Johanna Nygård

Upgrading the Microcontroller and Automation Test Suite for a Miniature Ventilation System

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

27 September 2023

Abstract

Author:	Johanna Nygård
Title:	Upgrading the Microcontroller and Automation Test Suite for a Miniature Ventilation System
Number of Pages:	34
Date:	27 September 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Smart Systems
Supervisors:	Keijo Länsikunnas, Principal Lecturer

This project aimed to update a miniature test system. The update will allow the system to work similarly to an automated ventilation system and employ its test automation suite that runs on a Robot Framework. Both ventilation systems are used as learning materials by students in the Embedded Systems Programming course at the Metropolia University of Applied Sciences.

The project involved developing software for the Raspberry Pi Pico microcontroller, the main objective of which is to collect and manage data from the miniature ventilation system. The data collection primarily relied on the Modbus, MQTT, and I2C communication protocols. The Robot Framework test suite, originally designed for automated ventilation systems, was also enhanced to function with the miniature ventilation system.

By incorporating MQTT, a wireless communication protocol, and a Robot Framework test suite for the miniature testing system, a system with more efficient communication between the different components is achieved. These updates ultimately enable early-stage software testing for students to improve the learning experience.

Keywords: Ventilation System, Modbus, MQTT, I2C, Robot Framework

Tiivistelmä

Tekijä:	Johanna Nygård
Otsikko:	Mikrokontrollerin ja testiautomaation päivittäminen ilmastointijärjestelmän pienoismalliin
Sivumäärä:	34
Aika:	27.10.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	tieto- ja vietintäteknologia
Ammatillinen pääaine:	Älykkäät Järjestelmät
Ohjaajat:	Keijo Länsikunnas, Lehtori

Tämä opinnäytetyö pyrkii päivittämään pienen testijärjestelmän. Päivityksen avulla pieni ilmastointijärjestelmä voi toimia samankaltaisesti kuin automatisoitu ilmastointijärjestelmä ja hyödyntää sen testiautomaatiosarjaa, joka toimii Robot Frameworkin avulla. Ilmastointijärjestelmää käytetään Metropolia Ammattikorkeakoulun sulautettujen järjestelmien ohjelmointi -kurssin opiskelumateriaalina.

Projekti keskittyy Raspberry Pi Pico -mikro-ohjaimen ohjelmiston kehittämiseen, jonka päätavoitteena on kerätä ja hallita tietoja pienoismallin ilmastointijärjestelmästä. Tietojen keruu perustuu pääasiassa Modbus-, MQTT- ja I2C-viestintäprotokolleihin. Lisäksi Robot Frameworkin testisarjaa on päivitettävä niin että, se pysty toimimaan myös pienellä ilmastointijärjestelmässä.

Päivitetty järjestelmä mahdollistaa tehokkaamman viestinnän eri komponenttien välillä. Nämä päivitykset mahdollistavat opiskelijoille ohjelmistojen varhaisen testauksen, mikä parantaa oppimiskokemusta.

Avainsanat: Ilmastointijärjestelmä, Modbus, MQTT, I2C, Robot Framework

Contents

List of Abbreviations

12C:	Inter-integrated Circuit
CRC:	Cyclic Redundancy Check
FIFO:	First in First Out
FTP:	File transfer protocol
HTML:	HyperText Markup Language
JSON:	JavaScript Object Notation
M2M:	Machine to Machine
PWM:	Pulse Width Modulation
RTU:	Remote Terminal Unit
SMTP:	Simple Mail Transfer Protocol
UART:	Universal Asynchronous Receiver-Transmitter
UAS:	University of Applied Sciences
USART:	Universal Synchronous and Asynchronous Receiver-Transmitter
USB:	Universal Serial Bus
WPA3:	Wi-Fi Protected Access 3
XML:	Extensible Markup Language

1 Introduction

Embedded system programming is constantly evolving, as seen in a project undertaken by students at the Metropolia University of Applied Sciences (UAS). The project is part of an embedded systems programming course, where students work in groups to develop a ventilation controller system for an automated ventilation system using an LPC1549 microcontroller. Over time, this project has evolved from students working on a miniature ventilation system to using a more realistic version of an automated ventilation system, providing valuable hands-on experience for students on the embedded systems programming course.

Currently, only one automated ventilation system is available, so the miniature ventilation system is still being utilized during the course. The code developed by the students for the miniature ventilation system should also be compatible with the automated ventilation system. Even though the ventilation systems are similar in design, there are still differences between the systems. Currently, only the more realistic ventilation systems support automated testing. The tests are conducted through a Robot Framework test suite which communicates with the ventilation system through the Message Queuing Telemetry Transport (MQTT) protocol, giving access to wireless data gathering.

Meanwhile, the microcontroller of the miniature ventilation system lacks a built-in WI-FI module and thus does not support wireless data gathering. The microcontroller instead gathers this data through a universal asynchronous receiver transmitter (UART), a wired protocol. Because of the differences, the miniature ventilation system is incompatible with the automated ventilation systems test suite and must still be manually checked by the lecturer.

This project aims to upgrade the miniature ventilation system to function similarly to the automated one. These upgrades will involve upgrading the miniature ventilation system and changing the debugging method to match the automated ventilation system. The Robot Framework test suite will also be upgraded to

support the debug information the miniature ventilation system provides. By doing these upgrades, the students can evaluate their code more thoroughly on the miniature ventilation system and expect most tests on the automated ventilation system to pass. This will also save time per student group on the automated ventilation system, preventing overcrowding.

This thesis consists of six sections. Following the introduction, Section 2 provides background information on the ventilation system and test automation. Section 3 is about project specifications and describes the current miniature ventilation system in more detail. Section 4 introduces the tools and materials used. Section 5 describes the results of the final year project. Lastly, section 6 summarizes this project and describes potential future developments for the ventilation systems.

2 Background

This chapter presents the automated ventilation system and the miniature system in detail. It also explains the background and usage of the Robot Framework, Modbus, and MQTT. Further information on a few segments is given in Chapter 4: Methods and Materials.

2.1 Ventilation System

The Embedded System programming course at Metropolia UAS offers a valuable learning experience focused on the programming aspect of a ventilation system. Through the course, students are tasked with developing a program capable of effectively controlling the system. The main objective of the student's program is to execute it on the LPC1549 microcontroller. The microcontroller serves as a central control unit responsible for controlling and managing the data flow between the external source and the relevant components of the system. On the course, the students get to experience how to work with embedded systems devices and work on various communication protocols, such as inter-integrated circuits (I2C), Modbus, UART, and MQTT.

2.1.1 Compositions of the Ventilation System

The ventilation systems consist of several main physical components, including the LPC1549 microcontroller, a WI-FI module, a fan, a vent, a pressure sensor, and a fan control module. As previously mentioned, the LPC1549 functions as the central control unit of the ventilation system, interpreting commands and facilitating the data flow between the external sources and system components, as figure 1 demonstrates.

