Cryostat Control

Real time control for a cryogenic refrigerator

B. Bekker M. Goudriaan
R.D. Meeuwissen L.H. Sikkes

Technische Universiteit Delft

TU Delft Delft University of Technology

Cryostat Control

Real time control for a cryogenic refrigerator

by

B. Bekker,
M. Goudriaan,
R.D. Meeuwissen,
L.H. Sikkes

in partial fulfillment of the requirements for the degree of

Bachelor of Science
in Computer Science

at the Delft University of Technology,
July 6, 2017

Supervisor:	Dr. R. Krebbers,	TU Delft
Project Owner:	Dr. J. Bueno,	SRON
	S. Hähnle,	SRON

TU Delft
Delft
University of
Technology

Abstract

In order to measure the spectrum of radio emissions from galaxies and other deep space objects, a new superconducting spectrometer, working at very cold temperatures close to the absolute zero, is developed. An advanced cooling system called a cryostat is used to cool down the spectrometer. The cool down of the cryostat involves the control of multiple sensors and actuators connected to the cryostat to achieve a final temperature below 250 millikelvin. A software program is used for this purpose. As extra hardware components have been added to the cryostat, the existing program does no longer fulfil the requirements. For this reason a new software program, which can monitor temperatures of all components and start control processes, is developed. The developed program consists of a client server structure. The server handles the logic of the cryostat using several controllers. It can send data to a native client, which is the graphical user interface, or a REST API. The native client displays sensor readouts received from the server and allows full control of server, which means it can start the cool down process as well as manual control processes. The REST API allows the user to have full control over the server using a Python script to achieve measurements which cannot be done from the native client. The increased automation, improved control and ability to integrate with external Python scripts allow the user to focus on the essential parts of an experiment making the developed program an improvement over the previous program.

Preface

This report summarizes our work on the bachelor end project for the Bachelor Computer Science at the Delft University of Technology. During this project we have developed a control program for a cryostat that consists of a server which controls the cryostat and a client which visualizes the data and allows control over the server. The server regulates the cryostat with the use of a finite state machine. The project was carried out by four computer science students who worked neatly together to achieve the final product. We would like to thank Robbert Krebbers for his guidance and support throughout the project. The meetings provided important feedback with which we could improve our project. We would also like to thank the Astronomical Instrumentation group and SRON for providing this project and for the office we could use during our project. During the project we had to dive in the world of space and physics before we could start with making software. It was an exciting project which let us not only test our computer science qualities, but also our knowledge about physics.

B. Bekker,
M. Goudriaan,
R.D. Meeuwissen,
L.H. Sikkes,
Delft, June 2017

Contents

1 Introduction

The Astronomical Instrumentation group of the TU Delft in collaboration with SRON, the Netherlands Institute for Space Research, is working on the design of a new Multi-Object Spectrometer named MOSAIC that can measure the spectrum of radio emissions from galaxies and other objects in deep space.

The purpose of this bachelor end project is to create a program that will assist the user during experiments by controlling an advanced cooling system called the cryostat. The program will take care of multiple processes automatically and gives insights of the process. This allows the user to focus on the parts of the experiments that matter.

The cryostat consists of three active components: a pulse tube refrigerator, a compressor and a Helium-7 cooler. Those active components need to be monitored and operated remotely. In the current system, the three components of the system are controlled separately with the use of different software programs or manual controls. The programs are able to control the cryostat, though they do not have all functionality the Astronomical Instrumentation group desires. The developed program integrates the previous programs into one single application. This application is able to communicate to all three active components, offers increased control and monitors the functionality of the cryostat. These improvements make it easier and faster to set up experiments with the cryostat at temperatures below 250 millikelvin and incorporate the cryostat temperatures into the results of the experiment.

In this report the process of developing the new control program is elaborated. First the project management will be elaborated in Chapter 2. Next the control methods used for controlling the cryostat are discussed in Chapter 3. In Chapter 4 the program and the software decisions and implementation will be discussed. Next, in Chapter 5 the quality of the program will be described. The ethics regarding this project will be discussed in Chapter 6. Then Chapter 7 will validate and evaluate the project. Finally a conclusion is given in Chapter 8.

The remainder of this chapter will give more background information to the project, starting with an introduction of the MOSAIC project in Section 1.1. To implement the program in this project it is required to understand the general working of the cryostat, this will be explained in Section 1.2. Finally, Section 1.3 elaborates on the problem by describing it in detail.

1.1. MOSAIC Project

MOSAIC is a spectrometer consisting of an array of 25 pixels that gives it the ability to measure the radio spectrum for 25 deep space objects at the same time. Additionally the beam of each pixel can be steered electrically to lock onto an individual astronomical object. MOSAIC will be installed on the 10 meter Japanese ASTE observatory, a world-class astronomical facility based at the Atacama desert in Chile at an altitude of 5000m.

MOSAIC is implemented on a single chip based on novel superconducting circuits, which need to be cooled down to cryogenic temperatures to function. Development and testing of the new instrument takes place at the TU Delft. Part of the test setup is a cryostat able to reach a temperature below 250 millikelvin (mK).

1.2. The Cryostat

The cryostat used for this project is able to reach a minimum temperature of 234mK and can hold that temperature for a day. Cooling down to the minimum temperature takes 12 hours. Figure 1.1 shows a schematic overview of the components, connections and insulation layers. Table 1.1 contains a short definition of the components referenced throughout this paper.

Cryostat	Complete cooling system including radiation shields, pulse tube refrigerator and He-7 cooler
He-7 cooler	Helium cooler consisting of a He-3 and He-4 stage that is able to reach temperatures <250mK
Pulse Tube Refrigerator (PTR)	Compression/Expansion based cooler that is able to cool from 250K down to 3K
Radiation shields and vacuum can	Device that holds the PTR and He7 cooler in vacuum and shields the different stages of the cryostat from thermal radiation. It insulates the experiment from outside heat using a vacuum can, a 50K radiation shield and a 3K radiation shield.
Compressor	Device that drives the pulse tube cooler by compressing and expanding helium gas.
Vacuum pump	External pump that creates a vacuum inside the vacuum can.
Lakeshore temperature controller	Controller connected to several temperature sensors controlling the cool down of the cryostat.
Agilent 34970a DAQ unit (Agilent)	Input and output device monitoring and controlling the He-7 cooler.

Table 1.1: Definition of system components which are mentioned in this paper

To achieve a temperature below 250mK, the temperature is lowered from ambient temperature (~250K) in multiple steps. The radiation shields insulate the experiment from the ambient temperature by holding a vacuum, created by a separate vacuum pump, to prevent convection, and has multiple barriers to shield the experiment from heat radiation. The first layer of the cryostat is cooled by the pulse tube refrigerator to 50K, the inner layer is cooled to 3K by a second stage of the pulse tube. The pulse tube refrigerator is driven by a compressor that compresses and expands the gas in the pulse tube.

In the core of the cryostat a He-7 cooler is mounted, consisting of a He-3 stage and a He-4 stage. The cooler cools its cold head to less than 250mK by evaporative cooling of ^{3}He. It also has a ^{4}He ”buffer” cooler that pre-cools to ~800mK, and is able to sustain a higher cooling load. The experiment is mounted on the He-3 cold head.

The evaporation of He isotopes cools the head by adsorbing latent heat during its liquid to gas transition. The gas particles are ”pumped” from the reservoir into heat pumps by lowering the pressure in the pump using adsorption. This reduces the vapor pressure in the head and promotes the evaporation process. The pumping works because gas particles form a film over the surface of an adsorbent in the pump, in this case activated charcoal, creating a lower pressure in the pump. When all helium is pumped from the reservoir the cooler is no longer able to keep the head cold and needs to be recycled. In the recycle process, the charcoal is heated to over 20 Kelvin (K) to make it release the helium gas. During the initial cool down, the charcoal must be kept heated above 20K, to prevent gas being directly adsorbed by the charcoal [1]. Pumping is activated using gas-gap heat switches, which activate or prohibit heat conductance away from the charcoal. The heat switches are activated using resistance heaters.

The ^{4}He has a higher liquefaction point and is used to cool the ^{3}He below its liquefaction point of 3.32K[1]. Because of the greater cooling capacity and greater availability of ^{4}He compared to ^{3}He, it is used to cool the experiment as low as possible before activating the He-3 pump, and to buffer all (conductive) connections to the experiment.

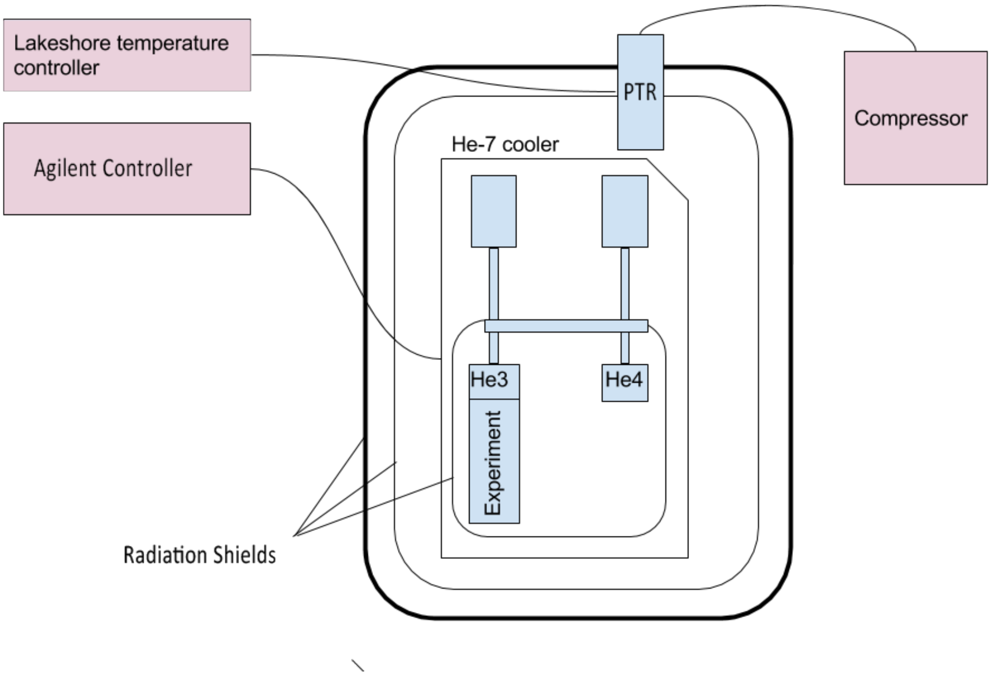


Figure 1.1: Schematic overview of the cryostat and control devices. The components are described in Table 1.1

1.3. Problem Description

In order to perform experiments, the Astronomical Instrumentation group needs to place their spectrometer in a cold space that is close to 0K. To create this cold space, they use the cryostat system, which consist of several components that work together to reach a temperature below 250mK. The working of this system was described in Section 1.2. Currently, the control over these devices is handled separately for each device, which reduces the efficiency at which the experiments can be set up. To improve their experiments, the Astronomical Instrumentation group would like to see more features as opposed to the current setup. Furthermore, there is a lack of overview in the programs and the option to control the devices is very limited. This section will describe the current setup and the problems that lay within.

1.3.1. Current Setup

The current situation requires a combination of actions on programs on a computer and manual control on the devices themselves. To get the cryostat to minimal temperature several components work together, the ones that need to be controlled will be described below.

First, a vacuum pump is used to create a partial vacuum inside the system. This pump needs to be attached at the start of a cool down procedure and takes about half an hour to reach its required pressure level. After reaching the desired vacuum level it can be detached and is no longer used during the cool down process.

Next the compressor, which is stored in a sound dampening box, must be turned on by pressing a physical on off switch. By turning on the compressor, the pulse tube refrigerator starts cooling down the radiation shields and pre-cools the He-7 cooler.

The He-7 cooler is controlled using a computer program called *He7Logger*. There are three control cycles that can be executed: cool down, recycle and warm up. These are used regularly to get the system ready for experiments. Besides these control cycles the program has the function to set voltages for heating the helium pumps. The program allows its user to set maximum or minimum voltage on these pumps. This program also shows the temperatures in a simple chart.

Finally, the Lakeshore temperature controller has a separate program which is used to monitor the two temperatures of the radiation shields which are cooled by the pulse tube refrigerator. This program also shows the temperatures in a simple chart.

1.3.2. Problems Encountered

After describing the current setup a list of the problems that exist within the programs will now be given:

- The compressor is not connected to a computer and currently needs to be physically controlled while it is stored in a hard to reach place. Not only does this make it hard to operate, but the values that are measured by this device are simply not used in the current setup, only the on/off functionality.
- The four devices that control the components of the cryostat need to be operated separately which greatly increases the effort needed to cool down the system. Furthermore this also leads to a lack of overview. Since the control of the system is separated over several programs, so is the overview of the entire system. When the user wants to look up two values of different devices this requires more effort than desired.
- The current programs have charts to show temperatures over time, but these fail to fulfill the requirements of Astronomical Instrumentation group. The charts only show a limited amount of data and do not allow any customization like zooming.
- The control program of the He-7 cooler is limited in use. It is able to cool down the system to below 250 mK and to monitor warming it back up. The device itself allows the use of more custom settings, but with the current program these are not accessible.
- The logs of the devices are handled by separate programs. This results in multiple files of the same time period. To actually get an overview of all devices these files would have to be merged into one.

1.3.3. The New Setup

In order to solve the main problems as described in the previous section, a new setup was desired. This new setup led to a full set of requirements, which can be found in Section 2.1. This section will give a global overview of what this situation will look like:

- To include the compressor in the new controlling program it will be connected to the computer, and be operated from there. This allows the user to get an overview of its values, as well as the option to operate it. The measurements taken by this device will be shown in the program and can be read out whenever needed.
- There will be a single program that controls three devices: the compressor, the Lakeshore and the Agilent. The most used states of the system, to reach below 250mK or to warm up to room temperature, can be reached with a single press of a button. The program will handle all logic regarding the control and collaboration of the three devices. There will be a manual control option, which allows custom control. Besides controlling the devices, the new program will also give an overview of the whole system. The vacuum pump that needs to be attached and detached at the start of each cool down procedure cannot be connected to a computer. Since it needs attaching and detaching anyway, it will be left out of the scope of the program.
- The new program will provide a new version of the charts. First off, these charts will show more data to give a better overview. Furthermore the chart will allow customization in the form of zooming, panning and filtering of data.
- Since there is a single program controlling all the devices, the logging will be conducted into one file. As a result of the file is instantly usable and contains all wanted values.

- Finally, the Astronomical Instrumentation group wants to be able to give commands to and retrieve data from the system by using a Python script. This new functionality will be used during experiments that require precise execution.

2 Project Management

When working on a project it should be clear on what to make and how to make it. To get a clear understanding of the program needed to implement the requirements of the program are stated. (Section 2.1). Followed by a discussion which concept to use, the concept describes the high level structure of the program. It is discussed whether to use a client server architecture concept or another concept. (Section 2.2). Finally the global organization of the project is explained (Section 2.3).

2.1. Requirements

Before starting the implementation of the program, a set of requirements had to be made. The requirements are an agreement between Astronomical Instrumentation group and the project team that contains all functionality of the program, to which both can fall back in case of a disagreement. At each phase in the project the requirements can be analyzed to measure how far along the program is and be changed whenever seen fit. The latest set of requirements can be found in the list below.

To implement the requirements the MoSCoW method was used. This method is a prioritization technique that divides all requirements into one of four categories, in descending order: Must have, Should have, Could have and Won't have. The must haves are the bare minimum that the program must have. Without these the program will not perform and failing to implement these will mean the project has failed. Should haves are requirements that have a high priority, but are not required for the program to function correctly. The could have category is for requirements that will be implemented when all must and should haves are implemented within reasonable time. Finally, won't haves are the problems with the lowest priority and these will most likely be left out of the project. These can be interesting for following projects though.

To further clarify the requirements a division between the components of the program was made. These components consist of the GUI, which visualizes the system and gives the user the ability to control the system; the charts, which are part of the GUI; the automatic and manual modes, which are used to control the system; the logging; the Python scripts; and the non-functional requirements.

Must Have

The GUI will contain:

- A button to start one of the automatic control cycles:
  - Start the cool down process.
  - Start the recycling process for the He-7 cooler.
  - Start the warm up process.
- The option to set certain attributes manually:
  - Turn the heater in the He-7 cooler on or off.
  - Turn the compressor on or off.

  - Change the voltages of the heater in the He-7 cooler.

- The option to start a specific logging file, which has a custom interval and has a choice which sensors will be logged.

- The option to configure settings that are used during the automatic control cycles.

- A notification panel that shows important information, errors and warnings.

There will be two charts showing the temperature over time:

- There will be a chart always shown in the bottom of the screen, this chart shows the temperatures of the 3K and 50K plates of the radiation shields and the He-4 and He-3 Head in the He-7 cooler.

- There will be a chart tab that contains a more detailed chart with customization options:

  - The option to zoom in by scrolling with the mouse wheel, as well as zooming in on a specific range specified by the user.
  - The option to show or hide lines in the graph, to focus on what is currently needed.
  - This graph will display the temperatures of the 3K and 50K plates of the radiation shields, as well as the He-3 Head, He-3 Switch, He-3 Pump, He-4 Head, He-4 Switch, He-4 Pump of the He-7 cooler.

The automatic control mode can:

- Start the cooling down process:

  - Start the compressor.
  - Start retrieving the sensor values every second.
  - Start the charts.
  - Start the general data logging.
  - Start the He-7 cooler, after reaching a temperature of 50K.

