ELEC2209 – Autonomous Rover Design

Adilet Kaiyrgeldi, Alexander Bincalar, James Howe, Josh James, Wiktor Lawniczak

Team B

Abstract: As with all major design projects, a vast amount of research, trials and experimentation is required before a final product is produced. This report details the design process, development and final performance of the remote-controlled vehicle produced by Team B over the course of Semester 1, as well as link this information to the expansive list of technical specifications predetermined by Professor Klaus-Peter Zauner.

1. Aim and Constraints

This project aimed to produce an off-road autonomous vehicle with strict technical specifications and a budget of £200. The vehicle's precision and robustness would then be tested on an off-road obstacle course, where it would compete for the highest score against the two rival teams' rovers. To secure the highest score, the rover would have to navigate the obstacle course with no human assistance.

1.1 Design Constraints

The design of the rover was left relatively unrestricted. Almost all aspects of it were left to the discretion of the team. Regardless, a number of technical requirements were set out in order for each team's final product to be safe and manufactured to a professional standard. Violation of any of these requirements resulted in a score capped to 40% and so it was absolutely paramount that these rules were adhered to. On top of the strict requirements, there was a number of technical recommendations provided. These did not have to be followed strictly, but were strongly advised, and any deviation from them would have to be well justified, otherwise it would be considered poor design resulting in a score penalty.

Firstly, no glue or adhesive substances were allowed anywhere on the rover with the exception of gluing tyres to rims, self-adhesive heat pads, and heat shrink tubing containing hot glue. This constraint did not impact the project significantly, as there was no intension of gluing any parts of the rover together simply from a durability and disassembly viewpoint. Instead, all joints would be screwed together to allow for a sturdy design, which could be taken apart and put back together if necessary. The only impact this did have on the design was that heat shrink tubing would have to be used for insulating cables as opposed to simply wrapping them in electrical tape. This was an understandable constraint as it results in a more polished final product.

Secondly, all metalwork would have to be finished appropriately to make it safe for handling ie. cut edges and corners would have to be filed down to be smooth, and any drilled holes deburred; any risk of cutting yourself on the final product would be an automatic failure of the design constraints. Again, this constraint was not of huge significance to the project, seeing as the majority of the chassis was cut out of 7mm acrylic sheet, and the corners were rounded in the CAD software with edges deburred by a metal ruler, while the tracks 3D printed. The only parts containing sheet metal were the aluminium brackets connecting the main body to the sides. These were cut, bent and drilled, and any parts which would be exposed to human interaction were filed down to make them safe for handling. The design was made to contain as little sheet metal as possible partly due to this consideration.

Thirdly, all circuitry would have to be mounted properly, on permanently soldered circuit boards, rather than on bread boards. This requirement is unsurprising, as the rover must be durable and so all connections must be solid, none of them can become disconnected accidentally or the consumer may not be able to fix it again. Accidentally reconnecting it to the wrong terminal could cause great damage to the whole circuit, possibly killing the whole rover. As a result, all circuitry has been done on perf-boards, which have 3D printed standoffs on which they are screwed into the chassis. Custom-printed PCBs were not considered for this project due to the added complexity and limited time constraints associated with having to wait for them to be printed. Additionally, all connectors have been polarised so that only male-female connections can be made, also to avoid any misconnections that could damage the circuitry.

Next, the power supply would have to be regulated for safety. The chosen batteries can supply close to 10 amps and so it imperative that there was both a kill switch directly from the batteries to the rest of the circuit, as well as an adequate fuse. The chosen switch also contains an LED indicator of when the rover is powered. An additional two switches are included for controlling power to each motor individually. Once again, the switches are mounted directly in the top panel of the chassis for a clean, robust design with easy access. Battery chemistry also plays an

important role in the safety and performance of the rover, so while a LiPo cell would have provided improved performance, it also would have required extra considerations for the safety aspects. As a result, two NiMH cells were chosen instead. These have lower cell voltage - which is why two batteries in series had to be used - but are far safer.

Finally, all load-bearing axles require bearings to ensure smooth, frictionless operation. To accommodate this, the design had to have enough space for the chosen ball bearings which had outer diameter of 25mm, inner diameter of 10mm and a width of 8mm.

1.2 Competition Rules

The basic concept of the competition is a test of precision, with no weight on the speed of completion of the obstacle course. This meant that when the motors were being chosen, emphasis was put on the torque, as opposed to the speed. Having said this, there is a maximum amount of time in which the rover must complete each challenge, or it will receive a did-not-pass penalty. All teams start with a set amount of points, and each time the rover is unable to successfully complete a challenge independently, a set number of points is deducted from the total.

The brief stated that the obstacle course could be indoors or outside, and parts of it may be submerged up to 40mm. This meant it was necessary to at least partially waterproof the rover, to protect the electronics from the rain above by adding a roof, and to ensure there are no electronics at least 80mm from the ground, for adequate clearance of the water and any other obstacles. The motors are the lowest part of the rover, and while their contacts are not waterproofed, the cables are protected by a 3D printed shield to prevent them snagging on anything below it. The water aspect of the challenge also introduced a number of concerns regarding the materials used: avoiding corrosive metals anywhere that could have contact with water was also imperative for the longevity of the rover. The chosen rods for connecting the track links were made from fibreglass, which would not react to the water leading to good longevity. However, due to problems with supply, there was no choice but to include a small number of steel rods in the connections between the tank tracks. While this would impact the durability aspect somewhat, it was not too significant since the rover only had to complete one test, and this would not be enough to cause any immediate corrosion to the track links.

Originally, the course was to be navigated autonomously by passing through marked gates that could be ‘seen’ by the rover. These gates would be a minimum of 400mm apart, so it was necessary to make the rover no wider than this distance, ideally a fair bit narrower to make it easier to pass through them. Touching a gate marker resulted in a 5 point deduction, and being unable to pass it altogether in 30 points being subtracted. The autonomous aspect of the rover had to be removed due to not having a computer vision system ready for the final trial, something which was out of the team’s control. The alternative was to make the rover wirelessly controlled via mobile phone, and to remove the gates but instead have a more subjective judging system of the rover’s capabilities in a more open obstacle course.

Finally, each time the rover is touched or requires repositioning due to any reason, whether that is getting stuck or toppling over, 5 or 10 points are deducted respectively. To avoid this, it was decided that the rover shall use a tank track system to move around, with high clearance so that it wouldn’t bottom out. The tracks allow for 100% of the torque to be applied at all times unlike with individual wheels, reducing the chances of it getting stuck. A balance also had to be struck between the height of the rover for clearance, and its width and height - for maximum stability it could not be too tall, or it ran a risk of toppling over.

1.3 Available Resources

The main limited resources available were money, time, tools and the skill set brought by the team.

Firstly, only £200 were available for entire project, with a £70 cut being taken involuntarily for initial drivetrain testing - which proved very useful for the development stage of building the final rover - as well as the computer vision system being provided (despite this not being used in the end). This left the team with an effective budget of £130 for everything needed for the rover, including a minimum of two motors, two batteries, a controller for processing the computer vision system’s outputs, as well as all the other smaller components such as bearings, switches etc. A full bill of materials can be found in Table 1.3.1 of Appendix B.

In addition to this, access was granted to the workshop during the week’s work hours (10:00-17:00) which crucially provided access to 3D printers, laser cutters, drills and much more. Early on, a decision was made to utilise these tools to their full extent in order to save as much money as possible. A prime example of this is the use of the 3D printer to manufacture all the necessary track links instead of buying expensive prefabricated ones online; these were especially difficult to find due to the limited number of approved vendors the university was allowed to order from. Usual places such as Amazon or eBay were not allowed due the ethical concerns associated with buying from such sources. Even with all these measures to save money in place, the bill of materials came dangerously close to the allowed budget and some sacrifices had to be made to not go over budget. Priority was given to the batteries and motors, and this is where 49% of the budget was allocated to.

Time was also a very significant resource which would have to be allocated with prior deliberation. The original timeline allowed for 4 weeks of preparation before rover design would have to begin, and then the following 7

weeks would involve designing, building and testing the final product for the competition to take place at the end of week 11. The initial 4 weeks were crucial for getting the team acquainted with one other and gathering ideas of what the final product should be. They comprised of the initial power train design project which showed each individual's strengths and weaknesses. Due to a delay in the arrival of ordered components, the entire timeline was pushed back, and final testing would not have to take place until the final week of the semester.

2. Design

2.1 Chassis

The rover is comprised of 4 body panels: the main chassis, two side panels and the top panel/roof. All the panels were designed in CorelDRAW 2019 and cut from 7mm acrylic. A thickness of 7mm was selected for its balance between strength and weight, as the overall rover would be reasonably heavy and so the chassis would have to support the combined weight of the batteries and motors, as well as any strains exerted during the traversal of the obstacle course. It was decided that adding walls to the rover was not necessary and would only restrict air flow which may lead to overheating. The schematics for the chassis, side panels and top panel can be seen in Figures 2.1.1, 2.1.2 and 2.1.3 respectively.

Securing the main chassis to the side panel was accomplished via 6 aluminium L-brackets, 3 on each side, with 4 screws per bracket. This gave a total of 24 connections between the chassis and two side panels, and each one was fixed via a M3 16mm screw. The L brackets are the weakest point of the chassis' design so it was essential that they could withstand the stresses required to hold the rover together. Multiple iterations of these brackets were manufactured, but this process was not documented.

The top panel seen in Figure 2.1.3 is suspended above the main chassis via 4 75mm M10 bolts, allowing the height of the roof to be adjusted for a maximum clearance of around 55mm between the chassis and top panel. This space had to be adequately large to accommodate the height of all the components on the main chassis, as well as the additional components mounted to the underside of the top panel. These components included 4 perf boards, 3 of which housed the motor controller circuits, and the final one both the fuses. These fuses would then be placed into the cut-outs in the top panel in such a way that only the fuses were accessible from above, and not the fuse holder itself which would have exposed contacts and could potentially pose an electrocution threat to the consumer. This was done as a safety feature. The final components found in the top panel are the 3 switches, one main power switch going directly from the two batteries, and 2 secondary switches for each individual motor.

The side panels were angled in the same way the tracks would be in order for the rover to be able to scale obstacles without any hinderance from the chassis. They contained 5 10mm holes along the bottom, 3 of which are populated by track guides in the final design. There is a 40mm hole at the back end of the side panel, made to accommodate the large circular gearbox enclosure of the motors. Designing a system which allowed the tracks to be installed and tightened for a secure fit was very simple. Since the track guides are mounted via bolts, a long, 10mm wide cut was made angled into the front top corner (top left of Figure 2.1.2). The bolt could then be inserted and tightened along the entire length of this cut changing the effective length of the tracks and tightening or loosening them.

Another consideration for the durability of the design is that the mounting holes cut out for screws could not be too close to the edge of the panel. Therefore, a self-imposed constraint for chassis design was not to have any load bearing M3 holes cut within a 3mm border from the edge of each panel. This constraint was extended to 5mm for the M10 bolts securing the top panel of the chassis.

One of the design constraints stipulated that the batteries be easy to remove for charging purposes. This was accomplished by having fixed brackets which the batteries could slide into, with the end secured by a single screw, which after unscrewing would allow for easy removal.

2.2 Drive System

The design of the drive system comprises of three elements, the first being the undriven cog component. The undriven cogs are located at three equal intervals along the bottom of the rover. This was done to ensure even distribution of the rover's weight, and the dissipation of force experienced upon impact of any obstacles. An additional, adjustable cog is located at the top of the front so to allow the rover to scale over obstacles. Specific dimensions can be found in Figure 2.2.1, while their relative location on the rover in Figure 2.2.6. The component consists of five elements: a 3D printed long washer and undriven cog, a steel washer, a 40mm M10 steel screw and bolt. Components were assembled as according Figure 2.2.1, from which one can see that the undriven cogs will be under a large amount of stress when in use due to them being the main point of contact with the ground. For this reason, when printing the cog element of the component, a high infill was used. This made the component denser, thus increasing rigidity and strength to the desired specifications.

The second component was the driven cog found at the rear of the rover. This was fabricated using 3D printed PLA to create the 12-sided dodecagon needed for the teeth of the tracks to line up with the cut-outs placed at each corner of the dodecagon. Dimensions for the driven cog, as well as a 3mm wall around the cog which stopped the tracks from slipping off, can be seen in Figure 2.2.3. The

cogs were attached directly to the drive shaft of the motors which where mounted to the underside of the main chassis and allowed to protrude through the side panel. This component also required a high infill as it will be under considerable stress when moving over obstacles which would increase the torque being transferred through to the tracks. It also has to be able to transfer the entire torque of the very powerful motors into movement.

The third and final section was the tracks. These were also 3D printed and were linked together using M3 fiberglass rods. Their dimensions can be found in Figure 2.2.2. The tracks include a rounded tooth element which is designed to interact with the cut-outs in the driven cog to transfer power from the motors to the tracks without slipping. The fiberglass rods were chosen for their strength and lightweight characteristics, making it more desirable than such materials as steel.

2.3 Electronics

The power train was considered to be the most important part of the design; thus, a large proportion of the budget was spent on battery packs, motors, motor drivers and the other necessary components needed for the electrical side of the project. LiPo batteries were disregarded due to their high volatility which imposed greater design constraints on them, therefore NiMH batteries were chosen instead. In addition to NiMH batteries being safer, they are simpler for the consumer to use, as they don't require a balance charger, unlike LiPo battery packs.

