Simachew Tibebu

THE DESIGN OF TEMPERATURE AND HUMIDITY CHAMBER MONITOR AND CONTROLLER
Technology and Communication
2016
The temperature and humidity chamber, (climate chamber) is a device located at the Technobothnia Education and Research Center that simulates different climate conditions. The simulated environment is used to test the capabilities of electrical equipment in different temperature and humidity conditions. The climate chamber, among other things houses a dedicated computer, the control PC, and a control software running in it which together are responsible for running and controlling these simulations.

The Electrical department in Technobothnia Education and Research Center has decided that the control PC is outdated with inconvenient input and storage devices. Also the software was found not easy to navigate through and therefore needed a replacement PC and software.

This thesis project was in an attempt to design a new control PC and the control software, by replacing the old control PC with a Raspberry Pi and developing a new software with better convenience. The main aim was to design and implement a desktop application software that monitors and exposes communication with the chamber to help facilitate control.

An investigation on how the chamber operates and the problems faced by the chamber operator was carried out. As a result, the following functions were included in the newly developed software: A user friendly user interface, a function to monitor the chamber, a function that exposes communication with the chamber to facilitate control and storing test and test result information in an easy to access manner.

The software was written using C++ programming language and Qt application framework, using Ubuntu, Linux for development environment and the Qt Creator integrated development environment, IDE.

The goals of this project were achieved.
CONTENTS

ABSTRACT

1 INTRODUCTION ................................................................................................................. 9
  1.1 Background .................................................................................................................. 11
  1.2 Objective ..................................................................................................................... 12
  1.3 Software Development Model .................................................................................... 13

2 RELEVANT TECHNOLOGIES ......................................................................................... 16
  2.1 C++ ............................................................................................................................... 16
  2.2 The Qt application framework ..................................................................................... 16
    2.2.1 Serial Communication in Qt .................................................................................. 17
    2.2.2 Qt Signals and slots .............................................................................................. 18
    2.2.3 The Qt Model/View Architecture ......................................................................... 19
    2.2.4 Serialization and File Processing in Qt ................................................................. 22

3 UNDERSTANDING THE OLD SYSTEM ......................................................................... 23
  3.1 Test Programs ............................................................................................................. 23
  3.2 Steps ............................................................................................................................ 23
  3.3 Cycle ............................................................................................................................ 23
  3.4 Devices in the chamber ............................................................................................... 24
  3.5 Communication .......................................................................................................... 25
  3.6 Controlling Tests ........................................................................................................ 28

4 SOFTWARE DESCRIPTION ............................................................................................... 30
  4.1 Project constraints ...................................................................................................... 30
  4.2 Quality Function Deployment ..................................................................................... 31
  4.3 Classes and Their Relationship .................................................................................. 32
    4.3.1 Communication Class .......................................................................................... 33
    4.3.2 ControlCommands Class ..................................................................................... 33
    4.3.3 Chamber Class ..................................................................................................... 33
    4.3.4 Program Class ...................................................................................................... 33
4.3.5 Step Class .......................................................... 33
4.3.6 PID Class .......................................................... 33
4.3.7 Controller Class ...................................................... 34
4.3.8 DataBackup Class ............................................... 34
4.3.9 Model Classes ....................................................... 34
4.3.10 Dialog Classes .................................................... 34
4.3.11 MainWindow Class ................................................. 35

5  GRAPHICAL USER INTERFACE DESIGN .......................... 37
   5.1 General Design Approach ......................................... 37
   5.2 Design of Different Parts of the GUI ............................ 38
      5.2.1 Main Window ................................................. 38

6  IMPLEMENTAION .......................................................... 48
   6.1 General description of Implementation .......................... 48
   6.2 Implementation of Iterations ...................................... 48
      6.2.1 Iteration: Serial Communication and Monitor .......... 48
      6.2.2 Iteration: Store Test Programs and Display .......... 54
      6.2.3 Iteration: Expose Control Command Messages ......... 60
      6.2.4 Iteration: Store PID Parameters and Display ......... 62
      6.2.5 Iteration: Settings .......................................... 63
      6.2.6 Iteration: Running Test Programs ....................... 64
      6.2.7 Iteration: Plotting Tests .................................. 68
      6.2.8 Iteration: Quick Test ....................................... 68

7  TESTING ........................................................................... 70

8  CONCLUSION ................................................................. 71
   8.1 Future Work .......................................................... 71

REFERENCES ....................................................................... 73

APPENDICES
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>Personal computer</td>
</tr>
<tr>
<td>EUT</td>
<td>Equipment Under Test</td>
</tr>
<tr>
<td>UI</td>
<td>User interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface</td>
</tr>
<tr>
<td>RPI</td>
<td>Raspberry Pi</td>
</tr>
<tr>
<td>I/O</td>
<td>Input output</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>MV</td>
<td>Model View</td>
</tr>
<tr>
<td>MVC</td>
<td>Model View Controller</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PID</td>
<td>Proportional, Integral and Derivative</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>H1</td>
<td>No.1 Compressor</td>
</tr>
<tr>
<td>H2</td>
<td>No.2 Compressor</td>
</tr>
<tr>
<td>T2</td>
<td>No.2 Heater</td>
</tr>
<tr>
<td>T1</td>
<td>No.1 Heater</td>
</tr>
<tr>
<td>C1</td>
<td>No.1 Compressor</td>
</tr>
</tbody>
</table>
C2  No.2 Compressor

V1  No.1 Refrigerant loop valve

V2  No.2 Refrigerant loop valve

V3  No.3 Refrigerant loop valve

V4  No.4 Refrigerant loop valve

FN1  Condenser cooling Fan

LNV  Extra cooler-gas valve

P1  No.1 Time Signal

P2  No.2 Time Signal

P3  No.3 Time Signal
LIST OF FIGURES

Figure 1. Inside of the climate chamber.............................................................. 10
Figure 2. The control PC and the control software.............................................. 12
Figure 3. The Agile software development model. /2/..................................... 14
Figure 4. Abstract connections of signals and slots of objects. /9/ ...................... 19
Figure 5. Data representation of a model in Qt MV architecture /10/ .............. 21
Figure 6. The old climate chamber.................................................................... 24
Figure 7. Climate chamber after new control PC installation........................... 25
Figure 8. Message format of communication between the control PC and control
box. /13/ ........................................................................................................... 25
Figure 9. The communication between the control PC and control box........... 26
Figure 10. Eight bytes of the control command.................................................... 28
Figure 11. Block diagram of a PID control system............................................. 29
Figure 12. Class diagram of the application........................................................ 32
Figure 13. Layout of QMainWindw layout /15/ .................................................. 36
Figure 14. The Monitor tab ................................................................................ 39
Figure 15. The Programs tab .............................................................................. 40
Figure 16. The Aux tab....................................................................................... 41
Figure 17. The Options tab................................................................................. 42
Figure 18. Control params page of the Options tab.......................................... 43
Figure 19. Quick start tab................................................................................... 45
Figure 20. Add step dialog.................................................................................. 46
Figure 21. Add program dialog.......................................................................... 47
Figure 22. Flowchart of communication.............................................................. 51
Figure 23. Sequence diagram of creating a new test program........................... 56
Figure 24. Program File content.......................................................................... 57
LIST OF CODE SNIPPETS