- Start the recycling process for the He-7 cooler.

- Start the warm up process:

  - Stop the compressor.
  - Stop the He-7 cooler.
  - Enhance the heating process of the He-7 cooler by warming up the heater in the 3K plate.

The manual control mode can:

- Turn the heaters on or off.

- Turn the compressor on or off.

- Change the voltages of the heaters.

The following information will be logged:

- The temperatures of the compressor, as well as its state(s)

- The temperatures of the radiation shields, as well as its state

- The temperatures and voltages of the He-7 cooler, as well as its state

There will be the possibility to use Python scripts to:

- Give commands to the cryostat.

- Retrieve sensor values.

Non-functional:

- The program must be reliable. Since the cryostat can be active up to a month, the program must be stable during this session and must not crash or give any complications meanwhile.
- The core of the program must run on a single computer that uses Windows.
- This computer is connected to the cryostat which means that it must communicate with three devices.
- Since the cryostat is only a tool to perform experiments, the operating program must be simple. The program must therefore be able to perform actions with a few clicks.

Should Have

The GUI will contain:

- Tabs which zoom in on the information about a certain component of the cryostat:
  - An overview of the cryostat as described in the Must have section.
  - A detailed overview of the He-7 cooler.
  - A detailed overview of the compressor.
  - A detailed overview of the radiation shields.
  - An overview of the logging settings.
  - An overview of the settings used in the automatic control cycles.

The automatic control mode can:

- Have a delay built in for the cooling down process of the whole cryostat.
- Have a delay built in for the recycling process of the He-7 cooler.

Could Have

The GUI will contain:

- The option to drag the tabs of the GUI to create a new window.
  - There could also be the option to place these windows back in the main window.
- Notifications when to start the cooler.
- An overview of the pressure per area in the compressor.
- Active heater voltage control to reach a set temperature point.
- Errors messages to indicate when an error occurs either in the program or with the cryostat.

Won’t Have

- A web interface for controlling and monitoring the cryostat.
- The controlling and monitoring of the vacuum pump.
- The controlling and monitoring of the heater in the radiation shields.

2.2. Design Concepts

In the research phase four different design concepts were made. The first three concepts make use of a client server architecture as network architecture with different interpretations of the client. The fourth concept is a single application on one computer without a network architecture. The choice of a client server architecture is made because it has the option to connect multiple clients to the control server. This allows the user to monitor the cryostat from multiple devices at the same time. A client server architecture also allows to access the Python interface together with the native client. A variation of the client server architecture is the web based client server architecture, where the client is a web page instead of a native application, which is used in concept 2.

The four concepts were:

- **Concept 1:** GUI client and an API for external Python scripts
- **Concept 2:** Web based GUI Client and an API for external Python scripts
- **Concept 3:** Integrated GUI attached to the server and an API for external Python scripts
- **Concept 4:** GUI with an extension for Python

The user liked the idea of the client server concepts, which can be extended to multiple platforms, therefore concept 3 and 4 were discarded as they do not have to ability to extend the program to other system. The user preferred a native application over a web page as they were already familiar with a native application and therefore requested a native application. They liked the idea of a web page as it allows one client to support multiple operating systems and, if developed correctly, is also mobile friendly. Unfortunately due to the time limitation only the first concept was implemented. With more time a website or mobile application could be implemented.

With the choice of concept 1 a server runs on a computer which is connected to the different components of the cryostat. A client can access the data from the components in real time by means of a network layer and control those components in a GUI or with a Python script. The advantage of this concept is that server has an API which could be used by multiple applications which can be written in various programming languages as long as they can connect to the server. This means that besides the native GUI client simple Python scripts can be used to easily read out specific data. Another advantage is that the cryostat can be controlled by multiple computers at the same time which can be at various locations. A disadvantage with the network layer is that the cryostat becomes vulnerable from outside threats via the Internet and the network layer has to be implemented. More details over the client server implementation can be read in Section 4.2.3.

2.3. Organization

In many of the courses given in the bachelor of computer science the importance of the way you organize the development process has been emphasized. For this project we have to be flexible in terms of changes to the requirements and testing software components during the project. For this reason scrum would be a good software development method for this project [2]. How scrum is used in this project will be discussed next.

The project team consisted of four student developers, a project owner (consisting of two members which will be the users of the software as well) and one supervisor. In the first phase of scrum (the planning phase) there were multiple meetings with the project owner to determine what software had to be developed. With the requirements (see the MoSCoW model in Section 2.1) acquired from these meetings a roadmap was created as an indication which work had to be done every week throughout the project. Besides the roadmap a planning for each week was created. In this planning it was agreed that a sprint plan would be made at the start of each week in order to have a clear set of tasks which had to be completed that week and who would be responsible for each task. These tasks were then imported in a scrum board which gave a clear visualization of the development process. In the daily scrum meeting each developer explained what he had done so far, what did not go well and what he was going to do next. Besides the daily meetings there was a meeting each week with the Astronomical Instrumentation group to discuss the progress and whether they were satisfied with the product so far. These meetings made it possible to make sure the software was as the Astronomical Instrumentation group desired.

After the planning phase the research phase was started. In this phase we investigated what was needed to develop the software in terms of software architecture, languages and tools, as well as hardware. The results of this research are processed in this report.

The next phase was development, which is the main phase of the project. In this phase all elements analyzed in the first two phases were applied. The result of the first part of the development phase was that we had a version containing all desired functions in four weeks. Despite that some features still needed testing and improvement, the basic functionality was established. The second part of the development phase consisted of this testing and improving of the quality of the program, as well as writing this report.

3
Control Methods

After examining the source code of the previous controller it was determined that controlling the cryostat was incomplete as it did not control the complete cryostat. Methods of controlling machines were researched which led to an enhanced control method. This chapter will describe the research conducted resulting in an implementation of this method.

The information sources used are the source code of the previous *He7Logger* program, papers describing cryostat control systems and basic control theory books. First the specific challenges encountered in the cryostat are described in Section 3.1. Then, the different elements to be controlled and the hierarchy between them is explained in Section 3.2. Finally, the research and design process of different controllers are described in Section 3.3 and 3.4.

3.1. Challenges

To successfully implement a new control method for the cryostat several challenges have to be solved. These challenges consist of:

- **Heat conductance**
  The cryostat is designed in such a way that the heat conductance between parts changes based on its temperature and whether the helium inside is liquid or gaseous. Heating one part of it can cause a chain reaction of other parts heating or cooling.
- **Changing heat capacity**
  The heat capacity of copper is not constant, and approaches 0 at low temperatures (see Figure 3.1) This means that the rate of temperature change due to heat being delivered changes when the temperature changes.
- **Thermodynamic processes**
  The thermodynamic processes inside the cryostat need to be predicted in order to successfully control the device for two reasons. First, the ability to cool down to the lowest possible temperature depends on keeping the helium in the correct phase before starting the final process. Second, phase change processes can keep parts of the cooler at the same temperature until all material has evaporated or condensed.
- **Time delay** The measurement sample rate and physical effects inside the cooler cause a time delay between actions performed by the controller and the response from the device.

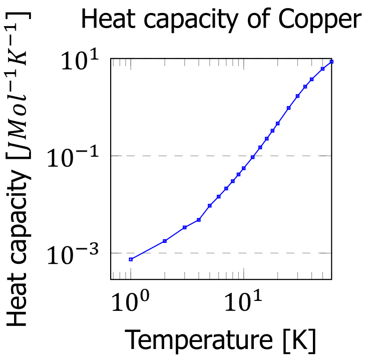


Figure 3.1: The heat capacity of copper from 1 to 60K in log scale on both x and y axis. Source: NIST [3]

The following sections will elaborate on how these challenges have been solved.

3.2. Controller Hierarchy

The controller is divided into several sub-controllers, which separates the responsibilities of the controller. A sub-controller can be replaced or maintained without affecting the other parts of the controller.

Figure 3.2 shows the hierarchy of the different controllers in the program.

On the top level is the sequential controller described in Section 3.3, this controller's responsibility is to automate the steps of the cool down process. It tracks the state of the cooler, and sends setpoints to the other controllers based on its state. It has to solve the issues of heat conductance and managing the thermodynamic processes in the cryostat.

The heater is controlled by a Lakeshore 355 temperature controller described in Section 4.2.1. It implements its own proprietary controller to heat the cryostat back to room temperature after an experiment. The compressor is the active element of the Pulse Tube Refrigerator that cools the cryostat down to 3K. It has its own internal control system and is able to be activated or deactivated by the sequential controller.

The He-7 cooler needs to be actively controlled by the program. The goal is to cool its cold head, but this cannot be controlled directly. The only elements that can be controlled on the He-7 cooler are heater elements mounted on the He-3 and He-4 pumps and the He-3 and He-4 heat switches. These heaters are used to keep the pumps they are mounted on heated at a certain temperature depending on the outputs of the sequential controller. The He-3 and He-4 heat pumps need to be actively temperature controlled using a closed-loop controller, which is explained in Section 3.4, to keep them at a constant temperature. The closed loop controller needs to be able to function with changing heat capacity and a large time delay. The heat switches are designed in such a way that a constant supply of heat from the heater will keep them at the required temperature, and therefore do not need active control with feedback.

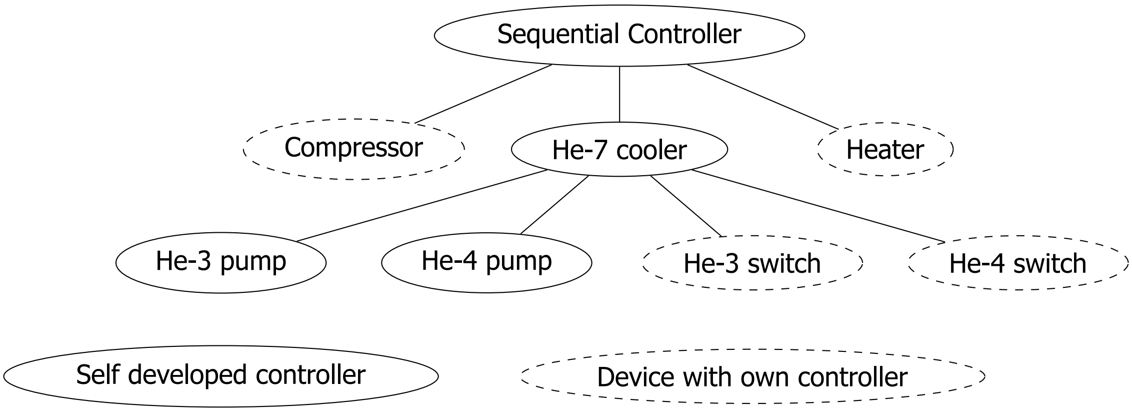


Figure 3.2: The controller hierarchy implemented in the program. The solid circles are new controllers implemented in this program. The dashed circles are devices that do not require active control, and only require on/off input from the higher level controller.

3.3. Sequential Control

A sequential controller automates a process by setting outputs based on events occurring or conditions being fulfilled. The program uses a sequential controller to automate the cool down and warm up processes.

The cool down process is dependent on the state of the helium contained in the he-7 cooler. The helium can be either in a gaseous non-adsorbed state, be adsorbed into the activated charcoal, or in a liquid state. Since this cannot be observed in the cooler using the attached equipment, the state of the controller cannot be defined by only the state of the measured quantities. Two options were considered for this problem: the first option is to always perform steps in the cool down sequence that force the helium into a known state, and let the state of the cooler depend on the previous steps taken. The second option is to estimate the state of the helium based on measured quantities. Because the risk of an incorrect estimation of the state of the helium is greater than the advantages it could have, it was decided to always follow a set of steps that cause the cooler to cool down from a known state.

A process where the current state is defined by the previous state and transitions can be described by a finite state machine (FSM). A FSM can be implemented as either a Moore machine [4], or a Mealy machine [5]. A Moore machine has outputs that are only dependent on the current state of the machine. The Mealy machine's outputs are defined both for states, and during state transitions, giving it the flexibility of having its output to be defined by its inputs.

The sequential controller in our program as can be seen in Figure 3.3 is defined as a Moore FSM. The choice to use a Moore machine rather than a Mealy machine was made to keep the definition of actions performed in each state as clear as possible by not having them depend on the inputs. This was done in order to minimize the chance of faults made in the programming and make it easy to be reviewed by the user. Giving each action performed a separate state also makes it easier to handle some hardware errors that might occur while performing an action: the state machine will stay in the same state and reattempt the same action until performed correctly, or some other state-transition condition has been fulfilled. The software implementation of the state machine is discussed in Section 4.2.7.

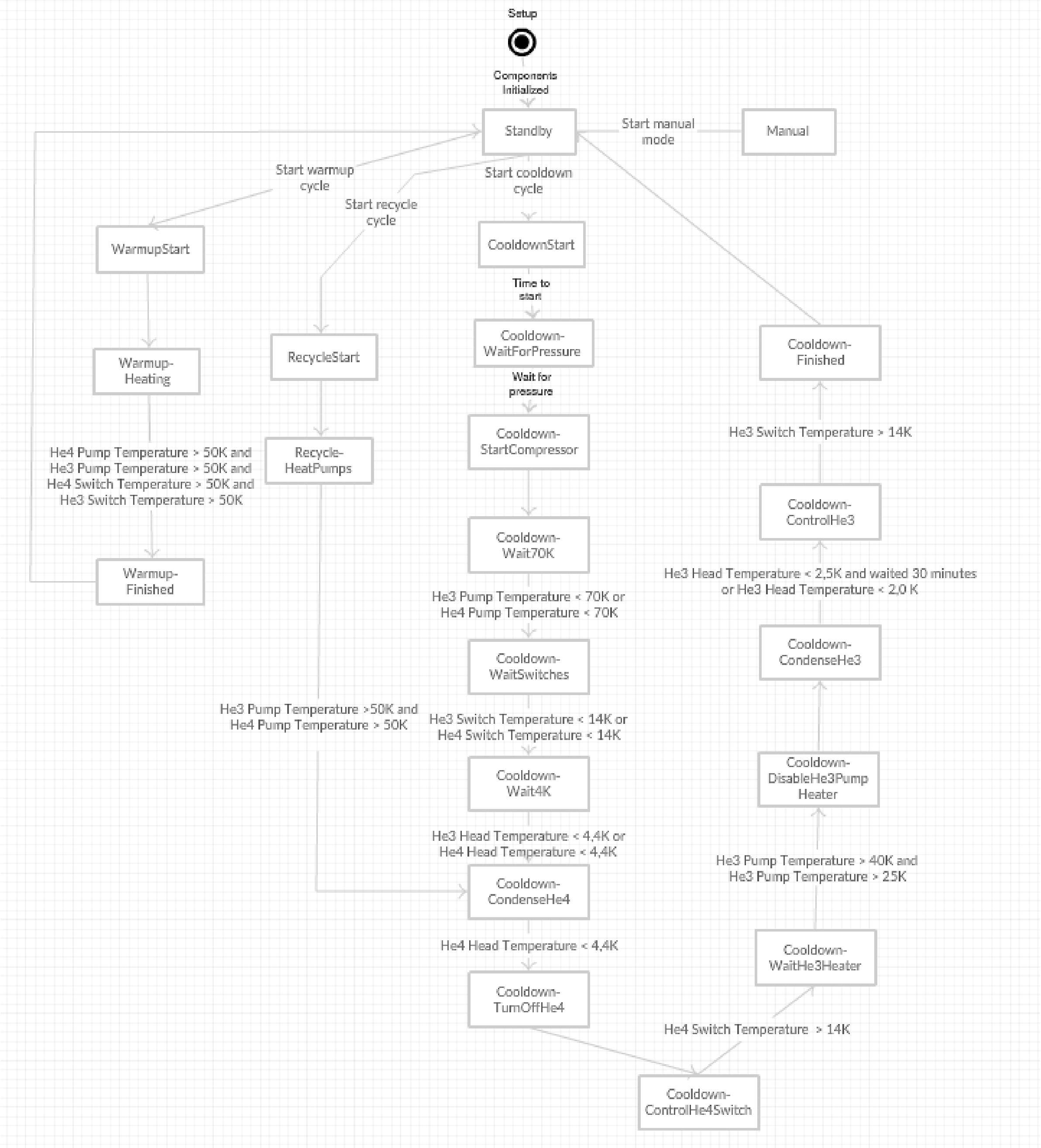

Setup
Components Initialized
Standby
Start manual mode
Manual
Start warmup cycle
Start cooldown cycle
Start recycle cycle
WarmupStart
CooldownStart
Time to start
RecycleStart
Warmup-Heating
Cooldown-WaitForPressure
Wait for pressure
Cooldown-Finished
He3 Switch Temperature > 14K
Cooldown-StartCompressor
He4 Pump Temperature > 50K and
He3 Pump Temperature > 50K and
He4 Switch Temperature > 50K and
He3 Switch Temperature > 50K
Recycle-HeatPumps
Cooldown-ControlHe3
Cooldown-Wait70K
Warmup-Finished
He3 Head Temperature < 2,5K and waited 30 minutes
or He3 Head Temperature < 2,0 K
He3 Pump Temperature < 70K or
He4 Pump Temperature < 70K
Cooldown-CondenseHe3
Cooldown-WaitSwitches
He3 Pump Temperature >50K and
He4 Pump Temperature > 50K
He3 Switch Temperature < 14K or
He4 Switch Temperature < 14K
Cooldown-DisableHe3Pump Heater
Cooldown-Wait4K
He3 Head Temperature < 4,4K or
He4 Head Temperature < 4,4K
He3 Pump Temperature > 40K and
He3 Pump Temperature > 25K
Cooldown-CondenseHe4
He4 Head Temperature < 4,4K
Cooldown-WaitHe3Heater
Cooldown-TurnOffHe4
He4 Switch Temperature > 14K
Cooldown-ControlHe4Switch

Figure 3.3: The state diagram of the implemented finite-state machine showing all states as blocks, and state transition conditions along the arrows. All actions are performed inside a state, rather than on transitions. Examples of this are the states CooldownTurnOffHe4 and CooldownStartCompressor.

