

Newcastle
University

Final Report

Automation of a Rail Buggy

MEC8095-Mechanical & Systems Engineering Project

Supervisor: Dr. Franklin

Name: Abdulrahman Mahmou

Student Number: 140362477

Declaration

I, Abdulrahman Mahmoud the author of this report, declare that this report is submitted as a requirement from the Mechanical Engineering School towards the end of the Mechatronics degree at Newcastle University, has not been submitted for any other degree at any other Universities. It is solely the work of Abdulrahman Mahmoud unless recognised otherwise in the text. It illustrates the work that was carried out at Newcastle University which is recorded in a project logbook. I am aware that plagiarism, fabricated results and the use of unacknowledged research, ideas or words will be dealt under the University's Assessment Irregularities Procedure and if needed subjected to a disciplinary action. Thus, I declare that this report is free from plagiarism, fabricated results and unacknowledged research.

Abdulrahman Mahmoud,

24/08/2018

Acknowledgements

I would like to direct my utmost gratitude for the assistance of my supervisor Dr. Francis Franklin for his guidance, vital care, encouragement, support and the recommendations made by him throughout the progression of the project. His continuous supervision upon my work was vital in the progression and completion of the project successfully. I would like to extend my gratitude for the support of my family members and friends particularly, Mohammad Al-Banna, Nicolas Abboud and Ajith Thomas for their help during the project.

Thank you,

Abdulrahman Mahmoud.

Abstract

Regular operation of trains will result in a progressive damage to railway tracks, even though the steel used is very durable and can withstand high loads. Through constant use the rails can develop small defects such as micro-cracks, gauge cornering cracking, rolling contact fatigue, wear etc. These small defects if not detected can grow and eventually cause a catastrophic failure if the rail breaks during operation.

The railway track buggy is a foldable lightweight vehicle that can be used to carry equipment, e.g. monitoring equipment to detect defects in the rail. Although the track buggy has motors, the control system is still being developed and the present report develops systems for feedback, and control using speed sensors at both driving and driven wheels.

The control system couples a Raspberry Pi with a Teensy, and a method of communication over I2C developed for the system. Photo-interrupter sensors were used as a way of measuring the speed of the track buggy on rail. An Arduino Leonardo board was also used to test the system.

System testing is shown, and the results are laid out to present the main strengths of the system whilst suggesting an improvement for the flaws. Results from both boards are shown and a conclusion is based on these results. Overall system implementation between all boards was a success, along with the speed feedback system.

Contents

1 – Introduction	5
1.1 - Aims and Objectives:	5
2 - Literature Review	5
2.1 - Raspberry Pi.....	5
2.2 - Arduino Shield	6
2.3 – Teensy 3.2.....	7
2.4 - Programming Languages Used	9
2.5 – Sharp GP1A57HRJ00F Photo Interrupter sensor	10
2.6 - I2C/IIC (Inter Integrated Circuit) Bus	10
3 – Mechanical Design	12
3.1 – Driven Wheel	12
3.2 – Driving Wheel	14
4 – System	15
4.1 – Raspberry Pi Code (Python).....	15
4.2 – Arduino/Teensy Code (C++)	17
5 - System Test	20
5.1 – Sensors	20
5.1.1 – Driven Wheels.....	21
5.1.2 – Driving Wheel.....	23
5.2 – RPi – Teensy/Arduino Communication	24
6 – Discussion	27
7 – Conclusion	28
8 – Future Work	29
9 - References.....	31
Appendix A – Driven Wheel Components Engineering drawings	33
Appendix B – Driving Wheel Components Engineering drawings	37
Appendix C – Raspberry Pi code (Python).....	40
Appendix D – Teensy code (C++)	41
Appendix E – Arduino code (C++).....	44

1 – Introduction

According to the Office of Rail and Road, there has been a total of 438 million passenger journeys made in Britain by a train operator in the 3rd quarter of the financial year 2018 [1]. Due to this large volume of journeys the rail must be maintained regularly and scanned for any defects that might cause disastrous events.

To inspect the rail, operators use various technologies and vehicles including hand-pushed buggies. These buggies are loaded with monitoring equipment and the operator manually pushes the buggy across the rail. The Track Buggy, being developed at Newcastle University, is a foldable, lightweight, self-powered railway vehicle that can be used to carry equipment e.g., monitoring equipment.

The Track Buggy is equipped with 2 MY1016 24V, 250W electric scooter motor. The driving wheels are not directly driven using the motor, instead it uses an 11-tooth sprocket to a 25-tooth sprocket that sits on an intermediate shaft. This intermediate shaft contains a 28-tooth sprocket that drives a 42-tooth sprocket located on the driving wheel. Chains are used to transmit the torque from the motors to the wheels. This allows the Buggy to reach speeds up to 15km/h [2].

The Track Buggy needs an integrated system that will allow it to operate automatically on the track. The system will have a speed feedback system to control the buggy's velocity on the track. The system can be controlled wirelessly.

1.1 - Aims and Objectives:

- Select optimum solution for control.
- Speed monitoring system.
- Establish an I2C¹ bus between the Pi and the other boards for non-serial communications.
- Build the system and test it.

2 - Literature Review

2.1 - Raspberry Pi

The Raspberry Pi is a cheap single board computer that can be used easily with a keyboard, mouse, a monitor and the free operating system Raspbarian. The small

¹ I2C: Inter Integrated Circuit communication protocol

computer supports multiple programming languages all aimed for programmers to learn a new program or for experienced users, to be use in any project.

Raspberry Pi Model 3 is equipped with 1GB of RAM and a Quad Core 1.2GHz 64-bit CPU. It has built in Wi-Fi and Bluetooth capabilities with a 40-pin extended GPIO (General-Purpose Input/Output) that can be used for I2C, SPI² or UART³ communication protocols, allowing the Pi to be versatile in communication. The Pi also has some limitations such as, no built in ADC (Analogue to Digital Converter) and cannot produce enough power to drive an inductive load, making it hard for the Pi to control hardware components. However, these disadvantages can be overcome by using a Teensy board or an Arduino Shield [3].

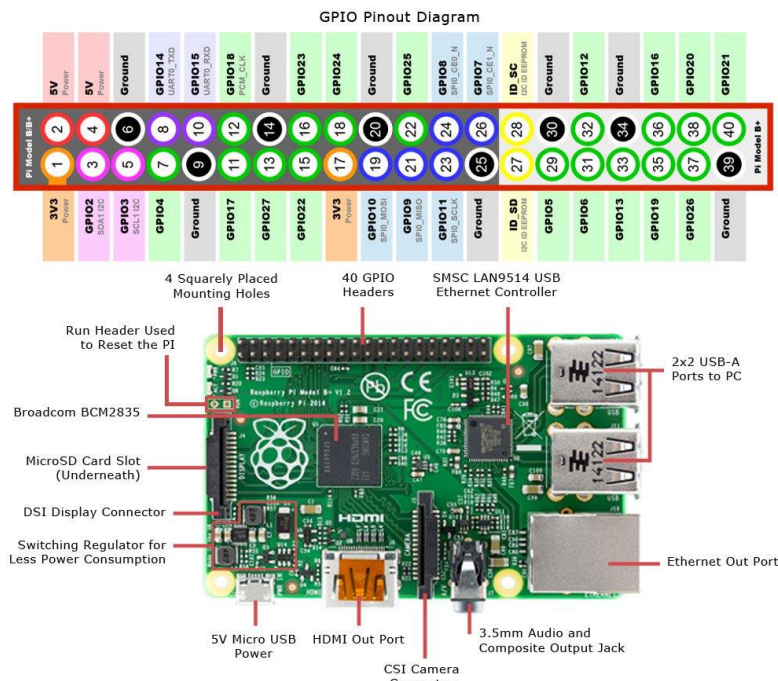


Figure 1: Raspberry Pi with pinout map [4]

2.2 - Arduino Shield

The Arduino is an open-source electronics platform based on easy-to-use hardware and software [5]. Generally, an Arduino can be regarded as a microcontroller with its I/Os, memory and CPU all integrated on one board. It uses an IDE (Integrated Development Environment) that can be downloaded on a computer where the

² SPI: Serial Peripheral Interface communication protocol

³ UART: Universal Asynchronous Receiver/Transmitter communication protocol

programmer can write and upload the code to the board with the use of a USB cable [5]. This IDE uses a simpler version of C++ programming language, making it easier to program the board [6]. This IDE can be downloaded on the Pi and then the codes can be uploaded to the Arduino using the Pi.

The Arduino offers different variety of pins such as, Ground, 5V & 3.3V, Analogue, Digital and PWM [6]. The board also supports the same communication protocols offered by the Raspberry Pi. To keep the microcontroller cost to a minimum, manufacturers did not include built in Wi-Fi or Bluetooth. However, it is good in controlling hardware but inadequate for communications. Thus, by combining a Raspberry Pi's excellent communication and an Arduino's ability to control hardware and run them, the result is a complete system.

The Arduino shield in use is based on the Arduino Leonardo Chip with its on board microcontroller chip being an ATmega32u4 with a clock speed of 16 MHz [7]. The shield also offers different Raspberry Pi I/O pins.

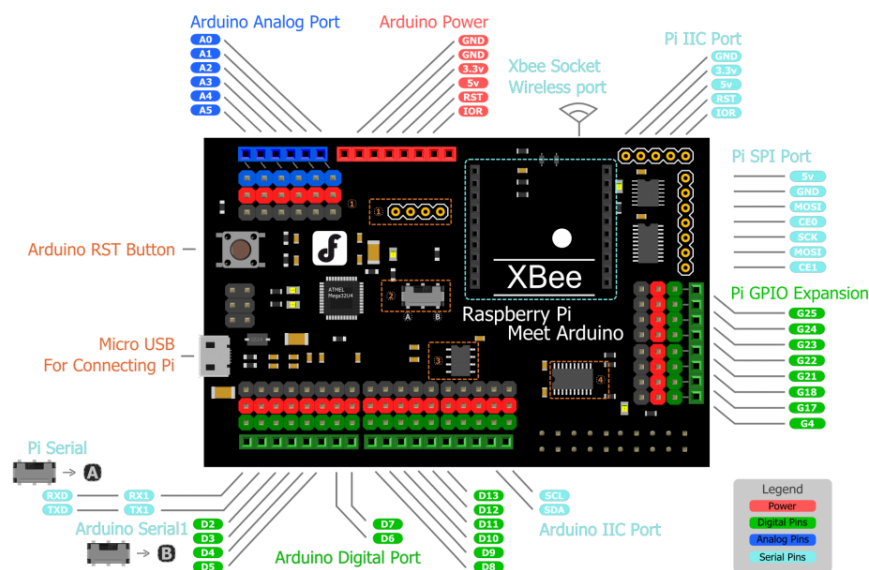


Figure 2: Arduino Shield used with pinout map [8]

2.3 – Teensy 3.2

The Teensy is a complete USB-based microcontroller, with its small size, high performance and large memory the board can manage many types of projects [9]. The Teensy board can be programmed using the Arduino IDE software, by installing an

add-on called Teensyduino [10]. Teensyduino also allows the user to choose the processor speed he would like to use.

The Teensy board offers multiple pins such as, Ground, 3.3V, Digital, Analogue and PWM. However, the Teensy is equipped with the MK20DX256VLH7 processor with a rated speed of 72 MHz which is faster than the Arduino Leonardo. Unlike the Arduino board, Teensy 3.2 cannot supply 5 volts of output power, as it is limited to a maximum of 3.3V [11]. Another important feature that is lacked in the Arduino is that on the Teensy board, almost all the pins can be modified as attach interrupt pins, pins that can pause the controller for a small amount of time to execute the Interrupt Service Routine (ISR⁴).

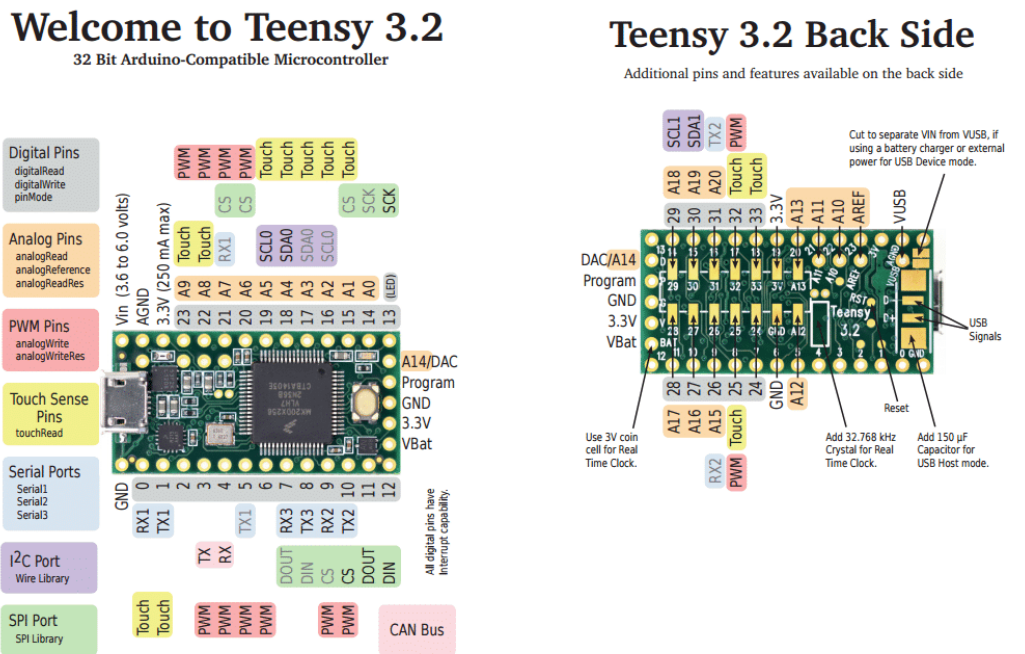


Figure 3: Teensy 3.2 board with Pin layout [12]

With the high performance and the capability to overclock the processor's speed, combining a Teensy board with the Raspberry Pi will create a powerful system that can operate at a fast rate. The following table compares the main features of a Teensy 3.2 board against an Arduino Leonardo board.

⁴ ISR: Software invoked process that interrupts the CPU from the current process. Once the ISR is complete the CPU resumes the interrupted process

Table 1: Comparison Between Teensy 3.2 and Arduino Leonardo [11] [13]

	Teensy 3.2	Arduino Leonardo
Processor	MK20DX256VLH7	ATmega32u4
Processor Speed (MHz)	72 (overclock-able to 95)	16
Length (mm)	17.78	68.6
Width (mm)	36.26	53.3
Digital I/O Pins	34	20
PWM Pins	12	7
Analogue Input Pins	21	12
Flash Memory (kbyte)	256	32
SPRAM (kbyte)	64	2.5
EEPROM (kbyte)	2	1
Interrupt Pins	All digital pins can be used as Interrupt Pins	5

2.4 - Programming Languages Used

A programming language can be regarded as rules that can instruct a computer or an electrical component on what to do and how to perform a task. There are many different languages such as, C++, C, Python and Fortran, all these languages vary a lot due to their syntax, which is the form of an instruction and how it should be set up [14].