Code Snippet 1. Function to open the serial device......................... 49
Code Snippet 2. Function to write data to on to the serial device. ............. 50
Code Snippet 3. Function to read data from the serial device. .................. 50
Code Snippet 4. Function to handle received data. ................................ 52
Code Snippet 5. Function to handle message containing chamber status information.............................. 53
Code Snippet 6. The dry temperature and humidity setters...................... 54
Code Snippet 7. Chamber value changed signals connection to GUI slots....... 54
Code Snippet 8. Function to save a Test Program.................................. 56
Code Snippet 9. Appending a step to a test program file. ......................... 57
Code Snippet 10. Reserving space for program in the model. .................... 58
Code Snippet 11. Setting column and row headers. .............................. 58
Code Snippet 12. Data to fill in the table view of steps. ......................... 59
Code Snippet 13. Definition of H1, H2, T1 and T2 along with the HT block of size one byte................................................................. 60
Code Snippet 15. Changing the bit corresponding to the peripheral device..... 61
Code Snippet 16. Assigning power of heater one.................................... 62
Code Snippet 17. Stream insertion operator......................................... 63
Code Snippet 18. Stream extraction operator....................................... 63
Code Snippet 19. Setting application name, domain and organization.......... 64
Code Snippet 20. Persisting Serial Port name to settings.......................... 64
Code Snippet 21. Initiating test start.................................................. 65
Code Snippet 22. Function called when the plotTimer interval runs out........ 66
Code Snippet 23. Controlling the test................................................ 67
Code Snippet 24. A function that calculates the control output............... 67
Code Snippet 25. Part of a function to control peripheral device operation state and power.............................................................. 68
1 INTRODUCTION

Environmental test chambers, also known as climate chambers, are used to ensure the reliability of industrial products, especially electronic devices, by subjecting them to a prolonged change of one or more environmental parameter. /1/

Such a climate chamber is in use by the Electrical Department of Technobothnia Education and Research Center. It is used to run environmental tests on clients’ products. The environmental parameters in this climate chamber are temperature and humidity. This climate chamber houses an EUT-chamber or just a chamber; an empty box where changes in temperature and humidity are manufactured and also where the device to be tested sits in, sensors; that are used to measure the temperature and humidity inside the chamber, peripherals; heaters, coolers, and humidifiers that directly manipulate the climate inside the chamber and others like compressor fan, time signal inputs, refrigerant valves and cooler-gas valves, a control box; a control device that manages the on/off sate and working power of the peripherals and a control PC; a computer that runs a single software which monitors and controls the state of the chamber by communicating with the control box via an RS-232 serial protocol. This thesis project is mainly focused on replacing this control PC and the software installed in it. In Figure-1 bellow, the control PC and control box inside the climate chamber can be seen.
Figure 1. Inside of the climate chamber.
1.1 Background

The software in the control PC accepts the desired climate conditions to be simulated as user input. These climate conditions that are to be simulated and run as environmental test are stored either permanently or temporarily into the computer, and are called Test programs. The software monitors the state of the chamber, meaning it displays the real time temperature and humidity of the chamber, at all times. In addition, when running test programs, it makes decisions on the operation power and state (on/off state) of the peripherals in the climate chamber. These decisions are made based on calculation (control algorithm) made using the real time temperature and humidity values that are reported by the control box. The decisions are reported back to the control box which in turn commands these decisions to be executed by the relevant peripherals.

The control PC is also responsible for storing the test programs. After tests are run information about the execution, the change of temperature and humidity through time, is stored in the form of a graph.

The operating system running in the control PC and hosting the application software is a 16 bit Microsoft Windows 3, which is out dated and no longer supported. The medium of storage is a floppy disk drive. This computer has a small monitor and a keyboard with a number pad and function keys. The computer has only a serial port, a floppy drive reader and no other I/O ports. The monitoring and controlling software have received complaints for its unfriendly and hard to navigate UI as well. Figure 2, shows a control PC as can be accessible to the operator.
The department took notice of the inconvenience that the control PC and its software has for the operating personnel and had to look for a solution. With this project a solution to replace the control PC with a Raspberry Pi and a compatible monitoring and controlling software to bring ease of operation was proposed.

1.2 Objective

The objective of this project is to develop a software program, named and referred to as the climate chamber monitor and controller, or shortly chamber controller, that will monitor the chamber and communicate with the control box to facilitate control. This software, the chamber controller, is to be installed and run on a Raspberry Pi (RPi) computer. Ultimately the objective of this project is to replace the control PC and control software of the climate chamber.
The RPI will be running a Linux distribution, named Raspbian, as an operating system. The software solution is a desktop application that will replace the old software. It will be running inside the Raspbian operating system and will replace most functions of the old software.

1.3 Software Development Model

The communication between the control box and the control PC takes place through a protocol that is strictly defined and followed by the two parties. Even though this project replaces the control PC of the chamber it still needs to cooperate with the control box, making this software closely tied and dependent on the protocol of the messages being communicated. The meanings of the messages were investigated, and it was apparent that the software development could change direction up on discoveries. It was necessary to keep inspecting the communication and also the client requirements and adopt the development to the results. Agile methodology was chosen for development as it best suits the inspect and adopt approach. Using Agile and its iteration concept also provided an opportunity to work on a concept while not knowing the full details of another concept that could build on top.
Figure 3. The Agile software development model. /2/
2 RELEVANT TECHNOLOGIES

In this section technologies that are used to build the climate chamber monitor and controller software are reviewed.

The software is written using the C++ language and the Qt application framework. It is developed using the Qt Creator IDE in Ubuntu 14.04.

2.1 C++.

C++ is a general purpose programming language that dates back to 1979. It was going by the name “C with class” until 1983, before it was renamed C++. Though the language was not standardized till 1998, some unofficial but widely accepted references to the language ware published since 1985. Currently C++ is an open ISO-standardized language by a committee of ISO. In the course of its lifetime C++ has grown to be one of the most frequently used programing languages in the world. This success, among other things, is due to the features that the language supports. /3//4/

Some features of the programing language include: it is a compiled language meaning it compiles directly to a machine native language making it to be fast, if used wisely, it is a strongly-typed unsafe language; putting strong restrictions on implicit conversions of types to one another but resulting in good control, It supports both manifest and inferred typing; allowing some degree of flexibility and avoiding wordiness, It supports both static and dynamic type checking; allowing type conversions to be checked both in compile-time and run-time; which also adds flexibility. In addition, having a wide range of compilers that run in different platforms makes the language portable. /5/

2.2 The Qt application framework

“Qt is used for device creation, UI and application development. It supports deployment to over a dozen platforms in desktop, mobile and embedded. Qt as a
whole is composed of the Qt framework, Qt libraries and Qt development tools, including an IDE, Qt Creator IDE, and other productivity tools."

The Qt application framework is one of the tools provided by Qt. It is a C++ library separated into several modules that together can be used to develop applications that run on multiple supported platforms. These modules are classified into three collections of modules: Qt Essentials, Qt Add-Ons and Value-Add Modules. The essentials contain general purpose modules that are available for all supported platforms. Examples are: Qt Core, Qt GUI and Qt Network. The add-ons bring additional use for specific purposes and may only be available on some platforms. Examples are: Qt Bluetooth, Qt Graphical Effects and Qt Serial Port. The value-add modules are modules that build on top of the Qt libraries which also provide additional value but have their own separate release and licenses. Some examples are: Qt Charts, Qt Purchasing and Qt Virtual Keyboard.

Qt also has a detailed documentation of its libraries. The documentation and the source of Qt are available from the Qt website.

### 2.2.1 Serial Communication in Qt

Serial communication in Qt is provided through the Add-On module Qt Serial Port. The Qt Serial Port module provides basic functionality for serial communication including configurations, I/O operations and control over the control signals of the RS-232 pinouts. This module contains two C++ classes namely QSerialPort class which provides access to serial ports and QSerialPortInfo class which provide information about serial ports existing in the system.