3.4. Closed-loop Controller

During several parts of the cool down sequence, the He-3 and He-4 pumps need to be held at a certain temperature. Since there is no internal feedback system able to keep the pumps at a certain temperature, a digital feedback system that is implemented in the program is required. The general term for a system that attempts to reach a value (in this case temperature) by controlling a different value (in this case heater voltage) using feedback (the temperature sensors) is called a closed-loop controller. All closed-loop controllers depend on a feedback system where the output of the system and the setpoint are subtracted to calculate the error [6] (see Figure 3.4). The goal of the control system is to minimize the error signal. In this project, the error signal is defined to be positive when the actual temperature is less than the temperature setpoint.

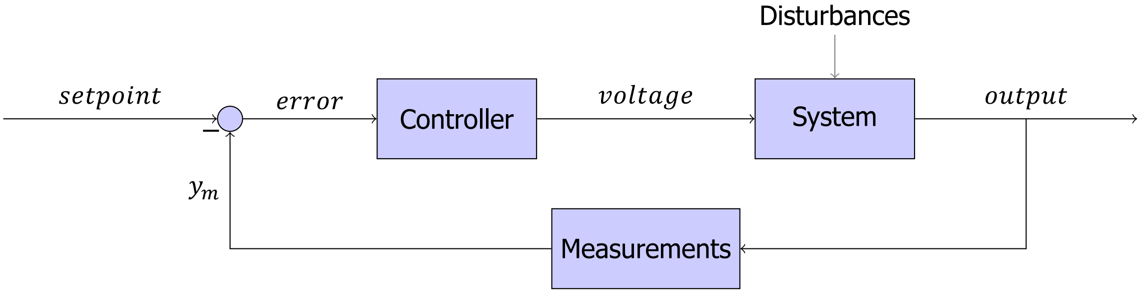


Figure 3.4: Schematic of a closed loop control system. The error is calculated by subtracting the system output and the setpoint. The controller changes the process variable, in this case voltage, according to the error. In the system, the voltage, external disturbances and unknown internal parameters generate a new output. This chapter discusses the implementation of the Controller block.

Prior Research

Most prior research found focuses on achieving optimal control performance over a high temperature range (between 0 - 300K) where the controller reaches its setpoint as quickly and matches it as closely as possible. This is challenging because of the changing heat capacity and physical delay in the system when varying the temperature over a large range. Previous research has been done by Dexter et. al. in multiple papers. In the first published papers they controlled the temperature of an element placed inside a liquid helium cryostat over a large temperature range using a self-learning neurofuzzy controller [7, 8]. This controller works based on a set of self-learned rules that define the relation between input and output. Later, the same authors used a fussy gain scheduler manually configured based on the open-loop response of the cryostat to change the gain parameters of a PID controller (explained later) based on the cryostat temperature [9]. They also note that for control in a small range at low temperatures (0 - 20K), a controller where the output depends on the error signal, and the integral of the error signal over time (PI) with fixed gain parameters achieves satisfactory results.

Based on these results, it was decided to start by implementing a PI controller, and possibly extend to a gain-scheduled PID controller if the achieved results are unsatisfactory.

On/off Controller

The performance of the on/off controller used in the old *He7Logger* program has been reviewed in order to compare the other control methods against it. It switches the heater on when $error > 0$ and off when $error \leq 0$. Figure 3.5 shows how this controller heats the He-3 pump in the cryostat. This type of controller causes large variations in temperature and significantly overshoots the temperature setpoint. The project owner also noticed that the large variations in electrical current also affected the measurements taken with experiments.

Proportional Controller

The controller that was first evaluated was a proportional controller which scales its output power with the error between the temperature setpoint and the current temperature (Figure 3.6). The scaling is determined by the proportional gain parameter Kp. The power is then converted to a voltage using the heater resistance as stated in the He-7 cooler manual. The gain parameter was determined by doing multiple experiments on the cooler with different values.

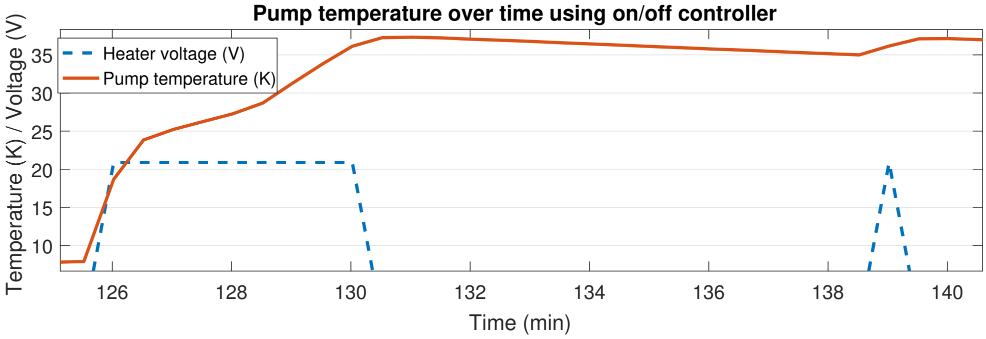


Figure 3.5: Response of a pump heater with an on/off controller used in the prior used program. Sample rate = 30s, setpoint = 35K.

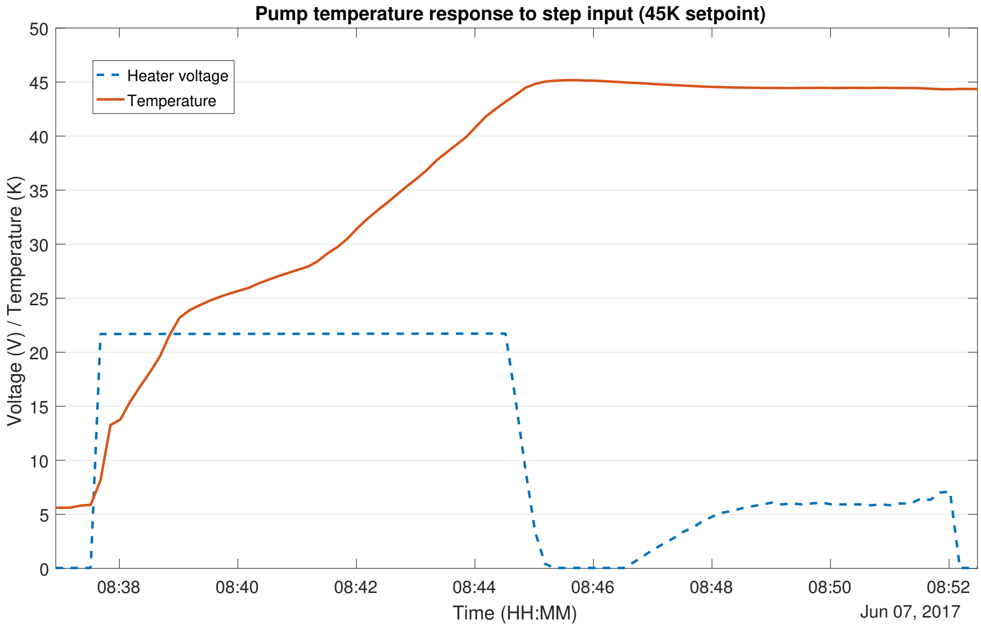


Figure 3.7: Response of a pump heater with a proportional controller. Sample rate = 5s, setpoint = 45K, Kp = 0.5

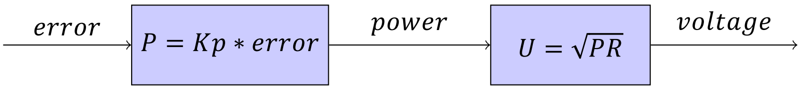


Figure 3.6: Schematic overview of the proportional controller.

The behavior of this controller can be seen in Figure 3.7. A downside of this controller is that the controller requires an error to give output. This makes the controller unable to remove a small constant offset called the steady state error caused by a constant disturbance. In this case, because of heat radiation and conduction, the actual temperature will always be less than the setpoint. To minimize this effect a high gain is required, but this causes a large overshoot during the initial heating, and can cause oscillations at even higher gains.

As a strategy to improve the performance at different temperatures it was attempted to scale Kp based on the setpoint value, giving a higher gain at higher temperatures. However, there was not enough time to test this over the complete temperature range, it was decided not to use this in the final version.

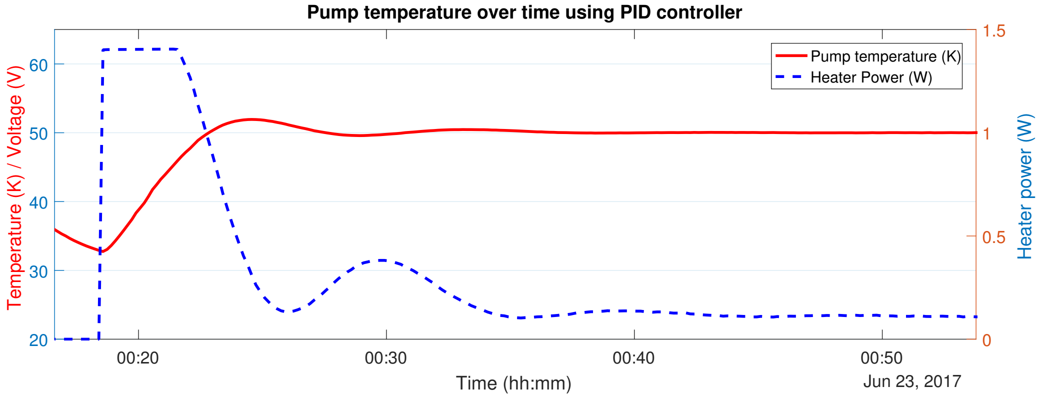


Figure 3.9: Response of a pump heater with a PID controller. Sample rate = 4s, setpoint = 50K, Kp = 0.18, Ki = 0.004, Kd = 0.5. This controller has a significant overshoot at first because of integrator windup.

PID Controller

A PID controller improves over the proportional controller by including an integral term which cancels constant offsets, and a derivative term which dampens the proportional term [6] (see Figure 3.8). The integral term gives an output based on the sum of all previous errors, causing it to increase the heater power in response to a steady state error. The derivative term gives an output based on the derivative of the error and decreases the total power if the temperature error is rapidly decreasing, and increases the power if the error is rapidly increasing.

Together, a properly tuned PID controller can overcome most of the limitations of the proportional controller: the integral term removes the steady state error, and the derivative term retards the temperature change to prevent large overshoots and instability due to delay.

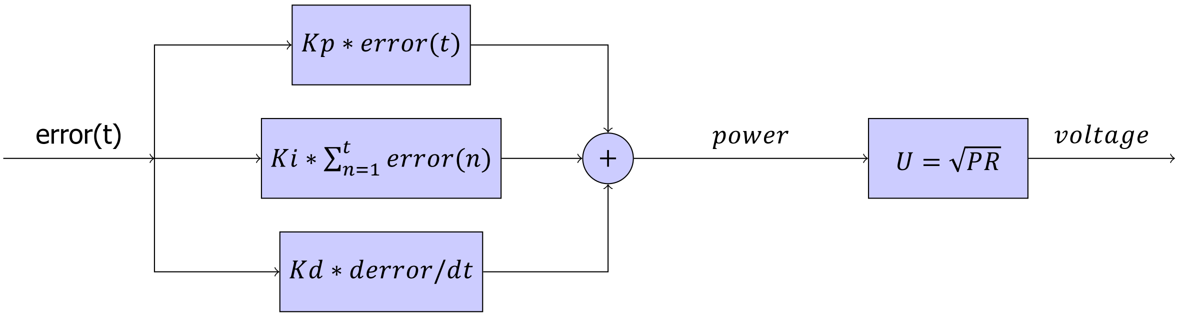


Figure 3.8: Schematic overview of PID controller.

The PID controller comes with several additional challenges. If the integral term reacts to the initial heating stage, where the response is limited by the maximum heater output, it will cause a large overshoot of the setpoint. This is called integrator windup. The effect can be seen in Figure 3.9. A simple approach to limit this effect is to stop integrating when the heater power is at its maximum, and set a maximum value of the integrator. Noise in the error signal can cause large outputs in the derivative term, a low pass filter can be used on the error signal to reduce this effect.

4
Software Engineering

In this chapter the software aspects of the program are described. To get a better understanding of the program, the program is explained with an overview of the server, graphical user interface and the external API (Section 4.1). Next the decisions relating to and implementation of the program are discussed (Section 4.2). In this section it is discussed why and how the main features of the program were implemented. This will include the justification for the used software architectures and if a framework is used or it was built from scratch.

4.1. The Program

The developed program consists of two parts: the server and the clients. The server is the heart of the system and contains all the logic of controlling the cryostat. The client consists of two types, called the native client and the external API which can be used for outside clients. The native client consists of the Graphical User Interface (GUI) and contains little logic. The client basically controls and visualizes the logic of the server. The external API allows the user to retrieve or set specific data of the cryostat during an experiment. The working of the whole system is explained by giving an overview of all its components.

4.1.1. Server

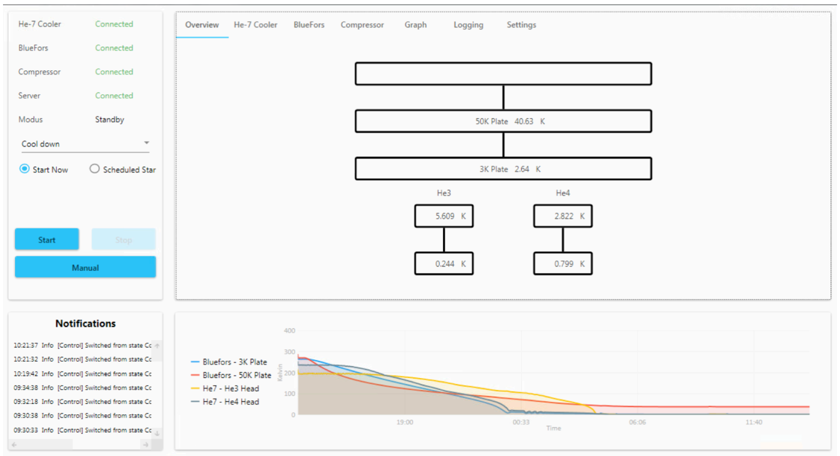
The server has an interface for the devices connected to the cryostat and a network interface for clients. The network interface consists of two parts: one for native windows clients (see section below) and one for python scripts (see last section). The network interface is elaborated in more detail in Section 4.2.3. The server is also responsible for controlling the processes regarding the cooling of the cryostat. It contains a FSM which controls cool down, recycle and warm up processes of the cryostat. Apart from the FSM the server can set the values for each heater to allow customized control, or test whether the system works correctly. The control system is elaborated in more detail in section 4.2.7. The server also contains the logic for logging the data, which can be sensor readouts, important information or errors. The functionality of the logger is available in Section 4.2.6.

4.1.2. Native Client

The Graphical User Interface (GUI) of the developed program is a way to visualize and control data of the server. The GUI consists of four frames: a general information frame that contains connection information and has the ability to start and stop the system, a notification frame showing important information and possible warnings and errors, a chart visualizing temperatures (see Section 4.2.5) and a frame consisting of multiple tabs, which should give a clear overview of all processes running on the server such as retrieving temperatures and setting manual controls of the cryostat for each device. These tabs separate information and display the data schematically which relates to the actual setup of the hardware. In the overview tab (see Figure 4.1) there is an overview of the data retrieved from sensor readings that the user considers most important. This data consists of temperatures of different parts of the system which are viewed in more detail in the other tabs. In the He7 Cooler tab (see Figure 4.2) there is an overview of all components associated with the He7 Cooler. In this tab the temperature of all these components is visualized and there is an option to set the voltage of some components manually. The Bluefors tab (see Figure 4.3) is consisting of a schematic overview as well, showing the temperature of the 50K and 3K shield. The compressor tab (see Figure 4.4) shows pressure values and temperatures and has the option to turn the compressor on or off. The other tabs do not have schematic overviews, it shows respectively a chart visualizing temperatures of many sensor readings (see also Figure 4.5), an option menu for logging specific data (see Figure 4.6) and a list of manual settings (see Figure 4.7) which gives the user the possibility to adjust the values for a cool down, recycle and heating process.

4.1.3. External API

The external API allows the user to do specific experiments without the need of a GUI. With an external API, which is based on REST architecture as explained in Section 4.2.3. The API is created for the user which can use a Python script to retrieve real time values of the available sensors. It is also possible to write values to the heaters and start or stop the control processes. Apart from Python scripts it is possible to access the API using external applications which can communicate via a REST API. In Appendix D a documentation can be found for this REST API.



