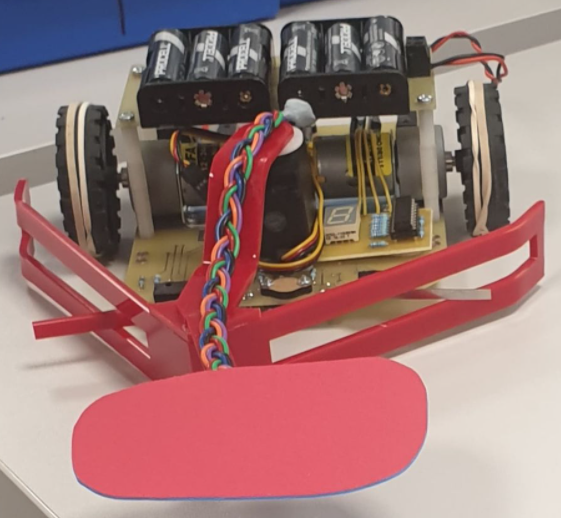
Derbot Technical Report

Team 1



Charlie Herbert (CH), Oliver Towers (OT)

and Mateo Ceballos Querol (MC)

Table of Contents

[List of figures 3](bookmark://_Toc59121972#_Toc59121972)

[Introduction (OT) 4](bookmark://_Toc59121973#_Toc59121973)

[PIC Resources and set-up (MC) 4](bookmark://_Toc59121974#_Toc59121974)

[White line following and stopping (MC) 5](bookmark://_Toc59121975#_Toc59121975)

[Hardware and circuit design 5](bookmark://_Toc59121976#_Toc59121976)

[Programming 7](bookmark://_Toc59121977#_Toc59121977)

[Evaluation 8](bookmark://_Toc59121978#_Toc59121978)

[Obstacle detection and moving (CH) 8](bookmark://_Toc59121979#_Toc59121979)

[Hardware and system design 8](bookmark://_Toc59121980#_Toc59121980)

[Programming 11](bookmark://_Toc59121981#_Toc59121981)

[I2C communication and display (OT) 13](bookmark://_Toc59121982#_Toc59121982)

[Hardware and circuit design 13](bookmark://_Toc59121983#_Toc59121983)

[Programming 17](bookmark://_Toc59121984#_Toc59121984)

[Testing and Merging codes (CH) 18](bookmark://_Toc59121985#_Toc59121985)

[Performance in the competition 19](bookmark://_Toc59121986#_Toc59121986)

[White line following (MC) 19](bookmark://_Toc59121987#_Toc59121987)

[Obstacle detection and removal (CH) 19](bookmark://_Toc59121988#_Toc59121988)

[I2C communication and display (OT) 19](bookmark://_Toc59121989#_Toc59121989)

[Conclusion (OT) 19](bookmark://_Toc59121990#_Toc59121990)

[References 20](bookmark://_Toc59121991#_Toc59121991)

[Appendices 21](bookmark://_Toc59121992#_Toc59121992)

[Appendix 1 – Team plan 21](bookmark://_Toc59121993#_Toc59121993)

[Appendix 2 – While line following and stopping code flow chart 26](bookmark://_Toc59121994#_Toc59121994)

[Appendix 3 – Obstacle decetion and removal code flow chart 27](bookmark://_Toc59121995#_Toc59121995)

[Appendix 4 – I2C communication flow chart 28](bookmark://_Toc59121996#_Toc59121996)

[Appendix 5 – Derbot full code 29](bookmark://_Toc59121997#_Toc59121997)

# List of figures

[Figure 1 - MCC pin module GUI 5](#_Toc59122078)

[Figure 2 –underside of the Derbot 6](#_Toc59122079)

[Figure 3 - ECCP1&2 configuration GUI 6](#_Toc59122080)

[Figure 4 - Timer 2 configuration GUI 6](#_Toc59122081)

[Figure 5 - whilte line following embedded c code using MPLAB X IDE 7](#_Toc59122082)

[Figure 6 - (a) practice track (b) replica stopping square 8](#_Toc59122083)

[Figure 7 - Diagram showing how the Derbot will react depending if the object is lit or unlit. 9](#_Toc59122084)

[Figure 8 – Picture of the prototype plough designed (left) and a picture from during the laser cutting of the acrylic for the final plough used (right) 10](https://unimailderbyac-my.sharepoint.com/personal/100483889_unimail_derby_ac_uk/Documents/Embedded%20system%20-%20Derbot%20shared%20files/Derbot%20Technical%20Report%20backbone.docx#_Toc59122085)

[Figure 9 - Picture showing the final acrylic plough sitting in place where it will be attached. 10](#_Toc59122086)

[Figure 10 –Diagram showing the new design for the LDR arm extending in front of the plough (left), and a picture showing the outcome of the new LDR arm design on the Derbot (right). 11](#_Toc59122087)

[Figure 11 - Screenshot from the MCC in MPLABX showing the general settings used for the ADC. 12](#_Toc59122088)

[Figure 12 - Screenshot of the ADC function used within the final code. 12](#_Toc59122089)

[Figure 13 - Screenshot of the two servo functions created within the final code. 12](#_Toc59122090)

[Figure 14 - Screenshot showing the code written based on the flow chart shown in appendix 2. 13](#_Toc59122091)

[Figure 15 - an image showing the actions of the serial data line (SDA) and the serial clock (SCL) as they send a single packet of data. (Learn.Digilentinc | Expanding I/O with the I/O Expander, 2020) 14](#_Toc59122092)

[Figure 16 - an image showing the control byte used for the read and write slave address allocation (Left). a schematic showing the pin layout for the MCP23008 I/O expander (right). (microchip, 2020) 15](#_Toc59122093)

[Figure 17 - a diagram showing the pin connections on the seven-segment display and which segment of the display they correspond to. Left a diagram showing the locations of the both the segments within the display and the pin layout. Right 15](#_Toc59122094)

[Figure 18 - an image showing the connection made on the breadboard for the seven-segment display and I/O expander (left). (right) an image showing breadboard to development board connections. 15](#_Toc59122095)

[Figure 19 - A circuit design built on Multisim showing the proposed and built PCB without the SDA and SCL lines fixed and taken before the pull up resistors (left). a circuit design showing the corrected SCL and SDA line placement (right). 16](#_Toc59122096)

[Figure 20 - a schematic showing the altered Multisim schematic within Ultiboard. The separation of the SDA and SCL line is Cleary visible here.(left) an image showing the printed circuit board based on the original design schematic. A double a batter has been included within the image to show scale. (right). 16](#_Toc59122097)

[Figure 21 - Picture showing the addiction of a shade at the end of the LDR arm to reduce the ambient light hitting the LDR. 18](#_Toc59122098)

# Introduction (OT)

This report details how and why crucial elements of Team 1’s Derbot were designed, programmed, constructed and analysed. The three main areas were distributed between each team member. These areas were: White line following and stopping which included the use of optos to control the speed of the motors and keep the Derbot on the line. Obstacle detection/movement consisted of LDR’s to detect whether the obstacle was lit or unlit and the construction of electro/mechanical hardware to move the obstacles accordingly. I2C focused on the communication between the Derbot and an exterior chip which controlled the seven-segment display showing the number of lit and unlit obstacles moved. Throughout this process the team worked collaboratively on the design, to merge and test the codes and setup additional factors such as the PIC resources.

# PIC Resources and set-up (MC)

Before compiling any of the derbot code together, the mcc file was generated containing each of the peripherals that would be used for the final design. The PIC18F25K22 consists of many different peripherals which through the mcc, make it easier to control certain I/O ports. Some of the peripherals that the PIC has available are ADC (Analogue to Digital Converter), DAC (Digital to Analogue Converter), ECCP (Enhanced Capture Compare PWM), MSSP (Master Serial Synchronous Port) (I2C) and the 7 timers provided (Timer 0 – Timer 6); many I/O ports are also included separated into PORT A, PORT B and PORT C each having 8 pins each for a total of 24 I/O pins.

Table 1 in appendix 1 shows the peripherals used for the final derbot challenge and how each contributed to the functioning of the embedded system. The ECCP 1 and 2 module were the most important as they were required to move the left and right motors. These provided the PWM signal with adjustable duty cycle which was proportional to the speed of the motors. It is important to note that in the end, timer 2 was the only timer used (for ECCEP) as the others were not necessary

Figure 5 in appendix shows a visual representation of the pinout of the SPDIP28 variant of the PIC18F25K22. An external crystal oscillator of 4MHz was connected to OSC1 and OSC2 (pins 9 and 10 of the PIC respectively). Since the derbot PCB was already made and provided with all the connections, it was only a matter of configuring the pinout in the MCC. Figure 4 shows the pin module of the MCC, this GUI provided a means of defining whether a pin was a digital or analogue input or output pin. Due to the design of the PIC, some peripherals were hard-wired to certain pins such as the ECCP or I2C, so all was needed was to define whether a specific pin was an input or output. For example, the LDR and optocouplers were inputs while the debugging LEDs and motors where outputs.

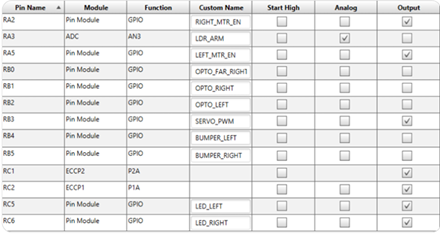


Figure 1 - MCC pin module GUI

Figure 1 shows the final pin module for the derbot with each pin’s custom names and whether they are analogue or digital, input or output. As mentioned, OSC1 and OSC2 pins were connected to a 4MHz crystal oscillator. MCLR button was reserved for the reset switch, this reset the PIC once pressed. Pins 27 and 28 (RB6 and RB7) are PCG and PGD, these are connected to the RJ11 port which allow bidirectional communication with the ICD4 used to upload the code. RC3 and RC4 are SCK and SDA respectively, used for I2C. These pins were not configured using the MCC but were configured in the TRIS bits in the code.

The only analogue pin was the input from the LDR, this determined if an obstacle along the white line was lit or unlit. As mentioned above, the SCL and SDA pins are not included in the mcc but were defined in the code using the TRIS registers.

# White line following and stopping (MC)

## Hardware and circuit design

The most important task of the derbot was making sure that it followed the white line until the end and then stopped in the middle of the square.

Circled in red in figure 2 are the three opto-couplers named (from left to right) “OPTO\_LEFT”, “OPTO\_RIGHT” and “OPTO\_FAR\_RIGHT” in the pin module of the MCC from the previously mentioned section PIC resources. The left and right optos were constantly checked to make sure that the derbot was on the line and the far right was used for the end, once all three were at a logical high this meant that it detected the horizontal line at the end, just before the stopping square.



Figure 2 –underside of the Derbot

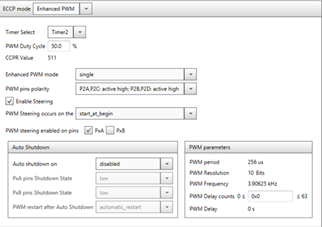


Figure 3 - ECCP1&2 configuration GUI

The only PIC peripherals used in the white line following/stopping were the ECCP1&2 and the Timer 2 modules. Figure 4 shows the MCC configuration for the ECCP1 and ECCP2 peripherals that were used to control the left and right motors. Both were set in single mode, meaning that there were two independent PWM signals generated from the PIC going into the L293D motor driver.