In building this application the QSerialPort class has been used to provide communication between the application and the control box in the chamber.
2.2.2 Qt Signals and slots

Qt’s signals and slots mechanism is a means of communication between objects. In programming, usually in GUI programming, when a state of an object, a widget, is changed the programmer might want to notify this change to other objects. In other tools this is achieved using callbacks. Callbacks have their drawbacks, mainly they are not type safe and they are strongly coupled. The programmer cannot be sure that the callback will be called with the correct arguments and that the processing function must be aware of the callback in the beginning. /9/

Qt’s signals and slot provide an alternative to callbacks that is free of the drawbacks. Signals are emitted when an event occurs or a state of an object changes. A slot is a function which is can be called in response to a signal or just as a member function. Qt’s widgets and classes have pre-defined slots and signals, but it is also possible to subclass these classes to define one’s own slots to handle signals. /9/

The signals and slots mechanism is type safe: the signature of the slot must have equal or less number of arguments as that of the signal. It can have less, because it can ignore arguments of the signal. Signals and slots are loosely coupled: a signal neither knows nor cares which slots receives it. Similarly, a slot neither knows or cares, what signals it is responding to. A signal can be connected to as many slots as needed and so can a slot. When a signal is connected to a slot the slot will be called with the parameters of the signal when the signal is emitted. The following figure shows connections of signals and slots of different objects. /9/
All classes that inherit from QObject or any of its subclasses can use the signals and slot mechanism. A class that contain signals and slots must declare the Q_OBJECT macro at the top of its declaration. /9/

2.2.3 The Qt Model/View Architecture

The Qt model/view architecture is a paradigm that creates a separation between the way data is stored and the way it is presented to the user. This separation of functionalities provides the developer with flexibility to interface data from a

Figure 4. Abstract connections of signals and slots of objects. /9/
wide range of data sources and customize their presentation. This separation makes it possible to present the same data in multiple types of views without changing the underlying data.

The M/V classes can be categorized into three groups, Models, Views and Delegates. The model communicates with the source of data and provides an indexed interface that can be used by other components of the architecture. The view retrieves data from the model using indexes and presents it. The delegate handles user interaction with the view and editing of the model. Communication between these components of the architecture happens through signals and slots.

### 2.2.3.1 Models

Models provide an interface that exposes the underlying data to be accessed by the view and delegate. The data itself doesn’t necessarily have to be stored in the model; it can be kept in a separate data structure, file, database or any other form of repository accessible to the data.

All models are based on the `QAbstractItemModel` Class. This class provides an interface that can be handled by views like trees, lists and tables. Qt provides convenient abstract classes `QAbstractListModel` and `QAbstractTableModel` to implement when working with list and table-like data structures respectively. Some ready-made model classes also come with Qt. For example: `QStringListModel`, used to store and present `QString` items and `QFileSystemModel` which provides information about directories and files in the local machine.

However, the items of the underlying data are structured. Qt models represent the data in a table-like hierarchical structure. Each item in the data model is represented by a `ModelIndex`. The concept of model indexes create separation by decoupling the view and the delegate from the underlying data. By providing a model index to the model, views and delegates get the information held by an item in the model. On the other hand, the model signals the view and delegate about changes made in the data. The model index holds a temporary reference to the in-
formation it represents and a pointer to the model that created it. Since the model data can rearrange in the life of the application model indexes are temporary. /10/

QModelIndex class provides model indexes. Rows, columns and the parent model index are properties of a model index.

![Figure 5. Data representation of a model in Qt MV architecture /10/](image)

### 2.2.3.2 Views

Views present the data that is interfaced by the model. The style of the view’s data presentation is completely independent of the style of the model’s data representation. For example, a table model can be presented by any of the view styles such as a list or a tree view. Model indexes are used to inquire a specific information stored in an item of the model. /10/

In addition to presentation, views also support other basic user interface features such as navigation through items, selection of items, context menus and drag and drops. Selection of items, by the user or in code by the developer, can be tracked
either separately by a single view or by multiple views that share a model. Some
views also display headers. /10/

Qt, as in models, also provides ready-made view classes. QListView,
QTableView and QTreeView. As their names suggest these models present items
from a model in the form of simple lists, tables and hierarchically nested lists. /10/

2.2.3.3 Delegates

Delegates are responsible for managing interactions with the user. Editing of
items in the view by the user is presented and persisted to the model through dele-
gates. The base class for delegates in the MV architecture is QAbstractItemDele-
gate. To allow user editing items in the view, some input methods are provided
and hence delegates are sometimes expected to render themselves when called up
on. /10/

2.2.4 Serialization and File Processing in Qt

Qt exposes the file system as a device that supports reading and writing data. The
QIODevice class serves as the base class that provides abstract interfacing as well
as common implementations to all I/O devices. The I/O devices can be a file, data
stream, memory buffer or a serial port. Other classes such as QFileDevice, QBlue-
toothSocket, Buffer and QSerialPort inherit QIODevice to provide a more specific
function to what they are dedicated to. QFileDevice provides a base interface to
reading and writing in to open files. More specific and applicable classes are
QDir: which works with directories and QFile: which works with files. QFile can
be used alone but using it with QTextStream and QDataStream stream classes
gives more convenience. /11/

In addition to working with QFile to read and write in to files, QDataStream can
be used to serialize objects. Some Qt data types are ready-made for serialization.
Custom objects can be made serializable by implementing two operators of
QDataStream class, the operators ()<< and ()>>. /12/
3 UNDERSTANDING THE OLD SYSTEM

As the objective this project, in general, is to replace the old system, understanding the old system is necessary. Studying the old system helped to decide how features should be implemented, what entities are involved in the old system and which ones should exist in the scope of this project.

A study of the communication protocol that includes commands that should exist in the scope of this project was provided by an independent party.

The following entities exist in the old system and are also retained in the new.

3.1 Test Programs

Test Programs are representations of environmental tests that can be carried out by the climate chamber. They contain information about the name of the test program, how many steps it has (steps are discussed in the next section) and how many cycles (cycle is discussed in section 3.3) the test has.

3.2 Steps

Steps are parts of a test program that primarily represent three conditions of the test. These conditions are the final environmental conditions (temperature and humidity) to be achieved called the set values, the time available to achieve it and information about what to do if the available time was not enough, called wait status. A test program normally contains multiple steps and the test goes from one step to another following the temperature and humidity set values and consequently changing the climate inside the chamber.

3.3 Cycle

Cycle represents how many times the test program runs. For example, if a test program has five steps and has three cycles, the software executes the steps ac-
cording to their sequential order and when it has finished running the fifth step, it repeats all the steps in their order until it has have done it three times.

### 3.4 Devices in the chamber

The climate chamber houses devices other than the control box. Some of the devices are directly and physically responsible for the climate change that occurs in the chamber. These devices are heaters, coolers and humidifiers. Wet and dry temperature sensors placed inside the chamber measure the temperature inside the chamber and the control box determines the humidity using these values.

![Figure 6. The old climate chamber](image)

---

[Diagram of the climate chamber with labeled components such as Control PC, Internal Control Box, EUT Chamber, Sensors, Heaters, Humidifier, Coolers, Valves.]
Valves are used to open and close ways from the cooler to the chamber.

### 3.5 Communication

A study of the communication between the control PC and the internal control box has been made. This serial communication has follows a baud rate 9600, stop bit 1, 8 data bits and no parity bit rules.

The message format in the communication is as shown in figure 8 below.

![Diagram](image)

**Figure 7.** Climate chamber after new control PC installation

**Figure 8.** Message format of communication between the control PC and control box. /13/

As figure 8 above shows the messages in the communication is partitioned in to blocks. Each block and what it represents is as follows:
- STX; start of text, 02 Hex
- ID1; the first id of the message, 30 HEX – 39 Hex
- ID2; the second id of the message, 40 Hex – 4F Hex
- COMMAND; The command to send to the receiving machine.
- EXT; end of text, 03Hex.
- CHKDSK; checksum, STX + ID + Command + EXT
- END; end of the message, 0D Hex.

The communication goes back and forth with four repetitive messages. The control PC sends one message, waits for the reply and sends the next message until it has sent and received all four messages and this repeats till the application is shut down.

All messages communicated between these two components are similar to the messages in figure 9 above. One message, sent from the control box contains only ACK (06 Hex). The rest of the messages have value zero (30Hex) for their ID1 and values A (41Hex), B (42Hex), I (49Hex) or O (4FHex) for their ID2. These messages are referred ACK or by their ID2 from here onwards.

In figure 9 above the first message line, Tx1, (O message) shows a message sent from the control PC. The next line, Rx1, (ACK) is a reply to the first message.
from the control box. The lines continue to alternate between messages from the control PC and the control box, in the same way as the first two lines.

