.5 Steering Radius Mapping

The servo is controlled by the *TFC_SetServo* function which takes in a dimensionless number between -1 and 1. Last year's team used measurements to derive a mapping between the dimensionless value and the real world steering radius value, implemented in *ServoMapping.c*. This was done to find the square root of the steering radius which is used for the active differential and the target speed in order to improve the vehicle's ability to handle turnings at high speed. Due to mechanical alterations to the servo, the steering radius mapping was re-done to ensure accurate measurements.

The equation to calculate the steering radius is:

$$R = \frac{w}{\sin(\alpha)}$$

The distance between the centre of each wheel, w, was measured as 0.2m. The servo was set at intervals of 0.05 in the maximum turning range from -0.4 to 0.4 and α was measured at each point. No readings were taken under a certain limit as the steering radius began to grow exponentially and can be regarded as the vehicle driving in a straight line.

Due to slight differences in the length of the servo arms, as well as the physical geometry of the servo cogs, the steering radius values would not be equal for both sides and thus a separate measurement was taken for each direction of turn.

Servo Value (dimensionless)	-0.4	-0.35	-0.3	-0.25	-0.2	-0.15	-0.1	-0.05
Steering Radius (m)	0.28	0.31	0.37	0.45	0.54	0.74	0.92	1.53

Table 5.5.1 – Steering Radius Right Turn

Servo Value (dimensionless)	0.4	0.35	0.3	0.25	0.2	0.15	0.1
Steering Radius (m)	0.36	0.44	0.51	0.74	0.88	1.18	4.18

Table 5.5.1 – Steering Radius Left Turn

These tables were graphed in order to determine the relationship between servo value and steering radius for both left and right turnings.

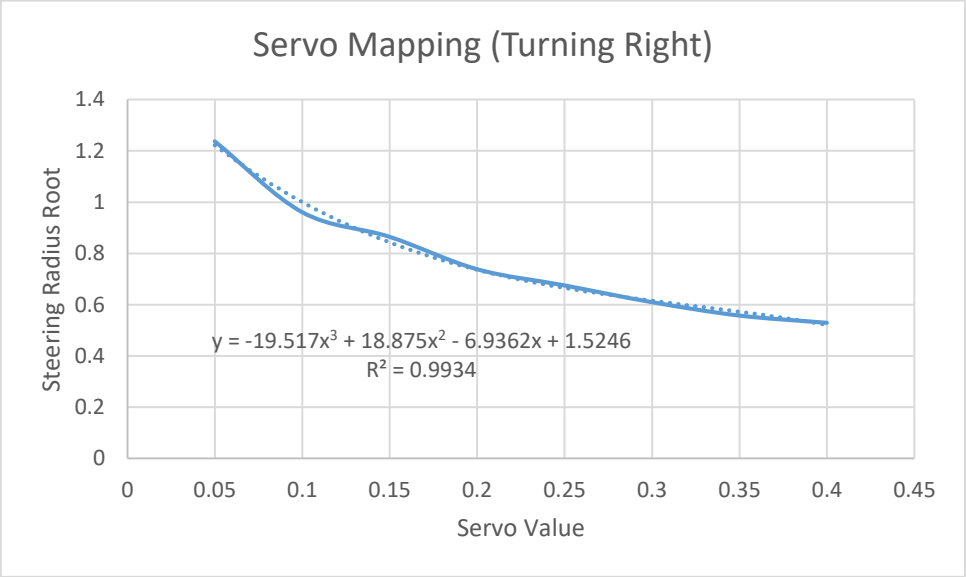


Figure 5.5.1 – Servo Mapping Relationship Right Turn

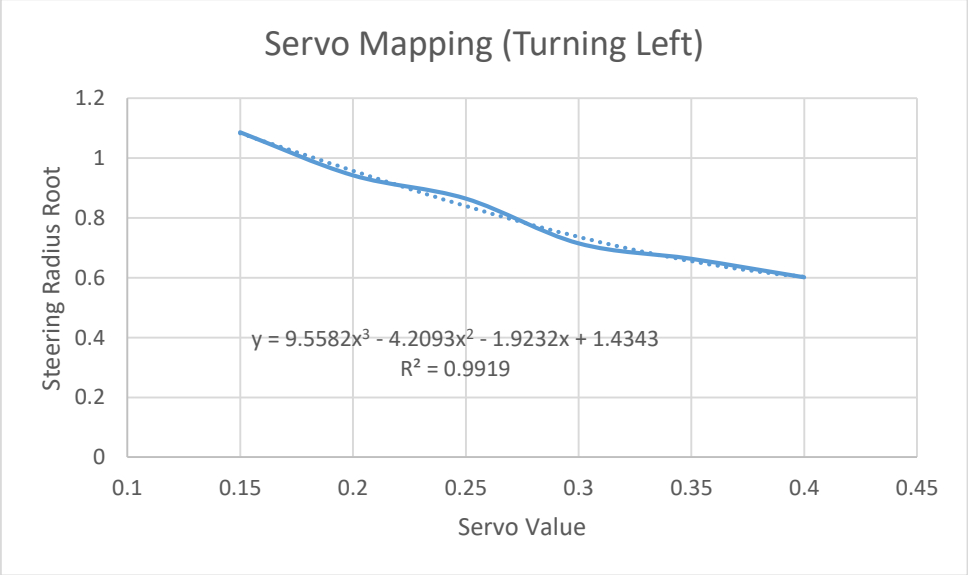


Figure 5.5.2 – Servo Mapping Relation Left Turn

A third order polynomial was used as it gave an R^2 value of >0.99 for each direction. Each equation was used to create a look-up table of 101 values, where a real world value of steering radius can be taken for any servo value between 0.05 and 0.4 for each side.

5.6 Speed Control

5.6.1 Target Speed Control

Speed control was used to optimise lap times by having the vehicle speed up for straight sections of the course and slow down for turnings.

Target speed was the speed the vehicle is currently trying to achieve. It was used as the set-point for the speed PID controller. Previous to last year's team, an algorithm was used which used the value of the error from track centre in order to control the target speed. If the vehicle is travelling on a straight piece of track, it should follow the centre of the track with low track error, and thus target speed should begin to increase. When the vehicle approaches a corner, the error will begin to grow as the vehicle turns and the target speed will decrease appropriately in order to slow the vehicle down so it can make the turning.

Last year's team improved on this further by considering the accelerating and decelerating forces working on the vehicle. It was found that zero torque should be applied when the vehicle was turning to keep it in constant speed. For constant speed whilst turning to be achieved, the vehicle must reduce speed for the corners. A formula derived was:

$$speed_{max} = \sqrt{\mu g R}$$

μ is the friction co-efficient, g is the gravitational constant (9.81 m/s^2) and R is the steering radius. The equation was used within *TargetSpeedControl.c*. The friction co-efficient was found through testing and can be altered with one of the potentiometers. Lowering the value reduces the chance of slipping, but also reduces speed. Increasing the value increases the chance of slipping, but improves lap times. The steering radius was found through the steering radius mapping.

As the steering radius was used for the target speed, it was desired that the steering be as smooth as possible in order to give a smoother change in target speed. To smooth the value, the servo value was low-passed before being used to set the target speed. This removed any sharp changes in servo value, which would cause the vehicle to not move at a constant speed during turnings, thus reducing the ability to take turning at higher speeds.

5.6.2 Straight Mode

In order to improve speeds on straight sections of track, a 'straight mode' was used. This is activated using both the second camera and the servo value. When the second camera could not see a line ahead within a certain distance and the servo value was at a low value, such as when driving in a straight line, the vehicle's target speed no longer uses the previously stated algorithm and was set to a maximum achievable speed.

When a line was detected ahead within a certain distance, this implies that a turn is approaching and the vehicle leaves straight mode and begins to use the target speed formula again. The straight mode distance needs to be low enough in order to not reduce handling of corners, however high enough to go into straight mode as soon as leaving a corner and moving on to a straight.

5.6.3 Active Differential

If both wheels move at the same speed when the vehicle was cornering, slipping starts to appear at higher speeds. In order to reduce slipping, the wheel speed must be modified with an active differential. The outside wheel must move at a higher speed than the inside wheel. Last year's team implemented the following active differential algorithm in *ActiveDifferential.c*:

$$speed_{AD} = \frac{R \pm d}{R} * speed_{max}$$

R is the steering radius and d is the distance from the centre of the rear wheel to the centre of the vehicle. Again, the servo mapping was used to calculate the steering radius. This algorithm was tuned further to add a co-efficient that would further increase the speed of the outer wheel and further decrease the speed of the inside wheel. This lead to the vehicle slowing down more on the corners and performing sharper turns, however allowed the vehicle to approach corners at higher speeds and thus actually improve overall lap times on some track layouts that had less corners and more straights.

For this reason, a function button was used so that one of the potentiometers on the vehicle could alter the co-efficient, so during competition races the co-efficient can be altered depending on the layout of the track.

5.6.4 Motor Speed Control

The speed of each motor was controlled by the low-level *TFC_SetMotorPWM* function. Pulse Width Modulation (PWM) was used to set the duty cycle of each motor from +1 to -1, with +1 being full speed forward and -1 being full speed in reverse. PI control was carried out on each motor using the target speed and the measured speed, via the Hall Effect sensors, as inputs, with the duty cycle of the motor PWM as the output, limited between set values.