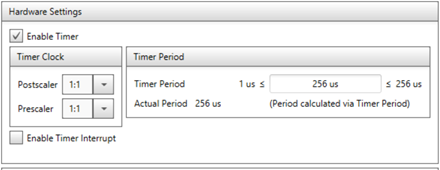


Figure 4 - Timer 2 configuration GUI

Figure 4 shows the Timer2 MCC peripheral. This was set to the default period of 256us and is what was used to generate the PWM signal for the ECCP1&2 peripherals.

## Programming

Once the MCC was set up correctly, the white line following algorithm had to be thought out. To reduce complexity in the algorithm, it was decided that polling would be used instead of using interrupts. Though this was easier for the white line following algorithm on its own, it presented a challenge when compiling the different parts of the derbot code together.

the figure in appendix 2 shows the flowchart which describes the algorithm for the white line following and stopping (see appendix 2 for a larger flowchart). Initially, the L293D is set to high for both motors. The speed is 100% for both except that by design of the motors, they are not the same. It was observed that the right motor was running slightly faster than the left so the speed was reduced to 90%. There are three conditions, one for each of the optos. At the beginning of the track, the derbot starts moving forward, as soon as track curves towards the right or left, the respective opto reads logical high. When the right opto reads high, the right motor reduces speed causing the derbot to turn the other way and vice versa for the left opto.

The final condition was when the far right opto was set to high. The opto was positioned further away from the other two so that it would not be triggered when moving along the curves, hence the allocated name. The far right opto was only triggered once the derbot reached the horizontal line at the end, which signalled the end of the track. From the flowchart, the delay of 450ms was for the derbot to reach the approximate middle of the square. Finally, it made a complete stop in the centre, in the derbot challenge, the I2C display function was executed indefinitely.



Figure 5 - whilte line following embedded c code using MPLAB X IDE

Figure 5 shows the code containing the main function with the super-loop as well as the motor function and Boolean definitions. As mentioned earlier, it was decided not to use interrupts for the white line following but rather polling, to reduce the complexity of the code. The #include contains the mcc file with the required peripherals: ECCP1&2 and Timer 2. The first line of code in the main function initialises the peripherals. Following the flowchart above, the super-loop starts with both motors on at full speed (right motor slightly slower).

The motor\_fwd() function is a custom sub-routine created to facilitate the enabling and disabling of the left and right motors while setting the speed at the same time. This function takes three input parameters: “L” or “R” to indicate which motor is being referenced, the speed (255 being 100% speed) and a Boolean value, high or low. This sub-routine makes it easier to see in the super-loop which motor is being enabled or disabled and at what speed. The alternative, longer solution would be to not have a sub-routine but to enable the left or right motor then set the speed.

Finally, three Boolean variables were defined. Each of these Boolean variables denotes one of the three optos. For each of the optos, it returned true when there was light, when the white line is detected on the track. While these Boolean variables were not necessary, it also provides a bit of clarity as it reduces the clutter within each of the conditional statements of the super-loop.

## Evaluation

The first few attempts at testing the algorithm were successful but the derbot was not very stable. The problem was that the speed difference was too high so 80% of the max speed was chosen through trial and error as a suitable speed. This speed resulted in smooth following of the track curves; shown in figure 6(a).

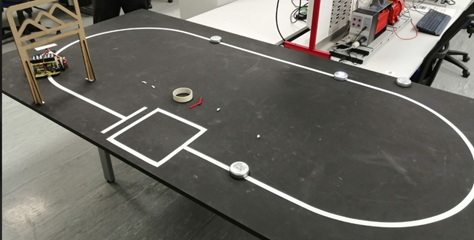
 

Figure 6 - (a) practice track (b) replica stopping square

Unfortunately, the track was not always available. As shown in figure 6(b), a replica of the square from the track was made. This measured 250mm2, with each of the edges measuring 17mm wide (Wilson, 2020). It was made of white paper, like the final challenge track. This is because the OPB608A can best detect white colours (Electronics, 2020).

# Obstacle detection and moving (CH)

## Hardware and system design

As mentioned in the design brief, the final Derbot must move objects to the left and right of the track depending if they are lit or unlit; the total number of each type of object should then be displayed at the end of the track. There were a few initial ideas of how this could be done, ranging from arms reaching out in front of the Derbot and knocking the objects to their correct side, to cups on extending arms to move the objects, however none of these designs provided both precision and speed; both characteristics which were desired from the Derbot. After redesigning, the final idea was submitted as part of our team plan document, shown in appendix 1. This idea provides both precision and speed with the objects still being detected, counted, and sorted correctly without the Derbot having to stop; although the final design had to be altered from the initial team plan submitted, the idea as a whole was kept constant throughout.

There would be an arrow shaped bumper, like a snow plough, fixed to a servo on the front of the Derbot, the servo would have 120 degrees of movement, allowing the plough to be in a left (0 degrees) or a right (120 degrees) position (Electronicos Caldas, n.d.). Attached to the front of the plough would be an arm extending by 100 mm, with an LDR (Light Dependent Resistor) fixed at the end. The bumper switches would be extended through the side of the plough, so they can be triggered as the object is pushed aside.

The ploughs default position would be to the right, unless the LDR read a high light level at which point the plough would be moved to the left position. This means that as the object is approached, it will be pushed to the left side of the track, triggering the left bumper switch as it slides down the left side of the plough. If the LDR detects higher levels of light, the plough will be moved to the left position, pushing the lit object to the right of the track before moving back to its default right position again. This is shown in figure 7. The bumper switches will be used to count the number of each object and these variables will be used with the I2C section of the Derbot.

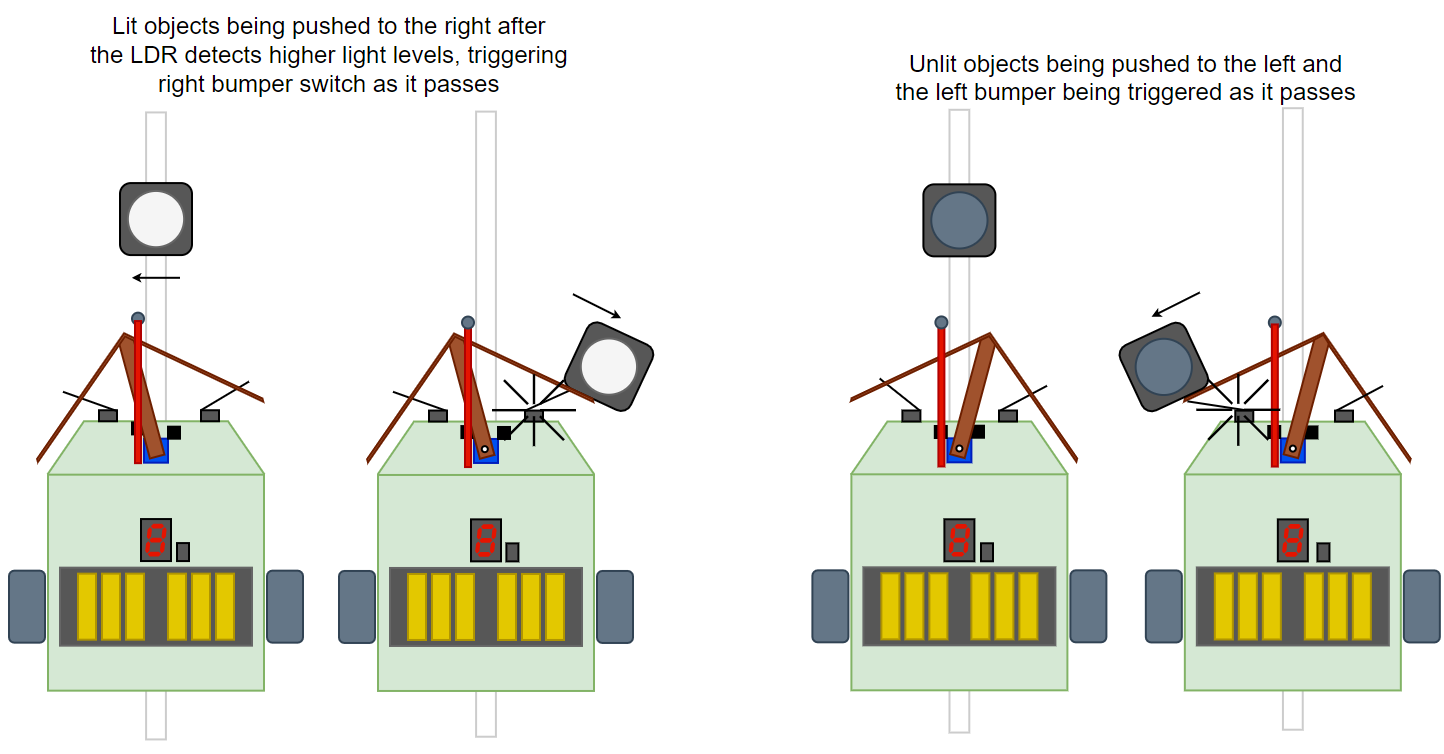


Figure 7 - Diagram showing how the Derbot will react depending if the object is lit or unlit.

To achieve this design, the Derbots rear LDR (LDR3) and servo connections on the PCB would be used as well as the two front microswitches. Additional hardware needed includes:

* Acrylic to build the bumper
* Some short lengths of acrylic for the extending arms
* A LDR
* A 10KΩ resistor
* A servo
* Glue dots or similar double-sided sticky pads

First, the soldering needed to connect the servo and the LDR to the Derbot where completed. The three connections needed for the servo were found at the front of the PCB; a short piece of multicore wire was soldered in place, with a 3X1 male header soldered to the other end. For the LDR, the 10KΩ resistor was put in place to create a voltage divider circuit, with the voltage across the LDR being sampled for the ADC process (Silonex, 2018).

The plough was to be designed and built using the laser cutter and acrylic in the Makerspace at the university. To start with, a prototype plough was made from cardboard and tested for size as shown in the left of figure 8 below.