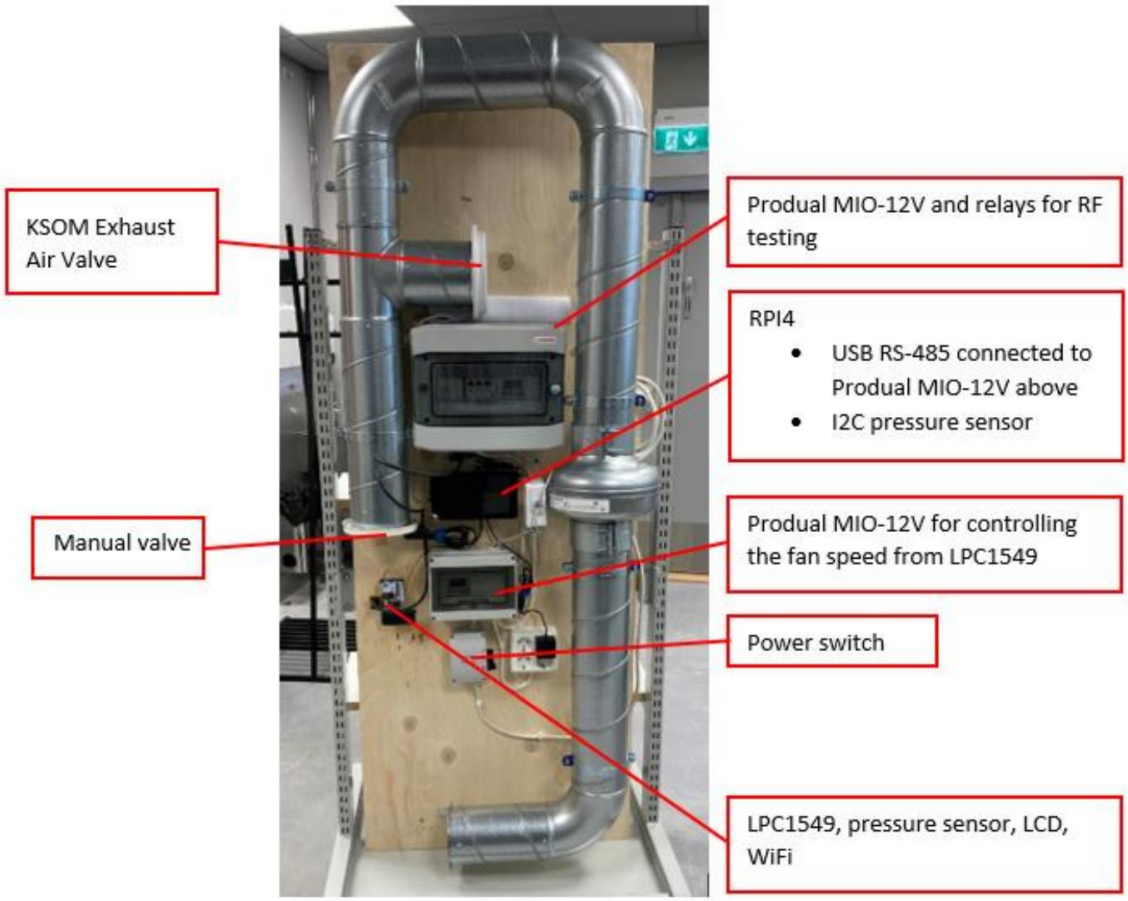

Figure 1. Physical components of the automated ventilation system [1]

It is important to note that the LPC1549 does not have built-in support for WI-FI connectivity. Therefore, an additional external Wi-Fi module is required to enable wireless connectivity and support the MQTT messaging protocol. This enables remote monitoring and control capabilities, which are also necessary for the current Robot Framework test.

Within the automated ventilation system, the MIO-12V module serves as the fan control module, which is responsible for receiving data from the LPC1549 microcontroller and performing crucial functions such as regulating the fan's speed and sending control data to the external source.

Simultaneously, the pressure sensor constantly measures the pressure inside the vent, which is influenced by the fan speed. With the data received from the pressure sensor, it is possible to adjust the fan speed to reach the desired pressure and monitor the ventilation system's airflow. One of the assigned tasks to students is to monitor the ventilation system and ensure that a specific pressure is reached and maintained. Each above component is essential to the ventilation system and contributes to its functions and performance.

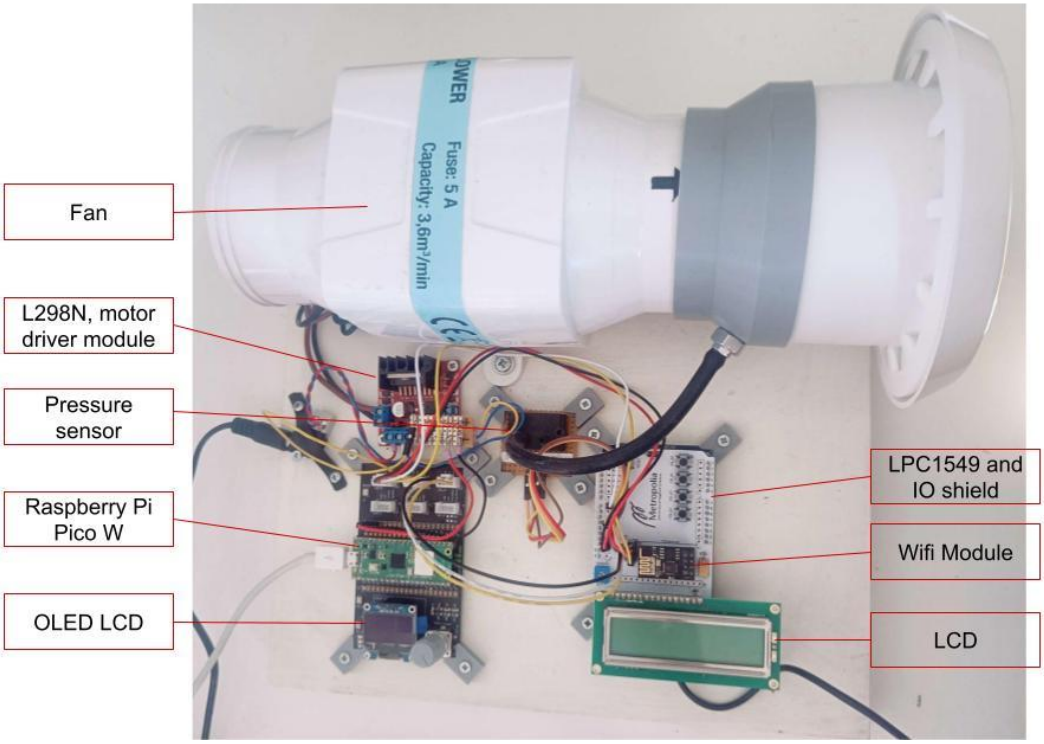

Figure 2. Physical components of the updated Miniature Ventilation System.

Moreover, the miniature testing system has an Arduino microcontroller that mimics the behavior of the MIO-12V module from the more realistic ventilation system, which is responsible for fan control based on Modbus messages received from the LPC1549 microcontroller. The miniature test system consists mainly of

similar parts and functions to the automated ventilation system, except that the Arduino microcontroller simulates sensor readings and that the miniature system cannot produce as high pressure as the actual system. Nonetheless, the software on the LPC1549, which is the embedded system programming course students' program, should function mostly identically on both systems.

2.1.2 Data Transmission

The data interpretation in the ventilation system utilizes different messaging protocols including I2C, Modbus, MQTT, and UART. The I2C (Inter-integrated Circuit) protocol is employed for communication between the pressure sensor and the ventilation system. In contrast, the Modbus protocol is utilized for communication between the LPC1549 microcontroller and the fan control module, enabling the fan`s speed regulation.

Moreover, both the MQTT and UART protocols are essential in data exchange and debugging. MQTT is a wireless communication protocol, which facilitates data exchange between the ventilation system and the external source. MQTT operates through an MQTT broker that can be configured to use IP addresses to communicate between different devices.

On the other hand, UART enables direct serial communication between the ventilation system and the external device. The utilization of each messaging protocol allows the microcontroller to support data sharing with the external source wirelessly or through wired connections. Figure 3 below illustrates the earlier-mentioned connections.