Two 7.2V, 4200mAh NiMH battery packs were used in a series configuration due to the motors being rated at 15V. The one advantage LiPo technology gives over NiMH is a higher nominal voltage, which would have removed the need for two battery packs, as a single 4S LiPo battery pack could give the required voltage. This would have placed less strain on the budget, but the complexity of the additional design constraints would have negated this cost saving by demanding more components and time to solve the additional challenges brought by LiPo cells.

The chosen motor dictated the amount of NiMH cells that were required. The MFA 919D1001 motors are rated from 4.5V to 15V, with a theoretical no-load speed of 184 RPM at 15V [1]. The chosen battery packs would provide a nominal voltage of 14.4V which would allow the motors to operate near their rated maximum values. Low RPM motors were an intentional decision as torque was being prioritised over speed, due to the nature of the challenge. Torque is the most important factor when dealing with obstacles, so RPM had to be neglected as torque and speed are inversely proportional to each other. High RPM motors would have been detrimental to the challenge, not just because of low torque, but also because the vision system wouldn't be able to cope with high speeds.

The MFA 919D1001 motors that were selected for the project made use of a standard single ratio gear box. Geared motors consisting of planetary (epicyclic) gearboxes (such as the MFA 942D1001 540/1) were considered, but due to stringent budget constraints, the standard single ratio geared variant had to be chosen, as this saved £18.14. This sacrifice was unfortunate, but necessary as many other important components were required. In a scenario with a larger budget, planetary geared motors would have been the preferred choice, as these have very robust and reliable gearboxes which could safely handle large loads.

As the rover was using tracks instead of wheels for movement, only two motors were required, one to drive each track. This may have saved money when compared to purchasing four individual motors for a four-wheel drive ATV.

Due to time and budget constraints, the popular L298 dual full bridge driver was selected to drive the motors. The dual channels of the L298 can be paralleled to a single motor, allowing double the current to be utilised, which was up to 4A of continuous current. This amount of current was acceptable as it was almost 50% of the motor's rated stall current. Preventing the motor from reaching 50% of the stall current would be facilitated through current limiting, necessary to avoid overheating the motor or even destroying the gearbox, as the torque is sufficient to cause costly damage. This current limit would also protect the L298 Motor drivers from overheating. Furthermore, current limiting was a design recommendation which should be adhered to as it will prevent the consumer from breaking the product (or at least the electrical section) by applying an unbearable load. Heat sinks were attached to the motor drivers to help with heat dissipation. This should allow the motor drivers to operate near their specified maximum ratings. The circuit design for the L298 was relatively simple as a lot of useful information such as schematics were provided in the datasheet.

A Raspberry Pi 3B+ was used as the main controller, chosen by the programming lead. Every team member was given a Raspberry Pi 3B+ in the first year of university, allowing one of these to be used in the project without incurring extra cost.

2.4 Programming

The program had to be able to interpret an input from several pins of the computer vision system and then steer the rover in the appropriate direction dictated by the input. In order to be a polished product, it should also be able to control the motors smoothly, independently and with minimal audible noise. As a safety measure and to minimise wear on the motors, the program should also make use of current sensors to calculate the current

running through the motors; if the current is too high, the motor speed should be limited.

The programming of the Raspberry Pi 3B+ could be done in either Python, C, or C++. The "Wiring Pi" library to control the pins on the device was available for each of these languages, so it was a matter of preference. As the most experience was with C++, this language was chosen.

In order to make the code readable and easy to debug or change, several abstractions were made. Firstly, a class was made for PWM signals. Each instance could set the pre-scaler, timer resolution and duty cycle for a PWM signal on a PWM-equipped pin. Secondly, a motor class was made. The required inputs for the motor controller chip were a PWM signal, and two control signals that operate the MOSFETs on the H-bridge. Hence, the motor class contained a PWM instance and the two pins required for the control signals.

For smooth motor operation, it was decided that the duty cycle should change slowly between values. To achieve this, a method was implemented in the PWM class. This would alternately step the duty cycle and delay the program, until the appropriate value was reached. The delay in milliseconds would be found by experimentation during development.

The main way to set the speed of each motor in the main program loop would be by calling a motor method that is passed the duty cycle of the wanted PWM signal. To account for the motor's rotation in both directions, this value should be able to be negative, indicating the opposite direction to positive values (ie. reverse motion). Depending on the sign of the input, the control signals should be manipulated accordingly, to change the direction of rotation.

Setting the duty cycle to 0% is not appropriate to stop the motors, as the components must be discharged. Therefore, a method for the motor class is required to set the control signals to discharge through the H-bridge in the motor controller.

The consumer could move the tracks whilst the device is off, inducing a backwards e.m.f. in the motors. To prevent this affecting the circuit there should be another method for the motor class. This method should isolate the motor from the controller, using the corresponding control signals. Before the motor is isolated, it should first be discharged. To ensure that this is the case, the discharging method should be called first, from within the isolating method.

Finally, there should be a way to prevent excessive current flow through the motor, such as when it is stalling. An ADC will be used as the Raspberry Pi has no analogue sensors of its own. The voltage across a resistor of known resistance to ground can be used as a current sensor. The equivalent ADC value at 50% stall current can be hard coded as a limit. The main loop should continually check if this limit is exceeded. If so, the duty cycle should be reduced smoothly to 0, avoiding current spikes.

A full copy of all programs used by the Raspberry Pi can be found in Appendix C.

3. Development

3.1 Chassis

During initial testing, 3D printing the chassis was trialled out. However, this quickly highlighted the shortcomings of 3D printing large objects – due to the small 3D printers available, the chassis was just 130x75mm, and took over 4 hours to complete. Despite this extended printing time, the final product was still not perfect, as the base plate moved during the printing process, shifting the top layers and causing them to misalign. After multiple initial attempts it was decided that this method of production was not reliable enough and a switch to laser cutting was made, which was far more reliable and took approximately 4 minutes for a 290x195mm panel with no engraving.

A total of 5 versions of the main chassis, 2 versions of the side panels, and 2 versions of the top panel were designed and manufactured, each iteration improving on the features/shortcomings of the previous. The exact changelog of each component and their corresponding Figures can be seen in Table 3.1.1 of Appendix B.

The original design included only the main chassis and side panels. As the design evolved, it was clear that some sort of top panel or roof would have to be added, both to protect the electronics from potential rain which could short circuit them, and to add additional space on which to mount components. In the final designs seen in Figures 2.1.1 and 2.1.3, the batteries, motors and Raspberry Pi were mounted on the main chassis, while the switches, fuses and motor controller circuitry were attached to the underside of the top panel.

Intermediate designs included a plain roof with no components and a secondary shelf below the main chassis. This design was briefly considered due to the benefit of added rigidity of adding a panel below the main chassis, however it was discarded due the concerns that the clearance of the rover would be reduced too severely. Additionally, mounting the panel below the chassis would have meant the secondary level was very cramped. On the contrary, the final design includes an adjustable top panel, which can be up to 55mm above the main chassis.

The final design of the main chassis removed the Pi Zero, which was originally designated for the computer vision system used to make the rover autonomous. After finding out that this was cancelled and there would be no vision system, the Pi Zero could be removed, allowing the batteries to be moved towards the centre. This was beneficial, as the mounting holes for the top plate also had to be moved inwards due to them interfering with the side panels. The entire chassis was also extended lengthwise

by 5mm, to 290mm to also accommodate the new position of the top panel mounting holes.

3.2 Drive System/Tracks

After researching different mechanical systems for off-road use, two main approaches were identified. The first was the more conventional: using four wheels and an individual suspension system attached to each one in order to better adapt to the obstacles found in an off-road environment. The second was to use the idea of gears and tracks in order to imitate the function of a tank design on a much-reduced scale. It was decided that the latter would be used for this project for the following reasons: the first was that it would simplify the drive system as only two motors would be necessary to fulfil the same functionality as four-wheel drive; the second being the simplification of steering the rover. Controlling the rover in forward and reverse directions would be the focus, then the commands used to initiate these movements could be modified to control the rover to move left and right by individually differing the speed of rotation of each track. The tracks can even be made to move in opposing directions, allowing the rover to rotate 360° on the spot. Finally, research showed that track design rovers simply performed better over obstacles if designed correctly which was a key part of the design brief.

The initial design used the link system of a bike chain. However, this was found to be too narrow, and would require two track chains linked together which would overcomplicate the design. Instead, the tracks were made wider so only one loop would be needed on each side. This also increased the friction that the track would have with the ground. Interlinking the tracks with the driven cog was facilitated by adding two teeth to each link which would interlock with the next track link via fibreglass rod. Initially, steel rod was used, but thanks to research into the two material's properties, as well as the limited supply of 3mm steel rod, fibreglass was favoured. The two outer holes were made to the same dimensions as the pins and the inner was slightly bigger so it could rotate. Figure 2.2.2 shows how the tracks were linked together.

The next challenge was to convey the power from the motor to the tracks. This meant designing a driven cog which would interlink with the tracks and the motor. The motor axial already had a flat surface designed to efficiently transfer torque and reduce the chance of slipping. However, after discovering that despite this the axial was still slipping a bit, the flat surface was filed down to be bigger in order to improve the chances of it not slipping. To create the driven cog a dodecagon was made, the sides of which had the same distance as that of the distance between the teeth of the track links. The first version of the driven cog had some slip, so each corner of the dodecagon's teeth was cut to improve the fit for the final version. These smoother edges allowed the teeth to transition better when the rover was moving for smoother traversal of terrain.

Finally, the last part of the development of the tracks was to design undriven cogs which would hold the tracks tight and allow for friction with the ground to initiate movement. It also had to allow the tracks to rotate fully without interfaces whilst preventing the tracks coming off. This was the simplest part of the design, yet it required the most components in order to position the cogs correctly. Initially, five cogs were going to be placed on the bottom and another at the upper part of the front, though with the size of each cog this was reduced to 3 on the bottom simply due to the size constraints. After testing it was found that mounting the tracks too tightly negatively affected performance, as the friction on the rover would be higher, and more current would be drawn. This was addressed by the adjustable cog at the front of the rover, which allowed for the tension in the tracks to be adjusted to the perfect level. A balance between low current draw with loose tracks, and slightly higher current draw with tighter, more secure tracks had to be found. A full schematic of the undriven cog system is shown on Figure 2.2.1

A full schematic of the final project can be seen in Figure 2.2.6, showing a full render of the rover both in 2D and 3D. This gives the consumer a good view of the construction of the rover without any tracks in place.

3.3 Electronics

The main design consisted of the battery, motor controllers and the motors, but fuses and switches were also implemented into the circuit design to protect the product and make it easier for the consumer to use. These were hard design constraints that had to be adhered to, but they were also very important for making a sound product.

There were three switches in total, one for the main power, and the other two were for each individual motor, allowing the user to turn one or both motors off when necessary. These switches had built in LEDs that could operate at around 2-3V. The LED had separate contacts to the switch contacts, so the operation of the LEDs were completely independent to the position of the switch, but the LEDs needed to turn on only when the power was turned on, so the positive lead of an LED was connected to the 14.4V lead after coming out of the switch. 680Ω resistors were placed after the negative pin of the LED contacts in a series configuration to adjust the voltage drop from 14.4V to only 2V which allowed the LEDs to be safely used. This configuration allowed the LEDs to be directly connected to the power, instead of going through the buck converter or a signal from the Raspberry Pi, making the hardware simpler and more robust as less problems can occur when using such a simple arrangement. This configuration allowed the switches to light up when turned on, which made it easy for the consumer to see what circuitry was active.

Initially, current sensors (based on the Hall effect) were bought to measure current without creating a significant, wasteful voltage drop, but when tested, these didn't give accurate readings as the rating on them was too high (up to 31A), resulting in the analogue output of the hall effect sensors giving an inaccurate result for 4A. To solve this, the 0.18Ω 'sense' resistors that were part of the hardware current limit were used by the ADC to calculate the current going through the motor. The PWM duty cycle could then be decreased or stopped to prevent damage to the motor, allowing both hardware and software current limiting to be utilised.

Initially, the Raspberry Pi 3B+ appeared to be a good choice due to its relatively high processing power compared to a simple microcontroller, but it was soon discovered that the hardware pins were limiting, lacking ADC pins. To solve this problem, a MCP3008 10-bit ADC pin chip was selected as an external ADC interface which could communicate to the Raspberry Pi over SPI. A great advantage of using the MCP3008 chip is that the external circuit is incredibly simple (basically non-existent) so no passive components are required between the MCP3008 and the Raspberry Pi, which omits the problem of not having built-in ADC. This allowed the use of software current limiting through the previously mentioned popular method of 'current chopping'. Hardware current limiting was also introduced by making use of IRLB8748 MOSFETs and NPN transistors, which can be seen in Figure 3.3.1.

The L298 motor drivers posed no problems when implemented due to the detailed datasheet which provided useful schematics which the circuit design in Figure 3.3.2 was adapted from [5]. Buck converters were used to step 14.4V down to 5V for the Raspberry Pi and the L298 logic power supply. The complete circuit design can be seen in Figure 3.3.3, where the motor controllers refer to Figure 3.3.2.