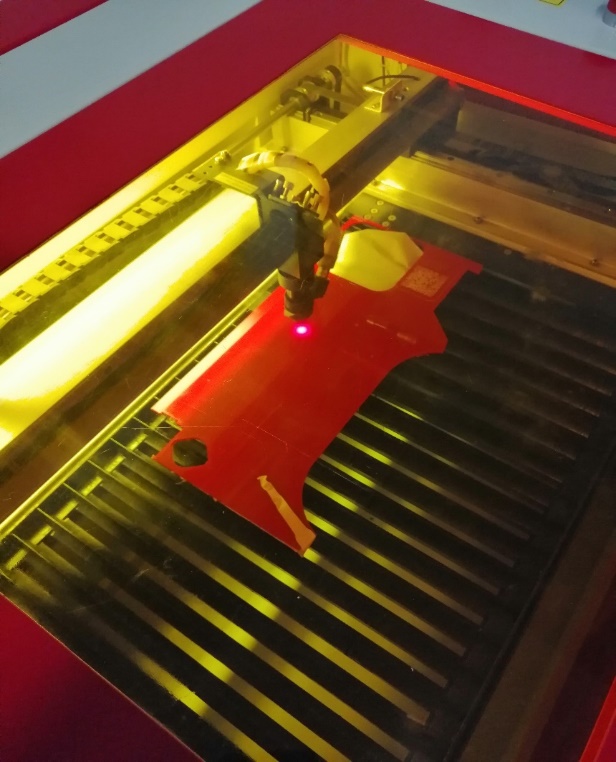
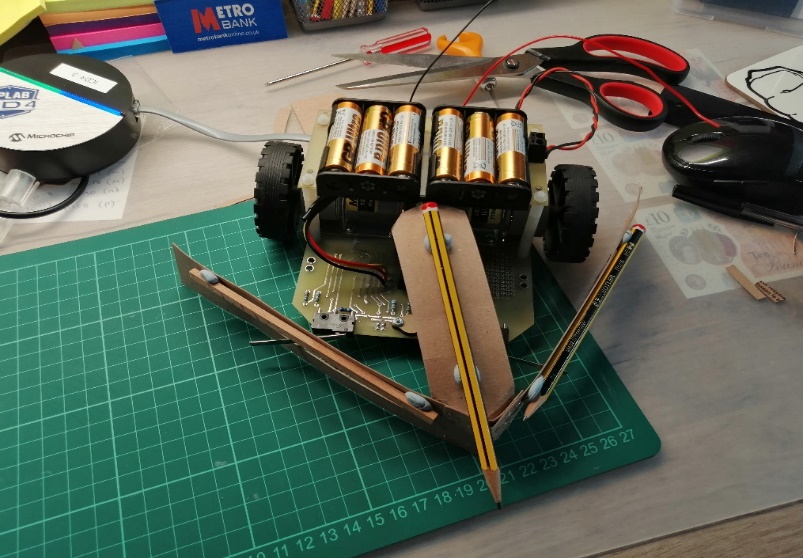


Figure 8 – Picture of the prototype plough designed (left) and a picture from during the laser cutting of the acrylic for the final plough used (right)

The dimensions from the prototype were recorded and taken into the Makerspace where they were put into the CAD (Computer Aided Design) software and cut out using the laser cutter also shown in the right of figure 8.

Once the parts were cut out and glued together using acrylic glue, the plough could be heat-bent and fitted to the servo, temporarily placed onto the Derbot to check everything fit together as shown in figure 9.

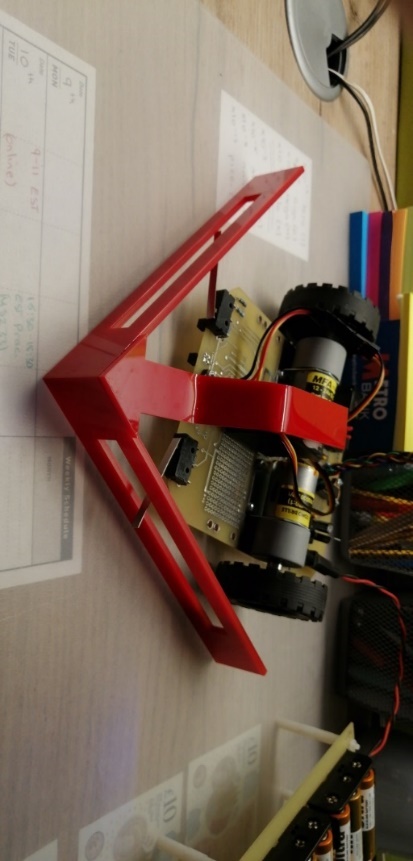


Figure 9 - Picture showing the final acrylic plough sitting in place where it will be attached.

The next part was to route the LDR connections from the back of the Derbot PCB to an extending arm in front of the plough. The rear LDR connections were chosen rather than the two on the front of the Derbot as the servo arm would need a clear area to move between the left and right position, placing the connections for the LDR in its path would have caused more problems that it solved, so the area was kept as clear as possible.

At this point, it became clear that the initial design had to be changed; instead of a separate arm reaching out in front of the plough for the LDR, an extension could be added to the front of the plough instead. This meant the LDR would be more stable and not jump around while the Derbot was in motion, something that was discovered when we first sent the Derbot around the track.

The arm would be made of a short length of acrylic and in a fixed position over the white line. It would be at a slight angle to counter for the default right position of the plough. This is shown in the left of figure 10 with the physical result shown in the right, with the LDR fixed in place and the wiring tidied up to make it more aesthetically pleasing.

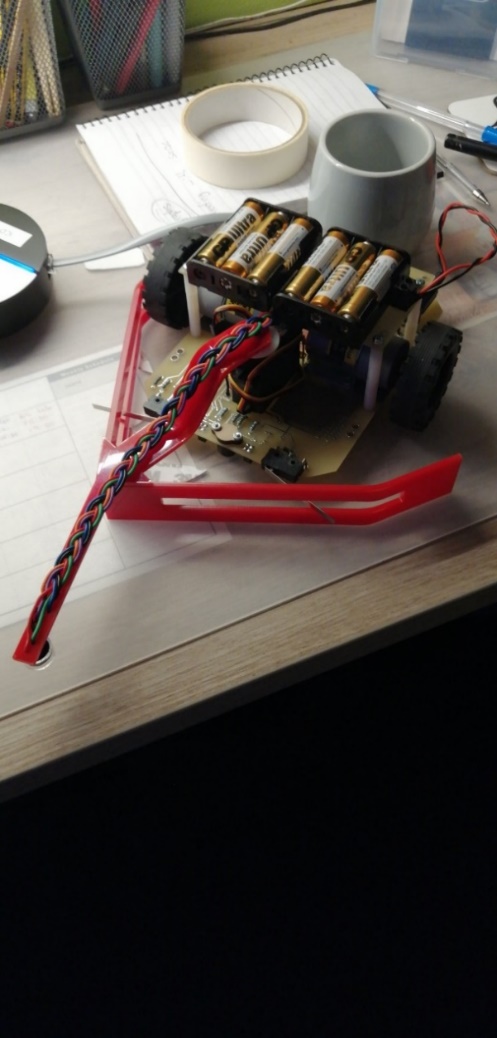
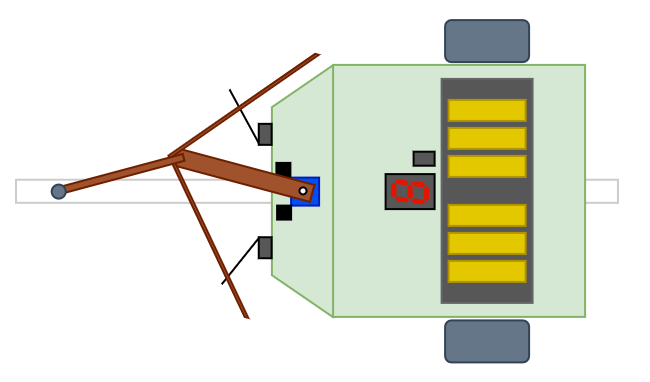


Figure 10 –Diagram showing the new design for the LDR arm extending in front of the plough (left), and a picture showing the outcome of the new LDR arm design on the Derbot (right).

## Programming

Once the hardware side of the build was complete, it was now time for the software side. The object detection and removal aspect of the system needed one peripheral which was set up in the MCC, the Analog to Digital Converter (ADC). This was set up to use the external clock source on the Derbot, be left justified and have its input set to AN3 with the rest of the setting set to their default values. This is summarised in figure 11 below.

First, a short function had to be created to run the ADC, put the result into a register and save the value of the register into a variable declared previously. This function was named ‘adcFucntion’ and is shown in figure 12 below. When the function is called, it sets bit 1 of the ADCON0 register to 1, this starts the conversion process, it then holds the program in a while loop until this bit is set back to 0, automatically done once the ADC is complete with the 10-bit result being placed across two registers (ADRESL and ADRESH). As the ADC was set up to left justify the result, the first eight most significant bits (MSB) were stored in ADRESH with the final two bits in ADRESL. As the light contrast between a lit and unlit object would be large (lit objects giving an ADC result of 0-30 and unlit 30–1024), the ADC result value did not need to very precise, therefore only the first eight MSB are used, ignoring the final two in the second register. (Microchip, 2016)

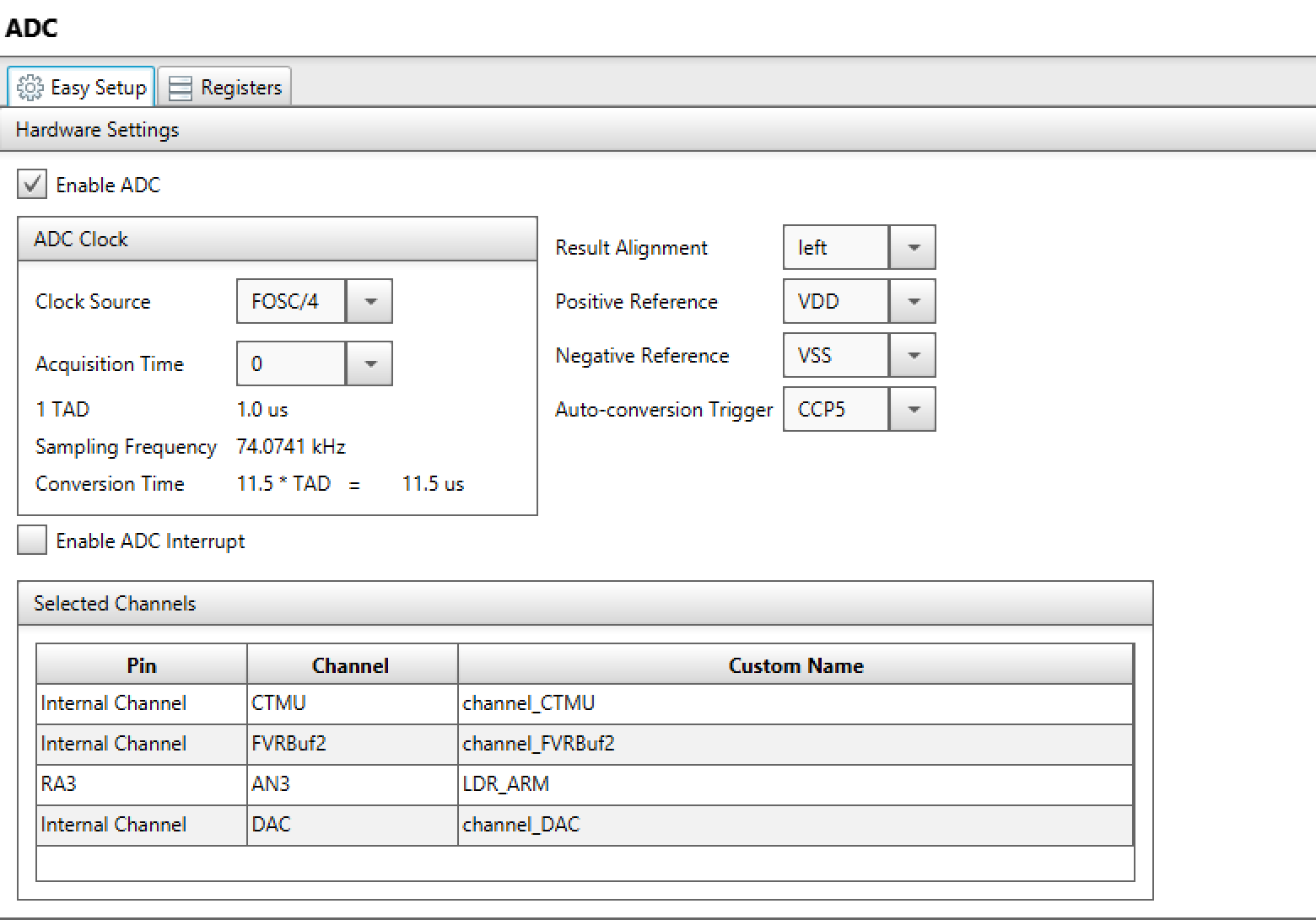
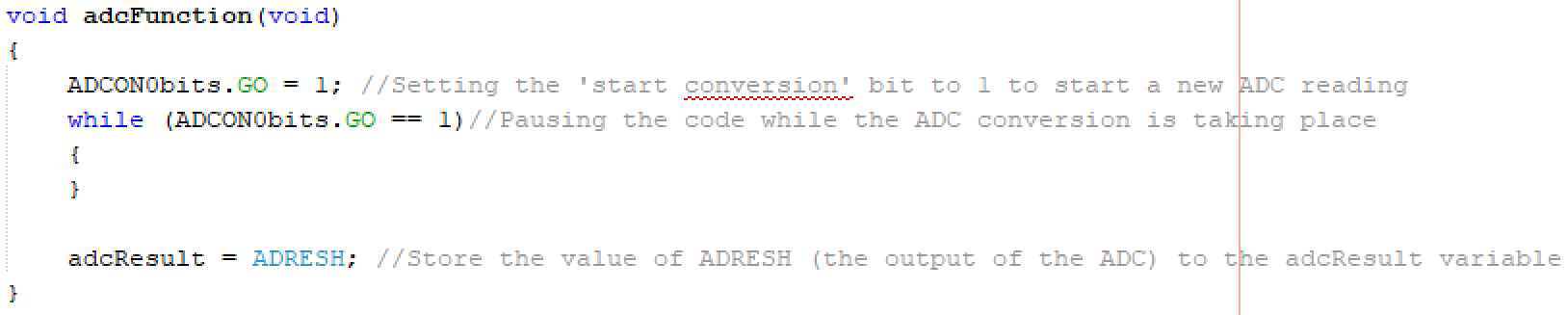