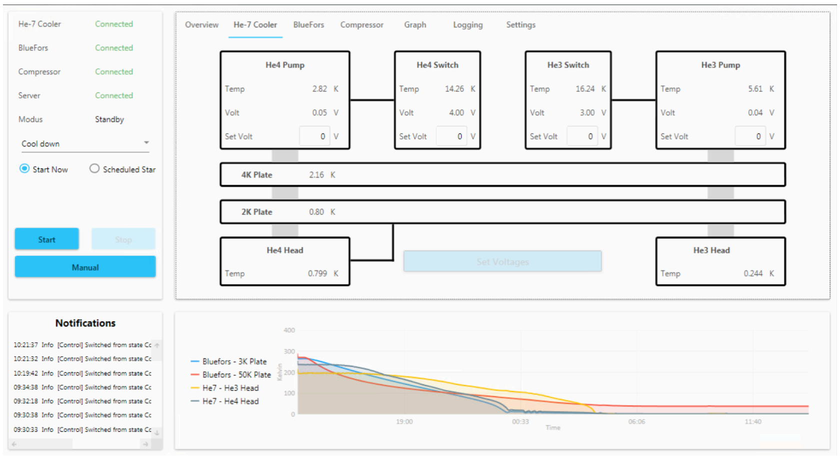
He-7 Cooler Connected
BlueFors Connected
Compressor Connected
Server Connected
Modus Standby
Cool down
● Start Now ○ Scheduled Star

Notifications

```
10:21:37 Info [Control] Switched from state Cc
10:21:32 Info [Control] Switched from state Cc
10:19:42 Info [Control] Switched from state Cc
09:34:38 Info [Control] Switched from state Cc
09:32:18 Info [Control] Switched from state Cc
09:30:38 Info [Control] Switched from state Cc
09:30:33 Info [Control] Switched from state Cc
```

50K Plate 40.63 K
3K Plate 2.64 K
He3 5.609 K, 0.244 K
He4 2.822 K, 0.799 K

Figure 4.1: The GUI of the program as it can be seen on start up. The GUI consists of four frames: on the left upper side an information screen consisting of connection information and the ability to start and stop the system. On the right upper side a tab frame consisting of multiple views (see also figures below). On the left lower side a notification screen displaying important information, warnings or errors. On the right lower side a graph displaying temperatures of the Bluefors and He-7.



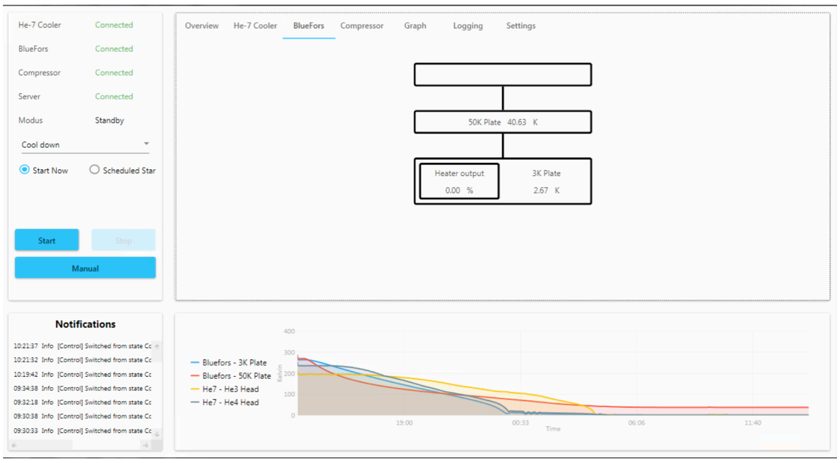
He-7 Cooler Connected
BlueFors Connected
Compressor Connected
Server Connected
Modus Standby
Cool down
● Start Now ○ Scheduled Star

Notifications

```
10:21:37 Info [Control] Switched from state Cc
10:21:32 Info [Control] Switched from state Cc
10:19:42 Info [Control] Switched from state Cc
09:34:38 Info [Control] Switched from state Cc
09:32:18 Info [Control] Switched from state Cc
09:30:38 Info [Control] Switched from state Cc
09:30:33 Info [Control] Switched from state Cc
```

He4 Pump: Temp 2.82 K, Volt 0.05 V, Set Volt 0 V
He4 Switch: Temp 14.26 K, Volt 4.00 V, Set Volt 0 V
He3 Switch: Temp 16.24 K, Volt 3.00 V, Set Volt 0 V
He3 Pump: Temp 5.61 K, Volt 0.04 V, Set Volt 0 V
4K Plate 2.16 K
2K Plate 0.80 K
He4 Head: Temp 0.799 K
He3 Head: Temp 0.244 K

Figure 4.2: The He-7 tab displaying all temperatures of the components of the He-7 Cooler and the ability to update the voltages of some of the components.

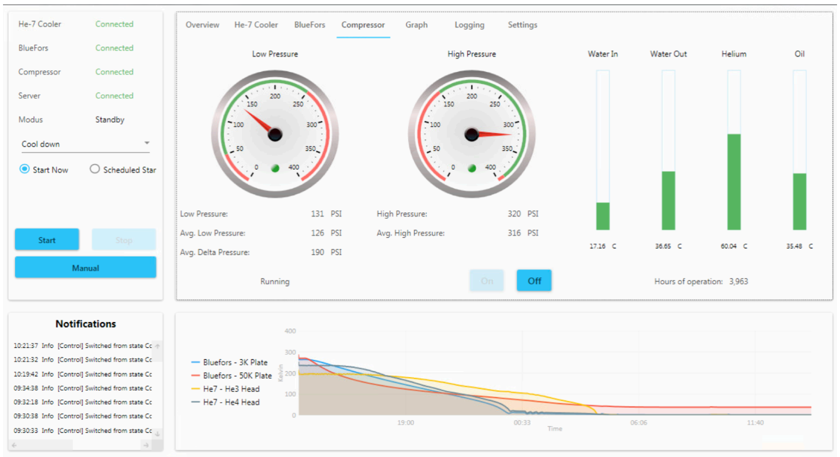


He-7 Cooler Connected
BlueFors Connected
Compressor Connected
Server Connected
Modus Standby
Cool down
● Start Now ○ Scheduled Star

50K Plate 40.63 K
Heater output 0.00 %
3K Plate 2.67 K

```
10:21:37  Info  [Control] Switched from state Cc
10:21:32  Info  [Control] Switched from state Cc
10:19:42  Info  [Control] Switched from state Cc
09:34:38  Info  [Control] Switched from state Cc
09:32:18  Info  [Control] Switched from state Cc
09:30:38  Info  [Control] Switched from state Cc
09:30:33  Info  [Control] Switched from state Cc
```

Figure 4.3: The Bluefors tab which displays the temperature of the 50K and 3K shield.



He-7 Cooler Connected
BlueFors Connected
Compressor Connected
Server Connected
Modus Standby
Cool down
● Start Now ○ Scheduled Star

Low Pressure: 131 PSI
Avg. Low Pressure: 126 PSI
Avg. Delta Pressure: 190 PSI
High Pressure: 320 PSI
Avg. High Pressure: 316 PSI

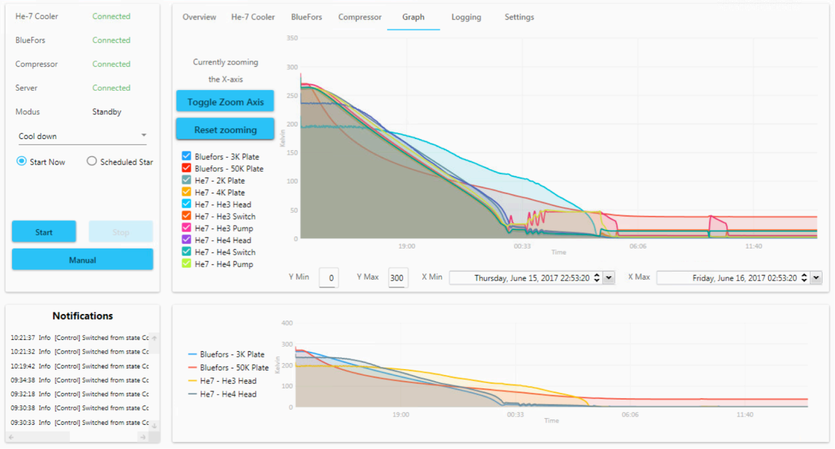
Water In 17.16 C
Water Out 36.65 C
Helium 60.04 C
Oil 35.48 C

Running

Hours of operation: 3,963

```
10:21:37  Info  [Control] Switched from state Cc
10:21:32  Info  [Control] Switched from state Cc
10:19:42  Info  [Control] Switched from state Cc
09:34:38  Info  [Control] Switched from state Cc
09:32:18  Info  [Control] Switched from state Cc
09:30:38  Info  [Control] Switched from state Cc
09:30:33  Info  [Control] Switched from state Cc
```

Figure 4.4: The Compressor tab showing gauges with important pressure information, temperatures of the liquids inside the compressor and it has ability to turn the compressor on or off



He-7 Cooler Connected
BlueFors Connected
Compressor Connected
Server Connected
Modus Standby
Cool down
● Start Now ○ Scheduled Star

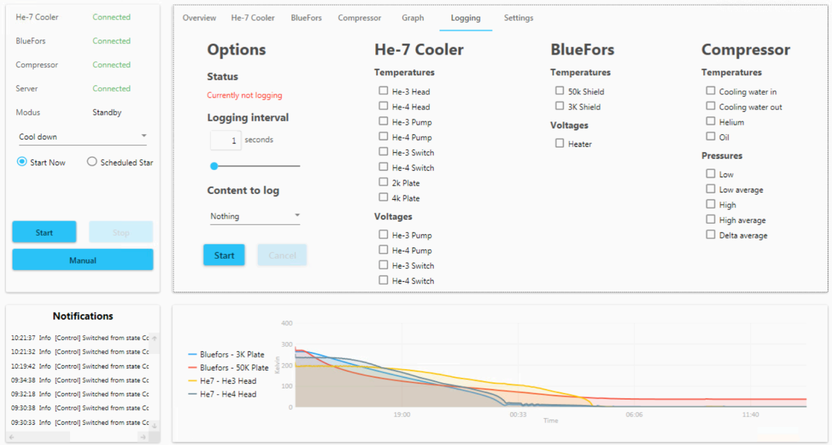
Currently zooming the X-axis

☑ Bluefors - 3K Plate
☑ Bluefors - 50K Plate
☑ He7 - 2K Plate
☑ He7 - 4K Plate
☑ He7 - He3 Head
☑ He7 - He3 Switch
☑ He7 - He3 Pump
☑ He7 - He4 Head
☑ He7 - He4 Switch
☑ He7 - He4 Pump

Y Min 0 Y Max 300 X Min Thursday, June 15, 2017 22:53:20 X Max Friday, June 16, 2017 02:53:20

```
10:21:37  Info  [Control] Switched from state Cc
10:21:32  Info  [Control] Switched from state Cc
10:19:42  Info  [Control] Switched from state Cc
09:34:38  Info  [Control] Switched from state Cc
09:32:18  Info  [Control] Switched from state Cc
09:30:38  Info  [Control] Switched from state Cc
09:30:33  Info  [Control] Switched from state Cc
```

Figure 4.5: The Chart tab visualizing temperatures of different components of the cryostat. It has the ability to zoom on x and y axis.



He-7 Cooler Connected
BlueFors Connected
Compressor Connected
Server Connected
Modus Standby

Cool down

● Start Now ○ Scheduled Star

Overview | He-7 Cooler | BlueFors | Compressor | Graph | Logging | Settings

Options

Status

Currently not logging

Logging interval

1 seconds

Content to log

Nothing

He-7 Cooler

Temperatures

- □ He-3 Head
- □ He-4 Head
- □ He-3 Pump
- □ He-4 Pump
- □ He-3 Switch
- □ He-4 Switch
- □ 2k Plate
- □ 4k Plate

Voltages

- □ He-3 Pump
- □ He-4 Pump
- □ He-3 Switch
- □ He-4 Switch

BlueFors

Temperatures

- □ 50k Shield
- □ 3K Shield

Voltages

- □ Heater

Compressor

Temperatures

- □ Cooling water in
- □ Cooling water out
- □ Helium
- □ Oil

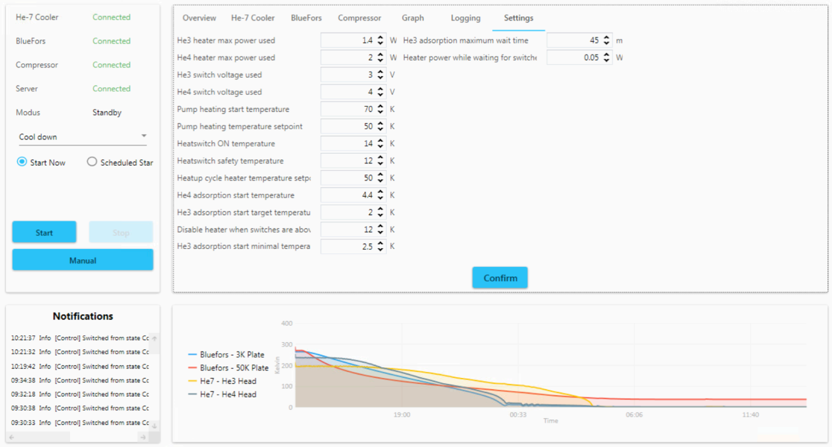
Pressures

- □ Low
- □ Low average
- □ High
- □ High average
- □ Delta average

Notifications

```
10:21:37 Info [Control] Switched from state Cc
10:21:32 Info [Control] Switched from state Cc
10:19:42 Info [Control] Switched from state Cc
09:34:38 Info [Control] Switched from state Cc
09:32:18 Info [Control] Switched from state Cc
09:30:38 Info [Control] Switched from state Cc
09:30:33 Info [Control] Switched from state Cc
```

Figure 4.6: The Logging tab allows the user to specify which data should be logged with which interval. There is also an option for a preset shown under the text ”Content to log”. Each time the logging is restarted, a new file will be created.



He-7 Cooler Connected
BlueFors Connected
Compressor Connected
Server Connected
Modus Standby

Cool down

● Start Now ○ Scheduled Star

Overview | He-7 Cooler | BlueFors | Compressor | Graph | Logging | Settings

He3 heater max power used	1.4	W	He3 adsorption maximum wait time	45	m
He4 heater max power used	2	W	Heater power while waiting for switche	0.05	W
He3 switch voltage used	3	V			
He4 switch voltage used	4	V			
Pump heating start temperature	70	K			
Pump heating temperature setpoint	50	K			
Heatswitch ON temperature	14	K			
Heatswitch safety temperature	12	K			
Heatup cycle heater temperature setp	50	K			
He4 adsorption start temperature	4.4	K			
He3 adsorption start target temperatu	2	K			
Disable heater when switches are abov	12	K			
He3 adsorption start minimal tempera	2.5	K			

Notifications

```
10:21:37 Info [Control] Switched from state Cc
10:21:32 Info [Control] Switched from state Cc
10:19:42 Info [Control] Switched from state Cc
09:34:38 Info [Control] Switched from state Cc
09:32:18 Info [Control] Switched from state Cc
09:30:38 Info [Control] Switched from state Cc
09:30:33 Info [Control] Switched from state Cc
```

Figure 4.7: The Settings tab shows all values that are necessary in order to do a cool down, recycle or heating process. The values will not change frequently, though the user should be able to tweak these settings if necessary.

4.2. Software Decisions and Implementation

In this section the decisions relating to and the implementation of the program is explained, starting with the choice of hardware cables and operating system (Section 4.2.1). Second the motivation of the choice of the programming language is discussed (Section 4.2.2). Next the client server is discussed in Section 4.2.3. Then the architecture of the native client is explained in Section 4.2.4. Following this, the charts and used framework in the client are discussed in Section 4.2.5. Next the different types of logging are explained, along with an explanation of how the log files are stored (Section 4.2.6). The section is closed with an explanation of the implemented controller, which explains how the control processes are controlled by the final state machine, including the control processes of the components of the cryostat (Section 4.2.7).

4.2.1. Hardware and Operating System

In the research phase the current hardware used to control the cryostat was investigated. The available hardware consisted of a simple Windows PC (Intel core i5-650 3.2GHz processor, 2GB physical memory), two controlling devices (a Lakeshore temperature controller [10] and an Agilent 34972A controller [11]) which were already connected to the PC, the cryostat, and another control device (a compressor [12]) which was was not connected to the PC (See Figure 4.8). An exact definition of these devices can be found in table 1.1

Hardware

The temperature measured by the Lakeshore can be read using a virtual com port over USB or an IEEE-488 (also called a GPIB: General Purpose Interface Bus) interface. Both are similar in capabilities but since USB has a cable length up to 5 meters [13], whereas IEEE-488 is limited by 2 meters, USB is the most suitable option. The Agilent controller has an USB and Ethernet interface. Ethernet cables are reliable with a length up to 100 meters [14]. On top of that Ethernet can be used with TCP which is a reliable protocol as it makes use of error control, which allows the sender to verify that package arrive. If a packet is lost it will be detected and will be resend to the correct device [15]. Therefore using the Ethernet interface is the best option for the Agilent controller. Regarding the compressor, the same arguments apply as for the Agilent controller, which makes the Ethernet interface the best solution as well.

The PC has only one Ethernet port, though it needs to have three Ethernet ports for communicating with the internet, the Agilent controller and the compressor. Therefore two Ethernet-to-USB converters were used in order to connect all devices properly.

Besides the three devices mentioned, there are also pressure sensors to measure the pressure inside the cryostat, see Section 1.2. Currently there is no method to read the output of these sensors, as it is not an essential element for the program.

