2023

Product Report

DANA GRONER (652553), LEON NGUYEN(670876), GIJS DREIJLING(659933)

BACHELOR PHASE

PROJECT: E-ESE-S3-PRJ

TUTOR: HUGO ARENDS

REGTERSCHOT RACING

HAN UNIVERSITY OF APPLIED SCIENCES

Preface

For our semester 3 project our team was tasked with developing an initial data logging system for the company Regterschot Racing. In order to document the project a product report was created. It consists of an initial analysis, what the product must be, how we make it and finally how we realised and tested it.

It is vital to create such a document in order to explain how we solved the main assignment as well as explain the development choices. Keeping a well-documented project ensures a complete understanding of what has been done even if the project has been set down for some time, and if a new team continues working with it.

This document is intended for teachers and students who have a background in software development and embedded systems. This product report is most useful for readers who are interested in data logging, how to develop the software to log and read the data, as well as how the sensors in the car are utilised to read their data.

This product report is not written to encourage developers to create their own system, but rather to gain an understanding of the already developed product and how to continue working with it to improve.

The project was not possible without the help of fellow team members in Regterschot Racing, but also the work rooms given to us by the staff at MIC and our project tutor: Hugo Arends.

Dana Groner, Leon Nguyen and Gijs Dreijling

Arnhem 2023

Summary

The company Regterschot Racing created a project for embedded systems engineering students as the company was in need of a development team for a data logging system. They need this system to track their race car when it races and to be able to analyse the data after the race to enhance performance. More specifically, the students would need to read the car's sensors and not only store them in a database, but they must also create a graphical user interface to see the data from each sensor in tables and graphs.

The team firstly needed to research what is the best hardware and software to use, as well as the speeds that can be reached in data transfer. The hardware includes the sensors, how they work as well as why they are useful for data analysis. The hardware also explains the ADC converter used in order to read the AC signals with the raspberry pi. Next the software was investigated. MQTT was researched as it is often used in IOT projects, the main question was whether or not it would transmit data fast enough.

After talking with the client, a list of requirements were created. This gives a clear idea as to what actually needs to be created in a clear and specific manner. That way, when the project is finished, both parties have the same expectation as to what the product must actually do.

In an architectural diagram the cohesion between the hardware is described. Here you can get an understanding as to what hardware is actually used and how it all works together. This goes into more detail as to how the described functionality will be achieved.

Realisation is where the functionality is achieved. Here you can find snippets of code written as well as pictures and schematics to see how to hardware is connected. Realisation explains the process of creating the product but also problems that were encountered and how they were solved. As the technical design gives more of a theoretical description as to how the product will be created, the realisation gives a clear idea to the real working process.

Finally the functionality is tested, this is done by doing acceptance testing while going through the list of functional requirements. No unit tests have been done. To summarise, all required functionality has been achieved with the exception of 2 functions. However this was due to time constraints and not to lack of knowledge.

The goals of the Regterschot Engineering project for the students was to continue to build up their knowledge as an embedded system engineer. And for the company it was for us to create a first version data logging system. Both have been achieved as the students were able to continue to learn about software development by creating a user interface and also learn more about databases and data communication. The students were also able to take a step into the automotive world and learn about car sensors as well as why their data is so important. Version number 1 of the data logging system has been successfully created and is now ready to be received by the next development team. This first version is a great base to work with and improve on for the future versions.

Table of Contents

1. Introduction

The company Regterschot Racing, owned by HAN student Erik Regterschot, are currently developing a race car for their DNRT race in May 2023. However, they want to take it to the next step. They want to develop their own data logging system in order to enhance race performance as well as do race analysis. That is why they have asked ESE students to create this for them. This data logging system will be the very first version/prototype however it must have the basic functionality required. Once this is established the company can continue improving the data logging system for further versions.

The objective of this project was not only to create a product for a company, but to gain knowledge as embedded systems engineers to learn more about embedded systems and software development. By being introduced to new projects and functional requirements, the students need to be flexible in their problem solving and find new ways to achieve the desired result.

However, next to improving as student engineers, the goal is to also create a "Version 1" working prototype. This needs to be functional according to the described functional requirements, but also malleable for the future developers who will continue working with this prototype to create the next version.

There are a few important things to note when it comes to how the project was worked on and the methods used. What was crucial to our understanding was starting with rapid prototyping. This give us a few weeks to rapidly expand our knowledge and experience for what we will continue working with. The next few weeks were spent working on documentation in order to do it correctly. Finally, with the help of daily stand-ups, the actually bulk of the code was created and hardware was integrated. To give a broad understanding to our method, the waterfall or 'V' model was used. In hindsight this could be substituted with agile instead to work with sprints.

In chapter 2 you will find the analysis where some research has been done to find the best hardware and software for the task. Next, in chapter 3 you will find the functional design where a list of functional and technical requirements are made. After that, in chapter 4, there is the technical design. Here is described how the functionality will be achieved. Chapter 5 describes the realisation of the product and chapter 6 shows how the product is tested. Then, in chapter 7 you will find the conclusion of the project as well as recommendations for the future. Finally, chapter 8, is the references. After that you will find the appendices.

2. Analysis

2.1. Introduction

A race car has many sensors that can give crucial insight to the performance of the car during the race. However not every sensor has the same level of importance, and not every sensor has to log data at the same speed.

Having to create a data logging system for the sensors of a race car, research must be done to find out how often each sensor must be logged as well as the priority of each sensor. The next problem is to research what hardware and what software is suitable for getting this job done.

The aim of this research analysis is to answer the following questions:

- What is the sampling period of each sensor in the race car.
- How fast can the raspberry pi, including the ADC converter (MCP3008), log data.
- How fast is MQTT as a communication protocol.

By answering these research questions we will be able to find out whether or not the hardware and software we already possess is fast enough to log the data at the desired speed for each sensor.

The structure of this report is as follows:

In chapter 2.2. you will find the sampling rate of each sensor that will be used in the car. Here you can also find an explanation as to why that specific sampling rate is chosen.

In chapter 2.3. there is research that describes the hardware speed. In this case that is the speed of the raspberry pi as well as the speed of the ADC converter.

Chapter 2.4. is about MQTT and what speed this communication protocol can reach. If MQTT ends up not being fast enough, further research will have to be done regarding a different communication protocol. Whether that be finding a different one or creating our own.

Finally in chapter 2.5. you will see the conclusions made by the research done. Here decisions will be made to whether or not the software and hardware are fast enough to use for the project.

2.2. Sensors

The automotive team's research heavily influences every choice that is made. All information on the selection of the sensor and the sample rate is provided in the automotive document. The results of the research will be in this section. The project currently uses five sensors: a wheel speed sensor, differential oil pressure and temperature sensor, gearbox oil temperature sensor, and fuel pressure sensor. The priorities of the sensors are listed as follows: a wheel speed sensor, differential oil pressure and temperature sensor, gearbox oil temperature sensor, and fuel pressure sensor. The reasoning for this is that wheel speed sensor needs constant update due to its volatile changes while the other sensors change slowly. With all the sensors combined the microcomputer should be at least 10hz fast to sample all the sensors. The next paragraph will explain the choices of sensors.

2.2.1. Fuel pressure sensor

VDO pressure sensor 0-5 Bar was chosen for the task and the reasoning is in automotive document. The sampling rate is 2hz.

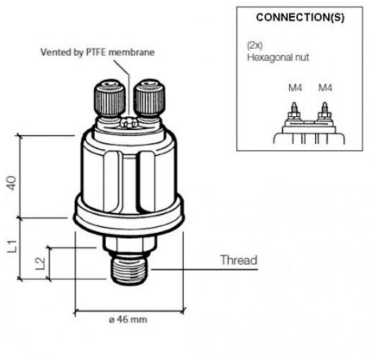


Figure 1: VDO pressure 0-5 Bar sensor

2.2.2. Gearbox oil temperature sensor

Bosch 0 281 002 170 was chosen for this project and the reasoning is found in the automotive document. The sampling rate is 1hz.

Figure 2: Bosch 0 281 002 170

2.2.3. Differential oil temperature sensor

The sensor will be the same as gearbox temperature sensor in Gearbox oil temperature sensor in 2.2. The sampling rate is 1hz.

2.2.4. Differential oil pressure sensor

VDO Pressure Sensor 0-10 Bar. The sampling rate is 2hz.

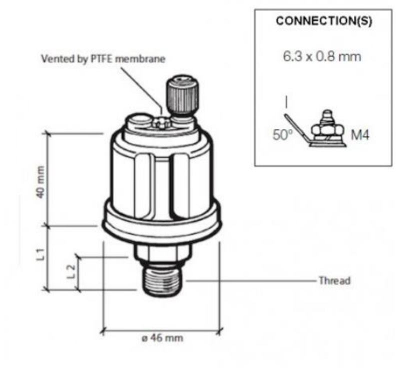


Figure 3: VDO pressure 0-10 Bar sensor

2.2.5. Wheel speed sensor

The sensor came from an already used BWM car. The sampling rate is 5hz.

2.3. Hardware

Raspberry pi and hat speed

Our main reasons for Choosing the raspberry PI 4 and the MCP3008 as Hat is because we happened to be able to get our hands on those easily, but even if that's the case, we still need to make sure both are capable of doing the tasks we require of it. For our part of the project, we only make the code for one sensor, so we limit our hardware accordingly, to make sure there are no issues with this first sensor and hardware. When these work as intended, we'll look at the rest.

2.3.1. Raspberry

The sampling rate of a raspberry pi 4 is 63Khz (*Raspberry Pi Datasheets*, n.d.). However, after searching through different sources, many had contradicting information. And so, this has gone untested.

2.3.2. Hat

The MCP3008 is an ADC (Analog to Digital Converter). It will turn the raw voltage coming in from the car sensors into readable data we can use for our code.

The MCP3008 is capable of 200.000 samplings per second (200Ksps) we if attach 5V's to it. If we apply 2.7V, we have 75Ksps. Meaning that regardless of the voltage, it will be more than sufficient for the task. This information can be found in the datasheet of the MCP3008.

Users online discussing it say that's not actually reachable though, and examples I could find didn't go higher than 96Khz, even with 5 Volts

2.4. Software

2.4.1. MQTT

MQTT (Message Queuing Telemetry Transport) is a messaging protocol used for messaging between machines. It uses a publish/subscribe system where each client communicates to each other via an MQTT broker. This broker is in charge of receiving published data and sending it to each subscriber (Cope, 2022). In the figure below you can see a diagram of how the clients work together with the broker.

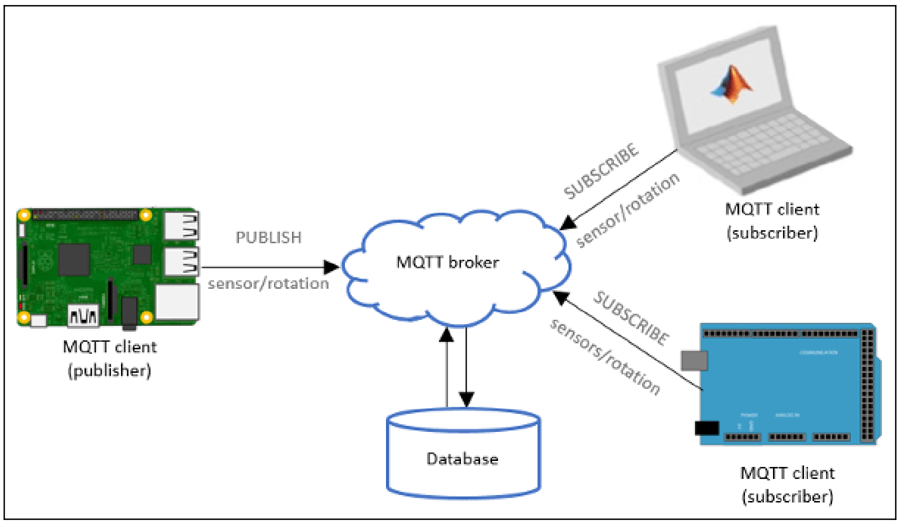


Figure 4: https://nl.mathworks.com/help/supportpkg/raspberrypi/ref/publish-and-subscribe-to-mqtt-messages.html

2.4.1.1. Usage

What is depicted in figure 4 is actually close to what we are trying to realise. The MQTT broker will be hosted on the admin laptop, as will the database. The raspberry pi that is in the race car will publish data to the MQTT broker, which will send the values to the database. Then via node-red, the user can connect to the database.

2.4.1.2. Pros and Cons

In the following source (Hübschmann, 2022) you can find a list of pros such as:

- Message delivery ensured
  By choosing a specified QoS, you manipulate the speed of the messaging as well as ensure that each message is delivered when needed.
- Lightweight
  Compared to a HTTP header that is around 8000 bytes, a MQTT header is only 2 bytes and a bit of code. This makes it great to use with small and low-powered IoT devices.
- Battery friendly
  As MQTT has been made for harsh conditions, it is built around low energy consumption. Again, compared to HTTP, MQTT uses 170 times less energy on 3G networks and even 47 times less energy on Wi-Fi networks.