Arduino's IDE is a simplified version of C++ with Libraries that can make writing a code easier and shorter. C++ is an object-oriented high-level language developed by Bjarne Stroustrup [15]. It is regarded as a compiled language. Compiled languages are converted directly into machine code thus, they can run significantly faster and more efficiently than interpreted languages [16]. This offers the developer control over the hardware aspects. However, the code needs to compile each time new changes are made which can make debugging, the process of identifying and removal of errors, the code complicated and time-consuming. Teensyduino software is an add-on for the Arduino IDE which allows the Teensy board to be programmed [10].

The Raspberry Pi offers different software for coding, but the easiest one to learn is Python [17], due to its easy syntax readability. This makes debugging the code easier as well, and the presence of multiple open-source libraries is a significant factor [18]. Python is an interpreted, object-oriented, high-level programming language. Interpreted languages do not require machine code to execute a program, instead they

will run through the program line by line and execute each command [16]. This is a main disadvantage which will slower the program execution speed.

2.5 – Sharp GP1A57HRJ00F Photo Interrupter sensor

The sensor in use is a standard transmissive photointerrupter with the emitter and detector in the same case but opposite to each other. The sensor contains a diode on the emitter side. The sensor requires a voltage input between -0.5 to 17 volts with a peak current of 1 ampere [19], and the recommended current for the emitter is 7mA. This sensor outputs 1 when there is nothing crossing the slot and 0 when there is something blocking the slot. A photo of the sensor can be seen in figure 4.



Figure 4: Sharp GP1A57HRJ00F Sensor [19]

2.6 - I2C/IIC (Inter Integrated Circuit) Bus

A Serial Protocol for two wire interfaces designed by Philips in the early 80's to facilitate communication between components on the same circuit board, hence the name Inter Integrated Circuit [20].

I2C is a half-duplex protocol, this means that the data can be sent in both to and from a device but not simultaneously. It uses two main wires for communication, a Serial Data wire (SDA) and a Serial Clock wire (SCL). SDA is used to transfer data shared between transmitters and receivers and SCL is used for synchronisation of the electrical component clocks on the bus. The protocol can support speeds up to 3.4 Mega-Bits/Second [21].

There are two types of device profiles on the bus a Master and a Slave. A master generates bus clock and begins a communication process on the bus, whereas a slave will respond to the commands that appear on the bus. In case of multiple slaves connected to the bus the master can address specific Slaves by their unique address ID, usually master devices do not have an address to eliminate the possibility of a slave transmitting commands to the masters. But in the case of having two or more master devices and two of them initiate a transmission on the bus, bus arbitration takes

place. Bus arbitration means that both SDA signals from the masters are compared to each other while the SCL signal is high, if one of the SDA signals is LOW and the other one is HIGH the master with the low SDA signals wins the bus arbitration process and thus, have access to use the bus. The other master will have to wait till a stop condition appears on the bus. Each device can be a transmitter, a receiver or both [22].

The SCL and SDA signals are bidirectional. They are connected via resistors to a positive power supply voltage, and they also have an open-collector to achieve a wired AND bus connection. A wired AND connection means that when the bus is free both the SCL and SDA signals are high (equivalent to 1) so to initiate a bus transmission, both lines must be pulled down to low state (equivalent to 0). Only a master can pull the clock signal into a low state to start the synchronisation of the clock and mark the start of data transmission, the master keeps the clock low till it receives the data from the receiver since the receiver cannot pull the SCL to the low state. To mark the end of the data frame the master has a stop function to indicate that the bus is free for use [22].

The typical data frame of the bus, see figure 5, shows the full data frame structure used in the bus. The frame starts with a Start Condition, which is the start function on the master. This is followed by a 7 or 10 bits address frame that is unique for different slave devices on the bus. The Read/Write bit is set as 0 if the master is writing to the slave and 1 if the master is reading from the slave. The data frame can send up to 8-bits (1 byte) of data and there is no limit of how many bytes you send if every byte ends with an Acknowledgement bit added by the receiver [22]. The I2C protocol is governed by the SMBus, System Management Bus, on the Pi and by the Wire library for the Teensy and Arduino boards. The SMBus & Wire library will manage the message frame.

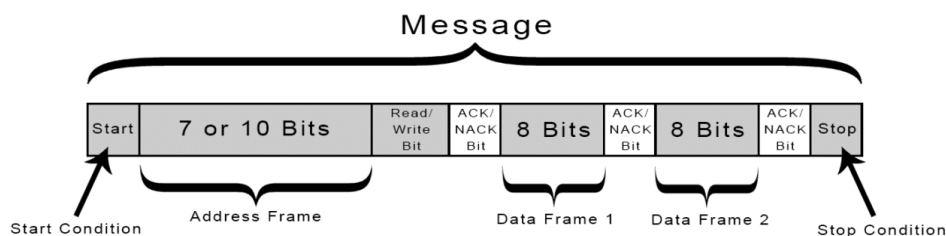


Figure 5: I2C data frame format [23]

3 – Mechanical Design

Mechanical components that will allow the mounting of the sensors to calculate the speed of the buggy. Additional components were added to the buggy to use the sensor with the recommended configuration.

3.1 – Driven Wheel

An arrangement of two sensors is used to get feedback about the direction of movement of the buggy, in a way that one sensor will be slightly leading the other in its reading. Please refer to section 4.2 for details about how the direction of the buggy is determined. This arrangement required the design of a disk encoder with teeth to not only get the direction, but the speed as well. The disk had to be designed and manufactured in house since there were no suppliers to purchase a disk with the desired dimensions.

The disk is made from 6mm black acrylic. The choice of black acrylic was to reduce weather effects on the disk and to fully absorb the light from the sensor's emitter. The design can be seen in figure 6.

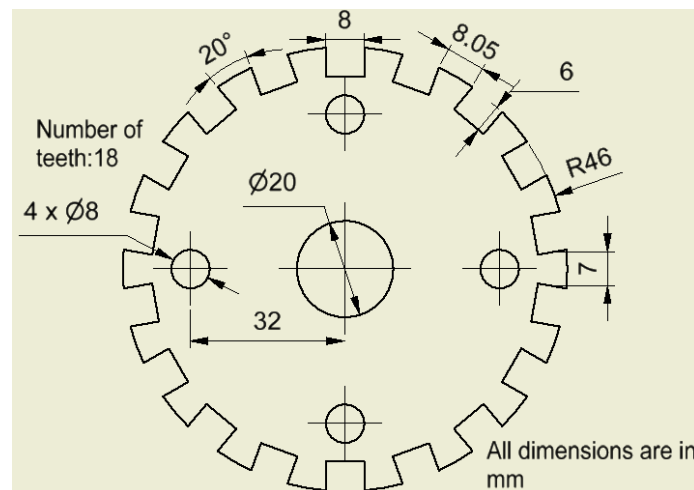


Figure 6: Disk Dimensions

The smallest diameter of the wheel is 93.2mm, therefore the diameter of the disk was chosen as 92mm. This is so that the disk will not hit the rail whilst the buggy is moving. For the sensor to have space, two 6mm spacers were added to push the disk away from the wheel. The spacers are made from plywood. See Appendix A for the CAD drawings.

The disk has a total number of 18 teeth, the number was calculated using the following formula:

$$\text{Number of teeth} = \frac{\text{Circumference}}{\text{Pitch}} \quad \text{Equation 1}$$

The pitch is the slot width and the pitch circle circumference for 1 tooth. A pitch of 16mm will calculate a number of 18.06 teeth. The slot width was kept at 8mm and the number of teeth to 18, this calculated a pitch circle circumference of 8.05mm.

A frame lip was designed to hold the components to the frame of the wheel, so that the sensor is far away from the ground. An adjuster was added to be able to change the height of the sensors from the disc. A small component was added to hold the sensors together but separated by a small distance, so that one sensor will be leading the other one. All these components were made from 3mm clear acrylic. The inventor assembly can be seen in figure 7.

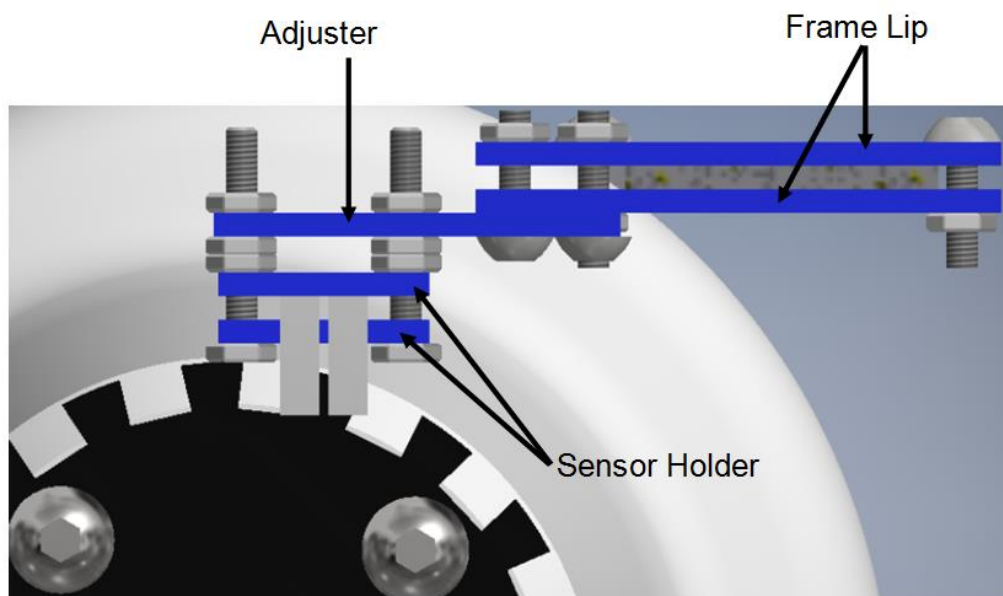


Figure 7: Inventor Assembly for the proposed arrangement

The assembly of this arrangement can be seen in figure 8 & 9. The engineering drawings of the other components can be seen in Appendix A.

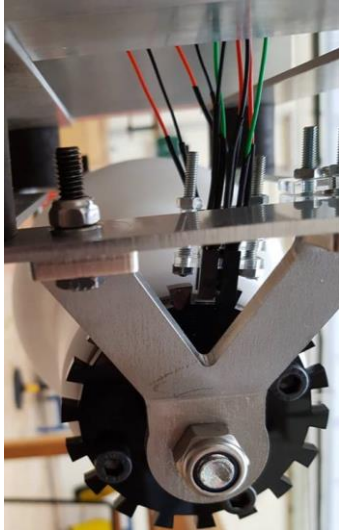


Figure 8: Arrangement for the driven wheel



Figure 9: Side view of the arrangement

3.2 – Driving Wheel

Due to the chains and the sprockets on the driving wheel, it was difficult to design the same disk as the driven wheel. So, the sensors were mounted adjacent to the chains for the chain links to break the sensors light. For this purpose, a different frame lip was designed that will allow a plate to slot through it. This plate will have angular slots to allow the sensor to slot inside. A tight fit lock is used to secure the sensor in its position. The components are made from 3mm clear acrylic. Figure 10 shows the inventor assembly of the arrangement.

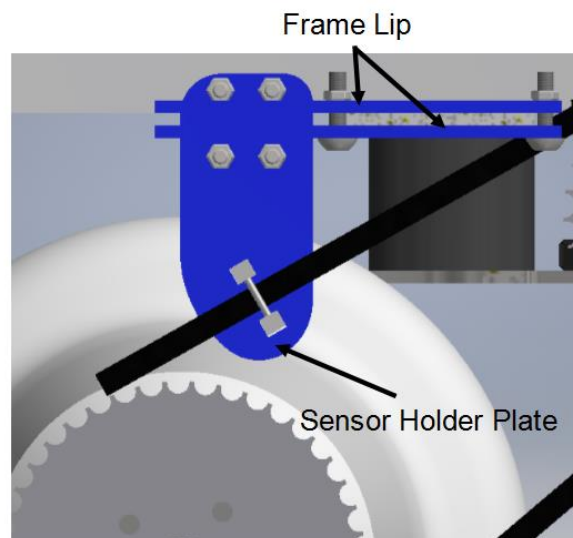


Figure 10: Inventor Assembly for the arrangement of the Driving Wheels

The whole arrangement can be seen in figures 11 and 12. The engineering drawings for the components can be seen in Appendix B.



Figure 11: Driving Wheel Arrangement



Figure 12: Driving Wheel Arrangement

4 – System

4.1 – Raspberry Pi Code (Python)

The Python code is simple and is understandable. The python code be seen in Appendix C. The main library used is the ***pigpio*** library. This library runs using the pigpio-daemon to facilitate the control of the general-purpose Input/Output pins available on the Pi [24]. The following table shows the functions used from the library along with their explanation.

Table 2: pigpio functions used [24]

Function	Explanation
pigpio.pi()	It connects to the pigpio daemon to grant access to the Pi's GPIO pins.
i2c_open(bus, address, flags)	Returns a " <i>handle</i> " to open the I2C communications to a specified bus and address. Currently there are no flags, so it's left empty.
pi.i2c_write_device(handle, data)	Sends data bytes to the device associated with the handle. This function can send up to 32 bytes of data.
(count, data) = pi.i2c_read_device(handle, Number of bytes to read)	This function reads up to 32 bytes of data from the device associated with the handle. Data received is presented as a bytearray. Count represents the number of bytes received.

The steps of the code can be seen in table 3.

Table 3: The logic of the code explained in steps

Step 1	Join address 5 as the master.
Step 2	Awaiting user input.
Step 3	Send the byte representation of the user input along with the stopping character '!'. !
Step 4	If the user input is "speed", use the function "readingSpeed". If not skip to step 10.
Step 5	Read 10 bytes from the transmission.
Step 6	Apply logical shift left by 8 bits to the first byte.
Step 7	Add the second byte from the transmission to the logically shifted first byte to obtain the speed.
Step 8	Print the rest of the bytes received as the direction of movement for the buggy.
Step 9	Return the value of the speed back to the main loop.
Step 10	The user input is not "speed", use the function "readingCharacter".
Step 11	Read up to 14 bytes of data from the device.
Step 12	Return the received values to the main loop.
Step 13	Print the returned values from the functions and return to step 2.

Step 2 happens on line 20 which shows a while True loop, this is the main loop of the code. Step 3 happens between lines 21 and 25. The command `str.encode()` returns the byte representation of the user input, so that it can be sent over the SDA. The use of a stop character "!" was to create a method to send two inputs from a single data frame for example, this method can be used to adjust the voltage input of the motor to alter its speed. In the other code, this stop character is recognised to separate the input in the data frame.