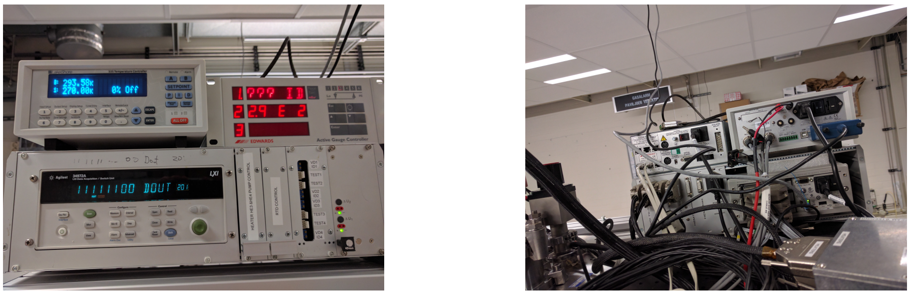


Figure 4.8: Control devices for the cryostat inside the laboratory. In the left: Agilent (green display) and Lakeshore (blue display) devices front panel. In the right: Rear side showing connections

Operating System

In this project, the PC functions as a server; it receives data from all devices and the program handles the logic, see Section 4.2.3. The operating system (OS) that runs on this PC is Windows, this was not a choice but a requirement, see Section 2.1. This OS is a preference of the Astronomical Instrumentation group and should be able to run the program. To conclude, the server should run on a Windows PC and, as the Astronomical Instrumentation group primarily makes use of Windows devices, the clients/GUI (see Section 4.2.3) should run this OS as well.

4.2.2. Programming Language

To choose a programming language for this project several commonly used languages, which allow the implementation of the desired client server architecture, were considered. Both the client and the server would have to run on a Windows computer, which lead to the considered languages being: C#, Java, C++ and Python. These languages were compared against four important functions, explained below.

To make the decision which programming language would be best, a list of required functionalities would have to be made. First off their ability to create a GUI was rated. In C# and the WPF framework provides developers with a consistent programming tool for building user interfaces. The only downsides to using WPF would be its limitation to Windows, which will be the system that is run on the computer, while it is widely used and well supported. Building a GUI in Java is also very solid and it is usable on multiple operating systems. While C++ can be powerful when creating a GUI, it has a steep learning curve and we do not have any experience with it. Finally, Python is easy to implement, but is generally not a good option for creating larger applications.

Since the project consist of a server, which will have to communicate with three devices over different network protocols, and a client the programming language has to support these functionalities. In Section 4.2.1 it was explained that communication over ethernet and USB is required. Both C# and C++ have solid support for networking through the native .NET framework. Java and Python can use external libraries to implement the networking.

Finally, to prevent errors and crashes and to ensure high quality of the program is it necessary to test the implemented code. In C# the Microsoft Visual Studio Test tools offer good unit testing capabilities. Java and C++ offer similar options, while with Python testing is quite limited.

An overview of the evaluation of all languages can be found in Table 4.1. After evaluating all languages the conclusion is that C# and Java would both be a solid choice that are well suited for implementing this project. The final decision was up to personal preference, so the decision to use C# was made.

	C#	**C++**	**Java**	**Python**
GUI	WPF is solid, limited to Windows	Powerful, but steep learning curve	Solid, multiple OS	Easy to learn, not suited for larger applications
Networking	Native support (.NET)	Native support (.NET)	External libraries	External libraries
Testing	Good testability	Good testability	Good testability	Bad testability

Table 4.1: An overview of the comparison of the programming languages. They have been compared against the ability to build a GUI, their capabilities of implementing a network architecture and finally their capability of integrating tests.

4.2.3. Client Server

In this section it is described why and how a client server architecture was implemented. In Section 2.2 it was decided to implement a client server architecture, to ensure extendability to multiple platforms. In Section 4.2.2 it was chosen to use C# as programming language, therefore the options for a client and server architecture in C# were researched. The research lead to multiple options to create a client server, namely creating a custom network layer on top of sockets and the usage of frameworks, both are separately explained. To conclude the choice for the client server implementation is discussed. In the second part the implementation of the client server is explained, including part of the Python client and the security.

Custom Network Layer

The initial search of a client server connection lead to the usage of plain sockets. There are several socket classes which can be used: the socket class [16, Socket], TCP socket classes [16, TCP Client, TCP Listener], the UDP socket class [16, UDP Client] and web socket [17, Web Socket]. The socket class implements the lowest level of network communication. This gives a low level of methods to use for the implementation of communication. The TCP and UDP classes are wrapper classes built on top of the socket classes, these classes provide additional methods on top of the socket class to reduce the amount of code needed to implement the communication. The web socket class provides the option to communicate easily over port 80. This is mostly used for web services who needs to communicate over port 80, as this is not needed, web socket is not evaluated further. The difference between TCP and UDP is that TCP is connection orientated and UDP is connectionless. With TCP is assured that a message is received, with UDP this is not the case. Although some sensor readings could be lost this does not count for commands and error messages, therefore a TCP connection is needed to ensure that the data is received. As a TCP connection is required the TCP classes can be used instead of the socket class to reduce the amount of code.

Frameworks

The usage of a framework decrease the amount of code needed to implement the network layer. Most of the basic features are already implemented, for example establishing a connection and sending data. There are a lot of frameworks available on the internet each with a different size of available methods. An important factor to chose a framework is the documentation and support, if the framework is not well documented and or supported, the framework does not give an advantage. From the research two larger frameworks were found namely ASP.NET [18] and Windows Communication Foundation (WCF) [19]. Besides the ASP.NET framework the ASP.NET Core framework [20] exist. This is an open source project which has a better performance than the original framework as it is optimized by the community. WCF and ASP.NET are high level frameworks available in the .NET framework.

Final Decision

There are three possible implementations for the client server architecture: TCP socket classes, ASP.NET framework and WCF framework. In Table 4.2 an overview is provided for the explained implementations. Out of those three implementations WCF is chosen.

The advantage of using a plain TCP socket is that the performance can be optimized regarding to the frameworks as there is no overhead in the data exchange, which is needed for the frameworks to follow the implemented protocol. WCF adds the most overhead and therefore has most performance loss. There is no need for a large amount of real time data transfer thus performance is not a critical factor. The advantage of the frameworks over a plain socket is that frameworks already have many methods and features implemented on top of the plain TCP socket, for WCF it is even possible to use an UDP socket. The usage of frameworks reduces the amount of code needed to implement the network communication. The ASP.NET framework is a lightweight framework which is designed for Representational State Transfer(REST) applications, while the WCF framework is designed for service-orientated applications(SOA). WCF uses from origin Simple Object Access Protocol (SOAP) although it can now also use REST. REST provides a public API which can be called via an URI. While SOAP is called via services predefined in an interface (further explained in Section 4.2.3). With the usage of REST and SOAP both frameworks can be extended to different platforms, although WCF needs to be hosted on a Windows server, which is no problem as it is required as described in Section 2.1. For the TCP socket classes extension is also possible but requires strict management of the type of data sent as the type of data might not be known by the receiver. For security the plain TCP socket does not have anything implemented yet, meaning a security layer has to be built which is less secure than the implemented security layers in the frameworks. WCF has a better security layer than ASP.NET [21] which is further described in Section 4.2.3. WCF provides the most reliable solution as there can be assured that an action is executed [21].

It is remarkable that WCF is a rather old framework, and SOAP is replaced in most cases with REST. This leads to some discussion if WCF is deprecated or not. According to an article WCF is dead [22], according to another article this is not the case [23]. From the official documentation of Microsoft deprecation is also countered as one can notice (30-3-2017 [19]). An advantage of an older framework is that most problems are already encountered and solutions are given. From the facts that WCF is extendable, offers the most reliability and security, and is easy to work with, WCF is picked as framework for client server architecture.

	TCP Socket	**ASP.NET**	**WCF**
Internet Protocol	TCP	TCP	TCP and UDP
Application Protocol	None	HTTP	HTTP
Message Protocol	None	REST	REST and SOAP
Expendability	Hard to implement	Good, with REST API	Good, with SOAP and REST API
Performance	High, no overhead	Good, little overhead	Decent, some overhead
Build in Security	None	Decent, transport security	Good, message security
Implemented features	None	Medium, focus on REST	A lot, focus on SOAP but also allows REST. It provides also a lot of settings.

Table 4.2: This table visualize the aspects used to compare the different options to implement a client server architecture. The options are: plain TCP socket, ASP.NET API and WCF. TCP socket is the bare minimum for creating a connection, while ASP.NET and WCF provide a framework for the connection together with communication. ASP.NET is a lightweight framework for REST API. WCF is designed for SOA which uses SOAP but also provides a REST API.

Service-Orientated Architecture

WCF enforces service-orientated architecture (SOA) which is an extension of the client server architecture, the server provides its services to the client who can use this service. The client can call the methods which are defined in the service and the rest of the server is a black box for the client. A difference is that by SOA the client and server are loosely coupled, meaning that the server can be used by multiple types of clients while by a client server architecture the server mostly exist for a single type of client.

Using WCF a service contract is created where operation contracts are stated. The service contract is stated in an interface and implemented in a regular class. A client can request the service contract from the server and the methods which are annotated with operation contract can be called from the client. Just like regular methods those methods can have a return type where data can be exchanged. For example the method *GetTemperature(int sensorID)* can return a double with the temperature of the sensor. Besides the operation contracts, data contracts can be used to sent a class with parameters. In the developed program mostly single values are used, so this feature is not used. Another type of contract which is used is a callback contract which allows the server to initiate communication with the client, which cannot be done with a normal operation contract. For error handling a fault contract is used which can send an exception to the client if an fault occurs. The client can handle the exception further, for example retry the call with a correct value.

The downside of those contracts is that only one service class can be hosted. This means that if there are multiple service contract interfaces which needs to be hosted with a single host, a single class must implement the interfaces. This is not ideal as a single connection is desired.

With the use of different bindings different types of communication protocols can be used. In the developed program wsHTTPBinding and wsDualHTTPBinding are used, both use SOAP over HTTP. The difference is that the dual binding also allows communication from the server to the client which is needed for callbacks. The difference with the basic HTTPBinding is that those bindings have additional WS- features.

External API Integration

A part of the program is to access the server with Python scripts during an experiments. This is done by creating an external REST API which is accessible by outside applications or scripts. The initial idea was to connect the Python scripts to the service contract, but there was no security layer possible between the server and the various Python libraries. The solution to this problem was to implement the REST API. To make a REST API the service contracts needed additional information to make a POST or GET method and which URI to use. The downside is that the callback methods are not supported, but this are mostly GUI updates which are not needed for the external API. The working of the external API was explained in Section 4.1.3.

Security

An important factor of network communication is the security between the client and the server. The cryostat is an expensive device which should not be accessible from unauthorized users. There are two different parts of the security namely communication security and authentication. Communication security is achieved by using message security for end to end encryption for the native client. The REST API uses HTTPS which is transport security which encrypts the data from point to point. Message security is more secure as it is end to end it. End to end encryption goes from application to application, whereas with point to point encryption goes from device to device. Message security requires more performance due to usage of XML-level security. This is not always available for each program which limits the usage of message security. HTTPS has a better performance and wider usage as it does not need XML-level security.

For authentication there are two certificates used. One for the REST API and one for the service contract. Besides the certificates a user name and a password is required to gain access to the services.

4.2.4. Client Architecture

This section will describe which architectural design pattern was chosen and implemented for the client. To implement the client, which consist mainly of a GUI to visualize data from the server, it was decided in Section 4.2.2 that the C# graphical subsystem WPF [24] would be used. MVVM is an architectural design pattern that separates the GUI from the logic in a program. There are three core components to the MVVM pattern: the model (the data), the view (the GUI) and the viewmodel (the link between the data and the GUI), see Figure 4.9. Since MVVM is well adapted to WPF and the most used pattern along with WPF, see [25], it was decided that the client would be build using the MVVM design pattern.

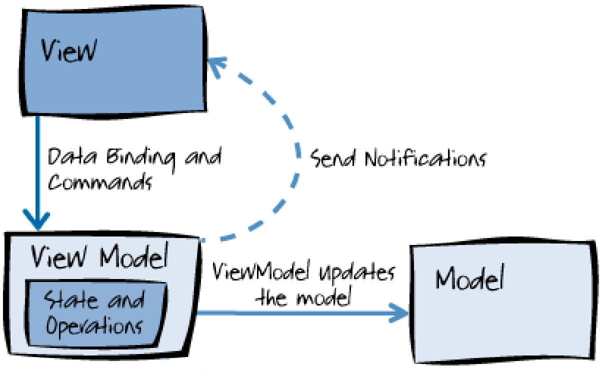


Figure 4.9: MVVM design pattern schematically. With the use of data bindings and commands a view is connected to a viewmodel. When the viewmodel is updated it updates the model and notifies its bindings and performs commands when needed, for instance when a button is clicked. Source: [26]

Model

The models are used to store all data and contains no logic regarding the views. Each of the three devices has its own model, which contains all information about that device. The modus model includes connection information with the devices and the server. The other models that were created are: the logging model, that contains which data should be logged; the message box model, which contains notifications; the chart model, which contains information about the charts; and the setting model which contains the settings.

Viewmodel

Each of the models has a view model which is the link from the model to the view. The view models are the link between what is shown in the view and what is stored in the model. They raise a property changed event whenever a value is updated, which notifies the views to update the data. When the view gets data from the view model it is converted by the view model to a value that is ready to be shown in the GUI. Most values are simply passed on with a decimal rounding, but some need to be converted from a double to a string. To bind a function to a button a relay command was created, which links a performed action to a method that will be executed.

View

Finally, the views are responsible for the structure, layout and appearance of the GUI. The views are defined in a XAML file that have little to no logic behind them. Unlike the model, not all view models have a single view. To that extend it was necessary to create a view model container that contained all view models. This made it possible to bind all fields in the GUI to the appropriate property.

4.2.5. Charts

A lot of data is retrieved from the server and in order to get a better overview of the cryostat, charts were implemented. The charts also have the option to customize them. First of all the charts should be able to zoom in on the data and it should be possible to scroll through the chart. Secondly it should be possible to filter out data, since sometimes the user is only interested in (a) certain line(s) of the chart, for both see the requirements in Section 2.1. Last of all the charts should not slow down the rest of the program, which was one of the hardest challenges regarding charts. To implement the chart it was decided a library would be used, since writing this code yourself would be like reinventing the wheel. A lot of libraries were paid so these quickly were rejected. It was decided to use the Live Charts library [27] since it is easy to use, allows the implementation of customization, looks professional and is free for non-commercial use. A preview of the implemented charts can be seen in Figure 4.5.

To implement the charts, all received data needed to be stored in order to be displayed in the charts. There are eight temperatures being read constantly. While implementing the charts a problem was encountered: adding all data points received from the server to the chart caused the chart to slow down the whole program. This already happened at around 200 points (which is equal to $200/8 = 25$ seconds with points being added every second). The graph update method was the issue and could not handle that the amount of data. The first solution was to not add all points to the chart but to take the average over a couple of seconds, and add that to the graph. Since the temperature is moving in the same direction constantly most of the time, this would cause minimal loss of data while increasing the quality of the charts. This indeed lead to a speed up of the program, but it still slowed down after a certain amount of points. The next solution was to limit the amount of data displayed in the graph. After reaching the maximum amount of points, the first point would be removed. The first problem was solved, but only a limited part of the chart was visible, which is not ideal. In order to solve this the regular version of Live Charts was updated to the Geared version, which was capable of handling bigger amounts of data. The use of this premium version was granted to us by the Live Charts developers.

Lastly the customization features of the chart were implemented: zooming and filtering. The Live Charts library supported zooming by default: it allows the user to define the axis to zoom in on (X, Y, both or none) and to use the mouse wheel to scroll. To toggle between these zooming modes a button was created. Next, button reset button had to be created to zoom out to show all data. Lastly a more precise zoom was created, which allows the user to enter the specific values of the area they want to see and to zoom in on that. Filtering could be done by changing the visibility of the lines. A check box was created for each line that changed the visibility of the corresponding line based on its value.

4.2.6. Logging

An important element of the system is that data retrieved from the cryostat is stored for later use. This data is needed when the user wants to do experiments with the cryostat. The designed logging system makes use of two separate data files: a general logging file and a specific logging file. The former will always log all data from the cryostat when the server is running. The latter will only log data specified by the user and will start and stop when the user clicks the button (see Figure 4.6). The specific logging file is introduced because it will reduce the amount of effort during experiments. It can be used when the user wants to do experiments and is only interested in certain data for a specific time. This will allow them to focus directly on their experiment instead of filtering the general logging file for their specific data first. These logging files both have the extension ”.csv” with a semicolon separated file format (see Appendix E), so it can be readout easily by spreadsheets, which is an advantage over non-separated file format.

The files will be saved in a folder according to the current date. If the program runs for multiple days, it will automatically create a new general logging file for each day to assure that files will not be too big in size.

Besides the data logging files there is a debug logging file as well. This file contains important information, warnings and errors that occur when the server is running. Whenever the system malfunctions, this file can help locate the problem. This debug information is also sent as notification to the client and visualized in the notification tab (discussed in Section 4.1).

4.2.7. Controller Implementation

The code of the control system is divided based on the structure described in Section 3.2.

To create a common interface for reading data from each device, the ISensor interface was created. Each device implements this interface and exposes its sensors over this interface. To set the outputs a Heater class exists, where for each He-7 heater an instance is created which controls the heater.