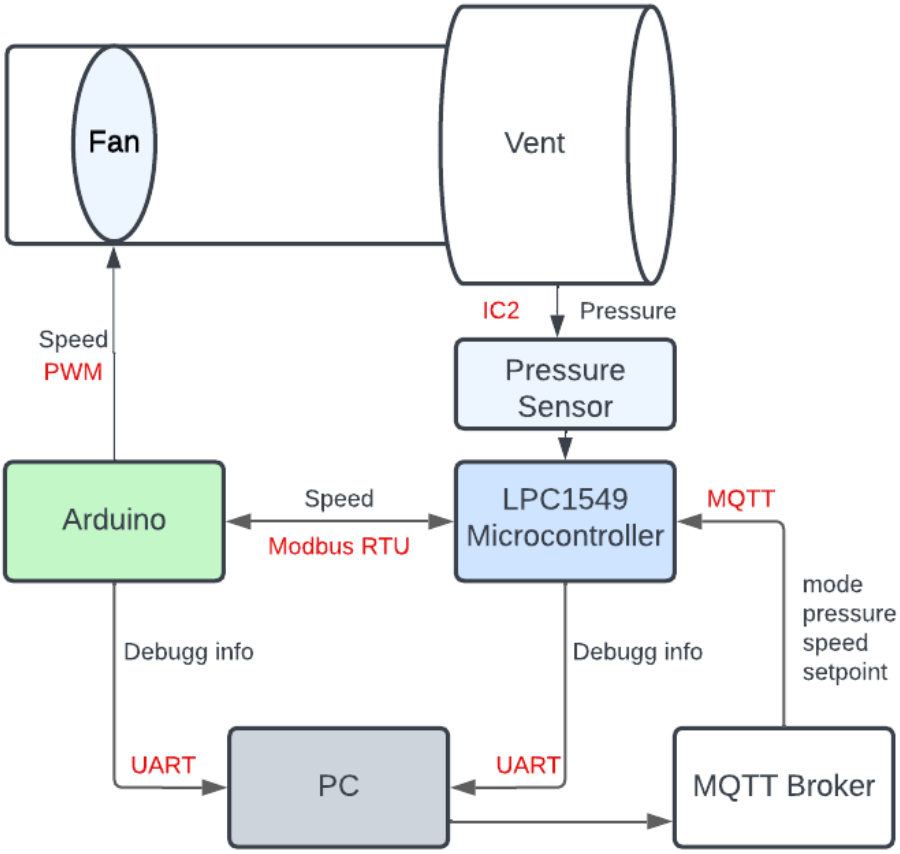

Figure 3. Overview of connection in the current miniature testing system

2.1.3 Message Handling

The ventilation systems are controlled through MQTT messages transmitted in JSON format. The messages consist of key-value pairs for mode and speed or pressure. Listing 1 illustrates received MQTT messages by the LPC1549 microcontroller, highlighting both manual mode (top) and automatic mode (bottom).

```
{"auto": false, "speed": 20}
{"auto": true, "pressure": 25}
```

Listing 1. MQTT message received by LPC1549: manual mode (top and automatic mode (bottom)

In response to received messages, the LPC1549 periodically transmits a status response comprising speed, pressure, setpoint, error, and mode. An illustrative

instance of an MQTT status message and its reply from the LPC1549 microcontroller is presented in Listing 2.

```
{
"speed": 100,
"setpoint": 100,
"pressure": 70,
"auto": false,
"error": false
}
```

Listing 2. MQTT status message example from LPC1549

The ventilation system obtains speed and pressure values using the Modbus and I2C communication protocols correspondingly. The fan`s speed is expressed as a percentage of its maximum revolution per minute (RPM) and can range from 0 to 100, while the pressure is measured in Pascal (Pa) with the maximum measurable value of 125 Pa.

The setpoint received by the microcontroller from MQTT refers to the target value for either speed or pressure. On the other hand, the error value serves as a control signal from the ventilation system. It indicates whether the fan is on, or the pressure has been successfully attained within the ventilation system. The error value functions as a Boolean parameter, usually set to False. When it is "true," it signifies that the setpoint has not been achieved, while "false" indicates the successful attainment of the desired pressure or speed.

Lastly, the auto mode is represented by a Boolean value of “true” for automatic mode or “false” for manual mode. In manual mode, a specific speed is set for the fan, and the ventilation system works to achieve that speed. In automatic mode, a desired pressure is set for the ventilation duct, and the ventilation system alters the fan speed to uphold the desired pressure.

The response received from the LPC1549 microcontroller may include additional key-value pairs not mentioned previously. However, the specified keywords are used for testing in the Robot Framework test suite. It is important to note that the order of the keys within the JSON object may vary, as the test suite does not rely on specific key-value pair order.

2.2 Test Automation

The Embedded System programming course has evolved, incorporating test automation on a more realistic ventilation system (cf. figure 1). The testing relies on a test automation tool, the Robot Framework test, which thoroughly evaluates the student's code. By including automated testing, instructors can significantly reduce their workload, sparing them from the laborious process of manual evaluation of the functionality of each student's code.

However, it is essential to note that the miniature testing system (cf. figure 2), although still utilized in the course, does not have its dedicated test suite. It is currently outdated compared to the automated ventilation system. This means the students must adjust their code to ensure compatibility with the more realistic system.

Test automation is a testing tool that can be used for various purposes, including comparing predicted with the actual outcome. Moreover, it significantly reduces manual labor by automating repetitive or complex tasks, increasing project efficiency and accuracy [2]. In this case, test automation reduces the manual workload for instructors and allows students to self-test their code with pre-existing software. The test process involves a test suite, a collection of test cases run upon a script that evaluates different scenarios on the ventilation system. This test suite automatically tests the ventilation system's most crucial functions, allowing students to test their scripts independently before submitting their code to the instructor, who also verifies the code through automated tests. This approach empowers students to identify and correct errors independently, enhancing their understanding and troubleshooting skills.

Furthermore, many test automation tools provide a valuable feedback function, enabling efficient identification of failures or errors. The Robot Framework used in the embedded system course automatically generates comprehensive reports that are easy to navigate and summarize. This feature benefits the instructor and students, providing insights into the code's performance and facilitating further improvements or optimizations. Additionally, using the Robot Framework test

tools provides a valuable troubleshooting experience for the students, empowering them to address and resolve issues effectively.

3 Project Specification

Both ventilation systems consist of similar components: a fan, a vent, a pressure sensor, an LPC1549 microcontroller, and a fan control module. Despite the similarity in the components, a few parts in the miniature system are smaller in size or simulated, causing differences in the performance or the usage of the system.

This chapter provides an overview of the present setup of the miniature ventilation system, compares the system with the automated system, and offers insights into the planned updates, the scope of the thesis, and the limitations of the project.

3.1 Scope

The thesis aims to enhance the miniature ventilation system to operate more similarly to the automated system. This will involve developing software for the Raspberry Pi Pico microcontroller, which replaces the previous Arduino microcontroller. The software aims to enable the Raspberry Pi Pico to receive speed and pressure data from the LPC1549 via the Modbus communication protocol. The microcontroller will then translate the received speed data into a PWM signal and transmit it to the fan. Additionally, the microcontroller will incorporate support for sending periodic debug messages through the MQTT protocol.

Furthermore, the project involved adapting the existing test automation framework from the automated ventilation system to be compatible with the miniature ventilation system. These enhancements bear the potential to amplify system functionality and enhance the testing process, providing students with more convenience in working with and adapting the miniature ventilation system code to the automated system.

3.2 Current Miniature Ventilation System

The miniature ventilation system is a portable, compact version of the automated system, making it possible for students to use it inside and outside the classroom. Despite its practicality, the miniature system exhibits several differences in functionalities compared to the automated system, such as using simulated sensor values instead of accurate values, differences in data transportation, and the need for test automation. These factors contribute to slight differences in functionality between miniature and automated systems. In this context, figure 3 represents a simplified version of the current miniature ventilation and its connections.

Regarding the present operation of the miniature ventilation system, Arduino involves simulating sensor values of SDP610 [3], a pressure sensor, of GMP252 [4], a carbon dioxide probe, and HMP65 [5], a humidity and temperature probe. Unlike the automated ventilation system, which provides accurate sensor readings, these simulated values do not directly affect the functionality of the ventilation system.

The miniature ventilation system offers two control modes: through a wired connection or wirelessly, utilizing universal asynchronous receiver-transmitter (UART) and MQTT protocols, respectively. It is essential to highlight that wireless communication is exclusive to the LPC1549 microcontroller due to the absence of a built-in Wi-Fi module and wireless support in the Arduino microcontroller. Making the debug information in ventilations systems fan control module accessible only through manual means, specifically via UART connection.

The arrangement described above presents a challenge for students, requiring them to connect to three UART ports simultaneously. This is required for powering the LPC1549, receiving its debug information, and accessing debug data from the ventilation system. In contrast, the automated ventilation system provides a more streamlined approach by supporting a wireless connection even in the fan control module. This eliminates the need for multiple physical

connections between the external source and the fan control module, allowing for a more convenient and efficient testing and debugging process.