3.4 Programming

The programming process was generally very smooth. The programming was done by connecting the Raspberry Pi 3B+ to a monitor via HDMI and to a keyboard and mouse via USB. The appropriate mains power plug was used.

The Geany Programmer's Editor IDE provided a useful environment to repeatedly build the code in, with customisable shortcuts to build the code. This made linking libraries very simple.

All program aspects were coded as set out in the design, except for the addition of multi-threading. As the smooth duty control method for the PWM class was blocking, only one motor could change its speed at any time. By creating a thread object for each motor, both motors could change speed simultaneously.

To identify when the program was running, and for debugging purposes, a simple thread was created to blink an LED attached to pin 17. The selected output pins for each motor instance were chosen to be adjacent to each other. This allowed for easier circuit debugging due to a better circuit layout. The layout can be seen in Figure 3.4.1.

After some testing, it was decided that a 10kHz PWM signal would be optimal for driving the motors. The WiringPi [4] library provided functions to set the pre-scaler and resolution. To find a corresponding pre-scaler and timer resolution for this frequency, the resolution was first set to 100. This would give duty cycle accuracy to the nearest percent. By experimenting with pre-scaler values and measuring the PWM output with an oscilloscope, a value of 18 was found to be appropriate. There were no worries about not being able to reach 10kHz at this resolution, as the clock speed of the Raspberry Pi 3B+ is 1.4GHz, which is far in excess of our requirements.

It was found that stopping the motor at speed by immediately putting the motor driver into brake mode produced current spikes and unpredictable behaviour. The current spikes were handled by the motor driver well, so circuit protection was not an issue, but the unpredictable behaviour needed fixing; the motor would occasionally turn in the opposite direction before fully stopping. The extra strain on the system was avoided by implementing a smoother braking method. This method set the PWM duty cycle to zero before using the control signals to brake and discharge the motor and capacitor.

A final extra thread was added to continually check the current passing through the motors. If the reading exceeded a value of 220, approximately equivalent to 3.5A, the smooth braking method would be used on the respective motor. The value of 220 is correct for 10-bit ADCs with a reference voltage of 3.3V. Experimentation was required to find the minimum acceptable delay between readings, to allow the ADC to settle. A value of 100ms was agreed upon. Faster sampling rates could be achieved, however that was deemed unnecessary, as it was only desirable for the rover to respond to sustained high current levels. 100ms allows the thread to be considerably less computationally expensive.

As the computer vision system was not yet available, a way of controlling the Raspberry Pi through an SSH server was devised. This was purely for testing purposes and would not make its way into the final product. The Raspberry Pi would run its own SSH server and would connect to a phone's mobile hotspot. An application [2] on the mobile phone would allow access to the SSH server and therefore access to a Raspberry Pi terminal. From this terminal, the appropriate file could be executed to control the rover. To specify which actions the rover should undertake, a simple command line text interface was coded. Generic actions such as moving forwards or backwards and turning were coded, as well as allowing

specific control of the duty cycle and direction of each motor.

Finally, upon testing with the motors, the smooth PWM duty cycle changing methods were found to be rather slow. To fix this, the delay between setting duty cycles was changed from 50ms to 20ms.

A bug was identified, where the current limiting thread and a motor-accelerating thread would 'fight' each other. If the current limit was exceeded, and a PWM signals duty cycle was being gradually increased by the motor-accelerating thread (SmoothSetSpeed), the effects of each thread would balance out, and the duty cycle would remain at 20% as opposed to changing to the desired level. As neither thread achieved the final duty cycle they wished to set, neither thread would terminate or join the main thread. Due to time constraints, this bug could not be fixed. Instead, an emergency stop command "emstop" was quickly implemented into the terminal interface. This set the duty cycle of both PWM signals to 0%. This allows both the current limiting thread to exit, as well as the other thread to achieve its duty cycle and exit too.

4. Evaluation

The design process went relatively successfully but not entirely to plan. The main issue was the timeframe – on multiple occasions the deadline had to be extended due to progress being slower than anticipated. This was further exasperated by external factors such as the delay in arrival of ordered parts. This, in addition to a very ambitious and time intensive design meant that the date of the final trial had to be delayed until the very last week of the semester.

Regular, weekly meetings allowed the team to make executive decisions on the design of the rover as a unit, and therefore to divide the work between each team member based on their abilities. Before any parts could be ordered and construction could begin, extensive research was made by each member into their chosen area of expertise. This meant James worked on the drivetrain, Wiktor on the power and chassis, and Josh on the programming. Alex was in charge of electronics/circuitry design, with Adilet on implementation and Alex also acting as lead integrator for the project. This was the most efficient way to split up the workload, based on previous experience: James had extensive knowledge of SolidWorks which is exclusively what was used for designing the drivetrain components. Josh can program in C, which was used for the initial IlMatto controller, as well as C++ which is used for the Raspberry Pi controller found in the final design. Adilet had done a large amount of soldering for personal projects making it a natural choice.

As a result of this well-matched team, overall progress on the project was good. All design decisions were done to streamline the process, especially since the team was aware of the limited time frame.

4.1 Evaluation of Chassis

The chassis design fulfilled all necessary requirements and so can be considered an overall success. It had adequate space to house all the components of the rover, without being so tall that it was top-heavy (making it unstable during steep inclines) nor so wide that it was too cumbersome and heavy. At 4.8kg, the entire rover was the heaviest in competition, but it also had the most powerful motors and batteries, so the weight was matched to the power.

The clearance from the ground was also adequate for the available obstacle course, as the rover did not bottom-out or get stuck on anything due to being too low. The printed out cable guard for the motors provided crucial protection for the cables. Despite this, a semi-waterproof layer below the main chassis would have been beneficial if the obstacle course had included any level of submersion – this was not included, although the ground was very damp during the outside test and there was evidence of this on the chassis where water was visible on the acrylic side panels.

This was the same regarding the top panel: while it did not rain during the outside trials, some drizzle could be felt, and the final design included unprotected fuses and switches on the top panel. These were intentionally left exposed for easy accessibility, but in a future version of the rover, some kind of waterproof guard could be added overtop of them in order for the rover to be suitable for all-weather.

The chassis was sturdy enough with the six aluminium L-brackets holding the main panel to the side panels, and no flex could observed even when colliding with obstacles at full speed. Having said this, an improved design would have had side panels that extended all the way to the top panel, with the main chassis resting in grooves cut partway up the side panel. This would distribute the mass evenly along the entire length of the side panels, making the rover even more sturdy, and the L-brackets would be used just to stop the main chassis from slipping out. It would also allow the top panel to be slightly higher as the current clearance was limited by the 75mm long M10 bolts, which were the longest available. The bolt design was useful as the top panel could removed and replaced very easily, and was far more sturdy than anything that could be 3D printed out of PLA.

4.2 Evaluation of Drivetrain

The rover was tested on several obstacles, both indoors and outdoors. Results showed that some aspects of the design worked without hinderance, while others showed room for improvement.

First were the tracks which performed well, converting the torque from the driven cog into motion. Having said this, two main issues did arise which could be amended in later versions of the project. While the rover performed admirably on rough, textured surfaces, it struggled on

smoother ones such as polished indoor floors. This meant the rover thrived outside, but did not do well on the artificial obstacle course created indoors. To improve its performance both inside and outdoors, a material with a high friction coefficient such as rubber could be added to the outside of the tracks, allowing for better performance when climbing slippery surfaces. The second improvement would be to decrease the angle of incline between the front two cogs. The current configuration was too steep with too little grip and as a result didn't cope well with obstacles that were tall as it tried to climb an angle which was too aggressive.

An issue was also discovered with how the tracks resonated as they rotated around the cogs. This issue was most present between the driven cog and the top (front) cog as they had the largest distance between them. This could be solved by providing tension to the track by added another undriven cog at the top. Alternatively, another method of reducing this resonance would be to add teeth cuts into all the undriven cog, but pursuing this method would be far more labour and material intensive, with all 72 track segments requiring reprinting.

Finally, the part of the mechanical design which showed the most wear was the driven cog. Although the design was relatively successful at transferring power to the tracks, it became clear at the end of the testing that the 3D printed PLA material was at its structural limits. The torque from the motor shaft wore away the flat side of the semi-circular connection which caused the motor shaft to rotate freely without turning the motor. This element could be fixed by having the sides of the cog made from a stronger material such as aluminium or harder plastics. This would solve the issue of the cog wearing away under strong torque without changing the geometry of the teeth which were well designed for transferring power efficiently.

In summary the tank design performed well, although further implantation of the methods mentioned above would have prevented some of the issues seen in the trial and allowed the rover to perform better overall.

4.3 Evaluation of Electronics

The electronics of the circuit performed well, but not perfectly. The current limiting hardware and software prevented any damage to the L298 H-Bridge driver and the motors. The rover could turn and travel forwards, but not in a straight line because there was no hardware such as gyroscopes or RPM counters that could be used in a feedback loop to keep the rover travelling in a perfectly straight line. Counting RPM using hall effect sensors with magnets was tested, but there was not enough time to implement this into the mechanical and software sections of the rover project. If this design project was to be conducted again, then a gyroscope would be used to ensure that the rover can move in a straight line.

A major problem occurred that restricted the movement of the rover to only one direction as one of the motors wouldn't reverse directions. This problem was investigated, and it was discovered that the Raspberry Pi's logic signals were incorrect, as it was outputting 2.4V instead of the expected 0V or 3.3V for different logic signals on some of the pins. This confused one of the L298 motor drivers as these drivers depend on two logic signals to set the direction of the motor, limiting one of the motors to only one direction. As this problem only appeared an hour before the trial, there wasn't enough time to fix the problem, but it is assumed that the GPIO pins may be damaged, or some bad solder connections were made which impeded the circuit. This problem wasn't severe, as the motor could still travel in one direction, which was enough for the challenge. It would have been nice to have solved this problem before the challenge, which would have required the final circuit to be soldered earlier on to allow sufficient time for testing. Next time when doing a project, the final product should be completed a few days prior to the official challenge to allow enough time for debugging, which would have had this problem solved before the deadline.

Looking back on the motor drivers, a driver with a higher current rating with built in 'current chopping' (a form of current limiting) would have been preferred, as it would have been better to have a speed controller with a higher current rating than the motor's stall current. The 'current chopping' feature would have transformed the issue of current limiting into a simple task that would no longer require convoluted external hardware and/or software to accomplish the same task. An example of a motor driver like this is Texas Instrument's DRV8873 which is rated at 10A and features built in current regulation. The problem with drivers like the DRV8873 is the fact that they can only be found as SMD packages, requiring a PCB to be designed and manufactured for them. This would have been a much better option, as soldering to a manufactured PCB would have been much quicker and efficient than soldering to perf boards. However, this would be an unattainable goal due to the time limitation, as the PCBs would need to be designed, tested and finalised before the bill of materials deadline.

Undertaking this project again, a bespoke PCB would be designed and manufactured for high quality SMD drivers such as the DRV8873. This would make soldering much easier and quicker than creating a whole circuit on a perf board. The only problem with this is that the circuit for the PCB would have to be designed and finalised very quickly, as changes to the PCB (once manufactured) would be near impossible, whilst through-hole components like the L298 allow you to test and build circuits on perf boards that can be changed at any point in time.

Overall, the electronics was a success as this component performed very well during the challenge. The electronics wasn't perfect due to the logic level problem that occurred with some of the GPIO pins, but this should be relatively easy to solve, and it did not affect the performance of the rover during the trial. A gyroscope would have allowed the rover to perform better by keeping it moving in a straight line.

4.4 Programming Review

The program worked successfully, given the time constraints. Given additional time, several aspects could be improved for a more polished final product.

Firstly, a thread handling object could be made to allow for the proper termination of unending threads. Each thread handler could include a timer, as well as an amount of time that the thread should be completed by. If the thread is not completed in this time, it should be automatically terminated, and print an error message.

Secondly, the file structure was very inefficient. Each header file should have been separated into header and source files. This would have dramatically reduced compile times, and therefore the time spent debugging. It would also improve the ability to understand the included functions in each source file, as the included functions, classes and methods would all be listed in the header. This would have no effect on the performance of the final rover, so is not entirely necessary, but is a healthier practice.

Most notably, the user interface could be improved. Even though the SSH server was a temporary solution, it was very clunky, and would be very difficult to use without prior knowledge of Linux terminals. The most convenient user interface would be a mobile application, that communicates with the rover via Bluetooth or Wi-Fi. A joystick could control the direction and turning, whilst a scroll bar could control the speed. This would require substantially more time to program given current experience but would be achievable.

Finally, various parameters of the program should have been defined using pre-processor definitions, in order to make them easily changeable during debugging. To improve convenience, a header file of these various settings could be created. This would include constants such as the duty cycle step during smooth duty changes, and the delay between these steps. The corresponding pins for each output could also be placed in this file.

5. Conclusion

Overall, the rover can be considered a success for the available timeframe and resources. Over the course of one semester, the team has produced an all-terrain rover which is capable of climbing steep inclines in outdoor conditions, with clearance for a respectable wading depth of at least 40mm, all under a budget of £200.