Hardware Interfaces

Both the Lakeshore and the Agilent use a plain text based protocol for communication [10, 11]. The Lakeshore uses a serial port connection, and the Agilent a TCP/IP connection. Because both devices use a plain text interface, a single class *managedstream.cs* is written that implements reading and writing strings to a stream. Sub-classes implement the functions for connecting to a TCP or serial port. Messages to and from the devices are ended with a newline terminator. A C# StringReader is used to read data send back from the devices until a terminator is detected. If no complete message is received within a set amount of time, an exception is thrown which will be handled by a higher level of the application. The Agilents commands are implemented in a class that reads and writes values to the different in and outputs of the device. All values are read using a single command. The device uses channel IDs to represent its inputs and outputs. The read values command uses an array of channel IDs as parameters and returns an array of read values. If the response is of an unexpected format, it will raise an AgilentException and the calling method should decide the correct action to resolve the issue. Because functions can spend time waiting between a write command an a read command, all functions acquire a lock on the connection to the device before starting communicating. The Lakeshore's commands are similarly implemented in a single class. All methods communicating to the device acquire a lock on the connection. Because the Lakeshore has a minimum interval between commands, the communications are delayed inside the lock until the minimum time has passed before starting communication.

Communication to the compressor uses a MODBUS interface. The NModBus4 [28] library is used to implement the interface in the application. Rather than starting a new reading, the interface reads values from registers on the device to respond to a request. Because of the fast interface, communication to the device happens on the main thread.

He-7 Cooler Controller

All functionality of the He-7 cooler is exposed over Sensor and Heater interfaces.

The raw sensor values read are voltages that represent the temperature of the sensor. To convert the voltages to temperature values, a calibration class is written. Calibration values are read from a file containing voltages and corresponding temperatures. After reading the file the data points are sorted based on voltage. When converting a voltage to a temperature, the two calibration points just above and under the read voltage are found, and the final value is linearly interpolated between them. Linear interpolation was chosen because the calibration files contain enough points to resolve non-linearity issues. An empty calibration without data points will return the same value as the input value. Values outside the calibration range will be capped to the highest and lowest calibrated values.

Communication with the Agilent is done asynchronously from the main thread since reading values can take over one second. Values are read at a set interval. When a sensor instance is created, it registers it self with the main He7cooler class which adds its channel ID to the list of values to read. Multiple sensor instances can exist using the same channel ID, and will result in the channel being read only once. The read values are cached in the He7cooler instance, and Sensor instances will use the cached value.

Heater instances can be used as simple output voltage devices, or used in closed-loop temperature control mode. Heaters can use a Calibration on their output, since the hardware used in the laboratory uses amplifiers to amplify the Agilent's output. If the heaters resistance is specified in the constructor, the output can be set and read using either the Voltage, Current or Power properties. If a temperature sensor is specified for feedback, the heater can operate in closed-loop temperature control mode. When new values are read by the main He7cooler thread, heaters are notified of new values. If temperature control is activated, a new power output is calculated and the voltage output is modified accordingly when the heaters are notified. Voltages are checked against minimum and maximum safety values at the last step before writing to the device, and are not set if the new value exceeds the limits. Voltage outputs are written to the device without delay.

The closed-loop controller is implemented as a PID controller the theory of which is described in Section 3.4. Anti-windup of the integral term is implemented by limiting the integrated value between 0 and 0.15W of output power, and only letting it integrate when the current heater power is less than the maximum. The derivative of the error is filtered with an exponential moving average. There are many better filters, but this one is easy to implement and very efficient, and should be good enough since the signal is very slow and does not have a lot of noise.

If the connection to the Agilent is interrupted, no new values will be read, but the cached values can still be used. After five seconds the TCP connection times out. The connection will be attempted to be re-established in the He7Cooler thread every 4 seconds.

Sequential Control Implementation

As described in Chapter 3, the automatic control functionality is implemented using a Moore type finite-state machine (FSM). The state machine runs in a separate thread, and is executed once every 4 seconds.

The steps in the FSM were initially based on the process used in both the *He7Logger* application, the He-7 cooler manual and the description by the Astronomical Instrumentation group. However, they have been adjusted based on tests on the cryostat and our understanding of the process. All parameters used in the state transitions and the state machine's outputs are modifiable using the GUI or REST interface and stored in a settings file.

The state machine is implemented in one function with a large switch statement. This design places all code of the state machine in one place. One might argue that having a large function is generally considered a bad practice in software engineering. Therefore we considered using the state pattern [29, page 338-348]. This pattern separates each state into a subclass of an abstract "state" class. Allowing the class to have different behaviors for multiple methods depending on the state, while keeping the cyclomatic complexity of each method low.

This pattern was not used because this specific use case contains a lot of states with little state specific behavior. Specifically, we have only one operation that changes with each state: the main loop, this loop has only one conditional: the main switch statement. Implementing the state pattern for this use-case would significantly increase the amount of non-functional code and the separation will result in a lack of overview.

Data Reader

To retrieve all the data from the server a data reader class is created which creates an array and fills it with data which can be public. This data consist of sensor readings from the ISensor classes which is implemented for each device, separate data from compressor which cannot be read out with the ISensor class and status information for each device, see the full list in Appendix F. For each sensor a ISensor class is created which gets the most recent value of the sensor. Those sensors are stored in a separate array which can be looped over when filling the data array. The retrieved data can be used for logging or sending the data to the client.

Writing values to a sensor is only implemented for the heaters inside the He-7 cooler. A heater class is created which can set the value of the heater. Turning the compressor on and off is done via a separate method in the compressor class.

Extendability

The extendability of the program is achieved by among others having the option of adding new devices with minimal changes to the existing code. Adding a new device to the cryostat, for example the vacuum pump, can be achieved by adding a new class which communicates with the device. The device class should implement ISensor for reading out the available sensors. If the ISensor interface is implemented, the only step that needs to be taken to integrate the new sensor is to add it to the data enumerator which contains all the available data.

If the device has values which can be set, the heater interface might be reused, or a separate interface similar to the ISensor should be created. As there is no indication that a device will be added soon no general interface, apart from the heater class, is used for setting values. Whenever the device should perform actions during the cool down, recycle or warm up process, this logic should be added to the corresponding state in the state machine.

5

Software Quality

To prevent errors and crashes and to ensure the quality of the program is high it is important to test the program. Testing can be done in many ways and a combination of several methods is the best way to guarantee that the program works correctly. In this chapter the different testing methods that were used are discussed. To ensure the whole program is working unit tests were written and system tests were performed by among others make use of a simulation (see Section 5.1 and 5.2). The use of continuous integration and reviewing on Github made sure that all new features were working (see Section 5.3). Next the feedback from the software improvement group, who checks the software quality by various software metrics, is discussed (see Section 5.4). To minimize the risk of failure a failure mode and effect analysis is performed (see Section 5.5).

5.1. Unit Testing

To test the functionality of small blocks of code, called units, Unit Testing was implemented. The goal here is to ensure the unit would do what was intended. The program consists of two parts: the client and the server.

Client

To ensure data received from the server is stored correctly on the client side, the properties have been tested. Besides the testing of properties, adding points to the chart is tested to ensure the correct data will be shown. Finally the notifications were tested by checking whether their maximum of displaying notifications would not be exceeded. This is tested to prevent possible memory overload.

The final test coverage of the client by amount of statements is 32%. This is a bit low because the client does not have much logic. It is a GUI and therefore it only visualizes data which is not relevant to test with unit tests.

Server

On the server side it is difficult to test whether the response of the devices will be handled correctly using unit tests because all communication works over the text-based data stream, rather than by calling some public methods that can be mocked. To test the working of the code, the device classes communicate with the devices over a stream wrapper interface, that separates the handling of the interface from the device communication class. Mocks, which are simulations of objects that mimics the behaviour in controlled ways, of the Agilent and Lakeshore were written that implement the stream interface and simulate the text response of the device based on the input they receive and a dictionary containing simulated sensor data. This process allowed the working of the device communication, sensors, calibration, and temperature control to be tested using only the publicly exposed methods and properties.

Other logic that was tested include the logging and commands in the API. Parts that are not unit tested are the state machine, which does not contain any complex logic, but rather a statement of outputs. This makes more sense to be tested with the actual cryostat in a system test. Also, parts that directly communicate with external interfaces such as USB, TCP or a library are not tested, as the external interfaces cannot be mocked. This resulted in a test coverage of the server by amount of statements of 40%.

5.2. System Testing

Since the program communicates with different devices it is wise to test what happens if connection is lost with one of the devices. What should happen is that the program gives a message of this and then proceeds to do what it was doing if that is still possible. What should not happen is that the program crashes and becomes unusable as long as the device is not connected. In order to test this the program was run on the computer in the lab and started with all devices connected. Then, one by one, all devices were unplugged to see how the program would react. When this was first tried, some errors arose. The server would throw an exception and would then proceed to shut down. After identifying and examining the exceptions the correct handling was implemented and the test was performed again. Eventually after several iterations of this process a loss of connection was handled correctly for each device.

5.3. Github

Besides sharing code, Github can be used as a helpful testing tool. Whenever a feature was finished a pull request would be opened to the dev branch. In order to merge the new feature in the dev it would need at least two approvals of team members in order to merge it. An approval would only be given after a thorough review of all new code, as well as to test whether its new function was working.

Since Visual Studio was being used as IDE, it was convenient to use their continuous integration tool: Visual Studio Team Services [30]. This tool was integrated within Github and would automatically check the code of each pull request to the dev to make sure all tests were working and to verify that the code would built. In order to merge the pull request, this check would have to pass. All pull requests in Github got two approvals and passed the continuous integration tool.

5.4. SIG Feedback

In order to achieve high code quality the intermediate code of the project was evaluated by the Software Improvement Group (SIG). This section will elaborate on the feedback that was given and secondly how this feedback was processed. SIG can help improving code quality by evaluating code based on a few criteria. The feedback that was given can be found in Appendix C.

Our code was given a score of 3.5 out of a maximum score of 5 which is above average. The main feedback involved Unit complexity, in particular in the state machine. This means that these parts of our code are more complex than average. In Section 4.2.7 it was explained why the state machine was implemented this way.

To reduce the unit complexity the communication in the client was refactored. Most of the client server communication went via the MainWindow class, which is the top level class of the main view and should only contain the logic for creating the four frames, as explained in Section 4.1.2. This logic does not include the client server communication. Therefore a static server instance is created, which can accessed by every class. Apart from the server instance a send method is created to run asynchronous calls to the server, meaning the GUI will keep on running if the server does not respond immediately. With this refactoring the lines of code in the MainWindow class were reduced by 150 lines of the 250 lines. This resulted in a helper class being deleted since communication calls are handled in the viewmodels.

Other methods with unnecessarily complexity, like the switch in the logger, have been refactored. The logger now makes use of a hashmap lookup as suggested.

5.5. Failure Mode and Effects Analysis

Since the cryostat is an expensive system and contains unique scientific experiments, risk of failure must be minimized. To that end, a failure mode and effects analysis (FMEA) has been performed. For each function of the application, a list of potential failures is composed, and ranked based on probability of failure, severity and ability to detect. This is expressed with a risk priority number (RPN). For every failure with a high risk (probability * severity * detection) a mitigation should be created which reduces the risk to an acceptable level. This analysis will be conducted using the standard template of an FMEA table.

After creating the FMEA table, that can be seen in Figures 5.1 and 5.2, all failures with a risk above 40 have been identified and a mitigation has been created.

- **Sensor defect**

  A sensor defect can cause heaters to overheat the cryostat. To mitigate this risk, a plausibility check should be performed before a cooldown starts, warning the user if sensors are outside their nominal range. This lowers the detection to a 1. This has not yet been implemented.

- **Heaters get too hot**

  To prevent heaters from overheating the system, hardcoded checks should be added that disable the heaters if the cryostat gets too hot. This reduces the probability to 0. This has been implemented in the code.

- **Calibration incorrect**

  This failure is similar to a sensor defect, and should be mitigated in the same way.

- **Temporary connection loss**

  If the TCP connection is interrupted for a short time, the connection will be broken, and the Agilent can no longer be controlled. Since the connection is likely to reconnect shortly if the failure is not terminal, the program should try to start reconnecting when the connection breaks. Reducing the severity to 1. This has been implemented in the code.

- **Complete connection loss**

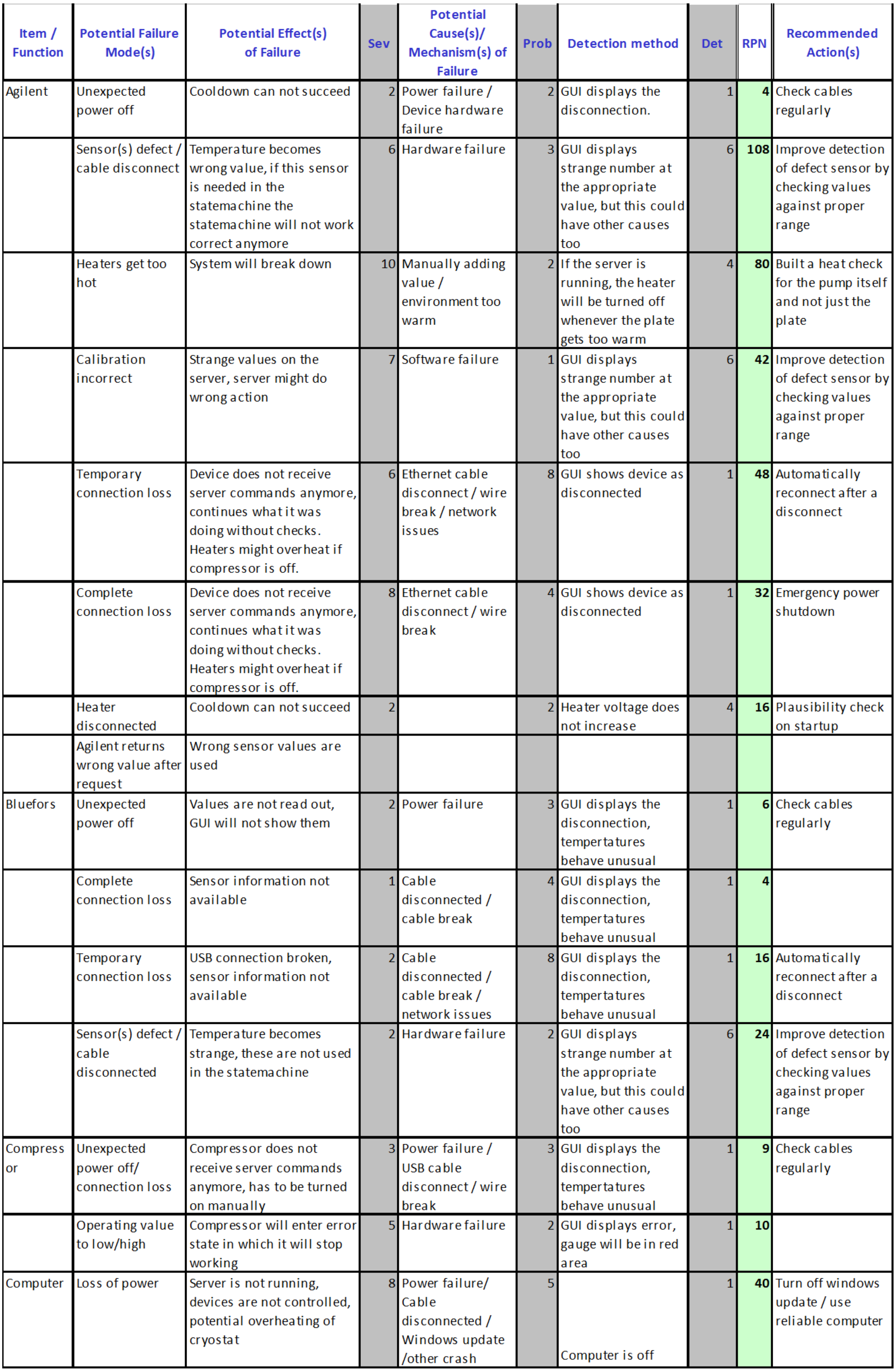
  If the connection is completely lost, the program can no longer affect the system. A potential mitigation would be to add a remote emergency power shutdown of the system.

- **Computer loss of power / crash**

  If the computer crashes while the heaters are active, they may overheat the system. To minimize this risk in the program, the controller should restart automatically when the computer restarts. This needs to be implemented at the deployment.

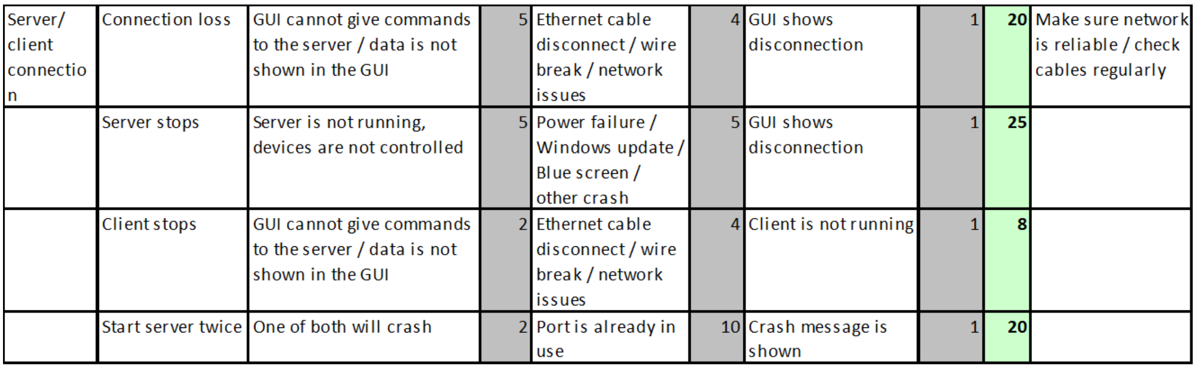
- **Lakeshore Heater**

  A failure with high risk is when the Lakeshore heater gets too hot. This is the only single point of failure where the system will break down and damage itself . Although everything had been done to prevent this from happening, it was decided, in agreement with the Astronomical Instrumentation group, that these heaters would not be controlled by the program. Operating the heaters will now have to be done manually on the device. This has no consequences on the cool down and recycle procedures, though the warm up will go slower.