(*7 Advantages of MQTT Protocol for IoT Devices - Esperso*, 2021)

This source also lists a few cons:

- No inbuilt security
  MQTT requires a layer of security on top built by the end-user. This adds a level of difficulty with those unexperienced in developing a secure environment.
- Latency issues
  Because the data must go through the cloud, the speed is significantly slower. For average IoT functions it is fast enough. However, with this specific project, the data of the race car must be sent as fast as possible.
- Bad developer friendliness
  In MQTT the clients don't converse directly with each other, rather through the MQTT broker/server. This means that a client that publishes will not receive a direct confirmation from the subscriber that its message has been received. This means you would have to create another program, that once again goes through the broker, that gives feedback when the subscriber has received a message. This inevitably slows down the whole program because everything must be streamlined through the broker.

2.4.2. Node-RED

Node-RED is a tool programmers use in order to wire together APIs, hardware devices and online services. By using a browser-based editor, the developer can put together flows using various nodes to create their desired program. The flows that are created are stored using JSON. This makes it easy to import and export when backing up and/or sharing with others. (*Node-RED*, n.d.)

2.4.2.1. Usage

In this project Node-RED is used in order to connect the user dashboard with the database. Node-RED supplies various nodes that can be utilised to change the look and use of the dashboard. The dashboard is what the user sees and is their GUI. The Node-RED application is connected to the database, here it will retrieve the user-specified data and then display it on the dashboard.

2.4.2.2. Pros and Cons

If you look at (*«Node-RED: From A to Z»: Advantages and Opportunities*, 2022) you can see that various pros of Node-RED are:

- Ease of use
  Node-RED uses components such as a built-in drag-and-drop. This makes it very easy to create and connect software even if you have little to know experience in coding.
- Debugging
  The debugging feature that Node-RED uses makes it very easy to see where you are going wrong. You can easily see where data flows and where it gets stopped simply by adding a debugger node and connect it to where you suspect it is going wrong. In the debugging window you can see what the node is actually receiving or sending.
- Availability

Since Node-RED is an open-source platform, it is very popular not only in the professional but the amateur field. This means there is a lot of help and examples regarding many of the features that Node-RED has.

However, Node-RED does have some cons such as:

- Slow
  Although it is praised for its speed in the professional field, Node-RED is still built on Node.js. When compared to C++, Node.js is noticeably slower. At this source you can see an experiment done to test this idea. (*Wayback Machine*, n.d.)
  Although Node.js does have its advantages, this specific project is reliant on speed in order to send data and in this case, C++ outperforms it.
  (Dev As Pros, n.d.)
- Node availability
  If you are a less experienced programmer, you are limited to the nodes already available on Node-RED. And if you have a more specific program you want to build, there might not be a node available for you to use. This means you would have to create your own, however this can be difficult when starting from scratch.
- Dashboard limitations
  Although the dashboard-ui node is useful, it is not the interface you would want to use for a finished product. It is more suited to rapid prototyping and demos. Again, if you were to want a more elaborate GUI then you would have to create your own node for this.

2.5. Conclusion

In conclusion, this analysis succeeded in confirming the already thought suspicions. For the required functionality of this version of the project, the described hardware and software will be good enough. Here the sampling rate needed for each sensor is described, therefore it can always be referenced when equipment needs to be improved.

Next to that the use of Node Red and MQTT were established. Because this version of the data logging system is focused on functionality and not speed, the speed of the software is less important than initially thought. After testing the speed of MQTT using several QOS, it was very suitable for the project. Node Red is also more than adequate for developing the GUI and interpreting the messages of sensor data. In the end, a few seconds of delay is not a problem as long as the given data has the correct time stamp.

For hardware the MCP3008 works in speed and has enough channels for the sensors, however this must eventually change because more sensors will be added. The raspberry pi is also fast enough for the current project, however this will eventually be swapped out for an industrial computer because it needs to withstand bumps and shakes, and more pins and modules are needed than the RPi has.

3. Functional Design

3.1. Introduction

A functional design is created to establish an understanding between engineer and client/customer. As a technical design is there to explain **how** the product will be made, a functional design is here to explain **what** is to be made. Once this document is created, the client can look and confirm that what the engineers want to create corresponds to what the client wants. You can see this document as a sort of contract as to what the end product must be.

To start, in chapter 3.2. you will find the **SMART** functional specifications. In short: these are **SMART** demands/specifications that are building blocks as to what the final product will be. At the start of this chapter, you can find more elaboration on what **SMART** means.

Then in chapter 3.3. there are the technical specifications. These are additional specifications (such as demands for hardware as well as software) made by the client.

Finally, in chapter 3.4., you will see the user interface. Here you can see sketches/estimates of the final product and a quick explanation on how it will work.

For the functional and technical specifications, we will work according to the **MoSCoW** method. This means that each demand/specification will have a label: Must-have, Should-have, Could-have and maybe even Won't-have. This gives the engineers a better idea as to what must take priority when creating the product. The Won't-have is also good to draw a line as to where the project will stop if other engineers pick up the project later. You will find the label for each demand in their corresponding table.

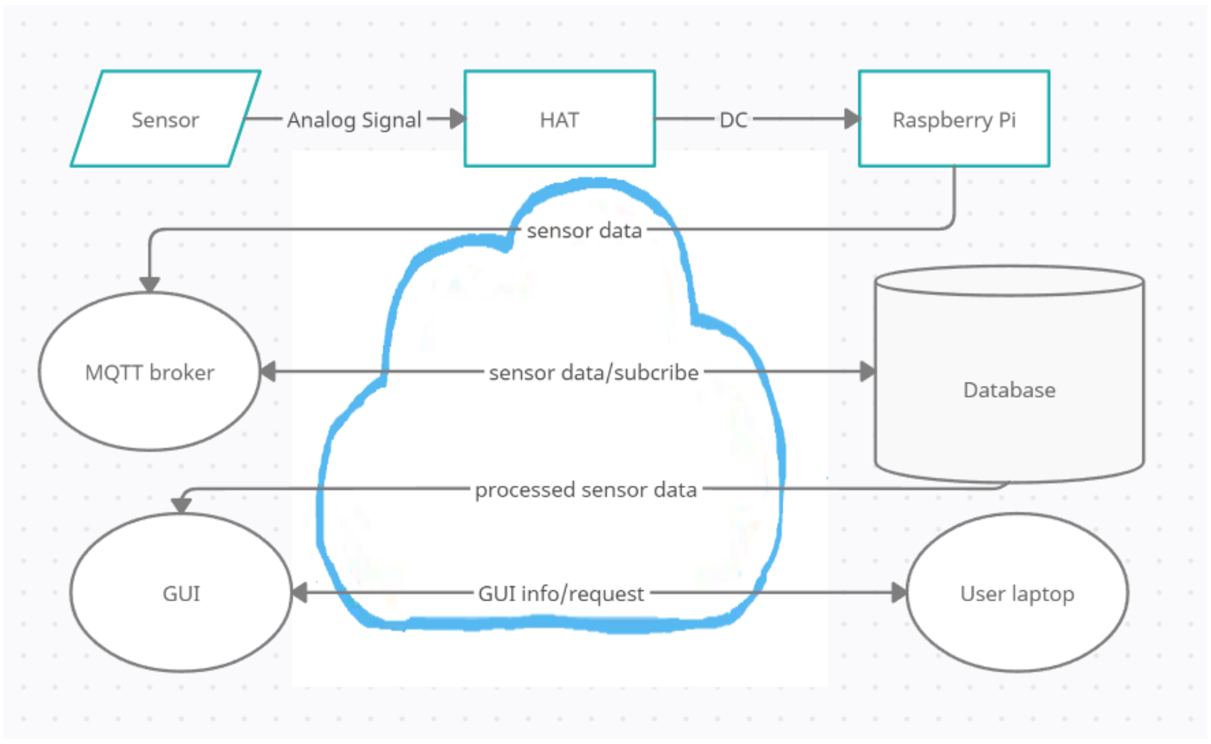
3.2. Functional Specifications

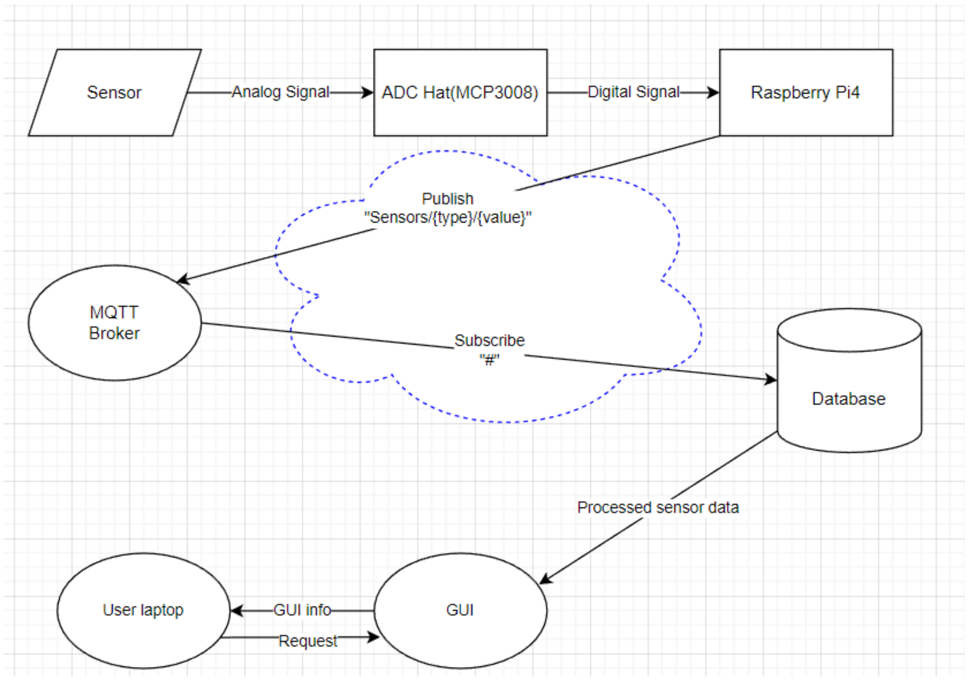
The functional specifications are demands that must be met for the product. These demands can be separated into sub-demands and so on. Each demand (and sub-demand) must be **SMART**. This means that each point will be Specific, Measurable, Assignable, Realistic and Time-related. This ensures that each demand is crystal clear in what it means.

Table 1: Functional Specifications

SMART Functional Specifications		
#	MosCoW	Description
F1	**M**	**The data logging system has a graphical user interface**
F1.1.	M	The GUI displays the data history in a table as well as a line graph
F1.1.1.	M	By default, the history page displays the whole table for the specified sensor. The line graph then displays the X* most recent entries of that sensor.
F1.1.2.	S	In the GUI history page, the user can specify the desired timeslot for the line graph and the table.
F1.2.	M	The GUI displays live data (X* most recent entries) in a line graph and shows the most recent entry on a speedometer
F1.3.	M	The user can use buttons to choose what sensor data they want to see
F1.4.	M	The GUI must be accessible for the racing engineers (PC Access) during the race
F1.5.	M	The GUI pulls the data from a remote database
F1.6.	C	The GUI must have a lap number attached to each data entry
F1.6.1	C	The user must be able to compare data according to the lap number
F2	**M**	**The raspberry pi logs incoming data from 5 types of sensors**
F2.1.	M	The system logs data from 4-wheel speed sensors (m/s)
F2.2.	M	The system logs data from 1 fuel pressure sensor (bar)
F2.3.	M	The system logs data from 1 gearbox oil temperature sensor (C)
F2.4.	M	The system logs data from 1 differential oil temperature sensor (C)
F2.5.	M	The system logs data from 1 differential oil pressure sensor (bar)
F2.6.	M	A research analysis will be held in order to determine how often each sensor will be logged, as well as the speed needed for the communication protocols (like MQTT)
F2.7.	M	Each data entry contains an ID (each entry gets its own ID number), timestamp (YYYY-DD-MM HH-MM-SS-MS) and measurement
F2.7.1.	C	Each data entry also contains lap number
F2.7.2.	C	Each data entry also contains geo coordinates
F3	**M**	**The raspberry pi sends all sensor data to a remote database**

F3.1.	M	The raspberry pi correctly processes the incoming sensor data and sends it to the database.
F3.2.	M	Database is located on a remote server
F3.3.	C	The database can be accessed using 4G





3.3. Technical Specifications

Additional requirements that the client or customer has for the product are included in these technical specifications. For instance, it may be the type of microcontroller or the programming language that must be used to create the product. The technical requirements are presented in a straightforward and organized manner using a table. For ease of use in the rest of the report, each specification has an identifying number.