While the autonomous aspect of this project could not be achieved in the timeframe, this was out of the control of the team, and the design is made in such a way that the remote-control can be adapted to accept inputs from a computer vision system should it be provided in the future.

The project was equally divided between the team, and each member was allowed to work on the part of the design with which they felt most confident. As a result, the final product was a vehicle produced to a high standard, which integrated well with it's neighbouring components. Given more time and resources, weak-spots in the design could be addressed, but the rover succeeded at the outdoor obstacle course, with some performance left to be desired indoors.

Most importantly, the manufacturing was all done to stay within the design constraints, and was completed before the final deadline.

References

[1] "919D Series 35mm Single Ratio Metal Gearbox", MFA/COMO DRILLS [Online]: https://www.mfacomodrills.com/pdfs/919D%20series.pdf [Accessed: 14-Jan-2020]

[2] Apple, "PiHelper App Store Preview," Qian Sha, 2019. [Online]: https://www.apps.apple.com/gb/app/pihelper/ [Accessed: 21-Jan-2020]

[3] @Gadgetoid and @RogueHAL13 (Twitter), "Raspberry Pi Pinout," pinout.xyz, [Online]: https://pinout.xyz [Accessed: 22-Jan-2020]

[4] @drogon, projects@drogon.net, "Wiring Pi," 2020, [Online]: https://wiringpi.com [Accessed: 22-Jan-2020]

[5] "L298 Dual Full-Bridge Driver". STMicroelectronics [Online]: https://www.st.com/resource/en/datasheet/l298.pdf [Accessed: 24-Jan-2020]

Appendix A

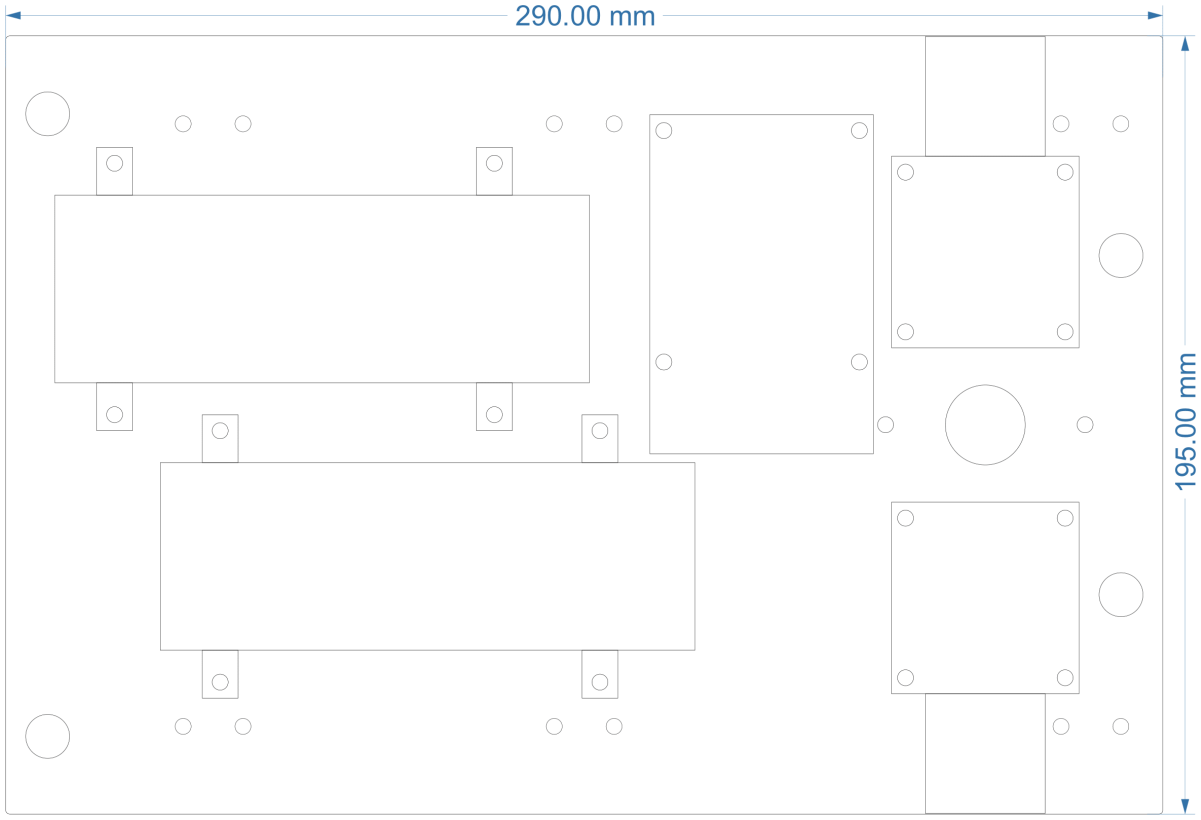

Figure 2.1.1: CorelDRAW 2019 design of main chassis - Version 5 (final). Circles are of hairline thickness, meaning they are cut out by laser cutter, rectangles mark outlines of components for reference, but are not cut.

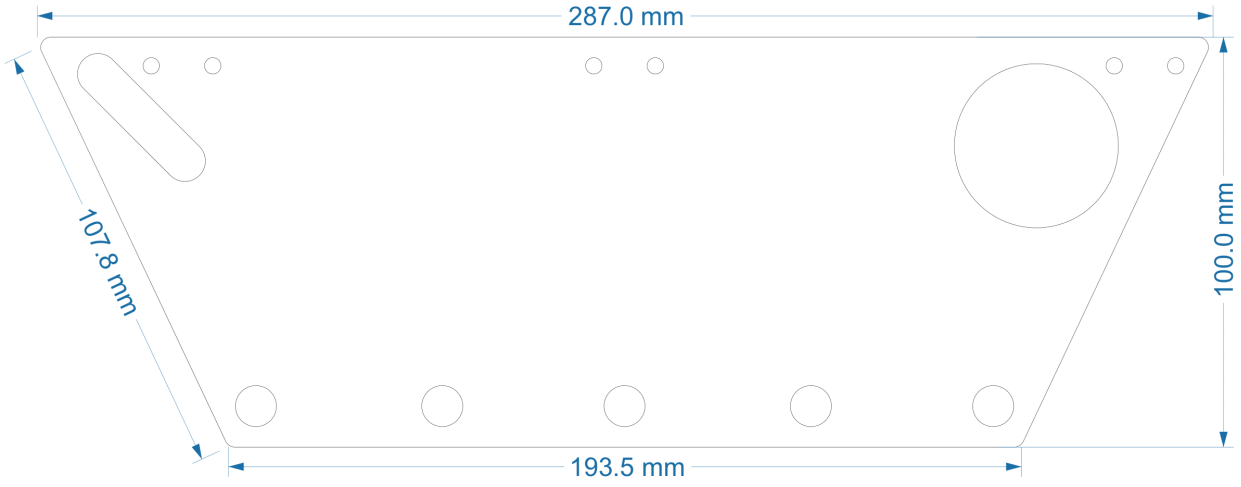

Figure 2.1.2: CorelDRAW 2019 design of side panel - Version 2 (final).

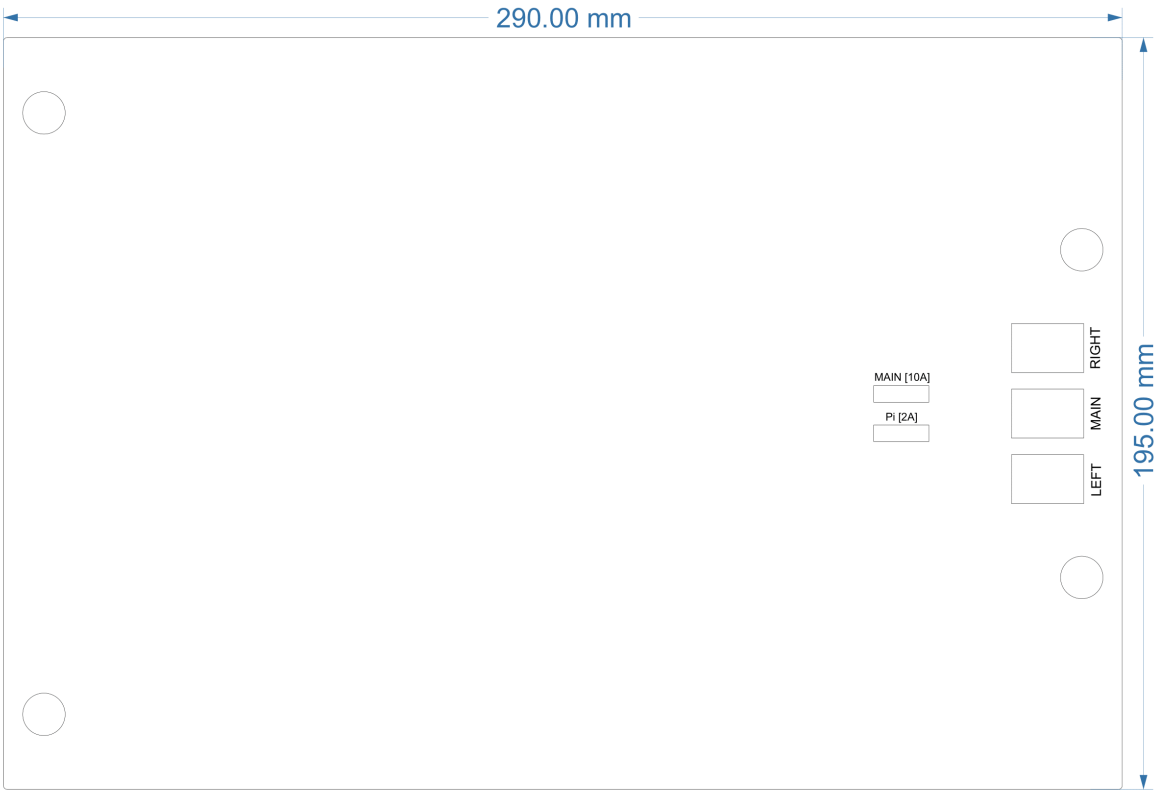

Figure 2.1.3: CorelDRAW 2019 design of top panel - Version 2 (final). Both circles and rectangles are of hairline thickness and therefore are cut out, while text is engraved onto panel by laser.

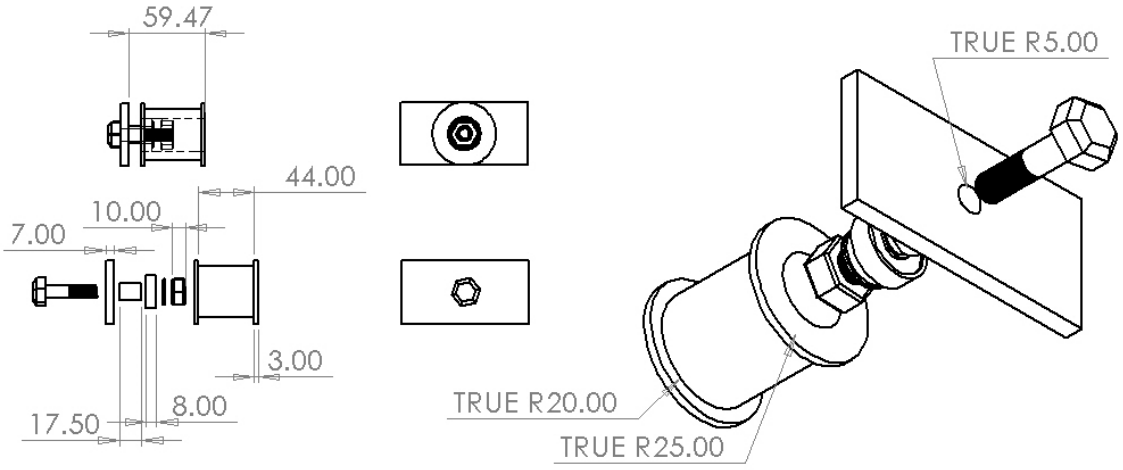

Figure 2.2.1: Model of undriven system. All dimensions in mm.
Top Left: assembled 2D. Bottom Left: disassembled 2D. Right: disassembled 3D.

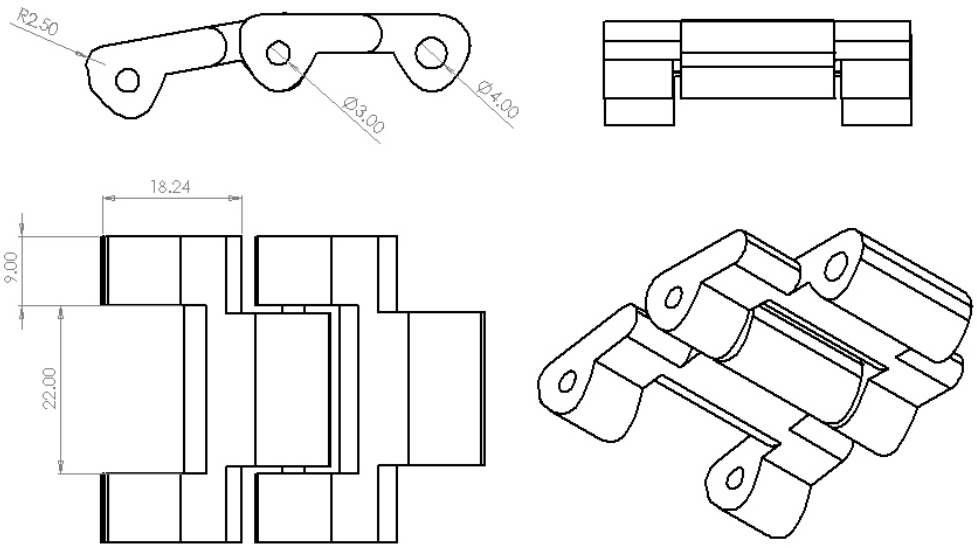

Figure 2.2.2: Model of track and pin links. All dimensions in mm.
Top Left: side view of two track links. Top right: front view. Bottom Left: top-down view of two track links. Right: 3D view.