Item / Function	Potential Failure Mode(s)	Potential Effect(s) of Failure	Sev	Potential Cause(s)/ Mechanism(s) of Failure	Prob	Detection method	Det	RPN	Recommended Action(s)
Agilent	Unexpected power off	Cooldown can not succeed	2	Power failure / Device hardware failure	2	GUI displays the disconnection.	1	4	Check cables regularly
	Sensor(s) defect / cable disconnect	Temperature becomes wrong value, if this sensor is needed in the statemachine the statemachine will not work correct anymore	6	Hardware failure	3	GUI displays strange number at the appropriate value, but this could have other causes too	6	108	Improve detection of defect sensor by checking values against proper range
	Heaters get too hot	System will break down	10	Manually adding value / environment too warm	2	If the server is running, the heater will be turned off whenever the plate gets too warm	4	80	Built a heat check for the pump itself and not just the plate
	Calibration incorrect	Strange values on the server, server might do wrong action	7	Software failure	1	GUI displays strange number at the appropriate value, but this could have other causes too	6	42	Improve detection of defect sensor by checking values against proper range
	Temporary connection loss	Device does not receive server commands anymore, continues what it was doing without checks. Heaters might overheat if compressor is off.	6	Ethernet cable disconnect / wire break / network issues	8	GUI shows device as disconnected	1	48	Automatically reconnect after a disconnect
	Complete connection loss	Device does not receive server commands anymore, continues what it was doing without checks. Heaters might overheat if compressor is off.	8	Ethernet cable disconnect / wire break	4	GUI shows device as disconnected	1	32	Emergency power shutdown
	Heater disconnected	Cooldown can not succeed	2		2	Heater voltage does not increase	4	16	Plausibility check on startup
	Agilent returns wrong value after request	Wrong sensor values are used							
Bluefors	Unexpected power off	Values are not read out, GUI will not show them	2	Power failure	3	GUI displays the disconnection, tempertatures behave unusual	1	6	Check cables regularly
	Complete connection loss	Sensor information not available	1	Cable disconnected / cable break	4	GUI displays the disconnection, tempertatures behave unusual	1	4	
	Temporary connection loss	USB connection broken, sensor information not available	2	Cable disconnected / cable break / network issues	8	GUI displays the disconnection, tempertatures behave unusual	1	16	Automatically reconnect after a disconnect
	Sensor(s) defect / cable disconnected	Temperature becomes strange, these are not used in the statemachine	2	Hardware failure	2	GUI displays strange number at the appropriate value, but this could have other causes too	6	24	Improve detection of defect sensor by checking values against proper range
Compressor	Unexpected power off/ connection loss	Compressor does not receive server commands anymore, has to be turned on manually	3	Power failure / USB cable disconnect / wire break	3	GUI displays the disconnection, tempertatures behave unusual	1	9	Check cables regularly
	Operating value to low/high	Compressor will enter error state in which it will stop working	5	Hardware failure	2	GUI displays error, gauge will be in red area	1	10	
Computer	Loss of power	Server is not running, devices are not controlled, potential overheating of cryostat	8	Power failure/ Cable disconnected / Windows update /other crash	5	Computer is off	1	40	Turn off windows update / use reliable computer

Figure 5.1: Failure mode analysis table.



Server/ client connection	Connection loss	GUI cannot give commands to the server / data is not shown in the GUI	5	Ethernet cable disconnect / wire break / network issues	4	GUI shows disconnection	1	**20**	Make sure network is reliable / check cables regularly
	Server stops	Server is not running, devices are not controlled	5	Power failure / Windows update / Blue screen / other crash	5	GUI shows disconnection	1	**25**	
	Client stops	GUI cannot give commands to the server / data is not shown in the GUI	2	Ethernet cable disconnect / wire break / network issues	4	Client is not running	1	**8**	
	Start server twice	One of both will crash	2	Port is already in use	10	Crash message is shown	1	**20**	

Figure 5.2: Failure mode analysis table.

6 Ethics

In this chapter the ethics that concern this project are discussed. When discussing the ethics of a computer program the main focus lies in the social impact of the program and whether the program can damage or violate the moral values and beliefs of a person or group as well as their safety. Since there is no personal data in this system the issue of privacy will be left out of scope.

An often discussed subject is the safety of the program. By hacking into the computer, could anything be damaged using the program? Therefore several safeguards have been created to prevent intrusion. First of all the communication with the server is secured using authentication, which makes it hard for outsiders to access it. Furthermore the server, which is responsible for controlling the devices, has built in safety checks that prevent the system from damaging itself (see Section 4.2.7).

Because the MOSAIC project is publicly funded, it was decided that the program should also be public. The program is licensed under the MIT licence. This license grants permission, free of charge, to any person obtaining a copy of this software to use, modify and redistribute. The only condition is that notice shall be included in all copies or substantial portions of the software. This could help later researches that also use a cryostat in their experiments. Furthermore in no event will be authors or copyright holders be liable for any claim, damages or other liability in the software.

Finally, in order to see stars at a large distance the Astronomical Instrumentation group uses a 250 GHz camera. Because there is much dust in space these stars would not be visible without a high frequency camera. Since this camera operates at a high frequency it is able to see through many objects. In theory, the research done for this project could be used to create a camera that violates people's privacy by seeing through clothes.

7
Evaluation and Recommendation

After developing the new program it is relevant to evaluate whether the program does what it needs to do by checking if all requirements are fulfilled and validating the automated processes. Next the client-server and GUI are discussed and some recommendations are made concerning these topics.

7.1. Requirements Analyzed

To verify that the program is working the requirements are evaluated. All must and should have's were implemented. The most important aspects will be stated:

- There is a button that can start the automated processes (discussed in the next section).
- In manual mode it is possible to turn the heaters in the He-7 cooler on or off, change their voltages and turn the compressor on or off.
- Logging has been implemented on the server side, one general log will always be active and another specific log can be started with user input.
- In the settings tab it is possible to configure the settings used in the automatic control cycles.
- To notify the user of important information, errors and warnings a notification panel has been created.
- Two charts have been implemented to visualize the process of the system.
- An external API has been made which is accessible by among others Python scripts.
- Multiple test methods have been used to ensure that the program is reliable and functions correctly.

7.2. Automated Cool Down

As the program should function properly without errors, many user tests have been carried out to validate the functionality of cooling down and recycling. In the first few tests the cooling down did not work properly because the pump heaters caused the heat switches to get too hot. This caused the heat generated at the pumps to conduct away, heating the system even more. This could be solved by changing the settings to reduce the pump heater power before the heat switches reached their conduction temperature. The final settings used in the program were established by doing multiple cool down sessions and discussions with the Astronomical Instrumentation group. Unfortunately because a cool down session can take over 12 hours, the amount of tests that could be done was limited. The current settings are still not perfect, but are able to cool the cryostat completely automated. (see Figure 7.1).

When the performance of the program is compared to the old application in Figure 7.2, the oscillations caused by the heat switches warming up become apparent. The performance does not seem to be significantly different using the new program. But there is clearly still room for improvement in the settings.

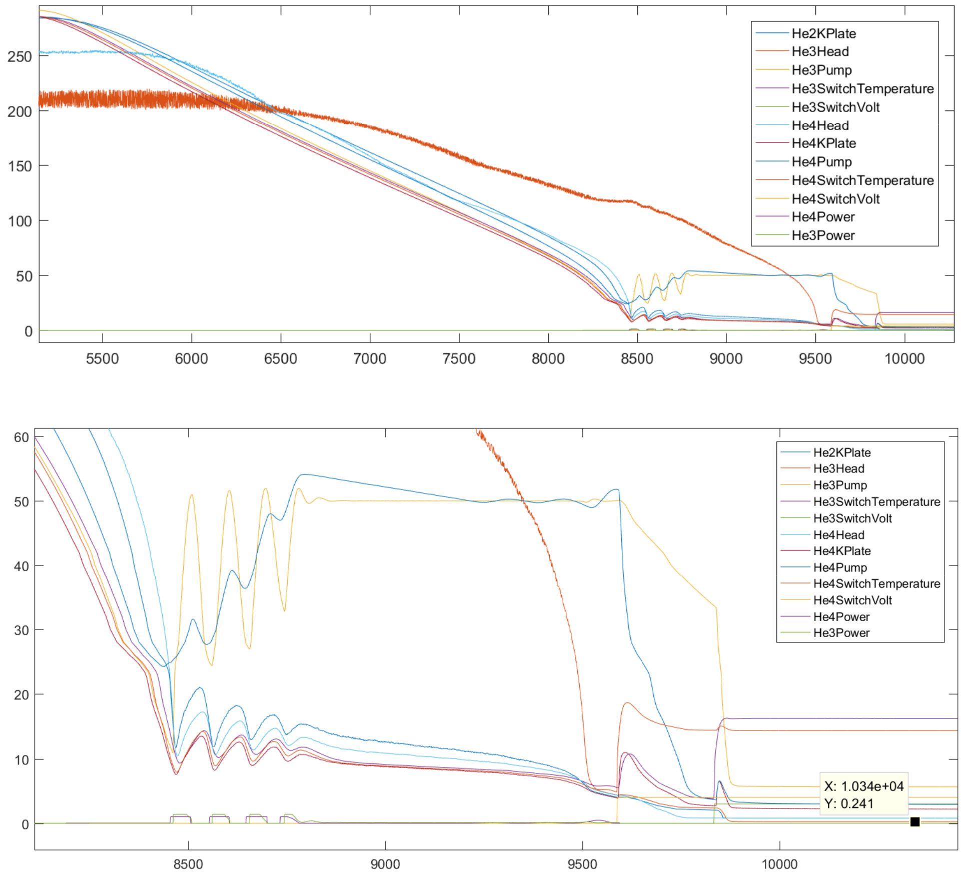


Figure 7.1: Graphs of successful cool down session. The first graph shows the complete cool down from room temperature. The second graph is zoomed in into the helium sorption steps where the program controls the He-7 cooler. At time step 8500 the pump heaters repeatedly turn off when the heat switches get too hot. The final temperature reached is 0.241K

7.3. Server Framework

The usage of Windows Communication Foundation (WCF) might be too complicated. The communication between the server and the native client could be easily implemented, but the communication with the Python scripts were harder. It occurred that the security layer could not be implemented via the server and Python libraries who could use the defined services. The solution was to make a REST API for Python calls which can also be used by other clients. The native client could also use the created REST API, meaning that the implementation of the service contracts is work which could be avoided. The only advantage of this implementation is the extra security level. So the usage of ASP.NET, the lightweight framework for REST, could be an improvement as the advantages of WCF will not be reflected, but is not necessarily better. Apart from the choice of the framework the client server connection could not be used apart from the local machine. This is due to some advanced network settings which needs to be set but cannot be set as those cannot be accessed.

7.4. Graphical User Interface

A web page as alternative GUI provides the extendability of opening a browser on any operating system which displays the same view. In Section 2.2 it was also proposed as potential client server concept, but due to the fact that a native client was preferred and the time for the project was limited, a web page was not created. An addition to the program would be to implement a web page, as it provides more accessibility than a single native client. This web page could eventually replace the current native client.

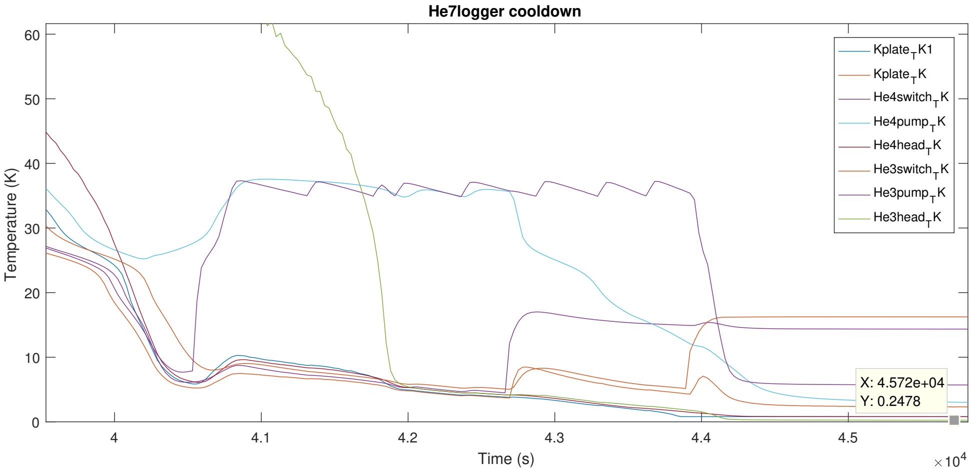


Figure 7.2: Graph of cool down using the old program.

7.5. Recommendations

Concluding from the evaluation, there are some recommendations to further improve upon this project.

Since there was limited time and the cool down process took a long time the amount of tests with the system were limited. The final program was able to cool the cryostat down fully automated. Though there were still points of improvement in the final cool down. Specifically the repeated turn off of the pump heaters due to the heat switches getting too hot. A possible solution might be to reduce the pump heater power, or lower the start temperature. Both can be changed by settings in the GUI. Solving this issue would lead to a quicker cool down.

Due to the limited access to the network system the client server could not be used over the network. A solution can be to create a private network with the needed settings. Downside is that the server and client computer should be connected with that network. Another solution, which was already discussed in Section 7.4, is to create a web page. Running a web page the communication goes through port 80 which should be opened in most networks.

8

Conclusion

To assist the experiments of the MOSAIC project a new control system for cooling the temperature of a cryostat below 250mK has been developed. The cryostat can remain cold for 24 hours before the system needs to be recycled. The previous control system was not able to get to this temperature without some manual intervention. The newly developed program does not need manual intervention and is therefore able to operate fully automatic.

The program is divided in two parts: a server and a client. The server is the core of the program and conducts all logic like cooling down and recycling the cryostat. The developed graphical user interface, the native client, is a Windows application, which enables full control of the cryostat. It displays temperatures and pressures of components, can start a cool down or recycle with a single buttons press, or can change and apply certain settings manually. Apart from the Windows application, the server offers a REST API which allows the user to communicate using Python scripts. The user can set and get specified data to or from the server without the need of using a GUI. By doing tests and validating the system the stability of the software has been confirmed. To conclude, the newly developed system is an improvement on the previous control software in terms of controllability, maintainability and extendability.

Bibliography

[1] R. Bhatia et al., *A three-stage helium sorption refrigerator for cooling of infrared detectors to 280 mk,* Cryogenics **40**, 685 (2000).

[2] K. Schwaber, *Scrum development process,* in *Business Object Design and Implementation* (Springer, 1997) pp. 117–134.

[3] G. White and S. Collocott, *Heat capacity of reference materials: Cu and w,* Journal of Physical and Chemical Reference Data **13**, 1251 (1984).

[4] Church, *Moore Edward gedanken-experiments on sequential machines. automata studies, edited by shannon c. e. and mccarthy j., annals of mathematics studies no. 34, litho-printed, princeton university press, princeton 1956, pp. 129–153,* **23**.

[5] G. Mealy, *A method for synthesizing sequential circuits,* The Bell System Technical Journal **34**, 1045 (1955).

[6] G. Franklin et al., *Feedback control of dynamic systems*, Vol. 3 (Addison-Wesley Reading, MA, 1994).

[7] M. Santos et al., *Temperature control in liquid helium cryostat using self-learning neurofuzzy controller,* IEE Proceedings-Control Theory and Applications **148**, 233 (2001).

[8] W. Tan and A. Dexter, *Self-learning neurofuzzy control of a liquid helium cryostat,* Control Engineering Practice **7**, 1209 (1999).

[9] M. Santos et al., *Control of a cryogenic process using a fuzzy pid scheduler,* Control Engineering Practice **10**, 1147 (2002).

[10] *Lakeshore 335 datasheet,* http://www.lakeshore.com/Documents/335.pdf, accessed July 6, 2017.

[11] *Agilent 34972a user manual,* literature.cdn.keysight.com/litweb/pdf/34972-90010.pdf, accessed July 6, 2017.

[12] Cryomech, *Cp2800 and cp1000 series compressor,* http://www.cryomech.com/research/cp2800-cp1000-series-compressor/, accessed July 6, 2017.

[13] HowStuffWorks, *How usb ports work,* http://computer.howstuffworks.com/usb4.htm (2000), accessed July 6, 2017.

[14] Dataq, *Usb or ethernet: Which do i choose?*https://www.dataq.com/data-acquisition/general-education-tutorials/usb-or-ethernet-which-do-i-choose.html, accessed July 6, 2017.

[15] A. Tanenbaum and D. Wetherall, *Computer networks* (Pearson Education, 2014) Chap. 4, pp. 282–285.

[16] Microsoft, *Documentation for the sockets namespace,* https://msdn.microsoft.com/en-us/library/system.net.sockets(v=vs.110).aspx (), accessed July 6, 2017.

