IMMERSION COOLED ENVIRONMENTAL MONITORING AND PREDICTION SYSTEM FOR THE MEERKAT IMAGER

by

Apiwe Hotele

BSc Computer Science (Hons)

A thesis submitted to the Department of Electrical Engineering,

The university of Cape Town, in fulfillment of the requirements

for the degree of

Master of Science

at the

UNIVERSITY OF CAPE TOWN

Supervisor:

Dr. Simon Winberg

© University of Cape Town

December 2017

University of Cape Town

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This thesis has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed by candidate

Signature of the Author

Cape Town

31 August 2017

Dedication

I dedicate this dissertation to my daughter

Abstract

This paper reports on an immersion cooling method used to reduce power consumption in a data centre. This study involves a case study of the MeerKAT Science Processor that is responsible for the MeerKAT imaging pipeline. Immersion cooling brings a coolant into direct physical contact with the chips and the circuit board by directly immersing computing equipment into a bath of cooling fluid. According to the National Security Agency's Laboratory for Physical Sciences (LPS), who acquired and installed an oil-immersion cooling system in 2012, the use of immersion cooling means that much of the infrastructure needed for cooling a data centre can be eliminated; it can moreover reduce server failures, and is cleaner and quieter than air cooling. In this study, an oil cooled environment is created and a prototype low-cost thermal management system for the system is built and tested. This prototyped system was called the Environmental Monitoring System (EMS), and it monitors humidity and temperature of the oil-cooled environment. In this study, discrete temperature data gathered by the thermal management system is used to build a prediction program that we called the Immersion Cooling Temperature Predictor (ICTP), which predicts temperature at locations not covered by sensors. The ICTP predicted measurements using a Gaussian process model, providing estimates for non-sampled locations to help make the monitoring systems more fault tolerant. Reducing the number of sensor nodes moreover reduces installation costs, as well as space utilized and power consumed by temperature sensors. The accuracy of detecting hotspots in an immersion cooled environment using the system is also investigated. From the experiments it was found that the ICTP had a mean error of 0, 0083, standard deviation of 2, 56 and predicted standard deviation of 2, 44 for predicting hotspots.

Acknowledgements

As with any major project, it takes a great team to make all the elements come together. I Thank **God** for choosing the following team members:

Dr Simon Winberg- *"my supervisor"*-without your constant meetings, constructive criticism, encouragement, belief in me and motivation this project wouldn't be successful.

The Science Data Processor (SDP) Team - Thank you for your advice and input you made in this project.

Khutso Ngoasheng *"my manager"*- Thank you for the constant encouragement and belief. Thank you for giving me time to work on my disseratation, you made working and studying achievable.

The Square Kilometre Array - South Africa (SKA SA) Thank you for the risk you took by sponsoring a young girl that you did not know, but believed in her and gave her a chance to be something in life.

Dr Ludwig -Thank you for your support and helping me understand certain concepts. Thank you for your patience, I will forever be grateful.

Christopher Scholar*"my mentor"*-Thank you for lending a shoulder when times were tough. When the boat was sinking, you always had the necessary tools to fix it. Thank You.

My Siblings: Thembakazi, Yolanda, Xabiso, Isasithanda and my nephew Simvuyele- Thank you for the support, love, encouragement and motivation. Blood really is thicker than water.

Yimigcobo Hotele *"my daughter"*- Thank you for giving me hope and always looking up to me.

And most of all Zukiswa and Lungile Hotele-*"my parents"*- I do not know where to begin when thanking you, for I know no one will ever love me the way you guys do. The two people who sacrifice many things for me to be educated. Thank you for the everyday phone calls with encouragement and motivation. I am truly grateful. I love you.

Nomenclature

CAM	Control and Monitoring
CBF	Correlator Beam Former
CFD	Computational Fluid Dynamics
CMOS IC	Complementary Metal-Oxide Semiconductor Integrated Circuit
CNC	Computer Numerical Control
CPU	Central Processing Unit
CRAC	Computer Room Air Conditioning
DDC	Direct Digital Control
EMS	Environmental Monitoring System
FFT	Fast Fourier Transform
GP	Gaussian Process
GPR	Gaussian Process Regression
GPU	Graphics Processing Unit
GRC	Green Revolution Cooling
HDF5	Highly Definable File version 5
HPC	High Performance Computing
ICTP	Immersion Cooling Temperature Predictor
I2C	Inter-Integrated Circuit
KATCP	Karoo Array Telescope Control Protocol

LOFAR	Low Frequency Array
NRF	National Research Foundation
PCB	Printed Circuit Board
PUE	Power Usage Efficiency
PWM	Pulse Width Modulation
RAM	Random Access Memory
RFI	Radio Frequency Interference
RH	Relative Humidity
Rpi	Raspberry Pi
SDP	Science Data Processor
SE	Systems Engineering
SIP	Single In-line Package
SKA	Square Kilometre Array
SKA SA	Square Kilometre Array South Africa
ssh	Secure shell
TiB	Tebibyte
UML	Unified Modelling Language
UV	Ultra Violet

Table of Contents

List of Figures

List of Tables

1 Introduction

The aim of this study is to investigate immersion cooling as a viable solution for cooling computers and conducting monitoring in an immersion-cooled environment, using the MeerKAT radio telescope as a case study.

In 2012, South Africa and Australia jointly won the bid to host the Square Kilometre Array (SKA) radio telescope. The Square Kilometre Array South Africa (SKA SA) is a company founded by the South African National Research Foundation (NRF) to design and construct a precursor instrument called the MeerKAT telescope outside Carnarvon, the South African site of the planned SKA telescope[1]. Carnarvon is a small town in the Northern Cape. MeerKAT, initially was developed as a demonstrator system to show South Africa's commitment and interest in hosting the SKA is instead going to be integrated into the SKA Phase 1. The SKA is anticipated to be more powerful than all currently existing radio telescopes in the world [2]. The telescope will moreover have huge data rates, with a concomitant increase in data volumes and an increase in processing requirements [3].

The imager, which is the object of this study, is a component of the data processing pipeline of the telescope, where the sampled data from the telescopes is placed on Ultra Violet (UV) grid. The grid is Fourier transformed to convert the UV-image to a sky image, in other words, an image of a measured region in the sky [4]. This process is computationally expensive with substantial processing requirements. In the Low Frequency Array (LOFAR), situated in the province of Drenthe in the northern region of The Netherlands, for instance, almost all post-observation processing time is spent on creating sky images [5]. The SKA is being built in two phases, where Phase 1 represents about 10% of the capacity of the whole telescope [6]. Australia will host the low frequency telescopes with more than 500 stations each containing 250 individual antennas, whereas South Africa will host an array of 200 dishes including the 64-dish MeerKAT precursor telescope [7]. The estimated power required for the image processing of the SKA Phase 1 is in the petaflop range, details on the calculations are available at [6]. High Performance Computing (HPC) clusters are needed to process the high volumes of data generated by radio telescopes. Accurate estimations of SKA power demand will be an outcome of detailed design and prototyping undertaken during the project execution phase, however

simplified calculations have produced demand estimates of the order of 100 megawatts (MW), split equally between the array proper and the associated HPC centre [8]. The estimates assume a significant demand in innovative strategies to minimise power in either of the following: receptors, digital signal processing, and computing and environmental conditioning particularly cooling. Cooling for electronic systems is a major concern at candidate SKA sites; MeerKAT and the Australian SKA Pathfinder (ASKAP) are studying interesting new approaches to overcome this challenge. MeerKAT is demonstrating evaporative pre-cooling to increase significantly the coefficient of performance of some of the refrigerate coolers. In Western Australia, Commonwealth Scientific and Industrial Research Organisational (CSIRO) and its partners are investigating a novel geothermal cooling solution for the Pawsey Centre for SKA HPC, an approach likely to reduce data centre operation costs greatly [8]. The SKA SA is investigating improved methods of storing and processing data and especially for more energy efficient cooling techniques for the MeerKAT telescope. While the power challenge is a large measure, the radio telescope imposes special considerations. Many cost-effective options will be rejected because of the difficulty of meeting stringent Radio Frequency Interference (RFI) and derived electromagnetic compatibility (EMC) requirements, making the data centre at the MeerKAT site even more complicated than other data centres.

