CODEN:LUTEDX/(TEIE-3109)/1-40/(2022)

Industrial Electrical Engineering and Automation

Development and implementation of a data collection system for temperature and climatic test systems

Marcus Sandell

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

Development and implementation of a data collection system for temperature and climatic test systems

Bachelor's Thesis

By: Marcus Sandell

Supervisor: Anton Karlsson, LTH & Fredrik Nilsson, BorgWarner

Examiner: Jörgen Svensson, LTH

Abstract

A large part of developing products is the testing and validation of the new products. This process requires expensive test systems and a great deal of time and space. Because of these factors it is very important to maximize the usage of the test systems. Therefore, a monitoring system has been developed by the test and validation department at BorgWarner Landskrona that can read the operational status of some of their test systems. This system collects the status data via a single-board computer and then logs status changes in a database over the local network. At the end of each day the status data is compiled and the total runtime of each status is displayed in the office. This thesis consists of expanding this monitoring system to include a type of test system called temperature and climatic test systems. These test systems are designed to subject the device under test to very high or very low temperatures and to quickly be able to change between the two. The temperature and climatic test systems have been identified as a bottleneck and therefore BorgWarner wants to gather status data from these test systems to be able to know if more test systems need to be purchased or if a change in usage would alleviate the bottleneck.

The data collection system developed during this thesis uses a single-board computer running a Python script. The python script contains the IP-address, port number and alias of the test systems it has been designated. When the script is started it connects to the local database and then gets the latest status for each alias. The script then cycles through the IP-addresses sending commands over the network and the test systems respond with their status. This status is compared to the last saved status and if they are different the new status is logged in the database.

Some improvements and changes are to be made but the data collection system works as intended and is in use. More time is needed for the data collection until conclusions can be drawn to ensure that the data is representative of the real-world long-term usage.

Keywords

Raspberry Pi, Python, Vötsch, MySQL

Sammanfattning

En stor del inom utvecklingen av nya produkter är test och validering. Denna process kräver användning av dyra och stora testsystem. På grund av dessa faktorer är det viktigt att maximera användningen av dessa testsystem. Till följd av detta har ett övervakningssystem utvecklats av test- och valideringsavdelningen på BorgWarner i Landskrona som kan läsa driftstatus (Till exempel på och av) av vissa typer av test system. Övervakningssystemet samlar statusdata via en enkortsdator som sedan loggar statusdata i en databas över det lokala nätverket. I slutet av dagen så sammanställs all data för varje testsystem och visas på en TV i kontoret. Detta examensarbete består av att expandera det existerande övervakningssystemet till att inkludera så kallade temperatur- och klimat-testsystem. Dessa testsystems uppgift är att utsätta enheten under test för mycket höga eller mycket låga temperaturer och att snabbt kunna växla mellan dessa. Temperatur- och klimat-testsystemen har identifierats som en flaskhals i test-och-valideringsverksamheten och BorgWarner vill därför samla data angående användningen av dessa testsystem för att kunna avgöra om flera testsystem behöver köpas eller om en ändring i användningen skulle kunna lätta på flaskhalsen.

Datasamlingssystemet som utvecklats under detta examensarbete använder sig av en enkortsdator som kör ett python-program. Python-programmet innehåller IP-adress, port nummer och alias till alla testsystem som enkortsdatorn har tilldelats. När programmet startas kopplar den upp sig mot en lokal databas och hämtar den senaste statusen för varje tilldelat testsystem. Programmet roterar igenom IP-adresserna och skickar ett förformaterat medelande till var och en, och testsystemen svarar med sin nuvarande status. Om den nuvarande statusen skiljer sig från den förra så loggas detta i databasen.

Vissa ändringar och förbättringar skall göras men datasamlingssystemet fungerar som specifierat och är för nuvarande i användning. Mer tid behövs för att kunna dra några slutsatser från datan för att säkerställa att datan är representativ av den riktiga långsiktiga användningen.

Nyckelord

Raspberry Pi, Python, Vötsch, MySQL

Acknowledgements

Thank you to all employees at the test and validation department at BorgWarner Landskrona for taking their time to show me how things work and being quick and eager to help if there were any questions. A special thanks to Fredrik Nilsson for taking on the role of supervisor at BorgWarner and Karl Hedelin for providing the resources and time for me to understand the previous systems.

Thank you to Anton Karlsson for taking on the role of supervisor and to Jörgen Svensson for taking on the role of assessor at Lund University's Faculty of Engineering.

Table of Contents

1. Introduction

1.1. Background

The electrification of the modern world promises to revolutionize almost all sectors of industry. This poses big challenges but also big opportunities for innovation. One of the sectors most impacted by this is the automotive industry, and one of the companies working on this is BorgWarner. BorgWarner employs about 49 000 people worldwide and has offices in 22 countries. The BorgWarner production plant in Landskrona develops and produces driveline products and systems for hybrid, electric and internal combustion engine vehicles. BorgWarner delivers products to some of the largest vehicle producers in automotive, such as Volkswagen, Jaguar Land Rover (JLR), Volvo, and are themselves one of the largest producers of four-wheel drive systems in the world. An example of the driveline systems partially developed in Landskrona can be seen in Figure 1. These driveline systems use a coupling device to switch from either front-wheel drive (FWD) or rear-wheel drive (RWD) to all-wheel drive (AWD) [1]. A large portion of BorgWarner's product and development is located at the Landskrona site.

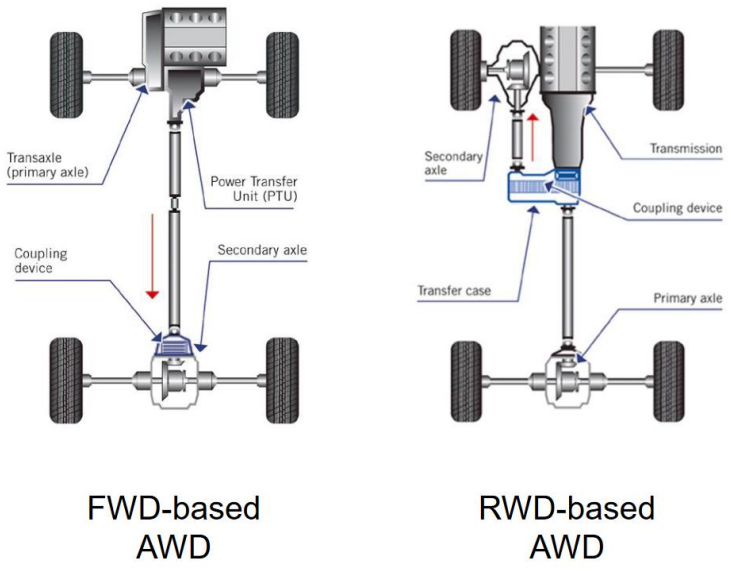

Figure 1: Showing two AWD drivelines, the left one being FWD based and the right one being RWD based. A coupling device is used to engage or disengage the AWD.