[17] Microsoft, *Documentation for the websocket namespace,* https://msdn.microsoft.com/en-us/library/system.net.websockets(v=vs.110).aspx (), accessed July 6, 2017.

[18] Microsoft, *Asp.net homepage,* https://www.asp.net/ (2017), accessed July 6, 2017.

[19] Microsoft, *What is windows communication foundation,* https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf (2017), accessed July 6, 2017.

[20] D. Roth et al., *Asp.net core homepage,* https://docs.microsoft.com/en-us/aspnet/core/ (2016), accessed July 6, 2017.

[21] E. Reitan et al., *Wcf and asp.net web api,* https://docs.microsoft.com/en-us/dotnet/framework/wcf/wcf-and-aspnet-web-api (2017), accessed July 6, 2017.

[22] T. Sneed, *Wcf is dead and web api is dying – long live mvc 6!* https://blog.tonysneed.com/2016/01/06/wcf-is-dead-long-live-mvc-6/ (2016), accessed July 6, 2017.

[23] R. Schiefer, *Wcf, "i'm not dead yet"!* http://www.dotnetcatch.com/2016/08/12/wcf-im-not-dead-yet/ (2016), accessed July 6, 2017.

[24] *Windows presentation foundation (wpf),* https://docs.microsoft.com/en-us/dotnet/framework/wpf/index, accessed July 6, 2017.

[25] E. Sorensen et al., *Model-view-viewmodel (mvvm) design pattern using windows presentation foundation (wpf) technology,* MegaByte Journal **9**, 1 (2010).

[26] *The mvvm pattern,* https://msdn.microsoft.com/en-us/library/hh848246.aspx, accessed July 6, 2017.

[27] *Live charts,* https://lvcharts.net/, accessed July 6, 2017.

[28] D. Turin, *Nmodbus4 github,* https://github.com/NModbus4/NModbus4, accessed July 6, 2017.

[29] G. et. al., *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader)* (Pearson Education, 1994).

[30] *Visual studio team services,* https://www.visualstudio.com/team-services/, accessed July 6, 2017.

Appendices

A
Infosheet

Real time control for a cryognic refrigerator
Client organization: Astronomical Instrumentation group in collaboration with SRON
Github: https://github.com/BBekker/CryostatControl
Presentation Date: 5 July 2017
Description:

In order to measure the spectrum of radio emissions from galaxies and other deep space objects, a new superconducting spectrometer, working at very cold temperatures close to the absolute zero, is developed. An advanced cooling system called a cryostat is used to cool down the spectrometer. The cool down of the cryostat involves the control of multiple sensors and actuators connected to the cryostat to achieve a final temperature below 250 millikelvin. A software program is used for this purpose. As extra hardware components have been added to the cryostat, the existing program does no longer fulfil the requirements. For this reason a new software program, which can monitor temperatures of all components and start control processes, is developed. The developed program consists of a client server structure. The server handles the logic of the cryostat using several controllers. It can send data to a native client, which is the graphical user interface, or a REST API. The native client displays sensor readouts received from the server and allows full control of server, which means it can start the cool down process as well as manual control processes. The REST API allows the user to have full control over the server using a Python script to achieve measurements which cannot be done from the native client. The increased automation, improved control and ability to integrate with external Python scripts allow the user to focus on the essential parts of an experiment making the developed program an improvement over the previous program.

Members of the project:
B. Bekker (B.Bekker@student.tudelft.nl)
M. Goudriaan (M.Goudriaan-2@student.tudelft.nl)
R.D. Meeuwissen (R.D.Meeuwissen@student.tudelft.nl)
L.H.Sikkes (L.H.Sikkes@student.tudelft.nl)

Name and affiliation of the client:
Dr. J. Bueno, SRON (J.Bueno@sron.nl)
S. Hähnle, SRON (S.Haehnle@sron.nl)

Name and affiliation of the project supervisor:
Dr. R. Krebbers, TU Delft (mail@robbertkrebbers.nl)

The final report for this project can be found on: https://repository.tudelft.nl

B

Project Description

Project Description

The Astronomical Instrumentation group is setting up a new laboratory to develop a new redshift survey instrument to measure the distance or age of galaxies that are aggressively growing in the early Universe. The instrument is fully based on novel superconducting circuits, which therefore needs to be cooled down to cryogenic temperatures.

We have a cryostat that allows us to reach temperatures around 0.25 K (-273 C). It consists of a sorption cooler, a pulse tube cooler and a compressor, which need to be controlled and operated remotely. The sorption cooler currently works with a C-based software; the pulse tube cooler runs with proprietary software based on Java; and the compressor is currently operated in manual mode.

The task of the student(s) is to make a single real time control unit that can be used to control and operate all aspects of the cryostat. The control unit will integrate data from the different parts of the cryostat, which will have to be processed efficiently in real time. The operation of the control unit will be based on a decision tree that the student(s) will have to develop. The final real time control unit will need to have the following functionality implemented to it:

1. Control modes:
   - Cool down from room temperature.
   - Recharge the sorption cooler.
   - Warm up.
2. Active:
   - Set the heater output of the sorption cooler.
   - Compressor control.
   - Start control modes for the sorption cooler.
3. Passive:
   - Temperature readings of the sorption cooler.
   - Temperature readings of the pulse tube cooler.
   - Heater output reading of the sorption cooler.
   - Status and pressure readings of the compressor.
   - Data logging.
4. Independent scripting access to basic functionality:
   - Access to temperature readings from an external script based on python.
   - Access to heaters settings from an external script based on python.
5. A Graphical User Interface for the operation of the real time control unit.

Company Description

SRON, the Netherlands Institute for Space Research, develops superconducting technology, typically working at a temperature of 0.1 K (-273 C), that will lead to astronomical instruments with the ultimate sensitivity.

Currently SRON is developing a new redshift survey instrument to measure the distance or age of these galaxies, using recent progress in superconducting nanotechnology, which can spectrally resolve a large fraction of the cosmic infrared background from the ground. The instrument is a Multi-Object Spectrometer with an Array of superconducting Integrated Circuits (MOSAIC), fully based on novel superconducting circuits. MOSAIC will be installed on the 10m Japanese ASTE observatory, a world-class astronomical facility based at the Atacama desert in Chile at 5000m altitude.

C

SIG Feedback

”De code van het systeem scoort 3,5 ster op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door een lagere score voor Unit Complexity.

Voor Unit Complexity wordt er gekeken naar het percentage code dat bovengemiddeld complex is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, makkelijker te testen is en daardoor eenvoudiger te onderhouden wordt.

Complexiteit ontstaat vaak doordat te veel verantwoordelijkheid op één centrale plek samenkomt. In jullie project is Controller.StateMachine daar een voorbeeld van. Deze methode is min of meer het hart van het systeem. Dat wordt een probleem op het moment dat de hoeveelheid functionaliteit blijft groeien. Het is dan beter om het gedrag van elke state van het state machine mechanisme te scheiden.

Daarnaast is code soms onnodig complex. De methode AbstractDataLogger.GetDeviceName doet bijvoorbeeld eigenlijk een map lookup, waarbij de keys van de map het datanummer zijn en de values de naam van het apparaat. Door de methode anders op te schrijven wordt de complexiteit van het grote switch-statement vermeden.

De aanwezigheid van test-code is in ieder geval veelbelovend, hopelijk zal het volume van de test-code ook groeien op het moment dat er nieuwe functionaliteit toegevoegd wordt.

Over het algemeen scoort de code bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.”

D

REST API Documentation

This documentation can be found on the GitHub page of the program: https://github.com/BBekker/CryostatControl/wiki/REST-API-documentation.

How To Use

All data requests should use a GET command and all write requests should use a POST command. All requests should be authenticated using HTTP Basic authentication.

Python

Minimal GET example

```
>>> import requests
>>> r = requests.get('https://localhost:18081/service/ReadSettings', auth=('cooler','ChangeMe!'),
verify=False)
C:/Users/Bernard/VE/cryostat/lib/site-packages/urllib3/connectionpool.py:852:
InsecureRequestWarning: Unverified HTTPS request is being made. Adding certificate
verification is strongly advised. See:
https://urllib3.readthedocs.io/en/latest/advanced-usage.html#ssl-warningsInsecureRequestWarning)
>>> r.json()
[1.4, 2, 3, 4, 70, 50, 14, 12, 50, 4.4, 2, 20, 2.5, 30, 0.05]
```

Minimal POST example

```
>>> r = requests.post('https://localhost:18081/service/Cooldown', auth=('cooler','ChangeMe!'),
verify=False)
```

POST with json body:

```
>>> r = requests.post('https://localhost:18081/service/WriteSettingValue', json={'setting':1,
'value': 2.4}, auth=('cooler', 'ChangeMe!'), verify=False)
```

Requests are made over a secure connection, but if the certificate is not signed with a CA authorety, requests will throw errors when connecting. Either set verify=False to disable the check, or make sure that the server certificate is trusted by the pc. Sessions can be used to set the authentication and verify only once.

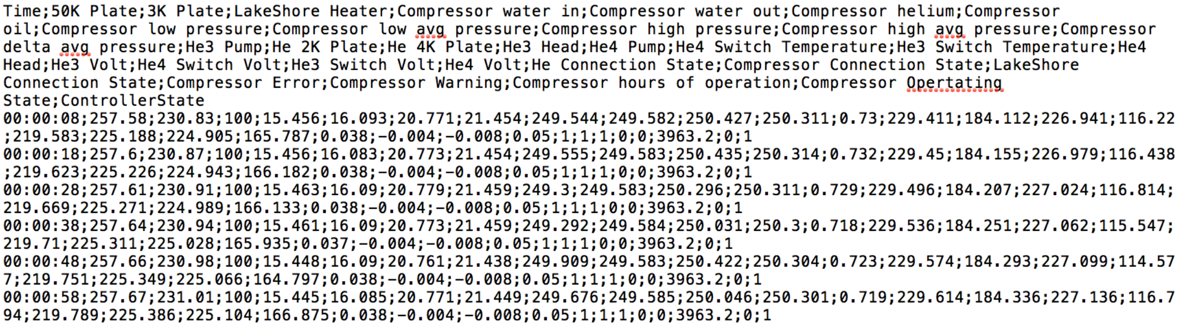
List of Commands

The REST commands are defined in:
https://github.com/BBekker/CryostatControl/blob/dev/CryostatControlServer/HostService ICommandService.cs

- `HTTPS://<address>:18081/service/` Main address
  - `GET IsAlive` Check if server is alive, returns true
  - `POST Cooldown` Start a cool down now
  - `POST CooldownTime` Start the cooldown at a time set in the request body. formatted in microsofts own format: `Date(<miliseconds since 1970-01-01 00:00>)`
  - `POST Recycle`
  - `POST RecycleTime`
  - `POST Warmup`
  - `POST WarmupTime`
  - `POST Manual`
  - `POST Cancel`
  - `GET GetState`
  - `GET GetStartTime`
  - `POST SetCompressorState`
  - `POST WriteHelium7`
    Write a value to a heater. Body must be of format: { ‘heater’:`<id>` , ‘value’:`<value in volts>`}
  - `GET value/<sensor id>` Read a sensor.
  - `GET ReadSettings` Returns an array of all settings ordered according to SettingEnumerator
  - `POST WriteSettingValue` Change a setting on the server, body must be of the format: { ‘setting’:`<setting id>`, ‘value’:`<new value`}
  - `POST StartLogging` { ‘interval’, ‘<integer in

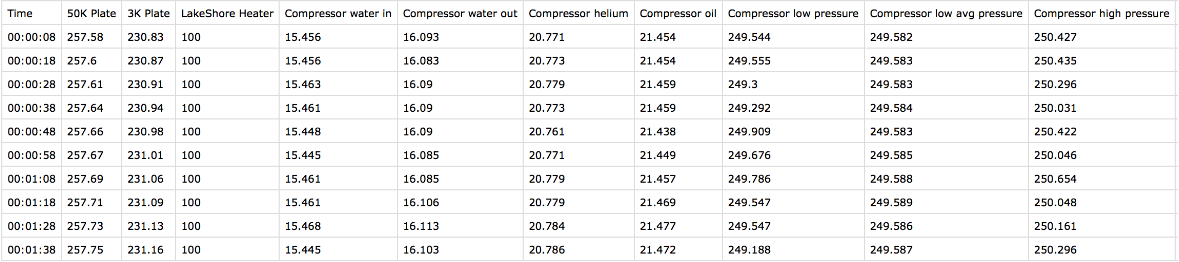
E

Logging Example



```
Time;50K Plate;3K Plate;LakeShore Heater;Compressor water in;Compressor water out;Compressor helium;Compressor
oil;Compressor low pressure;Compressor low avg pressure;Compressor high pressure;Compressor high avg pressure;Compressor
delta avg pressure;He3 Pump;He 2K Plate;He 4K Plate;He3 Head;He4 Pump;He4 Switch Temperature;He3 Switch Temperature;He4
Head;He3 Volt;He4 Switch Volt;He3 Switch Volt;He4 Volt;He Connection State;Compressor Connection State;LakeShore
Connection State;Compressor Error;Compressor Warning;Compressor hours of operation;Compressor Opertating
State;ControllerState
00:00:08;257.58;230.83;100;15.456;16.093;20.771;21.454;249.544;249.582;250.427;250.311;0.73;229.411;184.112;226.941;116.22
;219.583;225.188;224.905;165.787;0.038;-0.004;-0.008;0.05;1;1;1;0;0;3963.2;0;1
00:00:18;257.6;230.87;100;15.456;16.083;20.773;21.454;249.555;249.583;250.435;250.314;0.732;229.45;184.155;226.979;116.438
;219.623;225.226;224.943;166.182;0.038;-0.004;-0.008;0.05;1;1;1;0;0;3963.2;0;1
00:00:28;257.61;230.91;100;15.463;16.09;20.779;21.459;249.3;249.583;250.296;250.311;0.729;229.496;184.207;227.024;116.814;
219.669;225.271;224.989;166.133;0.038;-0.004;-0.008;0.05;1;1;1;0;0;3963.2;0;1
00:00:38;257.64;230.94;100;15.461;16.09;20.773;21.459;249.292;249.584;250.031;250.3;0.718;229.536;184.251;227.062;115.547;
219.71;225.311;225.028;165.935;0.037;-0.004;-0.008;0.05;1;1;1;0;0;3963.2;0;1
00:00:48;257.66;230.98;100;15.448;16.09;20.761;21.438;249.909;249.583;250.422;250.304;0.723;229.574;184.293;227.099;114.57
7;219.751;225.349;225.066;164.797;0.038;-0.004;-0.008;0.05;1;1;1;0;0;3963.2;0;1
00:00:58;257.67;231.01;100;15.445;16.085;20.771;21.449;249.676;249.585;250.046;250.301;0.719;229.614;184.336;227.136;116.7
94;219.789;225.386;225.104;166.875;0.038;-0.004;-0.008;0.05;1;1;1;0;0;3963.2;0;1
```

Figure E.1: Example of logging file separated with semicolons opened with a text reader.



Time	50K Plate	3K Plate	LakeShore Heater	Compressor water in	Compressor water out	Compressor helium	Compressor oil	Compressor low pressure	Compressor low avg pressure	Compressor high pressure
00:00:08	257.58	230.83	100	15.456	16.093	20.771	21.454	249.544	249.582	250.427
00:00:18	257.6	230.87	100	15.456	16.083	20.773	21.454	249.555	249.583	250.435
00:00:28	257.61	230.91	100	15.463	16.09	20.779	21.459	249.3	249.583	250.296
00:00:38	257.64	230.94	100	15.461	16.09	20.773	21.459	249.292	249.584	250.031
00:00:48	257.66	230.98	100	15.448	16.09	20.761	21.438	249.909	249.583	250.422
00:00:58	257.67	231.01	100	15.445	16.085	20.771	21.449	249.676	249.585	250.046
00:01:08	257.69	231.06	100	15.461	16.085	20.779	21.457	249.786	249.588	250.654
00:01:18	257.71	231.09	100	15.461	16.106	20.779	21.469	249.547	249.589	250.048
00:01:28	257.73	231.13	100	15.468	16.113	20.784	21.477	249.547	249.586	250.161
00:01:38	257.75	231.16	100	15.445	16.103	20.786	21.472	249.188	249.587	250.296

Figure E.2: Example of logging file opened with a spreadsheet program.

F

Data Enumerator

Sensors	
LakeShore (Radiation shields)	
0	50k radiation shield
1	3k radiation shield
Compressor	
2	Water in temperature
3	Water out temperature
4	Helium temperature
5	Oil temperature
6	Low pressure
7	Low pressure average
8	High pressure
9	High pressure average
10	Delta pressure average
Agilent (Helium-7 cooler)	
11	Helium 3 pump temperature
12	Helium 3 head temperature
13	Helium 3 switch temperature
14	Helium 4 pump temperature
15	Helium 4 head temperature
16	Helium 4 switch temperature
17	Helium 2K plate
18	Helium 4K plate
19	Helium 3 pump voltage
20	Helium 3 switch voltage
21	Helium 4 pump voltage
22	Helium 4 switch voltage
Miscellaneous	
Connection states	
23	Agilent connection state
24	Compressor connection state
25	LakeShore connection state
Compressor miscellaneous	
26	Error
27	warming state
28	hours of operation
29	Operation state

Table F.1: Overview of the data that is retrieved by the server from the components of the cryostat. This data is accessible for the clients with the given ID of the value.