Table 2: Technical Specifications

SMART Technical Specifications		
#	MosCoW	Description
T1	M	Use Raspberry Pi 4 to manage AC signals from the sensors
T2	M	Use Node-Red and Arduino IDE for software development
T2.1	M	Use Node-Red to intertwine/combine all the software (GUI, Sensor data, database)
T2.2	M	Use an Arduino Uno and the Arduino IDE for the rapid prototyping phase
T3	M	There are 5 mandatory types of sensors used in the car
T3.1	M	Wheel speed sensor
T3.2	M	Fuel pressure sensor
T3.3	M	Gearbox oil temperature sensor
T3.4	M	Differential oil temperature sensor
T3.5	M	Differential oil pressure sensor
T3.6	C	Brake line pressure sensor front brakes
T3.7	C	Brake disc temperature sensor all four wheels
T3.8	C	Fuel temperature sensor

3.4. User Interface

The User interface consists of 2 pages, one being a History page and the other a Live page. The History page is almost identical to the live page except for 2 things. A speed gauge and a time specification.
What both have is buttons to view specific sensors and the data they give, like the motor temperature, the m/s and others. In the example pictures, only these 2 were made with 2 extra buttons made for showcase purposes.

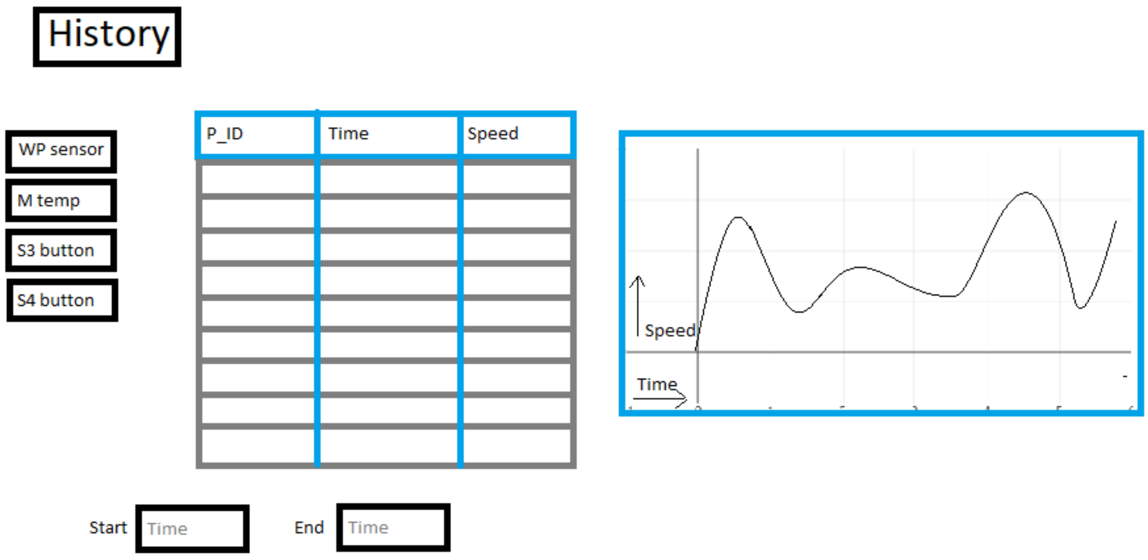


Figure 5: History Page of user interface

For the Live data, every second the graph and table get updated with the current speed with the speed gauge adding a nice visual touch to what's happening.

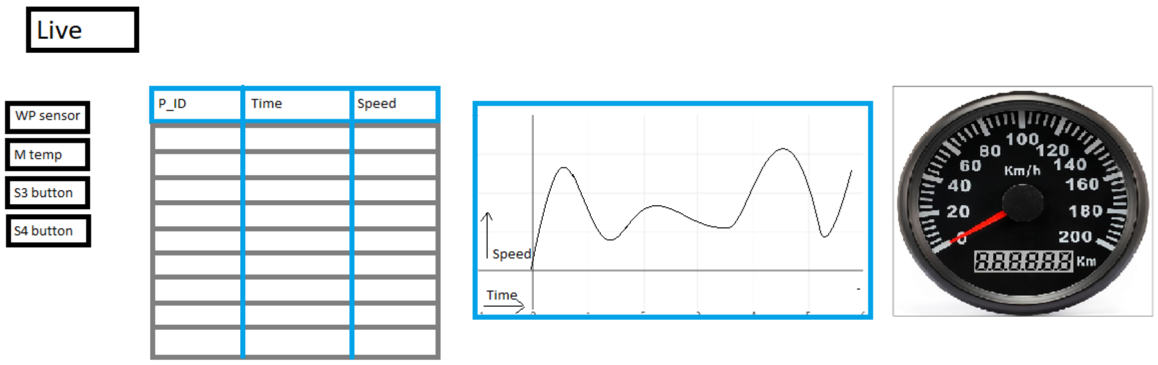


Figure 6: Live Page of user interface

4. Technical Design

4.1. Introduction

As you have seen in the functional design, up until now we have only discussed **what** will be created but little has been discussed about **how** we are going to achieve this. The answer to how this product will be created will be discussed in the technical design. By reading this document the reader will get a better understanding as to what the subsystems are of the product, the interfaces as well as some software architecture.

Firstly, you will find chapter 4.2. This will be about architecture. This divides the product into subsystems. Here you will see how each piece of hardware is connected with the next and how everything is intertwined.

The next chapter is 4.3. and will go into more detail about the interfaces between the hardware: how they are connected and how they communicate. Each interface found in chapter 4.2. will be elaborated on in this chapter.

Finally, there is chapter 4. 3. This chapter describes the software of the product. As the previous chapters have gone into more detail as to how the hardware components are connected, this chapter describes more the software architecture. Here you will find any diagrams made such as a flowchart, state diagram, sequence diagram, class diagram, etc.

4.2. Hardware Architecture

In the architecture diagram you can see how the hardware is connected to each other.

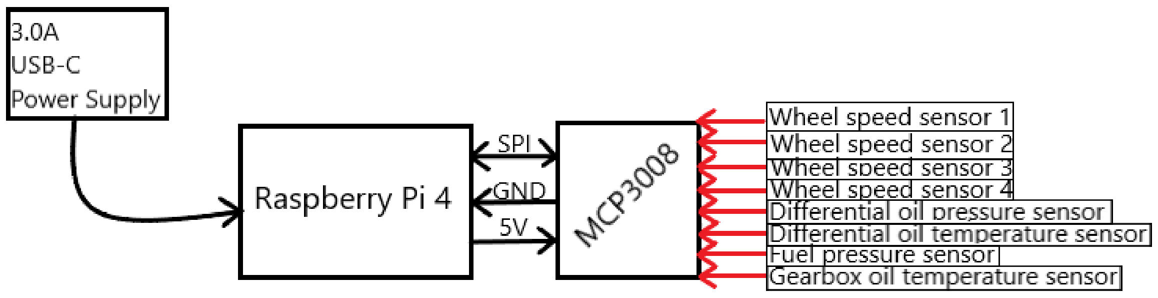


Figure 7: Hardware architecture diagram

On the left side of the diagram, you can see a 3.0A power supply (USB-C) that will be used to power the raspberry pi.

The raspberry pi is then connected to the MCP3008, an analog to digital converter, where it supplies power as well as uses SPI communication with the MCP3008 in order to receive the sensor data.

On the right-hand side, you can see all the sensors that are connected to the MCP3008, the red arrows represent the analog signals that are being sent to the MCP3008. The MCP3008 then converts these signals to a digital signal and then sends it to the raspberry pi.

In figure 7 you can find a software architecture diagram that will give further explanation as to what happens to the data when it is received by the raspberry pi.

4.3. Interfaces

Each interface's electrical and data transmission characteristics are explained in detail. While the decisions are shown and described, and a UML sequence diagram is created for each software driver's interaction with the microcontroller and other modules.

4.3.1. Power supply

The system has one power source. The first is 12V from the car. A low dropout (LDO) voltage regulator lowers this to 5V.

Specification:

The LDO must reduce the voltage of 12V to 5V. The maximum current is 50mA.

4.3.2. Wheel speed sensor

The wheel speed sensor is attached to the wheel to find out the speed of the car. It uses 12V to give digital signal.

Specification:

The sensor has 43 teeth, meaning a full rotation contains 43 changes to the voltage and the maximum rotation of the wheel is 30Hz, meaning at max rotations the number of changes becomes 2580Hz (43 * 30). The microcontroller needs to be fast enough to catch all the changes.

While the wheel rotates, the microcontroller will catch the signals and transform them into speed(m/s) within the program.

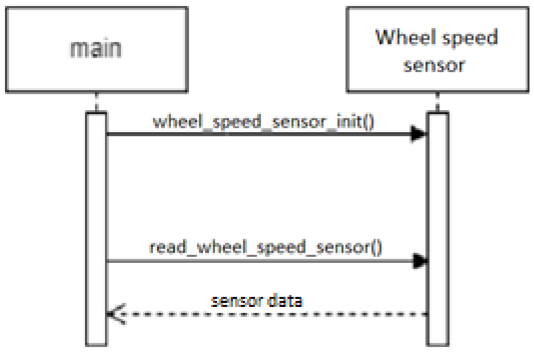


Figure 8: wheel speed sensor UML diagram

4.3.3. Differential oil temperature sensor

This sensor is for measuring oil temperature in differential. It uses 5V to give analog signal.

Specification:

The resistor changes to the temperature of the differential, giving a different current. It gives Celsius measure, but it can be changed to Fahrenheit. The ohm range will be specified later.

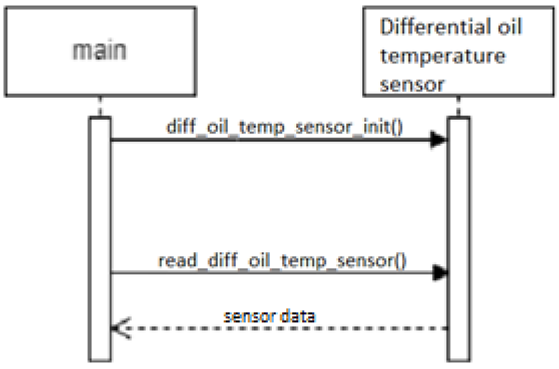

main
Differential oil temperature sensor
diff_oil_temp_sensor_init()
read_diff_oil_temp_sensor()
sensor data

Figure 9: differential oil temperature sensor UML diagram

4.3.4. Differential oil pressure sensor

This sensor is for measuring oil pressure in differential. It uses 5V to give analog signal.

Specification:

The resistor changes to the pressure of the differential, giving a different voltage, which results in change. It gives bar measurements. The ohm range will be specified later.

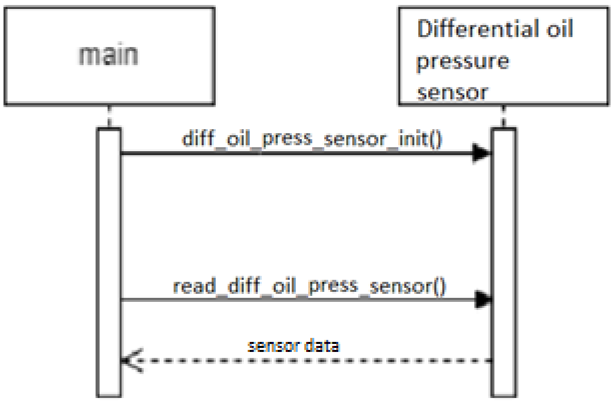

main
Differential oil pressure sensor
diff_oil_press_sensor_init()
read_diff_oil_press_sensor()
sensor data

Figure 10: differential oil pressure sensor UML diagram

4.3.5. Fuel pressure sensor

This sensor is for measuring oil pressure in the fuel tank. It uses 5V to give analog signal.

Specification:

The resistor changes to the pressure of the tank fuel, giving a different voltage. It gives bar measurements. The ohm range will be specified later.

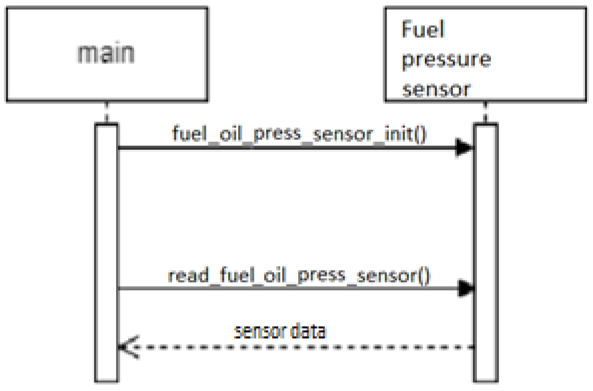


Figure 11: fuel pressure sensor UML diagram

4.3.6. Gearbox oil temperature sensor

This sensor is for measuring oil temperature in gearbox. It uses 5V to give analog signal.