For the 4th step, a separate read function is used to read the speed of the buggy due to the logical shift section inside the function. The logical shift left is applied to the first byte only, since this byte will be logically shifted right by 8 bits on the other board and sent to the Pi as the first byte. So, to obtain the full speed value, the shifted first byte is added to the second byte. This method was used since the maximum number the byte can hold is 255 thus, using this method the maximum speed value that can be

sent is 65535. The received speed is then returned to the main loop to be printed for the user.

4.2 – Arduino/Teensy Code (C++)

As mentioned in section 2.4 that both the Arduino and Teensy 3.2 boards can use the same sketches, this means that the code written can work with Arduino and Teensy 3.2 board with few differences. The full code can be seen in Appendix D, this code was used in the Teensy 3.2 testing and is the proposed code. As mentioned in section 2.5 the Wire library controls the I2C communications on the Arduino/Teensy board. The Wire library offers a few functions that are used in the code, the following table lists the used functions along with their explanation.

Table 4: List of used functions from the Wire library [25]

Function	Explanation
Wire.begin(address)	Placed in the void setup, this function allows the device to join the specified address as a slave. To join as a master the function is left empty
Wire.onReceive(function)	Placed in the void setup, this function calls a function when a slave device receives a transmission
Wire.onRequest(function)	Placed in the void setup, this function calls a function when a master requests data from the slave
Wire.read()	Reads a byte from the bus
Wire.write()	Writes data on the bus. Data sent can be in the form of a string (character pointer) array.

The flow chart of the code can be seen in figure 13.

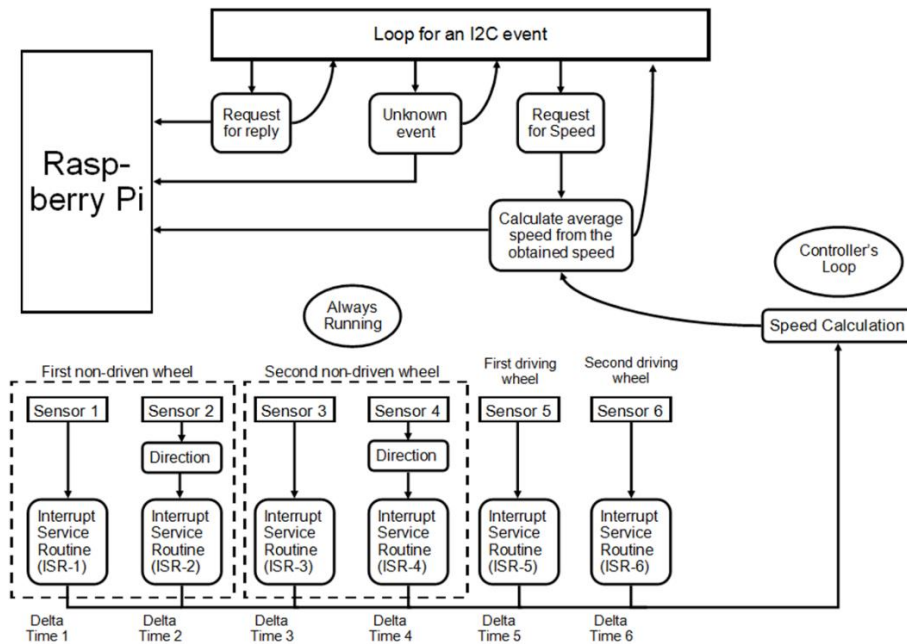


Figure 13: Flow chart of the C++ code

The following table presents the steps of the code and how it works.

Table 5: The logic implemented in the code

Step 1	Join address 5 as a slave.
Step 2	Identify the functions to be used for the <code>Wire.onReceive</code> and <code>Wire.onRequest</code> commands.
Step 3	Record the time for the <i>"fallTime"</i> variables along with the ISRs used for each sensor.
Step 4	If the wheel is moving, the speed is calculated till it is requested. If the wheels don't move for more than 0.3s then mark the speed as zero.
Step 5	Awaiting transmission from the master.
Step 6	Clear the variables to store incoming data.
Step 7	Loop for the stop character <i>"!</i> " and store the received data.
Step 8	Set a flag depending on the received data. If the received data is <i>"reply"</i> go to step 9, else if the received data is <i>"speed"</i> go to step 10, else flag it as an unknown event and go to step 13.
Step 9	Reply back with <i>"Hi Pi"</i> and then return back to step 5.
Step 10	Loop through all the speed values and add the values greater than zero then divide by the number of speed values to calculate the average speed.
Step 11	Apply logical shift left by 8 zero bits to the average speed value and store in the variable <i>"z"</i> , then apply logical AND with 0xFF (eight values of 1) to the average speed value and store it in the variable <i>"y"</i> .
Step 12	First send variable <i>"z"</i> and then <i>"y"</i> to the master, then send the direction of movement. Return back to step 5.
Step 13	Reply back with the message <i>"Unknown event received"</i> and go back to step 5.

For the 3rd step, the ISRs are activated using the “*FALLING*” condition, this condition represents that a transition from the HIGH state to the LOW state occurred. When the sensor goes through this transition the ISR will be activated. The variable “*fallTime*” is used across the ISRs of the sensors, it is used to calculate the delta time between the activation of the ISR for one sensor. The order of the ISRs is the same order that appears in the flow chart on figure XX.

The 4th step is always active. The speed calculation depends on the movement of the wheels, so when the wheels are moving the speed of the buggy is updated. Therefore, if the wheels are not moving for more than 0.3 seconds, the speed is set as zero.

For the 7th step, the controller reads all the characters within the transmission looking for the stop character to split the transmitted data. This is done because if another transmission received, the `Wire.onReceive` function will be repeated again losing any data from the previous transmission. The use of a second command in the system was aimed to control the speed of the buggy, in a way that if the first command received is “*speed*” and the second command is a number, then that number is used to alter the speed of the buggy.

The 8th step sets a flag depending on the first command. This flag is used in the `Wire.onRequest` function to set the correct data that is going to be sent back in reply to the received transmission.

Steps 10, 11 and 12 occur within the `Wire.onRequest` function. The speed values are obtained from a separate speed function that calculates the speed of the buggy every 0.1 seconds. The Teensy sends the Least Significant Bits (LSB) first, therefore in step 11 the average speed value is logically shifted to store the Most Significant Bits (MSB) of the speed. The average speed undergoes logical AND with 0xFF to store the LSB of the value. They are sent in the order of MSB first then LSB later since the Pi is expecting the first byte to be the MSB of the speed value.

Inside the speed function the first four sensors are regarded as the driven wheels. The following equation was used to calculate the speed from the sensor readings.

$$Speed = \left(\left(\frac{s \times 1000}{\delta t} \right) \div r \right) \times \left(\frac{180}{\pi} \right) \times \left(\frac{0.5}{3.0} \right) \quad \text{Equation 2}$$

Where s (in mm) is the distance between the ‘falling’ edges on the encoder disk, equivalent to 15mm (8mm gap width and approximately 7mm teeth width). The position of the sensor to the centre of the wheel is represented as r (in mm), from measurements the distance is 43mm. δt is the time since the ISR got activated till the next instance the ISR was activated again, its measured in milliseconds. The first bracket will calculate the radians per second, then multiplying it by $180/\pi$ to calculate it in degrees per second finally multiplying by $0.5/3.0$ to calculate the speed in RPM. 3 degrees/second is equivalent to 0.5 RPM. The rest of the sensors, which are mounted on the driving wheel, are calculated using the following formula.

$$Speed = \left(\left(\frac{P \times 1000000}{\delta t} \right) \div PCR \right) \times \left(\frac{180}{\pi} \right) \times \left(\frac{0.5}{3.0} \right) \quad \text{Equation 3}$$

Where p is the measured minimum distance between two chain links which is 2.40mm. Whereas PCR is the Pitch Circle Radius of the 42 teeth sprocket which is 42.5mm [26]. This calculation will calculate the angular velocity of the wheel. Multiplying the angular velocity by $180/\pi$ and then by $0.5/3.0$ will calculate the speed value in RPM.

5 - System Test

5.1 – Sensors

To follow the recommended input voltage and current. The sensor was connected to 3.3 volts and a resistor of value 660 ohms is chosen to limit the current to 5 mA through the emitter only. The receiver did not have the resistor. Figure 14 shows how the sensor should be connected, the “digital pin” refers to a Teensy digital pin.

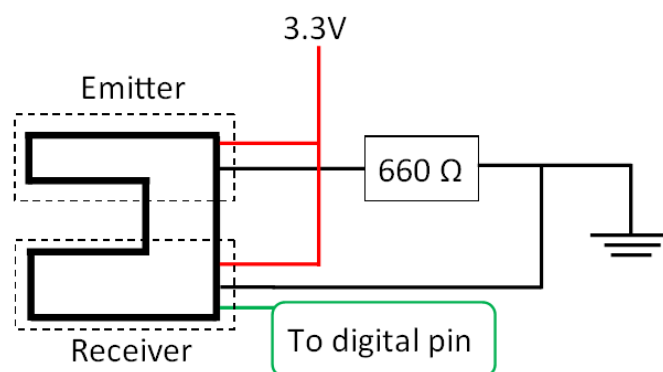


Figure 14: Circuit diagram for the Sensor in use

5.1.1 – Driven Wheels

Several methods were tested to investigate for the smoothest speed profile. The smoother the speed profile, the fewer the errors in the method used. The main aim was to have a constant feedback of the speed, so counting the number of teeth of the disk till it finishes a full cycle should not be implemented as the speed feedback method since, it is slow when the buggy is moving at a slow speed.

The first method was to calculate the speed every time a tooth passes by, as in measure the speed when the sensor changes state from 1 to 0 and then from 0 to 1. The sensor pin was connected to a normal digital pin. The method also included measuring the speed from one gap as well, same sensor logic as the tooth but inverted. The results of 1 tooth vs 1 gap can be seen in figure 15. Sensor 1 was measuring the teeth and Sensor 2 was measuring the gaps.

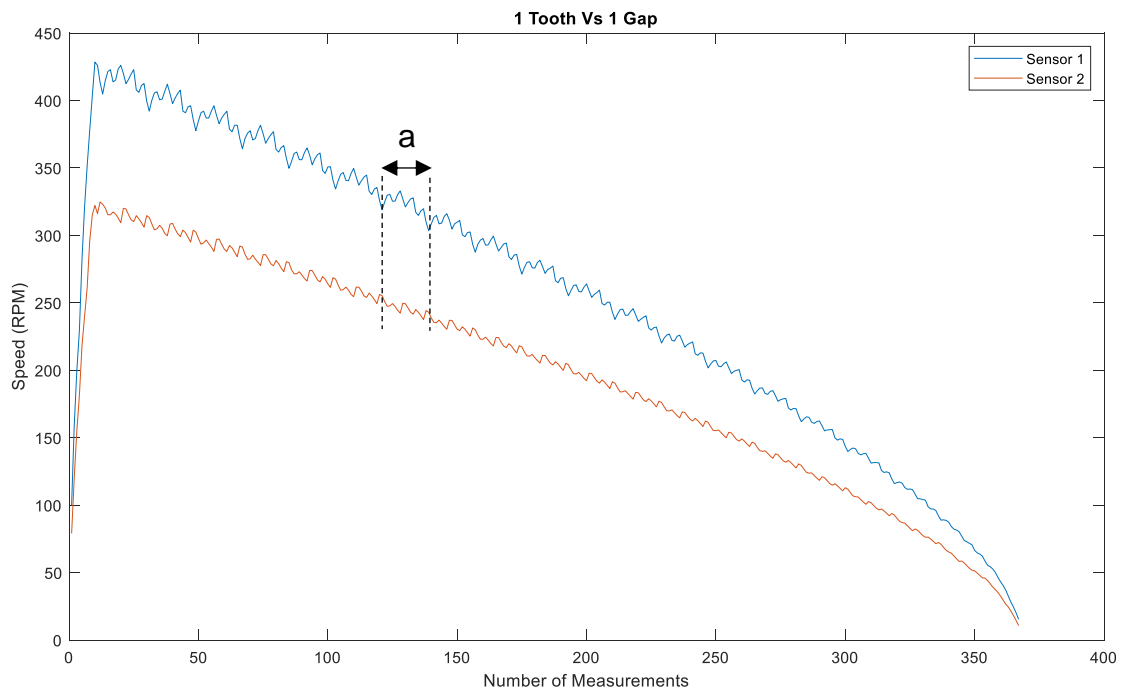


Figure 15: 1 Tooth vs 1 Gap Speed profile

From the graph, there is a huge difference between both speed profiles, this can be due to the machine error. The laser cutter used to manufacture the disk had a tolerance of 0.1mm, so the teeth of the disk may not have the same width. Moreover, the tooth width used is 7mm and the gap width is 8mm, in real life the distance may not be 15mm thus the rise of the huge difference between both speed profiles. The distortions that appear on the graphs are periodic. Letter 'a' represents the full

revolution of the disk (18 teeth). The errors have the same profile every full revolution of the disk, this can be due to machine error.

For the second method, instead of reading tooth by tooth or gap by gap, instead one sensor will only look for transitions from 1 to 0, and once it counts 2 transitions, it calculates the speed. The sensors were connected to a normal digital pin. This means that sensor have crossed a tooth and a gap, from the actual arrangement the total distance is approximately 15mm. The other sensor was left to calculate the speed every gap. This was done to check which method produces the smoothest speed profile. Sensor 1 was used to count the teeth transitions and sensor 2 was kept measuring the speed per gap. The chart can be seen in figure 16.