The focus of this thesis is the design of an efficient cooling and monitoring system for the imager; this is a subsystem of the science data processor (SDP), which in turn is a subsystem of the MeerKAT data centre. This study thus focuses on three main aspects:

1. Immersion cooling as a viable technique, in which coolant is brought into direct physical contact with the electronic chips to reduce power consumption in a data centre. A case study of this method is thus investigated with regard to the MeerKAT science processor, which is part of the MeerKAT imager.
2. A prototyped low-cost thermal management system called the Environmental Monitoring System (EMS) for the oil-cooled system. The EMS monitors humidity and temperature and stores the data in a file that is easily accessible to the team.
3. The Immersion Cooling Temperature Predictor (ICTP), a model that predicts the temperature at locations that are not covered by sensors. The discrete temperature data gathered by the thermal management system is used to create a model for the ICTP. The

ICTP thus not only presents information about non-sampled areas, but it is also fault tolerant because it can handle missing nodes and can be used to build a hotspot detector for any system. Guidelines are thus presented in this dissertation of how to build a hotspot detector for an immersion-cooling detector. Reducing the number of sensor nodes reduces the installation costs, the space utilized and the power consumed by the temperature monitoring system.

The following sections of this chapter present the motivation behind of the study followed by the summarised research focus, which then leads to research questions, which guide the investigative aspect of this project. The important terminology relating to the area of focus, together with definitions of specialized terms and acronyms that are used in this dissertation follow. In addition the limitations of this project are discussed. Finally, this chapter ends with an overview of the dissertation, explaining how the chapters are structured and briefly summarising each chapter.

1.1 Motivation

The SDP team is looking into minimizing the cooling requirements in order to reduce costs in their data centre, while nonetheless improving overall energy efficiency. They would like to achieve this by using immersion cooling as a cooling method instead of using air cooling. The challenge is that they have never implemented an immersion-cooled environment and thus do not know what the implications of implementing such a system would be. They want to know how immersion cooling works and what needs to be considered when implementing such a system. The team is also concerned about how they could identify and prevent hotspots in such an immersion-cooled environment, to prevent electronics from being damaged. They are also looking into building a thermal management system that can monitor this immersion-cooled environment by means of sensor nodes, which are constantly gathering information about the state of the data centre. They also want to reduce the cost of the sensor nodes they deploy by reducing the number of sensor nodes; this can be done by implementing a model that is able to predict values in locations where there is no physical sensor.

1.2 Contribution

In this case study, an immersion cooling environment is implemented at a small scale at the end of the thesis and recommendations and considerations are given with regard to such an implementation. Moreover, a prototype of an EMS is implemented; this system monitors the temperature and humidity of the immersion-cooled environment and stores the data in a file that is easily accessible to the team. In addition, a model that predicts the temperature in locations where there is no temperature sensor known as Immersion Cooling Temperature Predictor (ICTP) is implemented, using Gaussian Process Regression (GPR). The EMS and the ICTP are not specific to the immersion-cooled environment; they can be used in other scenarios too; however, in this thesis they are only used in this specific environment. The ICTP can be used to build a hotspot detector. This will not only allow the SDP to implement a hotspot detector for the MeerKAT array, but it can potentially be used by anyone who is interested in implementing a hotspot detector for an immersion cooling environment.

1.3 Research Focus

The project focused on the following:

- Implementing an immersion-cooled environment at a small scale and analysing the advantages and disadvantages of implementing such a system,
- Designing and implementing a prototype of EMS,
- Constructing a model that predicts temperature in locations where there is no temperature sensor.

1.4 Research Questions

Given the above-mentioned research focus, the following questions have been developed in order to gain insight into the problems concerned and in particular how to achieve the planned objectives.

The system is thus developed with the following main research questions in mind:

- Question 1: What implications should be considered when implementing an immersion-cooled environment?
- Question 2: Can the EMS be used to ensure sufficient monitoring?
- Question 3: Can a model be implemented to predict temperature in locations where there is no physical sensor?

1.5 Important Terminology

This section explains the important terminology used in this project.

Table 1-1: Terminology

Term	**Meaning**
Radio Telescope	An astronomical instrument used to detect radio-frequency radiation emitted by extraterrestrial sources.
Imager	A group of electronic components, such as servers, switches and etc.
Science Data Processor (SDP)	A system that analyses and stores the data from the telescope.
Immersion Cooling	A process where electronic components are submerged in a special kind of liquid that is not conductive.
MeerKAT	A radio telescope with 64 antennas.
EMS	A system that monitors the temperature and humidity for an immersion cooled environment.
ICTP	A model that predicts the temperature at locations that are not covered by sensors

1.6 Limitations

The immersion-cooled environment was only implemented on a small scale due to the limited resources available, the fluid used that cannot exceed 40 litres. The resources required for implementation are outlined in Section 4.1.2 of Chapter 4 in this dissertation.

1.7 Document Outline

Chapter 2 contains the necessary background for understanding the motivation for this thesis and the technology being investigated. This chapter discusses astronomy, SKA and the MeerKAT project. The chapter further elaborates on cooling techniques, mainly immersion cooling and air cooling, as well as different modelling techniques. The different sensors and sensor platforms are compared to gain a better understanding of which sensor and platform would be useful for this problem. **Chapter 3** introduces the research methodology and methodology approach for this project, divided into eight phases. The steps of each phase are explained in detail. **Chapter 4** covers the design and implementation process of the EMS. This includes specifying the goals, defining the users and setting out the user requirements. The design approach and the design decisions for different EMS prototype are produced. The implementation of this prototype is discussed in this chapter. **Chapter 5** discusses the design and implementation of the ICTP using the GPR. This includes specifying the goals of the system, choosing a covariance function and learning the hyperparameters. **Chapter 6** presents the results for the EMS and ICTP. This includes tests to show the efficiency of the monitoring system and the accuracy of the ICTP model. The chapter further provides experiments to test whether a hotspot can be detected in immersion cooled environment. The EMS was used to capture and store temperature values for these experiments. Lastly, **Chapter 7** presents conclusions on the project, clearly indicating aspects of successes and recommends areas of future work.

2 Literature Review

This chapter reviews the literature that is relevant for this study to demonstrate growth in the field. The literature is sourced from textbooks and journal articles. The chapter starts with the background of the study where radio astronomy, SKA, MeerKAT and SDP are discussed. Furthermore the chapter covers cooling techniques: immersion cooling and air cooling to better understand these techniques because one of the objectives of this study is to implement an immersion cooled environment on a small scale. Research is done on sensor networks this includes evolution of sensor networks, types of sensors and components of a sensor node to deduce which sensor is suitable for an immersion cooled environment. The chapter looks at different prediction models: linear regression, logistic regression and Gaussian regression, because one of the objectives of this study is to construct a model that predicts temperature in locations where there is no temperature sensor, to construct such a model there is need to understand different prediction models. Figure 2-1 illustrates an overview of the literature review.