Figure 11 - Screenshot from the MCC in MPLABX showing the general settings used for the ADC.

Figure 12 - Screenshot of the ADC function used within the final code.

Next were the fucntions to move the servo between the left and right position. With the two ECCP peripherals being used for the PWM (Pulse Width Modulation) to control the motors, the easiest way to recreate a PWM signal to control the servo would be to use a timer and pulse a high and low value out of a digital pin. The servo data line was connected to port RB3 of the PIC which had previously been set at a digital output during the setup. From the servos datasheet (Electronicos Caldas, n.d.), the control signal should be sent with around a freqeuncy of 50Hz, this was kept in mind when writing the code for the two fucntions shown in figure 13.

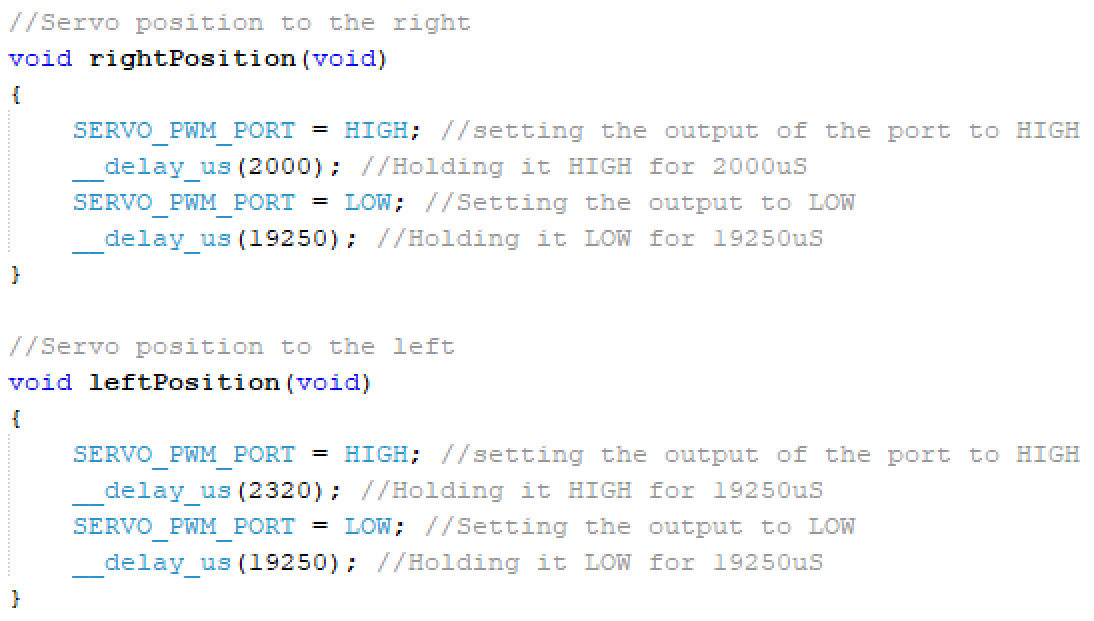


Figure 13 - Screenshot of the two servo functions created within the final code.

The frequency output for the right position function is around 47Hz and the left position function at around 46Hz as shown through the calculations below.

Once the functions and variables were created, the main block of code which would be in the infinite while loop was written. A flow chart was made to try and make this part of the code as efficient as possible to reduce the need for interrupts, this flow chart is shown in appendix 3 with the final code written from the flow chart shown in figure 14.

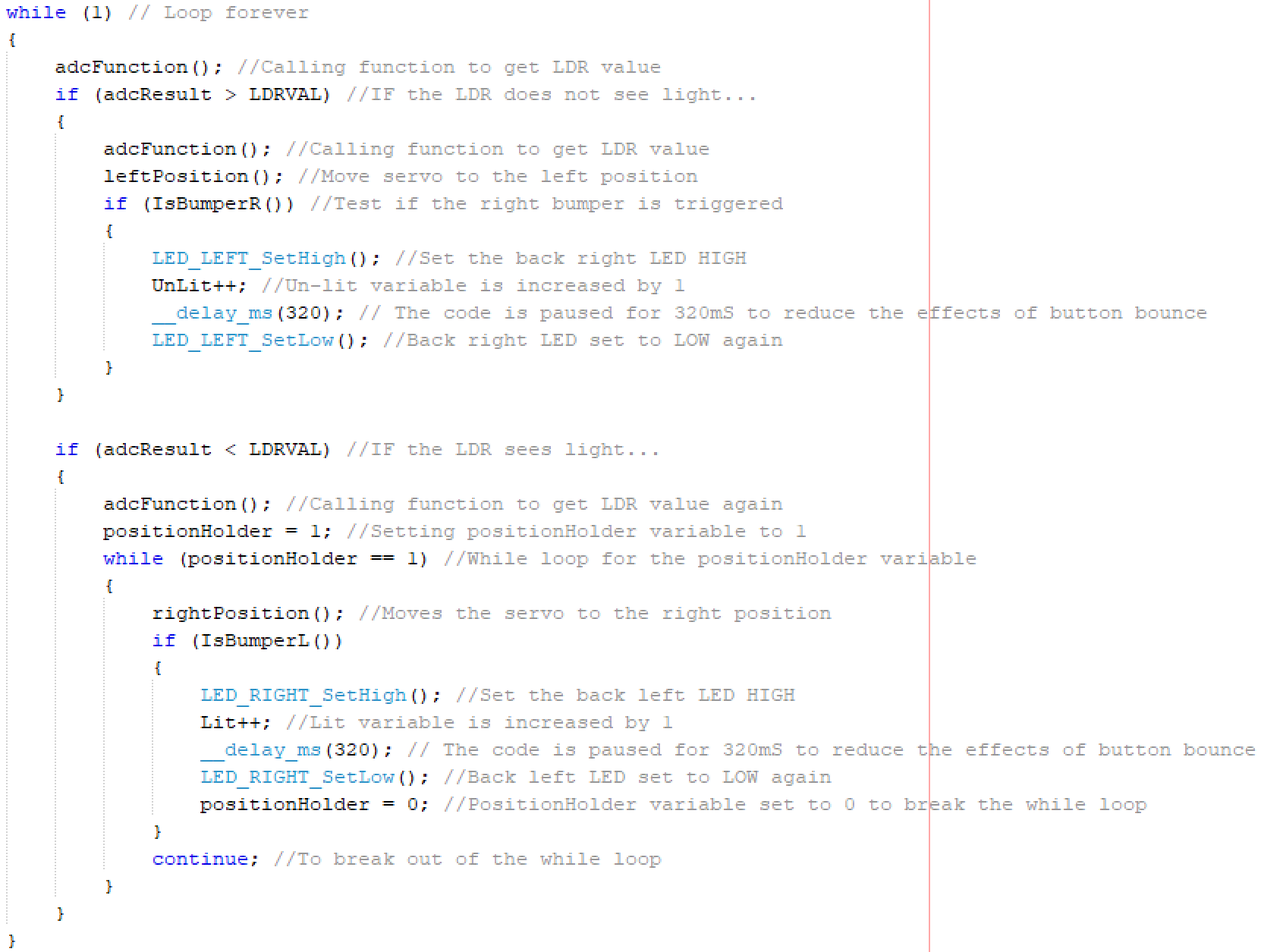


Figure 14 - Screenshot showing the code written based on the flow chart shown in appendix 3.

# I2C communication and display (OT)

## Hardware and circuit design

Within the I2C segment of the Derbot, the equipment used was relatively minimal. To begin with, an ICD4 was used in order to transfer data from the computer to the development board. The development board was then connected to a breadboard in the required locations (see Figure 18). The development board contained the PIC18F25K22 microcontroller as well as easy access ports to connect wire to the board.

Inter integrated circuit protocol, or I2C, allows for the communication between multiple peripheral (slave) chips to one or more controller (master) chips. So, in the case of the Derbot, the MCP23008 I/O expander, communicating with the PIC18F25K22. The SDA (data) link and the SCL (clock) link connect to pin 15 and 14 on the PIC microcontroller respectively. The SDA and SCL are held high by default by pull up resistors connected to power. The I2C requires a start and stop bit and all devices require an address because the 2 lines/wires are connected to every device.

Figure 15 shows an I2C data packet. The start bit is made by putting SDA low while SCL is high. Additional start bits can be added to change the mode from read or write. The required slave address is then sent following whether it is read or write. After Master actions, an acknowledgement (ACK) is sent back from the slave that is being communicated to, in order to verify communication of the right address. The data is then sent from the slave followed by an additional acknowledgement bit. The data packet then ends by SDA going from low to high whilst SCL is high (Learn.Digilentinc | Expanding I/O with the I/O Expander, 2020).