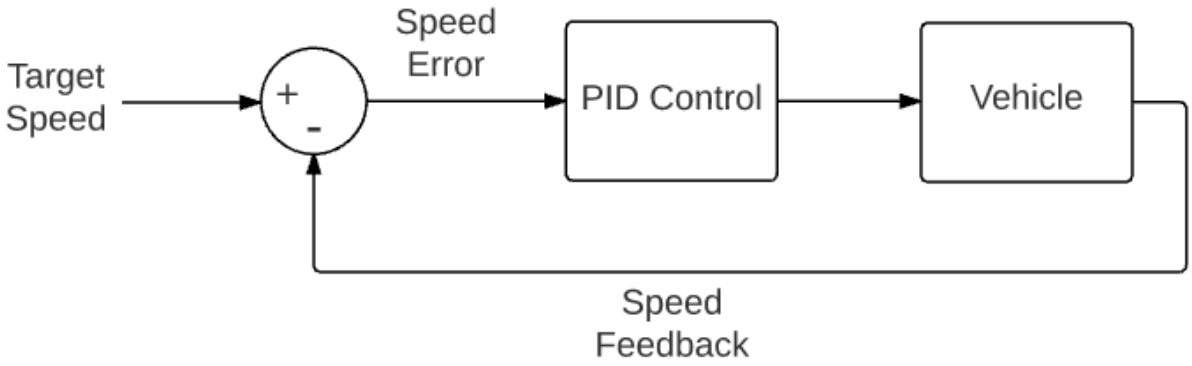


Figure 5.6.4.1 – Motor Speed Control

There were two improvements to be made on last year's PID algorithm. First the *dt* for the integral term was shared across each motor. This caused incorrect integral values to be calculated for the second motor to have PID control to it, causing incorrect motor PWM calculation. This was solved by using a second timer for the second motor. Secondly, the vehicle operated incorrectly, causing the wheels to lock up, when the desired speed was less than zero. This was fixed by having the output PWM set to zero when the target speed was less than zero, effectively having the vehicle slow down by rolling to reduce speed instead of temporarily having the motors reverse direction to slow down.

Although the PID controller was saturated the majority of the time, i.e. at maximum or minimum, the PID values must be tuned in order to maintain a constant speed when turning in order to reduce lap times and improve control. Below shows how well the vehicle follows the target speed (blue) while increasing the proportional constant. The red and yellow lines show the measured speed of the left and right motors.

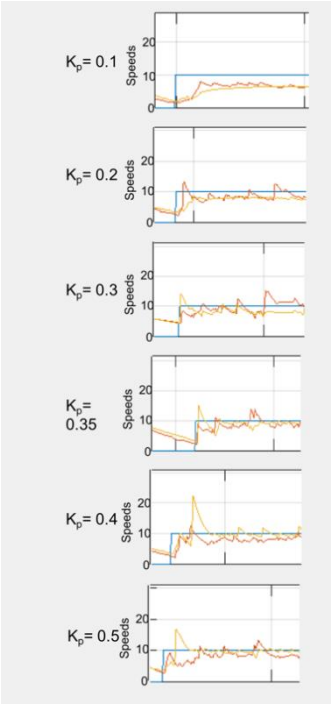


Figure 5.6.4.2 – Effect of Increasing Proportional Gain

5.7 Torque Control

It was speculated that using torque control, via current feedback, instead of speed control, via Hall Effect sensors, would lead to a more accurate and reliable way of measuring the speed of each motor and in turn allowing for more accurate control of the motors in order to improve the vehicle's turning abilities. The torque can be calculated from the current using the equation:

$$\tau = k_t * I$$

τ is torque of the motor, k_t is the 'torque constant' and I is the current feedback of the motor.

The TFC-Shield initially used had significant noise on the current feedback, making readings unreliable to be used for accurate torque control, however the custom PCB had the addition of two low pass filters on the current feedback paths reducing the current noise and allowing torque control to be used.

The value of torque constant was calculated using the telemetry from the vehicle in order to take readings of current and speed, via the Hall Effect sensors, in even steps of constant PWM speed. These readings were then plotted against each other in order to find the torque constant, which was the gradient of the graph. As the speed was given in CSU, this calculation also gives the torque in terms of CSU, however this was desired as stop-line and cross-section detection use speed values in CSU to make decisions. To save this code being modified for when torque control is to be used, it was decided to leave the torque in terms of CSU.

As shown below, in figures 5.7.1 and 5.7.2, the original motors used had vastly different relationships between speed and current, giving different torque constants. The line of best fit only had an R^2 value of approximately 0.88 for each. For improved torque PID control, motors that are well matched are greatly desired.

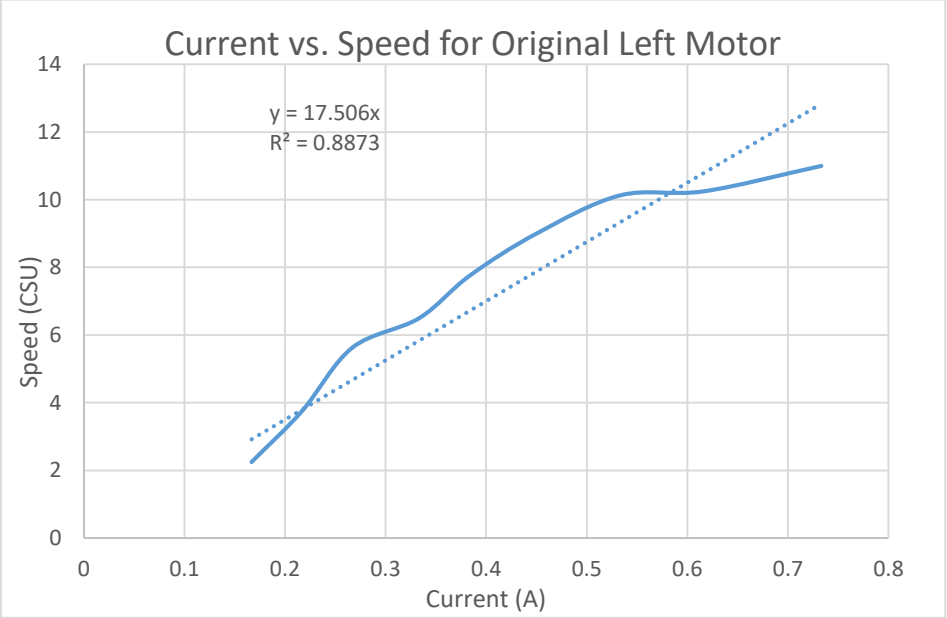


Figure 5.7.1 – Calculation of Torque Constant for Original Left Motor

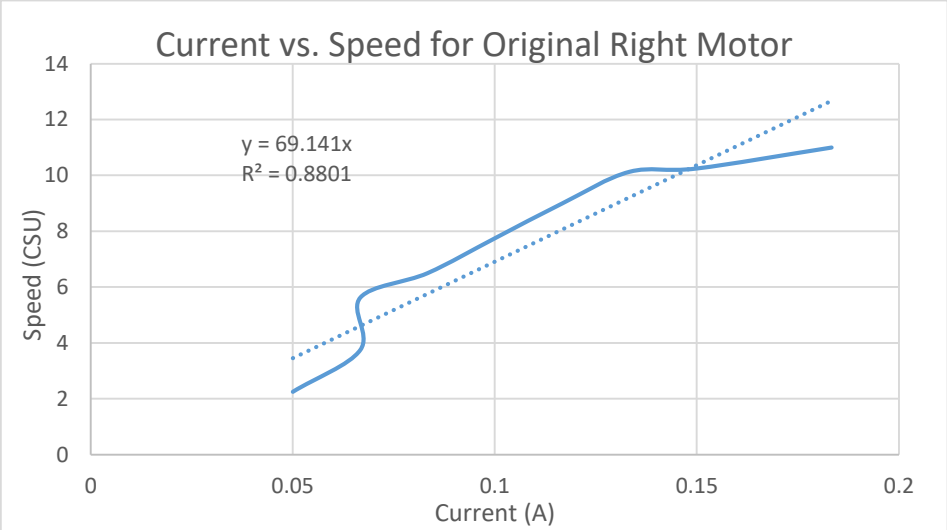


Figure 5.7.2 – Calculation of Torque Constant for Original Right Motor

All available motors had their current and current measurement tested on the vehicle in order to find the best matching pair, which were found and graphed to calculate the torque constant.

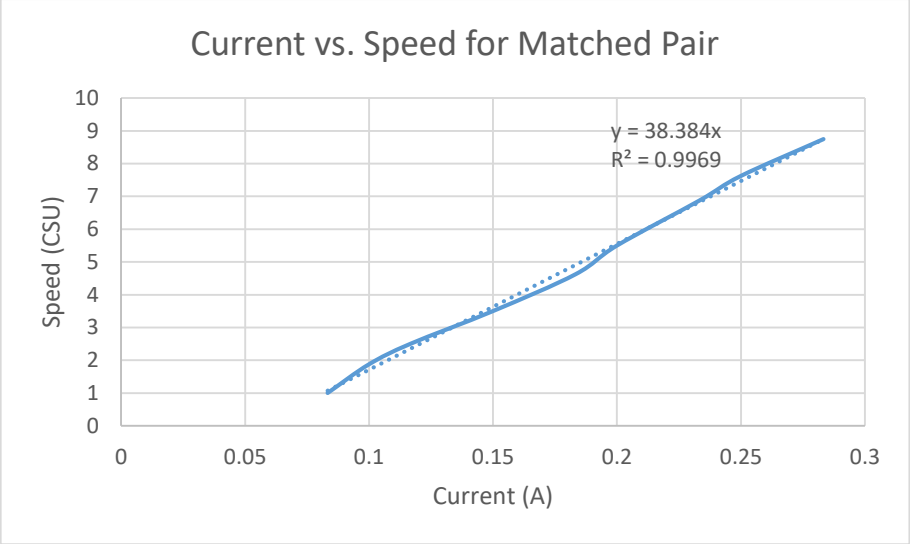


Figure 5.7.3 – Calculation of Torque Constant for Matched Pair

There are two different ways of implementing torque control, using two loops or one loop [25].