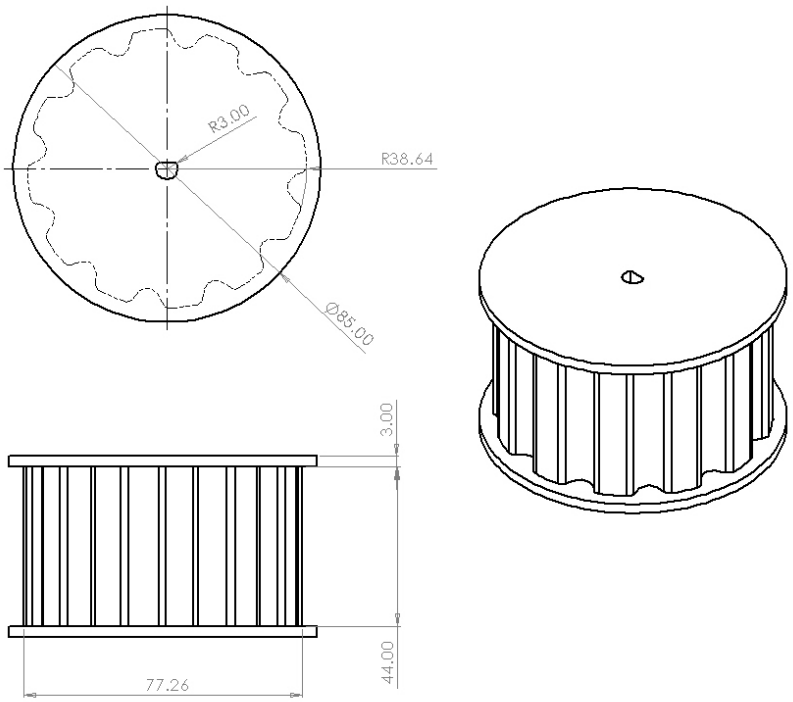

Figure 2.2.3: Model of driven cog. All dimensions in mm.
Top Left: top-down view. Bottom Left: side view. Right: 3D view.

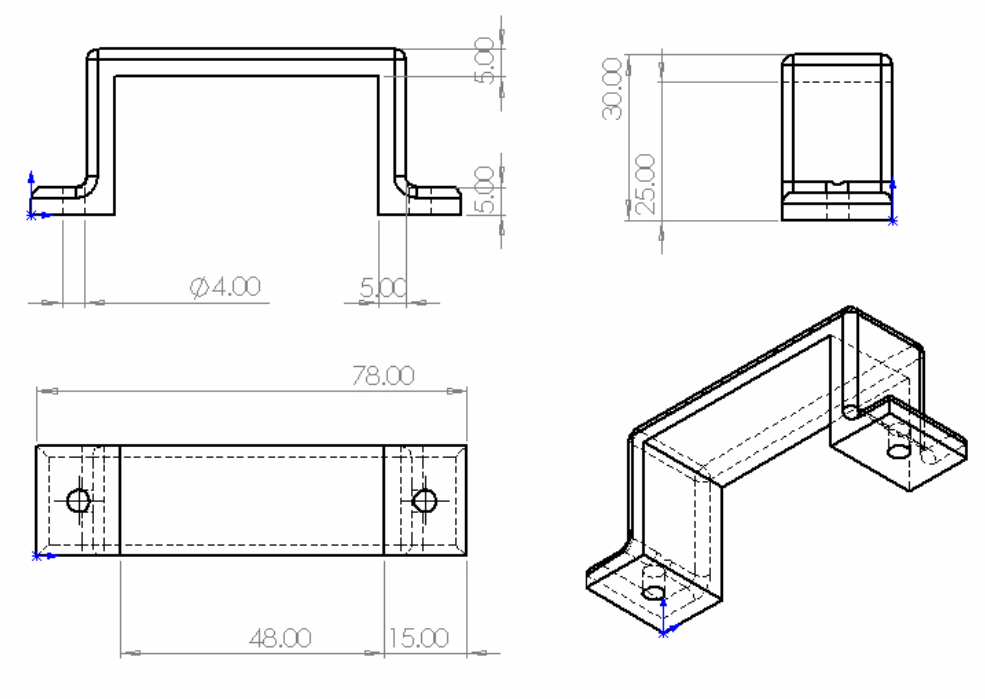

Figure 2.2.4: Model of battery mounts. All dimensions in mm.
Top Left: front view. Top right: side view. Bottom Left: top-down view. Right: 3D view.

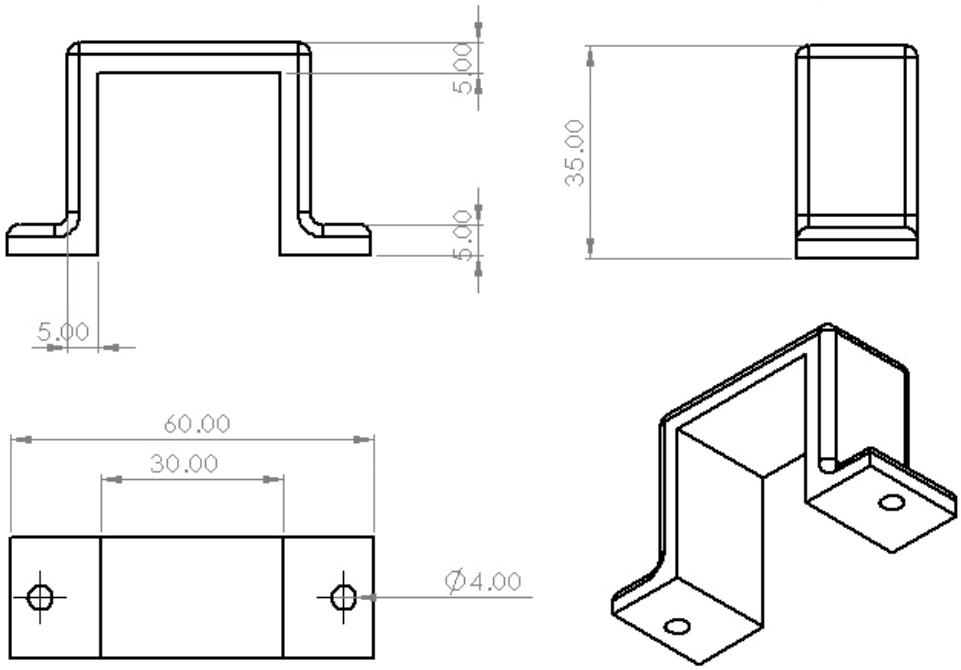

Figure 2.2.5: Model of motor cable guard. All dimensions in mm.
Top Left: front view. Top right: side view. Bottom Left: top-down view. Right: 3D view.

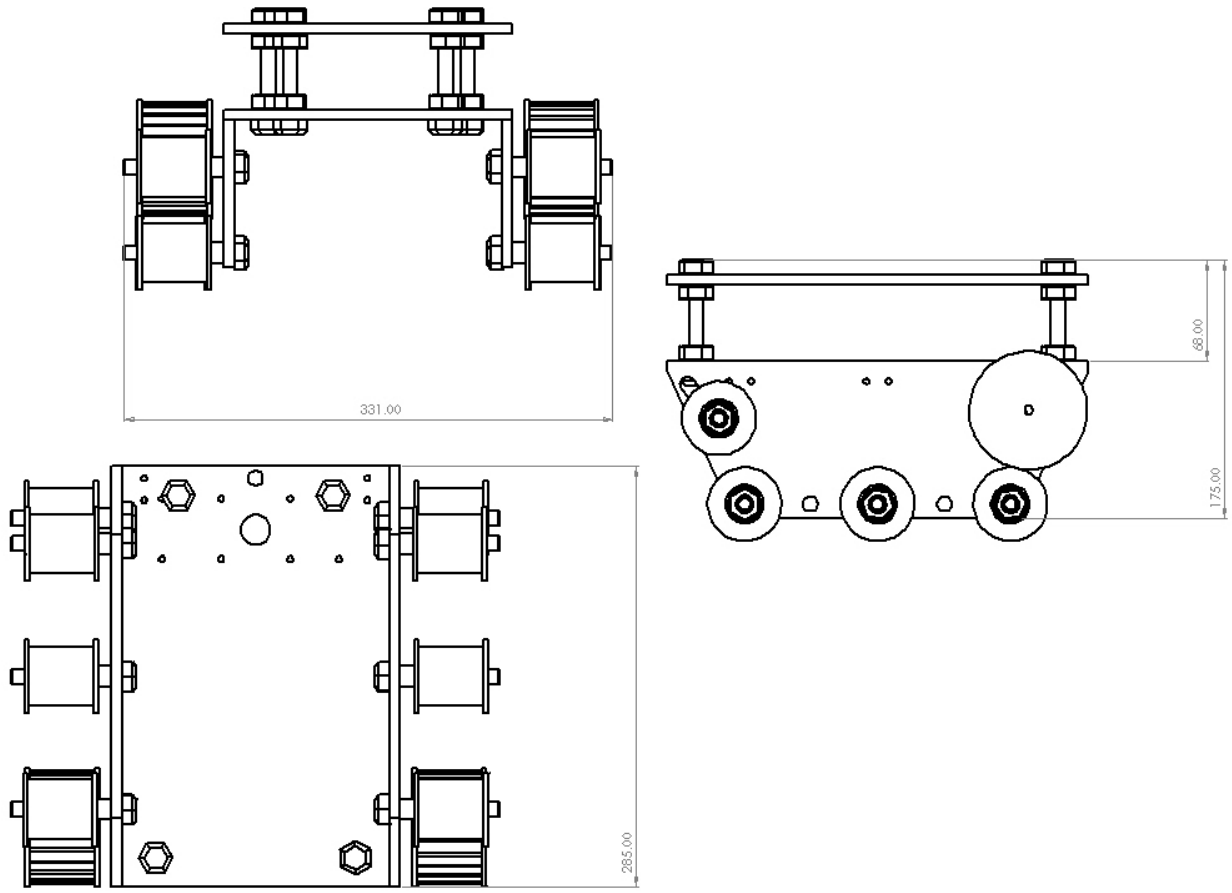

Figure 2.2.6a: 2D Model of entire rover chassis. Electronics and tracks not included. All dimensions in mm. Top Left: front view. Bottom Left: top-down view. Right: side view.

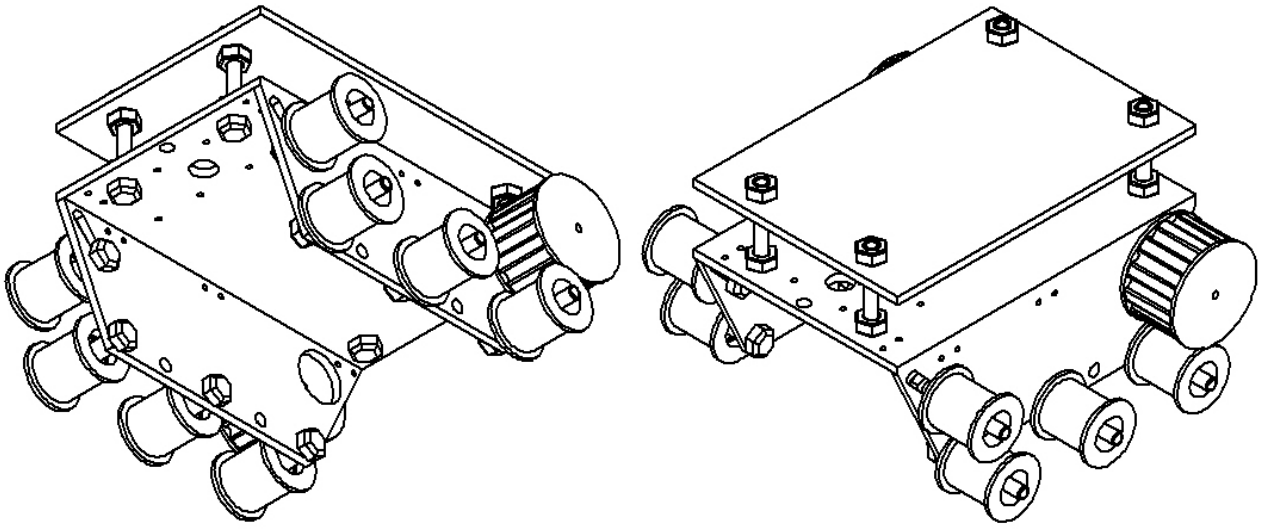

Figure 2.2.6b: 3D Model of entire rover chassis. Electronics and tracks not included. All dimensions in mm. Left: view from below. Right: view from above.