Lastly, the miniature ventilation system is not supported by the test automation test suite as it is in the automated ventilation system. This is primarily due to the presence of different components, such as Raspberry Pi and MIO-12V, and their distinct methods of operation. Additionally, as mentioned earlier, the Arduino microcontroller in the current miniature system is not programmed to send debug information through MQTT messages, further contributing to the lack of compatibility with the automated tests. As a result, the existing automated test designed for the more realistic system cannot be directly applied to the miniature ventilation system.

3.3 Enhancement to the Miniature Ventilation System

This project aims to address and improve the differences in the miniature ventilation system. The aim is to align it more closely with the automated system. By doing so, the process of students implementing the code developed on the miniature system into the automated system will be easier, less time-consuming, and enhance the overall integration process. Additionally, implementing automated testing to the miniature system will enable possibilities to start testing at the initial stages of the student’s development process.

The enhancement to the miniature ventilation system aims to align its functionality with the automated system. This includes changing the microcontroller responsible for fan control to be more like the automated systems, creating a test automation framework compatible with the miniature system, and improving communication between the different components. The new microcontroller will also receive new features like real-time sensor values through the pressure sensor. The figure below represents a simplified version of the planned enhancement for the miniature ventilation system and its connections.

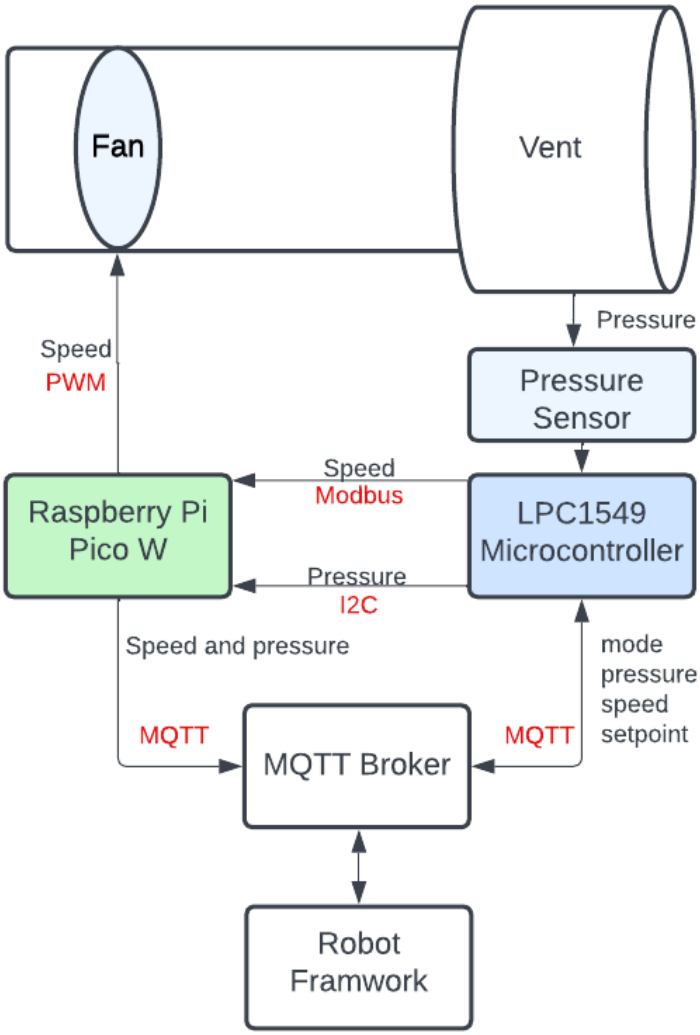

Figure 4. Overview of connection in the Miniature testing system

One key enhancement involves substituting the Arduino microcontroller with Raspberry Pi Pico W, a compact version of the Raspberry Pi employed in the automated system. The significant advantage is that both the Raspberry Pi and Pico versions support the Micropython coding language, providing opportunities for further updates and improvements on both ventilation systems.

Compared to Arduino, Raspberry Pi Pico has its own features and capabilities alongside a different coding language. Consequently, new software must be created to support the ventilation systems' functionalities, with the addition of some the upcoming updates. The upcoming updates will include removing

redundant sensor simulation, receiving actual pressure sensor values, and an upgrade that sends the debug information through the MQTT communication protocol. The debug messages will be gathered by an MQTT broker, from where the user can subscribe to content. This update allows the fan control module to have seamless debugging between the ventilation system and the external source, removing the need to connect each device separately to a UART cable.

Lastly, the Robot Framework test suite will be enhanced to accept and process data from the miniature ventilation system, ensuring that the automated testing will work similarly on both systems. This will allow students to assess their code in the initial stages of their project, enabling them to identify potential issues before moving on to the automated ventilation system or presenting it to their instructor. These updates enhance the miniature ventilation system to operate more similarly to the automated system.

3.4 Limitations

A limitation of this project is that the ventilation system must be connected to a broker to send debug data through MQTT. This makes it mandatory for the students to self-arrange a broker, especially if working outside the classroom, where a ready-made broker with a set IP address is unavailable. To connect to the broker, both the Raspberry Pi Pico and LPC1549 microcontroller need to be connected to the broker with its specific IP address. While it is possible for students to manually input the IP address into both microcontrollers, this process could prove inconvenient, particularly for the Raspberry Pi Pico microcontroller, as its operation falls outside the scope of the course curriculum.

One potential solution could involve the integration of the broker directly into the Raspberry Pi Pico, creating a dedicated IP address for the ventilation system, or developing a user interface for the students where they can add the MQTT and internet configurations. However, this falls outside of the scope of the thesis.
The test automation in the automated ventilation system may include tests for specific components and their functions that are not available in the miniature system. Consequently, it is possible that not all the tests conducted in the

automated system can be replicated in the miniature system, making the test automation for the miniature ventilation system less thorough.

4 Methods and Materials

This chapter covers the software, protocols, and hardware used in the project. The software and services section covers MQTT, Micropython, Modbus RTU, and the Robot Framework. The hardware section comprises the Raspberry Pi Pico W and the LPC1549 microcontroller.

4.1 Software and Protocols

This chapter describes the software and the protocols used in this project. The software parts cover MQTT, a messaging protocol, the Robot Framework, a test automation tool and Micropython, a programming language. Additionally, this chapter discusses Modbus RTU, a communication protocol.

4.1.1 Robot Framework

Robot Framework is an open-source framework used for test and robotic process automation (RPA) [6]. Robot Framework is used in this thesis because it is already utilized in the automated systems test automation, offers support for various testing methods, and can be adapted to different systems. Robot Framework is used by both individual practitioners and companies in their software development. Some notable companies in Finland that are utilizing this Framework are ABB, Aktia, Norda, Nokia, and Posti [6].

The Robot Framework uses a keyword-based testing approach written in tabular format to create test cases. The test cases are composed in plain text with clear separation through multiple spaces [7]. This framework offers a wide range of built-in keywords and standard libraries that contain these keywords [6]. Moreover, Robot Framework is based on Python and can be further extended by

developing custom libraries using the same or other programming languages such as C or Java [8].

Robot Framework provides detailed and user-friendly outputs generated immediately after test execution. These outputs are accessible via the command line and HTML files, which contain reports, logs, and output reports. The command line output includes the test suite, executed test cases, their pass/fail status, and links to the output files. The figure below illustrates a typical command line output.

```
==============================================================================
Example test suite
==============================================================================
First test :: Possible test documentation                             | PASS |
------------------------------------------------------------------------------
Second test                                                           | FAIL |
Error message is displayed here
==============================================================================
Example test suite                                                    | FAIL |
2 tests, 1 passed, 1 failed
==============================================================================
Output:  /path/to/output.xml
Report:  /path/to/report.html
Log:     /path/to/log.html
```

Figure 5. Robot Framework command line output [8]

The HTML output files offer more detailed information about the executed test suite than the command line output. The output file generated includes all test execution information in the XML format. Based on the output file, log and report files are typically generated. The log file contains a more detailed description of each test case and even provides the success status for each step conducted in the test case. The report file provides an overview of the test execution results and includes links to the log files [8]. The image below illustrates the three different output files.