The most important messages in this communication are the messages at Tx1, message O, and Rx3, message A, row. Message A, coming from the control box informs the PC of the dry temperature, wet temperature and humidity in the chamber, in this case 23.63, 12.38 and 23.4 respectively. This information is critical to the control PC as it is used to display the state of the chamber and it is based on this values that the PC decides on what to do next while running a test.

Tx, message O, is sent from the control box. Included in this message are the control commands that tell the control box what to do next. This message contains 26 bytes of information. Six bytes are reserved for etx, id1, id2, etx, checksum and end blocks of the message, each taking up one byte of space. The rest of the 20 bytes are taken by the command block of the message, the control commands. The first 12 bytes of the control command block are set to zero in ASCII (30 Hex). The next 8 bytes represent the operation state and operation power of devices in the climate chamber. Each peripheral device is represented by a bit in these bytes. At least four devices are represented in one byte. The representation of each device starting from the 13th byte, sequentially, is as follows:

- H1, H2, T2, T1; Humidifier 1 and 2 and Heaters 2 and 1 respectively.
- P3, P2, P1, LNV; Time Signals 3,2,1 and extra cooler-gas insertion valve
- C1, V4; Compressor 1 and refrigerant loop valve 4
- V3, V2/C2, V1, FN1; refrigerant valve 3, valve 2 and compressor 2 represented by one and same bit, Condenser cooling fan
- Humidifier power 1; the power at which the first humidifier works.
- Humidifier power 2; the power at which the second humidifier works.
- Heater power 1; the power at which the first heater works.
- Heater power 2; the power at which the second heater works.
The eight bytes containing control commands are arranged as shown in figure 10 below.

**Figure 10.** Eight bytes of the control command.

### 3.6 Controlling Tests

The environmental test executed is controlled by an algorithm that implements the PID control system theory. A PID (proportional-integral-derivative) controller system theory is based on a feedback loop. The controller takes a measured value from a process and compares it with a reference set value. The deference, also called error, is used to calculate a new input to the process (output of the controller) to bring the measured value closer to the set value. The PID controller can also adjust its control based on the history and the rate of change of the error. /14/
Figure 11. Block diagram of a PID control system.
## 4 SOFTWARE DESCRIPTION

The Climate Chamber Monitor and Controller software is what this project aims to build. It is a desktop application that will be installed and run in a Raspbian, Linux distribution, OS. It is intended to serve as monitor for the state of the chamber; by displaying temperature and humidity inside the chamber and facilitate communication with the chamber to accommodate control; by allowing messages that contain information about the operation state and power of devices to be sent.

The climate chamber monitor and controller software needs to work with the control box of the chamber to operate. Massages need to constantly be sent and received to and from the control box. Messages received from the control box contain information that need to be interpreted and acted upon accordingly. This information is used to display the status of the chamber, specifically the temperature and humidity inside the chamber and to perform calculations and make a decision on the operation state of the climate chamber. Control decisions like whether the temperature & humidity needs to be increased or decreased and what devices should be on or off are made based on this information. The decision is in turn communicated back to the control box as control commands. The software also notifies the user about these decisions by displaying information about them.

### 4.1 Project constraints

Having a nature of reverse engineering, this project became subject to restrictions imposed on it by the characteristics of the old system. The protocol of communication between the software and the control box has already been defined by the old system. Therefore, in developing this software it is necessary to adopt the already existing protocol which the control box understands. Messages must also be transmitted via an RS-232 compliant cable, since the control box only has a serial port. Another restriction on this project is due to the choice of platform on which the finished product is expected to run on, that is Raspberry Pi. Though the Pi is
an easily available and powerful device, developing on the Pi lacks variety when it comes to choices.

4.2 Quality Function Deployment

The functionalities defined by the requirement are grouped in to three groups. The must haves with high priority, good to haves with medium priorities and nice to have features with low priorities.

Must have features:

The software must be:

- Able to run in a Raspbian OS of Raspberry Pi.
- Able to receive messages from the control box and interpret them to temperature and humidity values
- Able to send control commands to the control box.

Using the software, the operator must be able to do the following

- Save test program information.
- Select and run saved test programs.
- To view the status of the climate chamber’s components.
- View the temperature and humidity inside the chamber.
- Access saved test programs via a USB port.

Good to have features:

User must be able to:

- Follow the progress of test program execution.

Nice to have features:

The software should:
- Store test execution progress.
- View tests that have been executed in the past.

The software should allow the user to:

- View list of tests that have been executed in the past.
- View past tests as plot of changes of temperature and humidity in time.

4.3 Classes and Their Relationship

The climate chamber controller application is built using the object oriented programming paradigm. Entities, tasks and GUI components in the software are represented with object models. Classes that make these objects are shown in figure 12 below. The Qt M/V architecture is also used to present data to the user and handle user’s interaction through the GUI.

![Class diagram of the application.](image-url)

**Figure 12.** Class diagram of the application.
4.3.1 Communication Class

Communication class is used for accessing the serial port which is interfaced as a device by the QSerialPort class. It provides a connection to the control box and functions to write and read messages to and from the serial port.

4.3.2 ControlCommands Class

The ControlCommands Class provides interface to represent messages that are used in communication between this software. Messages are represented in their format defined by the communication protocol. Other classes in the software can gain access to the bodies of the messages if they need to modify the message.

4.3.3 Chamber Class

The Chamber Class is a representation of the EUT chamber’s state. It mainly represents the real time temperature and humidity of the chamber.

4.3.4 Program Class

The Program Class represents and holds all the information of an environmental test, the name of the program, its cycle and its steps. It will need to create and hold objects of the Step class to represent its steps.

4.3.5 Step Class

The Step Class is a representation of one single step in a test program. It represents what temperature and humidity need to be achieved in a test, how long a time is available for executing the test and the wait status.

4.3.6 PID Class

The control system used for controlling the progress of tests in the chamber is PID control. The PID class is a representation of the PID parameters that the controller needs to work with. This class also provides the function where a control calcula-
tion is performed. Operators (>>) and (<< of the QDataStream are also imple-
mented for this class to make it serializable.

4.3.7 Controller Class

This class is where all the logics in the other classes are incorporated in. This class
creates objects of the Program class, the ControlCommands class, the Chamber
Class and two objects of the PID class. The fact that control calculations are per-
formed in the PID class allows for it to happen for both temperature and humidity
with no repetition of code.

4.3.8 DataBackup Class

The DataBackup class, as the name suggests, is the class responsible to persist da-
ta to storage. All data that this application stores are saved in to a file. This class
gives functions to save data on to File.

4.3.9 Model Classes

There are two classes in this software that serve to represent data as a model in the
M/V architecture of this software. These two classes are the StepsModel class and
the PidListModel class.

StepsModel inherits from QAbstractTableModel. It is used for modeling a collection
of Steps in a test Program in to a table model, which later can be used by the
GUI to create a visual representation of the steps.

PidListModel inherits from QAbstractListModel and is used for creating a list
model of PID parameters that the application has saved. This list model of the PID
parameters can also be used by the GUI to visualize the data.

4.3.10 Dialog Classes

Some UI classes in the software are inherited from QDialog. These classes are
AddPid, AddProgram, AddStep, LoadProgram and RenameDialog. These dialog
classes are serving as means of commination with the user. LoadProgram class is used to view a test program in a separate pop-up window. The reset of these classes are used for collecting information from the user, by presenting a form to be filled and submitted by the user. The information submitted with the form is used to create a new test Program, Step, PID, or a new name to an already existing test program.

4.3.11 MainWindow Class

The MainWindow class is the class that provides the framework for building the UI of this application. It is the main window of the application and all the other GUI parts of the application are built on top of it. MainWindow is inherited from QMainWindow.