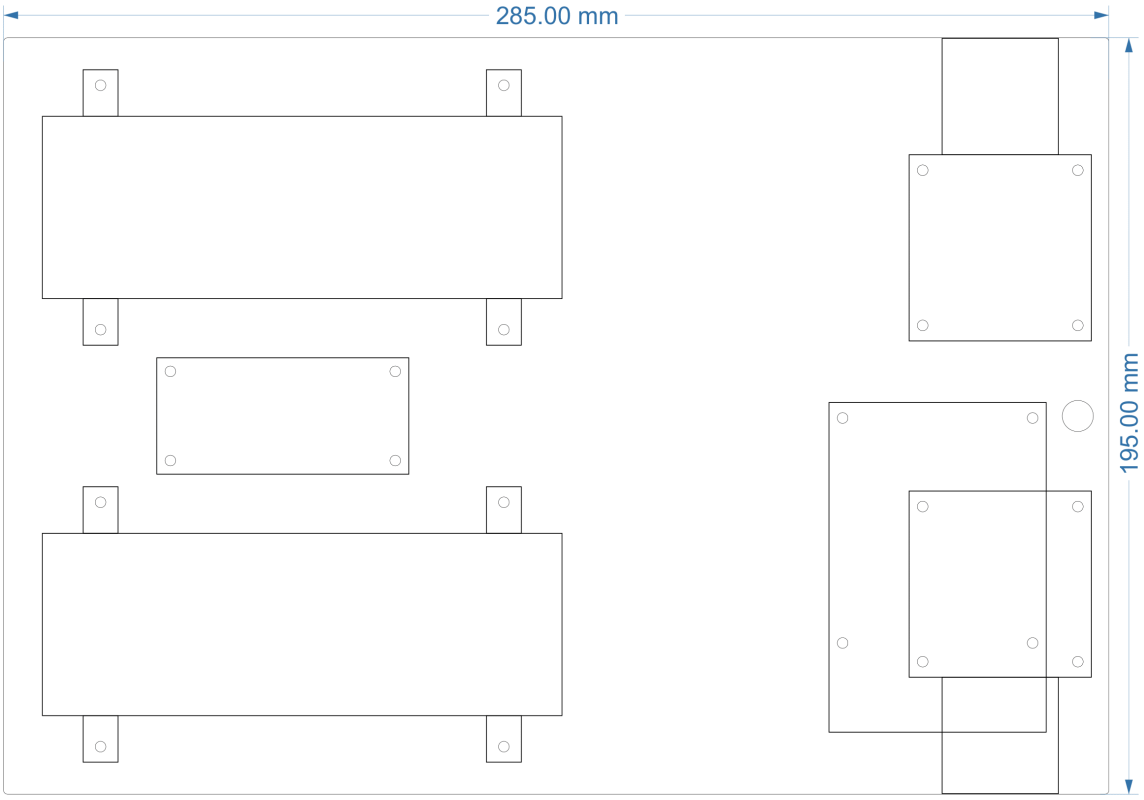

Figure 3.1.1: CorelDRAW 2019 design of main chassis - Version 1.

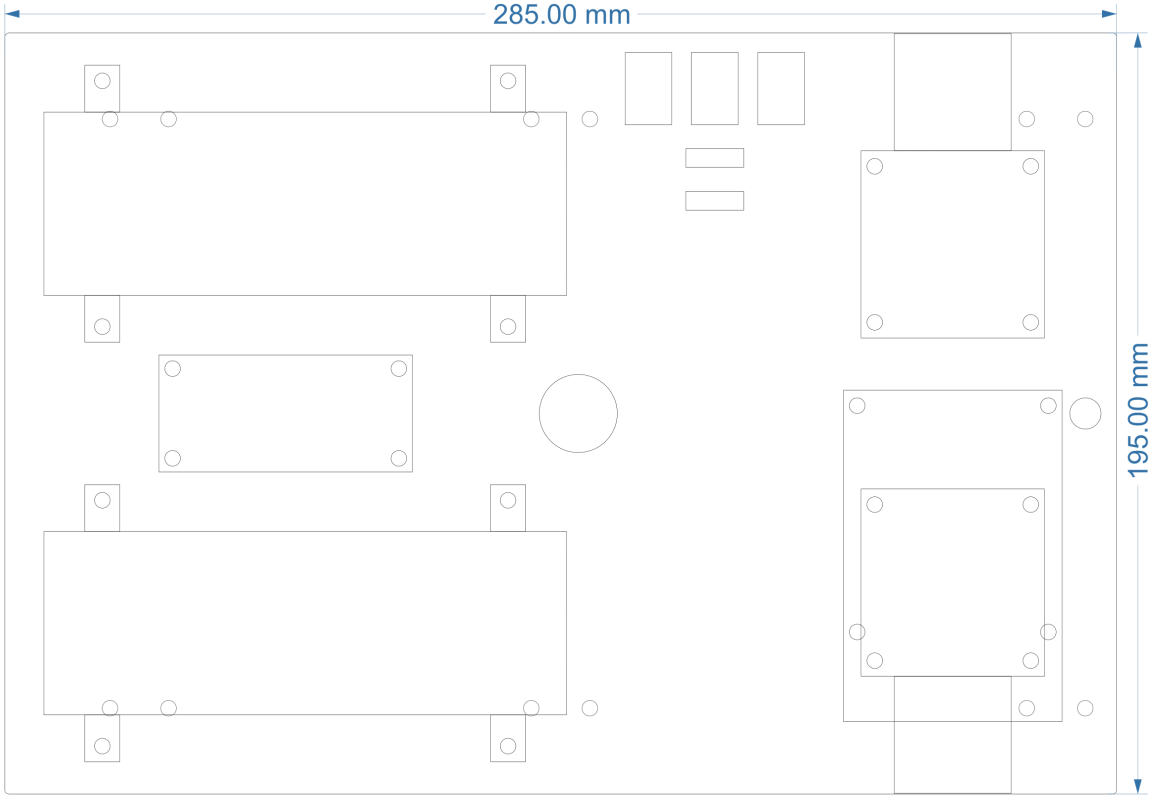

Figure 3.1.2: CorelDRAW 2019 design of main chassis - Version 2.

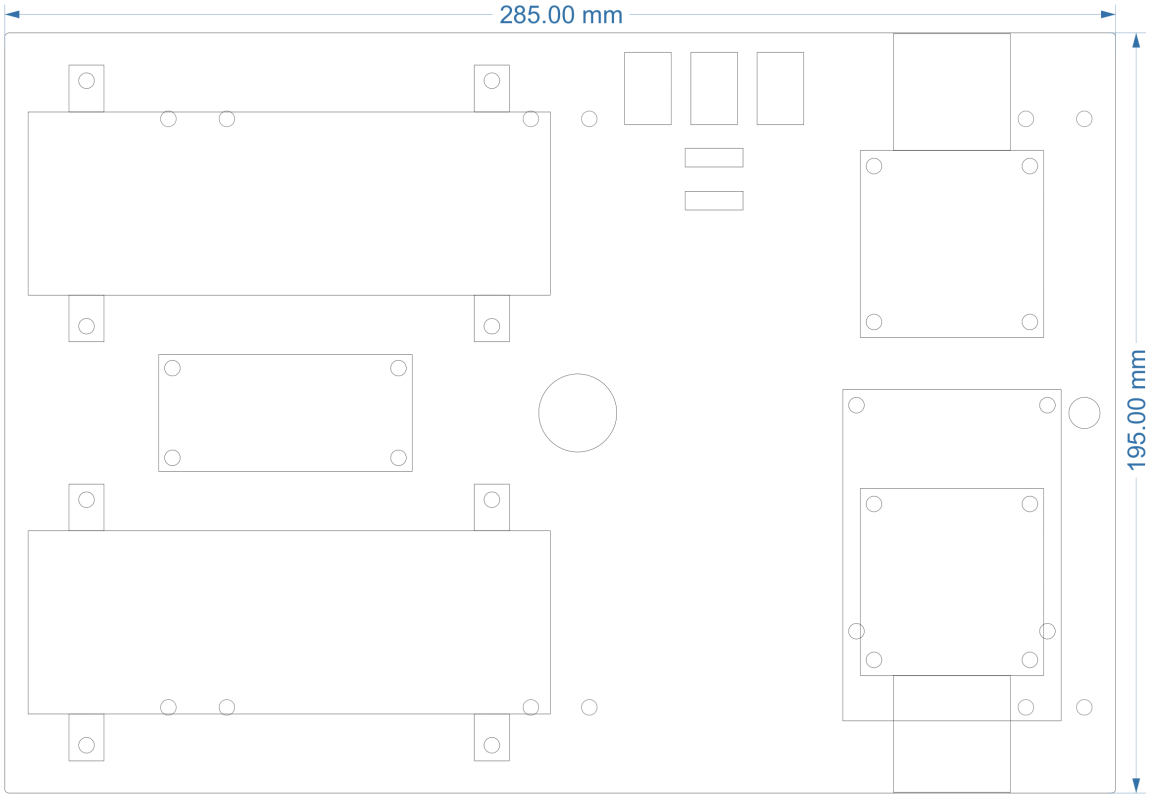

Figure 3.1.3: CorelDRAW 2019 design of main chassis - Version 3.

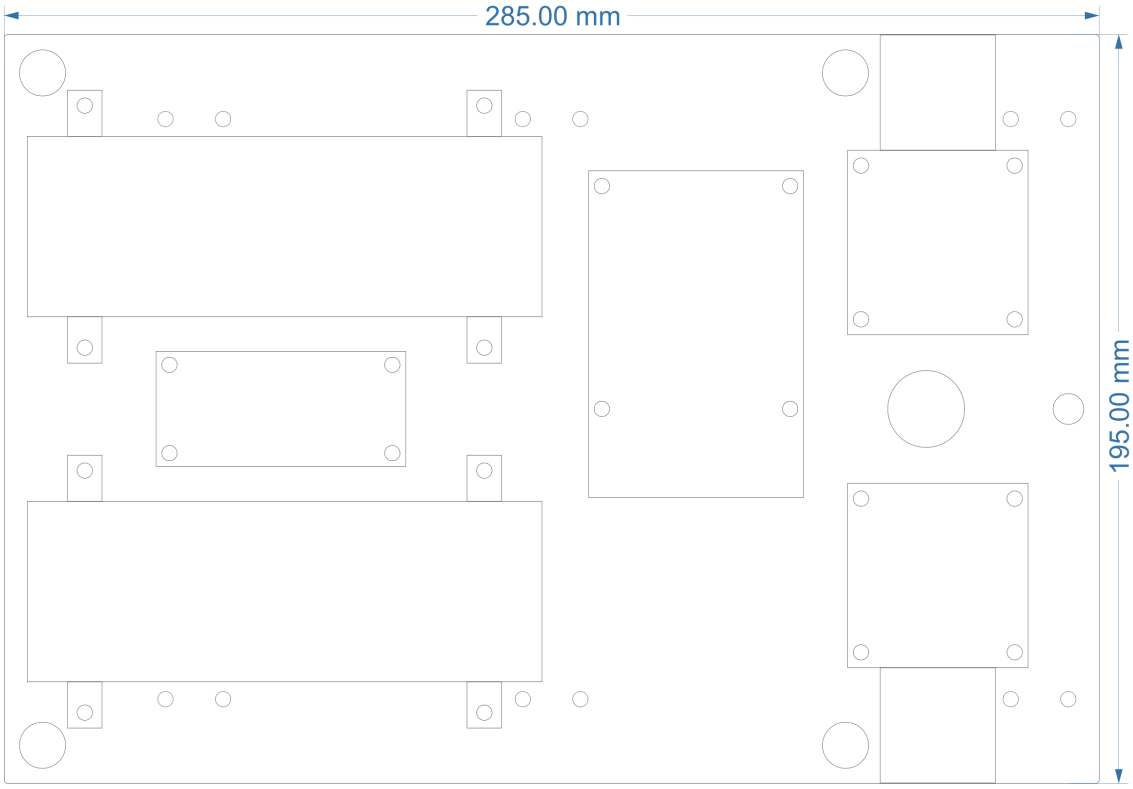

Figure 3.1.4: CorelDRAW 2019 design of main chassis - Version 4.

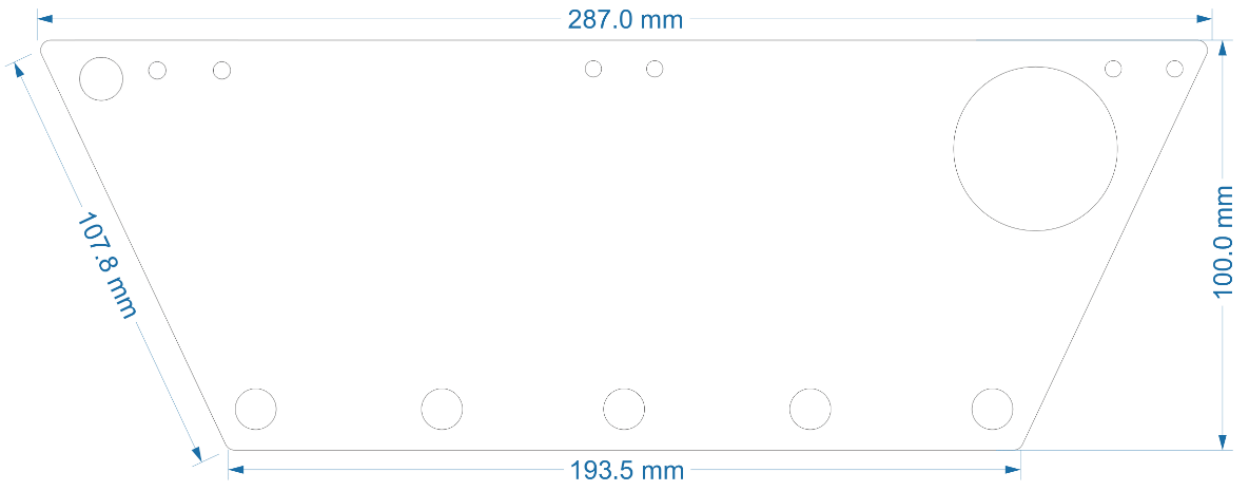

Figure 3.1.5: CorelDRAW 2019 design of side panel - Version 1.