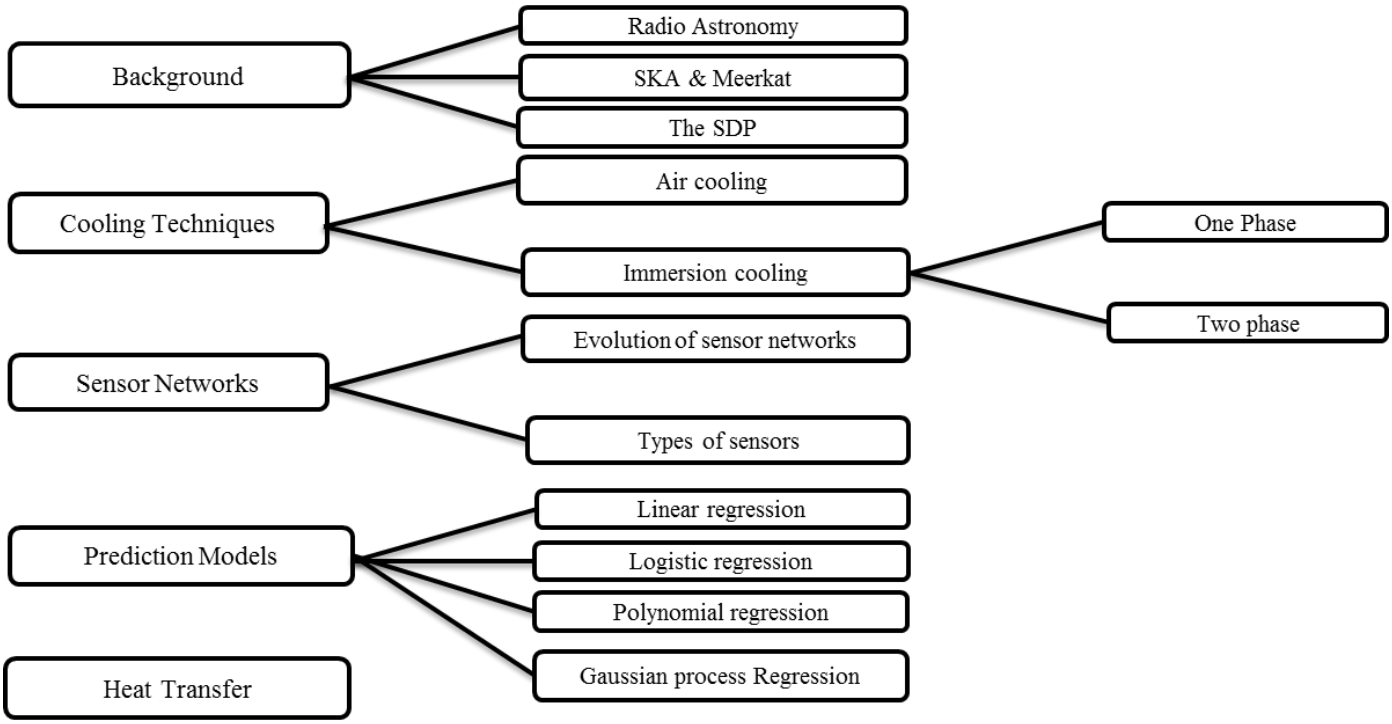


Figure 2-1: An overview of the literature review

2.1 Background

This section provides a brief overview of radio astronomy and the SKA project, before focusing on the specific computing infrastructure that needs to be cooled by means of immersion cooling.

2.1.1 Radio Astronomy

Radio Astronomy is the study of radio emissions from celestial objects through radio waves [9]. Radio waves are emitted from the moon, the sun's gaseous atmosphere and planets. The debris of exploded stars, neighbouring galaxies and distant galaxies, clouds of gas in the spiral arms of our Milky Way galaxy emit radio waves. Radio studies have taken the lead in mapping the structure of our galaxy, and have revealed that the Milky Way is a tightly wound spiral galaxy [9]. Our solar system is traveling along the inner edge of a spiral arm at a distance of 30,000 light years from the centre of rotation. Radio astronomy observations are made at frequencies that are also used in television, FM radio, microwave relay links, rocket and control, and radar [10]. Radio waves are thousands of times longer than light waves, making it easier to use radio telescopes to look deeper into the universe. For instance, the interaction of radio waves with interstellar gas, solar and planetary atmospheres mean that large regions of the universe are forever inaccessible to optical study because of heavy obscuration by interstellar dust; however, this is transparent to radio waves. This emphasises the benefit of using radio telescopes for astronomy [11].

The radio emissions of the universe and the stars can be studied by using radio telescopes. All such radio telescopes have two components, i.e., an antenna and a receiver. Together, they define the sensitivity of a radio telescope. The area and efficiency of the antenna influences whether weak sources of radio emissions can be measured. The sensitivity of the radio receiver determines which signals can be detected, how much such signals can be amplified, and what duration the observation can have. Astronomical radio sources vary in that some are strong, whereas others are extremely weak. Weak radio sources can only be captured by means of very large and sensitive radio telescopes [12].

A well-known radio telescope type is the radio reflector, which consists of a parabolic antenna known as a dish; this focuses the incoming radiation onto a feed. In the context of radio

telescopes, the feed is a waveguide horn connected to a sensitive radio receiver. Solid-state amplifiers, which are cryogenically cooled and have a very low internal noise, are used to obtain the best possible sensitivity. Figure 2-2 illustrates how data travels in a radio telescope: The incoming radio waves hit the primary reflector (or dish), which focuses the reflected waves towards the subreflector, which in term focuses the waves tightly towards the feed horn. The feed may be closely coupled with a receiver and a digitizer. The received signals are digitized into a form where they can be further processed and analysed [13].

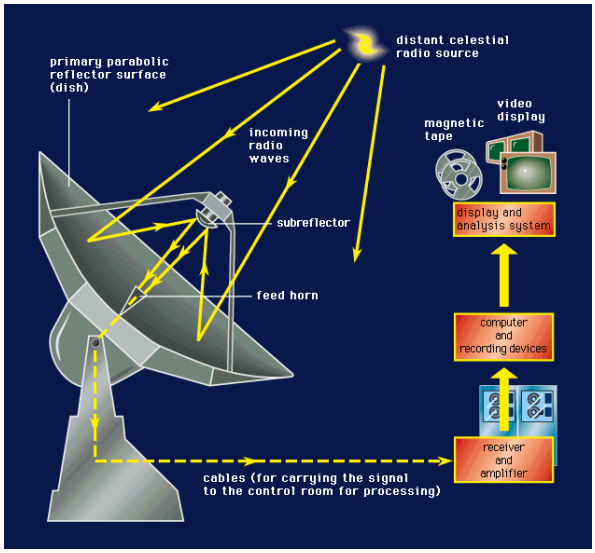


Figure 2-2: The pipeline of data in a radio telescope (image source [16])

2.1.2 SKA and MeerKAT

The SKA project is an international coalition to build the largest and most sensitive radio telescope in the world. It is envisaged that this telescope will be 50 times more sensitive and 10,000 times faster than the currently available radio telescopes [2]. The construction of this instrument draws together experts in a range of fields, including astronomy, physics, computer science and engineering (electrical, mechanical and civil). The success of this instrument depends on the integration of expertise from all these fields and the effective collaboration of all these experts working together efficiently. The data rates involved in the SKA are anticipated to exceed those of the entire global internet traffic per day. Acquiring and processing such an amount of data is going to require a significant amount of computing resources and power. It is not easy to build this instrument with current technology; as a result, precursor projects such as

MeerKAT, ASKAP and Karoo Array Telescope (KAT-7) have been built to test its design principles and develop the technologies that will ultimately make the SKA possible. KAT-7 was the first precursor, designed as a technology and system demonstrator for MeerKAT [14]. The KAT-7 telescope is a radio interferometer consisting of seven fully steerable 12m antennas, completed in 2011[15]. KAT-7 was built a few kilometres north of the Losberg site where the MeerKAT telescope is currently being built in Northern Cape region that has a particularly low population density; more importantly, however, this area has the lowest levels of radio interference in Southern Africa. In fact, part of the Northern Cape surrounding the SKA site was declared a Radio Astronomy Reserve through the Astronomy Geographic Advantage Act of Parliament in 2007[16]. Figure 2-3 illustrates KAT-7 dishes. MeerKAT will be the most sensitive centimetre wavelength instrument in the Southern Hemisphere; it will provide high-dynamic range and high-fidelity imaging over almost an order of magnitude in resolution [17], [18]. MeerKAT will be used by researchers to make new discoveries in radio astronomy.

Figure 2-3: An illustration of the KAT-7 dishes (image sources[19]***)***

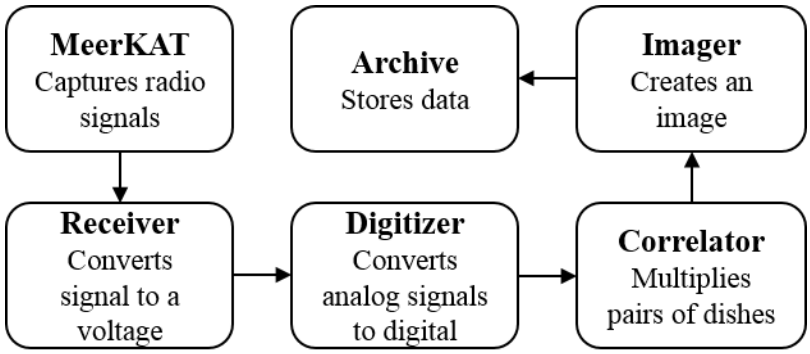


Figure 2-4: MeerKAT data flow

Figure 2-4 illustrates the flow of data in the MeerKAT telescope. Signals from the sky are captured by the receiver and sent to the digitizer where the analog to digital signal conversion is done. The digitizer sends the output to the correlator beam former (CBF). On the CBF the spectrum from the sky is split into gigabit chunks called channels. The CBF also correlates the signal; this data is then sent to the SDP, where the imaging of the radio data happens.

2.1.3 Imaging radio astronomy data

Imaging is one of the most expensive operations in the data processing pipeline. Imaging occurs when the sampled data from the telescopes is added to a grid, after which the grid is Fourier transformed to create an image of the sky[4].