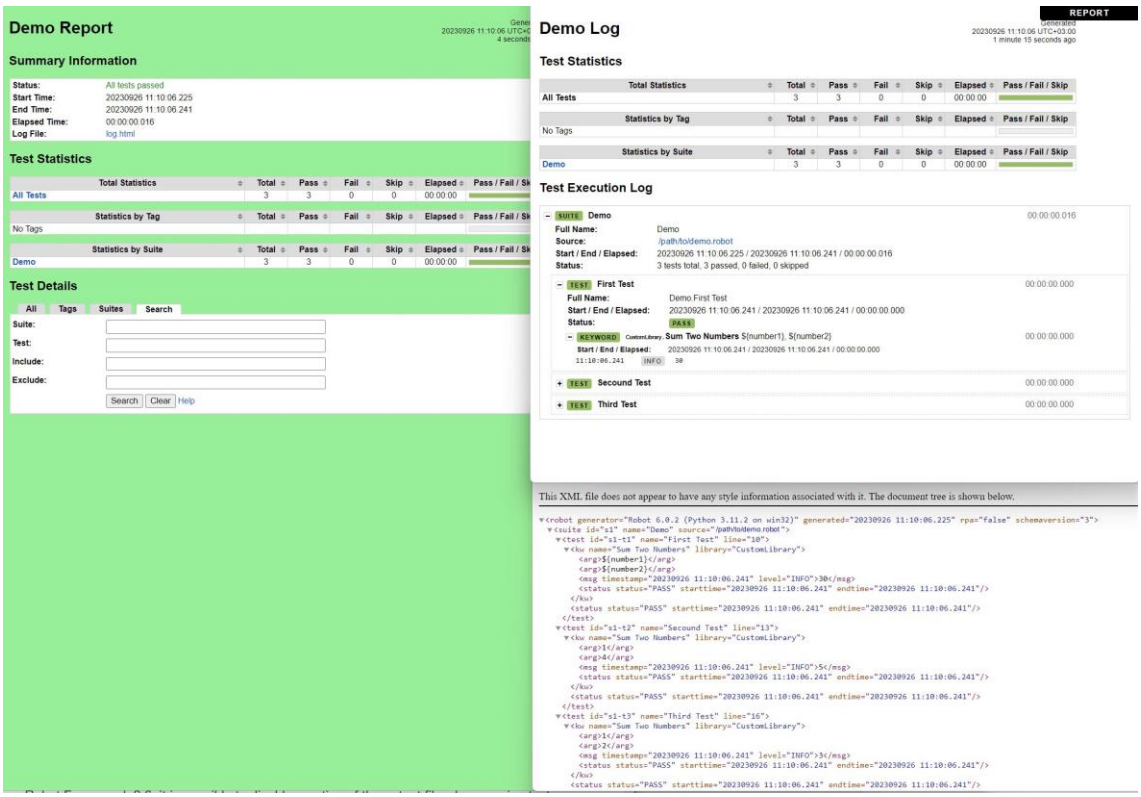

Figure 6. Robot Framework Output Files. The report file is on the left, the log file is in the upper right corner, and the output file is directly below.

4.1.2 MQTT

MQTT is a lightweight, open-source messaging protocol with a small code footprint. Initially, the MQTT stood for "MQ Telemetry Transport". Currently, the MQTT is no longer an acronym and has become the name of the protocol. The "MQ" in MQTT refers to the "MQ series" a product by IBM that supported MQTT [9]. MQTT is a subscribe and publish-based messaging protocol used for communication between various sources, such as machine-to-machine (M2M) and for the Internet of Things (IoT) [9]. This protocol is well-suited for devices or software with limited memory, battery life, and processing power, which must maintain communication with a resource-limited network and bandwidth [10]. This makes it suitable for small microcontrollers such as Raspberry Pi Pico and LPC1549 used in this thesis project.

In MQTT, the software or device subscribing or publishing data, is usually referred to as a client. These clients utilize MQTT libraries, which are available in various

programming languages, including Python, Micropython, and Robot Framework. Notably, these programming languages are also used in this thesis project. These clients are coordinated by a broker. The broker has the responsibility of authenticating and authorizing clients, as well as managing the distribution of the data to correct clients.

The messages in MQTT are expressed in byte format, in contrast to other messaging protocols such as File Transfer Protocol (FTP) and Simple Mail Transfer Protocol (SMTP), which use text-based formats. This byte-based format reduces message size, making MQTT ideal for low-latency networks. [9] The message consists of two parts: topic, a UTF8 string that acts as a message filter, and the data being sent through MQTT. The topics can be expressed in several levels, separated by a forward slash [11]. The figure below illustrates a topic used in this project by LPC1549 when receiving commands from an external source.

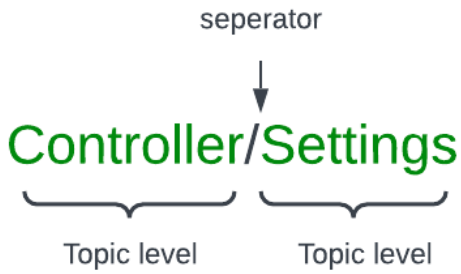

Figure 7. Example of a two-level topic used in this project.

The topics do not require prior initialization because the broker accepts any valid topics [11]. However, it is essential to emphasize that the topics are case-sensitive. To ensure an accurate message distribution, it is crucial to ensure precise spelling. If the topic in the image above is published as "Controller/Settings," a client cannot subscribe to it as "controller/settings."

A client also has the capability to subscribe to multiple topics at once by using wildcards, such as a single level that goes by a plus (+) symbol or a multiple level that goes by a hashtag symbol (#). Single-level wildcard allows replacing any topic in a single topic level. Meanwhile, a multiple-level wildcard can receive all

the topic levels prior to the “#” symbol. The figure below illustrates an example of a single-level wild card.

Figure 8. Description of a single level wildcard [11]

4.1.3 Micropython

Micropython is a lightweight Python programming language. It is optimized for microcontrollers and for constrained environments that demand less power. Meanwhile, Python is designed for larger environments [12]. The software used in the project utilizes Micropython with Thonny, a Python IDE.

Micropython is free, open-source software that includes functional-specific libraries such as MQTT, Modbus, and machine. The machine module contains functions related to the hardware on embedded boards, such as pin, PWM, UART, SPI, and 12C [13].

4.1.4 Modbus RTU

Modbus RTU is a communication protocol that transfers information through serial lines between electronic devices. It is a request-response protocol that uses a client-host relationship. The device initiating the request is a host. In the ventilation system, it is the LPC1549. The devices responding are called client [14], which in the miniature ventilation system is the Raspberry Pi Pico W. The principle for the client-host model is that one device (the host) controls one or more devices (the client). The host is the only device to initiate communication with the client, whereas clients can only reply to the host device. In Modbus, data is sent in the form of a frame and later stored and read from the registers of the Modbus devices.

A Modbus RTU frame usually contains a host address, function code, data, and error check. The address of a specific client is usually a unique number from 1 to 247. The client address is set on both host and client devices. Function codes define the commands to be performed by the host device, such as to read data, write data, or report status. The data contains additional information needed to perform the requested function, such as the data values to be written into the device's memory. The error check represents the Cyclic Redundancy Check (CRC). It checks that the message from the client has been successfully received [15]. The table below embodies a frame sent from the LPC1549 (host) to the Raspberry Pi Pico W (client).

Table 1. Example frame for writing to clients holding register.

Frame Part	**Hexadecimal value**	**Description**
Client Address	0x1	Address to client
Function code	0x6	Write a single holding register
Data	0x0	Register address
	0x40	Data to be written to register
Error Check	0x0	CRC (returns 0 on success)

In the Modbus protocol, there are several types of registers, including input registers, holding registers, coil, and discrete input registers. Input registers are read-only registers used for reading input from the client device. The difference between the input registers is that the discrete input register is a 1-bit register, and the input register is a 16-bit register. A holding register is a 16-bit read-write register. A coil is a 1-bit read-write register, with bit 1 representing ON and bit 0 OFF.