Specification:

The resistor changes to the temperature of the gearbox, giving a different voltage. It gives Celsius measures, but it can be changed to Fahrenheit. The ohm range will be specified later.

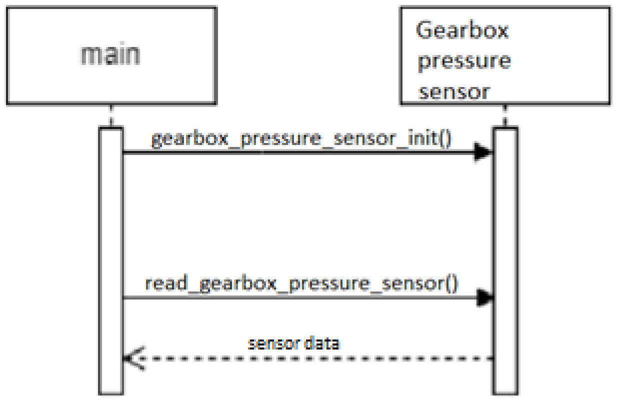

main
Gearbox pressure sensor
gearbox_pressure_sensor_init()
read_gearbox_pressure_sensor()
sensor data

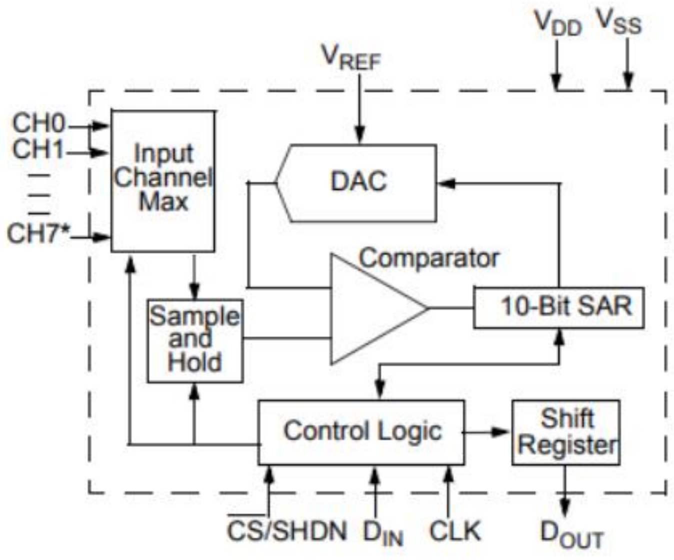
Figure 12: gearbox oil pressure sensor UML diagram

4.3.7. HAT IC MCP3008

Makes analog signals into digital signals.

Specification:

Compares the voltage reference to the voltage input, which then uses the formula Uout = 5/1023 * Uin. Which converts the sin waves into number ranging from 0 to 1024 to the raspberry pi.



4.4. Software

This is the Sequence Diagram for the code we will be making.

It is straightforward, car drives - sensors pick up data, send this to SQTT which will turn it into useful data. And send that Data to the database for analysis and to users that want to look at it live.

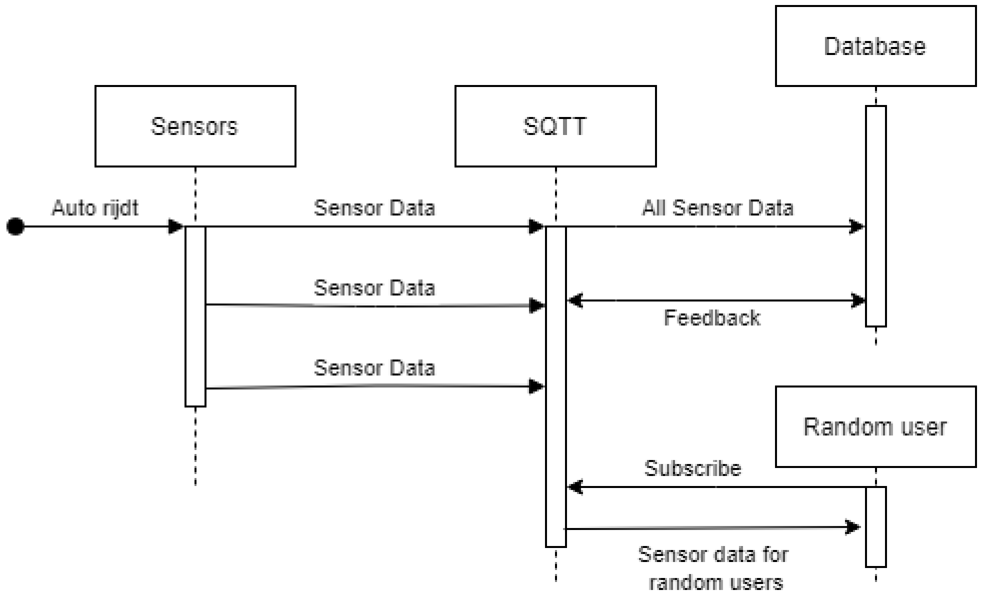


Figure 13: Software architecture diagram

The sensors constantly send new signals to the SQTT, the SQTT will constantly turn that into readable data and send that to 1(Or more) receivers.

The program is not one big loop but multiple smaller loops.

New data is recorded at 100Hz, so you can look back at precise data van analyzing the car’s performance. And this way the cloud and Database are not cluttered with excessive amounts of data.

5. Realisation

A Detailed explanation of both the hardware and software used for the project so far. But the biggest part of it is most certainly the coding

5.1 The hardware so far is limited to some sensors and a MCP3008, much more will be added to that eventually. But considering this period most of what we did is just getting the project on its feet, a lot of time was spend figuring out what works and what doesn't, what can we use and what can we not, etc.

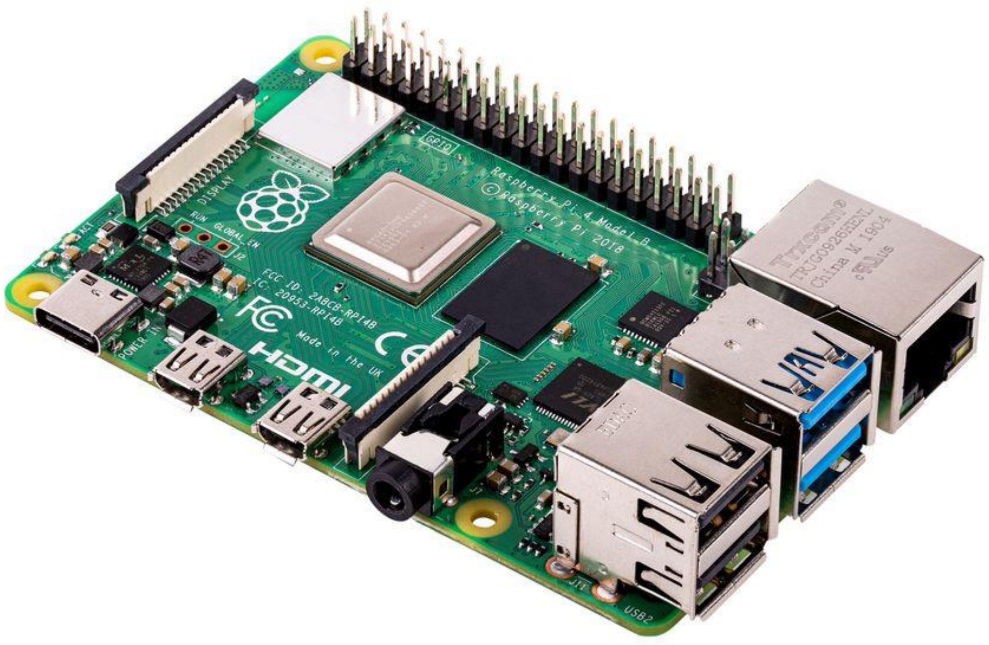
5.2 The software is explained through explanation mostly with some code snippets, the code is nothing if not huge and complicated and explaining in detail how every line of code works would be unnecessary and not to mention a huge amount to read so only some of the most important snippets will be brought attention to and explained.

5.1. Hardware

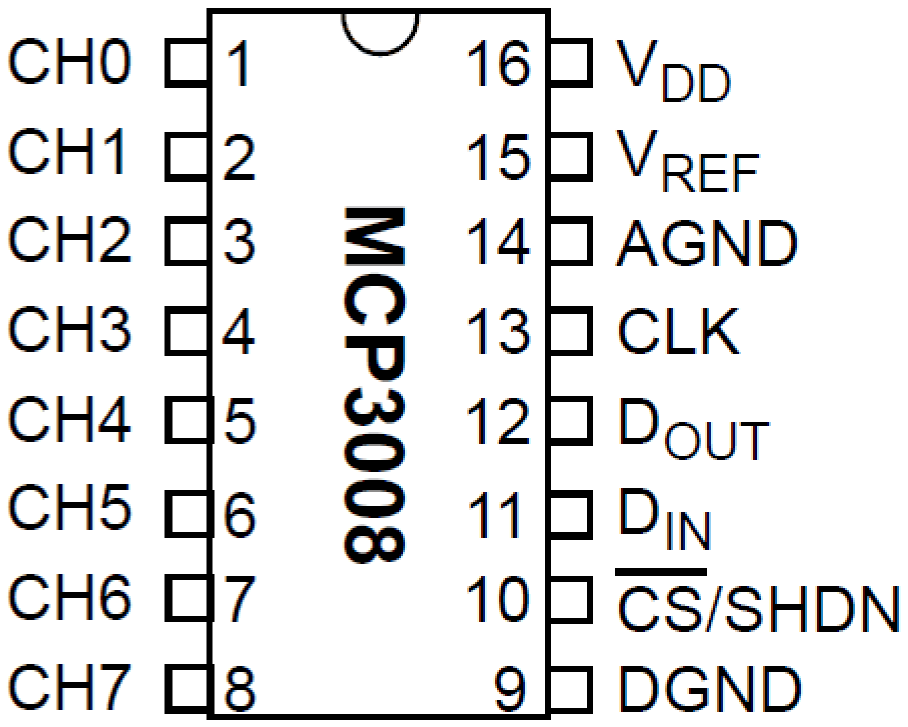
Here some of the hardware we use will be showcased, it is very little in comparison to the code as the project is still a year or more away from completion. And with us being the first to start to work on it most of it was preparation work, planning and coding.

5.1.1. Raspberry PI 4

This is the temporary Raspberry PI we will be using for this project, it is a borrowed one from one of the teachers so as the project progresses a different model might be used later.