An important part of the product development at BorgWarner is test and validation. The products under development are tested to ensure quality and reliability. The testing includes tests such as high voltage testing, dynamometers and temperature shock degradation. The test equipment's job is either to exceed projected real-world usage of a product, through wear or environment, to ensure a product will perform as expected by both producer and client as well as to validate and calibrate products, such as the braking strength of a regenerative braking system. When a product has gone through a test, the attribute under test, whether it be the flow rate of a pump, rotational friction of a ball bearing or anything else, is measured. If the results are unsatisfactory this might lead to a design change to better cope with whatever real-world situation the test is emulating.

The tests can be time intensive, and the test equipment might be expensive, thusly it is important to maximize the usage of the test equipment. It is therefore of value to monitor the usage of the test equipment that play a central part of the test and validation operation at the Landskrona site. A monitoring system to log some of the test equipment activity has therefore previously been developed by the test and validation team some years ago. The test equipment is connected to one Raspberry Pi each. A Raspberry Pi is a small, single-board computer that is often used in product development due to its compact size, networking capabilities and software defined General Purpose In-Out (GPIO) pins. How the Raspberry Pi is connected is dependent on what output signals the test equipment can provide and what output signals are relevant. Some test equipment with an integrated computer can report their own status (for example, if pin one is high a test is running) otherwise the output signal could be a door lock or if the test equipment draws more than some specific number of Amperes. The Raspberry Pi is also connected to the local network where a MySQL database has been set up for this purpose. A simplified diagram can be seen in Figure 2. The operational status (for example "running" or "inactive") is read from the testing equipment by the Raspberry Pi and then reported to the database.

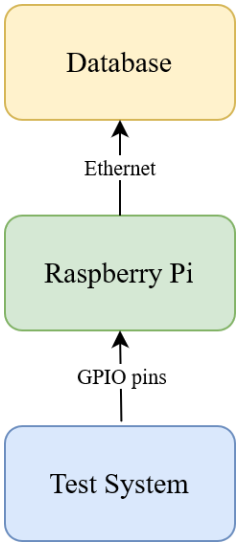

Figure 2: A diagram depicting the data monitoring system. The test systems' status is read by the Raspberry Pi via the GPIO pins and then communicated to the database over the local network via ethernet.

The status of the test equipment is read by the Raspberry Pi's GPIO pins connected to a D-sub connector where it reads the test equipment's status. The Raspberry Pi then calls a stored procedure in the MySQL database to report the name of the test equipment, time and new operational status to the database. The Raspberry Pi is running a Python script connected to the MySQL database containing logs of the operational status and the run time of each status. This way, each day, all the status logs from the previous day can be summed up to get the total time for each status during the previous day. A TV in the office is connected to a Windows PC running PHP: Hypertext preprocessor (PHP) scripts that collect the data from the MySQL database and then converts the data to charts using the Google Charts Library. The summed-up data is then shown on the TV-screen, mostly in the form of pie-charts as can be seen in Figure 3.

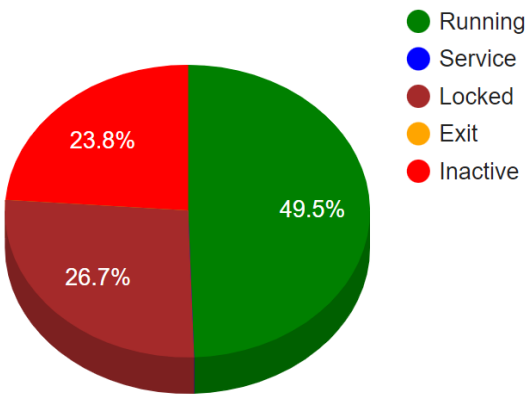

Figure 3: An example of how the compiled data is presented in the form of a pie-chart.

The current solution needs to be expanded to include other test equipment, more specifically temperature and climatic test systems made by Vötsch, a German company who develops and manufactures products regarding environmental simulation and heating technology. They have since rebranded to Vötschtechnik but will be referred to as Vötsch in this thesis. The addition of the temperature and climatic test systems to the monitoring system was planned from the very creation of the monitoring system but has not been implemented due to time constraints. Temperature and climatic test systems is a type of test equipment that is used to determine a product's tolerance to operating in very warm or cold environments as well as determining how a product degrades and loses performance after thermal cycling. The operating temperature varies based on the model of temperature and climatic test system, but most models are capable of -100 ^{O}C to +200 ^{O}C. The temperature and climatic test systems used by BorgWarner in Landskrona is made by Vötsch and there are many models, although most are the VT^{3} model [2]. All models have varying degrees of networking capabilities and various analog and digital outputs. The functionality of these outputs is to be examined to determine which one is the most reliable, versatile and easy to work with. The main usage of these test systems at the Landskrona site is testing of the actuator used in the coupling device seen in Figure 1. The actuator has multiple purposes, one being the coupling of the AWD system and another one being enabling the park lock feature which prevents unwanted movement as the vehicle has come to a stop.

When the usage-data is collected it can be used to make descriptive statements about the usage of each individual temperature and climatic test system as well as all of them as a

whole. The data will be used to aid the usage optimization of the temperature and climatic test systems.

1.2. Purpose

The purpose of this thesis is to develop an extension to the previous monitoring system in order to describe and analyze how BorgWarner use their temperature and climatic test systems. The extension is important as this equipment, beyond a high upfront cost, have a high maintenance cost in the form of upkeep and calibration as well as requiring lots of space. Because of these factors, it is of great importance to optimize the usage of the testing equipment, and in order to optimize the usage, the usage must first be recorded and described.

1.3. Goals

The goal is to collect data about the operational status of the temperature and climatic test systems and integrate the data collection into the already existing monitoring system. The data needs to be related to the operational status of the temperature and climatic test system. The data should also be formatted in such a way to easily be integrated in the already existing system. This would result in the daily and monthly status of the temperature and climatic test systems being displayed on a display in the office.

1.4. Questions

- What relevant signals do the temperature and climatic test systems output?
- How is the signal sampled?
- How is this information communicated to the database?
- How is this data presented?

1.5. Motivation of thesis

The thesis was chosen because of its high relevancy to Electrical engineering as the project involves working with both software and hardware as well as the integration of the two. BorgWarner provided this thesis because the temperature and climatic test systems can be a big bottleneck in the test and validation operation that can incur big costs. The data provided by this thesis is intended to partly solve this problem.

1.6. Delimitation / boundaries

The thesis should only include work regarding the temperature and climatic test systems and the hardware and software that is needed to create or collect relevant data and communicate with the database.

2. Technical Background

The new monitoring system, developed during this thesis, works by connecting a Raspberry Pi to one or multiple temperature and climatic test systems via an ethernet port and via another ethernet port it is also connected to the local network. The Raspberry Pi runs a Python script that contains a hard coded two-dimensional list (Python's version of an Array) of all connected test system's attributes, where the place in the list corresponds to each individual test system. The attributes contained in the list are IP-address, port number and alias (E.g. testSystem[1][2] would correspond to test system one's port number). The Raspberry Pi also contains a list of the previous status of each test system. The script cycles through the IP-addresses to send a predetermined message to the test systems. The test systems send a return message consisting of the test systems current operational status. If the status the Raspberry Pi receives is different from the previous status it then sends a MySQL query to the database to update with the new status and alias. This is repeated every 5 minutes for each test system. A diagram of the layout of the new monitoring system can be seen in Figure 4.