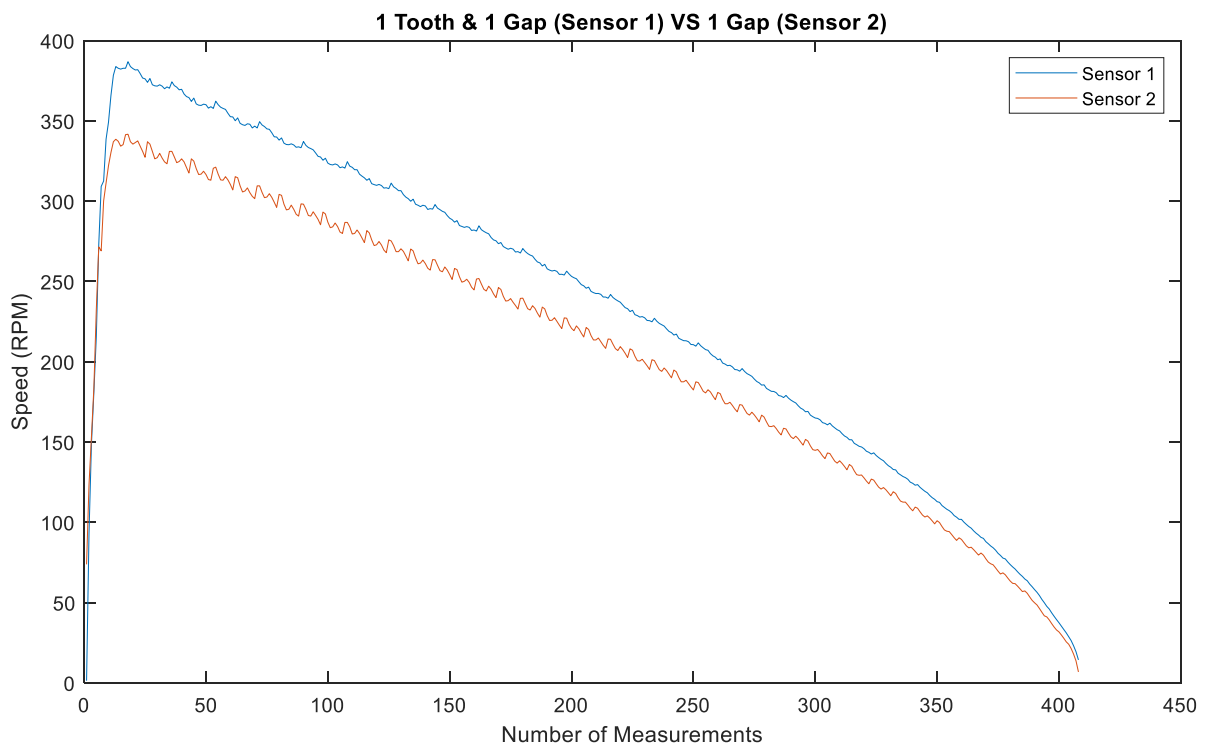


Figure 16: 2 Teeth transition VS 1 Gap speed profile

There is still periodic appearance of the errors on both profiles, but for the first sensor the distortions have been minimised. The difference between both profiles have been reduced as well, the distance was not changed the method of calculating the speed only changed.

The 2 teeth method is quick and produces a much smoother profile. To further eliminate errors, it was decided to use interrupt pins and measure the speed every 0.1 seconds for the third method.

Finally, the last method was to test 2 teeth transitions and 2 gaps transitions (based on interrupt pins) and check which provides the smoothest profile. Results are shown in figure 17.

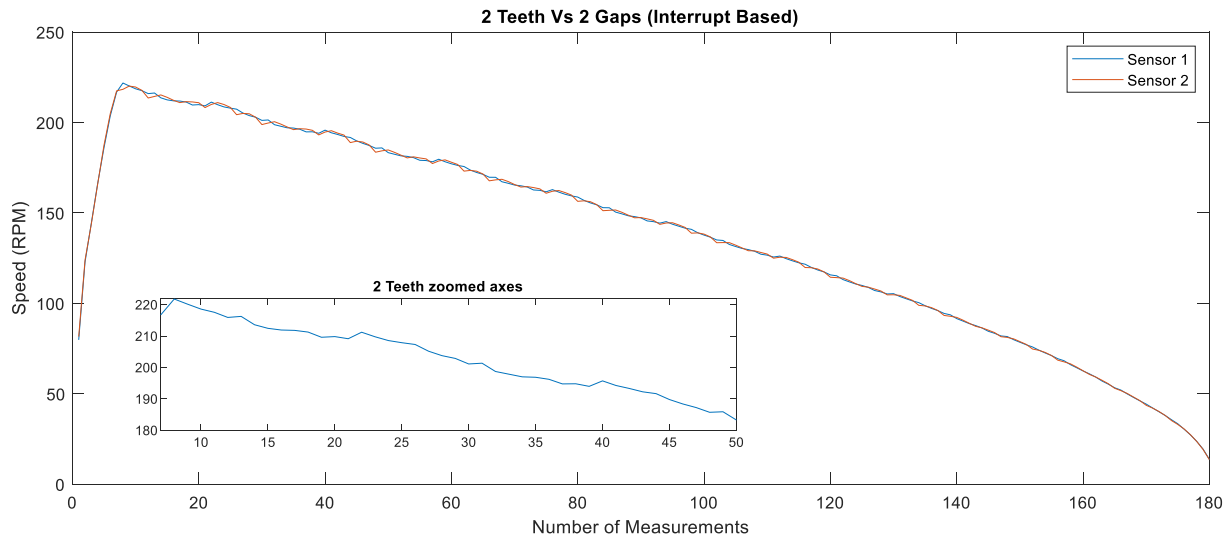


Figure 17: 2 Teeth transition VS 2 Gap transition speed profile

Both profiles seem similar, but the 2 teeth transition was chosen as the primary method to calculate the speed of the buggy. This is because its profile seems to be the smoothest, as it can be seen from the zoomed axes on figure 17. There are still periodic errors, but they have decreased significantly when the interrupt method was used. For this reason, the main method to calculate the speed of the driven wheels is the 2 Teeth transition interrupt based.

5.1.2 – Driving Wheel

For the driving wheel, the method chosen was to mount the sensor adjacent to the chain to measure the chain's linear speed and consequently the wheel's angular velocity.

The speed profile can be seen in figure 18. There is a lot of distortion visible, this is due to the vibration of the motor's frame when it was running.

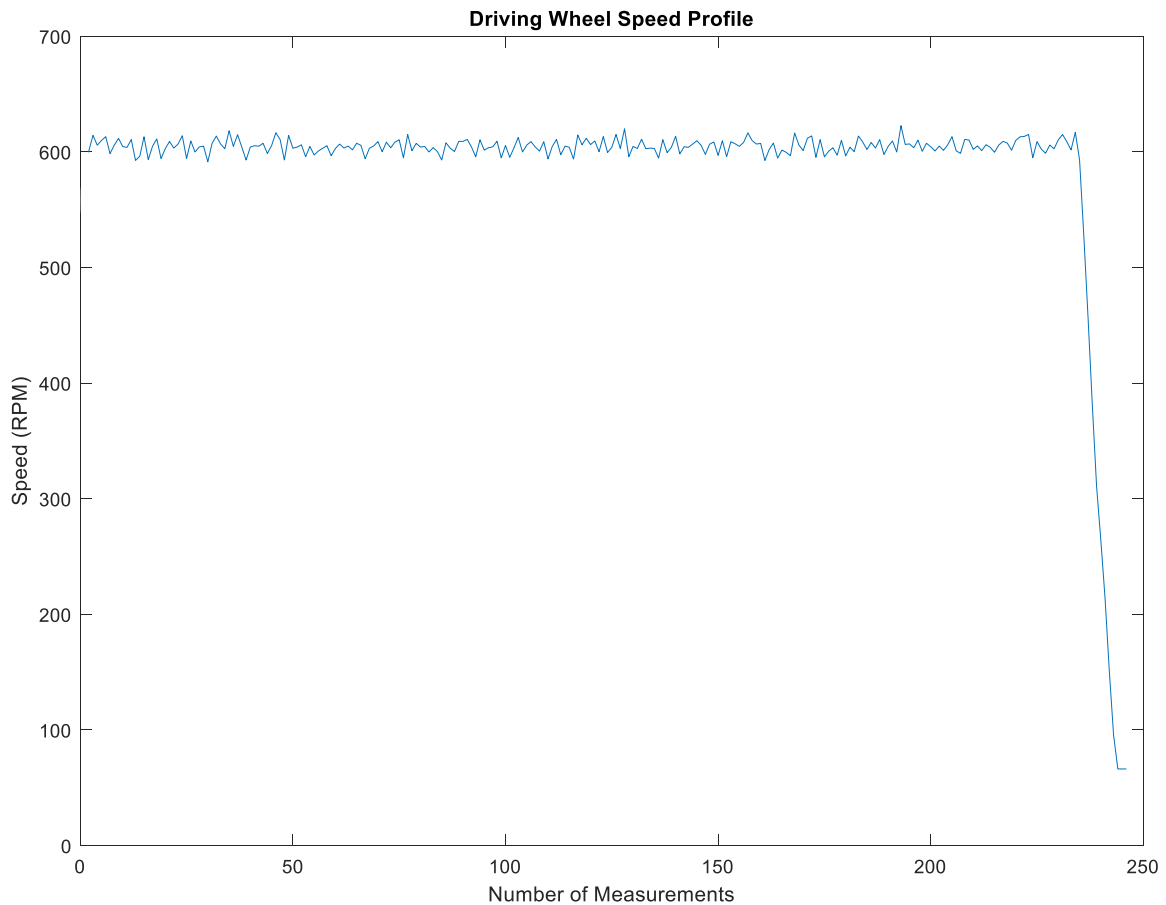


Figure 18: Speed profile of the Driving Wheel

5.2 – RPi – Teensy/Arduino Communication

There are two codes, both are fundamentally similar. The code that was used to test the Arduino board can be seen in Appendix E. This code works perfectly on the Teensy board with minimal errors.

When testing the Arduino board, the first byte that is sent on the I2C bus is always faulty. The results of the communication can be seen in figure 19.

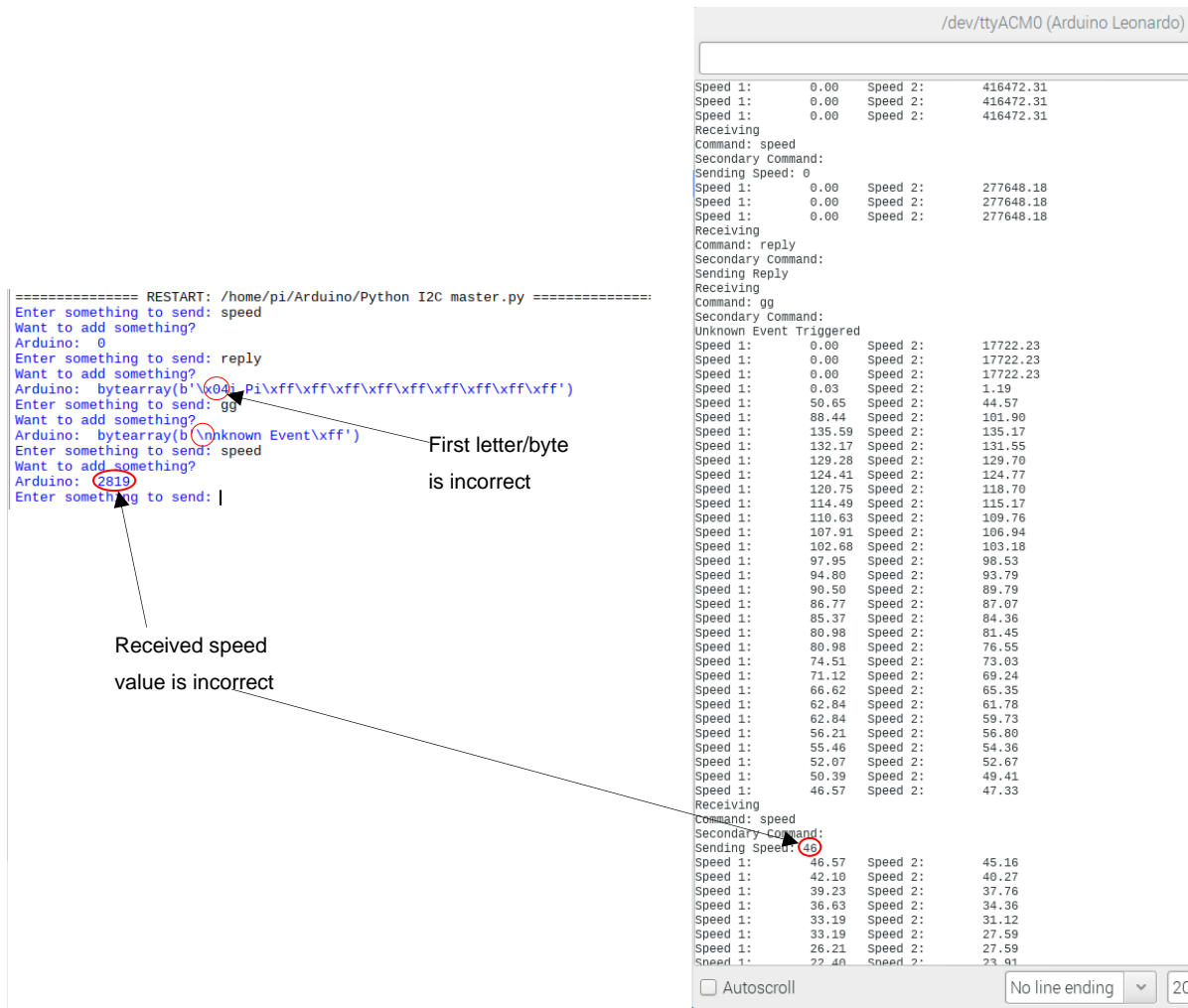


Figure 19: System testing. Raspberry Pi (on the left-hand side - python) and Arduino (on the right-hand side - Serial Monitor)

From the figure, it can be seen that from the Arduino's side speed 2 is incorrect, even when the wheel is not rotating. Furthermore, at higher speeds (approximately 300 RPM) the ISRs don't have enough time to execute thus sometimes the speed might be shown as a zero, partially because of the sensors being so close to each other and partially caused by the Arduino's processor speed. Thus, at higher RPMs the ISRs are activated more frequently. Therefore, when one ISR is being processed and the next ISR is triggered, this causes a slight delay and a prediction of a lower speed. This effect is not seen at lower RPMs because the ISRs are not activated frequently.

The Arduino applies logical shift right by 8 bits to the average speed, to store the most significant bits. It then sends the most significant bits as the second byte and the least significant bits as the third byte. This way the Arduino can send up to 2 bytes of data.