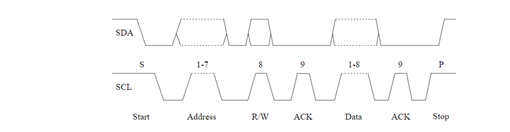


Figure 15 - an image showing the actions of the serial data line (SDA) and the serial clock (SCL) as they send a single packet of data. (Learn.Digilentinc | Expanding I/O with the I/O Expander, 2020)

The MCP23008 is an I/O expander used to add additional peripherals to a microcontroller. From figure 16 the pin outputs can be viewed. For the Derbot, this chip was used to add additional inputs/outputs using the 8 bidirectional GP (general purpose) pins to allow for the use of the seven-segment display.

For the purpose of the I2C only 2 registers were notably important, however a few more were used in order to initialise the MCP230088. These two notable registers being GPIO and IODIR. The GPIO (general purpose input/output port) register reflects the value on the port. The IODIR register or I/O direction, controls the direction of the data from the pins, setting them to either an input or an output. The MCP23008 read and write addresses had to be allocated as well. This was done using the byte seen in figure 16. The first four highest Bits are pre-determined. The lowest four bits are defined based on the wiring of the hardware address pins, A0, A1 ,A2 on the I/O expander, along with whether the address is for read or write.

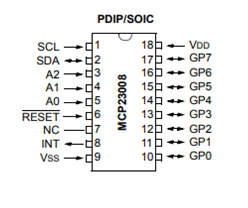
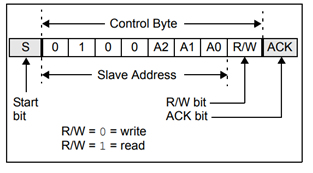


Figure 16 - an image showing the control byte used for the read and write slave address allocation (Left). a schematic showing the pin layout for the MCP23008 I/O expander (right). (microchip, 2020)

Before anything was carried out on Multisim/Ultiboard, a rough breadboard circuit was created to ensure that the skeleton code given would work successfully with the circuit created. This verified that not only all components were working successfully, and the connections were correct, but it also made designing the schematic of the PCB far easier.

Figure 18 shows the circuit created to get the I2C/IO expander working effectively. Seven resistors were used to connect the general-purpose pins of the MCP23008 to the pins on the seven-segment display which corelate to the appropriate segments within the display (see figure 17). Additionally, the display was connected to ground.

The power, ground, SCL and SDA pins on the MCP23008 were connected to the development board as seen in figure 18. Pins A0, A1, A2 and reset were all connected to ground. The reset occurs when the chip is pulled low so therefore the reset must be kept high since it does need to occur. In this instance the reset is inverted therefore connecting it to ground is the equivalent of connecting a non-inverted reset to power. The required pull up resistors were also put into place connecting the SDA line and the SCL line to power (Learn.Digilentinc | Expanding I/O with the I/O Expander, 2020).

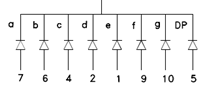


Figure 17 - a diagram showing the pin connections on the seven-segment display and which segment of the display they correspond to. Left a diagram showing the locations of the both the segments within the display and the pin layout. Right

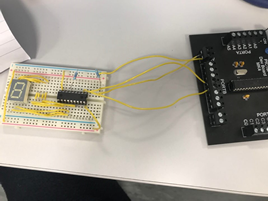
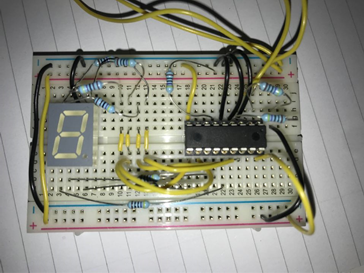


Figure 18 - an image showing the connection made on the breadboard for the seven-segment display and I/O expander (left). (right) an image showing breadboard to development board connections.

Figure 18 shows the connections made between the development board and the bread board. These connections are also seen between the PCB and the Derbot on the final build. The connections made here were the power (5v) and ground, SDA (data) which connected from port c pin 4 to pin 2 on the MCP23008 and the SCL (clock) which connected from port C pin 3 to pin 1 on the MCP23008.

Fundamentally the circuit for Multisim was copied from that which was created on the breadboard. On the multi sim design however, the interrupt was connected to the header J1. This was done in case interrupts were required down the line at some point by other members of the team.

Below shows figure 19. Both circuits are identical besides one-minute detail which has been circled on both figures. Figure 19 left was the original circuit design, the SCL and SDA connections were made after the pull up resisters. This mistake, however, was only noticed when testing was carried out on the PCB. To fix the problem the connections for the SCL and SDA were made before the resistors as seen in figure 19 right.

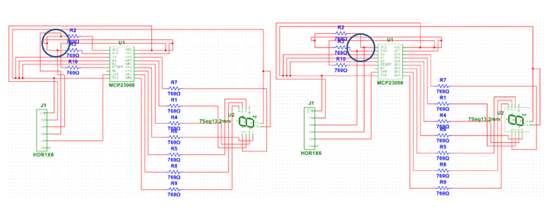


Figure 19 - A circuit design built on Multisim showing the proposed and built PCB without the SDA and SCL lines fixed and taken before the pull up resistors (left). a circuit design showing the corrected SCL and SDA line placement (right).

The issue from both SDA and SCL lines combining into one signal made there uses obsolete. Fixing this issue seen in figure 19 right resulted in the ultiboard schematic: Figure 20 left, which shows clearly the two green copper tracks of the SDA and SCL lines independent from one another.

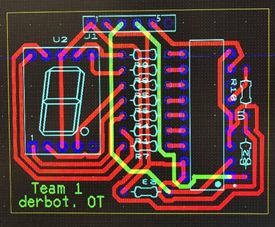


Figure 20 - a schematic showing the altered Multisim schematic within Ultiboard. The separation of the SDA and SCL line is Cleary visible here.(left) an image showing the printed circuit board based on the original design schematic. A double a batter has been included within the image to show scale. (right).

Figure 20 right shows the printed circuit board before any issues were discovered or alterations were made. It was at this point of the design when the combined signals from SDA and SCL were noticed, however, due to time constraints the fixed Ultiboard design in figure 20 left could not be printed and therefore manual hardware changes had to be made to the PCB. Adjusting the board was relatively simple. 2 circuit breaks were made which isolated the SCL and SDA holes and a wire was soldered from each of these holes to the required location to the MCP23008 on the underside of the PCB. Originally the PCB was to be placed on top of the power pack for the Derbot. After scaling down the PCB to such a small-scale, it was decided that the PCB would be better off sitting at the front of the Derbot by the motors.

## Programming

When creating the code for the I2C, a backbone code was used in order to set foundations. This code was adapted greatly to suit the needs of team 1s specific Derbot. The code started by setting the baud rate of the I2C. This is the speed at which information is transferred within a channel of communication (Phongchit, 2020). The rate set of 100kHz is suggested via the MCP23008 datasheet.

The characters used for the seven-segment display had to then be defined. The seven-segment display does in fact have an additional part of the display. That being a decimal point. This means there are 8 total LEDs on the display. In order to create numbers/letters on the display, the required LEDs needed to be lit. A hexadecimal number is used within the code to express these letters/numbers, however in order to find these hex numbers, the binary equivalent was worked out first. With 1 in binary lighting the led and 0 keeping it off, each bit of the 8-bit binary number represented each of the 8 segments.

The binary order is based on the connections made from the MCP23008 and the display. So, the first pin (pin 10) out of all the general-purpose pins on the MCP23008 was connected to pin 7 on the display which in turn correlated to segment a on the display (see figure 17). The order on the GP pins correlates directly to the bits within the required binary number. This first GP pin represented the least significant bit and the last GP pin (17) represented the highest significant bit. The rest of the connections were made in the order seen in figure 17 right. Once the binary numbers were converted into hex they were added to their corresponding number/letter on the code and defined.

An array was then used to store the list of characters used for the seven-segment display. Originally unlit and lit were first on the array followed by the numbers, however it seemed wiser to put them at the end since arrays start from 0 and our first number was 0 meaning the numbers would follow their position on the array. The registers and read and write addresses for the MCP23008 were then defined.

The function required for the I2C were initialised and consisted of multiple elements such as the starting and stopping of the I2C and the sending of data. Following this was the sub routines for each function. One very notable function was the I2C. The I2C sub routine was originally within the main of the code however this was changed for troubleshooting purposes. The function is called at the end of the Derbots run and displays U using the character array, followed by a wait, then the Unlit variable. This unlit variable is created based on the number of times the bumper switch is hit for unlit obstacles. This process is then repeated for the lit obstacles (see appendix 4 for flow chart).

# Testing and Merging codes (CH)

Once each sub-part of the system was working independently, they were merged into one file shown in appendix 5. It was then uploaded and the Derbot was sent around the test track for the first time, with the attitude that any problems would be fixed as they arose. The first run highlighted a few problems which were mainly down to the codes being merged incorrectly, such as the code becoming stuck inside loops with no way to break the conditions. To merge the codes more effectively, they were added bit by bit, making sure the current code worked as expected before adding the next few lines for each part. Once the codes were merged fully and worked, the timings had to be altered to get it to react to the different inputs happening all at once; something that was a priority as we were not using any interrupts.

While running the Derbot through some of the most extreme scenarios it could be up against in the competition, it was found that with a combination of its speed and the reaction time to the lit/unlit objects in its path caused a problem. When the LDR detected a lit object, the plough would move to the left position immediately, it would then hold this position until the right bumper switch had been triggered, before going back the default position and returning the LDR back over the white line. This process caused a problem when a lit object was directly followed by an unlit object at the minimum distance away (300mm) as the Derbot was moving too fast to be able to return the plough back to the default position in time to detect the next object. This problem was partially fixed; the Derbot was slowed down slightly and the delay between the bumper being triggered and the plough moving back was shortened. This fixed the problem for most cases however not all of them with factors such as reduced battery power and different object combinations changing the speed of the delays and movement.

The objects had a mirrored section in the middle which surrounded the LED light source inside them. When the room lights were on, the ambient light would reflect off this mirror and onto the LDR meaning unlit objects were seen as lit objects. To solve this problem, a small shade was added over the LDR as shown in figure 21. This stopped the ambient light hitting the objects as the LDR took the reading, reducing the likelihood of a false positive reading.

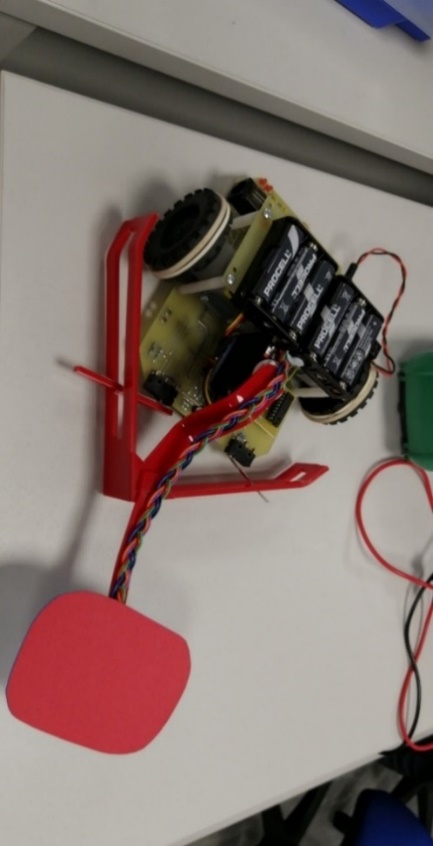


Figure 21 - Picture showing the addiction of a shade at the end of the LDR arm to reduce the ambient light hitting the LDR.