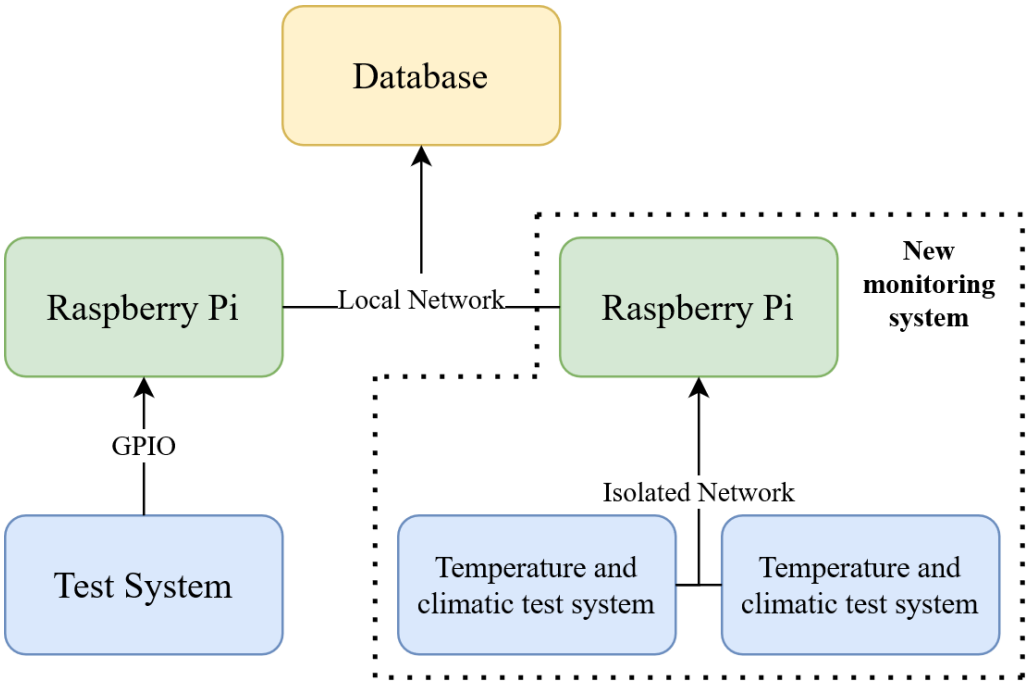

Figure 4: A diagram of how the layout of the new and old monitoring systems. The new one works by using a Raspberry Pi that is connected to both the database and the temperature and climatic test systems via ethernet. This creates an isolated network within the local network where the Raspberry Pi can communicate on both networks while keeping the test systems off the local network.

2.1. Raspberry Pi

The Raspberry Pi is a credit card sized single board computer developed and produced by The Raspberry Pi Foundation [3]. While the Raspberry Pi Foundation has developed and produced many products, the one used in this thesis is the main line of Raspberry Pi computers, specifically the Raspberry Pi Model 3B. It uses an ARM-processor and most often runs some variation of a Unix operating system [4]. In this project the operating system Raspberry Pi OS Lite is used, which is a command line-based operating system with a very limited graphic user interface (GUI) that still retains most functionality [5]. Compared to many other development boards the Raspberry Pi contains a microprocessor instead of a microcontroller. The Raspberry Pi is therefore capable of natively (with a compatible operating system) supporting more complex functions such as video, sound, USB and networking. The Raspberry Pi is often used in development due to this native support and also its the GPIO pins that can be controlled through software as both input and output. The ability to gain remote access via Secure Shell (SSH) also increases convenience when developing on the Raspberry Pi. A picture of a Raspberry Pi Model 3B can be seen in Figure 5.

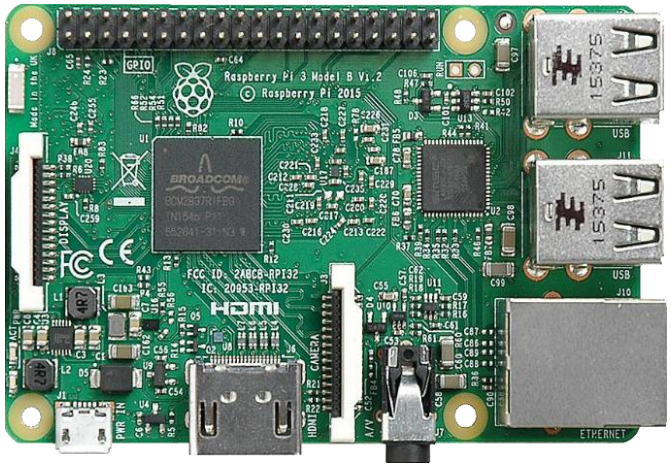

Figure 5: Raspberry Pi 3B. GPIO pins can be seen on the top side, USB and ethernet on the right side, HDMI, audio and power (via Micro USB) on the bottom side and a ribbon cable on the left. [6]

2.2. Secure Shell Protocol

SSH is a cryptographic network protocol that enables the safe communication between a server and a client over an insecure network and is by default implemented in Windows, Unix and MacOS [7]. SSH is used to gain remote access to a computer, often to the target computer's command line or desktop. This enables the user to read, write and execute files on the target computer that the user might not have physical access to. Even if the user has physical access to the hardware, it can be much more practical to access the hardware from a different computer to avoid having to set up a display and keyboard wherever the hardware happens to be located and instead being able to sit at one's desk, this is the case during this thesis.

2.3. Visual Studio Code

In the integrated development environment and code editor Visual Studio Code (VSCode) Microsoft has developed an extension that allows direct access to SSH within the editor. This allows the user to make changes to code and execute code on a different computer without accessing that computer. Apart from not having physical access to the computer this also makes writing code a lot easier when an IDE is not available in the operating system, such is the case with Raspberry Pi OS Lite. This extension of course shares many of the benefits as SSH but also decreases the amount of work having to be done, in this thesis' case, in the Linux terminal and instead being able to open, read and write to files in a convenient GUI.

2.4. Temperature and Climatic Test Systems I/O

The temperature and climatic test systems are used to test a product's tolerance and performance while thermal cycling. The operating temperature of the test systems can range from -100 ^{O}C to +200 ^{O}C and the test systems can make rapid changes to their operating temperature. The manufacturer of all the test systems that is used during this thesis are made by Vötsch. A picture of a Vötsch VT3 4034 can be seen in Figure 6.

Figure 6: Picture of a Vötsch VT3 4034 Temperature and climatic test system [8], the most commonly model used by BorgWarner in Landskrona.

The different models have different outputs depending on the model, but they all have an analog and digital D-sub connector, the digital one using the RS-232 protocol, a “dry” 4-pin contact, an Ethernet port and Wi-Fi [8]. A dry contact means that the state of the contact is not communicated through voltage or current just by the continuity between two pins, they are also known as relay outputs. Relay outputs are used to let a relay, which is controlled by the test system, connect power to the load. They can usually support higher currents than transistor outputs. The analogue D-sub connector, seen in Figure 7, outputs the temperature or humidity in the form of a voltage between 0V and 10V which corresponds to -100 ^{O}C to +200 ^{O}C and 0% to 100% humidity. This means that the analogue D-sub is very easy to read from using an analogue to digital converter in combination with the Raspberry Pi’s GPIO pins. This however does not directly give access to the status of the test system. A function would have to interpret the temperature and some other value, such as time, to infer the status of the test systems status.