Hence, the Pi applies logical shift left by 8 bits and then adds it to the received third byte. A value of 46 is expected to be the result of the transmission but a garbled transmission of 2819 was received.

```

===== RESTART: /home/pi/Arduino/Python I2C master.py =====
Enter something to send: reply
Want to add something? no
Teensy: bytearray(b'Hi Pi\x00\x00\x00\x00\x00\x00\x00\x00\x00')
Enter something to send: hello
Want to add something?
Teensy: bytearray(b'Unknown Event\x00')
Enter something to send: speed
Want to add something?
Direction: bytearray(b'Backward')
Teensy: 0
Enter something to send: speed
Want to add something?
Direction: bytearray(b'Forward\x00')
Teensy: 390
Enter something to send: |

/dev/ttyACM0 (Teensy 3.2)
Receiving
Command: reply
Secondary Command: no
Sending Reply
Receiving
Command: hello
Secondary Command:
Unknown Event Triggered
Receiving
Command: speed
Secondary Command:
Sending Speed: 0
Receiving
Command: speed
Secondary Command:
Sending speed: 390
  
```

Figure 20: System testing. Raspberry Pi (on the left-hand side - python) and Teensy (on the right-hand side - Serial Monitor)

For the Teensy board, the first byte is sent out correctly, therefore the Teensy can fully use the 32-byte limit. The results of the tests can be seen in Figure 20. Add to that, the Pi performs the logical shift to the corresponding byte and the expected speed matches the received speed. Figure 21 shows the array that contains the speed along with a provisional sign that represents the direction of movement.

```

+ 45.42 37.19 0.00 0.00 0.00 0.00
+ 54.72 56.53 0.00 0.00 0.00 0.00
+ 55.92 55.09 0.00 0.00 0.00 0.00
+ 51.08 52.64 0.00 0.00 0.00 0.00
+ 48.89 47.93 0.00 0.00 0.00 0.00
+ 46.65 45.84 0.00 0.00 0.00 0.00
+ 42.39 43.56 0.00 0.00 0.00 0.00
+ 40.33 41.66 0.00 0.00 0.00 0.00
+ 38.83 37.99 0.00 0.00 0.00 0.00
+ 36.67 35.92 0.00 0.00 0.00 0.00
+ 34.63 33.83 0.00 0.00 0.00 0.00
+ 32.43 31.52 0.00 0.00 0.00 0.00
+ 30.05 31.52 0.00 0.00 0.00 0.00
+ 27.25 29.02 0.00 0.00 0.00 0.00
+ 27.25 26.13 0.00 0.00 0.00 0.00
+ 24.02 22.61 0.00 0.00 0.00 0.00
+ 19.98 22.61 0.00 0.00 0.00 0.00
+ 19.98 18.20 0.00 0.00 0.00 0.00
+ 19.98 18.20 0.00 0.00 0.00 0.00
+ 14.61 18.20 0.00 0.00 0.00 0.00
+ 14.61 11.80 0.00 0.00 0.00 0.00
+ 14.61 11.80 0.00 0.00 0.00 0.00
+ 0.00 11.80 0.00 0.00 0.00 0.00
- 3.36 0.00 0.00 0.00 0.00 0.00
- 18.69 3.33 0.00 0.00 0.00 0.00
- 45.23 57.58 0.00 0.00 0.00 0.00
- 68.27 68.04 0.00 0.00 0.00 0.00
- 66.59 65.66 0.00 0.00 0.00 0.00
- 62.89 60.92 0.00 0.00 0.00 0.00
- 57.30 58.12 0.00 0.00 0.00 0.00
- 53.58 54.34 0.00 0.00 0.00 0.00
- 51.36 52.09 0.00 0.00 0.00 0.00
- 48.75 47.78 0.00 0.00 0.00 0.00
- 45.20 45.90 0.00 0.00 0.00 0.00
- 43.43 44.10 0.00 0.00 0.00 0.00
- 41.18 39.42 0.00 0.00 0.00 0.00
- 38.61 37.84 0.00 0.00 0.00 0.00
- 36.93 35.52 0.00 0.00 0.00 0.00
- 34.60 32.86 0.00 0.00 0.00 0.00
- 31.00 30.77 0.00 0.00 0.00 0.00
  
```

Figure 21: The speed array that holds the speed from different sensors. The "+" means the direction is clockwise and "-" is anti-clockwise

The test was carried out using two sensors (1 driven wheel), this is because they don't require an external power source to drive the motor. Moreover, the track buggy can't be unfolded inside the lab to take readings from all six sensors.

Operating the board at a processor speed of 72 MHz is fast enough to not allow one ISR clashing with the other ISR. But at speeds exceeding 600 RPM the ISRs start to clash with each other. The test was repeated using a processor speed of 96MHz (overclock – running at higher speeds than what was intended by the manufacturers) where this issue was not visible. However, overclocking reduces the processor's life.

Both boards were able to join the I2C as a slave for the Pi, but the Teensy performance is much better than the Arduino.

6 – Discussion

The method used for calculating the speed of the buggy for the driven wheel is the 2 teeth transition based on interrupt pins, due to the profile being the smoothest out of the tested methods. When further investigating the errors for a trendline within this method. The zoomed profile shown in figure 17 showed that there is a trendline. This trendline may be due to the 0.1mm tolerance of the laser cutting machine that was used to manufacture the disk.

Furthermore, for the driving wheel the suggested method showed a lot of distortion. It is expected to have small distortions in the profile, due to the chain vibration. However, when the test was carried out the wheel was not touching the ground and the motor when turned on started to vibrate. Add to that, the sensors were not constrained to the chains. There were difficulties in constraining the sensor mounts on the frame because of the motor housing. Due to this, the sharp changes in speed and the extreme distortions are visible on the profile presented.

When investigating the error when ISRs start to clash with each other. It was discovered that the *“micros()”* timer function is updated using its own timer interrupt, with multiple functions. Even when the ISRs are kept short to avoid long pauses on the CPU, they still clash due to the *“micros()”* command.

Finally, the communication between the Raspberry Pi and the Teensy 3.2 board via I2C is error free. Whereas, for the Arduino board when the bus was checked for the devices on the I2C bus, an address of 68 appeared. The address is still present even when the Arduino joins the bus (please refer to pages 11 and 20 in the logbook). The source of the address is unknown, and its removal was not possible. When the shield is removed from the pins of the Pi the address disappears, which suggests that it belongs to the shield itself. There is a possibility that this address is the source of errors within the Arduino transmissions.

7 – Conclusion

Summarizing the report sections, the proposed system aims to automate a Track Buggy. The Buggy is a lightweight vehicle that have foreseen design changes through the years, but so far, the Buggy has never seen a system implementation. The developed system aims to start the automation of the buggy. Keeping the cost as low as possible, a combination of a Raspberry Pi and a Teensy 3.2 board is used to be the main components of the system.

The Raspberry Pi with its support for multiple communication protocols, low cost (£32), and it is a microcomputer with low power consumption, makes it the ideal board to be the main interface of the system and the master on the bus.

The Teensy 3.2 is a development board that is small and low in cost (£16), paired up with a powerful processor and a capability of controlling hardware effectively, makes it an effective microcontroller. Communication between the Raspberry Pi and the Teensy via I2C was free of errors. Yet the Teensy face a problem with ISR clashing with each other at higher RPMs but when the CPU was overclocked, there was no clashes. The Buggy's maximum speed is 15km/h. Assuming that the radius of the wheel that is in contact with the rail is 56.5mm, the angular velocity of the wheels is 73.7 rad/s or approximately 704 RPM. This speed will be an issue for the Teensy if the CPU is not overclocked.

Whereas for the Arduino Leonardo shield, the slow processor speed, caused the ISRs to clash at speeds over 300 RPMs. This meant that for the current arrangement the use of an Arduino board to use interrupt pin to calculate the speed was difficult.

Moreover, the appearance of the board's address when the shield is connected to the pins of the Pi might be the cause of the errors appearing in the transmissions from the Arduino to the Pi. Due to this, communication via I2C was not a success for the Arduino Shield. Therefore, the Shield was not used.

Furthermore, a proposed method to calculate the speed of the driving wheel by mounting the photo-interrupter adjacent to the chain. However, there were limitations in constraining the sensor mount on the frame of the wheel to keep the sensor adjacent to the chain. Yet, it was successful in calculating the speed of the driving wheel, but the profile showed a lot of distortions due to the vibrations of the chain.

Overall, aims of the project have been met. A board to control the hardware has been chosen to be the Teensy 3.2. The Teensy system testing was a success, communications via I2C ran with no errors to account for, and the implementation of an accurate method to calculate the speed of the Buggy was achieved.

8 – Future Work

To further to develop the system, wiring up of the motor to a battery pack is essential to provide a power source for the motors and the electrical components of the buggy. The use of a motor driver or a MOSFET Transistor could be used to drive the and control the MY1016 motor used for the Buggy. The motor driver should be able to drive up to 24V and 13.7A to the motor through the PWM pins of the Teensy board. The suggested motor driver is the Pololu G2 high power motor driver, it can supply up to 13A and a range of 6.5V to 40V. This driver can user bidirectional control of the motor.

To avoid weather effects damaging the sensors on the driven wheel, a compartment should be designed to cover the sensors and a part of the encoder disk. Furthermore, for the driving wheel the proposed method can be improved by using optical encoders. The readings of an optical encoders are accurate, and after revising the frame of the intermediate shaft, it is possible to mount an optical encoder on the frame of the shaft to record readings from the intermediate shaft. Kubler 8.3700.1322.1024 Incremental Encoders were found inside the box of components for the buggy, they can be used for such arrangement.

In terms of controlling the Raspberry Pi wirelessly, VNC Server is pre-installed on the Pi. VNC Viewer software can be downloaded on any personal computer or mobile phone for free. A small portable Wi-Fi should be used to connect the Pi to the internet to be able to use VNC Server. This software will provide full wireless control on the Pi from a computer or a mobile phone.

Moreover, to avoid ISRs interrupting each other a counter can be used inside the ISR instead of calling the function *“micros()”*. The counter will initiate as zero and once an ISR is activated, the counter will increment inside the ISR. Inside the *loop* an *“if”* statement can be written to check when the counter reached 1, then the time can be recorded. Within the first *“if”* statement, another *“if”* statement can be used to record the stop time when the counter reaches 2 counts. The start time should now be equal the stop time and the whole method is repeated. This will eliminate the use of *“micros()”* within the ISR for the sensors, therefore the ISR should not interrupt each other.

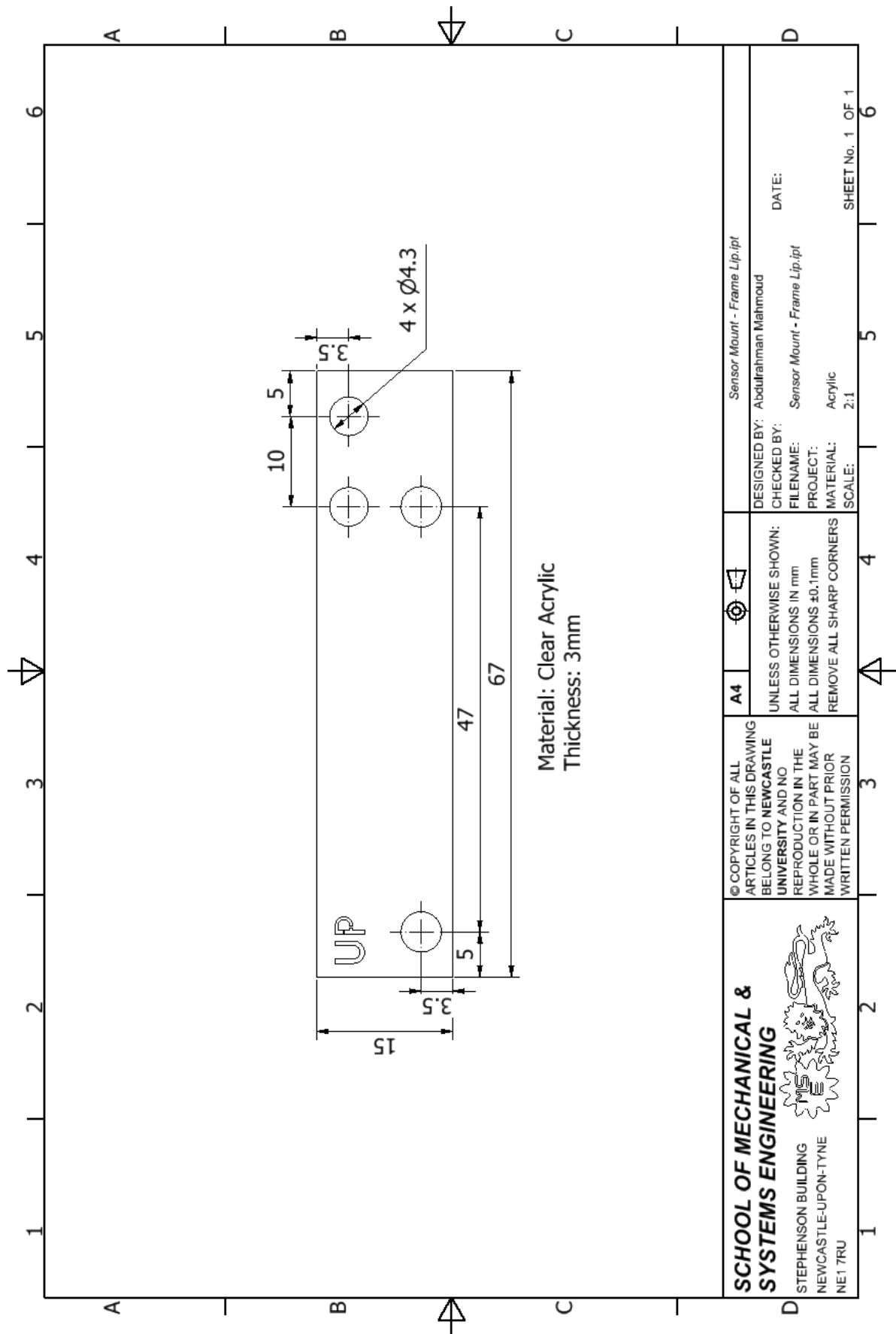
Finally, the implementation of all the improvements onto the code and testing the fully developed track rail buggy on the rail.