The I2C section was also verified with the PCB connected to the board. The left and right bumpers were triggered to simulate lit and unlit objects passing, with the display working as expected.

After some further minor tweaking to the code and hardware, such as the speed the Derbot went around the corners, stopping in the finish zone and adding an elastic band around each wheel to give it more grip to the track, it was ready for the competition.

# Performance in the competition

## White line following (MC)

The derbot did a great job with regards to following the white line on the competition track. The batteries were somewhat depleted as it entered the track so it did not move as fast as was intended. However, it managed to move almost all of the obstacles presented on the track. The derbot followed the curves very smoothly with no wobbling. The only issue was in the end, the white box was ignored, and points were lost due to inaccurate stopping.

## Obstacle detection and removal (CH)

On the competition track, the obstacle detection and removal aspect of the Derbot worked correctly for the first two sections of the course. With the Derbots speed and reaction times tuned correctly during the practice runs, the first five objects were moved and sorted correctly. However, for the final section, the last three objects were unfortunately set up in the similar scenario mentioned in the testing section of this report, with the three lights being a in an unlit-lit-unlit combination, all at the 300mm spacing. The plough could not move fast enough to keep up with the objects, resulting in the final unlit object being wrongly moved to the right side of the track and not being counted at all.

This was due to the same problems discovered in testing: battery life, the speed the Derbot and the object combination all being factors. However, after looking back at the code after the competition, there was a small problem with when new ADC results were being created. Once the ADC gave an initial result, if the value indicated no lit objects, the code would enter an IF loop which would not regenerate a new ADC result until one of the front opto sensors had been triggered. This has a simple solution by adding a line of code to call the ADC function while in the IF loop to recheck the ADC result, however, in the competition this could have increased the Derbots reaction time, adding to the factors causing the difficulty with the final object.

## I2C communication and display (OT)

The I2C components and software worked coherently with the other areas of the Derbot. This in turn resulted in all aspects working and all obstacle counts being displayed correctly. This success is possibly down to the extensive testing carried out to ensure the reliability of the display and communication.

# Conclusion (OT)

In conclusion the build was a success. The Derbot completed the course in 26.37 seconds, remaining on the white line throughout, correctly detecting and moving 7 of the 8 obstacles but did not stop accurately in the box. The intricate plough like hardware allowed the Derbot to move the obstacles without slowing and achieve a fast time and fit through the starting gate. The white line following worked seamlessly. However, the Derbot did not stop in the box due the far right opto not sensing the correct line. 1 of the obstacles were not detected due to the positioning of the LDR, which was not over the white line on the corners. This was a compromise in the design between speed and detection, keeping the LDR at a distance enabled greater speed but reduced detection capability. The number of lit and unlit obstacles moved was accurately displayed on the external display.

# References

Electronicos Caldas, n.d. MG996R High Torque Metal Gear Dual Ball Bearing Servo. [Online]   
Available at: https://www.electronicoscaldas.com/datasheet/MG996R\_Tower-Pro.pdf  
[Accessed 13th December 2020].

Microchip, 2016. PIC18(L)F2X/4XK22 Datasheet. [Online]   
Available at: https://www.microchip.com/stellent/groups/picmicro\_sg/documents/devicedoc/cn547043.pdf  
[Accessed 13th December 2020].

Electronics, T. (2020). Reflective Object Sensor - OPB608A. www.ttelectronics.com

Wilson, D. (2020). Embedded Systems Assignment 1 : Transducers (pp. 1–4). University of Derby.

Learn.sparkfun.com. 2020. I2C - Learn.Sparkfun.Com. [online] Available at: <https://learn.sparkfun.com/tutorials/i2c/all> [Accessed 16 December 2020].

Learn.digilentinc.com. 2020. Learn.Digilentinc | Expanding I/O With The I/O E5xpander. [online] Available at: <https://learn.digilentinc.com/Documents/302> [Accessed 16 December 2020].

Ww1.microchip.com. 2020. [online] Available at: <https://ww1.microchip.com/downloads/en/DeviceDoc/MCP23008-MCP23S08-Data-Sheet-20001919F.pdf> [Accessed 16 December 2020].

Kingbright.com. 2020. Kingbright. [online] Available at: <http://www.kingbright.com/attachments/file/psearch/000/00/20160808bak/SC52-11SURKWA(Ver.10A).pdf> [Accessed 16 December 2020].

Phongchit, N., 2020. What Is Baud Rate & Why Is It Important?. [online] Setra.com. Available at: <https://www.setra.com/blog/what-is-baud-rate-and-what-cable-length-is-required-1> [Accessed 16 December 2020].

Silonex. (2018). *NORPS-12*. <https://docs.rs-online.com/4184/0900766b815e8a2a.pdf>

# Appendices

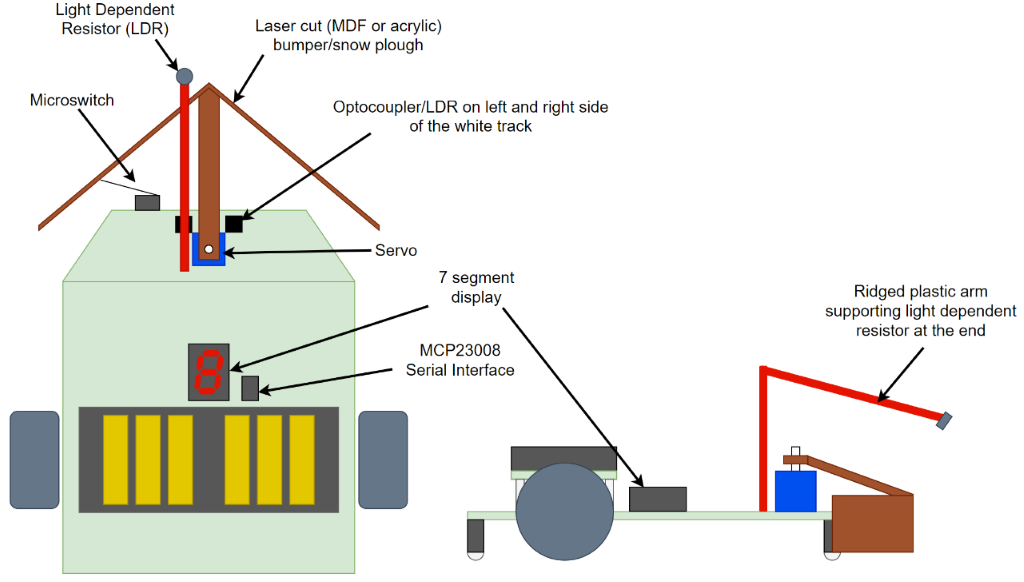
## Appendix 1 – Team plan

Hardware strategy

From the challenge brief, the final Derbot design needs to be able to:

* Follow a white line around the course and stop in the finish zone
* Be small enough to fit through a gate way at the start
* Detect if the object in front of it is lit or unlit
* Move the object to the left if unlit and to the right if lit
* Display the number of lit and unlit object there were around the course on a seven-segment display once inside the finish zone

Figure 1 shows a diagram of the proposed Derbot design. This design should meet all the needed criteria mentioned above.



*Figure 1 - Diagram showing the plan view (left) and the side elevation (right) of the Derbot design.*

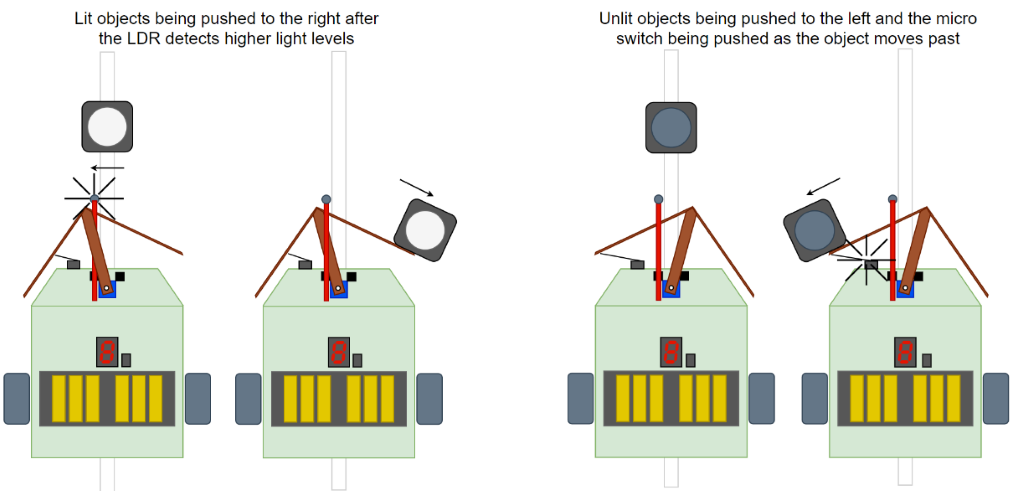
The two optocouplers (or LDRs) will be used to keep the Derbot on the track, allowing the code to adjust the two rear motors depending on their values.

It will have an arrow shaped bumper to push the objects to the left or right as it moves forward; an arm extending from the bumper will be connected to a servo allowing it to be moved left and right.

 The default position of the bumper will be in the right position thus pushing everything to the left unless, the Light Dependent Resistor (LDR) at the end of an extended arm detects high level of light from the object, where the servo will move the bumper to the left position pushing the lit object to the right as the Derbot carries on moving forward along the line.

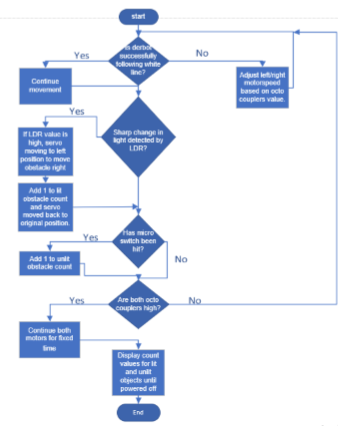
To detect the unlit objects being pushed to the left, the built in microswitch on the left of the Derbot will be connected to the bumper, the bumper will slightly flex when the object is push passed, clicking the microswitch and detecting the object. Both scenarios are shown below in figure 2. By setting the default position of the bumper to the left, the Derbot will not have to slow down when approaching each object, but only change the servo position when it detects high light levels.

There will also be a seven-segment display fitted to the Derbot to show the number of lit and unlit objects detected at the end of the course. This will be via I2C and a MCP23008 serial interface chip.