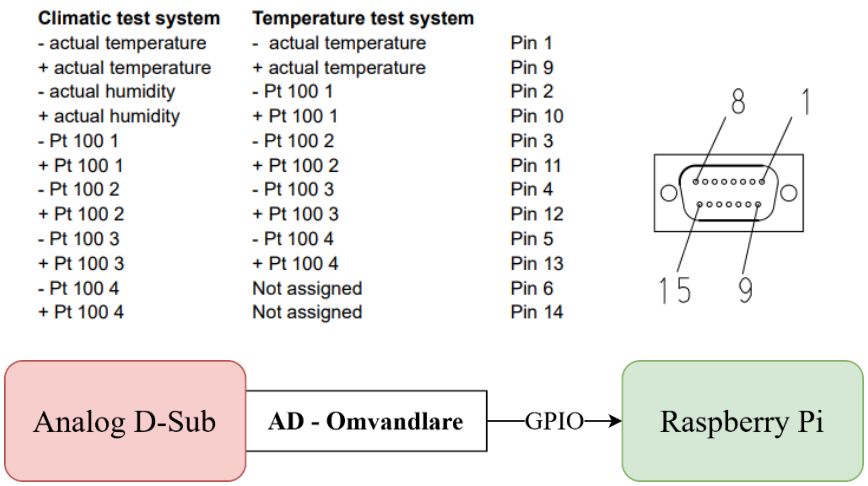

Figure 7: A diagram of the analog D-sub connector. [8]

The digital RS-232 D-sub, seen in Figure 8, is configurable from the inbuilt screen on the test systems and could output temperature and status. An implementation of the RS-232 communication protocol would have to be written to use this connector. This connector also uses relay outputs.

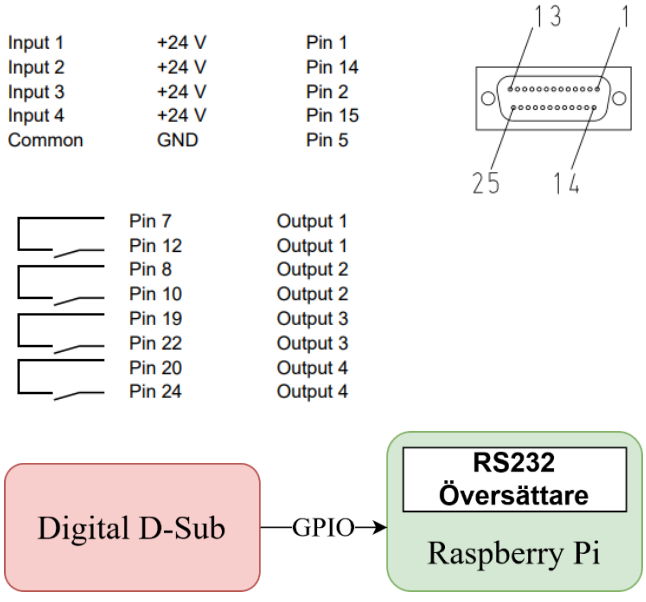

Figure 8: A diagram of the digital RS-232 D-sub contact. [8]

The 4-pin dry contact, seen in Figure 9, is the simplest of these three connectors. If the test system is running pin 2 and 3 will close and if there is a fault, test stop or pause, pin 1 and 3 will close. This could act as a switch connected to the GPIO pins and it can be easily read if the test system is running or not.

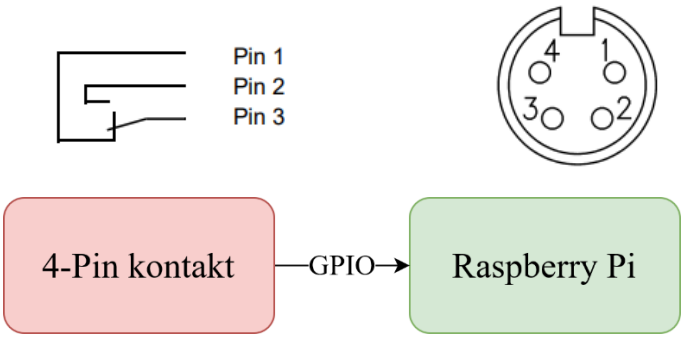

Figure 9: A diagram of the 4-pin dry contact. [8]

2.5. Python TLS/IP

To communicate with the temperature and climatic test systems over a network Transport Layer Security (TLS) and Internet Protocol (IP) is used. Together these two protocols enable cryptographically secure communication between a client and a host. Python has a built-in module called socket.py. socket.py provides a low-level networking interface that can be used to send TLS/IP packets on local networks as well as on the internet [9]. socket.py uses the Berkeley Socket (BSD) to allow two applications to communicate. BSD requires a server/client structure where the server binds both an IP-address and a port number and then listens for incoming connection requests. Once the connection request from the client has been received by the server it accepts and a TLS three-way handshake is initiated to secure the connection. Once the handshake is completed the client and server can start to send data back and forth. When the client is done it closes the socket and sends an *end* message to the server to notify that the client has closed its socket. The server then closes its own socket. A diagram of this process can be seen in Figure 10.

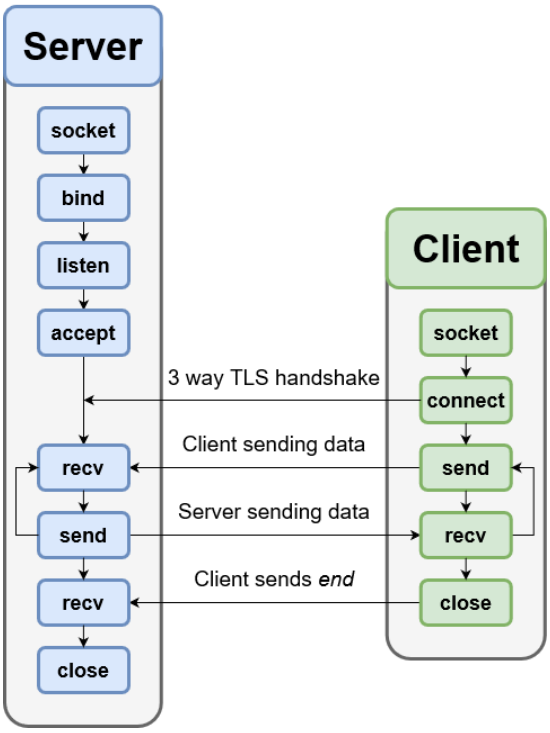

Figure 10: Diagram showing how TLS/IP communication is established, maintained and terminated in socket.py

2.6. Encoding

To send data over a network it must first be encoded. Encoding is the process of assigning numbers to represent characters. A simplified example could be A = 0, B = 1, C = 2 and D = 3. If a message of "ABCD" is to be sent this can be encoded to binary as "00 01 10 11". In this example there is only space for four distinct characters because we only used two bits. In modern encoders all characters, upper and lower case, special characters and in almost all languages are represented. Encoding can be done in many ways but the most common is Unicode Transformation Format 8-bit (UTF-8). About 98% of the internet uses UTF-8 [10]. UTF-8 can represent 1112064 different characters in the form of one to four bytes where the first seven bits are identical to the older seven-bit encoder American Standard Code for Information Interchange (ASCII) to ensure backwards compatibility [11]. Because of this, all ASCII text can be decoded with UTF-8, but all UTF-8 text can't be decoded with ASCII. UTF-8 is therefore said to be an extension of ASCII. Many more extensions of ASCII were developed to utilize the 8th bit. The most used single byte encoders today are ISO/IEC 8859-1 and Windows-1252. Both encoders are ASCII extensions that use the 8th bit for characters,

they are very similar and often get mixed up. Windows-1252 removed the control characters (128 to 159) from ISO/IEC 8859-1 but they are otherwise the same.