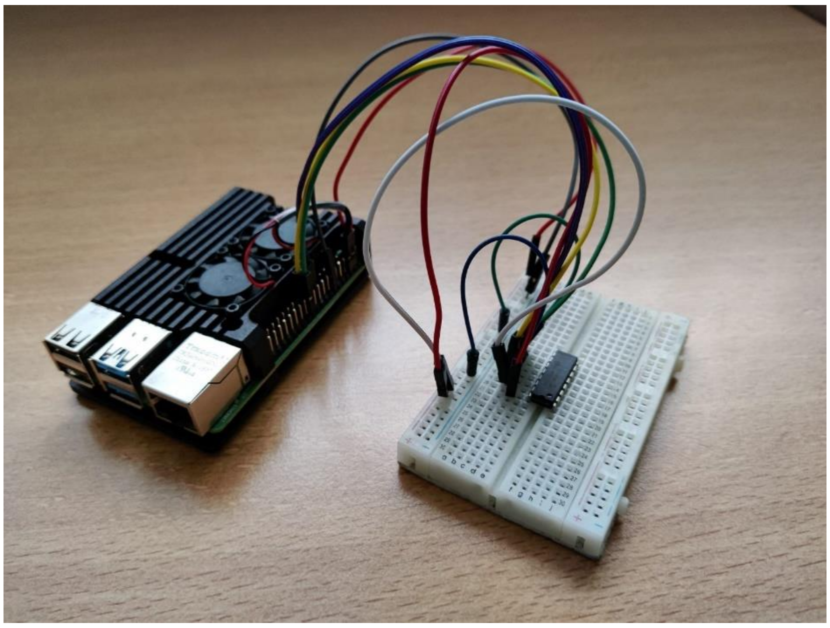
5.1.2. MCP3008

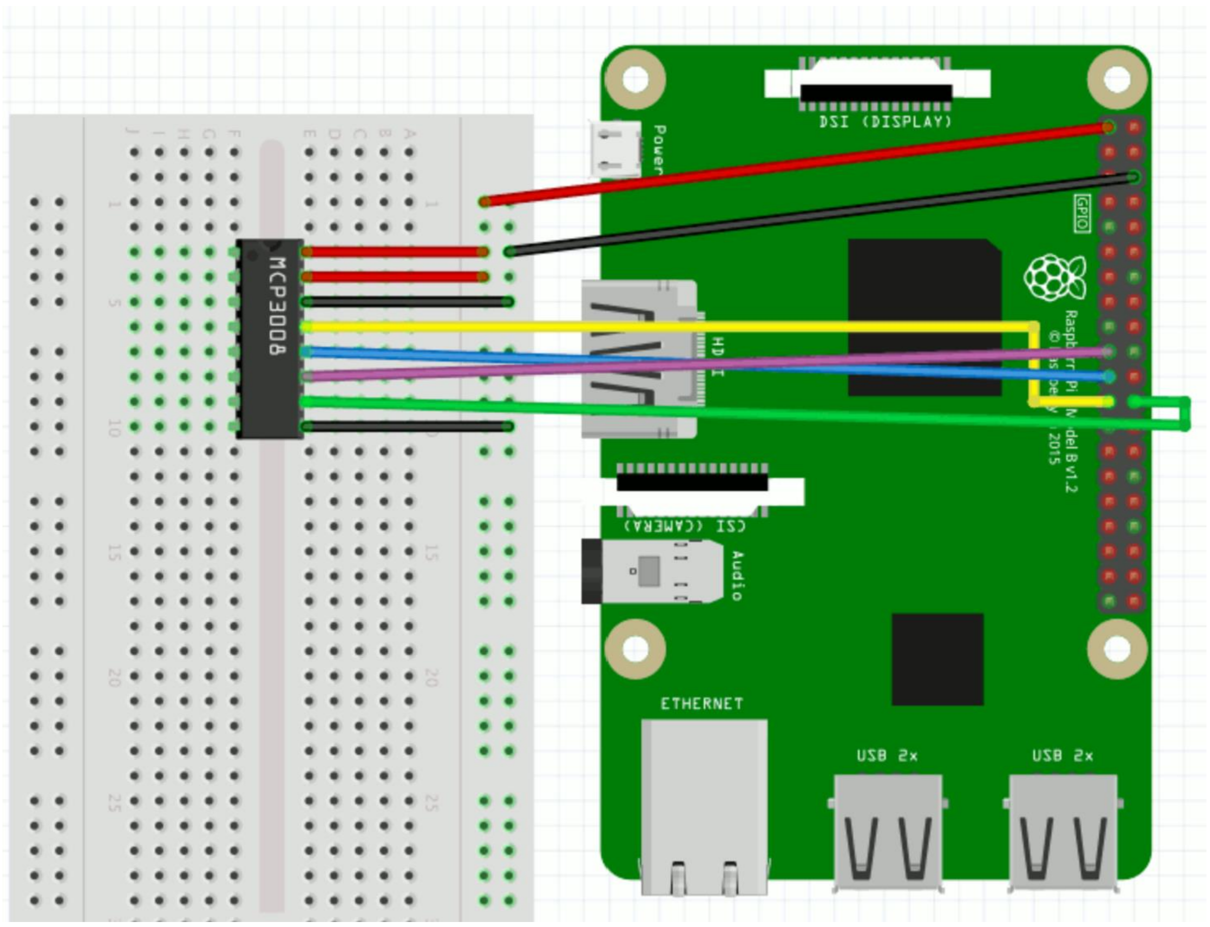


The MCP3008 is our Analog to Digital converter, ADC for short. The Raspberry PI cannot read analog signals directly, so it needs a filter of sorts to make it readable, this 'filter' is the MCP3008 who has 8 different channels to read 8 different sensors simultaneously

5.1.3. Setup

We made a simple setup to test the MCP3008 with the raspberry PI shown in the picture below, the pins are also shown



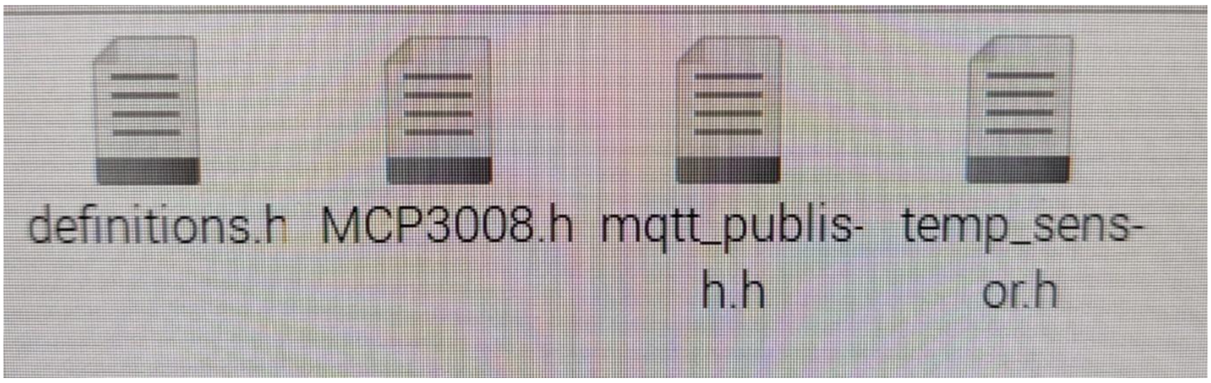


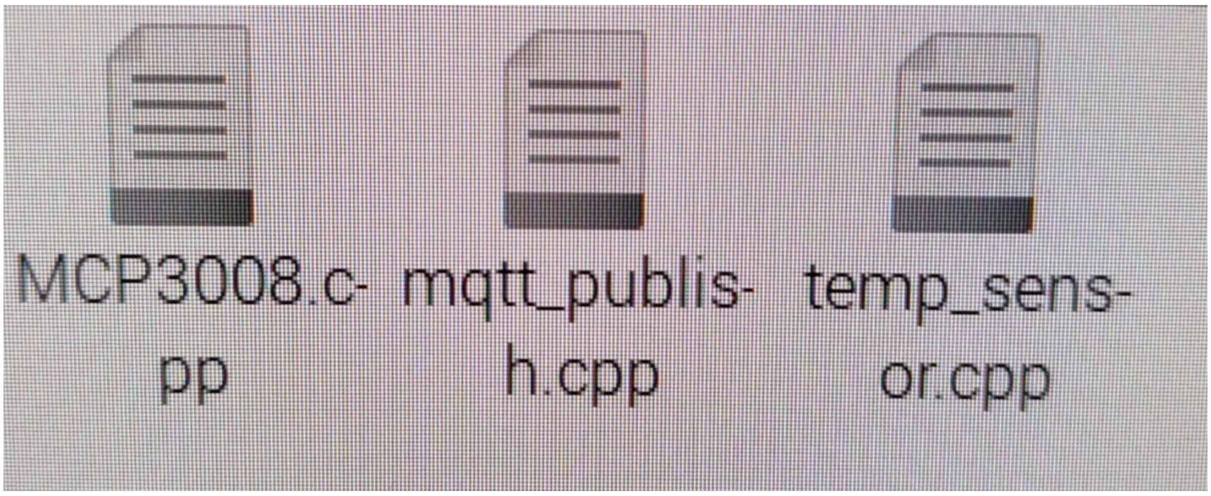
5.2. Software

Here multiple snippets of the code will be showcased and explained, mainly the code used to read the sensors, the code for the MCP3008 and for MQTT to send the data over to another computer.

5.2.1. Organization

The code is split up into multiple Header and CPP files to keep the code clean and easily readable.

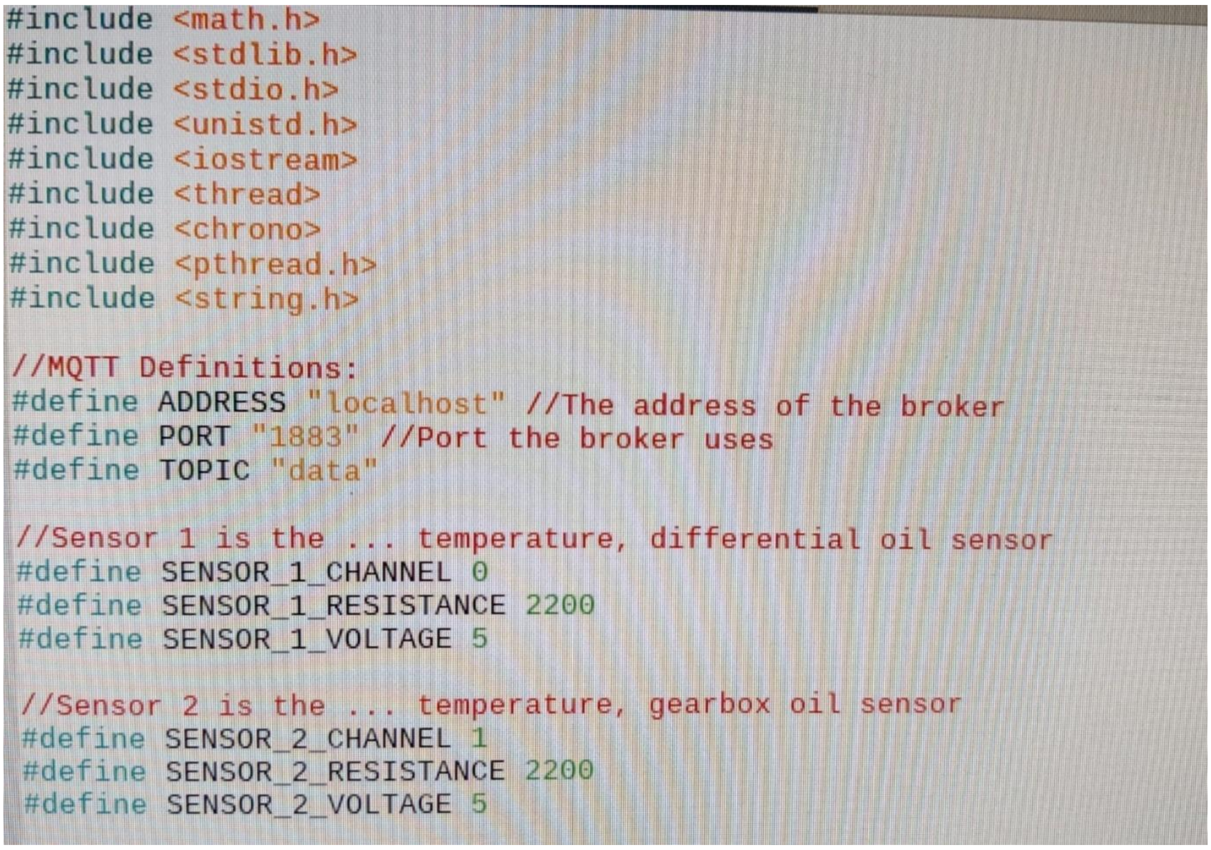




5.2.2. Definitions.h

The header file "Definitions.h" is of special note as this is the place where you type in the IP address of whichever device you wish to send the recorded data to.

Other info such as which sensor is attached to which specific channel, the voltage and the resistance of the sensor. This is all written like in the picture with only the first 2 sensors being shown as all 8 have the exact same format. Number, what sensor it is, and then the defines.