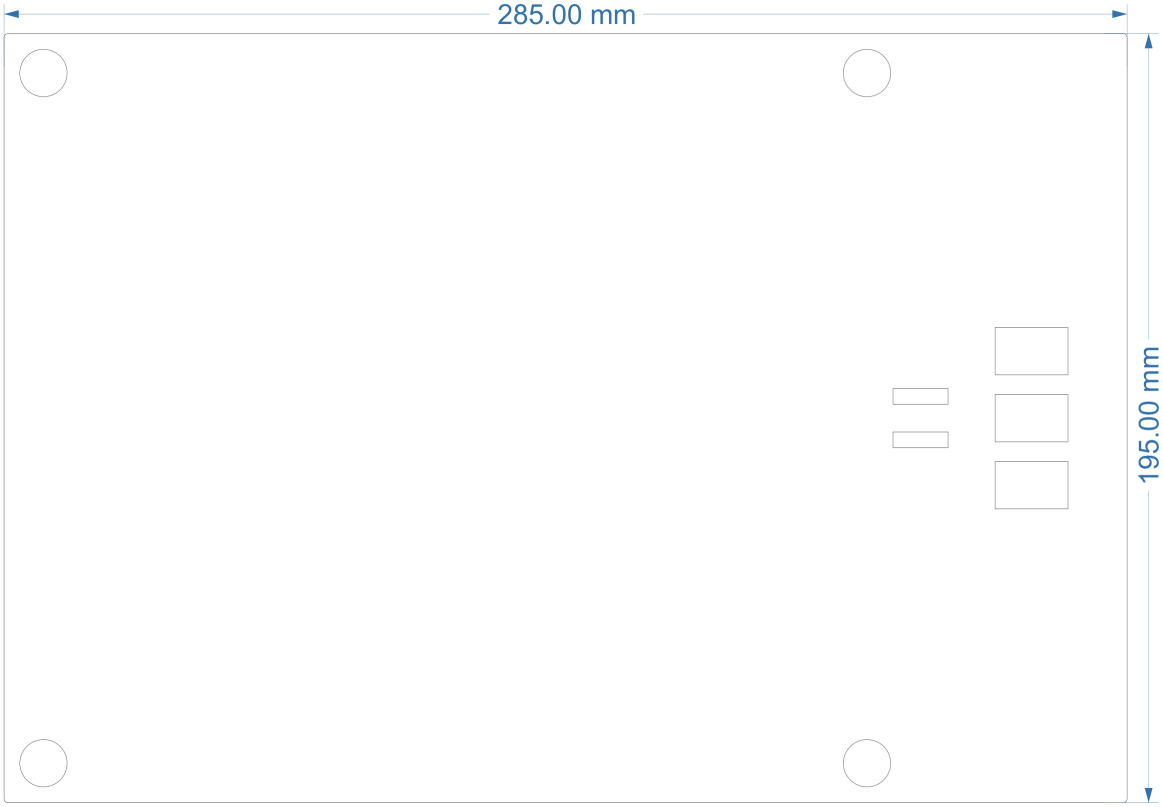

Figure 3.1.6: CorelDRAW 2019 design of top panel - Version 1.

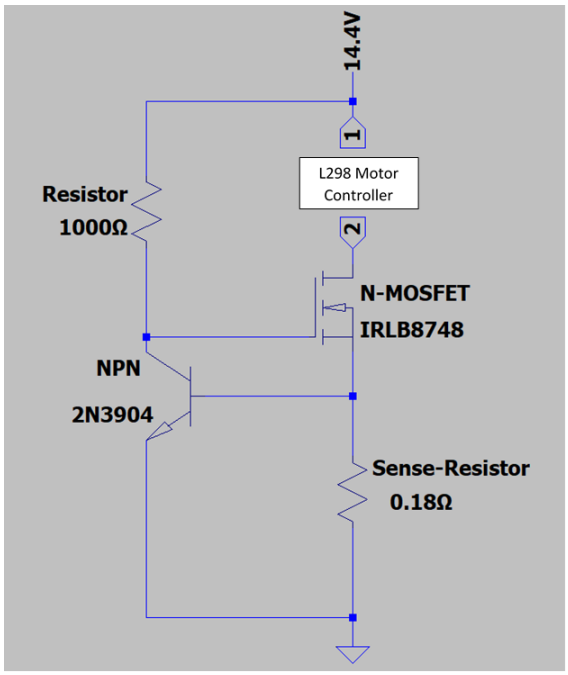

Figure 3.3.1: Schematic of current limiter circuit.

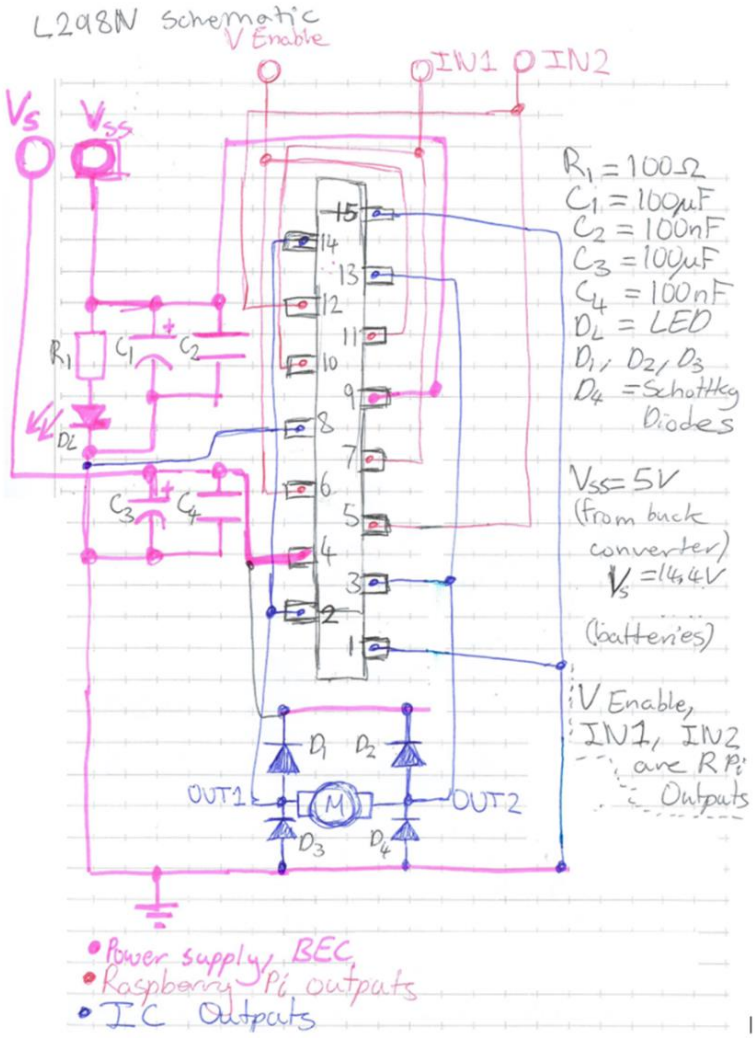

Figure 3.3.2: Schematic of L298N.

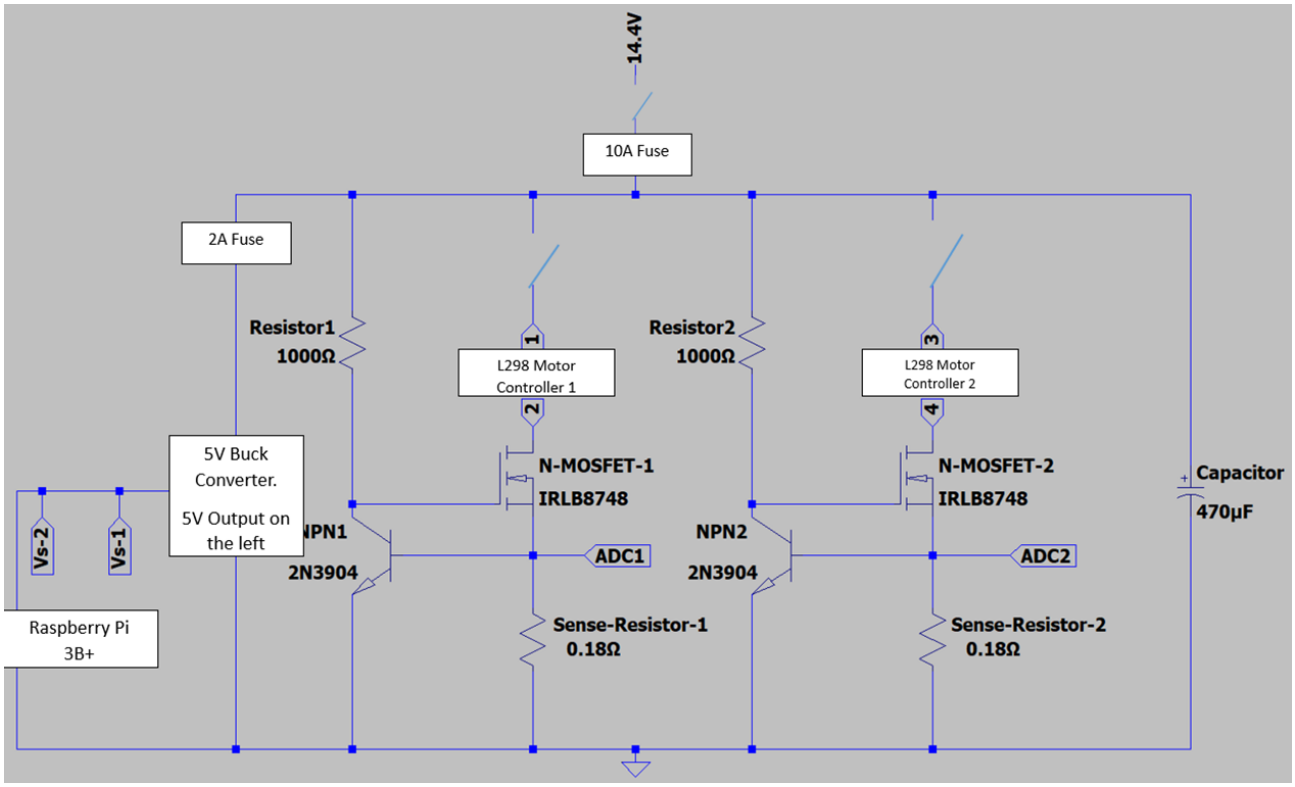

Figure 3.3.3: Complete circuit. Vs-1 and Vs-2 are the logic supply voltages (5V) to the L298 motor controllers.

Figure 3.4.1: Pin placements as dictated by code. Labelled version of [3].