QMainWindow class has a layout to which one can add useful components like a toolbar, a status bar or a menu bar. QMainWindow also has a central area that any type of a widget can occupy. Figure 13 below shows the layout of a QMainWindow and places where other widgets can occupy inside of it. This central area is where all the widgets in the main window of this application are placed.
Figure 13. Layout of QMainWindow layout /15/
5  GRAPHICAL USER INTERFACE DESIGN

The graphical user interface of this software is built with Qt C++ Widgets and the Qt M/V architecture is used when necessary. Qt Designer tool is used to create and edit these widgets.

Qt Designer is a tool provided by Qt that comes as stand-alone tool or integrated in to the Qt Creator IDE. Qt Designer allows the developer to design and build UIs by assembling and editing readily available Qt widgets in to the main window. Following the developer’s designs, the tool creates an XML format UI definition (.ui) file. This file is compiled by the User Interface Compiler of Qt to create a corresponding C++ header file. Behaviors of Qt Designers UIs are managed with signals and slots which integrates them seamlessly with the rest of application code. /16/

5.1 General Design Approach

The UI of this application is designed to bring ease of use as much as possible. Navigation and using the application is made to be similar to a regular website, with familiar UI components that present familiar means of navigation, selection and interaction.

A uniform look and feel is given to the application by introducing a color theme. The color theme contains three main colors:

- A shade of Light sky blue; used as background color of the main window of the application.
- A light steel blue; used to shade the area of the application that is of interest to the user and a selected tab.
- A shade of orange; which is used to mark selections in the lists.
5.2 Design of Different Parts of the GUI

The GUI in this application is designed and built with widgets that come with Qt. Building custom widgets was not necessary. In the following section the building of some parts of the GUI is discussed.

5.2.1 Main Window

Central area of main window of the application, where all other widgets can occupy, is filled with a QTabWidget. QTabWidget is a widget class that provides a group of widgets that are tagged with a tab. It includes a tab bar and stack of pages, where each page is related to one tab. /17/

The tabs in the GUI of this application are arranged at the top of the widget. There are four tabs each representing a group of related functionalities of the application. These four tabs are the Monitor tab; used to provide monitor to the climate chamber and a progress of environmental test, The Program tab; dedicated to functionalities related to test programs, The Aux tab; dedicated to preview the history of test programs that have been executed before, The Options tab; where functionalities related to global settings of the application and the climate chamber are provided and The Quick Start tab; where a quick test can be executed, on the fly, without having to save in to the file system.

5.2.1.1 Monitor Tab

The monitor tab, as the name suggests, is where the user can monitor the state of the climate chamber and a test progression. It is divided in to four sections.
Starting from top left, the first section is responsible for displaying the status of a test program while an environmental test is progressing. The second section displays the measured temperature and humidity in the EUT chamber. Also, if a test is on progress it displays the set values of both temperature and humidity that are to be achieved by the current step. The third section displays the operating powers of the heaters and humidifiers in percentiles using progress bars. The fourth section monitors the on/off state of devices in the climate chamber.

5.2.1.2 Program Tab

This tab is where functionalities related to a test program are available.
On the left is a QListView, which is a Qt view class that provides a list or icon view from a model /18/. The Qt M/V architecture is used to present a list of test programs that were saved in the application. The model behind the list is QFileSystemModel. On the right is a QTableView which presents a spreadsheet-like table view of a model. This table view is used to display the steps and content of each step in a test program. The table view draws itself for each test program the user selects from the list of test programs on the left. The buttons with plus and minus icons are for adding and removing a step from a test program respectively. When the plus button is clicked a pop-up form is presented, for the user to fill in information about the step that needs to be added.

The row of buttons at the bottom adds interactive functions to this tab. The buttons and their functions are listed as follows:

- New button; to create new test programs.

**Figure 15.** The Programs tab.
- Load button; to preview test programs in a separate pop up window.
- Rename button; to rename a test program.
- Delete button; delete a test program.
- Start button; to start test program execution and
- Stop Button; to stop test program execution.

5.2.1.3 Aux Tab

The Aux tab lists and visualizes tests that have been executed in the past.

![Figure 16. The Aux tab.](image)

A list of tests that were executed is displayed in a QListWidgetItem at the left. The name of each test execution is made out of the name of the test program together with the date and time that the test was executed.
When a test program executes the change in measured temperature and humidity values are sampled and saved on to a file in. Up on selection of an item from the list of executed tests, a plot of the change in temperature and humidity vs time is presented on the right bottom frame of the tab.

### 5.2.1.4 Options Tab

Options tab provides a view for the user to keep settings of the application and a general information about the application.

![Options tab](image)

**Figure 17.** The Options tab.

In figure 17, On the right is placed a stack of pages that serve as a platform to contain other widgets. It contains three pages; system info, system params and control params pages. Navigation through these pages is provided by the vertical bar on the left. The navigation bar is made up of QToolButtons arranged vertical-
ly. A QToolButton is similar to regular buttons in GUIs but it adds flexibility of styling when using it with icons.

The system info page is used to display some information about the application. The system params page helps keep settings of some climate chamber’s parameters; the maximum allowed temperature, the minimum allowed temperature and the entry point, a start temperature of a test program. The control params page is UI where all saved PID parameters can be viewed, new ones can be added and the default is set.

![Climate Chamber - Options](image)

**Figure 18.** Control params page of the Options tab.

Figure 18 shows the control params page of the Options tab. It has a tabbed widget in the left and a vertical oriented frame on the right. The tabbed widget has two tabs:
• Temperature PID tab; which displays all saved temperature PIDs.
• Humidity PID tab; displays all saved humidity PIDs.

Each of the tabs use a QListView to present the list of PIDs.

The vertical frame on the right has three buttons:

• Make default button; which marks and persists the selected PID value on the list as a default PID value for the application’s use. There is one default for each of the PID types, temperature and humidity. The PID value that is marked default is shaded with a sky blue background color that matches the application theme.
• The Plus Sign button; adds new PID value to the list. It does so by presenting a pop-up window to collect the PID parameters form the user. Once the form is submitted the newly added PID appears in the list.
• Minus sign button; this does the opposite of the plus sign. It simply removes the selected PID from the list. If the selected PID is one of the defaults a pop-up message informing the user that a default PID is about to be removed and requesting confirmation is shown. The default PID is then removed based on the reply or the user to the confirmation message. Once a default PID is removed the user has to choose another default PID again.

5.2.1.5 Quick Start Tab

The Quick start tab presents a use case where the user can run tests much like the programs tab except, here the user is not able to choose a saved program rather create and run a new test program without having to save it to a file.
5.2.1.6 Dialogs

There are dialogs presented to the user throughout the use of the application. These dialogs are used to collect information from the user using forms, to communicate warnings or general information to the user and to ask confirmation on user actions to prevent undesirable effects due to accidental user actions. Some dialogs in the application are shown in figure 20 and 21.

Figure 19. Quick start tab.
Figure 20. Add step dialog.
Figure 21. Add program dialog.
6 IMPLEMENTATION

6.1 General description of Implementation

The software is implemented using OOP principles. Entities and group of tasks are represented with objects, which helps code responsible for single use case to be grouped together. The Qt M/V is used to create a fluid, intractable data visualization. The ability to build on top of the software, to extend its functionality in the future was considered important. Therefore, it was built keeping extensibility in mind. Documentation was written to help future developers understand the code.

The application is designed to keep and load some information that are critical to the operation of the application as settings. These setting are saved using QSettings class from Qt, which provides a cross platform and portable persistence of settings by saving them on the system registry for windows, the property list files for OS X and .ini text files for Unix. All classes are implemented to load relevant settings by themselves.

6.2 Implementation of Iterations

The implementation of the software has been broken down in to smaller deliverable iterations, as it is the principle in Agile methodology. With each iteration a functionality (use case) is added and a working product that is ready for test is delivered. Iterations that were implemented at the beginning were improved with the introduction of implementations from later iterations. The iterations and their implementations, at their completion, are described below.