Telescopes often combine data from multiple antennas to increase the sensitivity and resolution of the images. The electromagnetic spectra sampled by each antenna are digitized and converted into the amplitude and phase of the signal(s) coming from observed source(s). Data from each antenna is correlated by multiplying samples of antenna pairs. The output data rate is managed by integrating products over some time interval and thus the output is called visibility. Each visibility has (u,v,w) coordinates, depending on the position of the observed source, the position of the antennas, the frequency of the observed signal, and the time. The (u,v,w) coordinates for an antenna pair change over time because the rotation of the earth alters the antenna's positions with respect to the observed source. After correlation, the visibilities undergo processing, such as removal of interference and calibration to improve the quality of the data. A final two-

dimensional Fast Fourier Transform (FFT) then converts the UV-image to an image of the sky. As discussed earlier, imaging is computationally expensive and requires an HPC cluster[5].

2.1.4 The Science Data Processor (SDP)

Imaging for the MeerKAT telescope is done by the Science Data Processor (SDP). The SDP is responsible for processing the data from the telescope with the aim of producing a data product, primarily an image. Imaging for the MeerKAT telescope happens at the SDP imaging pipeline. Figure 2-5 illustrates the different components of the SDP; the components of the science processor also known as SDP are explained below; most of this information is from the SDP design document found in [20].

.

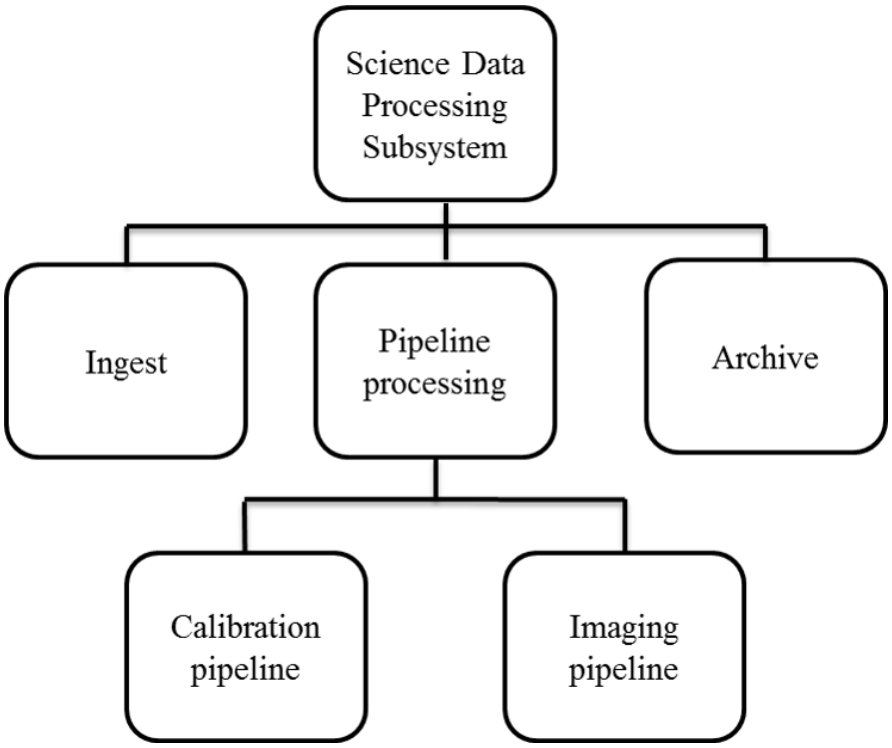


Figure 2-5: SDP subsystems

2.1.4.1 Ingest

The ingest process runs in real time at the full output rate of the CBF; raw data is produced by the correlator at nominally 58 Gbps for MeerKAT. The ingest process performs the necessary

steps to produce visibilities that are of the required temporal and spectral dimension, that have an appropriate gain factor calculated, and that are provided with sufficient flagging to allow pipeline processing to occur. Level-0 data products are visibility data that have undergone basic conditioning. The philosophy of the ingest process is to modify the visibility data as little as possible to preserve information; this conditioning, therefore, excludes higher-level calibration steps.

These visibilities are also packaged with sufficient meta-data to allow further processing to occur without resorting to an additional meta-data request from the Control and Monitoring system. The high data rates for ingest motivate the use of GPU acceleration. GPUs are preferred over other accelerators (such as Xeon Phi); however, both OpenCL and CUDA backends will be supported, which will make it possible to support a range of accelerators, including Xeon Phi. The output of the data ingest process is called a level-0 data product.

2.1.4.2 Pipeline Processor

The pipeline processor performs two distinct functions, which run on separate hardware. These are calibration and imaging. The pipeline operates in a quasi-real-time streaming fashion, ideally producing images shortly after the end of an observation.

Calibration Pipeline

The calibration pipeline takes the level-0 data products from the ingest step, computes gains from interleaved observations of known calibration sources and applies these to produce level-1 data products. The level-1 data products then become the input to the imaging pipeline. The hardware platform for the calibration pipeline consists of 9 physical nodes. Each node receives either a full narrowband spectrum or 1/4 of the wideband spectrum. Each node is identical and runs on Ubuntu 14.04 LTS as a base OS installation with the minimum of additional packages installed, to allow the containerized calibration packages to run.

Imaging Pipeline

After a source scan has been calibrated, the level-1 data product is sent to the imager. The imager buffers data for an observation, and begins processing only once the entire observation is

complete; it does so in less time than the observation took (so that it does not fall behind on average). Figure 2-6 provides an illustration of the SDP imaging pipeline. Wideband fine data requires roughly 63 Tebibyte (TiB) for a 12-hour observation period. Multiple self-calibration and imaging loops are performed. The pipeline also performs preliminary source finding, in the form of elliptical Gaussian fitting; the results are stored along with the images. Source catalogs are used to update sky models for future calibration calculations. More than one pipeline instance is running at the HPC facility, as reprocessing of existing data with alternate pipeline settings is allowed in order to improve on automated imaging products. The time and resource allocation toward these additional pipeline instances is handled by the staff astronomer.

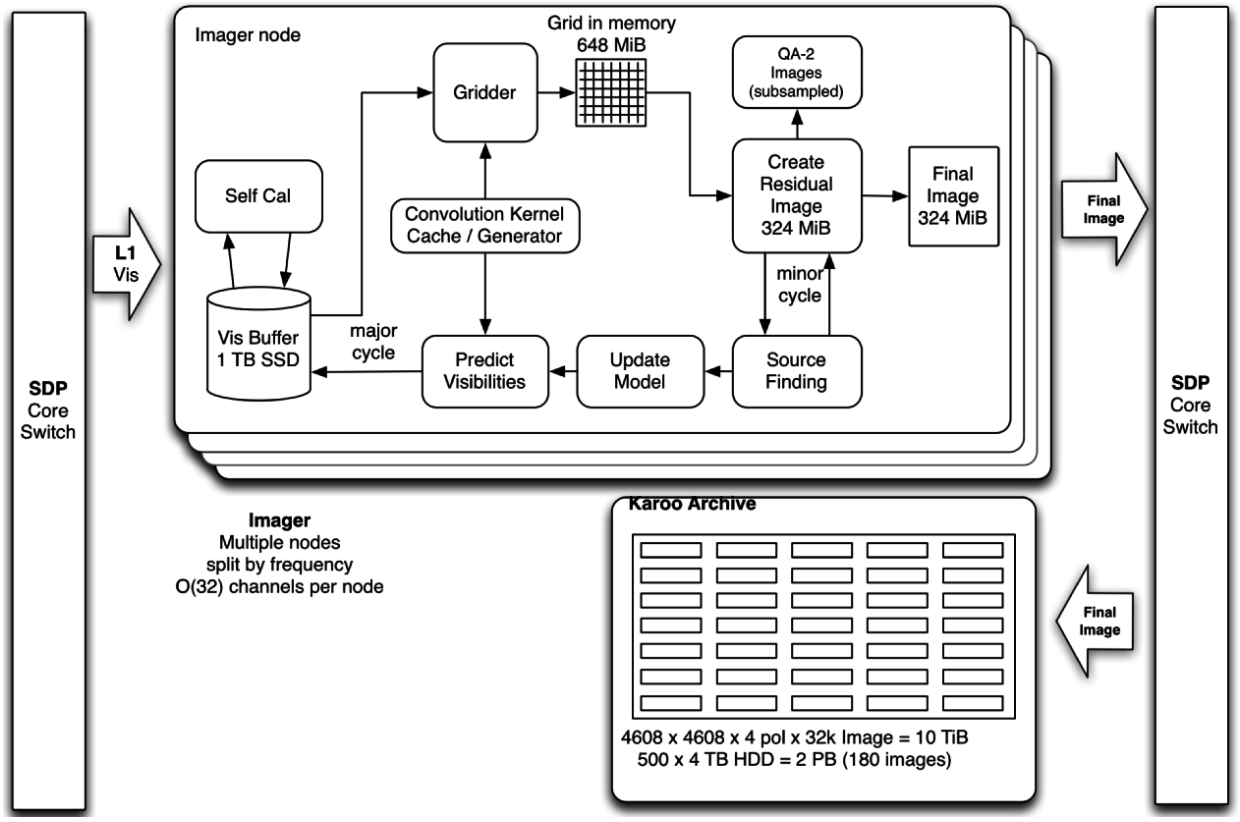


Figure 2-6: SDP Imaging pipeline (image source [20]***)***

The pipeline imager is built from a number of nodes, with parallelization occurring across channels. Each node thus only handles a portion of the band, which minimizes per-node storage and bandwidth requirements. Each node contains sufficient storage to buffer two complete observations for the channels it is processing. Given the HPC requirements, GPU acceleration is being used.