Appendix B

TABLE 1.3.1: Bill of Materials	
Component	**Quantity**
Turnigy 7.2V 4.2Ah 10C NiMH	2
Tamiya Female Connector	10
Tamiya Male Connector	10
5V 3A Buck Converter	2
Motor Controller Heatsink	2
L298 Motor Controller IC	2
2A Fuse	1
10A Fuse	1
Fuse Holder	2
Stripboard	1
Illuminated LED switches	3
MFA 919D1001 Motors	2
25x8mm Ball Bearings	8
MCP3008 10-bit ADC	1
3W 0.18Ω Resistors	2
IRLB8748 Power MOSFETs	2
2N3904 NPN Transistors	2
Raspberry Pi 3B+	1
DIL socket (for MCP3008)	1
LEDs (Green)	2
Schottky Diodes	8
Electrolytic Capacitors: 100 μF	4
Capacitors: 100 μF	4
Capacitors: 470 μF	1
M10 75mm Bolt	4
M10 50mm Bolt	8
M10 Nut	16
Headers (1x3)	2
Headers (1x20)	2
Acrylic Sheets	4
30x50mm Aluminium Sheet	6
M3 36mm Steel Rod	10
M3 36mm Fibreglass Rod	62
PLA (3D printer filament)	~700g
AWG Threaded 16 gauge copper wire	~4m
Single Core Wire (for wiring between components)	~1m
Threaded Wire (for wiring between entire circuits)	~10m

TABLE 3.1.1: Changelog of chassis components			
Version	**Component**		
	Main chassis	**Side Panel**	**Top Panel**
1	285x195mm. Contains 2x motor, 2x battery, 1x Raspberry Pi, 1x Pi Zero W mounts and 1 cable hole. *Figure 3.1.1*	285x100mm. Contains 6x mounts for track guides, 6x mounting holes for attaching to main chassis, 1x hole for motor gearbox enclosure. *Figure 3.1.5*	285x195mm. Contains 4x M10 holes for mounting to Version 4 of main chassis, as well as switch and fuse cut-outs. *Figure 3.1.6*
2	285x195mm. Adds side panel mounting holes, 1 additional cable hole, switch and fuse cut-outs. *Figure 3.1.2*	285x100mm. Moved mounting holes to match Versions 3-5 of main chassis. Changes fixed track guide into adjustable one for tightening tracks. *Figure 2.1.2*	290x195mm. Length extended and M10 mounting holes relocated to match Version 5 of main chassis. Added engraving to switches and fuses for easy identification. *Figure 2.1.3*
3	285x195mm. Relocates some side panel mounting holes as they interfered with battery mounts. *Figure 3.1.3*	-	-
4	285x195mm. Adds top panel mounting holes, shifts batteries inward as a result. Removes switch and fuse cut-outs, and swaps Raspberry Pi with cable hole. *Figure 3.1.4*	-	-
5	290x195mm. Moves top panel mounting holes towards the centre, causing the chassis to be extended by 5mm. Pi Zero W is removed from design, shifting batteries inwards. Smaller cable hole removed, mounts for motor cable guard added. *Figure 2.1.1*	-	-

Appendix C

main.cpp

```
//main.cpp
#include "motor.h"

#include <thread>
#include <string.h>
#include <mcp3004.h>

#define PINLED  17
#define PINPWM0  (unsigned int) 18
#define PINCTL00 (unsigned int) 15
#define PINCTL01 (unsigned int) 14
#define PINPWM1  (unsigned int) 13
#define PINCTL10 (unsigned int) 6
#define PINCTL11 (unsigned int) 5

bool thread_EndFlag = false;
bool currentLimiter_EndFlag = false;

const int BASE = 100; //Starting pin number for MCP3008 virtual pins

void BlinkingLoop()
{
	while(!thread_EndFlag)
	{
		//blink for activity
		digitalWrite(PINLED, HIGH);	delay(500);
		digitalWrite(PINLED, LOW);	delay(500);
	}
}

//motors
motor m0;
motor m1;

int m0_Current;
```

```
int m1_Current;
int maxReading = 220; //50% of stall current

void currentLimiter()
{
	while(!currentLimiter_EndFlag)
	{
		m0_Current = analogRead(BASE + 0);
		delay(100);
		m1_Current = analogRead(BASE + 1);
		delay(100);



		//checking m0
		if(m0_Current > maxReading)
		{
			float m0_Duty = m0.GetDuty();
			if(m0_Duty < -0.1f)
			{
				m0.SmoothBrake();
			}
			else if(m0_Duty > 0.1f)
			{
				m0.SmoothBrake();
			}
		}
		if(m1_Current > maxReading)
		{
			float m1_Duty = m1.GetDuty();
			if(m1_Duty < -0.1f)
			{
				m1.SmoothBrake();
			}
			else if(m1_Duty > 0.1f)
			{
				m1.SmoothBrake();
			}
		}
```

```
	}
}

void setup()
{
	//activity LED
	pinMode(PINLED, OUTPUT);

	//motor zero
	m0 = motor(pwm(PINPWM0, 18, 100, 0.0f), PINCTL00, PINCTL01);

	//motor one
	m1 = motor(pwm(PINPWM1, 18, 100, 0.0f), PINCTL10, PINCTL11);

	//ADC chip
	mcp3004Setup(BASE, 0);
}

int main()
{
	if(wiringPiSetupGpio() == -1) return 1;

	setup();

	std::thread blinkingled(BlinkingLoop);
	std::thread m0_Thread = std::thread(&motor::Off, &m0);
	std::thread m1_Thread = std::thread(&motor::Off, &m1);
	std::thread currentThread = std::thread(&currentLimiter);

	while(1)
	{
		std::cout << "Enter a command:" << std::endl;
		char buffer[20];
		std::cin >> buffer;
		std::cout << "You entered '" << buffer << "'." << std::endl;

/*exit*/if(strcmp(buffer, "exit") == 0)
		{
```

```
                std::cout << “Stopping motors and exiting…” << std::endl;
                break;
        }

/*m0*/  else if(strcmp(buffer, “m0”) == 0)
        {
                int duty;
                std::cout << “Enter percentage duty cycle:” << std::endl;
                std::cin >> duty;

                if(m0_Thread.joinable()) m0_Thread.join();
                m0_Thread = std::thread(&motor::SmoothSetSpeed, &m0, duty / 100.0f);
        }

/*m1*/  else if(strcmp(buffer, “m1”) == 0)
        {
                int duty;
                std::cout << “Enter percentage duty cycle:” << std::endl;
                std::cin >> duty;

                if(m1_Thread.joinable()) m1_Thread.join();
                m1_Thread = std::thread(&motor::SmoothSetSpeed, &m1, duty / 100.0f);
        }

/*brake*/else if(strcmp(buffer, “brake”) == 0)
        {
                //NOTE: - This may cause one wheel to brake before the other if too many commands are
    input.
                If(m0_Thread.joinable()) m0_Thread.join();
                m0_Thread = std::thread(&motor::SmoothBrake, &m0);
                if(m1_Thread.joinable()) m1_Thread.join();
                m1_Thread = std::thread(&motor::SmoothBrake, &m1);
        }

/*fw*/ else if(strcmp(buffer, “fw”) == 0)
        {
                if(m0_Thread.joinable()) m0_Thread.join();
                if(m1_Thread.joinable()) m1_Thread.join();
                m0_Thread = std::thread(&motor::SmoothSetSpeed, &m0, 0.6f);
```

```
                m1_Thread = std::thread(&motor::SmoothSetSpeed, &m1, 0.6f);
        }

/*bw*/ else if(strcmp(buffer, “bw”) == 0)
        {
                if(m0_Thread.joinable()) m0_Thread.join();
                if(m1_Thread.joinable()) m1_Thread.join();
                m0_Thread = std::thread(&motor::SmoothSetSpeed, &m0, -0.6f);
                m1_Thread = std::thread(&motor::SmoothSetSpeed, &m1, -0.6f);
        }

/*lt*/ else if(strcmp(buffer, “lt”) == 0)
        {
                if(m0_Thread.joinable()) m0_Thread.join();
                if(m1_Thread.joinable()) m1_Thread.join();
                m0_Thread = std::thread(&motor::SmoothSetSpeed, &m0, 0.6f);
                m1_Thread = std::thread(&motor::SmoothSetSpeed, &m1, -0.6f);
        }

/*rt*/ else if(strcmp(buffer, “rt”) == 0)
        {
                if(m0_Thread.joinable()) m0_Thread.join();
                if(m1_Thread.joinable()) m1_Thread.join();
                m0_Thread = std::thread(&motor::SmoothSetSpeed, &m0, -0.6f);
                m1_Thread = std::thread(&motor::SmoothSetSpeed, &m1, 0.6f);
        }

/*off*/else if(strcmp(buffer, “off”) == 0)
        {
                if(m0_Thread.joinable()) m0_Thread.join();
                if(m1_Thread.joinable()) m1_Thread.join();
                m0_Thread = std::thread(&motor::Off, &m0);
                m1_Thread = std::thread(&motor::Off, &m1);
        }

        else if(strcmp(buffer ,”emstop”) == 0)
        {
                        m0.Brake();
```

```
                    m1.Brake();
            }
        }

        //ending threads
        m0_Thread.join();
        m1_Thread.join();

        thread_EndFlag = true;
        blinkingled.join();

        currentLimiter_EndFlag = true;
        currentThread.join();

        //stop and turn off motors
        m0_Thread = std::thread(&motor::Off, &m0);
        m1_Thread = std::thread(&motor::Off, &m1);

        m0_Thread.join();
        m1_Thread.join();

        return 0;
}
```

motor.h

```
//motor.h

#include "pwm.h"

enum direction{forwards, backwards, stopped};

class motor
{
private:
        pwm m_Pwm;
        unsigned int m_CTRLPIN0;
        unsigned int m_CTRLPIN1;
        direction m_Dir;
```

```
public:
    motor(){};
    motor(pwm pwmObject, unsigned int controlPin0, unsigned int controlPin1)
        :m_Pwm(pwmObject), m_CTRLPIN0(controlPin0), m_CTRLPIN1(controlPin1)
    {
        pinMode(m_CTRLPIN0, OUTPUT);
        pinMode(m_CTRLPIN1, OUTPUT);
        Off();
    }

    void Brake()
    {
        m_Pwm.SetDuty(0.0f);
        digitalWrite(m_CTRLPIN0, HIGH);
        digitalWrite(m_CTRLPIN1, HIGH);
        m_Dir = stopped;
    }

    void SmoothBrake()
    {
        m_Dir = stopped;
        m_Pwm.SmoothSetDuty(0.0f);
        digitalWrite(m_CTRLPIN0, HIGH);
        digitalWrite(m_CTRLPIN1, HIGH);
        delay(200);
    }

    //use with care, sets the speed instantaneously.
    void SetSpeed(float duty)
    {
        //+ve ->forwards
        //-ve ->backwards
        if(duty > 0)
        {
                digitalWrite(m_CTRLPIN0, HIGH);
                digitalWrite(m_CTRLPIN1, LOW);
                m_Pwm.SetDuty(duty);
```

```
                m_Dir = forwards;
        }
        else if(duty < 0)
        {
                digitalWrite(m_CTRLPIN0, LOW);
                digitalWrite(m_CTRLPIN1, HIGH);
                m_Pwm.SetDuty(-duty)
                m_Dir = backwards;
        }
}

//slowly sets the correct speed. Accounts for direction changes.
void SmoothSetSpeed(float duty)
{
        //+ve ->forwards
        //-ve ->backwards
        if(duty > 0)
        {
                if(m_Dir == backwards)
                {
                        SmoothBrake();
                }
                digitalWrite(m_CTRLPIN0, HIGH);
                digitalWrite(m_CTRLPIN1, LOW);
                m_Dir = forwards;
                m_Pwm.SmoothSetDuty(duty);
        }
        else if(duty < 0)
        {
                if(m_Dir == forwards)
                {
                        SmoothBrake();
                }
                digitalWrite(m_CTRLPIN0, LOW);
                digitalWrite(m_CTRLPIN1, HIGH);
                m_Dir = backwards;
                m_Pwm.SmoothSetDuty(-duty);
        }
```

```
    }

    void Off()
    {
        //discharge first
        m_Pwm.SmoothSetDuty(0.0f);
        digitalWrite(m_CTRLPIN0, HIGH);
        digitalWrite(m_CTRLPIN1, HIGH);

        m_Dir = stopped;

        //wait for discharge
        delay(100);

        //off
        digitalWrite(m_CTRLPIN0, LOW);
        digitalWrite(m_CTRLPIN1, LOW);

    }

    float GetDuty(void)
    {
        if(m_Dir == forwards) return m_Pwm.GetDuty();
        if(m_Dir == backwards) return -m_Pwm.GetDuty();
        return 0;
    }

};
```

pwm.h

```
//pwm.h

#include <iostream>
#include <wiringPi.h>



class pwm
{
private:
```

```
        unsigned int m_Pin;
        float m_Duty;
        unsigned int m_Range;

public:

        //Constructors
        pwm(void){};
        pwm(unsigned int pinNo, unsigned int prescaler, unsigned int range, float duty)
                :m_Pin(pinNo), m_Duty(duty), m_Range(range)
        {
                pinMode(pinNo, PWM_OUTPUT);
                pwmSetClock(prescaler); //pre-scaler equivalent
                pwmSetRange(range); //duty resolution
                pwmSetMode(PWM_MODE_MS);
                pwmWrite(pinNo, (unsigned int)(duty * range));
        }

        //Immediately sets duty to the specified value
        void SetDuty(float duty)
        {
                if(duty < 0.0f || duty > 1.0f)
                {
                        std::cout << "SetDuty(float) was passed an invalid value." << std::endl;
                        return;
                }
                m_Duty = duty;
                pwmWrite(m_Pin, (unsigned int)(duty * m_Range));
                return;
        }

        //Will progress the pwm's duty towards the specified value.
        //Should be called by a thread, ideally
        void SmoothSetDuty(float duty)
        {
                if(duty > 1) duty = 1;
                if(duty < 0) duty = 0;

                while(1)
                {
                        //NOTE: epsilon should always be smaller than step.
```

```
                float step = 0.02f;
                float epsilon = 0.019f;

                if(m_Duty > duty + epsilon)
                {
                        m_Duty -= step;
                }
                else if(m_Duty < duty - epsilon)
                {
                        m_Duty += step;
                }
                else
                {
                        std::cout << "value found" << std::endl;
                        break;
                }

                if(m_Duty < epsilon) m_Duty = 0.0f;
                if(m_Duty > 1.0f - epsilon) m_Duty = 1.0f;

                std::cout << "m_Duty: " << m_Duty << std::endl;
                pwmWrite(m_Pin, (unsigned int)(m_Duty * m_Range));
                delay(20);
        }
    }

    float GetDuty(void) {return m_Duty;}
};
```