6.2.1 Iteration: Serial Communication and Monitor

The communication class is the gateway in to the serial device for this application. It defines objects of QSerialPort and QSerialPortInfo for use in relation to serial communication, a QByteArrayList to hold data received from the control box, A QTimer object to use it when checking for connection status of the application, a
Controller object to access all use cases implemented by the class and a QSetting object to persist settings. It has functions to open, read, write and close the serial device.

```cpp
bool Communication::openPort(){
    bool open = false;
    serial->setPortName(settings.value("portName","ttyUSB1").toString());
    if(!serial->isOpen()){ // serial is a pointer to a QSerial object and settings is a QSettings object. The port name is loaded from the setting, which presumably the user has saved before. If the port name is not found in the setting it is given the name “ttyUSB1”. If the device was not already open, this function attempts to open it for both read and write access, but if it already was open, the application is marked as connected to the control box using the “chamberConnected” property of the communication class.
        open = serial->open(QIODevice::ReadWrite);
        serial->setBaudRate(QSerialPort::Baud9600);
        serial->setDataBits(QSerialPort::Data8);
        serial->setStopBits(QSerialPort::OneStop);
        serial->setParity(QSerialPort::NoParity);
        serial->flush();
    }else{
        open = serial->isOpen();
        setChamberConnected(open);
    }
    return open;
}

Code Snippet 1. Function to open the serial device.

void Communication::sendData(const QByteArray data){
    if(!serial->isOpen()){ // serial is a pointer to a QSerial object and settings is a QSettings object. The port name is loaded from the setting, which presumably the user has saved before. If the port name is not found in the setting it is given the name “ttyUSB1”. If the device was not already open, this function attempts to open it for both read and write access, but if it already was open, the application is marked as connected to the control box using the “chamberConnected” property of the communication class.
        openPort();
    }
    serial->write(data);
```
Code Snippet 2. Function to write data to on to the serial device.

In the above Code Snippet 2 the function receives a QByteArray, which is an array of bytes, as the name suggests, and writes the array on the serial device. However, first it checks to see if the serial device is open, if it was not the function will attempt to open it.

```cpp
QByteArray Communication::readData()
{
    *dataReceived->append(serial->readAll());
    QByteArray *end = new QByteArray(1, 0x0D);
    if(*dataReceived->data() == ControlCommands::ACK ){
        emit newDataArived(*dataReceived, ControlCommands::ACK);
        dataReceived->clear();
    } else if(dataReceived->endsWith(*end)){
        char nameOfCommand = dataReceived->at(2);
        if(nameOfCommand == ControlCommands::A){
            emit newDataArived(*dataReceived, ControlCommands::A);
        }else if(nameOfCommand == ControlCommands::B){
            emit newDataArived(*dataReceived, ControlCommands::B);
        }else if(nameOfCommand == ControlCommands::I){
            emit newDataArived(*dataReceived, ControlCommands::I);
        }else{
            emit unusualDataArived(*dataReceived);
        }
        dataReceived->clear();
    }
    return *dataReceived;
}
```

Code Snippet 3. Function to read data from the serial device.

The above Code Snippet 3 shows a function that reads data from the serial device as soon as the data has arrived. This function is connected to the `readyRead` signal of the QSerial object, serial. The `readyRead` signal is emitted whenever a new data is available for reading in the device. Therefore, whenever a new data arrives
on the serial device the *readData* function in the above Code Snippet is called. The function collects the arriving data in to a byte array till it reached the end of a message, it checks the ID2 of the message to find out the type of message (type of message is represented as an Enumeration in the ControlCommands class), emits the *newDataArrived* signal of the communication class using the arrived data and the type of data as its argument.

![Flowchart of communication.](image)

**Figure 22.** Flowchart of communication.
The emitted *newDataArrived* signal is in turn consumed by the *on_newDataArrived* slot of the same class shown below in Code Snippet 4.

```cpp
void Communication::on_newDataArrived(QByteArray newDataArrived,
                                         ControlCommands::CH_COMMAND chCommand)
{
  if (chCommand == ControlCommands::A)
  {
    on_ACommandArrived(newDataArrived);
  }
  else if (chCommand == ControlCommands::B)
  {
    if (!isChamberConnected() && getStateCounter() > 2)
    {
      setStateCounter(0);
    }
    setStateCounter(1);
    setChamberConnected(true);
    emit replyReady(chCommand);
  }
  else if (chCommand == ControlCommands::A)
  {
    emit replyReady(chCommand);
  }
  else if (chCommand == ControlCommands::I)
  {
    setChamberConnected(false);
    emit replyReady(chCommand);
    return;
  }
  else if (chCommand == ControlCommands::ACK) {
    
  }
}
```

**Code Snippet 4.** Function to handle received data.

The *on_newDataArrived* function manages the received data. It calls the *On_ACommand* function when the message informs the chamber status and emits *replyReady* signal for all messages that don’t.

```cpp
void Communication::on_ACommandArrived(QByteArray ACommand)
{
  QStringList list;
  double temp, humid;
  QString str(newDataArrived);
  list = str.split(" ");
  temp = list[0].mid(4, 9).toDouble();
  humid = list[2].left(4).toDouble();
```
void Chamber::setDryTemperature(double value){
    if(getDryTemperature() != value){
        dryTemperature = value;
        emit dryTemperatureChanged(QString("%1").arg(dryTemperature, 6, 'f', 2, '0'));
        emit dryTemperatureChanged(value);
    }
}

void Chamber::setHumidity(double value){
    if(getHumidity() != value){
        humidity = value;
        emit humidityChanged(QString("%1").arg(humidity, 4, 'f', 1, '0'));
        emit humidityChanged(value);
    }
}
**Code Snippet 6.** The dry temperature and humidity setters.

As shown in the Code Snippet 6 above, Setting the dry temperature and humidity of the Chamber Object with the current temperature and humidity of the chamber emits signals showing the change in value. These signals are connected to the `setText` slot of the GUI components that display the real time temperature and humidity.

```c++
// chamber connection to gui
connect(communication->pidController->chamberParams,
    SIGNAL(dryTemperatureChanged(QString)),
    ui->tempRealValueLabel, SLOT(setText(QString)));
connect(communication->pidController->chamberParams,
    SIGNAL(humidityChanged(QString)),
    ui->humidRealValueLabel, SLOT(setText(QString)));
```

**Code Snippet 7.** Chamber value changed signals connection to GUI slots.

This updates the GUI as new data arrives making the monitor complete.

### 6.2.2 Iteration: Store Test Programs and Display

Test programs are saved in to a text file in a human readable text format. A test program is first saved to a file that is given the same name as the test program and the no of cycles in the test program is written in to the file. Steps are then appended to the file one by one.
Figure 23. Sequence diagram of creating a new test program.

The above sequence diagram shows the process of creating a new Test Program.

```cpp
bool DataBackup::saveProgram(Program *p) {
    bool programSaved;
    QDir appDir = QDir::current();
    // check if programs directory doesn't exist and create it
    checkDir(PROGRAMS_DIR_NAME);
    appDir.cd(PROGRAMS_DIR_NAME);
    QString programFilePath = appDir.path() + QDir::separator() + p->getProgramName() + PRGM_FILE_EXT;
    Qfile programFile(programFilePath);
    programSaved = programFile.open(QIODevice::WriteOnly);
    QTextStream out(&programFile);
    out << "Cycle: " + QString::number(p->getCycle()) << endl;
    programFile.close();
    return programSaved;
}
```

**Code Snippet 8.** Function to save a Test Program.

In the above Code Snippet 8, name of the Test Program is used as the name of the file and Cycle of the Test Program is written on the first line of the file. Steps are then appended to the file, one by one, as shown below in Code Snippet 9.

```cpp
bool DataBackup::writeStepToFile(Step *step, Program *prgm) {
    bool isAdded = false;
    QString prgmName = prgm->getProgramName();
    QString path = fileLives(PRGM, prgmName);

    if(path.isEmpty()){
        isAdded = false;
        return isAdded;
    }
    QFile prgmFile(path);
    if(prgmFile.open(QIODevice::Append)){
        QTextStream ts(&prgmFile);
        ts << QString("%1") << step->getTemperature() << 6, 'f', 2, '0') << ":" << QString("%1") << step->getHumidity() << 4, 'f', 1, '0') << ":" << QString("%1") << step->getHours() << ":
        << QString("%1") << step->getMinutes() << ":
        << QString("%1") << step->getWaiting() << ":
        << QString("%1") << step->getHR() << ";
```

After the above Code Snippet 9 executed and inserted some steps, the program file reads as the following Figure 24 shows.