2.7. Temperature and climatic test system TLS/IP

To communicate with the temperature and climatic test system over Ethernet or Wi-Fi, Vötsch has implemented "function commands". Function commands are stored commands that can be accessed via TLS/IP and there are about 60 of them. The most relevant command for this thesis is "GET OPERATING STATUS", this command returns a hex value corresponding to the test system's current operating status, either Available, Run, Warning or Error. These commands have to be formatted according to Vötsch's S!MPATI, which is Vötsch's own proprietary service used to communicate with the temperature and climatic test systems using a PC program. Every function command consists of a command, separators (¶, ASCII 182), chamber index, between zero to four additional arguments and a carriage return (\r, ASCII 13). The make-up of the function command can be seen in Table 1.

Table 1: Showing the structure of the Vötsch function commands. The message is started with the command number, followed by a separator (¶, ASCII 182), followed by up to four arguments with a separator in between the arguments. The arguments are command dependent and can represent things like chamber index.

Make-up									
Command number	Separator	Chamber Index	Separator	Arg 1	Separator	Arg 2	Separator	Etc., up to 4 arg	Carriage return

In the form of a string the GET OPERATING STATUS might look like "10012¶1\r". Exactly which decoder the function command uses is not mentioned in the datasheet. It cannot be the original ASCII as the character "¶" is not included. The datasheet does label "¶" as ASCII 182 but ASCII only has 128 characters. This is used during the project to conclude that it uses Windows-1252 or ISO/IEC 8859-1. This could probably be attributed to the fact that at least the Vötsch VT^3 runs an integrated version of Windows XP.

3. Method

3.1. Execution

The project consists of four phases: information gathering, investigating the current monitoring system, investigating and testing the temperature and climatic test systems and implementing the new solution. How the information of each phase fed into the other phases can be seen in Figure 11.

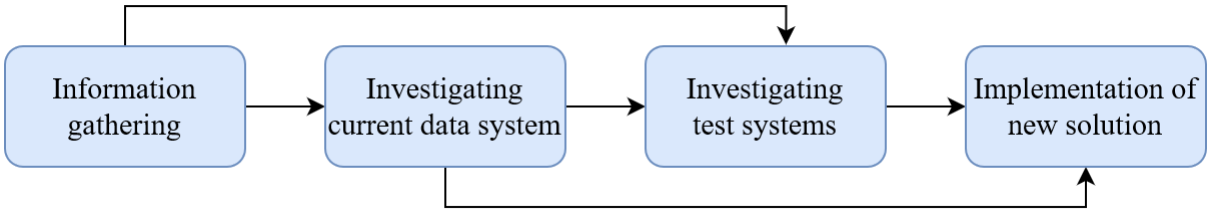

Figure 11: A diagram showing the phases and how the information from each phase fed into the other phases.

3.1.1. Information gathering

The project starts by compiling information regarding the temperature and climatic test systems as well as determining what signals are relevant for the thesis. This is done by reading the datasheets from the different models of temperature and climatic test systems, focusing on the output signals, and the difference in the output signals between the models. All the testing equipment have some degree of networking capability, though heavily dependent on the model. An important piece of information gathered in this phase is that some of the older models are running older operating systems, like Windows XP, which is no longer supported by Microsoft. This results in the operating system no longer receiving security updates making them vulnerable to cyberattacks.

3.1.2. Investigating the current monitoring system

BorgWarner provides all the resources needed to learn how the already existing monitoring system operates. The goal in this phase is to be able to read through and understand the code of the existing monitoring system as well as interface with the database using a Raspberry Pi

running a python script. This is accomplished by setting up a MySQL database with the same insertion table and stored procedure as the one in the already existing monitoring system and then trying to insert data into the table via Python's MySQL.Connector library. The new MySQL database is created to avoid the risk of making unintended changes to the BorgWarner database. To interface with the database code is written and then tested and validated on a PC. When the code can read and write to the database, the Raspberry Pi is loaded with Raspberry Pi OS Lite, SSH enabled, and the code transferred. To transfer the code to the Raspberry Pi the SSH extension is enabled in VSCode and the code is copied over. VSCode is then used to edit code directly on the Raspberry Pi. All work done regarding the Raspberry Pi is done via SSH and the code done in VSCode.

3.1.1. Investigating the test system

The next phase is to investigate the temperature and climatic test systems to establish which solution is the most practical. This is done, in part, through a meeting with BorgWarner employees who give feedback and insight from their knowledge and expertise and taking their insight into account with the information gained from phase one. One of the big decisions made during this thesis is how to gather data from the temperature and climatic test systems. At the meeting it is decided to investigate having the old test systems data collected by the 4-pin dry contact or one of the D-sub connectors instead of the Ethernet port, while the newer systems could connect directly to the local network. Although using the networking capabilities on all test systems would provide an elegant and versatile solution it is decided not to use the networking capabilities on the old test systems as they are unknown (According to the datasheet maybe only usable via a Vötsch proprietary service, S!MPATI). The older test systems do not run modern software and are therefore not protected against modern network-based attacks and thus it could be a safety hazard to have them use the company's local network.

In the end the ethernet port using TLS/IP is chosen because one of the employees has recently used and documented network-based communication with the old temperature and climatic test systems and reported it working well. A new decision is made regarding the old models, to use the Raspberry Pi as a gateway. This means that the Raspberry Pi communicates with the test system using an Ethernet port to get the status of the test system and then report this via a different ethernet port to the database over the local network. This way the old test systems

would be on a separate network from the rest of BorgWarner's enterprise, reducing the risk of malicious actors gaining access through the outdated software, and there is no proprietary solution needed for each model, making it easier to modify or add functionality in the future. A diagram showing how the communication between the test systems, Raspberry Pis and database as well as the network structure can be seen in Figure 12.

The Raspberry Pi needs two Ethernet ports, one for the temperature and climatic test system and one for the company's internal network. The Raspberry Pi Model 3B used has one integrated ethernet port so an USB-A to ethernet adapter is used as the second port. As there is currently a shortage of Raspberry Pis a network switch will be placed in each room so that one Raspberry Pi can access all the old test systems in that room. The newer models are not limited to this issue as they only require access to the local network and can therefore be located anywhere on the premise.