*Figure 2 - How the Derbot will react to a lit object (left) and how it will react to an unlit object (right).*

Programming Strategy

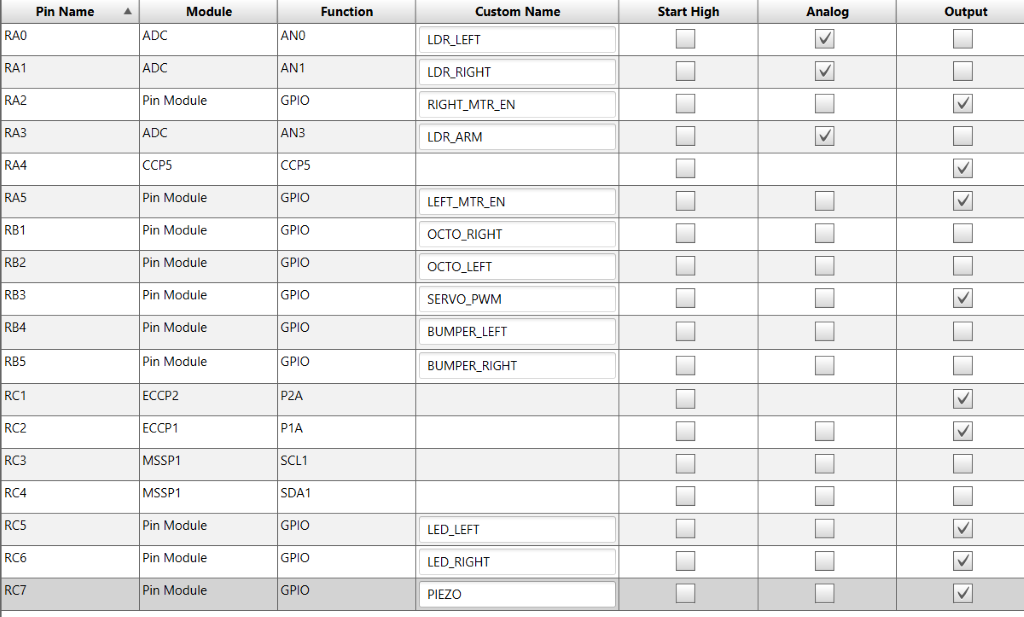




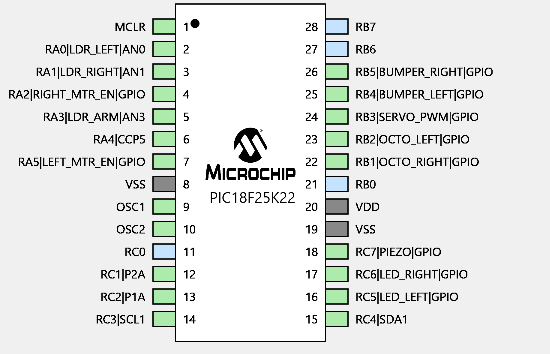
Pic resources

*Table 1 - Table showing the peripherals to be used from the PIC microcontroller, their functions, and their purpose on the Derbot.*

|  |  |  |
| --- | --- | --- |
| **Peripheral** | **Function** | **Use in Derbot** |
| ACD – Analog to Digital Converter | Convert analogue signals into digital values | Read analogue voltage from pins 2, 3 and 5 (the LDRs) |
| FVR – Forward Voltage Reference | Set the voltage reference for the ADC (1.024V, 2.048V or 4.096V) | Setting the voltage reference of the ADC to 1.024V. the ADC has a 10bit resolution which will result in a resolution of 1mV |
| ECCP1 – Enhanced capture compare module 1 | Enhanced PWM mode 1 | This will generate the PWM signal for the left motor of the DERBOT |
| ECCP2 – Enhanced capture compare module 2 | Enhanced PWM mode 2 | This will generate the PWM signal for the right motor of the DERBOT |
| TMR2 – internal Timer 2 of the PIC | This timer will be used for both PWM pulses for the left and right motor | The PWM values are sent into the IN pins of the L293D |
| TMR4 – internal Timer 4 of the PIC | This timer will be used to generate a PWM signal in-code (instead of using a CCP module) | The in-code-generated PWM signal will be used as the data for the SERVO |
| TMR6 – internal Timer 6 of the PIC | This timer will be used at the end of the program to determine the stopping time | When the ending white line is detected, the DERBOT will stop when TMR6 overflows (1s) |
| MSSP1 | I2C – PIC will be the MASTER | The PIC will be the Master |
| MSSP2 | I2C – external I/O will be the SLAVE | The I/O expansion will be the SLAVE |



*Figure 4 - Screenshot from MPLABX showing the pin names and custom names which will be used.*



*Figure 5 - Screenshot from MPLABX showing a graphical representation of the same data from figure 4.*

Allocation of work and time

Allocation of work

Charlie

* Object detection and moving

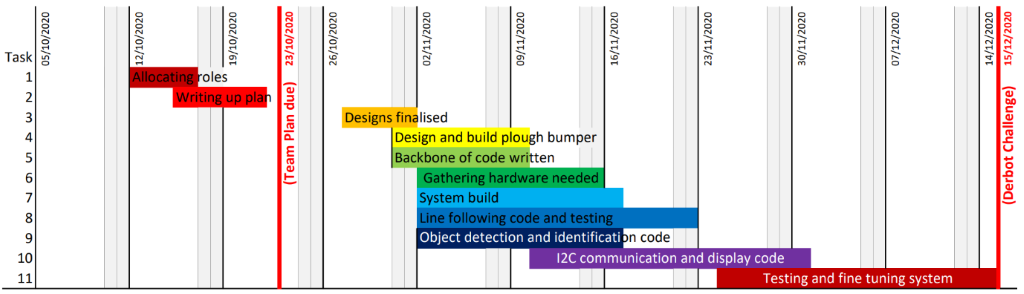
Oliver

* I2C communication and seven segment display

Mateo

* White line following/starting and stopping

Gannt chart

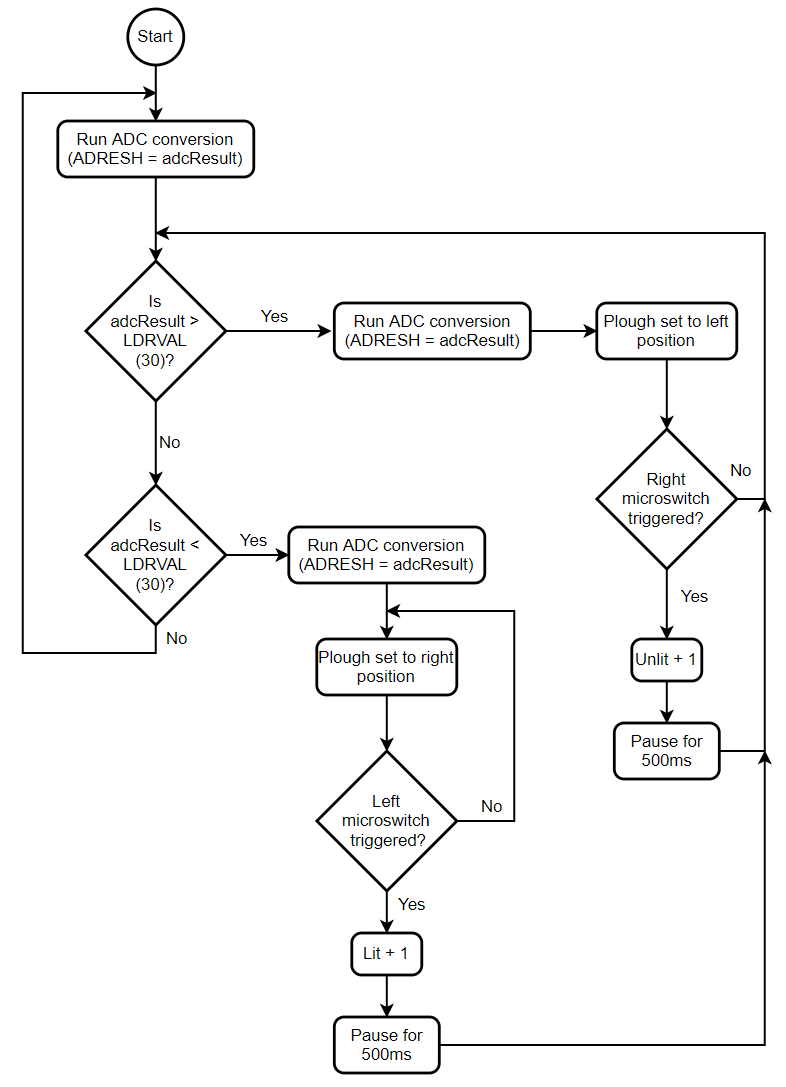


*Figure 6 - The Gannt chart for the design and build of the Derbot system.*

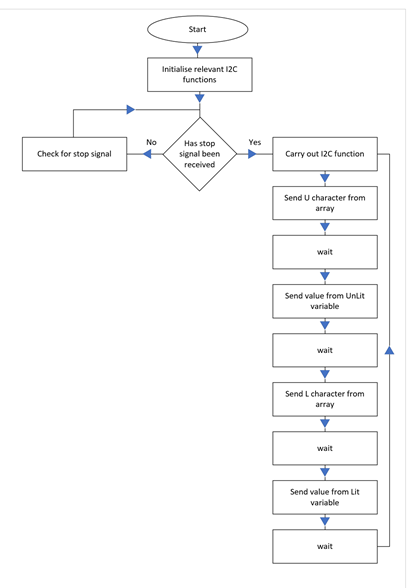
## Appendix 2 – While line following and stopping code flow chart



## Appendix 3 – Obstacle detection and removal code flow chart



## Appendix 4 – I2C communication flow chart



## Appendix 5 – Derbot full code

#include "mcc\_generated\_files/mcc.h"

#define I2C\_freq 100000 //this is teh baud rate and is the speed at which data travels through a channel of communication. This value is suggested by the MCP23008 datasheet.

//defining characters on the seven segment display with their hexadecimal counterparts which indicate which segemnts light up.

#define unlit 0x3e

#define lit 0x38

#define zero 0x3f

#define one 0x6

#define two 0x5b

#define three 0x4f

#define four 0x66

#define five 0x6d

#define six 0x7d

#define seven 0x7

#define eight 0x7f

#define nine 0x67

unsigned char digits[] = {zero, one, two, three, four, five, six, seven, eight, nine, unlit, lit}; //creating an array based on the characters for the seven segement display.

//defining addess locations and setting for the communication/I2C/MCP23008. many are pre set hex numbers on the datasheet.

#define mcp23008\_address\_write 0x40 //these addresses are changed base on the hardware address pins (a0,a1,a2) and whether the address is read or write

#define mcp23008\_address\_read 0x41

#define MCP\_IODIR 0x00//sets whether the pin is an input or an output. Hex value is 0 because all pins are outputs.

#define MCP\_GPIO 0x09//reflects the logic level on the pins

#define MCP\_IPOL 0x01//configures teh polarity of the GPIO port pins

#define MCP\_IOCON 0x05// contains many bits which configure the device, including the slew rate of the SDA pin, enabling/disabling the hardware address pins

#define MCP\_GPPU 0x06//this register controls the pull up resistors connected to the SDA and SCL pins. data sheet specifies this hex value.