Figure 24. Program File content.

The Figure 24 above shows the content of a test program file, the same test program that is shown in section 5.2.1.2 in Figure 15. To display this file as in figure 15, the Qt M/V architecture was used. This was achieved by implementing a model class, StepsModel, that inherits from the QAbstractTableModel to provide information that fills the steps table.

class StepsModel : public QAbstractTableModel
{
  private:
    Q_OBJECT
    Program *pgmToShow;

enum HEADER_TYPE {NUM = 0, TEMPR = 1, HUM = 2,  
    HRS = 3, MINS = 4, WAIT =5,  
    HR = 6, ONE = 7, TWO = 8, THREE = 9};

Code Snippet 10. Reserving space for program in the model.

The above Code Snippet 10 shows the definition of a pointer to the test program whose steps will be used as the underlying data for a table model and a definition of integers representing column headers.

QVariant StepsModel::headerData(int section, Qt::Orientation orientation, int role) const  
{  
    if(orientation == Qt::Vertical && role == Qt::DisplayRole)  
    {  
        return section;  
    }  
    if(orientation == Qt::Horizontal && role == Qt::DisplayRole)  
    {  
        switch (section) {  
            case NUM:  
                return "Num.";
            break;
            case TEMPR:  
                return "Temp.°C";
            break;
            case HUM:  
                return "HUM %RH";
            break;
        }
    }

Code Snippet 11. Setting column and row headers.

The Code Snippet 11 above shows how the header data is provided to the GUI. The UI element, in this case the QTableView, requests the model for what data to put on the column and row headers just before it draws them on the screen. It gives the model what section, what orientation and for what role the data is needed. This model responds to only display role, and gives back the section for vertical orientations, those are rows, and the name of the column for horizontal orien-
tations. This function finally gives the rows a numbered header and the appropriate names for the columns. In the same way the table data is filled from the steps of the program as shown in the Code Snippet 12 bellow.

```cpp
QVariant StepsModel::data(const QModelIndex &index, int role) const
{
    int row = index.row();
    int stepSize = pgmToSort->getSteps().size();
    bool containsNot = !pgmToSort->getSteps().contains(row);
    if(!index.isValid())
    {
        return QVariant();
    }
    if(role != Qt::DisplayRole)
    {
        return QVariant();
    }
    if(row < 0 || row > pgmToSort->getSteps().size()
        || !pgmToSort->getSteps().contains(row))
    {
        return QVariant();
    }
    Step *s = pgmToSort->getSteps().value(index.row());
    switch (index.column()) {
    case NUM:
        return s->getStepNumber();
    break;
    case TEMPR:
        return s->getTemperature();
    break;
    case HUM:
        return s->getHumidity();
    break;
    }
}
```

**Code Snippet 12.** Data to fill in the table view of steps.

The function in the above Code Snippet 12 fills the table with the underlying data, the steps of test program. The GUI requests the data by passing a QModelIndex and a role. When the role is Display role and the model index is valid in the model, the data in the collection of steps is returned to the GUI.
6.2.3  Iteration: Expose Control Command Messages

The ControlCommands class is represents messages in the communication (see section 4.3.2). Properties representing peripheral devices are defined in this class and they are assembled in to a block of command in respect to the communication protocol.

private:
    bool h1;
    bool h2;
    bool t1;
    bool t2;

/**
 * @brief htBlock this array represents the command block that holds information
 * about H1, H2, T1, and T2 devices activation.
 */
    unsigned char htBlock;

**Code Snippet 13.** Definition of H1, H2, T1 and T2 along with the HT block of size one byte.

The above code shows how the peripheral devices humidifier 1, humidifier 2, heater 1, heater 2 and the message block that will contain the representation of these devices are defined in the class (See Figure 10).

Each of these properties have a setter and getter function implemented as follows.

```cpp
bool ControlCommands::getH1(){
    return h1;
}
```
void ControlCommands::setH1(bool value){
    if(h1 != value){
        h1 = value;
        emit chPartChanged(value, ControlCommands::H1);
    }
}

**Code Snippet 14.** Getter and setter of H1.

Every time a class member that represents a peripheral device in the message, like h1, is changed a `chPartChanged` signal is emitted as shown in the Code Snippet 14 above. This signal is absorbed by a function and the respective block of message is changed accordingly. The code below shows how the `htBlock` class member is changed when the `h1` member is changed.

```cpp
void ControlCommands::on_chPartChanged(bool value, ControlCommands::CH_PART part){
    switch (part) {
    case H1:
    {
        std::bitset<8> bits(htBlock);
        if(value){
            bits.set(2);
        }else{
            bits.reset(2);
        }
        htBlock = static_cast<char>(bits.to_ulong());
    }
    break;
    }
}
```

**Code Snippet 15.** Changing the bit corresponding to the peripheral device.

In the above code, the number two bit of the `htBlock` in the LSB 0 bit order is set to same value as `h1`. Therefore, if any other code wants to change the on/off state
of heater one, it calls on the setter of \texttt{h1} member of ControlCommands class and in turn part of the message that holds this information is changed.

```cpp
void ControlCommands::setTemperaturePowerRate1(std::bitset<8>*value)
{
    std::bitset<8> bits(temperaturePowerRate1);
    bits[4] = (*value)[0];
    bits[5] = (*value)[1];
    bits[6] = (*value)[2];
    bits[7] = (*value)[3];
}
```

\textbf{Code Snippet 16}. Assigning power of heater one.

The above code shows how the power rate is incorporated in the message blocks.

When it is time to send message to the control box, the full content of the message is constructed by appending the message blocks one after the other in their correct order. This constructed message is given to a communication object where it is sent via a serial port.

\textbf{6.2.4 Iteration: Store PID Parameters and Display}

Using Serialization of PID objects is how PID parameters are saved to file. To make the PID class serializable, the two stream operators of the QDataStream class were overloaded.

```cpp
QDataStream &operator <<(QDataStream &out, const PID &pid)
{
    out << pid.getKp() << pid.getKi() << pid.getKd() << pid.getChoosen();
    return out;
}
```

```cpp
QDataStream &operator>>(QDataStream &in, PID &pid) {
    double kp;
    double ki;
    double kd;
    qint8 choosen;
    in >> kp;
    in >> ki;
    in >> kd;
    in >> choosen;
    pid.setKp(kp);
    pid.setKi(ki);
    pid.setKd(kd);
    pid.setChoosen(choosen);
    return in;
}
```


These two operators serialize and de-serialize a PID object. Kp, Ki and Kd are properties of the PID class representing the proportional, integral and derivative constants. The getter and setter functions called on the PID object on the above code snippets are getters and setters of these properties. The DataBackup class invokes these operators on a QDataStream object that is pointing to a file and the PID object to be serialized or de-serialized.

6.2.5 Iteration: Settings

The application persists some information as its settings. Each class loads its own settings and settings are persisted by a function call in the UI class where the user chooses the settings. Therefore, a QSetting is used in so many different places in the code. To make this use easy the application name, organization name and organization domain which Qt uses to find the setting for a specific application are set up at the entry point of this application as shown below.
QCoreApplication::setOrganizationName("technobothnia");
QCoreApplication::setOrganizationDomain("simachew.com");
QCoreApplication::setApplicationName("climate_chamber_controller");

**Code Snippet 19.** Setting application name, domain and organization.

The most important of these settings for the application to function is the default name of the serial device in the system.

```cpp
void MainWindow::on_serialPortNameUpdateButton_clicked()
{
    QSettings setting;
    setting.setValue("portName", ui->serialPortComboBox->currentData().toString());
}
```

**Code Snippet 20.** Persisting Serial Port name to settings.

In the above code, the default serial port name is saved in to settings after the user chooses to update the value.