In this thesis project, Raspberry Pi Pico W receives data from LPC1549 through Modbus holding and input registers. The holding register is a register where users

can both retrieve and set values, while the input register can only retrieve. The Raspberry Pi Pico software utilizes the umodbus library, which offers readymade Modbus Libraries for the Micropython programming language.

4.2 Hardware

This chapter describes the hardware used in this thesis project. The hardware that is covered includes the LPC1549 and Raspberry Pi Pico W microcontrollers. The other hardware involved in the project, such as the fan, vent, pressure sensor, motor drive module, and WI-FI module, is not elaborated upon in this thesis.

4.2.1 LPC1549

LPC1549 is a microcontroller from NXP Semiconductors. It is based on the ARM Cortex-M3 architecture and is designed to offer high performance and low power consumption for embedded applications [16]. The microcontroller is equipped with many peripheral connections, making it suitable for many embedded applications. Furthermore, it comes with 4 kB of electrically erasable programmable read-only memory (EEPROM), 256 kB of flash memory, 32 kB of Read-only memory (ROM), and 36 kB of static random access memory (SRAM) [17].

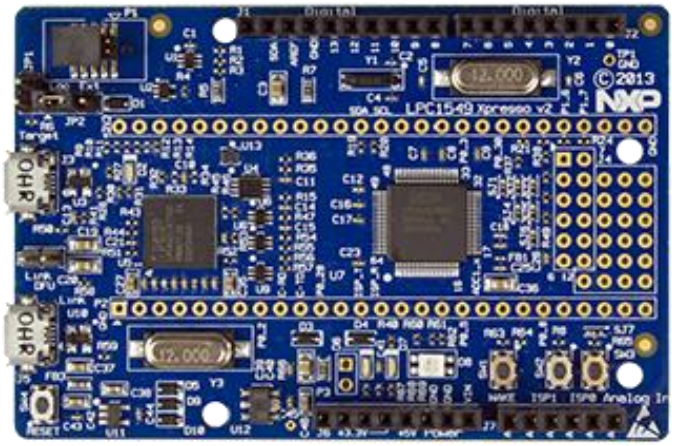

Figure 9. LPC1549 microcontroller(2)

The LPC1549 microcontroller includes various connectivity options, including a USB 2.0, a Universal Synchronous and Asynchronous Receiver-Transmitter (USART), a Serial Peripheral Interface (SPI), and I2C [17]. Other beneficial peripheral connections are Pulse-Width Modulation (PWM), timers, and analogue and digital converters. This makes LPC1549 great for this particular use case because it allows connecting to different devices simultaneously.

Furthermore, LPC1549 supports C and C++ programming languages and can be programmed using the MCUxpresso IDE, which provides advanced editing, compiling, and debugging capabilities [18]. Students of the Embedded System programming course are given the task of programming the microcontroller and therefore, making changes to LPC1549not the primary focus of the thesis. However, the commands received from the microcontroller need to be compatible with the Raspberry Pi Pico W.

4.2.2 Raspberry Pi Pico W

Raspberry Pi Pico W is a low-cost, high-performance microcontroller. It is composed of an RP2040 microcontroller chip [19] equipped with a dual-core ARM Cortex M0+ processor with 264kB of SRAM and 2MB of onboard flash memory [20]. It has various peripheral connections and supports even wireless connectivity.

A notable feature of Raspberry Pi Pico is its support for the Micropython computer language. More comprehensive information on Micropython is found in section 4.1.2. Furthermore, the Raspberry Pi Pico supports C and C++ programming languages, making it suitable for a wide range of embedded system applications.

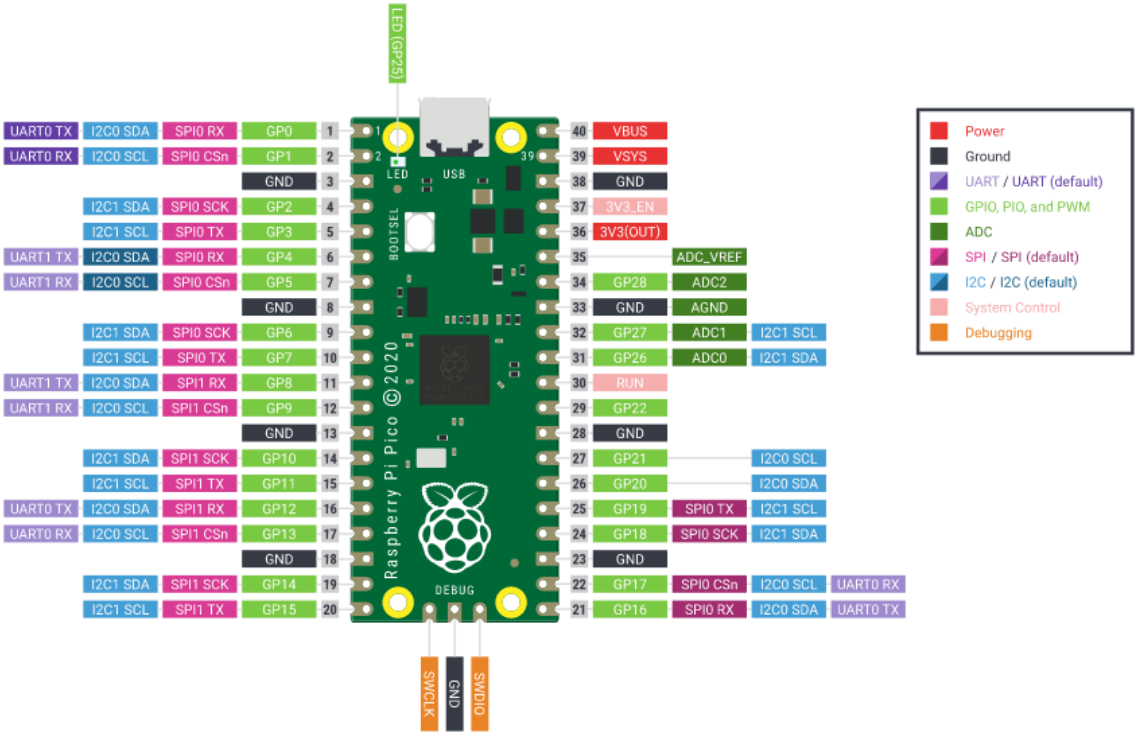

Figure 10. Raspberry Pi Pico and its peripherals connections [20]

Another essential feature of Raspberry Pi Pico is that it has a WI-FI module that allows wireless communication. The module operates with a 2.4 GHz frequency and has a connection protected with a WI-FI Protected Access 3 (WPA3), the latest security certification developed by Wi-Fi Alliance [3]. This allows a more secure wireless connection to the embedded system projects.

Additionally, the Raspberry Pi Pico of the board features several beneficial peripheral connections: USB 1.1 with device and host support, GPIO pins, SPI (Serial Peripheral Interface), I2C, UART (Universal Synchronous and Asynchronous Receiver-Transmitter), and PWM (Pulse-Width Modulation) channels [20]. Thus, the microcontroller should be able to support MQTT and Modbus connections, making the Raspberry Pi Pico W to be a more than ideal microcontroller for this thesis project. It provides a powerful platform for building embedded systems and IoT devices.

5 Results

This chapter covers the results of this thesis project, describing how the miniature ventilation system works and the upgraded Robot Framework test suite.

5.1 Raspberry Pi Pico

One of the objectives of the Raspberry Pi Pico software is to establish a modbus interface that functions like that of the MIO12V. This ensures that the programs developed by students for the LPC1549 microcontroller can seamlessly run on both the miniature and automated ventilation systems without any distinctions.

The Raspberry Pi Pico software contains four main parts: transferring speed received through modbus into a PWM signal and forwarding it to the motor driver module, capturing the pressure sensor value through I2C, connecting to Wi-Fi, and sending debug information through MQTT. The following chapters will describe each part in more detail.

5.1.1 Retrieving Speed Data from Modbus

The Raspberry Pi Pico W modbus connection software was initially developed based on the Brainelectronics RTU Client example on GitHub [21] and was further modified using Micropython-Modbus RTU examples [22]. The result is a class for Modbus RTU Client with the following main functions:

1. **Creating a modbus connection:** This function initializes a modbus connection and sets up necessary parameters such as Modbus address, UART pins, and connection baudrate. This function establishes a connection between Raspberry Pi Pico and the target device.