2.1.5 High Performance Computing for SDP Imager

When looking at the data rates and the amount of processing required the MeerKAT imager, it is clear that it requires an HPC facility, mainly a data centre. HPC data centres have been rapidly growing, both in number and size. Current trends in data centre growth show a rapid increase in both computing, storage capacity, and energy consumption [21]. The standard method used to calculate the data centre's energy efficiency is the Power Usage Efficiency (PUE). PUE is the ratio of the power coming into a data centre to the power used to run the computers inside the centre. The PUE statistic show that many traditional data centres consume as much energy expelling heat as they do performing useful computation [22]. Although recently built data centres exhibit better cooling, cooling energy consumption is still a significant portion of the total energy consumption of a data centre [23]. A key challenge toward optimizing the operation of a data centre is to minimize the cooling requirements and, as a result, improve its overall energy efficiency. Another factor in data centres is thermal management; a good thermal management system can monitor the environment through sensor nodes, which gather information about the state of the data centre and which thus detect early faults and possibly maintenance issues. Such a system can also produce the results of power consumption and power distribution[24].

As mentioned above the SKA-SA is investigating new techniques to reduce power consumption in a data centre, one of these solutions is immersion cooling. In 2016 the MeerKAT SDP team conducted an internal survey comparing air cooling with immersion cooling: the results are summarised in Table 2-1. These calculations are only specific to the imager and not the science processor as a whole. Table 2-1 also presents the items required for each cooling system.

Table 2-1: An internal survey comparing air cooling with immersion cooling for the SDP

Immersion cooling					**Air cooling**				
Item	Unit Power	Unit Cost	Total Power	Total Cost	Item	Unit Power	Unit Cost	Total Power	Total Cost
Ground loop	0 W	$1,500	0 W	$16,500	20kw load CRAC	3524 W	$11,00	3524 W	$11,00
Oil pump	14 W	$100	154 W	$1,100	Server room (per RU)	0 W	$2,200	0 W	$96,00
Loop pump	31 W	$500	341 W	$5,500					
Radioactive element	0 W	$500	0 W	$5,500					
Total			495 W	$23,100	Total	3524 W			$107,800
PUE			1.03		PUE			1.15	

It was decided that geothermal cooling will be used for immersion cooling; this process works by using the underground to heat and cool a home or building. Geothermal cooling works because the ground beneath the earth is warmer than the outside air in the winter and cooler in the summer. Figure 2-7 provides an illustration of geothermal cooling. A series of small pipes are inserted into the ground to connect to heat pumps, which is called a ground loop. These pipes allow heat to be transferred to and from the element that needs to be cooled or heated. Furthermore, an oil pump will be placed inside the oil to circulate the oil and create temperature variations.

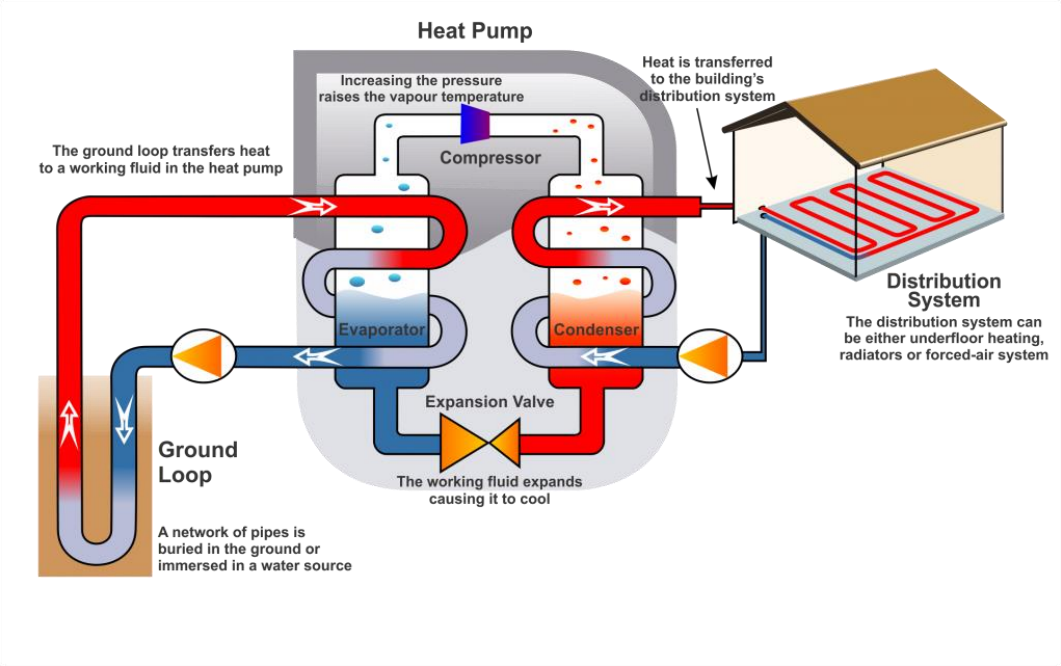


Figure 2-7: An illustration of geothermal cooling (image source [25]***)***

According to the results summarised in Table 2-1, immersion cooling is much cheaper that air cooling and the ratio of the power coming into a data centre to the power used to run the computers inside the centre known as the Power Usage Efficiency (PUE) for immersion cooling is better than that air cooling. This motivated the investigation of immersion cooling, which is one of the objectives of this thesis.

The SDP team are in the process of designing a more energy efficient and cost efficient solution to handle the currently estimated science workload and a system that will be used in an immersion-cooled environment. Figure 2-8 illustrates a possible design of the imager; the number of nodes required will depend on the performance characteristics required. A low-power system-on-chip solution, based on the newly released Tegra "Erista" platform is being investigated. Nvidia Tegra k1 board is illustrated in Figure 2-9 and the NVdia chip is illustrated in Figure 2-10. This combines the CPU, the GPU and an interconnect in a single package, which allows for better performance per watt