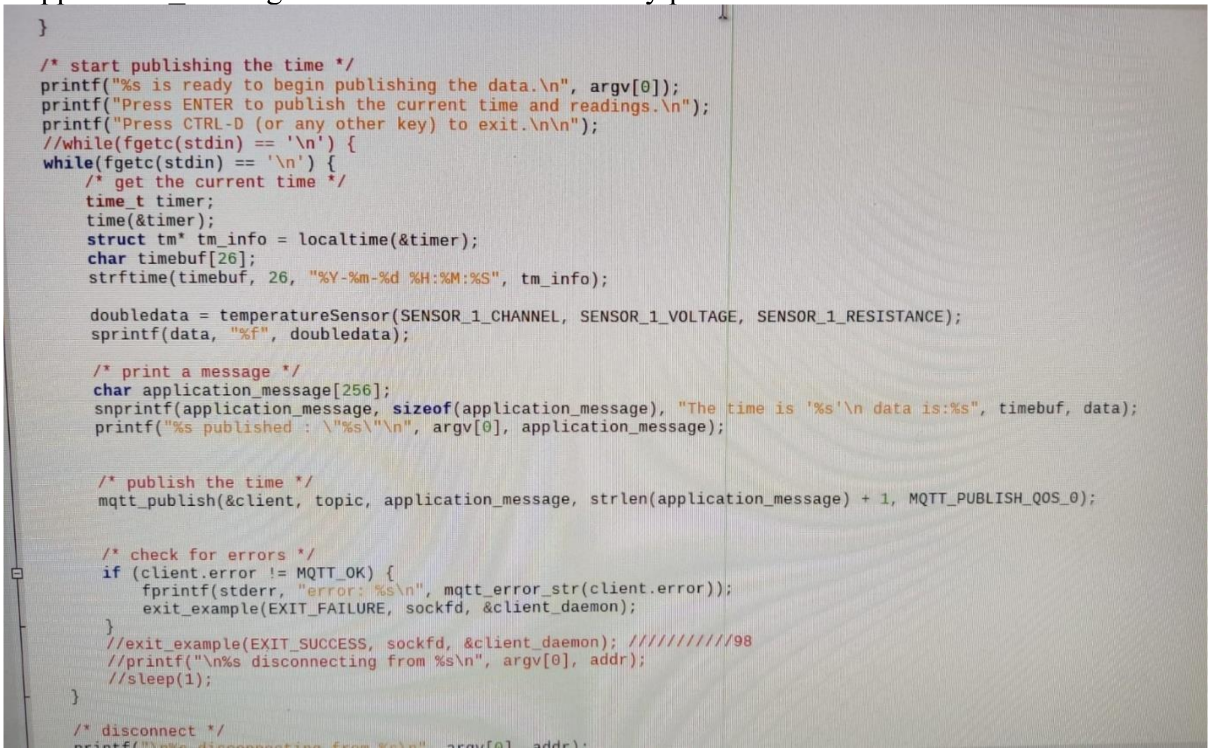


```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <iostream>
#include <thread>
#include <chrono>
#include <pthread.h>
#include <string.h>

//MQTT Definitions:
#define ADDRESS "localhost" //The address of the broker
#define PORT "1883" //Port the broker uses
#define TOPIC "data"

//Sensor 1 is the ... temperature, differential oil sensor
#define SENSOR_1_CHANNEL 0
#define SENSOR_1_RESISTANCE 2200
#define SENSOR_1_VOLTAGE 5

//Sensor 2 is the ... temperature, gearbox oil sensor
#define SENSOR_2_CHANNEL 1
#define SENSOR_2_RESISTANCE 2200
#define SENSOR_2_VOLTAGE 5
```

5.2.3. The message

This is the code that creates the message among some other things. Currently it only sends a message with every press of a button, this is for tests reasons as MQTT Is not fully setup on the actual server and we are still adding things to the code. This “press enter to publish” function came with the code we used when it was still unaltered and since we are still adding stuff to it and will be even more expanded upon, we deemed it better to just keep it for test reasons and change that part later as how quickly it publishes is not of any consequence for now and not for a while either.

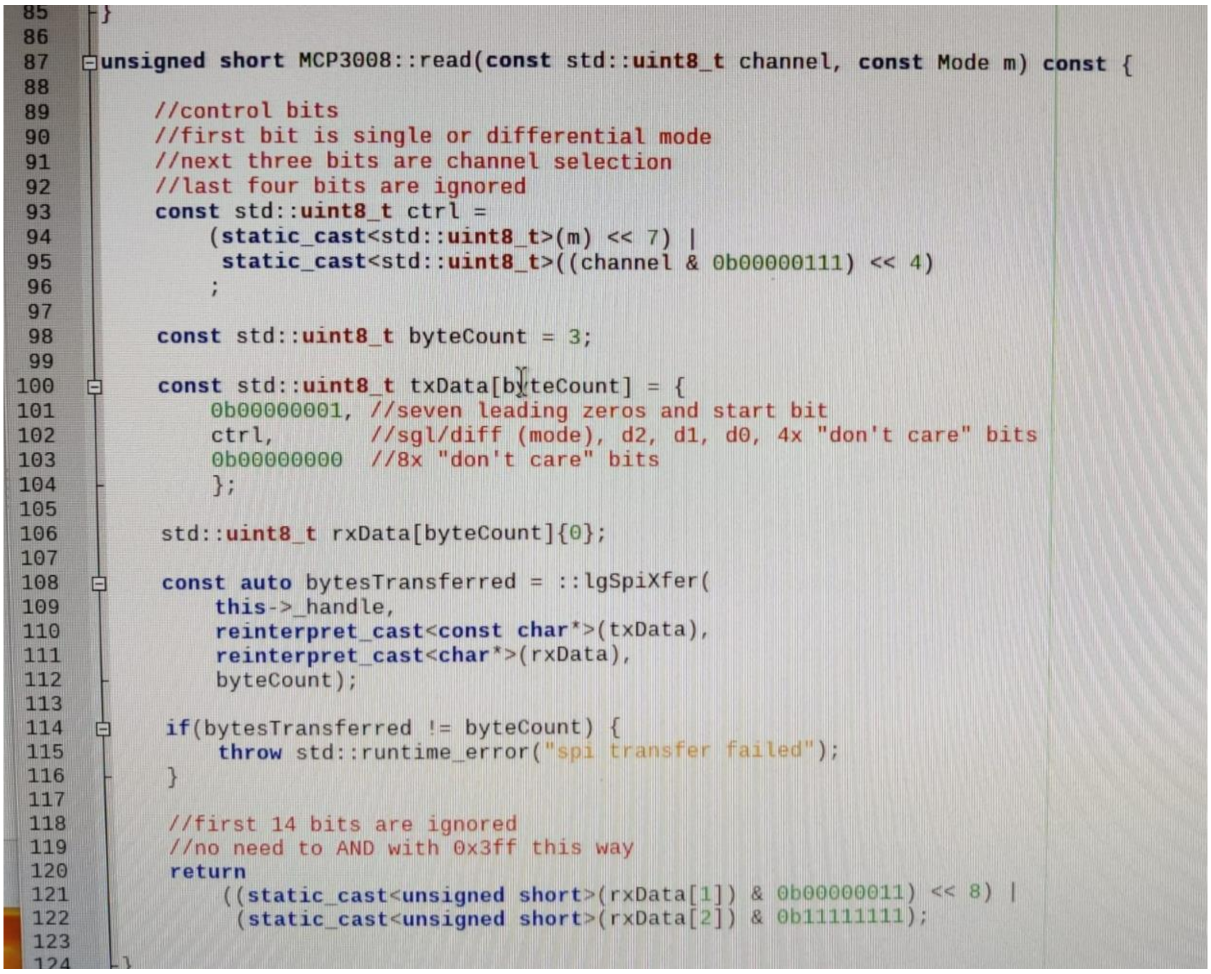
In the line with the red text of “Print a message” is where the message is made, starting with the current time which is in the string “Timebuf” followed by "data” which is the data recorded by the MCP3008, those two combines into the char “Application_Message” and that is send with every press of the “enter” button.



```
}

/* start publishing the time */
printf("%s is ready to begin publishing the data.\n", argv[0]);
printf("Press ENTER to publish the current time and readings.\n");
printf("Press CTRL-D (or any other key) to exit.\n\n");
//while(fgetc(stdin) == '\n') {
while(fgetc(stdin) == '\n') {
    /* get the current time */
    time_t timer;
    time(&timer);
    struct tm* tm_info = localtime(&timer);
    char timebuf[26];
    strftime(timebuf, 26, "%Y-%m-%d %H:%M:%S", tm_info);

    doubledata = temperatureSensor(SENSOR_1_CHANNEL, SENSOR_1_VOLTAGE, SENSOR_1_RESISTANCE);
    sprintf(data, "%f", doubledata);

    /* print a message */
    char application_message[256];
    snprintf(application_message, sizeof(application_message), "The time is '%s'\n data is:%s", timebuf, data);
    printf("%s published : \"%s\"\n", argv[0], application_message);


    /* publish the time */
    mqtt_publish(&client, topic, application_message, strlen(application_message) + 1, MQTT_PUBLISH_QOS_0);


    /* check for errors */
    if (client.error != MQTT_OK) {
        fprintf(stderr, "error: %s\n", mqtt_error_str(client.error));
        exit_example(EXIT_FAILURE, sockfd, &client_daemon);
    }
    //exit_example(EXIT_SUCCESS, sockfd, &client_daemon); ///////////98
    //printf("\n%s disconnecting from %s\n", argv[0], addr);
    //sleep(1);
}

/* disconnect */
printf("\n%s disconnecting from %s\n", argv[0], addr);
```

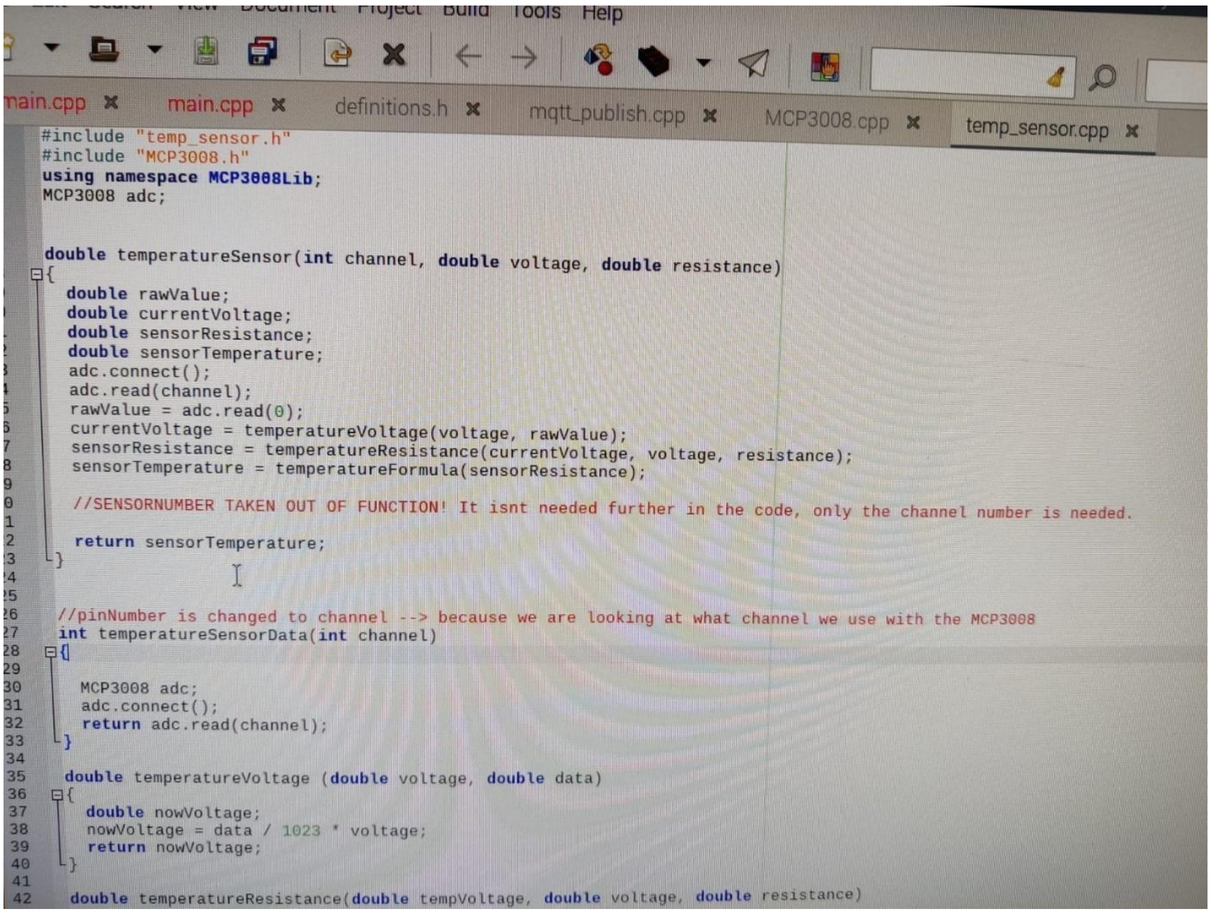
5.2.4. MCP3008 code

This is the code used to talk to the ADC converter, it reads the inputs and outputs that to the raspberry pi who can then make sense of the 0 to 1023 value it gives out. That value then goes to then goes to the temperature code to translate that number into relevant data that will be logged into the server.