2. **Setup holding and input registers:** This function allows for the setup of necessary parameters for Modbus holding and input registers, such as the address of the register, the default value, and the callback functions. These

parameters define how data is stored and accessed within the modbus network.

3. **Implementing set and get callback functions:** These callback functions facilitate the setting and retrieval of values from the Modbus registers. The set callback is executed after a new value has been set, while the get function is executed after a value has been requested.

4. **Process:** The process function manages modbus requests and responses, continuously invoking the callback functions.

The image below illustrates the continuous execution of the previously mentioned process function in the Raspberry Pi Pico, which, in turn, persistently invokes the callback functions. Specifically, the Raspberry Pi Pico requests data from the input Register (Igres), which is mapped to register address 4, and currently holds the value of 0. Concurrently, the Holding Register (hgres) callback is triggered whenever the LPC1549 microcontroller at register address 0 sends a new value. Notably, the Holding Register is presently storing a value of 800, denoted in the second-to-last square brackets.

```
Custom callback, called on setting HREGS at 0 to: [700]
Custom callback, called on getting IREGS at 4, currently: [0]
Custom callback, called on getting IREGS at 4, currently: [0]
Custom callback, called on setting HREGS at 0 to: [800]
Custom callback, called on getting IREGS at 4, currently: [0]
```

Figure 11. Raspberry Pi Pico modbus callback messages.

5.1.2 Transforming speed into PWM Signal for Fan Control

In the previous chapter, the Modbus set callback function was introduced. However, it did not cover the fact that this callback function additionally calls a PWM (Pulse Width Modulation) function that enables the fan control. This PWM function plays a crucial role in transforming the speed into a duty cycle and relaying it to the motor driver module. It is important to note that the duty cycle in

Raspberry Pi Pico can vary between 0 and 65535. When the duty cycle reaches a certain threshold, it triggers the rotation of the fan.

The speed received from LPC1549 ranges from 0 to 1000. To convert this speed value into a duty cycle, it must first be translated into a percentage. Subsequently, it is multiplied by the maximum duty cycle of the Raspberry Pi Pico, which is 65535.

The code listing below highlights the essential libraries and functions necessary for the Raspberry Pi Pico to process the data and transform it into a duty cycle. In the example, an input value of 500 is transformed into a percentage by dividing it by the maximum value that will be sent from LPC1549, which is 1000. This percentage is then multiplied by the Raspberry Pi Pico`s maximum duty cycle of 65535.

```
# Importing necessary libraries
from machine import Pin, PWM

# Initialize PWM on pin 26 with a frequency of 1000 Hz
vent = PWM(Pin(26))
vent.freq(1000)

# Input speed value (between 0 and 1000)
input_speed = 500

# Transform speed into duty cycle
duty_cycle = int(input_speed / 1000 * 65535)

# Set the duty cycle
vent.duty_u16(duty_cycle))
```

Listing 3. Example code for transforming speed value of 500 into duty cycle and setting it.

5.1.3 Retrieving Pressure Sensor Data

The SPD600 series pressure sensor communicates its digital output through an I2C interface to the LPC1549 microcontroller. Subsequently, the Raspberry Pi Pico then retrieves this data by monitoring the communication between the pressure sensor and the LPC1549, both of which are using the same protocol.

Within the Raspberry Pi Pico, the data undergoes processing utilizing a PIO (Programmable Input Output) state machine. This state machine enables the

execution of smaller programs in Raspberry Pi Pico to process in parallel with the main tasks, thereby enhancing data synchronization between these activities.

More specifically, the state machine employs the FIFO (First-In-First-Out) method for data transfer. In this context, the FIFO method acts as a buffer or temporary storage for I2C communication data. Its role is to ensure that the captured data remains organized in a queue-like structure, preserving the chronological order of the data.

The data received is then divided by a scale factor provided in the pressure sensors datasheet [3]. The scale factor is used to convert the received value from a specific pressure sensor into Pascal, thereby rendering it more coherent and understandable by the student.

5.1.4 Wireless connectivity and MQTT integration

To establish a wireless MQTT connection for the Raspberry Pi Pico, it must first connect to a network. This network connection is established using the Raspberry Pi Pico's network library. The following listing illustrates the usage of the network library on the Raspberry Pi Pico to establish a wireless connection:

```
# Importing necessary libraries
import network

# network variables
ssid = "the name of the network"
password = "the secret password"

#Create an object WLAN
wlan = network.WLAN(network.STA_IF)

#Turn on the WI-FI on Raspberry Pi Pico
wlan.active(True)

# Connect to the network with the variables created
wlan.connect(ssid, password)
```

Listing 4. Connecting Raspberry Pi Pico to a network

After the connection is established, it is possible to create a wireless MQTT connection on the Raspberry Pi Pico, primarily by utilizing Micropython MQTT

libraries. The software running on the Raspberry Pi Pico will focus solely on publishing debug information.

The thesis project has three distinct kinds of MQTT messages: the debug information from the ventilation system and LPC1549 and the messages from the Robot Framework test suite. The messages are expressed in key-value pairs in the JSON format. The image below illustrates Raspberry Pi Pico publishing a message containing speed and pressure with the topic “ventilationstate.”

```
Client null received PUBLISH (d0, q0, r0, m0, 'ventilationstate',
es))
{"speed": 10.0, "pressure": 0.0125}
```

Figure 12. MQTT messages from the Raspberry Pi Pico.

Figure 13 illustrates the status messages sent from the LPC1549 microcontroller, which uses the topic “controller/status.” The message contains the number of messages sent, the current speed settings, the target speed or pressure, the current pressure settings, the current mode, the control value, and the sensor values.

```
Client null received PUBLISH (d0, q0, r0, m0, 'controller/status', ... (126 bytes))
{"nr": 115, "speed": 5, "setpoint": 5, "pressure": 0, "auto": false, "error": true, "c
o2": -1.00, "rh": -0.10, "temp": -0.10 }
Client null received PUBLISH (d0, q0, r0, m0, 'controller/status', ... (129 bytes))
{"nr": 116, "speed": 10, "setpoint": 10, "pressure": 0, "auto": false, "error": false,
 "co2": -1.00, "rh": -0.10, "temp": -0.10 }
```

Figure 13. Controller status message from LPC1549.

Figure 14 highlights the different MQTT messages from the Robot Framework test suite of the miniature ventilation systems; the topic for these messages is “controller/settings.” These messages contain the mode and speed or pressure settings for the LPC1549 microcontroller.

```
Client null received PUBLISH (d0, q0, r0, m0, 'controller/settings', ... (28 bytes))
{"auto": false, "speed": 50}
Client null received PUBLISH (d0, q0, r0, m0, 'controller/settings', ... (27 bytes))
{"auto": false, "speed": 5}
Client null received PUBLISH (d0, q0, r0, m0, 'controller/settings', ... (28 bytes))
{"auto": false, "speed": 10}
Client null received PUBLISH (d0, q0, r0, m0, 'controller/settings', ... (30 bytes))
{"auto": true, "pressure": 70}
```

Figure 14. Test suites command to LPC1549.

5.2 Robot Framework

The test suite for the miniature ventilation system is designed to test the miniature system`s performance over MQTT communication. These tests are modeled to resemble those carried out in the automated ventilation system and are built upon the existing test automation test suite, utilizing shared functions and libraries. The recent upgrade includes modifying the MQTT Python file to accommodate an alternative broker address and handling messages from the miniature ventilation system. Additionally, a new test suite is created that includes tests that are compatible with the miniature ventilation system.

While the test suite for the miniature ventilation system shares similarities with the automated system, there is a noticeable difference in the number of tests in the test suite. The automated ventilation system includes a set of 12 distinct tests. In contrast, the current ventilation system comprises only four. This variance in the test count can be attributed to the complexity gap between the automated system and the miniature ventilation system.

In particular, the automated system incorporates an MIO-12V module that utilizes dip switches to manage the ventilation operations. A significant portion of the tests in the automated ventilation system are dedicated to evaluating the functionality of these dip switches. However, as the miniature ventilation system lacks this dip switch mechanism, it accounts for the marked difference in the number of tests. There is a brief description below of the tests conducted for the miniature ventilation system.