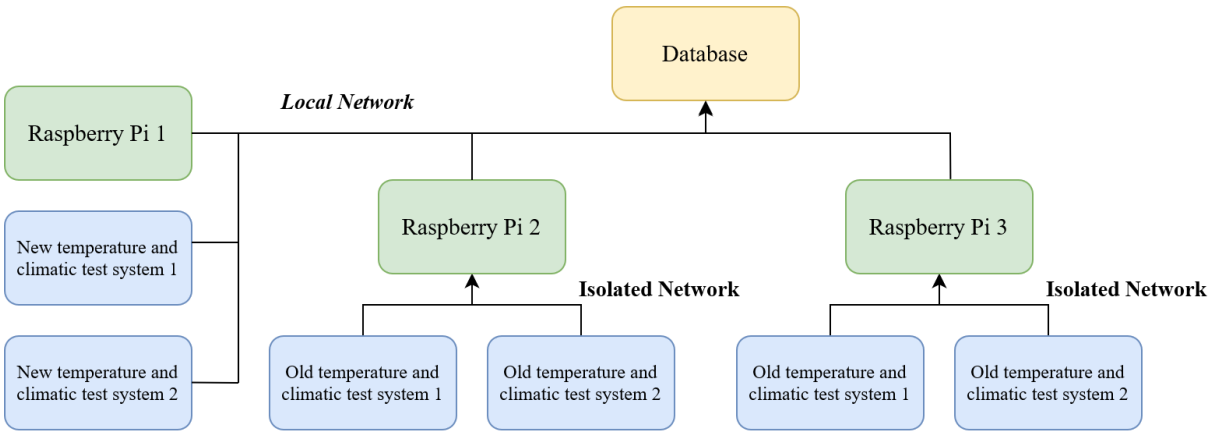

Figure 12: Showing the layout of the Raspberry Pis in relation to the database and Temperature and Climatic Test Systems. The new Test Systems communicate to Raspberry Pi 1 using the local network whilst the old test systems communicate over an isolated network to their respective Raspberry Pi that have access to both networks. All Raspberry Pis then communicates to the database over the local network. All communication over both networks is done using TLS/IP.

3.1.2. Implementation of the new solution

Implementing the new monitoring system developed through this project starts with establishing TCP/IP communication with the temperature and climatic test systems. This is done by reading datasheets regarding Vötsch's S!MPATI communication and function

commands and asking employees who have experience working with the temperature and climatic test systems.

When implementing the new monitoring system using a Raspberry Pi running Python is done because the previous monitoring system is built using these components. Using the same components ensures maximum compatibility between the old and new system and because the experience from the employee who had developed the previous monitoring system could be utilized. A big consideration during this whole project is that the temperature and climatic test systems are running constantly, allowing little time for safe experimentation. Almost all tests are conducted while the temperature and climatic test system is running, leading to a great degree of caution being needed not to invalidate the test currently running. The database is also being used and a certain degree of caution is needed to not delete years of datalogging by mistake.

Next is the implementation of TLS/IP and determining how the syntax and formatting of the function commands works. What is not known or written in the datasheet was what encoder the temperature and climatic test systems uses. Python uses UTF-8 as its default encoder over TLS/IP as it is by far the most common encoder. When sending function commands to the temperature and climatic test systems a plain text string is returned, a sign that things are working. The response is “READ failed: tag” followed by the message that was initially sent. Every combination of formatting the function command is tried but they all yield the same response. When encoding using ASCII Python reported a syntax error since “¶” (character 182) is outside of ASCII’s 128 characters. All that is known is that UTF-8 is an extension of ASCII so all ASCII characters in UTF-8 should be encoded the same as ASCII. Coupled with this the response is in plaintext so all signs pointed towards the encoding not being the problem. A different program, LabVIEW, is used to successfully send the function commands. Finally, when trying to find a solution online, it was suggested on a similar problem, to use the encoder CP1252 (Windows-1252) is suggested. This works instantly.

A python script is written that connects to the database and then updates the previous status of each test system that the Raspberry Pi has been designated. This is done by sending queries asking the database for the last logged status for each test system and then saving each status

in the List prev_status[]. The script then uses the IP-address and port number to send the *GET OPERATIONAL STATUS* function command to each test system. The Python script isolates the integer that represents the status of the test system and then compares that to the previous status. If the status has changed it gets reported to the MySQL database via the stored procedure. It then waits a predetermined amount of time before calling the next test system. The time waited is calculated by inputting how often the user wants to call each test system, and then divides that time by the amount of test systems designated to the Raspberry Pi. This way the function commands are distributed evenly in time.

3.2. Full implementation

The implementation consists of both the hardware and the software. The hardware is the simpler of the two, it consists of a Raspberry Pi model 3B with an added ethernet port. This is done by adding an USB-A to ethernet adapter to one of the USB-A ports. One of the ethernet ports is connected to the local network (where the database and SSH client is located) and the other is connected to one or more temperature and climatic test systems. This ensures that the test system is not directly connected to the local network and instead the Raspberry Pi is acting as a gateway between the two networks. To connect to multiple test systems a network switch must be used.

The software is a Python script. The script starts with importing *sleep* from the library time, the socket library, the mysql.connector library and *datetime* from the datetime library. Then the function, log_info, is defined. The functions' purpose is to enable pre-formatted logging to a .txt file for troubleshooting. The function will output the current year, month, date, hour, minute, second, a colon and a space and the input text in a file named log.txt. The list of the test systems attributes, (IP-address, port number and alias) a default value of previous status and the time between updates are defined. This code can be seen in Figure 13.

```
def log_info(text):
    log = open("log.txt", "a")
    log.write(str(datetime.today().strftime("%Y-%m-%d %H:%M:%S")) + ": " + text)
    log.write("\n")
    log.close()

# Settings for each tempChamber connected. [index][attribute]
tempChamber_address =   [["xxx.xxx.xxx.xxx",2049,"MarcusTest"],["xxx.xxx.xxx.xxx",2049,"MarcusTest2"]]
prev_status =           ["-1"                                 ,"-1"]

time_between_update = 5*60
```

Figure 13: Showing the log_info function, the test system attribute list (tempChamber_address), the list of previous statuses (prev_status) and the time between update variable.

The script then tries to connect to a specified database and logs the connection status in log.txt. The previous status list is then updated in a *for* loop. For each element in the list a SQL query is sent to get the last logged status of the test system with a specified alias. If no logged status is found it is set to “-1”.

Then the main *while* loop is started with an incrementing integer *i* representing a test system in the list. The loop starts by controlling if the connection to the database is still active and if not, tries to reconnect. A socket is created and tries to connect as a client to the test system with index *i* with its IP-address and port number. The message “10012\xb61\r” (the GET OPERATIONAL STATUS command) is encoded using Windows-1252 and sends the message to the test system. A response is received, decoded with Windows-1252 and the integer representing the status is isolated. This code can be seen in Figure 14.

```
try:
    client.connect((tempChamber_address[i][0], tempChamber_address[i][1]))

    fct_no = "10012"                                        # GET OPEARTION STATUS code
    message = fct_no.encode("cp1252") + b"\xb61\r"          # 10012¶1¶1\r b"\xb61\xb61\r"
    client.send(message)                                    # Get operating satus

    response = client.recv(4096).decode("cp1252")
    status = response.partition("¶")[2]                     # Isolate the status string
    status = ''.join(c for c in status if c.isprintable())  # Removes hidden new line
```

Figure 14: Showing the encoding and sending of the TLS/IP message as well as the decoding of the response and isolating the status integer.