#define MCP\_OLAT 0x0A//controls access to output latches

//these set of registers are all to do with interupts, not being used for team 1s Derbot

#define MCP\_DEFVAL 0x03

#define MCP\_INTCON 0x04

#define MCP\_INTF 0x07

#define MCP\_INTCAP 0x08

#define MCP\_GPINTEN 0x02

//Declairng and intialising funtions

void diagnostic(void);

void motor\_fwd(unsigned char dir, unsigned char spd, bool STAT);

void movement(void);

//initialsing functions for the communication/I2C/MCP23008/PIC

void PIC\_Initialise(void); //Pic

void I2C\_Initialise(void); //i2c

void MCP23008\_Initialise(void); //mcp

void IdleI2C(void); //Idlei2c

void StartI2C(void); //start i2c

void I2C\_send\_data(unsigned char device\_address, unsigned char register\_address, unsigned char register\_data); //i2c send data

void WriteI2C(unsigned char data); //writei2c

void StopI2C(void); //stopi2c

void I2C(void); //this is the final I2C function called within the main

void rightPosition(void); //Declaring rightPosition, this function is used to move the servo to the right position

void leftPosition(void); //Declaring leftPosition, this function is used to move the servo to the left position

void adcFunction(void); //Declaring adcFucntion, this function will return the value from the ADC on pin AN3

int adcResult; //Integer variable to hold the value of the ADC result from the adcFucntion()

int UnLit = 0; //Integer variable to hold the number of un-lit objects detected

int Lit = 0; //Integer variable to hold the number of lit objects detected

int LDRVAL = 40; //Integer variable which will compared to the ADC result to determine if a lit object is in front or not

int positionHolder = 0; //Integer variable used to escape the IF loop used

bool IsBumperL(void); //Declaring IsBumperL, this function returns true if the left bumper is triggered

bool IsBumperR(void); //Declaring IsBumperR, this function returns true if the right bumper is triggered

bool IsOptoL(void); //left opto triggered

bool IsOptoR(void); //right opto triggered

bool IsOptoFarR(void); //far right opto triggered

void main(void)

{

SYSTEM\_Initialize(); // Initialize the device - generated by MCC

I2C\_Initialise(); //initialising communication

MCP23008\_Initialise(); //initialising the MCP23008 I/O expander

// diagnostic();

while (1) // Loop forever

{

diagnostic();

motor\_fwd('L', 255, HIGH); //left motor full speed

motor\_fwd('R', 250, HIGH); //right motor 90% speed

// initial motor conditions

adcFunction(); //Calling function to get LDR value

if (adcResult > LDRVAL) //IF the LDR does not see light...

{

adcFunction(); //Calling function to get LDR value

leftPosition(); //Move servo to the left position

if (IsOptoR()) // is right opto triggered

{ // 0 for opto-coupler is dark and 1 is light

motor\_fwd('L', 250, HIGH); //left motor full speed

motor\_fwd('R', 105, HIGH); //reduce right motor speed

}

else if (IsOptoL()) //is left opto triggered

{ // 0 for opto-coupler is dark and 1 is light

motor\_fwd('L', 105, HIGH); //reduce left motor speed

motor\_fwd('R', 250, HIGH); //right motor full speed

}

else if (IsOptoFarR()) //is far right opto triggered

{

\_\_delay\_ms(450); //wait for derbot to reach centre of square

motor\_fwd('L', 255, LOW); //set left motor off

motor\_fwd('R', 255, LOW); //set right motor off

I2C(); //calling the I2C function to display the characters whent the derbot sends the stop signal

break;

}

if (IsBumperR()) //Test if the right bumper is triggered

{

LED\_LEFT\_SetHigh(); //Set the back right LED HIGH

UnLit++; //Un-lit variable is increased by 1

\_\_delay\_ms(320); // The code is paused for 320mS to reduce the effects of button bounce

LED\_LEFT\_SetLow(); //Back right LED set to LOW again

}

}

if (adcResult < LDRVAL) //IF the LDR sees light...

{

adcFunction(); //Calling function to get LDR value again

positionHolder = 1; //Setting positionHolder variable to 1

while (positionHolder == 1) //While loop for the positionHolder variable

{

if (IsOptoR()) // is right opto triggered

{ // 0 for opto-coupler is dark and 1 is light

motor\_fwd('L', 250, HIGH); //left motor full speed

motor\_fwd('R', 105, HIGH); //reduce right motor speed

}

else if (IsOptoL()) //is left opto triggered

{ // 0 for opto-coupler is dark and 1 is light

motor\_fwd('L', 105, HIGH); //reduce left motor speed

motor\_fwd('R', 250, HIGH); //right motor full speed

}

else if (IsOptoFarR()) //is far right opto triggered

{

\_\_delay\_ms(450); //wait for derbot to reach centre of square

motor\_fwd('L', 255, LOW); //set left motor off

motor\_fwd('R', 255, LOW); //set right motor off

I2C(); //calling the I2C function to display the characters whent the derbot sends the stop signal

break;

}

rightPosition(); //Moves the servo to the right position

if (IsBumperL())

{

LED\_RIGHT\_SetHigh(); //Set the back left LED HIGH

Lit++; //Lit variable is increased by 1

\_\_delay\_ms(320); // The code is paused for 320mS to reduce the effects of button bounce

LED\_RIGHT\_SetLow(); //Back left LED set to LOW again

positionHolder = 0; //PositionHolder variable set to 0 to break the while loop

}

continue; //To break out of the while loop

}

}

}

}

void diagnostic(void) // Toggles leds for 0.3s each (Tcy = 1us)

{

LED\_LEFT\_Toggle(); // PORTCbits.RC5 = 0

LED\_RIGHT\_Toggle(); // PORTCbits.RC6 = 1

\_\_delay\_ms(300); // 0.3 second delay

LED\_LEFT\_Toggle(); // PORTCbits.RC5 = 1

LED\_RIGHT\_Toggle(); // PORTCbits.RC6 = 0

}

void motor\_fwd(unsigned char dir, unsigned char spd, bool STAT)

{

if (dir == 'L') //left motor selected

{

CCPR2L = spd; //left motor speed

LEFT\_MTR\_EN\_PORT = STAT; // HIGH or LOW for left motor

}

else //left motor selected

{

CCPR1L = spd; // right motor speed

RIGHT\_MTR\_EN\_PORT = STAT; // HIGH or LOW for right motor

}

}

void I2C\_Initialise()//setting special function registers to start up the I2C

{

SSP1STAT = 0x00; //serial synchronous port status register

SSP1CON1 = 0x28; //serial synchronous port control register1

SSP1CON2 = 0x00; //serial synchronous port control register2

SSP1CON3 = 0x08; //serial synchronous port control register3

SSP1ADD = 0x09; //serial synchronous port address and baud rate register

}

void MCP23008\_Initialise(void)//creating the function for starting up the MCP23008

{

I2C\_send\_data(mcp23008\_address\_write, MCP\_IODIR, 0x00);

IdleI2C();

I2C\_send\_data(mcp23008\_address\_write, MCP\_IPOL, 0x00);

IdleI2C();

I2C\_send\_data(mcp23008\_address\_write, MCP\_GPINTEN, 0x00);

IdleI2C();

I2C\_send\_data(mcp23008\_address\_write, MCP\_IOCON, 0x3E);

IdleI2C();

I2C\_send\_data(mcp23008\_address\_write, MCP\_GPPU, 0x00);

}

void I2C\_send\_data(unsigned char device\_address, unsigned char register\_address, unsigned char register\_data)//configuring the settings required for sending messages through the I2C. this function uses many of the other functions created.

{

//communication package, inlcluding start and stop bits, address checks and data transfer

StartI2C();

WriteI2C(device\_address);

IdleI2C();

WriteI2C(register\_address);

IdleI2C();

WriteI2C(register\_data);

IdleI2C();

StopI2C();

}

void IdleI2C(void)//keeps the communication idle while 1 of 2 special function registers are active with specific settings

{

while ((SSP1CON2 & 0x1F) || (SSP1STAT & 0x02));

}

void StartI2C(void)//creating the start bits required for communication packet

{

PIR1bits.SSPIF = 0; //changing specific bits whithin the special function registers

SSP1CON2bits.SEN = 1; //enabling the start condition bit

while ((SSP1STAT & 0x04) || (SSP1CON2 & 0x1F)); //

// while (!PIR1bits.SSPIF);

PIR1bits.SSPIF = 0;

}

void WriteI2C(unsigned char data)//writing data

{

SSP1BUF = data; //receive buffer/transmit register

while (SSP1STATbits.BF);

}

void StopI2C()//creating the stop point of the package

{

SSP1CON2bits.PEN = 1; //eanbling the stop condition bit

}

void I2C(void)//creating the function called within the main when teh stop signal is sent and the character diplay can occur

{

// for(int x=0; x<12; x++)

{

I2C\_send\_data(mcp23008\_address\_write, MCP\_GPIO, digits[10]); // sending and displaying 10 of the array, U

\_\_delay\_ms(1000); //waitng to allow for time to see the display

I2C\_send\_data(mcp23008\_address\_write, MCP\_GPIO, digits[UnLit]); //sending and displaying variable unlit, this value is from the amount of times the bumpers have been hit on the unlit side

\_\_delay\_ms(1000);

I2C\_send\_data(mcp23008\_address\_write, MCP\_GPIO, digits[11]); // sending and displaying 11 of the array, L

\_\_delay\_ms(1000);

I2C\_send\_data(mcp23008\_address\_write, MCP\_GPIO, digits[Lit]);

\_\_delay\_ms(1000);

}

}

//Servo position to the right

void rightPosition(void)

{

SERVO\_PWM\_PORT = HIGH; //setting the output of the port to HIGH

\_\_delay\_us(2000); //Holding it HIGH for 2000uS

SERVO\_PWM\_PORT = LOW; //Setting the output to LOW

\_\_delay\_us(19250); //Holding it LOW for 19250uS

}

//Servo position to the left

void leftPosition(void)

{

SERVO\_PWM\_PORT = HIGH; //setting the output of the port to HIGH

\_\_delay\_us(2320); //Holding it HIGH for 19250uS

SERVO\_PWM\_PORT = LOW; //Setting the output to LOW

\_\_delay\_us(19250); //Holding it LOW for 19250uS

}

//Analoge to Digital Conversion (ADC) function

void adcFunction(void)

{

ADCON0bits.GO = 1; //Setting the 'start conversion' bit to 1 to start a new ADC reading

while (ADCON0bits.GO == 1)//Pausing the code while the ADC conversion is taking place

{

}

adcResult = ADRESH; //Store the value of ADRESH (the output of the ADC) to the adcResult variable

}

bool IsBumperL(void) // Return true when left switch is pressed

{

return (BUMPER\_LEFT\_GetValue() == 0);

}

bool IsBumperR(void) // Return true when right switch is pressed

{

return (BUMPER\_RIGHT\_GetValue() == 0);

}

bool IsOptoL(void) // return true when left opto is dark (dark = 0, light = 1)

{

return (OPTO\_LEFT\_GetValue() == 0);

}

bool IsOptoR(void) // return true when right opto is dark (dark = 0, light = 1)

{

return (OPTO\_RIGHT\_GetValue() == 0);

}

bool IsOptoFarR(void) // return true when right opto is dark (dark = 0, light = 1)

{

return (OPTO\_FAR\_RIGHT\_GetValue() == 0);

}