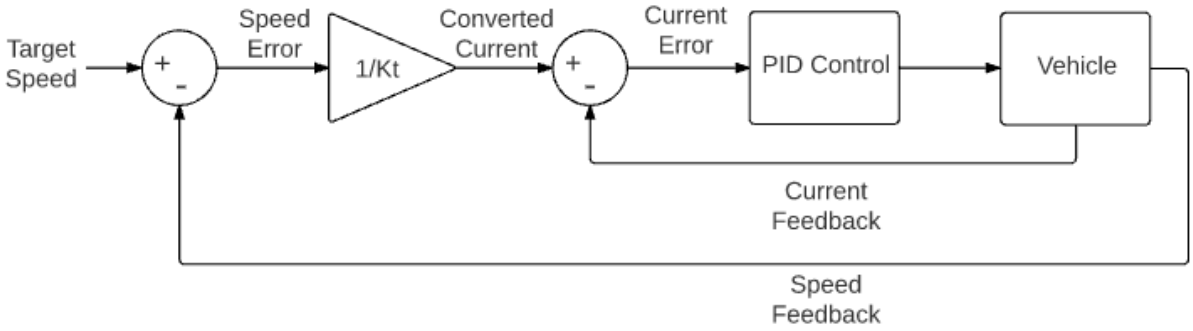


Figure 5.7.4 – Torque Control with Two Loops, Speed Outer Loop

In the two loop method, target speed was compared against measured speed via the Hall Effect sensors to find a speed error. This error is converted into current using the calculated constant for the motor. This converted current was compared against the motor current obtained from *TFC_ReadCurrent*. PID control is then performed to control the vehicle. Two loop control can also be used with current as the outer loop, as shown below.

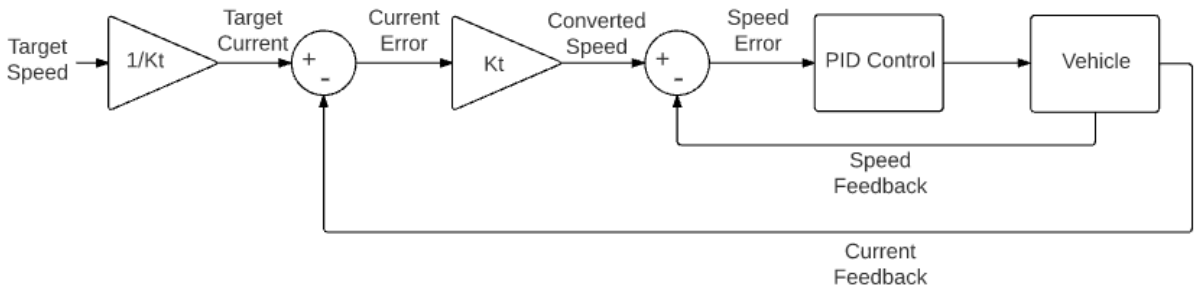


Figure 5.7.5 – Torque Control with Two Loops, Current Outer Loop

There are also two methods of single loop torque control.

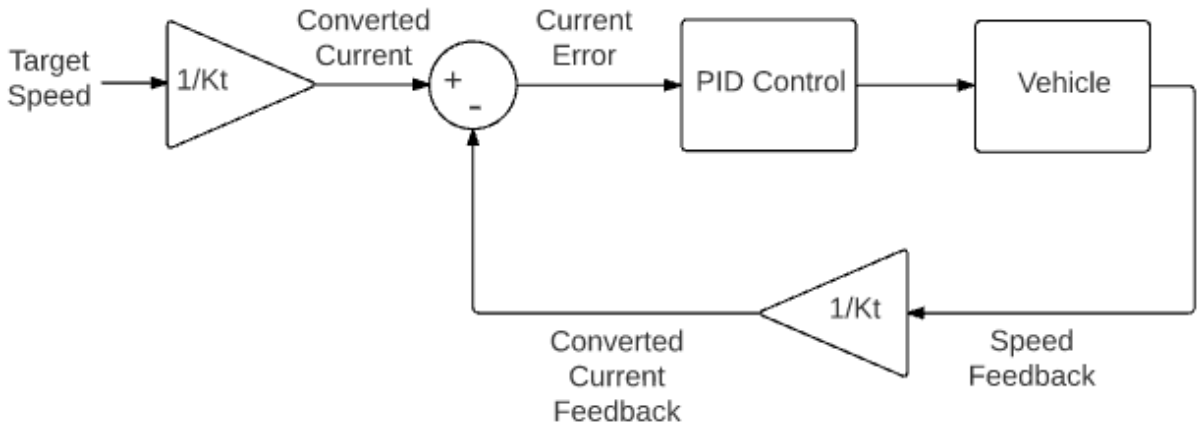


Figure 5.7.6 – Torque Control with Single Loop, Speed Feedback

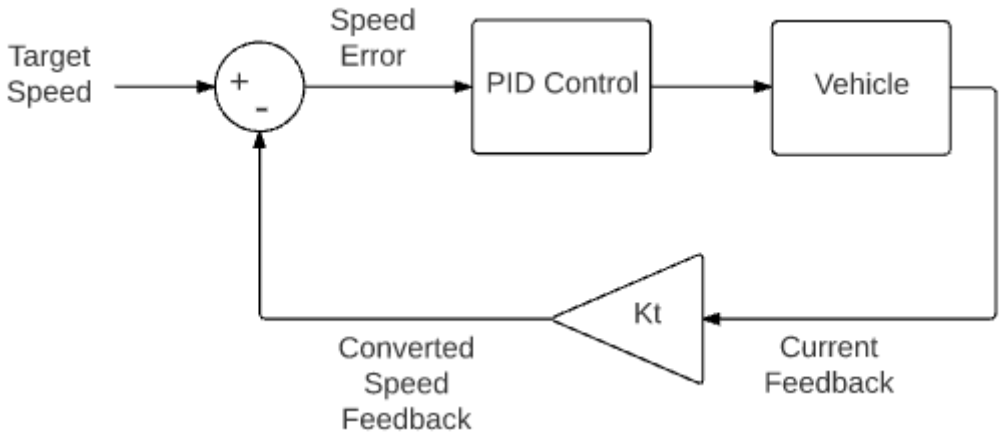


Figure 5.7.7 – Torque Control with Single Loop, Current Feedback

Although two loop methods have a higher level of control, the main purpose of using torque control was to remove the need for the Hall Effect sensors. The only torque control method that does not require Hall Effect sensors is the single loop with current feedback, therefore this was selected to be used. This also had the benefit of providing current in terms of speed. This was useful as both the stop line and cross section detection use a derivative calculated via speed in CSU and the stop line detection delay uses the speed calculated in CSU. If the torque control did not use speed in CSU, these would have to be altered.

The PID control was implemented in code in a similar way to speed control, by using the desired torque, measured torque and a measurement availability flag to set the motor PWM.

The desired torque was a target torque being calculated using the bottom camera, which was positioned so it reads vertical lines to determine if a turning is ahead. If there are no turnings or lines detected ahead the vehicle goes into ‘straight mode’, where target torque is set to a maximum, determined by the maximum stall torque and current of the motors. If a turning was detected ahead, the vehicle’s target torque became a function of ‘max speed’ (the maximum speed the vehicle can take a corner without leaving the track) multiplied by a value determined by the potentiometer on the vehicle, effectively allowing the user to control the speed of the vehicle when not in ‘straight mode’.

Measured torque is calculated using the *TFC_ReadCurrent* function, taking the current at each motor, and multiplying it by the calculated torque constant in order to convert it into a speed. The accuracy of the measured speed was critical for good PID control, hence why the measurement of the torque constant and having a good pair of matching motors with proportional torque constant at all speeds was important.

The measurement availability flag was used to ensure the motor PWM was only set once for each current reading, it was set high in *TFC_ADC.c* whenever the ADC reads a new value of current.

The PID constants were originally calculated in MATLAB's Simulink environment using values from the datasheet of the given motors, however due to motors having differences from ideal values due to manufacturing and general usage, these values were only used as a general guide for PID constants. Actual PID values were calculated through observations of the vehicle's performance on a track, with the constants being altered through telemetry.

Torque control could be further improved if a two loop control method was used, however this would require a more accurate speed reading via the Hall Effect sensors.

5.8 Program Flow

The main body of the controlling program runs within the *main* function in *main.c*. The vehicle first initialises all peripherals within *TFC_Init* before entering an infinite while loop that will cause the vehicle to repeatedly run *lineFollowingMode*, which contains all functionality for the vehicle following the track.

The table below shows the main functions, in order, that are ran in *lineFollowingMode*.

N	Function	Description
1	Processes terminal functionality	Only used when connected to terminal
2	Checks dip switches	Only used when in race mode
3	Reads potentiometer to set maximum speed	Allows changing speed for debugging
4	Read battery voltage	For low battery level indication
5	Process LED feedback	Indication of vehicle's current state
6	Write to LCD screen	Displays vehicle's motor speed and current
7	Perform telemetry	Write, read and perform any actions
8	Check accelerometer value	Used for bump and hill detection
9	Process camera image	Turns image into lines
10	Process found lines	Turns lines into track
11	Update servo	Steers vehicle depending on track
12	Look for a stop line	Uses pre-determined stop line images
13	Calculate auto exposure	Improves track detection in poor lighting
14	Calculate target speed	Uses second camera to look ahead
15	Read motor current	Used for torque control
16	Calculate active differential	Alters speed of wheels for improved turning
17	Detect speed of wheels	Used for speed PID
18	Perform speed or torque control	Performs PID control
19	Perform stop line detected functionality	Calculates delay before stopping

Table 5.8.1 – Main Loop Functions

Some functions such as reading the battery voltage, writing to the LCD screen, performing telemetry and checking the accelerometer reading are only done once every set time interval. The interval being controlled by the user using the low-level *TFC_Ticker* counters. Detecting the speed of the wheels and performing speed control was only performed once the low-level Hall Effect sensor drivers set a flag high via interrupts.

The torque control was only performed once a new current measurement flag had been set high via the ADC. This was done to minimise processing and only run functions when necessary, due to the image processing taking the majority of the processing time and being the most critical function to ensure correct performance of the vehicle.

5.9 Performance Feedback

Performance feedback was used to give more information about the current state of the vehicle. Last year's team used LED and wireless telemetry to take live measurements from the vehicle. The LED and wireless telemetry were improved, and an LCD was added for additional feedback.

5.9.1 LED Indication

The TFC-Shield used by last year's team had 4 LEDs, originally used to display the battery level, configured to display the current state of the vehicle. The LED feedback was moved to its own *LED.c* file.