The new status integer is compared to the previous status integer saved in the list and if they are different a MySQL query is sent that logs the test chambers alias and new status. The list is then updated with the new status. The socket is then closed and the sleep function is called. The delay of sleep is determined by the *time between updates* variable divided by the length of the test system attribute list. The index *i* is incremented and if *i* is larger than the length of the test system attribute list *i* is set to zero to reset the loop. If there are any errors when logging the status, if the connection to the database is terminated or if the program exits unexpectedly this is logged in the .txt file to make troubleshooting easier.

This is only an overview and the code in its entirety can be found in the appendix.

3.3. Evaluation of sources

[1], [2], [3], [4], [5], [6], [7], [8], [9] and [11] are all official sources from the creators or publishers of each of the products or services and are therefore considered trustworthy.

[10] W3Techs runs surveys to collect data about the usage of certain technologies used when building websites. They report that they have no affiliations with any other technology provider and was therefore considered trustworthy.

Another source of information that needs consideration is the code found online. When using code found online great care should be taken as to not add unknown functions. As mentioned in the method chapter the solution to the encoding was found online. The solution was found on https://stackoverflow.com/. When code was found online care was taken to only use small amounts and to experiment with the code in a safe environment to be able to understand what it's actually doing before using it in a larger more vulnerable environment.

4. Results and discussion

The result of this thesis is a method and program that can log the status of all Vötsch's IP/TLS-compatible temperature and climatic test systems and log that status in a database. In the event of a crash or an unexpected error the program also logs its own status in a local .txt file. To collect and insert the status-data in the database to then be able to describe BorgWarner's usage of their temperature and climatic test systems is the primary purpose of the thesis. In this respect the thesis was a success. It is however, too soon to draw any conclusions from the data collected. Months, if not years, of data is needed to be able to make prescriptive claims regarding BorgWarner's future actions regarding their temperature and climatic test systems. A sample of the data collected by the new monitoring system can be seen in Figure 15.

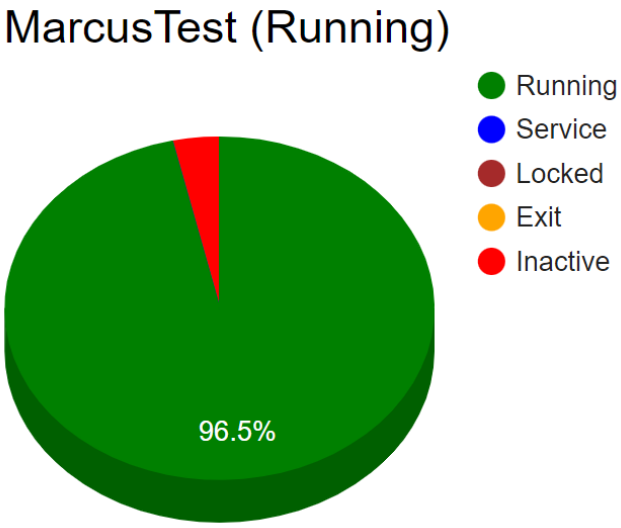

Figure 15: The pie-chart generated from the data collected using the script developed in this thesis.

There are however some predictions that can be of what the data will look like. If all test systems have a high percentage of usage (high percentage of the *running* status) that would indicate that there is a high amount of utilization of each test system and if more capacity is needed, then more test systems would have to be purchased. If all the test systems have a low percentage of usage (high percentage of the *inactive* status) that would indicate poor overall

usage. This might suggest that most employees try to run test the same day and once the test is complete another test is not started immediately, an explanation could be that employees tend to favor to start tests last thing before the weekend. The last alternative would be if some test systems have a high amount of usage whilst others have a low amount of usage. This might indicate that the location of the test systems might hinder or prohibit the usage or that employees favor newer test systems instead of old ones. This alternative does however require more analysis to figure out how the low usage test systems vary from the high usage ones.

There are things left that could be improved or changed. Automatic deployment is to be implemented to make it easier to add additional Raspberry Pis if needed. The automatic deployment would make it so that the same disk-image could be etched on each new micro-SD card. Some individual modifications would be needed such as inputting the IP-addresses of the test systems that the new Raspberry Pi is designated.

How to present the data is not yet decided. At the completion of the thesis the graph presentation system in the office only has six spaces for graphs and five of these spaces are already occupied by the test systems from the previous monitoring system. The presentation system can be seen in figure 16. This leaves only one chart for all the temperature and climatic test systems, compared to the 10+ spaces needed to present all the data collected. Some suggestions to solve this is either to have the presentation system periodically scroll through sets of test systems or to rework the presentation system to be able to display more test systems simultaneously. Another change that could be made is a composition of all temperature and climatic test systems usage into one graph. This would perhaps not be as interesting for employees at the test and validation department but could give executives an overview of the total usage.

Figure 16: How the presentation system is looks currently. Only six places are available to display the generated pie-charts.

Another question is whether separate (one per room) local isolated networks or BorgWarner's dedicated quarantined network is to be used. Using the quarantine network would only require one Raspberry Pi for all test systems as they would all be on the same network. The downside of this approach is that all the ethernet ports used would have to be configured to the quarantined network by the IT-department. This would make it hard to move the test systems and increase the risk of an unknowing employee connecting the test system to a non-quarantined port.

It was determined that these changes and improvements fell outside the scope of this thesis.

5. Conclusion

The goal of this thesis work has been achieved. The main goals of collecting status data, formatting the data and then integrating the data in the already existing system is achieved. Three out of the four questions are also answered.

- What relevant signals do the temperature and climatic test systems output?
    - Multiple relevant signals were found. The ones investigated where the analog D-sub connector, the digital D-sub connector, the 4-pin dry contact and the ethernet port and all of them could provide status data in some form. The ethernet port was chosen due to the simplicity and expandability of this approach. Collecting the status data from any other connector one Raspberry Pi would have to be permanently attached to each test system, while the ethernet enables usage of established networking protocols.
- How is the signal sampled?
    - This question was posed when the non-ethernet ports were discussed and lost its relevance when the ethernet port was chosen. An analog to digital converter or the ability to read the RS-232 protocol would be needed for the non-ethernet ports.
- How is this information communicated to the database?
    - The information is communicated to the database pre-formatted over the local network. This is on the “safe” side of the Raspberry Pi gateway so it’s safe to send over the local network.
- How is this data presented?
    - This is still to be determined. Discussions are still ongoing regarding who wants what data and what modifications would have to be made to accommodate these wants.

5.1. Future work

Some future work is still to be done. An auto deployment feature is to be implemented; this would make it easier to add more Raspberry Pis if another room needed to be covered. This

feature would enable the same disk image to be etched onto new micro SD-Cards. The presentation system is to be changed as it was not designed to display the amount of test systems now logging in the database. Exactly what this would look like is not yet decided but a couple of suggestions are to change the size of every test system to be able to fit more test systems on the display at once or to periodically cycle through test systems.