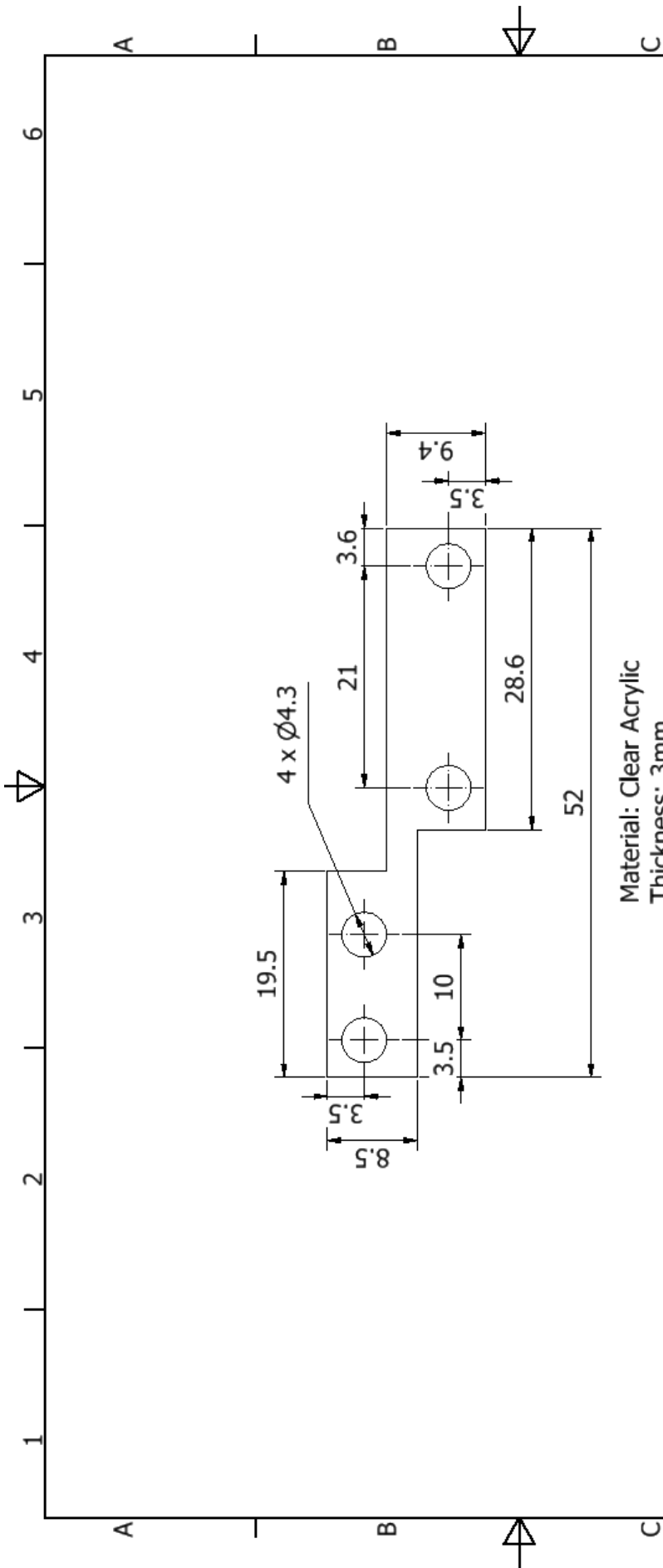
9 - References

- [1] ORR, "ORR," [Online]. Available: <http://orr.gov.uk/statistics/popular-statistics/how-many-people-use-the-railway>.
- [2] A. Coley, "Design, Optimisation and Testing of a Portable Track Vehicle," 2017.
- [3] "Raspberry Pi 3," [Online]. Available: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [4] "Raspberry Pi Pinout Diagram," [Online]. Available: <https://www.jameco.com/Jameco/workshop/circuitnotes/raspberry-pi-circuit-note.html>.
- [5] "What is Arduino?," [Online]. Available: <https://www.arduino.cc/en/Guide/Introduction>.
- [6] "What is an Arduino," [Online]. Available: <https://learn.sparkfun.com/tutorials/what-is-an-arduino>.
- [7] "Arduino Shield," [Online]. Available: <https://www.dfrobot.com/product-1211.html>.
- [8] "Raspberry Pi Meet Arduino Shield Pinout," [Online]. Available: https://www.dfrobot.com/wiki/index.php/Raspberry_Pi_Meet_Arduino_Shield_SKU:DFR0311.
- [9] "Teensy Board," [Online]. Available: <https://www.pjrc.com/teensy/>.
- [10] "Teensyduino," [Online]. Available: <https://www.pjrc.com/teensy/teensyduino.html>.
- [11] "Teensy Features," [Online]. Available: <https://www.pjrc.com/teensy/teensy31.html>.
- [12] "Teensy Picture," [Online]. Available: https://cdn.shopify.com/s/files/1/1093/9912/products/teensy-3_2-pinout-05_2048x2048.png?v=1527181603.
- [13] ArF, "Arduino Features," [Online]. Available: <https://store.arduino.cc/usa/arduino-leonardo-with-headers>.
- [14] "Programming Languages," [Online]. Available: <https://www.computerhope.com/jargon/p/proglang.htm>.



- [15] "C++," [Online]. Available: <https://www.computerhope.com/jargon/c/cplus.htm>.
- [16] "Programming Concepts," [Online]. Available: <https://thesocietea.org/2015/07/programming-concepts-compiled-and-interpreted-languages/>.
- [17] "Top Programming Language," [Online]. Available: https://www.webopedia.com/TERM/P/programming_language.html.
- [18] "What is Python," [Online]. Available: <https://www.python.org/doc/essays/blurb/>.
- [19] Sharp, "GP1A57HRJ00F," [Online]. Available: <https://www.sparkfun.com/datasheets/Components/GP1A57HRJ00F.pdf>.
- [20] "What is i2c," [Online]. Available: <https://www.i2c-bus.org/>.
- [21] I2CvUARTvSPI. [Online]. Available: <http://www.rfwireless-world.com/Terminology/UART-vs-SPI-vs-I2C.html>.
- [22] "i2c bus specification," [Online]. Available: <http://i2c.info/i2c-bus-specification>.
- [23] "How i2c works," [Online]. Available: <http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>.
- [24] "PIgpiod," [Online]. Available: <http://abyz.me.uk/rpi/pigpio/python.html>.
- [25] "Wire Library," [Online]. Available: <https://www.arduino.cc/en/Reference/Wire>.
- [26] P. N. S.-4. 42 Teeth Sprocket Datasheet. [Online]. Available: http://www.hpcgears.com/pdf_c33/13.12-13.13.pdf.

Appendix A – Driven Wheel Components Engineering drawings

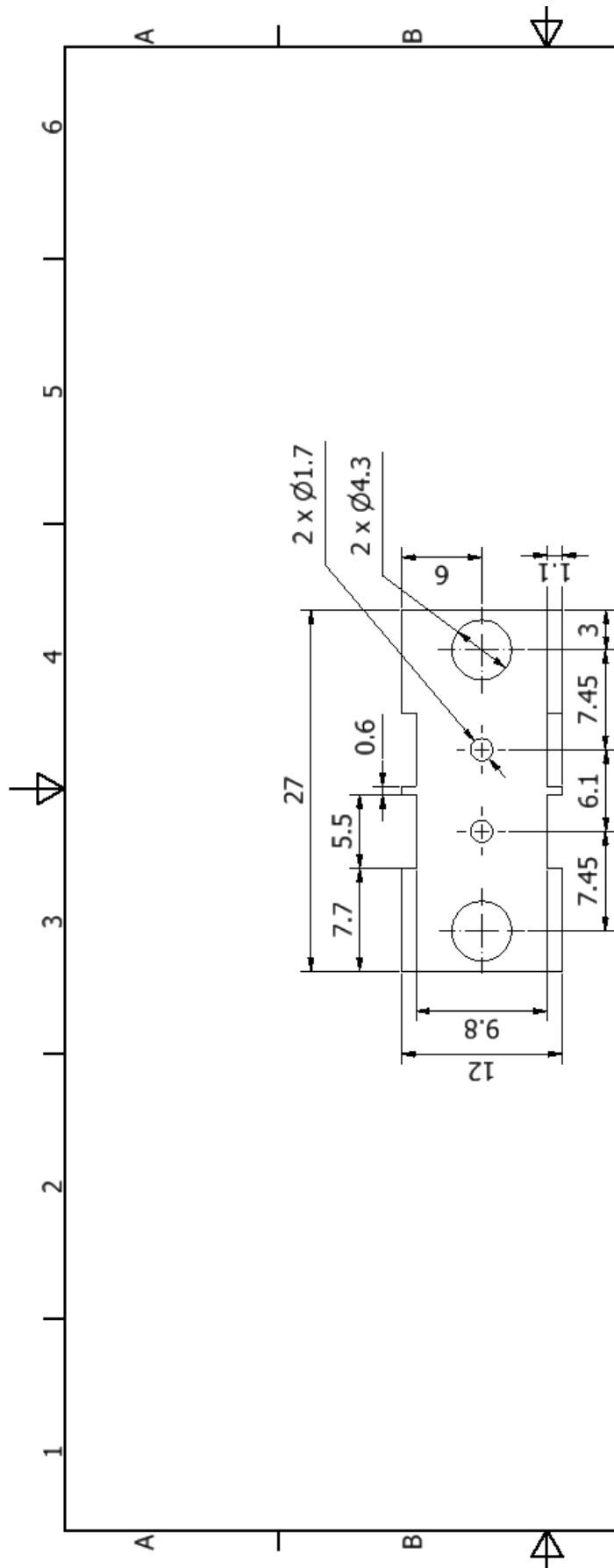






Material: Clear Acrylic
Thickness: 3mm

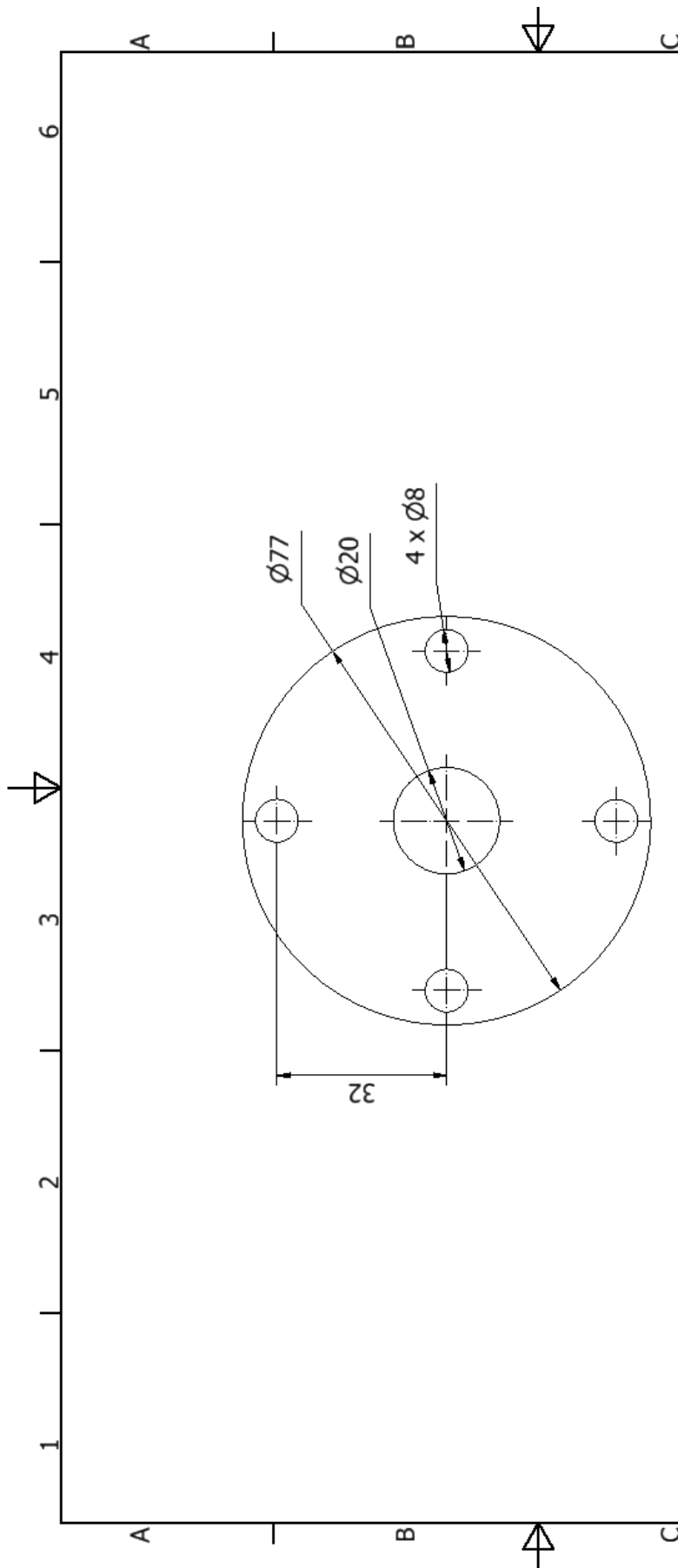
<p>SCHOOL OF MECHANICAL & SYSTEMS ENGINEERING</p>  <p>STEPHENSON BUILDING NEWCASTLE-UPON-TYNE NE1 7RU</p>	<p>© COPYRIGHT OF ALL ARTICLES IN THIS DRAWING BELONG TO NEWCASTLE UNIVERSITY AND NO REPRODUCTION IN THE WHOLE OR IN PART MAY BE MADE WITHOUT PRIOR WRITTEN PERMISSION</p>	<p>A4 </p> <p>UNLESS OTHERWISE SHOWN: ALL DIMENSIONS IN mm ALL DIMENSIONS ±0.1mm REMOVE ALL SHARP CORNERS</p>	<p>Sensor Mount - Adjuster.ipt</p> <p>DESIGNED BY: Abdulrahman Mahmoud CHECKED BY: Sensor Mount - Adjuster.ipt FILENAME: Sensor Mount - Adjuster.ipt PROJECT: Acrylic MATERIAL: Acrylic SCALE: 2:1</p> <p>DATE:</p>
---	--	--	---

SHEET No. 1 OF 1





Material: Clear Acrylic
Thickness: 3mm

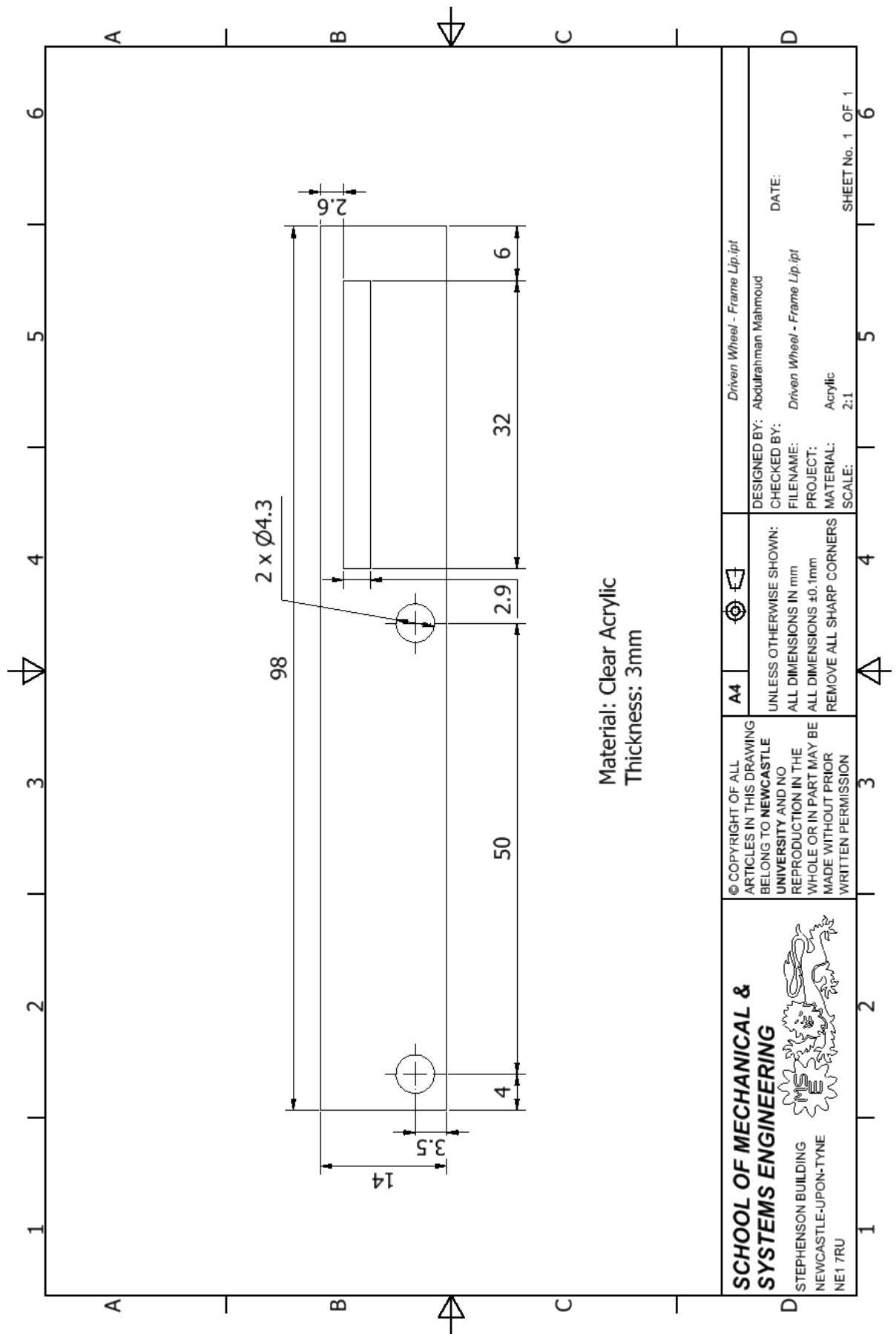
<p>SCHOOL OF MECHANICAL & SYSTEMS ENGINEERING</p> <p>STEPHENSON BUILDING NEWCASTLE-UPON-TYNE NE1 7RU</p> 	<p>© COPYRIGHT OF ALL ARTICLES IN THIS DRAWING BELONG TO NEWCASTLE UNIVERSITY AND NO REPRODUCTION IN THE WHOLE OR IN PART MAY BE MADE WITHOUT PRIOR WRITTEN PERMISSION</p>	<p>A4</p> 	<p>DESIGNED BY: Sensor Mount - Sensor Holder.ipt CHECKED BY: Abdulrahman Mat'moud FILENAME: Sensor Mount - Sensor Holder.ipt PROJECT: Sensor Mount - Sensor Holder.ipt MATERIAL: Acrylic SCALE: 2.5:1</p>	<p>DATE: _____</p> <p>SHEET No. 1 OF 1</p>
---	--	--	--	--