The LEDs are turned on and off by manipulating pins 8-11 on port B. To ease readability macros to turn the LEDs on, off or toggle the LEDs were added.

Initially, additional functionality was added by having the LEDs display the desired max speed percentage through the value of the potentiometer. By having an LED light up for every 25% of desired maximum speed, a very rough estimate of the potentiometer value was achieved. This would then switch to display the vehicle's state once the 'start' button had been pressed.

An improvement on the rough estimate for maximum speed was to display the desired maximum speed percent in a binary representation. The 4 LEDs would allow for 16 bits of precision, allowing a binary counter that counted in steps of 6.6. This was deemed too complicated to use and displaying of desired maximum speed percentage was moved to the LCD screen when implemented.

The displaying of the current state uses a combination of 2 of the 4 LEDs, it was determined that adding multiple extra states displayed in 3 LEDs would reduce the clarity of the state reading, therefore the only extra state added to the LED was to display when the vehicle detected a bump section as it was easier to read from LEDs than the LCD screen.

LED Number				**State Indicated**
1	**2**	**3**	**4**	
OFF	OFF	**ON**	OFF	Right Line Found
OFF	OFF	OFF	**ON**	Left Line Found
OFF	OFF	**ON**	**ON**	Both Lines Found
OFF	**ON**	OFF	OFF	Line Temporarily Lost
ON	OFF	OFF	OFF	Line Lost
ON	OFF	OFF	**ON**	Cross Section Detected
ON	**ON**	OFF	OFF	Stop Line Detected
OFF	**ON**	**ON**	OFF	S-Mode Activated
ON	**ON**	OFF	**ON**	Bump Section Detected
ON	**ON**	**ON**	**ON**	Low Battery Level

Table 5.9.1.1 – LED State Indications

Only one state can be displayed at a time, as such they are sorted in order of priority as:

Low Battery Level > Stop Line > Bump Section Detection > Cross Section > S-Mode > Line Detection

Additional LEDs were a consideration for the custom PCB, however it was decided that the number of LEDs are sufficient as difficulty already exists in determining the state of the vehicle when it is travelling at high speeds and that an LCD screen would be required to display more accurate information feedback.

5.9.2 LCD Screen

In order to improve the visual feedback offered by the LEDs on the vehicle, it was decided than an LCD screen was to be fitted. Initially a 2x16 character display was used.

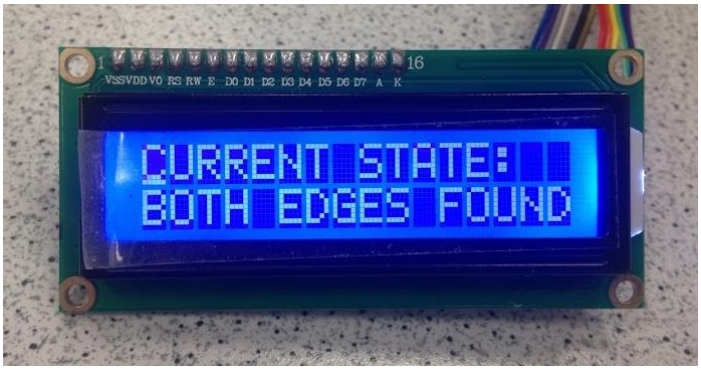


Figure 5.9.2.1 – 2x16 Character LCD Display

The LCD display simply prints out characters that are sent over UART. The display was programmed to display the current state of the vehicle, i.e. both edges found, cross-section mode, stop-line detected, etc. One of the function buttons on the PCB was used to cycle through different displays, for example displaying the current from each motor or the speed of each wheel. Although this proved useful for debugging, it was decided that a higher resolution screen was needed to display more information simultaneously.

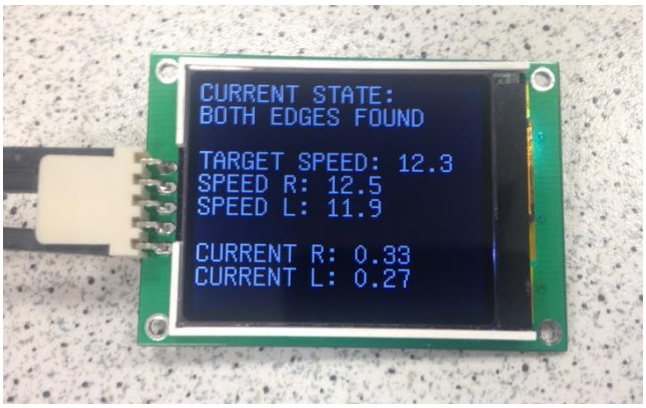


Figure 5.9.2.2 – 160x128 Pixel LCD Display

Although all of the information could be display via the telemetry, the LCD feedback was quicker and clearer. The improved LCD display also had the ability to draw to the screen instead of simply writing text strings. This was useful for debugging, however a method of using the LCD screen during competition races was required.

The ability to draw to the display allowed the live camera data to be drawn to the screen. This would be useful to calibrate the camera lens focus between attempts at competition races, if required. Another method of using the LCD display during competition races was to ‘freeze’ the screen once the line had been lost in order to determine the last track layout the camera had seen before leaving the track to assist in finding out why the vehicle had left the track.

This initially proved to be an insufficient method, however was expanded upon by saving the last 10 camera images, all 0.1s apart. A function button would 'scroll' through the images. This showed improvement over the initial method, however the majority of time it was still unable to determine exactly why the vehicle had left the track from these camera images.

An oversight in design of the custom PCB was that the LCD connections designed were only to use the LCD display over UART. As only UART2 is available to the user, it was used for both the wireless telemetry and the LCD screen. Therefore, the LCD screen and wireless telemetry cannot be used at the same time as attempting to send data over Bluetooth causes the same data to be written to the LCD display. An improvement would be to use an LCD display that uses I2C or SPI.

5.9.3 Wireless Telemetry

Wireless telemetry was carried out using a HC-05 Bluetooth module and was used to communicate between the vehicle's MCU and a laptop. *TFC_UART.c* contains many functions to transmit and receive data, such as *uart_putchar* to place a character onto the UART TX FIFO whenever space was available. This function was used in a *telemetryTransmitData* function in *main.c* by the previous team as their only method of transmitting data to a MATLAB script which would plot a live feed of numerous variables measured from the vehicle.

The MATLAB script was edited to show the servo value from the vehicle as this was determined to be a more accurate way of displaying how close the vehicle was following the centre of the line than the 'edge type'. The accelerometer value was added to the plot for debugging hill and bump section detection and the calculated torque was added to debug torque control.

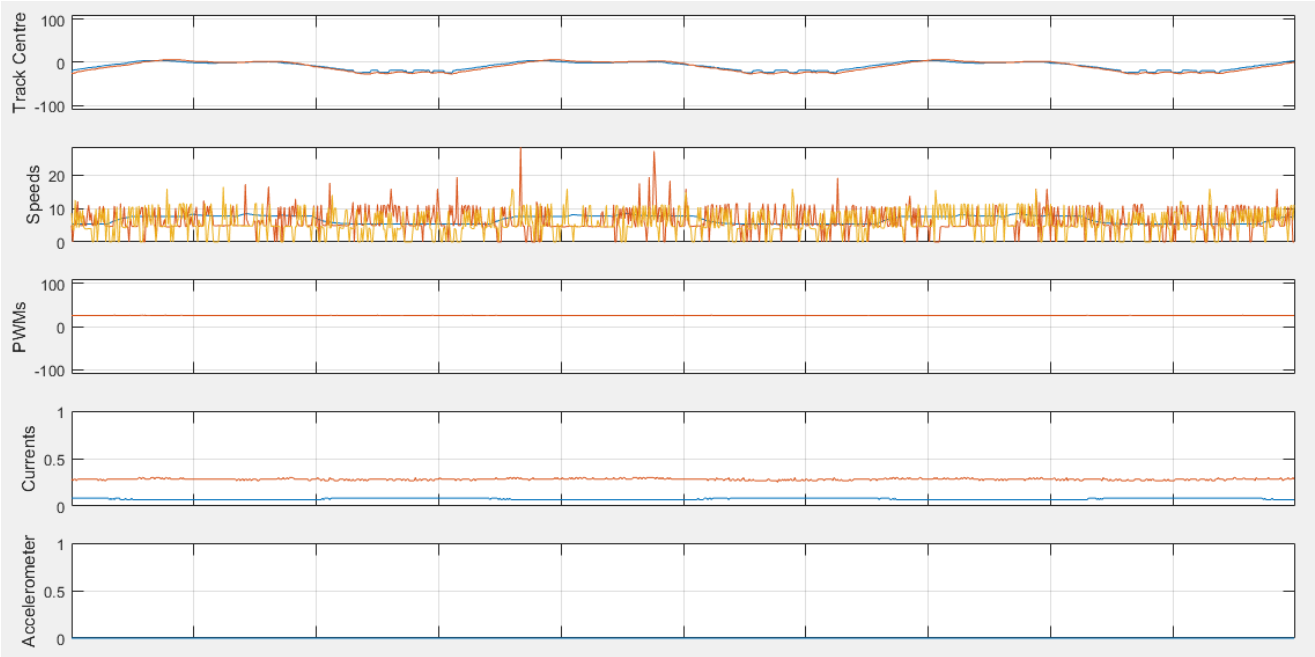


Figure 5.9.3.1 – Example of Telemetry Live Feedback

A new MATLAB script was created in order to transfer data from a laptop to the vehicle. Data would be three bytes long and consist of a byte each for: a Bluetooth flag, an identifier value and the desired new value. A *telemetryReceiveData* function was created in *main.c* to receive the data.

Initially, this was programmed as a state machine. The initial state would consist of the vehicle polling the receive data register for UART2, the Bluetooth UART, to check if the receive data register was full. If so, it would return that data, repeating this until the Bluetooth receive flag was received. When the Bluetooth flag had been received it would move to the next state where it would wait for

the next byte which would identify what type of data would be received in the next byte, i.e. a new speed percentage value, a new PID constant, etc. The third byte would be the numerical value of this data limited to a byte in size. After receiving the value, the identifier was used in a switch statement to determine the function to call, with the value passed as an argument. Each step would have an allotted time to receive the next byte of data or else it would reset back to waiting for the Bluetooth flag.