5.2. Reflection of ethical aspects

A somewhat unintended side effect of this thesis is the potential reduction of industrial waste and avoiding greenhouse gas emissions. If after analyzing the data collected by the new monitoring system a change in procedure is enough to alleviate any bottleneck, then that would avoid the unnecessary production as well as the transportation of a large and heavy test system which would presumably have a high carbon footprint.

6. Terminology

- JLR – Jaguar Land Rover, a British automotive manufacturer
- FWD – Front-wheel drive
- RWD – Rear-wheel drive
- AWD – All-wheel drive
- GPIO – General Purpose In Out
- MySQL – A relational database management system
- GUI – Graphical User Interface
- SSH – Secure shell
- IDE – Internal Development Environment
- TLS/IP – Two separate protocols, Transport Layer Security and Internet Protocol. Together they enable cryptographically secure communication between a client and a host.
- BSD – Berkeley Socket
- UTF-8 – Unicode Transformation Format 8-bit
- ASCII - American Standard Code for Information Interchange
- Gateway – A type of networking hardware that enables controlled communication between two networks

7. References

[1] BorgWarner, "The sixth generation of integrated AWD Actuators," Mattias Magnusson, 10 May 2019. [Online]. Available: https://eawdcongress.com/download/mattias_magnusson.pdf. [Accessed 12 05 2022].

[2] "Temperature Test Chambers VT3 VTS3 Product sheet," Weiss Technik, April 2016. [Online]. Available: https://www.weiss-technik.com/fileadmin/Redakteur/Mediathek/Broschueren/WeissTechnik/Umweltsimulation/Voetsch/Voetsch-Technik-VT3-VTS3-EN.pdf. [Accessed 12 May 2022].

[3] "Raspberry Pi Homepage," The Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.com/. [Accessed 25 February 2022].

[4] "Raspberry Pi 3 Model B+ Product Brief," The Raspberry Pi Foundation, [Online]. Available: https://static.raspberrypi.org/files/product-briefs/Raspberry-Pi-Model-Bplus-Product-Brief.pdf. [Accessed 25 February 2022].

[5] "Raspberry Pi OS Documentation," The Raspberry Pi Foundation, [Online]. Available: https://www.raspberrypi.com/documentation/computers/os.html#introduction. [Accessed 25 February 2022].

[6] Herbfargus, "commons.wikimedia.org," 13 March 2016. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=47497384. [Accessed 15 March 2022].

[7] "SSH (Secure Shell) Home Page," SSH Communications Security Corp, [Online]. Available: https://www.ssh.com/academy/ssh. [Accessed 12 Mars 2022].

[8] "INSTALLATION AND OPERATION MANUAL Temperature and Climatic Test Systems with SIMPAC controller," Vötsch, 11 2010. [Online]. Available: https://assets.fishersci.com/TFS-Assets/CCG/EU/Voetsch-

Industrietechnik/manuals/VOE042_EN%20CLIMATE%20IN%20PERFECTION%20VT3%20VC3.pdf. [Accessed 13 January 2022].

[9] "socket — Low-level networking interface," Python Software Foundation, 26 April 2022. [Online]. Available: https://docs.python.org/3/library/socket.html. [Accessed 10 Mars 2022].

[10] "Usage of character encodings broken down by ranking," [Online]. Available: https://w3techs.com/technologies/cross/character_encoding/ranking. [Accessed 05 April 2022].

[11] "The Unicode Standard Version 6.0 - Core Specification," Unicode Consortium, February 2011. [Online]. Available: https://www.unicode.org/versions/Unicode6.0.0/ch02.pdf. [Accessed 10 05 2022].

8. Appendix

```python
from time import sleep
import socket
import mysql.connector
from datetime import datetime

def log_info(text):
    log = open("log.txt", "a")
    log.write(str(datetime.today().strftime("%Y-%m-%d %H:%M:%S")) + ": " + text)
    log.write("\n")
    log.close()

# Settings for each tempChamber connected. [index][attribute]
tempChamber_address =   [["xxx.xxx.xxx.xxx",2049,"MarcusTest"],["xxx.xxx.xxx.xxx",2049,"MarcusTest2"]]
prev_status =           ["-1"                                 ,"-1"]

time_between_update = 5*60

dbconn = mysql.connector.MySQLConnection()
# Connect to DB
try:
    dbconn.connect(

        host="",
        user="",
        password="",
        database=""

    )
except mysql.connector.Error as e:
    log_info("ERROR connecting to db: {e}")
else:
    log_info(f"Connected to server successfully")
    dbcur = dbconn.cursor()

```

```python
# Updates the last logged status of all chambers
for i in range(len(prev_status)):
    dbcur.execute(f"SELECT status_id FROM status_log INNER JOIN rig_names ON
status_log.rig_id = rig_names.rig_id WHERE rig_name = '{tempChamber_address[i][2]}' ORDER BY timestamp DESC LIMIT 1")
    prev_status[i] = ''.join(x for x in str(dbcur.fetchall()) if x.isdigit())
    # If there is no logged status set prev status to -1
    if prev_status[i] == None:
        prev_status[i] = "-1"

```

```python
#------Main loop--------
# Cycle the IP addresses in the tempChamber_adress list
i = 0

while(1):

    client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    if(i not in range(0, len(tempChamber_address))):
        i = 0

    # Check connection and try to reconnect
    #--------------------------------------
    if (not dbconn.is_connected()):
        log_info("Lost connection to mysql server")
        reconnect_count = 0
        while not dbconn.is_connected():

            # wait before reconnecting
            if reconnect_count > 5:
                log_info(f"Reconnect count: {reconnect_count}, Exiting ")
                exit(1)
            elif reconnect_count > 3:
                log_info(f"Reconnect count: {reconnect_count}, waiting 60s.. ")
                sleep(60)
            elif reconnect_count > 0:
                log_info(f"Reconnect count: {reconnect_count}, waiting 10s..")
                sleep(10)


            # reconnect
            try:
                dbconn.reconnect()
            except mysql.connector.Error as e:
                log_info(f"ERROR reconnecting to mysql server: {e}")
            else:
                dbcur = dbconn.cursor()
                log_info(f"Reconnected to mysql server successfully")
            reconnect_count += 1

    try:
        client.connect((tempChamber_address[i][0], tempChamber_address[i][1]))

        fct_no = "10012"                                    # GET OPEARTION STATUS code
        message = fct_no.encode("cp1252") + b"\xb61\r"      # 10012¶1¶1\r b"\xb61\xb61\r"
        client.send(message)                                # Get operating satus

        response = client.recv(4096).decode("cp1252")
        status = response.partition("¶")[2]                 # Isolate the status string
        status = ''.join(c for c in status if c.isprintable())    # Removes hidden new line

        if(status != prev_status[i]): # 1 = inactive, 3 = running

            # Update database
            try:
                dbcur.callproc('update_rig_status', (tempChamber_address[i][2],status))
                dbconn.commit()
            except mysql.connector.Error as e:
                log_info(f"ERROR Updating status-db: {e}")
            else:
                log_info(f"Logged '{tempChamber_address[i][2]}' status={status} successfully")
                prev_status[i] = status

            client.close()
    except socket.error:
        log_info("Fail to connect")

    sleep(time_between_update/len(tempChamber_address))
    i += 1

log_info("Unexpected error, exiting...")
quit()
```