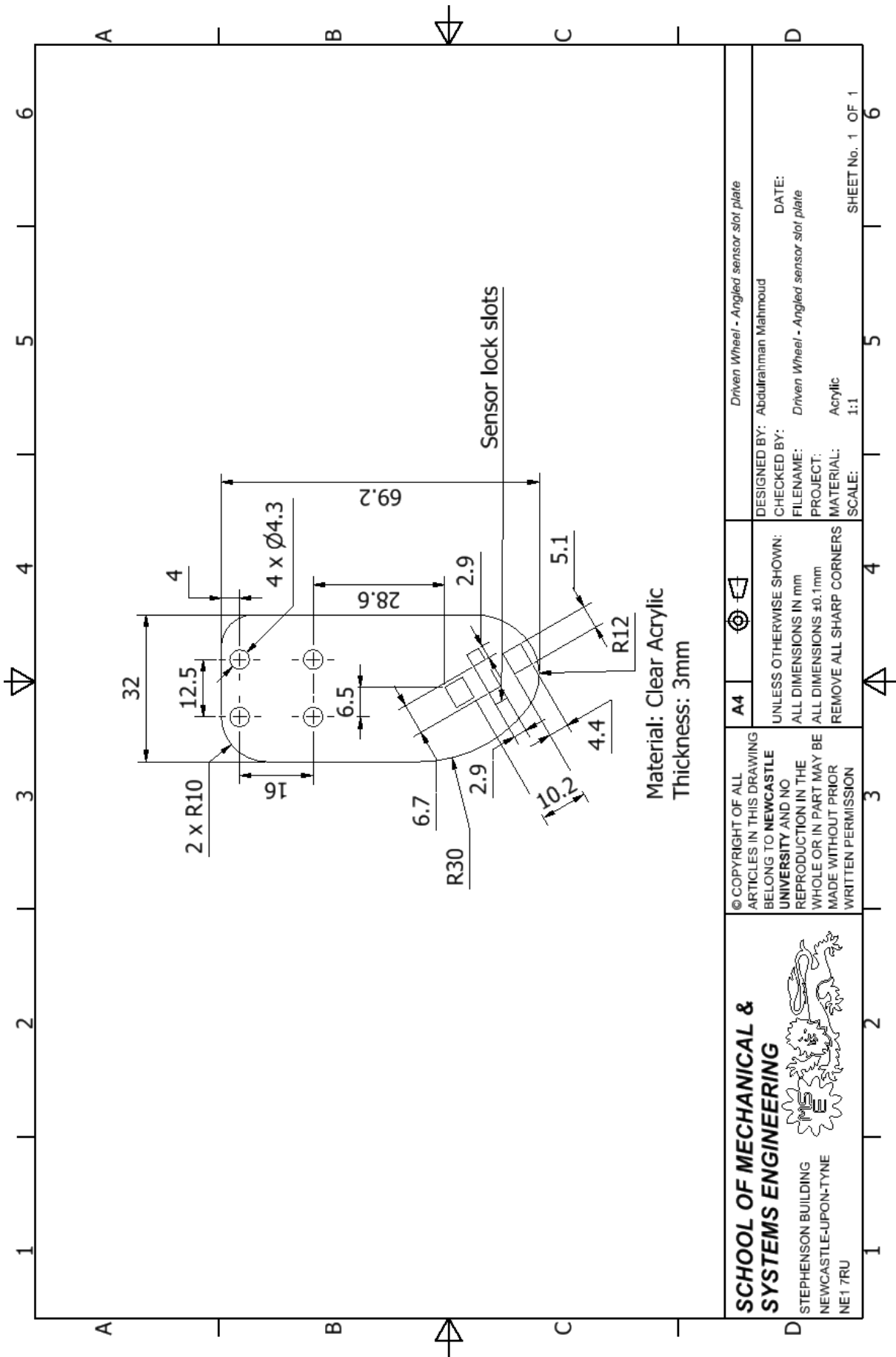



Material: Plywood
Thickness: 6mm

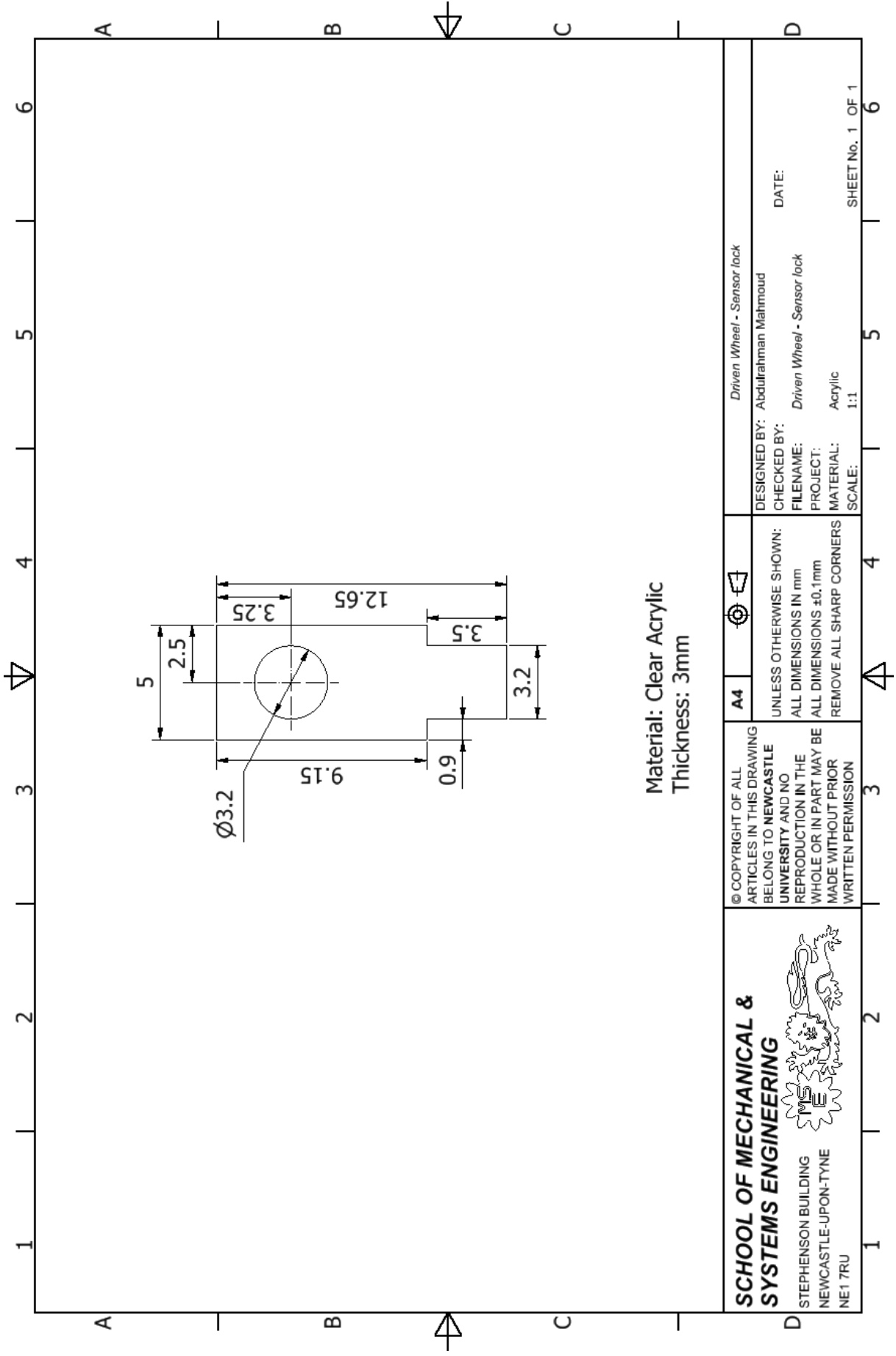
<p>SCHOOL OF MECHANICAL & SYSTEMS ENGINEERING</p>  <p>STEPHENSON BUILDING NEWCASTLE-UPON-TYNE NE1 7RU</p>	<p>© COPYRIGHT OF ALL ARTICLES IN THIS DRAWING BELONG TO NEWCASTLE UNIVERSITY AND NO REPRODUCTION IN THE WHOLE OR IN PART MAY BE MADE WITHOUT PRIOR WRITTEN PERMISSION</p>	<p>A4</p>  <p>UNLESS OTHERWISE SHOWN: ALL DIMENSIONS IN mm ALL DIMENSIONS ±0.1mm REMOVE ALL SHARP CORNERS</p>	<p>6mm spacer.ipt</p> <p>DESIGNED BY: Abdulrahman Mahmood CHECKED BY: FILENAME: 6mm spacer.ipt PROJECT: MATERIAL: Plywood, Finish SCALE: 1:1</p> <p>DATE: SHEET No. 1 OF 1</p>
---	--	---	--

Appendix B – Driving Wheel Components Engineering drawings





<p>SCHOOL OF MECHANICAL & SYSTEMS ENGINEERING</p> <p>STEPHENSON BUILDING NEWCASTLE-UPON-TYNE NE1 7RU</p> 	<p>© COPYRIGHT OF ALL ARTICLES IN THIS DRAWING BELONG TO NEWCASTLE UNIVERSITY AND NO REPRODUCTION IN THE WHOLE OR IN PART MAY BE MADE WITHOUT PRIOR WRITTEN PERMISSION</p>	<p>A4</p> <p>UNLESS OTHERWISE SHOWN: ALL DIMENSIONS IN mm ALL DIMENSIONS ±0.1mm REMOVE ALL SHARP CORNERS</p>	<p>DESIGNED BY: Abdulrahman Mahmood</p> <p>CHECKED BY: DATE:</p> <p>FILENAME: Driven Wheel - Angled sensor slot plate</p> <p>PROJECT: MATERIAL: Acrylic</p> <p>SCALE: 1:1</p>	<p>Driven Wheel - Angled sensor slot plate</p> <p>DATE:</p> <p>SHEET No. 1 OF 1</p>
---	--	---	---	---



Appendix C – Raspberry Pi code (Python)

```
#This code is written by Abdulrahman Mahmoud, 140362477
#The numbers on the left hand side represent the line numbers.

1- import pigpio      #load pigpio library
2- bus = 1            #specifying to use I2C on bus 1
3- address = 5       #the slave's address is
4- pi = pigpio.pi()  #connect to pigpio daemon
5- h = pi.i2c_open(bus,address) #the handle function for the Teensy
6-
7- def readingNumber(): #speed reading function
8-     (count, bi) = pi.i2c_read_device(h,10) #count is the number of bits received and "bi" is the bytearray of the data
9-     MSB=bi[0] << 8 #logical shift left by 8 zero bits
10-    LSB=bi[1] #reading the second byte is the LSB of the transmission
11-    Speed=MSB+LSB #adding both values for the speed
12-    Direction = bi[2:10] #the direction is expected to be the next data bytes
13-    print("Direction: ", Direction) #displaying the direction
14-    return Speed
15-
16- def readingCharacter(): #general reading function
17-     (count, b) = pi.i2c_read_device(h, 14) #reading general transmission on the bus, limiting it to 14 bytes
18-     return b
19-
20- while True:
21-     inp = input("Enter something to send: ")
22-     sending = str.encode(inp) #this is to send the command as bytes on the bus. This is command_received in the other code
23-     inpl = input("Want to add something? ")
24-     sending1 = str.encode(inpl) #this is the second command bytes_received in the other code.
25-     pi.i2c_write_device(h,sending + b'!' + sending1 + b'!') #sending the command inputs with their stopping character as bytes
26-     if (inp == "speed") or (inp == "Speed"):
27-         ReceivedBytes = readingNumber() #to use the speed read function
28-     elif (inp != "Speed") or (inp != "speed"):
29-         ReceivedBytes = readingCharacter() #use the other read function
30-     print("Teensy: ", ReceivedBytes) #print the returned data bytes.
```