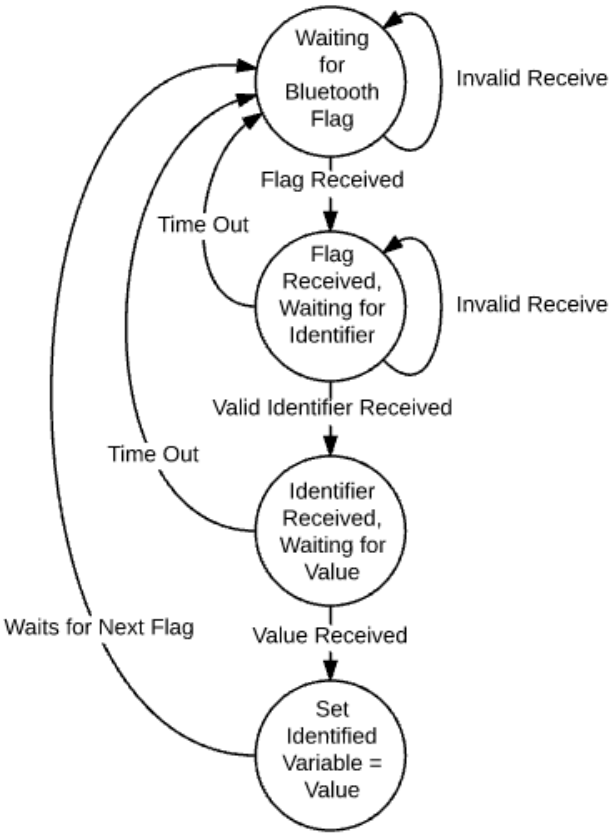


Figure 5.9.3.2 – Bluetooth State Diagram

Although this worked for writing single values, there was no way to queue up multiple bytes, meaning if it was desired to execute two receive telemetry functions consecutively, there was a strong possibility that the first would not be executed correctly, thus also causing the second to execute incorrectly. This method is also inefficient as it used polling of registers, which would take up processing time, even if no data was needed to be read.

In order to improve the telemetry, it was decided that:

- All telemetry related functionality would be moved into a new *telemetry.c* file
- The ByteQueue functionality configured for the SDA UART, UART0, used for UART2
- Using interrupts for UART2

The ByteQueue functionality, contained in *TFC_Queue.c* was used to handle an array of bytes and has previously defined functions for returning the number of bytes in the queue, adding data to the queue and popping data from the queue. Bluetooth transmit and Bluetooth receive ByteQueues were created in order to transmit data, via adding data to the transmit queue, and to receive data, by popping it off the receive queue.

The UART2 initialisation was edited to enable receive interrupts on UART2. A *UART2_Process* function was added after data was transmitted. This checks if the transmission queue is not empty, i.e. there was data to send, and the UART's transmit data register is empty, i.e. the UART is able to send data. If so, transmitted interrupts are enabled.

A *UART2_IRQHANDLER* function was created to handle interrupts. When an interrupt was received, the 'receive data register full' flag was checked, if it was high then there was received data on the data register, and this was placed on the Bluetooth receive ByteQueue. The transmit data register was then checked, and if it was empty and there was data on the Bluetooth transmit queue waiting to be transmitted, then this was popped off the queue and added to the UART's data register to be transmitted. Once all data has been transmitted, the Bluetooth transmit ByteQueue will be empty and transmit interrupts are disabled until they are enabled again by the *UART2_Process* function.

The *telemetryReceiveData* function polls the receiving byte queue to see if it had at least three bytes, which should be the Bluetooth flag, the identifier and the data value. If it did, the queue was looped through to find the Bluetooth flag. If found, the next two bytes were taken from the queue to determine the identifier and the value. The identifier was used in a switch statement to determine the function to call, with the value passed as an argument.

A basic method of checking the transmitted data for errors via modular sum was used. This was where the transmitted data was now made of 5 bytes: Bluetooth flag, identifier, identifier checksum, value, value checksum. The checksum would be the twos compliment of the respective byte and they would be XOR'ed together on the vehicle to check the XOR sum was 0. It was not necessary to implement the checksum as the amount of errors received was negligible and it did reduce the overall data rate. Data rate was not a priority, however the main issue was that the additional checksum bytes increase the likelihood of one of the bytes having an error, thus causing the whole set of bytes to be discarded. Also, the fact that telemetry was only used in debugging and that bits in error would not cause a catastrophic failure meant that the checksum value was mostly disabled and unused.

The two-way telemetry allowed several features for debugging, such as changing PID constants while the vehicle is running, allowing the vehicles speed to be changed, remotely stopping the vehicle, turning on and off cross-section detection, etc. However, as establishing the remote connection via MATLAB and transmit data to the vehicle took longer than the time taken to flash the vehicle, as well as the fact that the motors needed time to cool between tests, the majority of the time any changes to values were done by re-flashing the vehicle.

5.9.4 Live Camera Feed

The linescan camera calibration process was done using LabVIEW by last year's team. Due to problems getting the LabVIEW to work successfully with the vehicle this year, initially the cameras were calibrated using an oscilloscope. Although this gives an accurate reading of the camera when static, it cannot be used when the vehicle is running. Hence, a wireless live camera feed was implemented.

The live camera feed was implemented by transmitting the array of camera data via Bluetooth to a PC running a MATLAB script. The MATLAB script displays the array as a graph of light intensity values. The vehicle first sends a start flag before the array is send to keep the MATLAB script synchronised with the vehicle's camera feed. Once the start flag has been successfully sent, the intensity of each pixel was sent, one pixel at a time, at the maximum baud rate. After the entire array has been sent, another Bluetooth flag was transmitted to indicate the start of the next camera image.

The wireless live camera feed was useful for camera calibration. This was due to the fact that calibration of the linescan cameras needed to be done often. Hence, the live camera feedback provided an easy way to calibrate the camera focusing, camera focal length, camera exposure time, camera look-ahead distance and the camera angle. In addition, important camera information during cross section, cornering and stop line can be obtained for tuning purposes in order to increase the car performance when going through these challenges.

The only issues with the camera feed was that data transfer, such as speed, current, accelerometer, etc., readings cannot be done at the same time. Also, as the baud rate for the Bluetooth could only be set to a maximum of 9600 the FPS of the feed is moderate and could be improved by using a Bluetooth module with a higher maximum baud rate.

6 Progress Summary

The initial vehicle provided was not deemed as a good starting point and the majority of the initial project work was spent getting the vehicle into a comparable state to last year's group 1 vehicle.

Luckily, the line detection algorithms from previous group 1 proved to be sufficient for Hotrod and were used. However, as all of the finely tuned control algorithms were tuned for a different vehicle that had major mechanical differences, a lot of time was spent tuning the vehicle.

Once the vehicle was in a working acceptable state, the first major improvement was the camera mounting. The previous mounting system used was poor and had problems with oscillations, providing insufficient camera readings of the track and causing steering to be unpredictable. After the new mounting system was implemented, major improvements were seen.

The new servo arm, new position and mechanical improvements to increase the length of the servo link arms vastly improved the turning of the vehicle. This allowed the vehicle to turn faster allowing it to achieve significantly improved lap times. Problems with damaging the servo cogs were remedied by the creation of a servo housing.

The implementation of the LCD screen allowed variables to be displayed which increased ease of debugging significantly, however a significant use for the LCD screen for competition races was not found by the team. The inability to use the LCD screen and the wireless telemetry at the same time was a minor issue.

Implementing the two-way telemetry vastly reduced the time taken to tune the vehicle as now PID constants for the steering and speed control could be done while the vehicle was running. This coupled with the data transmitted from the vehicle to a PC allowed incredibly useful information, such as how well the vehicle is following the centre of the track or how fast the vehicle is achieving target speed, to be viewed instantaneously and was a major benefit to the team.

The custom PCB was successful in completing its objective and provided much smoother current, allowing torque control to be implemented. However, more time was needed to further tune torque control to give comparable lap times compared to speed control.

A set-up guide is provided in Appendix E for future teams to use.

7 Competition Results

The first competitive race attended by the team were the UK round of the EMEA qualifiers held in London on the 5th of April 2016 [26]. 7 teams from the UK and Greece competed, with the top 3 teams qualifying to the EMEA finals. The competition took place in a single day, with the morning used for testing and the actual qualifying race in the afternoon.

During the testing portion of the day, it was obvious that although the vehicle was not the fastest at the competition, it was still faster and more reliable than the majority. Numerous other teams had problems with a turn straight after a hill section, however Hotrod did not. The lighting at the competition was inconsistent, with certain corners of the room having sections of track that the vehicle had a 0% success rate in making corners. To remedy this, the minimum and maximum exposure time was increased. This slightly improved overall reliability of the vehicle, however stop line detection still had a poor success rate.

The actual competition track was deemed to be standard difficulty compared to other 2016 tracks, with 1 hill section, 1 bump section, 1 chicane, 1 cross-section and 26% straights. The cross section had 1 entry from a straight section and 1 from a turn, which was seen as the standard layout. Hotrod was the last team to attempt the track, usually this would be a problem due to dust on the track reducing friction and reducing steering quality, however the room was relatively dust free and this was not an issue.

Three teams had previously set a time before Hotrod's attempt, 12.6 seconds, 13.9 seconds and 20.7 seconds. It was decided that the vehicle would attempt the track at 60% of maximum speed as this speed was found to give a high reliability of completing the track and would still be fast enough to at least beat the 20.7 second time to qualify. The vehicle successfully completed the track in 14.4 seconds with an average speed of 1.73 m/s, qualifying in third place.