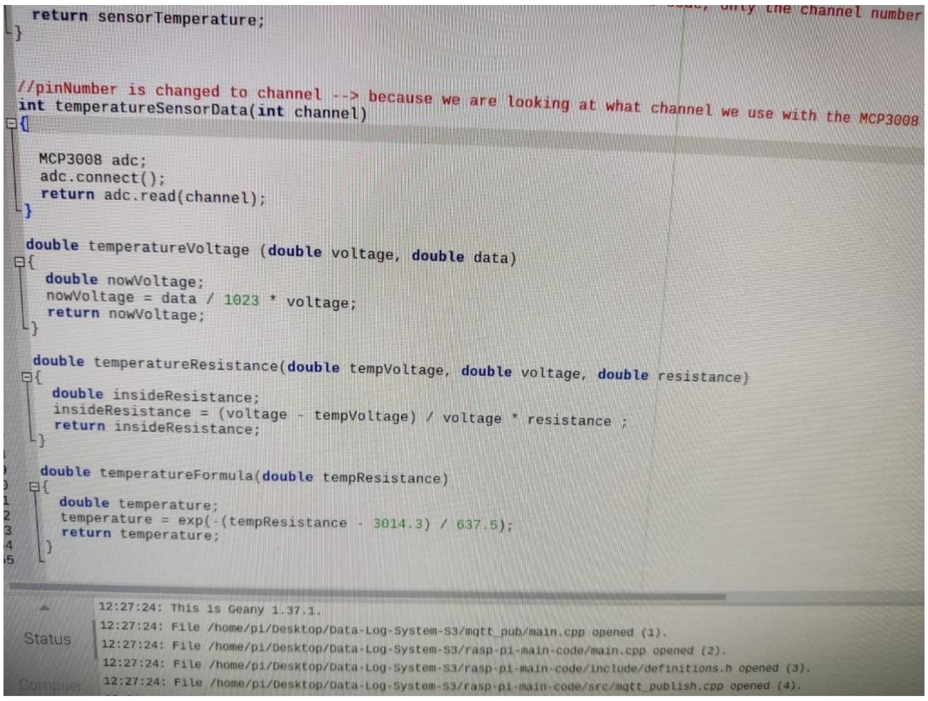
```
}

unsigned short MCP3008::read(const std::uint8_t channel, const Mode m) const {

    //control bits
    //first bit is single or differential mode
    //next three bits are channel selection
    //last four bits are ignored
    const std::uint8_t ctrl =
        (static_cast<std::uint8_t>(m) << 7) |
         static_cast<std::uint8_t>((channel & 0b00000111) << 4)
        ;

    const std::uint8_t byteCount = 3;

    const std::uint8_t txData[byteCount] = {
        0b00000001, //seven leading zeros and start bit
        ctrl,       //sgl/diff (mode), d2, d1, d0, 4x "don't care" bits
        0b00000000  //8x "don't care" bits
        };

    std::uint8_t rxData[byteCount]{0};

    const auto bytesTransferred = ::lgSpiXfer(
        this->_handle,
        reinterpret_cast<const char*>(txData),
        reinterpret_cast<char*>(rxData),
        byteCount);

    if(bytesTransferred != byteCount) {
        throw std::runtime_error("spi transfer failed");
    }

    //first 14 bits are ignored
    //no need to AND with 0x3ff this way
    return
        ((static_cast<unsigned short>(rxData[1]) & 0b00000011) << 8) |
         (static_cast<unsigned short>(rxData[2]) & 0b11111111);

}
```

5.2.5. Temp sensor code



```cpp
#include "temp_sensor.h"
#include "MCP3008.h"
using namespace MCP3008Lib;
MCP3008 adc;


double temperatureSensor(int channel, double voltage, double resistance)
{
    double rawValue;
    double currentVoltage;
    double sensorResistance;
    double sensorTemperature;
    adc.connect();
    adc.read(channel);
    rawValue = adc.read(0);
    currentVoltage = temperatureVoltage(voltage, rawValue);
    sensorResistance = temperatureResistance(currentVoltage, voltage, resistance);
    sensorTemperature = temperatureFormula(sensorResistance);

    //SENSORNUMBER TAKEN OUT OF FUNCTION! It isnt needed further in the code, only the channel number is needed.

    return sensorTemperature;
}


//pinNumber is changed to channel --> because we are looking at what channel we use with the MCP3008
int temperatureSensorData(int channel)
{

    MCP3008 adc;
    adc.connect();
    return adc.read(channel);
}

double temperatureVoltage (double voltage, double data)
{
    double nowVoltage;
    nowVoltage = data / 1023 * voltage;
    return nowVoltage;
}

double temperatureResistance(double tempVoltage, double voltage, double resistance)
```

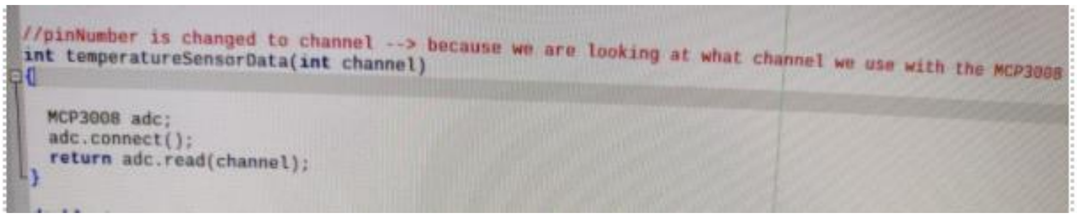
The temperatureSensor consists of multiple functions put together. Mainly temperatureSensorData, temperaturVolatge, temperatureResistance and temperatureFormula.



```cpp
    return sensorTemperature;
}

//pinNumber is changed to channel --> because we are looking at what channel we use with the MCP3008
int temperatureSensorData(int channel)
{

    MCP3008 adc;
    adc.connect();
    return adc.read(channel);
}

double temperatureVoltage (double voltage, double data)
{
    double nowVoltage;
    nowVoltage = data / 1023 * voltage;
    return nowVoltage;
}

double temperatureResistance(double tempVoltage, double voltage, double resistance)
{
    double insideResistance;
    insideResistance = (voltage - tempVoltage) / voltage * resistance ;
    return insideResistance;
}

double temperatureFormula(double tempResistance)
{
    double temperature;
    temperature = exp(-(tempResistance - 3014.3) / 637.5);
    return temperature;
}
```

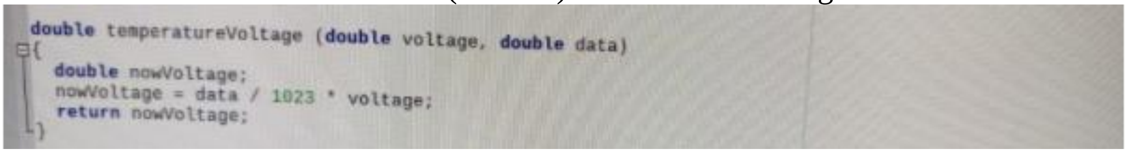
```
12:27:24: This is Geany 1.37.1.
12:27:24: File /home/pi/Desktop/Data-Log-System-S3/mqtt_pub/main.cpp opened (1).
12:27:24: File /home/pi/Desktop/Data-Log-System-S3/rasp-pi-main-code/main.cpp opened (2).
12:27:24: File /home/pi/Desktop/Data-Log-System-S3/rasp-pi-main-code/include/definitions.h opened (3).
12:27:24: File /home/pi/Desktop/Data-Log-System-S3/rasp-pi-main-code/src/mqtt_publish.cpp opened (4).
```

Here is an overview of all the functions



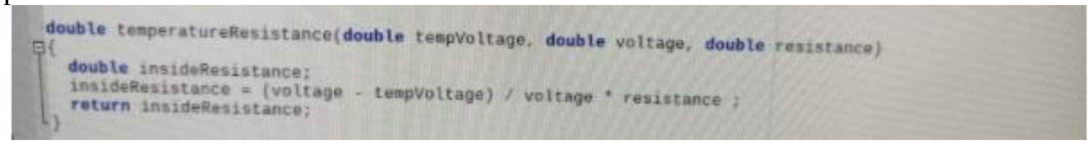
```
//pinNumber is changed to channel --> because we are looking at what channel we use with the MCP3008
int temperatureSensorData(int channel)
{

    MCP3008 adc;
    adc.connect();
    return adc.read(channel);
}
```

The function temperatureSensorData gets raw data from the sensor. But for now, we substituted this function with adc.read(channel) for smoother testing



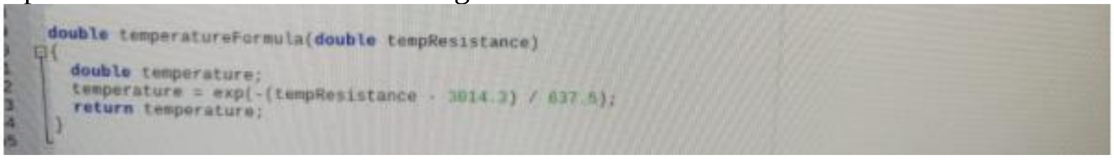
```
double temperatureVoltage (double voltage, double data)
{
    double nowVoltage;
    nowVoltage = data / 1023 * voltage;
    return nowVoltage;
}
```

TemperatureVoltage converts that raw value into voltage using the formula mentioned in the picture above.



```
double temperatureResistance(double tempVoltage, double voltage, double resistance)
{
    double insideResistance;
    insideResistance = (voltage - tempVoltage) / voltage * resistance ;
    return insideResistance;
}
```

TemperatureResistance converts voltage into resistance inside the sensor



```
double temperatureFormula(double tempResistance)
{
    double temperature;
    temperature = exp(-(tempResistance - 3014.3) / 637.5);
    return temperature;
}
```

And finally, temperatureFormula takes the resistance inside the sensor and makes it into temperature

6. Testing

This section will show the functions that are implemented and tested.

SMART Functional Specifications			
#	MosCoW	Description	Finish
F1	**M**	**The data logging system has a graphical user interface**	**Yes**
F1.1.	M	The GUI displays the data history in a table as well as a line graph	**Kind of**
F1.1.1.	M	By default, the history page displays the whole table for the specified sensor. The line graph then displays the X* most recent entries of that sensor.	**Kind of**
F1.1.2.	S	In the GUI history page, the user can specify the desired timeslot for the line graph and the table.	**No**
F1.2.	M	The GUI displays live data (X* most recent entries) in a line graph and shows the most recent entry on a speedometer	**Yes**
F1.3.	M	The user can use buttons to choose what sensor data they want to see	**Yes**
F1.4.	M	The GUI must be accessible for the racing engineers (PC Access) during the race	**Yes**
F1.5.	M	The GUI pulls the data from a remote database	**Yes**
F1.6.	C	The GUI must have a lap number attached to each data entry	**No**
F1.6.1	C	The user must be able to compare data according to the lap number	**No**
F2	**M**	**The raspberry pi logs incoming data from 5 types of sensors**	**Yes**
F2.1.	M	The system logs data from 4-wheel speed sensors (m/s)	**Yes**
F2.2.	M	The system logs data from 1 fuel pressure sensor (bar)	**Yes**

F2.3.	M	The system logs data from 1 gearbox oil temperature sensor (C)	**Yes**
F2.4.	M	The system logs data from 1 differential oil temperature sensor (C)	**Yes**
F2.5.	M	The system logs data from 1 differential oil pressure sensor (bar)	**Yes**
F2.6.	M	A research analysis will be held in order to determine how often each sensor will be logged, as well as the speed needed for the communication protocols (like MQTT)	**Yes**
F2.7.	M	Each data entry contains an ID (each entry gets its own ID number), timestamp (YYYY-DD-MM HH-MM-SS-MS) and measurement	**Yes**
F2.7.1.	C	Each data entry also contains lap number	**No**
F2.7.2.	C	Each data entry also contains geo coordinates	**No**
F3	**M**	**The raspberry pi sends all sensor data to a remote database**	**Yes**
F3.1.	M	The raspberry pi correctly processes the incoming sensor data and sends it to the database.	**Yes**
F3.2.	M	Database is located on a remote server	**Yes**
F3.3.	C	The database can be accessed using 4G	**No**

6.1. Data Logging system testing

In this section we will test the functionality of the data logging system with the table above.

Number	Function	Testing	Result
F1	Data logging system has a GUI	GUI is activated	GUI is functional and visible
F1.1	GUI showing a history page	Check if the history page works	Rapid prototyping has this feature, but due to time restrictions it will not be implemented to the product
F1.1.1	GUI history page showing past entries	Activate the GUI history page and show the entries	Rapid prototyping has this feature, but due to time restrictions it will not be implemented to the product
F1.1.2	GUI history page can show the desired time slot	Activate the GUI history page and select the time slot	Due to time restrictions this has not been implemented
F1.2	The GUI displays live data	Activate the GUI and see the live data	Live data is seen and shows when it was received
F1.3	Buttons can be used to change sensors	Activate the GUI and press the buttons to change the sensors	GUI buttons are visible and change the sensors
F1.4	GUI can be accessed by another computer	Two computers will try accessing the GUI at the same time	Two computers successfully connected to the GUI
F1.5	GUI can pull data from the remote database	Have the GUI pull the data from another computer	The GUI successfully pull the data from another computer

F1.6	The GUI has lap data	Check if GUI has lap data in it	Nothing happened as this function has not been implemented yet
F1.6.1	The GUI can compare the lab data	Compare the lap data	Nothing happened as this function has not been implemented yet

6.2. Raspberry Pi testing

In this section we will test Raspberry Pi.