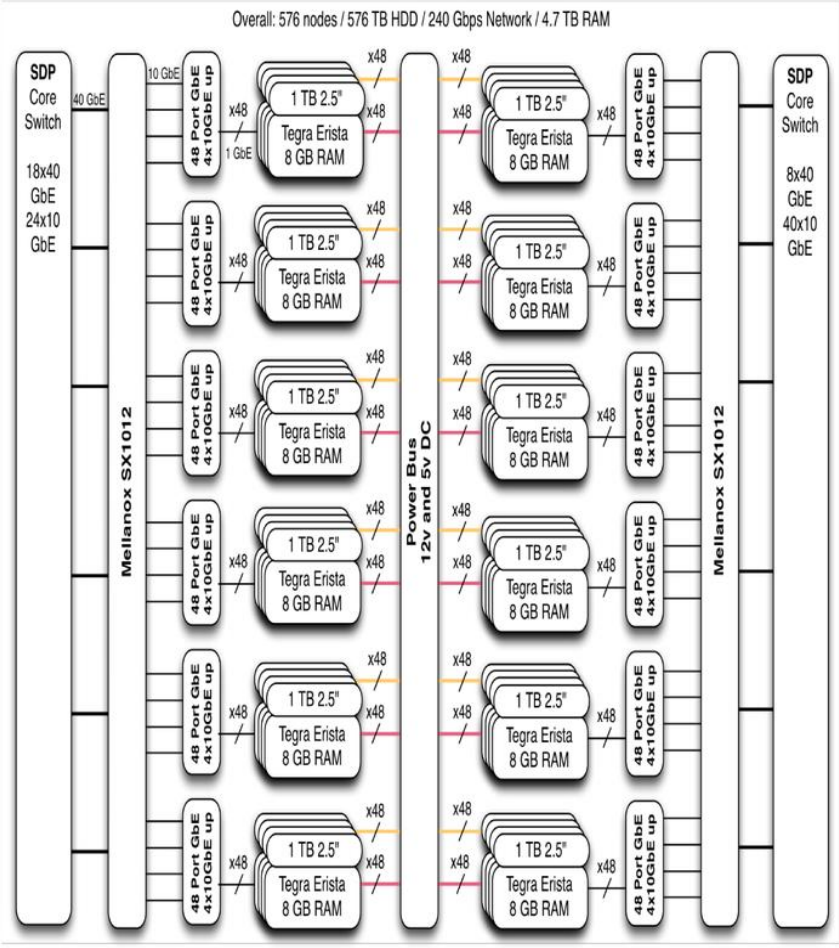


Figure 2-8: Physical design of the Imager (image source [20]***)***

Figure 2-9: Nvdia Tegra K1 board

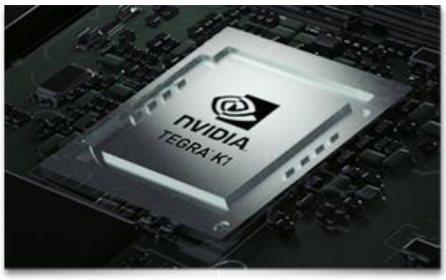


Figure 2-10: Nvdia tegra K1 chip (image source [26]*)*

2.2 Cooling techniques

In this section, air cooling and immersion cooling techniques are discussed.

A data centre is cooled by circulating cool air through the chassis of the computer system [27]. All of the fans in a computer system require power to run; at the same time, they create heat when they run, because they are not perfectly efficient. Fans also create audible noise, contributing to the noise levels in a data centre. Many traditional data centres consume as much energy expelling heat as they do performing useful computation. Power used to run computer fans is often counted as computer load, even though it does not contribute toward actual computation and thus ends up in the denominator of the PUE calculation, as explained in Section 2.1.5 above.

2.2.1 Air Cooling

Traditional air cooling systems use a method known as Computer Room Air Conditioning (CRAC) to pass cool air. CRAC units are placed on the raised floor in the computer room and blow cold air into the underfloor plenum. Perforated floor tiles placed in front of racks of computers allow cold air to enter the computer room. The warm air from the computers travels to the top of the CRAC unit, where it is absorbed, cooled and blown back under the floor [27]. Temperature is controlled at the hot air return of the CRAC units, away from the equipment racks [28]. In air cooling, the computer room needs a source of chilled water, because CRAC units use a chilled-water coil to cool the air. The chilled water (usually 45–55°F) is supplied by the data centre chiller plant [23]. The computer room heat is exhausted to the atmosphere outside via evaporative cooling towers [27]. Figure 2-11 illustrates the process of air cooling. Air cooling

does not only require operation costs, buts also infrastructure costs, such as raised floor and perforated tiles.

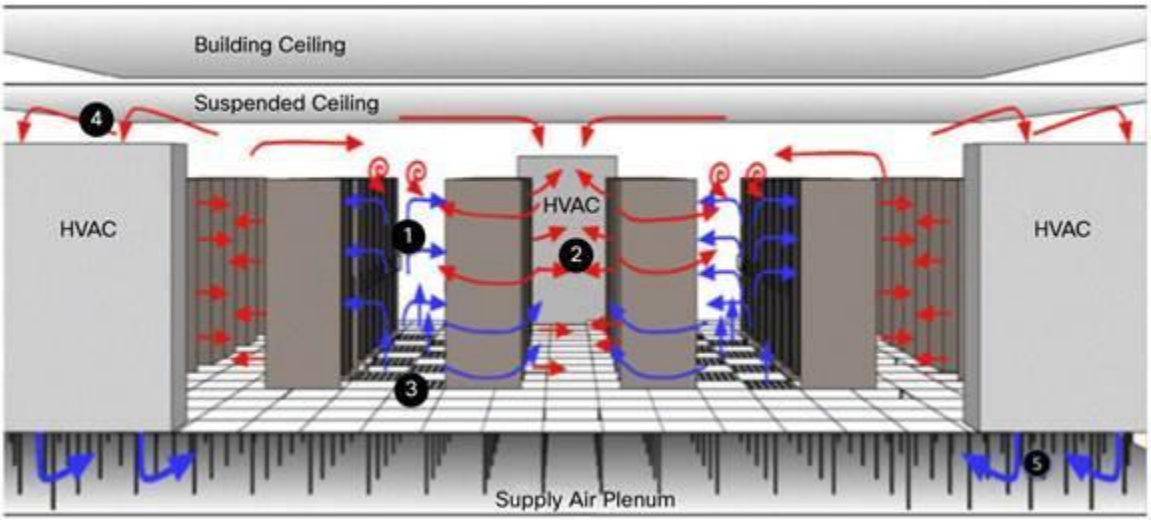


Figure 2-11: An illustration of air cooling from the Sanity technology (image source[29]***)***

2.2.2 Immersion Cooling

Immersion cooling is a cooling technique where the coolant is in direct physical contact with the chips [30]. In this case, the computing equipment is directly immersed in a bath of cooling fluid. Consequently, it is essential that the coolant is chemically compatible with the chips and other packaging materials exposed to the liquid. Several coolants may be able to provide adequate cooling, but only a few are chemically compatible. Water, for example, has very good heat transfer properties but is undesirable for direct immersion cooling because of its chemical and electrical characteristics. In contrast, fluorocarbon liquids (e.g., FC-72, FC-86, FC-77, etc.), in spite of their poorer thermo-physical properties, are considered to be the most suitable liquids for direct immersion cooling. The type of fluid used is not the subject of this project, but the efficiency of the technique is the focus this project. Oil-immersion systems like CRAC units need a source of cooling water because they use an oil-to-water heat exchanger to expel heat. The difference lies in the degrees: CRAC units need 45–55°F water, whereas oil-immersion systems can cool water as warm as 85°F [23]. Oil-immersion systems can thus take advantage of different passive heat sinks, including radiators, geothermal wells, or nearby bodies of water. There are two types of immersion cooling, viz., one phase and two phase, as explained below.

One phase immersion cooling

In one-phase immersion cooling, the heat from the computer components is conducted into the liquid and circulated using a chiller or heat exchanger to shed heat and maintain the liquid's temperature. Immersion systems such as the CarnotJet from Green Revolution Cooling (GRC) use this straightforward and efficient process. Figure 2-12 illustrates one-phase immersion cooling. The main difference between one-phase and two-phase cooling is that one-phase uses a pump for circulation, whereas in two-phase cooling, as explained below, there is no need for a pump to circulate the oil.

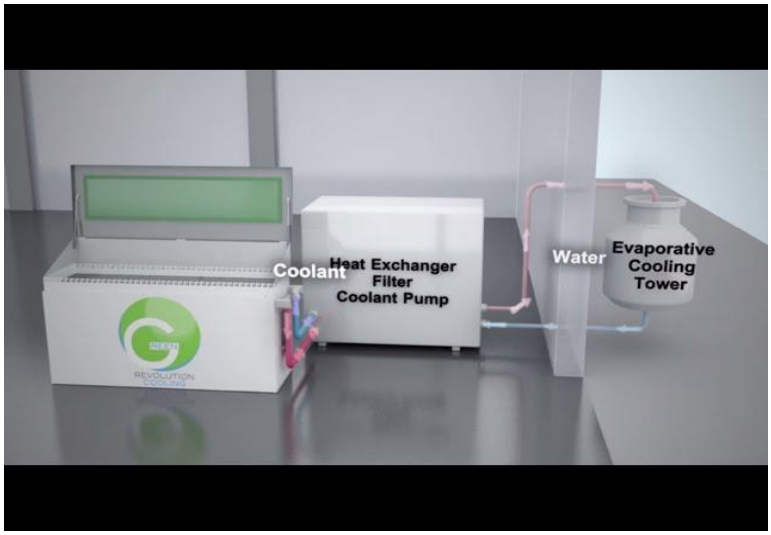


Figure 2-12: CarnotJet from GRC uses one-phase immersion cooling (image source [31]***)***

Two phase immersion cooling

During two-phase immersion cooling, servers and other hardware reside in a liquid bath, similar to one-phase cooling; the difference lies in the type of coolant. The coolant used in two-phase cooling has a much lower boiling point than water or even other immersion media, usually close to 49° Celsius (about 120° Fahrenheit). Heat from the server's processors and other components causes the liquid to boil, creating vapour that carries the heat away into a chilled coil or condenser, where it is collected and reused. In two-phase systems, the liquid is not pumped. Therefore, there is no need for a chiller pump; the circulation process is a passive process, where heat is removed from the liquid in the form of vapour, which condenses on the local condenser; it is then cooled through a chilled water loop. This consumes far less energy than one-phase

cooling and CRAC. Figure 2-13 illustrates two-phase immersion cooling. The major disadvantage with two-phase immersion cooling is the cost of the liquid used.