Position	Team	Country	Time (s)	Average Speed (m/s)
1	Team LineRider	UK	12.6	1.98
2	Team 0x2A	Greece	13.9	1.79
3	Team Hotrod	UK	14.4	1.73

Table 7.1 – Table of Results

8 Recommendations

There are many possible improvements to still be made to the vehicle. However, time to research, develop and implement them was limited. The following recommendations stated here would be for any continuation to the project and are split into 3 parts: mechanical, hardware and software.

8.1 Mechanical Recommendations

The performance of the vehicle can be vastly improved by reducing weight, as experienced by using different batteries of different weight. The distribution of the weight about the vehicle also effects performance. Weights were made and placed on different parts of the vehicle in order to determine an optimal performance, however more tuning and testing needs to be done. A mount was created that allows the battery to be positioned on the back of the vehicle, however this requires re-tuning of the vehicle which there was not enough time for.

It was specular that heating the batteries before use would give improved performance. This would work by increasing the rate of the chemical reaction within the battery. A suitable method of reliably heating the battery without permanently damaging it was not found and this was not implemented.

The previous year's team recommended to build a custom chassis to be used instead of the stock chassis. This would reduce the weight of the vehicle and allow the camera mount to be incorporated as part of the chassis, possibly reducing vibrations on the cameras. Not enough time was available in order to design and create one, however it was speculated that this would give a large performance increase.

An idea of having the vehicle use rear wheel drive, having the servo control the back wheels and the motors on the front wheels, was discussed. This would allow the vehicle to obtain a much tighter turning circle. This would require a large amount of time to implement, test and tune with no guarantee of improved performance. Further research was needed to see if it would provide any benefit.

8.2 Hardware Recommendations

The LEDs used to aid the vehicle in poor lighting conditions worked successfully. However, the large number of LEDs pulled a large current, rapidly decreasing the battery life. A new set of LEDs could be created, with less LEDs on each to still give improved vision in poor lighting conditions. As competition races usually have decent lighting and the LED lights were not used often.

The linescan cameras proved sufficient in reading the track. However, CCD cameras that provide a real image of the track could be used in order to vastly improve cross-section and stop-line detection. This would require greater processing power and a complete re-write of the existing line detection code so would be a major task, however may provide the greatest benefit.

The size of the PCB could be further reduced in order to reduce weight and allow the PCB to be positioned on the vehicle differently in order to distribute weight. This could be done by using a 4-layer design. However, as stated in the PCB design section, this would be more difficult to replace tracks in the event of one faulting.

In order to increase processing power to reduce the time it takes to process each image captured by the cameras the FRDM-K64F could be used instead of the FRDM-KL25Z. This was initially considered, however the fact that all of the low-level drivers had already been written for the KL25Z meant more time could be spent on more important tasks. It is unable to determine if the increased processing power would provide an acceptable performance improvement considering the time this would take.

Due to an oversight, UART2 is shared by both the LCD screen and the telemetry, meaning that both cannot be used at the same time. Ideally the custom PCB is edited so that I2C or SPI ports are made available to connect to in order to program the LCD screen to use I2C or SPI, allowing it to be used simultaneously with the telemetry.

The speed readings via the Hall Effect sensors provide smooth readings at low speeds, however once speed begins to increase large spikes begin to appear even though the wheels are moving at a constant speed. It was suspected that noise or interference was the cause, however further research was needed.

The magnets on the cogs of the wheel that are used by the Hall Effect sensors only use a single rising and falling pulse per rotation. A circular magnet that provided numerous positive and negative poles could be used to give more readings per rotation and hence a more accurate speed reading.

8.3 Software Recommendations

Although the software the vehicle functions correctly, it has been repeatedly added to over the last few years. Many functions are unused and #define functions serve no purpose, i.e. it is required they are always defined for correct operation of the vehicle. To improve overall readability of the code, it would be desirable to re-program the vehicle from scratch.

To improve the vehicle's handling of hill sections and differentiating between a hill and a bump section, a hill section detection feature should be added. The accelerometer is useful for detecting sharp changes in acceleration, such as the vehicle crossing bumps, however for accurate hill detection a gyro-meter could be used. This will allow the vehicle to determine its 3D position, thus giving a more accurate prediction of when it is on a hill so the vehicle can speed up or slow down as appropriate.

Although torque control was implemented and works successfully, it does not provide comparable performance to the speed control. This should not be the case as the current feedback readings are low noise, whilst the speed readings from the Hall Effect sensors suffer from interference and give noise spikes, reducing the accuracy of the reading. Further time is needed to tune the torque control to give a fair comparison against speed control, however it is predicted that correctly tuned torque control will provide the superior method of controlling the vehicle.

Having the LCD screen display variables such as speed and current of each motor is useful for debugging. However, a use for the LCD display during competitions needs to be implemented. Currently the LCD screen is used to display the 10 camera images from the last second once the line is lost. This is sometimes useful to figuring out why the vehicle failed the track, however a more improved use could be implemented if found.

Cross-section detection and stop-line detection were implemented by last year's team and had small changes made to them. Their reliability is sporadic, however attempts to improve them, by adding different stop-line detection methods and by altering how the vehicle reacts to possible cross-sections showed no real improvement, and in some cases gave worse performance. As a turning into a cross-section proved to be the most difficult track layout for the vehicle to pass, further work on improving cross-section detection is required.

9 Conclusion

Considering the initial vehicle provided to the team was not able to successfully finish a lap and Hotrod was able to qualify in the national qualifiers, it was deemed to have surpassed last year's team. The fact that is was able to achieve a qualifying time at only 60% of maximum speed proves that the vehicle has potential to be one of the faster cars in the competition.

The main objectives to implement two-way telemetry and build a custom PCB were successful. The custom PCB provides less noise on the current feedback lines, allowing torque control to be implemented. Torque control was implemented successfully and the vehicle is able to achieve acceptable lap times using torque control, however further time was required in order to fully tune the torque control to make it comparable to speed control. The two-way telemetry allowed the vehicle to be debugged whilst running, allowing tuning to be vastly improved, both in time taken and accuracy of tuning.

The overall performance of the vehicle was improved due to numerous mechanical improvements. The improved camera mounting system gave more reliable camera feedback allowing for better line detection. The mechanical improvements to the servo allowed the vehicle to handle corners at much higher speeds, allowing for vastly reduced lap times.

Overall, the project was considered a success and would provide a much improved starting point for other teams to work on compared to the vehicle initially provided.

10 References

[1] – Freescale Cup Overview (2012, September) [Online]
https://community.freescale.com/docs/DOC-1011
[2] – NXP Cup Race Track Details (2016, January) [Online]
https://community.freescale.com/docs/DOC-1092
[3] – NXP Cup General Rules (2016, January) [Online]
https://community.freescale.com/docs/DOC-93225
[4] – RC Truck Stop (2011, December) [Online]
http://rctruckstop.com/2011/12/24/6-awesome-household-products-for-rc/
[5] – Tamiya Cera-Grease HG (2009, July) [Online]
http://www.tamiya.com/english/products/87099grease_hg/index.htm
[6] – FRDM KL25Z Homepage [Online]
http://www.nxp.com/products/software-and-tools/hardware-development-tools/freedom-development-boards/freedom-development-platform-for-kinetis-kl14-kl15-kl24-kl25-mcus:FRDM-KL25Z
[7] – TSL1401 Linear Array Sensor Datasheet
[8] – NXP Cup Line Scan Camera Use (2013, April) [Online]
https://community.freescale.com/docs/DOC-1030
[9] – Carbon Fiber vs Aluminum vs Steel vs Titanium (2013, December) [Online]
http://www.ilovebicycling.com/carbon-fiber-vs-aluminum-vs-steel-vs-titanium/
[10] – NXP Cup EMEA Rules (2016, January) [Online]
https://community.freescale.com/docs/DOC-105423
[11] – NXP Cup DC Motor Specifications (2012, November) [Online]
https://community.freescale.com/docs/DOC-93309
[12] – MCU 101: Pulse Width Modulation for Servos (2013, April) [Online]
https://community.freescale.com/docs/DOC-1027
[13] – FRDM-KL25Z Schematic
[14] – Freescale Cup Shield for the Freedom KL25Z (2011, October) [Online]
https://community.freescale.com/docs/DOC-93914
[15] – Ning-Cheng Lee, “Reflow Soldering Processes and Troubleshooting: SMT, BGA, CSP and Flip Chip Technologies” (2002)
[16] – Atmel, “AVR186: Best Practices for the PCB layout of Oscillators “(2008)
[17] - Texas instruments, “High-Speed Layout Guidelines” (2006)
[18] - Texas Instruments, “PCB Design Guidelines for Reduced EMI” (1999)
[19] – Texas Instruments, “TWL1200 PCB Design Guidelines” (2009)
[20] – Clyde F. Coombs, “Printed Circuits Handbook” (2007)
[21] – Raman Maini, “Study and Comparison of Various Image Edge Detection Techniques”, *International Journal of Image Processing. Vol.: 3. Issue: 1.* (2009)
[22] – Ruye Wang, “Canny Edge Detection” (2013)
[23] – Jarrod Snider, “Automatic Steering Methods for Autonomous Automobile Path Tracking” (2009)
[24] – Martin Lundgren, “Path Tracking for a Miniature Robot” (2003)
[25] – Austin Hughes, “DC Motor Drive Basics”, *EE Times* (2008)
[26] - NXP Cup 2016 - EMEA Qualifications – London (2016, April) [Online]
https://community.freescale.com/docs/DOC-330005