Number	Function	Testing	Result
F2	The raspberry pi can interpret data from three different sensors	Three sensors are run and send data to raspberry pi	Three sensors can successfully send data, and the raspberry pi can receive the data
F2.1	Four wheel speed sensors can send data to the raspberry pi	The wheel sensor is run and send data to raspberry pi	The sensor successfully sends data, and the raspberry pi can receive the data
F2.2	The pressure sensor can send data to the raspberry pi	The pressure sensor is run and send data to raspberry pi	The sensor successfully sends data, and the raspberry pi can receive the data
F2.3	The pressure sensor can send data to the raspberry pi	The pressure sensor is run and send data to raspberry pi	The sensor successfully sends data, and the raspberry pi can receive the data
F2.4	The temperature sensor can send data to the raspberry pi	The wheel sensor is run and send data to raspberry pi	The sensor successfully sends data, and the raspberry pi can receive the data
F2.5	The temperature sensor can send data to the raspberry pi	The wheel sensor is run and send data to raspberry pi	The sensor successfully sends data, and the raspberry pi can receive the data
F2.6	The research of individual speed of sensors and the speed needed MQTT	Research using the sensors and GUI	The sensors and the GUI has enough speed to keep up with raspberry pi
F2.7	Each data contains an ID, timestamp, and measurement	Have the sensors run and check the GUI	The GUI displays ID, timestamp, and

			measurement correctly

6.3. Raspberry Pi and Database testing

In this section we will test Raspberry Pi with the database.

Number	Function	Testing	Result
F3	Raspberry Pi can send all data to the database	Have the raspberry pi run with the database and see if all the data goes into the database	The data arrives to the database
F3.1	The raspberry pi can interpret the incoming data and process it	The raspberry pi is run with sensors and database	Raspberry pi can successfully interpret the data and sends it to the database correctly
F3.2	The database can be accessed remotely	Have two computers connect to each other's database	The computer can access another computer's database
F3.3	The database can be accessed using 4g	Have two computers connect to each other's database using 4g	Nothing happened as this function has not been implemented yet

7. Conclusions and recommendations

The project's primary goal was to research and prototype the database and numerous automobile sensors. Computers ranging from an Arduino to the Raspberry Pi may be used. It is also required to be remotely accessible from other PCs. The project's objective was accomplished, and several sensors and the prototype database - are operating as expected.

Due to its adaptability, the Raspberry Pi was ultimately chosen. Nevertheless, the team concluded that it could not withstand high stress and that a new computer must be purchased and installed in the automobile. The wires are the same - they are too delicate to be used on public roads.

The MQTT protocol is used by the data logging system to convey data. It was determined that while it was quick enough for the prototype, further testing with more than ten sensors will be required.

The prototype may be used with the current GUI, even though a new one created by the programming department will eventually replace it.

Overall, the prototype was successful, but the parts need to be updated to seek usefulness in the automobile.

8. References

7 Advantages of MQTT protocol for IoT Devices - esperso. (2021, December 13). Medium. https://medium.com/@esperso/7-benefits-of-mqtt-protocol-for-iot-e463f6a97100

Cope, B. S. (2022, July 6). *Beginners Guide To The MQTT Protocol*. |. http://www.steves-internet-guide.com/mqtt/

Dev As Pros. (n.d.). *C++ vs Node.js | What are the differences?* StackShare. https://stackshare.io/stackups/cplusplus-vs-nodejs

Hübschmann, I. (2022, March 22). *The Pros and Cons of Using MQTT Protocol in IoT*. Nabto. https://www.nabto.com/mqtt-protocol-iot/

Node-RED. (n.d.). https://nodered.org/

«Node-RED: from A to Z»: Advantages and Opportunities. (2022, July 14). 42FLOWS.TECH. https://www.42flows.tech/blog/node-red-from-a-to-z-advantages-and-opportunities/

Raspberry Pi Datasheets. (n.d.). https://datasheets.raspberrypi.com

Wayback Machine. (n.d.). https://web.archive.org/web/20180324192118/http://benchmarksgame.alioth.debian.org/u64q/compare.php?lang=node

Appendix A – POA Rapid Prototyping

Background

This project is from the company Regterschot Engineering, they gave us the task to build a data-logging system for their race car. As you might think this project is a mountain of work so in order to start, we are going to start with rapid prototyping a small part of what the final product must be. This Plan of Action is the outline the rapid prototyping. During our rapid prototyping we are simultaneously researching the differences between using a microcontroller and microcomputer. We are going to use an Arduino as well as a Raspberry Pi. At the end of the rapid prototyping, we will make another POA for the rest of the semester project because then we will have a better idea of what the project entails.

Project Result

By the end of the rapid prototyping, we should have 1 sensor (an upright car sensor) connected to one of the boards and then read the analog values. These values must then be logged and then read from a simple GUI. Once we have made this, we will have a small demonstration for the automotive students so that they can get a good idea of what we are working on.

Quality

Basics, not to extreme

Our goal is to Build the first few steps for a much bigger system which will be created by students that come after us. So, our first and foremost priority is making one sensor work as fully intended and make it so the next students can easily pick up where we left off.

Modulaire coding

To make it not only easier for us but also for the any future students, we'll be making the code Modulaire. Meaning that we don't have to make separate code for every sensor and function.

Testing for full functionality

Making sure the software works fully, and doesn't crash or stop working fully after prolonged use

Anticipate potential issues.

Largely connected to Modulaire coding. Using clear Variable names so you can tell what serves what purpose at first glance.

Project Organization

ESE students were allocated to the project for the business Regterschot Engineering. Hugo Arends, a lecturer at Han university, is the mentor. Erik Regterschot (info@regtershotracing.com) is the team manager. Mika Tuijp (m.tujip@regtershotracing.com) is the technical manager. Dana Groner (Dj.groner@student.han.nl) is the ESE manager.
These are the ESE members:
Gijs Dreijling (659934)
Email: G.Dreijling@student.han.nl
Leon Nguyen (670876)
Email: L. Nguyen4@student.han.nl
Hugo Arends will request project updates at the beginning of each week.

Planning

This plan of action is made specifically for the rapid prototyping phase and will only consist of the first 3 weeks of the project if everything goes to plan.

Week 1	• Introduction to the project • Start POA for Rapid prototyping • Start Rapid Prototyping?
Week 2	• Meet with Hugo (Monday) • Work on rapid prototyping
Week 3	• Meet with Hugo (Monday) • Finish rapid prototyping • Demo for AUM students (Tuesday)

Cost and Benefits

Here you can find the pros and cons between a raspberry pi microcomputer and an Arduino microcontroller. Things will most likely be added to this list as we work on them.

Raspberry Pi

Pros

- Since it supports an operating system, It can perform complex operations like Weather monitoring, Controlling robots, etc.
- You can use it as a portable computer because it has everything- from CPU (Central Processing Unit) to ethernet port and Wi-Fi support.
- It has many GPIO (General-Purpose Input/Output) pins (the famous model of Raspberry Pi has 40 GPIO pins). Therefore, it can support many sensors.
- It has superior processing power. The 4 B variant of Raspberry Pi comes with a 1.6 GHz processor.
- It can run all kinds of applications (including MS Office and Email).

Cons

- Raspberry Pi's hardware and software are closed source. It means that you cannot customize your own Raspberry Pi single-board computer (SBC).
- Raspberry Pi does not have any internal storage; it requires a micro-SD card to work as internal storage.
- It sometimes overheats during heavy operations.

Arduino

Pros

- Both hardware and software of Arduino are open source. You have the liberty to select from the codes already available or you can customize your own Arduino board.
- It is less expensive than Raspberry Pi.
- It is good for beginners as it is easy to learn and use.
- It is quite easy to program Arduino through IDE (Integrated Development Environment).
- Arduino has a huge community and a wide range of applications.

Cons

- It has very less processing power when compared to Raspberry Pi.
- Arduino boards do not support internet and wireless connectivity.
- 8-bit CPU architecture
- Arduino is incapable of performing complex tasks.

In the end, the raspberry pi is superior to the Arduino because of its faster processing, internet access, support for a variety of sensors, and capacity for complicated tasks. Arduino works well for routine chores like turning on and off lights, opening and shutting doors, but it is not suited for our project.
(Source: https://www.interviewbit.com/blog/arduino-vs-raspberry-pi/#:~:text=power%20source%20connector.-,Key%20Differences,IDE%20(Integrated%20Development%20Environment).)

Appendix B – POA Project

Background

Erik Regterschot is the creator of this project and runs the business Regterschot Engineering. The team is in the MIC building at IPKW (Industrie Park Kleefse Waard).

Making a foundational prototype for the data log system for the racing car was the group's allocated assignment. A functional sensor that is providing data to the data logging system is required. Any sensor that is utilized in a future car could be used for this purpose. Data must be received by the data log system and shown on the GUI during the race. The group has complete discretion over its decisions and actions, but everything must be documented. The steps required to accomplish the objective specified by the group manager are outlined in this plan of approach.

Project Result and Boundaries

Result

The planned result of this project from the Embedded Systems part is to have a Data Log System up and running that will receive and log any data send from different sensors from the car, this includes wheel speed sensors, GPS data, fuel pressure sensor, gearbox oil temperature, differential oil temperature sensor as well as differential oil pressure sensor.

We will be creating said Data Log System and write a program to work with these sensors, send it to a remote database and make a GUI to read that information easily.

Boundaries

Our Specific task is to build the Data Log System and have one sensor work fully with it. The Data Log System doesn't need to be fully fleshed out beyond that purpose but must be made in a way where others can continue where we left off. Many more sensors will eventually be added so we need to make the code in a way that it's easily added to. See 2.3. Functional Requirements to see what requirements must be met at the end of the semester.

Functional Requirements

SMART Functional Specifications		
#	MosCoW	Description
F1	**M**	**The data logging system has a graphical user interface**
F1.1.	M	The GUI displays the data history in a table as well as a line graph
F1.1.1.	M	By default, the history page displays the whole table for the specified sensor. The line graph then displays the X* most recent entries of that sensor.
F1.1.2.	S	In the GUI history page, the user can specify the desired timeslot for the line graph and the table.
F1.2.	M	The GUI displays live data (X* most recent entries) in a line graph and shows the most recent entry on a speedometer
F1.3.	M	The user can use buttons to choose what sensor data they want to see
F1.4.	M	The GUI must be accessible for the racing engineers (PC Access) during the race

F1.5.	M	
F1.6.	C	The GUI pulls the data from a remote database
F1.6.1	C	The GUI must have a lap number attached to each data entry The user must be able to compare data according to the lap number
F2	**M**	**The raspberry pi logs incoming data from 5 types of sensors**
F2.1.	M	The system logs data from 4-wheel speed sensors (m/s)
F2.2.	M	The system logs data from 1 fuel pressure sensor (bar)
F2.3.	M	The system logs data from 1 gearbox oil temperature sensor (C)
F2.4.	M	The system logs data from 1 differential oil temperature sensor (C)
F2.5.	M	The system logs data from 1 differential oil pressure sensor (bar)
F2.6.	M	Each data entry contains an ID, timestamp and measurement
F2.6.1.	C	Each data entry also contains lap number
F2.6.2.	C	Each data entry also contains geo coordinates
F3	**M**	**The raspberry pi sends all sensor data to a remote database**
F3.1.	M	The raspberry pi correctly processes the incoming sensor data and sends it to the database.
F3.2.	M	Database is located on a remote server
F3.3.	C	The database can be accessed using 4G

Project Activities and Intermediate Results

In order to achieve this result we will have weekly meetings with Hugo Arends as well as write weekly progress reports. We are trying to keep everything documented in order to make our project reproduceable. This also makes it easier to look back on what decisions we made as well as why we made them.

Intermediate results of this project are for one the rapid prototyping. Here we quickly take 1 sensor, in this case a wheel speed sensor, and read the values with an Arduino and then send these values to a data logging system, and then display it on a user interface. In this case we are using Node-Red to create this. After this we will slowly add the rest of the sensors one by one to the system.

Project Organization

We communicate through multiple ways. The first way is through a weekly meeting including the 3 ESE students, Eric and Hugo Arends every Monday morning. Following this we also have a Teams, GitHub and a WhatsApp group we use to send files and messages to each other.

Mainly during the meetings, we talk about our plans that week. Our progress from last week is brought up, any questions we have and our current status. This way we receive feedback from either Hugo or Eric on the spot and can immediately adapt our plans to it.

Planning

We have a Dynamic schedule we use to plan what we do week by week; it’s made in a way where it’s easily adaptable and can be looked at by the Automotive students who communicate with us regarding the electronics in the car.

This dynamic schedule can be found in teams and is also translated to the teams dashboard to keep up with who is doing what as well as deadlines. In the teams dashboard you can see all the tasks ahead of us as well as who is currently working on what. It can go into more depth compared to the dynamic schedule.

Risks

During this project there are things that could go wrong such as a member getting sick or our group falling behind schedule. For most cases we can discuss that as the issue comes up. We have 2 planned dates to work on the project but, if need be, we can work on it on different days as well.

We will take the project week by week and take any problems that come one at a time. We have each other to lean on in case one of us gets sick or to just help each other out. Falling behind schedule is something that will happen eventually, however we are prepared to work in our own time if needed.