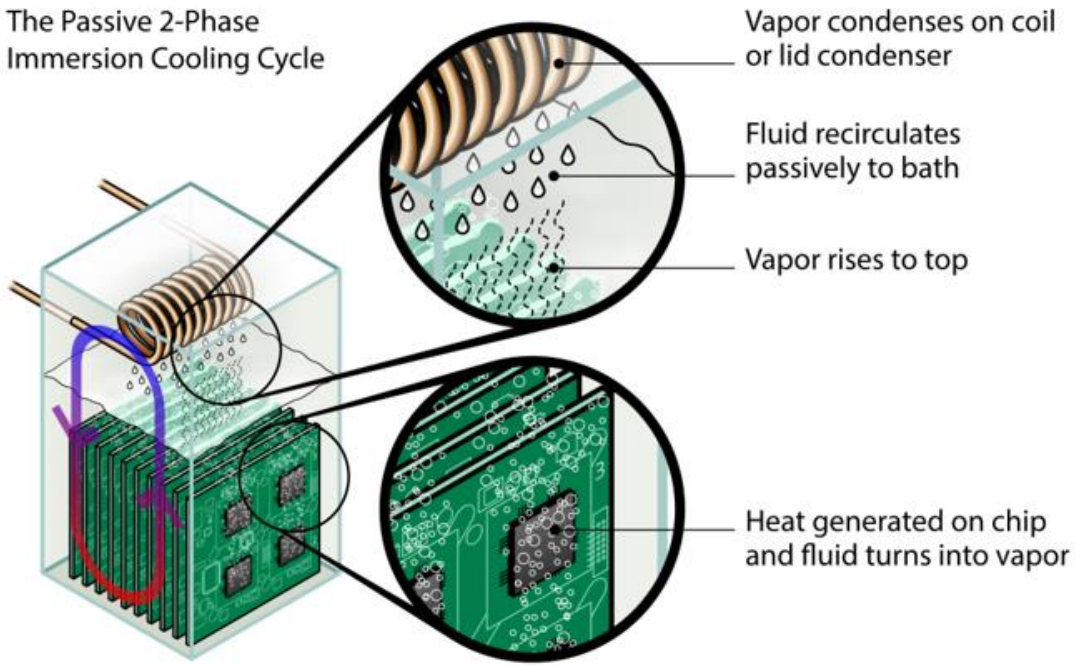


Figure 2-13: Two phase immersion cooling (image source [36])

One of the primary benefits of immersion cooling is the removal of cooling fans from the data centre. The removal of these fans saves significant amounts of energy, and it means that the CRAC units and the chillers can be removed too. Due to this fact alone, immersion cooling uses approximately 10% less energy than air cooling. Internal server fans, however, are not the only fans required for air-cooled computers. A viable solution for this project is one phase immersion cooling due to the costs of the liquid used in two-phase immersion cooling.

2.3 Sensor Networks

The need for sensor networks as a means for environmental monitoring has grown over the years. A sensor network is composed of many tiny sensor nodes that are set out in an environment; these sensor nodes capture information about the environment and either report to a graphical interface or record the information in a database.

Table 2-2: Comparison of wireless and wired sensor networks based on (Table modified from [32]***)***

Characteristic	**Wireless sensor network**	**Wired sensor networks**
Installation	Easy installation (neat and clean, no untidy cables are used in this)	Difficult to moderate (Because more no. of components are used during installation and require cables to be connected to each and every computer in the network.
Visibility node to node on same network	Many nodes on a wireless network cannot connect all of the other wireless nodes on the same network.	All of the nodes on a wired network can connect all the other nodes.
Network to network visibility	Wireless networks are often visible to other wireless networks. One wireless network can affect the performance of other wireless networks.	Networks are invisible to other wired networks. The presence of one wired network has no effect on the performance of another wired network.
Installation time	Shorter installation time (primarily because no untidy cable installation).	Longer installation time (because every single computer in the network must be connected).
Cost	Higher cost (wireless adapters and access points are quite expensive).	Lower cost (Ethernet, cables and switches are not expensive).
User connectivity and network cabling	Connectivity is possible beyond the bounds of physical locations.	Connectivity is possible only to or from those physical networks to which the cabling extends.
Mobility	Outstanding mobility (wireless users are able to connect to the network and communicate with other users).	Limited mobility the network operates only on connected computers that are linked with the network).
Reliability	Reasonably high (if a major section, such as the router breaks down, the whole network will be affected).	High (Ethernet cables and switches are reliable because manufacturers have improved the technology over several decades).
Speed and bandwidth	Low, up to 54 mbps	High, up to 100 mbps
Cables	No cables needed, as system works on radio waves and microwaves.	Ethernet, copper and optical fibres.
Hubs and switches	No need for hubs and switches.	Needs hubs and switches for connections.
Security	Weak security (wireless communication signals travel through the air and can easily be intercepted, but the risk of this can be reduced by encryption techniques).	Good security (software such as firewall software etc. can be used).

A sensor network can be wireless or wired. A wired sensor network makes use of physical cables to transfer data between different devices, whereas a wireless network uses infrared or radio frequency signals to share information and resources between devices, Table 2-2 above provides comparison of a wireless and wired sensor network [32]. Consequently, the evolution of wireless and wired sensor networks will be briefly reviewed to looking at how these have evolved with time, what advantages and challenges are presented by each type of network, and in which scenarios they work best. The wired sensor network is a viable solution for this research because it is cheaper.

2.3.1 Evolution of sensor networks

Initially, the need for wireless sensor networks was driven by military and defence services[33]. This was a time when wireless sensor and network based solutions were only accessible for specific high priority areas due to development costs; now through recent enhancements in sensor design and wireless communication networks, sensor networks have become accessible for many other industrial applications[34]. The availability of low-cost sensors and communication networks has resulted in the development of many potential applications, from infrastructure security to industrial sensing. Due to research in these changing fields, sensor networks have evolved from their use in military contexts to other industries, such as data centre monitoring. Data centre owners are faced with power and thermal challenges on top of traditional IT challenges, such as server management, security, and performance. Such challenges are caused by the huge volumes of people requiring data centre services, which is resulting in massive data centres [35].

2.3.2 Types of sensors

DS18b20 Temperature sensor

The DS18B20 is a three-pin temperature sensor that communicates over a 1-wire bus, Figure 2-14 illustrates the DS18B20. The 1-wire bus requires ground and a data line for communication with a central microprocessor. Each DS18B20 sensor is identified by a unique 64-bit serial code, which allows multiple DS18B20s to function on the same 1-wire bus. This makes it easy to identify temperature sensors and easy to detect a faulty one, especially in large environments [36].

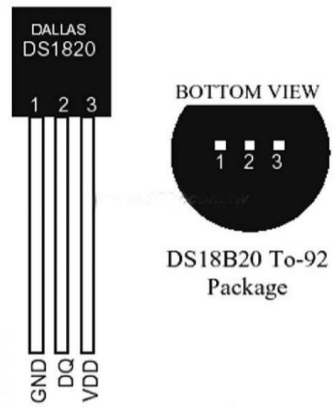


Figure 2-14: The DS18B20 temperature sensor (image source[36]***)***

DHT22 pins	
1	VCC
2	DATA
3	NC
4	GND

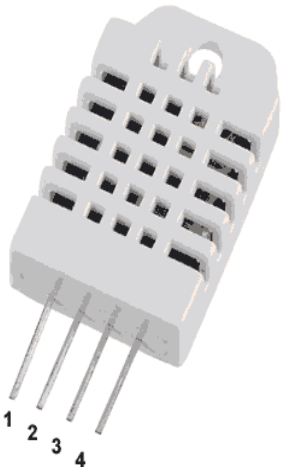


Figure 2-15: The DHT 22 Temperature and humidity sensor (image source [37]***)***

Figure 2-16: TH02 Temperature and Humidity Sensor (image source[38]***)***

DHT 22 Humidity sensor

Relative humidity, usually expressed as a percentage of humidity, measures the amount of water vapour contained in the air. DHT22 also known as AM2302 is a humidity and temperature sensor. This sensor has proven to be reliable and stable. The output of the DHT22 is a calibrated digital signal, which can be interfaced directly to an Arduino Uno port pin and raspberry pi[37]. It calibrates automatically by means of a digital-signal-collecting-technique and humidity sensing technology. With its small size, as illustrated in Figure 2-15, its low power consumption, and its ability to function in all kinds of harsh environments, the DHT22 is a suitable choice for monitoring humidity in various systems[39].