Appendix

Appendix A

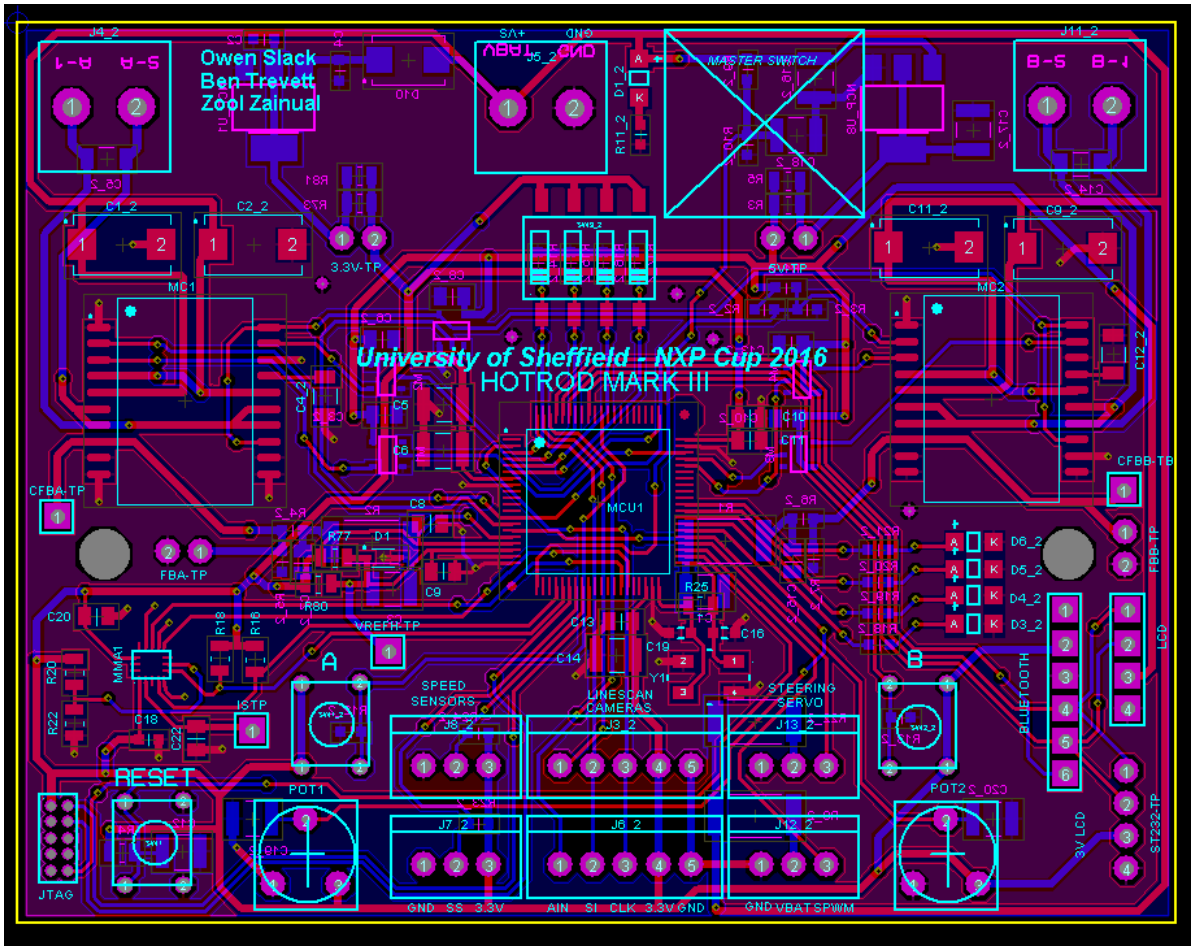


Figure 4.9.4.1 – Final 2 Layer Design

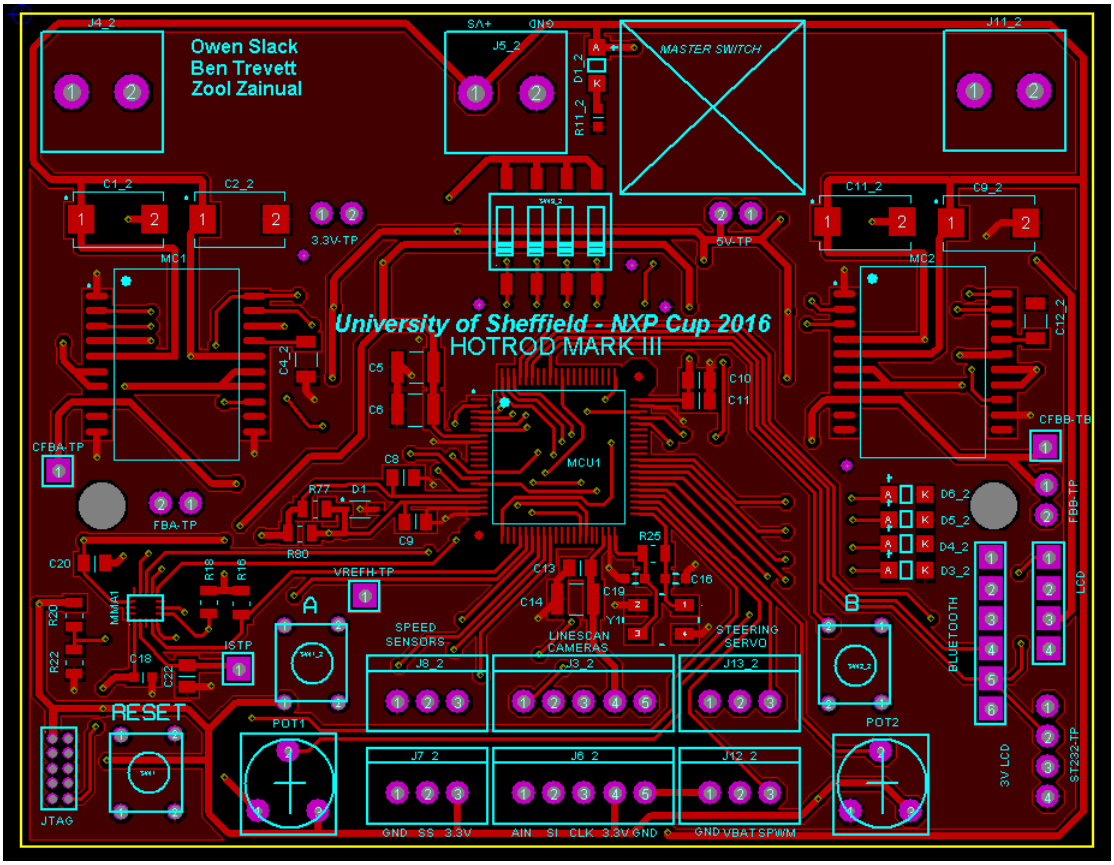


Figure 4.9.4.2 – Final 2-Layer Design, Top Only

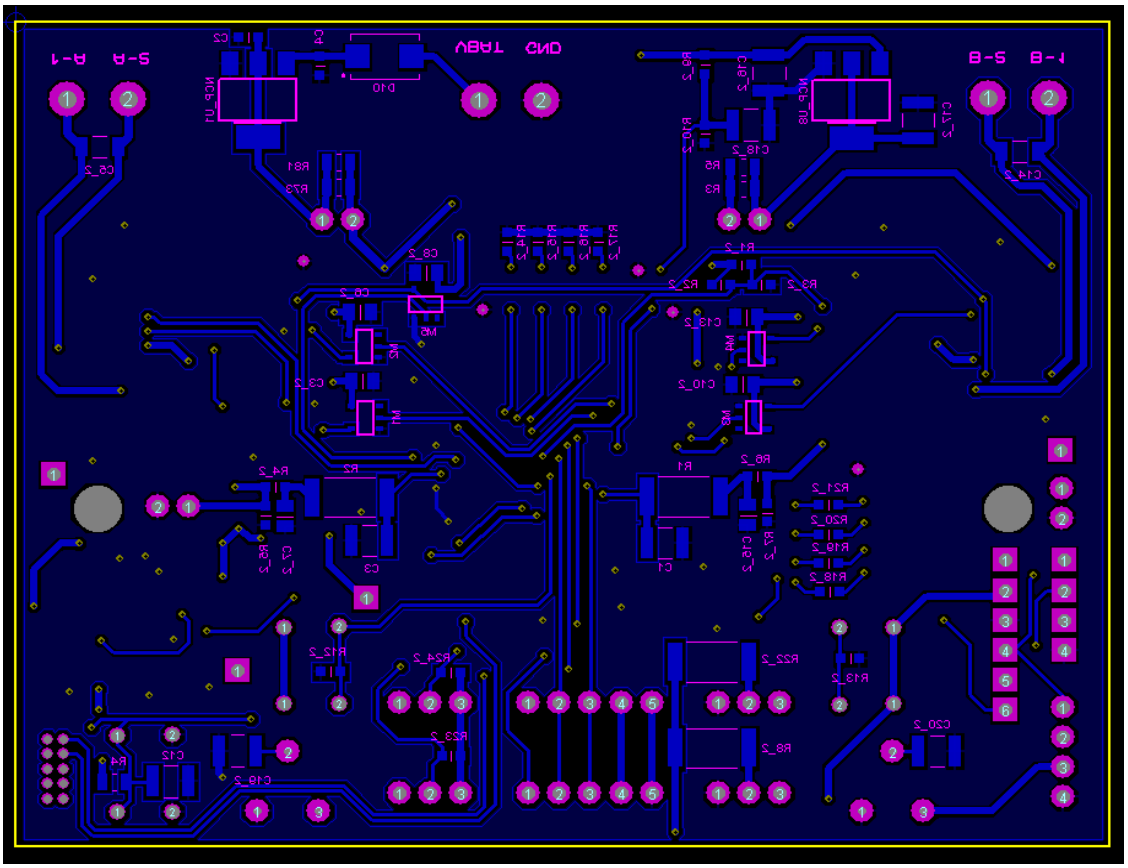


Figure 4.9.4.3 – Final 2-Layer Design, Bottom Only

Appendix B

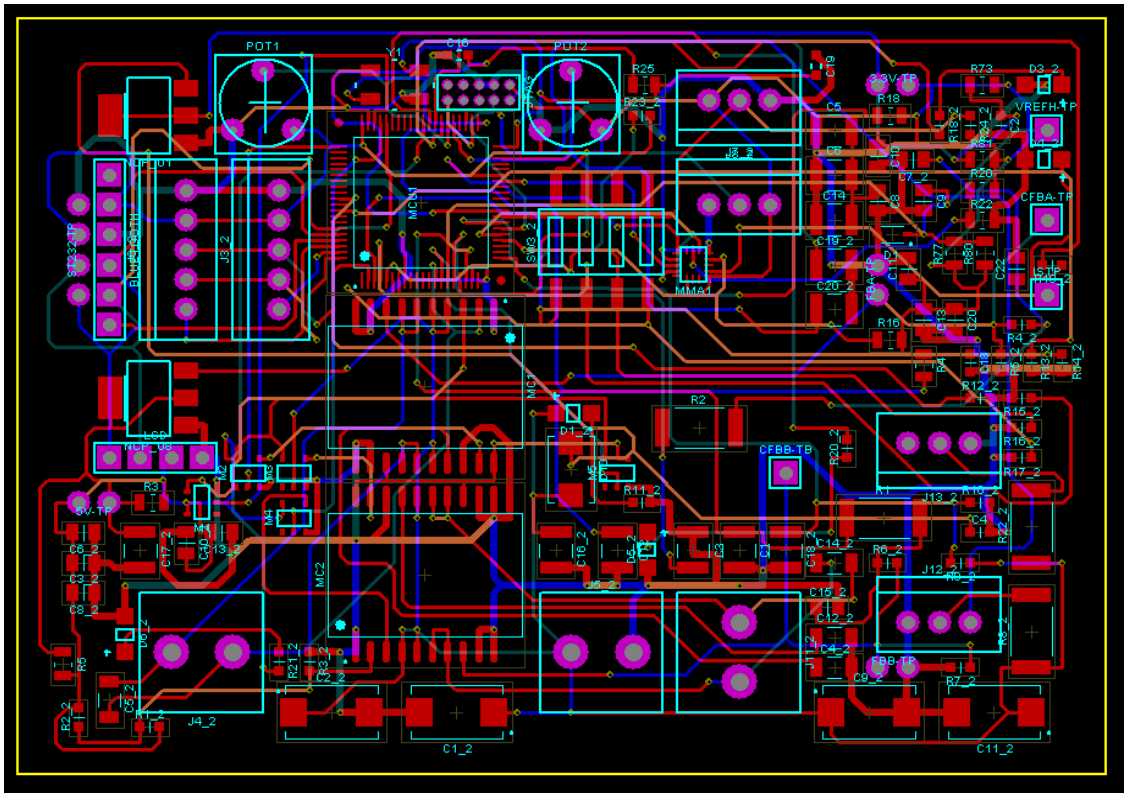


Figure 4.9.5.1.1 – Initial 4-Layer Design with Auto Place and Route

Appendix C

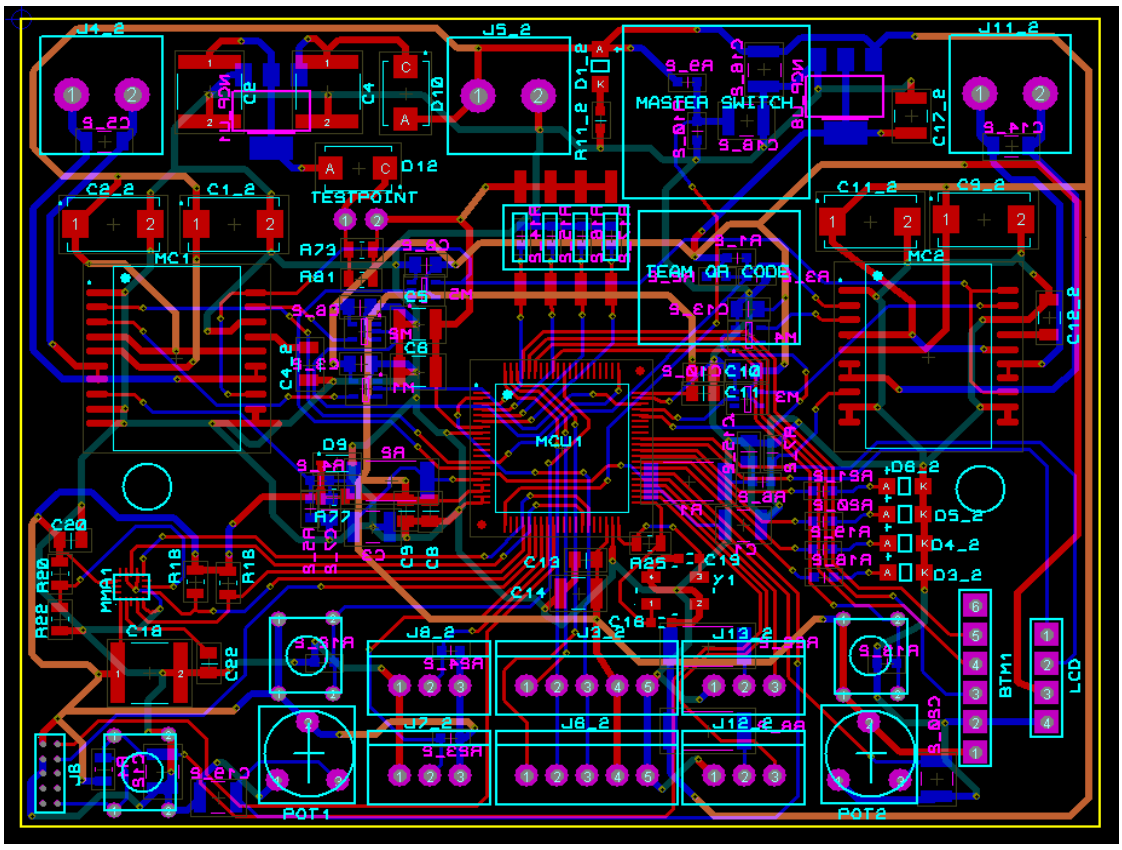


Figure 4.9.5.2.1 – Final 4-Layer Design

Appendix D

Quantity	References	Value	Part Number Farnell	Description	Max Voltage
5	M1, M2, M3, M4, M5	7VHC1G BUFFER	1654338		
4	R1, R2, R8_2, R22_2	1	2324189	2512 1W 5%	200V
14	R1_2, R2_2, R3_2, R4_2, R6_2, R10_2, R12_2,	10.0K	9238603	6030 1/10W 1%	50V
	R13_2, R14_2, R15_2, R16_2, R17_2, R23_2, R24_2				
3	R3, R73, R80	0	1099786	0805 1/8W 5%	150V
2	R5, R81	1K	2073606	0805 1/8W 1%	150V
3	R4, R20, R22	10K	9234136	0805 1/8W 5%	150V
6	R5_2, R7_2, R18_2, R19_2, R20_2, R21_2	220	9238409	0603 1/10W 1%	50V
1	R9_2	47K	9238689	0603 1/10W 1%	50V
1	R11_2	470	9238441	0603 1/10W 1%	50V
2	R16, R18	4.7K	9234098	0805 1/8W 5%	150V
1	R25	1M	2057738	0805 1/8W 5%	150V
7	C1, C3, C16_2, C17_2, C18_2, C19_2, C20_2	1.0u	9227962	1210 10%	50V
4	C1_2, C2_2, C9_2, C11_2	47u	1692397	CASE D TANT %20	25V
3	C2, C4, C18	10u	2469398	0603 20%	25V
7	C3_2, C6_2, C7_2, C8_2, C10_2, C13_2, C15_2	0.1u	2496944	0805 10%	50V
4	C4_2, C5_2, C12_2, C14_2	0.033u	3606170	1206 10%	50V
4	C5, C6, C12, C14	1.0u	2332901	1210 20%	100V

7	C8, C9, C20, C22, C11, C10, C13	0.1u	2332788	0805 20%	50V
2	C16, C19	22p	498543	0603 5%	50V
5	D1_2, D3_2, D4_2, D5_2, D6_2	DIALLIGHT RED LED	1519490	1.5x2mm 20mA	1.7V
1	D10	DIODE SCHOTTKY	1431044	SOD-123FL	20V
1	JTAG	SWD CONN	1865285	10, 1.27 mm	
2	MC1, MC2	H-Bridge	2308043	20HSOP	
1	MCU1	32bit, cortex-m0+ MCU	2212797	LQFP-80	
1	MMA1	Accelerometer	1842359	QFN	
1	NCP_U1	IC Linear voltage REG	1652366	SOT-223-3	3.3V
1	NCP_U8	LDO	9778209	SOT 223	5V