Test 1 - Send Speed command:

- Send speed value 50 to the LPC1549.
- Wait for 20 seconds to secure the connection.
- Compare the speed values from LPC1549 and the Ventilation System.
- Test fails if the speed difference between both devices is greater than 2.

Test 2 – Read MQTT message:

- Test MQTT connection for the LPC1549.

Test 3 – Check Error status:

- Send speed value 5 to the LPC1549.
- The fan should not start.
- Wait for 45 seconds for the target to adapt to changes.
- Error status from the target device should be set to True.
- Send speed value 10 to the LPC1549.
- Wait for 20 seconds for the target to adapt to changes.
- The Fan should start.
- Error status from the target device should be set to False.

Test 4 – Send Pressure command:

- Send Pressure value 70 to the LPC1549.
- Wait for 20 seconds for the target to adapt to changes.
- Compare the pressure values from LPC1549 and the ventilation system.
- Test fails if the pressure difference between both devices is greater than 3.

Test 3 focuses on testing the status of an error flag. As discussed in section 2.1.3, the error value within the JSON object is depicted as a Boolean value. When it is "True," it can indicate one of three scenarios: either the fan is not activated, the fan speed is not high enough, or the system has been unable to attain the desired pressure. Figure 13, in the previous chapter, highlights how the speed is set to 5, which gives an error flag of true meanwhile. When the speed is set to 10, it gives the error flag false.

6 Summary and Conclusion

This final year project aimed to make a miniature ventilation system, which was to resemble an automated ventilation system. The old ventilation system did not have designated test automation and was not able to support the wireless connectivity required for the test suite.

The reason for the lack of wireless connectivity was that the Arduino microcontroller does not have an inbuilt WI-FI module to support it. The new system replaced the Arduino with a Raspberry Pi Pico W microcontroller, which supports wireless connectivity. The central aspect of this project was to create software for the new microcontroller and to create test automation for the miniature ventilation system.

The requirements for the software were to support MQTT connectivity for the debug information, connect to a network, receive data from the LPC1549 microcontroller, transfer the data to modify the speed of the ventilation system, and receive capture data from the pressure sensor. The outcome of the updates in more detail can be found in Section 5, covering the results of this project.

The latest updates of the miniature ventilation system enable access to debug information through a common wireless interface, reducing the need for physical cables. Additionally, the implementation of the automated tests is expected to make it easier and less time-consuming for the students in the embedded system course when integrating their code with the automated ventilation system.

For future development, one aspect could involve integrating the GMP252 and HMP65 sensors, which are currently missing in the miniature ventilation system. While the absence of these readings may not hinder the core functionalities of the miniature ventilation system, it does restrict the amount of debug information received from the MQTT messages.

Additionally, there is room for improvement in terms of system efficiency, such as in enhancing the Raspberry Pi Pico system by enabling parallel processing of MQTT communication and the collection of the speed data received from the LPC1549. This change could further enhance the performance of the system. It is worth noting that this aspect was not covered in this project but could offer potential for further development.

In conclusion, the implementation of this project, the miniature ventilation system, has taken a significant step towards resembling and functioning more like the more realistic ventilation system.

References

1. Crockford-Härkönen E. Building a Test Automation Suite for an Automated Ventilation Control System. [online]. Available from: http://www.theseus.fi/handle/10024/750737 [cited 9 May 2023].

2. Tonislava D. The complete guide to test automation [online]. Available from: https://www.getxray.app/blog/the-complete-guide-to-test-automation-best-strategies-frameworks-and-tools?source=google&network=g&campaign={campaign}&adgroup={adgroup}&keyword=&placement=&creative=615661433011&utm_medium=cpc&utm_term=&utm_campaign=GG_SEARCH_TOFU+%5BXRAY%5D+-+DSA+Blog+Posts+-+EU1&utm_source=adwords&hsa_acc=9970092548&hsa_cam=17994134125&hsa_grp=141374238713&hsa_ad=615661433011&hsa_src=g&hsa_tgt=dsa-1391637226698&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gad=1&gclid=Cj0KCQjw2eilBhCCARIsAG0Pf8s0st6V_3rKG_kDTIkc-cNklEl9zyScTshlq7pHC6EHOyiiQwlx5MIaArApEALw_wcB [cited 25 July 2023].

3. Sensirion. Differential Pressure Datasheet SDP600Series. [online]. Available from: https://sensirion.com/media/documents/63859DD0/6166CF0E/Sensirion_Differential_Pressure_Datasheet_SDP600Series.pdf [cited 19 September 2023].

4. Vaisala. CO_2 Probe GMP252 [online]. Available from: https://www.vaisala.com/en/products/instruments-sensors-and-other-measurement-devices/instruments-industrial-measurements/gmp252 [cited 9 May 2023].

5. Vaisala. HMP60 Humidity and Temperature Probe [online]. Available from: https://www.vaisala.com/en/products/instruments-sensors-and-other-measurement-devices/instruments-industrial-measurements/hmp60 [cited 9 May 2023].

6. Robot Framework [online]. Available from: https://robotframework.org/ [cited 21 September 2023].

7. Palosirkka. Robot Framework [online]. Available from: https://en.wikipedia.org/w/index.php?title=Robot_Framework&oldid=1080768424 [cited 21 September 2023].

8. Robot Framework. Robot Framework User Guide. [online]. Available from: https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#extending-robot-framework [cited 21 September 2023].

9. HiveMQ. Introducing the MQTT Protocol - MQTT Essentials: Part 1 [online]. Available from: https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/ [cited 22 September 2023].

10. Amazon Web Services, Inc. [online]. What is MQTT? - MQTT Protocol Explained - AWS. Available from: https://aws.amazon.com/what-is/mqtt/ [cited 2023 22 September 2023].

11. Team H. MQTT Topics, Wildcards, & Best Practices – MQTT Essentials: Part 5 [online]. Available from: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/ [cited 23 September 2023].

12. Micropython. Proper Python with hardware-specific modules [online]. Available from: http://Micropython.org/ [cited 9 May 2023].

13. Loker D. Embedded Systems using the Raspberry Pi Pico [online]. 2022. Available from: https://peer.asee.org/embedded-systems-using-the-raspberry-pi-pico.pdf

14. NI. What is the Modbus Protocol & How Does It Work? [online]. Available from: https://www.ni.com/fi-fi/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/the-modbus-protocol-in-depth.html [cited 2023 9 May 2023].

15. Moore Industries. Using MODBUS for Process Control and Automation White Paper Moore Industries [online]. Available from: https://www.miinet.com/images/pdf/whitepapers/Using_MODBUS_for_Process_Control_and_Automation_White_Paper_Moore_Industries.pdf [cited 9 May 2023].

16. Sadasivan S. An Introduction to the ARM Cortex-M3 Processor. [online]. Available from: https://class.ece.uw.edu/474/peckol/doc/StellarisDocumentation/IntroToCortex-M3.pdf [cited 5 September 2023].

17. NXP. LPC15xx [online]. 2015. Available from: https://www.nxp.com/docs/en/data-sheet/LPC15XX.pdf

18. NXP. MCUXpresso Integrated Development Environment (IDE) [online]. Available from: https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-integrated-development-environment-ide:MCUXpresso-IDE [cited 9 May 2023].

19. Raspberry Pi. RP2040 [online]. Available from: https://www.raspberrypi.com/documentation/microcontrollers/rp2040.html#why-is-the-chip-called-rp2040 [cited 9 May 2023].

20. Raspberry Pi. Raspberry Pi Pico and Pico W [online]. Available from: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html [cited 9 May 2023].

21. brainelectronics. GitHub. Micropython Modbus Examples. Available from: https://github.com/brainelectronics/Micropython-modbus/blob/develop/examples/rtu_client_example.py [cited 5 September 2023].

22. brainelectronics. Examples — RTU [online]. Available from: https://Micropython-modbus.readthedocs.io/en/latest/EXAMPLES.html#rtu [c 5 Sept. 2023].