Appendix D – Teensy code (C++)

```
1 /* This code was written by Abdulrahman Mahmoud to Dr. FJ Franklin for his project, automation of the rail track buggy */
2
3 #include <Wire.h>           //Wire Library for i2c
4 String command_received=""; //empty string for the first command received
5 String bytes_received="";  //empty string for the second command received
6 const char * reply = "Hi Pi"; //character pointer array for the write function
7 const char * error_msg = "Unknown Event";
8 const char * Forward = "Forward"; // This line and the next concerns the direction of the buggy
9 const char * Backward = "Backward";
10 boolean Command;          //flag used for the request function
11 boolean error;           //flag used for the request function
12 const uint8_t pin1=3;    //interrupt pin for the first sensor
13 const uint8_t pin2=4;    //interrupt pin for the second sensor
14 const uint8_t pin3=5;
15 const uint8_t pin4=6;
16 const uint8_t pin5=7;
17 const uint8_t pin6=8;
18 volatile unsigned long fallTime[6]; //an array to hold all the falltime values
19 volatile unsigned long dt[6]; //an array to hold all the Delta-Time values
20 float now_speed[6]; //an array to hold all the speed values
21 unsigned long Interval=100;
22 unsigned long previous;
23 volatile boolean bDirection1 = false; // used in the ISR to get the direction
24 volatile boolean bDirection2 = false;
25 int avg_speed;
26
27 void setup()
28 {
29   Wire.begin(5);          //join the i2c as slave on address 5
30   Serial.begin(2000000); //start the serial port at a baud rate of 2,000,000 bits/sec
31   Wire.onReceive(rec);    //This function is activated everytime a transmission occurs on the bus
32   Wire.onRequest(req);   //This function is activated after the onReceive function has finished
33   pinMode(pin1, INPUT);  //identifying that pin1 is input
34   pinMode(pin2, INPUT);
35   pinMode(pin3, INPUT);
36   pinMode(pin4, INPUT);
37   pinMode(pin5, INPUT);
38   pinMode(pin6, INPUT);
39   previous = millis();
40   for (int i = 0; i < 6; i++) // a for loop to record time for all 6 values in the array
41   {fallTime[i] = micros ();}
42
43                                     // interrupt function is activated everytime the
44                                     // sensor transitions from 1 to 0. Sensor1 being ISR
45   attachInterrupt(digitalPinToInterrupt(pin1), Sensor1, FALLING); //ISR for the 1st sensor on the 1st Non-Driven Wheel
46   attachInterrupt(digitalPinToInterrupt(pin2), Sensor2, FALLING); //ISR for the 2nd sensor on the 1st Non-Driven Wheel
47   attachInterrupt(digitalPinToInterrupt(pin3), Sensor3, FALLING); //ISR for the 1st sensor on the 2nd Non-Driven Wheel
48   attachInterrupt(digitalPinToInterrupt(pin4), Sensor4, FALLING); //ISR for the 2nd sensor on the 2nd Non-Driven Wheel
49   attachInterrupt(digitalPinToInterrupt(pin5), Sensor5, FALLING); //ISR for the 1st sensor on the 1st Driven Wheel
50   attachInterrupt(digitalPinToInterrupt(pin6), Sensor6, FALLING); //ISR for the 1st sensor on the 2nd Driven Wheel
51
52 }
53
54 void loop()
55 {
56   /*
57   if(millis()-previous>Interval) // this if function initiates the calculation of the speed every 0.1 seconds (debugging)
58   {
59     previous += Interval;
60     boolean bActive = false;
61     for(int i=0;i<6;i++) // this loop checks all the speed values if they are greater than zero
62     {
63       if (Speed (i))
64         {bActive = true;}
65     }
66     if (bActive) // This is a debugging if statement to print the speed everytime a change occurs
67     {
68       if (bDirection1 && bDirection2)
69         {Serial.print ("Forwards ");
70         }
71       else
72         {Serial.print ("Backwards ");
73         }
74       for(int l = 0; l < 6; l++)
75       {
76         Serial.print(now_speed[l]);
77         if (l < 5)
78           {Serial.print (" ");}
79         else
80           {Serial.println ();}
81       }
82     }
83   }*/
84 }
```

```

83
84 void rec(int numBytes)    //numBytes is the number of bytes received during the transmission
85 {
86     Serial.println("Receiving");
87     command_received="";    // emptying the string to make it available for the next transmission
88     bytes_received="";    //secondary string, can be used for speed control when motor is wired up
89     while(Wire.available()>0)    //a while loop to read the data on the bus when there is transmission
90     {
91         char c;
92         char b;
93         while(true)
94         {
95             c = Wire.read();
96             if(c=='\0')    //this is the stop character to separate the data inside the message
97                 {break;}
98             command_received += c; //concat the character to the string
99         }
100         while(true)
101         {
102             b = Wire.read();
103             if(b=='\0')
104                 {break;}
105             bytes_received += b;
106         }
107     }
108     Serial.print("Command: "); //debugging
109     Serial.println(command_received);
110     Serial.print("Secondary Command: ");
111     Serial.println(bytes_received);
112     if(command_received == "reply" || command_received == "Reply")
113     {
114         Command = true; //this flag is used to differ what data to be sent in the onRequest function
115     }
116     else if(command_received == "speed" || command_received == "Speed")
117     {
118         Command = false;
119     }
120     else if(command_received != "reply" || command_received != "Reply" || command_received != "speed" || command_received != "Speed")
121     {
122         error = true;
123     }
124 }
125
126 void req()
127 {
128     if(error)    //if error is true
129     {
130         Wire.write(error_msg);
131         Serial.println("Unknown Event Triggered");
132         error = false; //if not set to false the onRequest function will keep on sending this case
133     }
134     else if(!Command) //the command flag is used here to test for the cases && if command is not true
135     {
136         int sum=0;
137         int n=0;
138         for(int i=0;i<6;i++)
139         {
140             if(now_speed[i]>0) // the if statement calculates the speed and choses the values greater than zero
141             {
142                 sum+= (int)now_speed[i]; // compound addition to the speed value and increasing the "n" count
143                 n++;
144             }
145         }
146         avg_speed=sum/n; // the average speed is the sum divided by the number of speed values greater than 0 used
147         //next section is about sending a number over 255, since on the i2c its limited to 8 bits, so 9th bit will be ignored.
148         int z = avg_speed >> 8; //to capture the most significant bits
149         int y = avg_speed & 0xFF; //to capture the least significant bits, thus leaving the MSB as zero
150         Wire.write(z);
151         Wire.write(y);
152         if(bDirection1 && bDirection2) // send the direction within the transmission
153         {Wire.write(Forward);}
154         else
155         {Wire.write(Backward);}
156         Serial.print("Sending Speed: ");
157         Serial.println(avg_speed);
158     }
159     else if(Command) //if command is true
160     {
161         Wire.write(reply);
162         Serial.println("Sending Reply");
163     }
164 }

```

```

165
166 float Speed(int i) //speed calculation function for all the sensors. its an array function that takes the first
167 // 4 sensors as the non-driven wheels and the 5th & 6th sensors as the driven wheels
168 {
169   if(micros() - fallTime[i] > 300000) //if there is no operation for 0.3 seconds
170   {
171     now_speed[i] = 0; //if there is no operation for all the sensors, make them all zero
172   }
173   else
174   {
175     if (i <= 3) { //for the first 4 sensors calculate the speed of the non-driven wheels
176       now_speed[i]=((15E3 / (float) dt[i]) / 0.043) * (180.0 / 3.141) *(0.5/3.0);
177     }
178     else { //for the rest of the sensors calculate the speed of the driven wheels
179       now_speed[i]=((2.4 * 1000000)/(dt[i] * 42.5)) * (180.0 / 3.141) * (0.5/3.0);
180     } //Chain Speed with relation to the wheel's angular velocity
181   }
182   return now_speed[i];
183 }
184
185 void Sensor1()
186 {
187   unsigned long Now1 = micros(); //when the ISR is activated, record the time
188   dt[0] = Now1 - fallTime[0]; //calculate the time since the last time the ISR was called.
189   //the first time the code runs, the falltime is taken from the setup.
190   fallTime[0] = Now1;
191 }
192
193 void Sensor2()
194 {
195   bDirection1 = digitalRead (pin2);
196   unsigned long Now2 = micros();
197   dt[1] = Now2 - fallTime[1];
198   fallTime[1] = Now2;
199 }
200
201 void Sensor3()
202 {
203   unsigned long Now3 = micros();
204   dt[2] = Now3 - fallTime[2];
205   fallTime[2]= Now3;
206 }
207
208 void Sensor4()
209 {
210   bDirection2 = digitalRead (pin4);
211   unsigned long Now4 = micros();
212   dt[3] = Now4 - fallTime[3];
213   fallTime[3] = Now4;
214 }
215
216 void Sensor5()
217 {
218   unsigned long Now5 = micros();
219   dt[4] = Now5 - fallTime[4];
220   fallTime[4] = Now5;
221 }
222
223 void Sensor6()
224 {
225   unsigned long Now6 = micros();
226   dt[5] = Now6 - fallTime[5];
227   fallTime[5] = Now6;
228 }

```

Appendix E – Arduino code (C++)

```
1 /* This code was written by Abdulrahman Mahmoud to Dr. FJ Franklin for his project, automation of the rail track buggy */
2
3 #include <Wire.h> //Wire Library for i2c
4 String command_received=""; //empty string for the first command received
5 String bytes_received=""; //empty string for the second command received
6 const char * reply = "Hi Fi"; //character pointer array for the write function
7 const char * error_msg = "Unknown Event";
8 boolean Command; //flag used for the request function
9 boolean error; //flag used for the request function
10 const uint8_t pin1=3; //interrupt pin for the first sensor
11 const uint8_t pin2=4; //interrupt pin for the second sensor
12 volatile unsigned long fallTime1; //used to calculate delta time for each falling edge of the sensor (abbreviation 1 means its the first sensor, 2 is second sensor)
13 volatile unsigned long dt1; //delta time for the first sensor
14 unsigned long previous1;
15 unsigned long Interval1 = 100;
16 volatile unsigned long fallTime2;
17 volatile unsigned long dt2;
18 unsigned long previous2;
19 unsigned long Interval2 = 100;
20 int Direction; // This variable is multiplied by the speed equation
21 float now_speed1;
22 float now_speed2;
23 int avg_speed;
24
25 void setup()
26 {
27   Wire.begin(5); //join the i2c as slave on address 5
28   Serial.begin(2000000); //start the serial port at a baud rate of 2,000,000 bits/sec
29   Wire.onReceive(rec); //This function is activated everytime a transmission occurs on the bus
30   Wire.onRequest(req); //This function is activated after the onReceive function has finished
31   pinMode(pin1,INPUT); //identifying that pin1 is input
32   pinMode(pin2,INPUT);
33   fallTime1 = micros();
34   previous1 = millis();
35   fallTime2 = micros();
36   previous2 = millis();
37   attachInterrupt(digitalPinToInterrupt(pin1),Sensor1,FALLING); // interrupt function is activated everytime the sensor transitions from 1 to 0. Sensor1 being ISR
38   attachInterrupt(digitalPinToInterrupt(pin2),Sensor2,FALLING);
39 }
40
41 void loop()
42 {
43   int sen1 = digitalRead(pin1); //this function and the next one are used to calculate the direction of movement
44   int sen2 = digitalRead(pin2);
45   Direction = -1; //this line and the next function sets the direction, if sen1 is not leading sen2 then the direction is -1
46   if(sen1 != sen2)
47   {Direction = 1;}
48   if(millis() - previous1 > Interval1) //this function always calculates the speed every 0.1 seconds
49   {
50     previous1 += Interval1;
51     now_speed1 = speed1();
52   }
53   if(millis() - previous2 > Interval2)
54   {
55     previous2 += Interval2;
56     now_speed2 = speed2();
57   }
58 }
59
60 void rec(int numBytes) //numBytes is the number of bytes received during the transmission
61 {
62   Serial.println("Receiving");
63   command_received=""; //to empty the string to make it available for the next transmission
64   bytes_received="";
65   while(Wire.available()>0) //a while loop to read the data on the bus when there is transmission
66   {
67     char c;
68     char b;
69     while(true)
70     {
71       c = Wire.read();
72       if(c=='!') //this is the stop character to separate the data inside the message
73       {break;}
74       command_received += c; //concat the character to the string
75     }
76     while(true)
77     {
78       b = Wire.read();
79       if(b=='!')
80       {break;}
81       bytes_received += b;
82     }
83   }
84   Serial.print("Command: "); //debugging
85   Serial.println(command_received);
86   Serial.print("Secondary Command: ");
87   Serial.println(bytes_received);
88   if(command_received == "reply" || command_received == "Reply")
89   {
90     Command = true; //this flag is used to differ what data to be sent in the onRequest function
91   }
92   else if(command_received == "speed" || command_received == "Speed")
93   {
94     Command = false;
95   }
96   else if(command_received != "reply" || command_received != "Reply" || command_received != "speed" || command_received != "Speed")
97   {
```

```

98     error = true;
99 }
100 }
101
102 void req()
103 {
104     if(error) //if error is true
105     {
106         //Wire.write(0x00); //remove this line when using teensy the first byte is sent out correctly
107         Wire.write(error_msg);
108         Serial.println("Unknown Event Triggered");
109         error = false; //if not set to false the onRequest function will keep on sending this case
110     }
111     else if(!Command) //the command flag is used here to test for the cases && if command is not true
112     {
113         //Wire.write(0x00);
114         if(now_speed1 > 0 && now_speed2 > 0)
115             {avg_speed = (((int) now_speed1) + ((int) now_speed2) / 2);} //casting the retrieved speed to calculate the average speed
116         if(now_speed1 == 0 || now_speed2 == 0) //sometimes the ISR interrupts the other ISR thus, the speed might be zero
117         {
118             //This if function just checks if a speed value is zero and sets the average speed to the non-zero value
119             if(now_speed1 == 0 && now_speed2 == 0) //if both speeds are zero, do nothing!
120             {}
121             if(now_speed1>0 && now_speed2==0)
122             {avg_speed = now_speed1;}
123             else if(now_speed1 == 0 && now_speed2 >0)
124             {avg_speed = now_speed2;}
125         } //next section is about sending a number over 255, since on the i2c its limited to 8 bits, so 9th bit will be ignored.
126         int z = avg_speed >> 8; //to capture the most significant bits
127         int y = avg_speed & 0xFF; //to capture the least significant bits
128         Wire.write(z);
129         Wire.write(y);
130         Serial.print("Sending Speed: ");
131         Serial.println(avg_speed);
132     }
133     else if(Command) //if command is true
134     {
135         //Wire.write(0x00);
136         Wire.write(reply);
137         Serial.println("Sending Reply");
138     }
139 }
140 void Sensor1()
141 {
142     unsigned long Now1 = micros(); //when the ISR is activated, record the time
143     dt1 = Now1 - fallTime1; //calculate the time since the last time the ISR was called.
144     //the first time the code runs, the falltime is taken from the setup.
145     fallTime1 = Now1;
146 }
147
148 float speed1() //speed calculation function for the first sensor
149 {
150     if(micros() - fallTime1 > 300000) //if there is no operation for 0.3 seconds
151     {return 0;} //return speed as zero
152     return ((15E3 / (float) dt1) / 0.043) * (180.0 / 3.141) * (0.5/3.0); //otherwise, calculate the speed
153 }
154
155 void Sensor2()
156 {
157     unsigned long Now2 = micros();
158     dt2 = Now2 - fallTime2;
159     fallTime2 = Now2;
160 }
161
162 float speed2() //speed calculation function for the second sensor
163 {
164     if(micros() - fallTime2 > 300000)
165     {return 0;}
166     return ((15E3 / (float) dt2) / 0.043) * (180.0 / 3.141) * (0.5/3.0);
167 }

```