1	Y1	Crystal 8MHz	2467818	5x3.2mm	
1	SW3_2	DIP switch 4 way	1960899		

Table X – Bill of Materials

Appendix E

To set-up the vehicle when testing, the following steps need to be taken:

- Ensure that both motors, the servo, the Hall Effect sensors and both cameras are connected correctly, and that the battery is sufficiently charged. The Bluetooth module and LCD screen can be connected if needed.

- Place the vehicle in the centre of a piece of straight track, connect the battery and turn the power switch on

- If the two end LEDs are not turned on, the vehicle cannot see both outside lines and the camera must be calibrated. This can be done in three ways:
    - Trial and error, through adjusting the lens until both lights turn on
    - Connecting to Bluetooth and running the camera feed MATLAB script to give a live camera feed to aid adjustments
    - Connecting an oscilloscope to the SI (serial input) pin on the camera and using the waveform to aid adjustments

- Once the camera can see both lines, button '1' can pressed to start the vehicle and potentiometer 0 can be used to control the speed of the vehicle

- If the vehicle is not turning correctly, the top camera must be adjusted
    - If the vehicle is turning too early, the camera is looking too far ahead and must be tilted down slightly
    - If the vehicle is turning too late, the camera is not seeing the corners in time and the camera must be tilted up slightly

- If the vehicle is entering corners at the incorrect speed, the bottom camera must be adjusted
    - If the vehicle is moving too fast, the bottom camera must be tilted up slightly to detect the corners and slow down earlier
    - If the vehicle is slowing down excessively for the corners, the bottom camera must be tilted down slightly

- The DIP switches can be used to turn on and off the following features:
    - Switch 1: S-Mode
    - Switch 2: Cross Section Detection
    - Switch 3: Stop Line Detection