This default value is loaded in to the application by the Communication class, by the `openPort` function. (see section 6.2.1)

### 6.2.6 Iteration: Running Test Programs

In this iteration, running test programs and progression of their steps is implemented. Besides communicating with the control box, tasks the application needs to take care of in order to run test programs are:

- Initiate the start of test
- Manage the step progression and notify the user
- Control peripheral devices
- Keep record of the real values on a file
- And in the end stop the test

Once the user has selected a test program and pressed the start button, the test is initiated by preparing tools needed throughout the test. The Controller class is responsible to manage the progress of a test. This class uses two PID objects and occasionally uses a DataBackup object (see section 4.3.7). Two timers, one to time the length of the currently running step and one to time when test progress record is kept, are created and initiated. A file to keep test progress record is also created.

```cpp
plotTimer->setInterval(settings.value(PlotInterval, "2").toInt() * 1000);

void Controller::setUpStart()
{
    stepTimer->setSingleShot(true);
    int hrs = currentStep->getHours();
    int min = currentStep->getMinutes();
    int timeOut = hrs * 3600000;
    timeOut += (min * 60000);
    stepTimer->start(timeOut);
    plotTimer->start();
}
```

**Code Snippet 21.** Initiating test start.

As seen in the above code snippet, the plot timer interval is loaded from the application setting. If it is not found in the setting it will be initiated with an interval of two minutes. The step timer is set to run out (fire) only one and its interval is set to be as long as the length of the current step.

When the plot timer runs out, the real temperature and humidity values and the time elapsed in the current step are appended to a file as shown below.
void Controller::on_plotTimerOut()
{
    double temp = temperaturePID->getMeasuredValue();
    double humid = humidityPID->getMeasuredValue();
    int totalMinutes = currentStep->getHours() * 60 + currentStep->getMinutes();
    int remainingMinutes = (stepTimer->remainingTime() / 1000) / 60;
    int elapsedMinutes = totalMinutes - remainingMinutes;
    DataBackup db;
    db.appendPlot(testPgm->getProgramName(), temp, humid, elapsedMinutes);
}

Code Snippet 22. Function called when the plotTimer interval runs out.

The above code shows a function that is connected to the timeOut signal of the plotTimer, meaning it is invoked in each interval of the plot timer. In this function the real temperature and humidity and the time elapsed since the start of the current step are collected and the appendPlot function is called to append this information to a file.

When the stepTimers timeOut signal is emitted the connected changeStep slot is invoked. If the target set value has not been reached and the wait status is on moving to the next step will be canceled until the target is reached otherwise the test is progressed to the next step. When the test step has changed it is notified by emitting a signal. Interested parties like the GUI consume this signal and update themselves.

The PID class has a function where a control output is calculated based on the inputs, the inputs being the real temperature and humidity values. Whenever a control is requested the following function is invoked.

void Controller::controlTest()
{
    temperaturePID->control();
    humidityPID->control();
}
runDeviceControll();
emit controlready(ControlCommands::I);
}

**Code Snippet 23.** Controlling the test.

This function in turn makes a call to two other functions. The *control* function of both the temperature and humidity PID objects; where a simple PID control system logic is implemented in and the *runDeviceControl* function; where the devices are turned on or off based on the control output of the two PID objects. These two functions are shown below.

```cpp
int PID::control()
{
    error = setValue - measuredValue;
    if(error > 190 || error < -190){
        output = 0;
        return 0;
    }

    integral = integral + (error * dt);
    derivative = (error - previousError) / dt;
    proportional = kp * error;

    output = proportional + (ki * integral) + (kd * derivative);
    previousError = error;
    if(output < MIN_OUT){
        output = MIN_OUT;
    } else if(output > MAX_OUT){
        output = MAX_OUT;
    }
    return output;
}

**Code Snippet 24.** A function that calculates the control output.

```
controlCommands->setTemperaturePower(temperaturePID->getOutput());
controlCommands->setHumidityPower(humidityPID->getOutput());

if(temperaturePID->getOutput() > 0){
    controlCommands->switchHeaters(on);
    controlCommands->switchCooler(off);
}

Code Snippet 25. Part of a function to control peripheral device operation state and power.

The above code snippets show an example of where a PID control system is implemented and where the power rate and peripheral device states are changed based on the control outputs of both temperature and humidity PID objects.

When the test reached its end or the user decides to stop it manually, all of the peripheral devices will be set to off, operating powers of heaters and humidifiers are set to zero and the application is set to idle and no control will be requested until another test is requested.

6.2.7 Iteration: Plotting Tests

Plotting of a test to a graph is possible by reading the file that a test progress record was kept on. A third party library, QCustomePlot is used to turn the record into visual graph.

6.2.8 Iteration: Quick Test

Quick test is a function that allows running tests without having to save the test program into a file (see section 5.2.1.5). This iteration is implemented so that it can create an experience where the user has to do as little as possible to run test programs. It is not a must to name the test program nor is it to give it a cycle. The
test programs name will be set to “untitled” and the cycle will be one. The user can add steps and start the test and the same process as in section 6.2.6 is carried out to run the test.
7 TESTING

Iterations resulted in delivering major functionality to the overall development of the application. Each of these iterations are made of smaller increments of deliverables. Tests were carried out at the end of each iteration and in each smaller deliverable in an iteration.

Throughout this project the climate chamber was in use by the customer. This limits the availability of the climate chamber for tests. To be able to run tests at any point during the development, another desktop console application to simulate the control box was built for test purposes.

The control box simulator responds to messages from the climate chamber monitor and controller application in a similar manner as the control box. Messages were of the same format and identical to the real control box with the exception of the real temperature and humidity values embedded in the messages being fabricated. Communication tests were executed on the simulator before testing with the actual climate chamber.

The smaller deliverables were tested with the control box simulator application and by the end of each iteration an integration test was carried out to see how well the newly introduced functionality works with the rest of the application and a test with the actual climate chamber was carried out to confirm the results with the real machine.


8 CONCLUSION

At the beginning, Understanding the old system was essential for this software to work seamlessly with the climate chamber. This study revealed restrictions imposed on the project by the system. These restrictions and the requirements of the software influenced the choice of technologies to use. A research was carried out to find powerful, well documented and community supported technologies. Based on the result of this research the technologies used in this project were chosen, irrespective of the skill of the developer. The use of these technologies were studied before the implementation and continually done so throughout the implementation. In the end the project was successful as it delivered a user friendly GUI, Monitor to the chamber and facilitates control by providing access to manipulate the control commands. A simple PID control logic, device control logic and quick start functionality were also implemented. A documentation describing each class and functions in the source code is provided to help future developers integrate new features and improve already existing ones.

8.1 Future Work

In addition to improvements that could be implemented to the already existing code of this software, some functionalities can be added in the future.

The climate chamber is known to notify the user on some events. Some of these events include: water tank for the humidifier being empty and the chamber temperature passing both high and low limit. A feature to handle this events by taking safety measures could be implemented. As indicated before a simple PID control system logic and logic to turn the peripheral devices on and off was implemented. A better controller logic and tuning of the controller could be implemented to have an accurate control of environmental tests. A thorough investigation of what factors affect the state of peripheral devices could be carried out and implemented.
REFERENCES

/1/ Environmental test chambers. Accessed 03/02/2016
http://www.globalspec.com/learnmore/labware_test_measurement/product_material_testing/environmental_test_chambers

/2/ Agile software development model. Accessed 03/02/2016.
https://meetings.vtools.ieee.org/m/11457


/4/ The ISO C++ committee Accessed 04/02/2016 http://www.open-std.org/jtc1/sc22/wg21/

http://www.cplusplus.com/info/description/


74

/13/ Messages in the communication Remote manual, King Son INS.