TH02 Humidity and temperature sensor

The TH02, a monolithic Complementary Metal-Oxide Semiconductor Integrated Circuit (CMOS IC) is a digital relative humidity and temperature sensor. The CMOS IC integrates temperature and humidity sensor elements, an analog-to-digital converter, signal processing, calibration data, and an Inter-Integrated Circuit (I2C) host interface. Figure 2-16 provides an illustration of the THo2 humidity and temperature sensor. The sensor is factory-calibrated, and the calibration data is stored in the on-chip non-volatile memory. This ensures that the sensors are fully interchangeable, with no recalibration or software changes required [40].

SHT15 Humidity and temperature sensor

SHT15 belongs to the family of Sensirion's surface mountable relative humidity and temperature sensors, as illustrated in Figure 2-17. The sensor elements and signal processing are integrated on a tiny footprint that provides a calibrated output. The temperature is measured by a band-gap sensor, while the humidity is measured by a unique capacitive sensor element. SHT15 uses a 2-wire serial interface and internal voltage regulation that allows for easy and fast system integration [41].

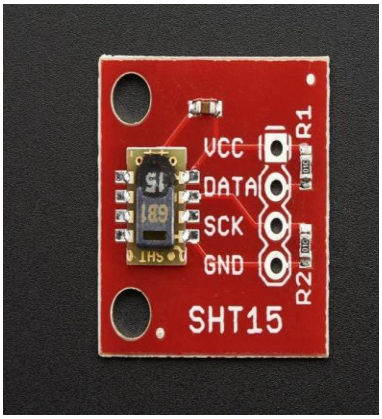


Figure 2-17: SHT15 Humidity and temperature sensor (image source [42]***)***

Figure 2-18: Honeywell Humidity Sensor (image source [43]***)***

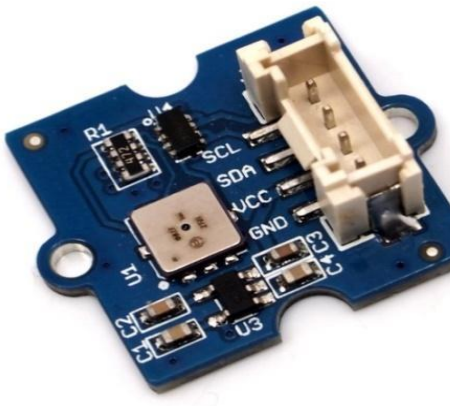


Figure 2-19: Grove temperature sensor (image source [44]***)***

Honeywell's Humidity sensor

The HIH-4000 Series humidity sensor allows direct input to a controller or other device through the sensor's near linear voltage output. The sensor delivers instrumentation-quality RH (Relative Humidity) sensing performance in a competitively priced, solderable SIP (Single In-line Package). Figure 2-18 illustrates the Honeywell humidity sensor [45].

Grove Temperature sensor

The Grove temperature sensor uses a thermistor to detect the ambient temperature. The resistance of a thermistor increases when the ambient temperature decreases. The sensor has an

accuracy of ±1.5ºC, and a detectable range of -40 - 125ºC. Figure 2-19 illustrates the Grove temperature sensor[44].

LM35 Temperature sensor

The LM35 is a precision integrated-circuit temperature device with an output voltage linearly proportional to the Centigrade temperature. The sensor has 0.5°C ensured accuracy at 25°C. Figure 2-20 illustrates the LM35 temperature sensor [46].

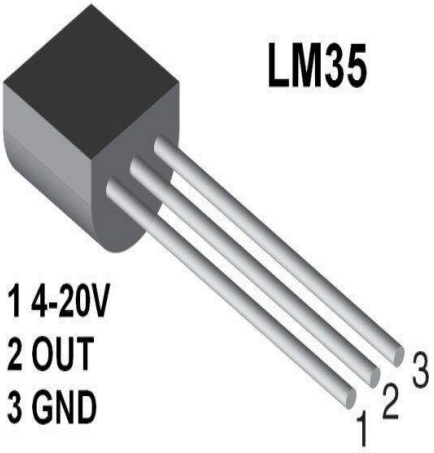


Figure 2-20: LM35 Temperature sensor (image source [47]***)***

Figure 2-21: TMP100 Temperature Sensor (image source [48]***)***

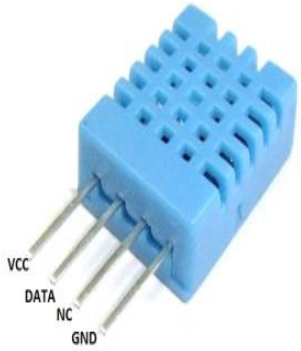


Figure 2-22: DHT11 Humidity and Temperature sensor (image source [49]***)***

TMP100 Temperature sensor

The TMP100 is a 2-wire, serial output temperature sensor available in SOT23−6 packages. Figure 2-21 illustrates the TMP 100 temperature sensor. The TMP100 is capable of reading temperatures with a resolution of 0.0625°C, requiring no external components. The TMP100M operates over a temperature range of −55°C to +125°C[50].

DHT11 Humidity and temperature sensor

The DHT11 Temperature and Humidity Sensor uses exclusive digital-signal-acquisition technique and temperature and humidity sensing technology, to ensure excellent long-term

stability and high reliability. The DHT11 element is calibrated in the laboratory. It uses a single-wire serial interface, making system integration quick and easy. The sensor is small, uses low power, and has up-to-20-meter signal transmission. Figure 2-22 illustrates the DHT11 temperature and humidity sensor[49].

LM75 Temperature sensor

The LM75 temperature sensor includes a digital over temperature detector and a delta-sigma analog-to-digital converter. The host can read the temperature at any time by querying the LM75 through its I²C interface [51]. Figure 2-23 illustrates the LM75 temperature sensor.

Figure 2-23: The LM75 Temperature Sensor (image source [52]***)***

Table 2-3: Evaluation of Sensors

Name of Sensor	Type	Bus Support	Power Supply	Range	Accuracy	Analog / Digital	Cost	Language	Board Support
DS18B20	Temp Sensor	Dallas 1-wire	3.0V to 5.5V	–55°C to +125°C	±0.5°C accuracy from –10°C to +85°C	Digital	$4.25	Python, C++,Java	Raspberry Pi
RHT03 (DHT-22)	Humid Sensor	MaxDetect 1-wire	3.3-6V DC	0-100% RH & -40 - 80 degrees C	± 2% RH & ± 0.5 ºC	Digital	$9.95	Python, C++	Raspberry pi
TH02	Humid and temp sensor	I2C	2.1 to 3.6 V	0 to 100% RH & –40 to +85 ºC	± 4.5 % RH & ±0.5 ºC	Digital	$14.90	Python, C++	Galileo
SHT15	Humid and temp sensor	2-wire interface		0-100% RH	± 2% RH	Digital	$28.95	C,C++,Python	Raspberry pi
Honeywell's (HIH-4030)	Humidity	I2C	4-5.8V	0-100% RH	±3.5%	Analog	$16.95		Galileo Gen1,
Grove Temp sensor (NTC Thermistor)	Temp	I2C	5V	-40 to 125°C	±1.5ºC	Analog	$2.90	C,C++,Python	Galileo Gen1,Raspberry Pi
LM35DZ	Temp	I2C	4V to 30V	-55 to 150°C	±0.6ºC	Analog	$7.45	C++	Intel Edison
TMP100	Temp	I2C	2.7V to 5.5V	-55 to 125°C	± 2ºC	Digital	$11.55	C++	Raspberry pi
DHT11	Humid and temp sensor	MaxDetect 1-wire	3.5V-5.5V	20%~90%RH & 0~+50°C	±5.0%RH & ±2.0ºC	Digital	$0.95	C++,Python	Raspberry pi
LM75	Temp	I2C	3.0V-5.5V	-25 -100°C	± 2ºC	Digital	$2.21	C++	Raspberry pi

Table 2-3 is an evaluation of the temperature and humidity sensors. The sensors are compared according to the bus they support, their range and accuracy, whether they are analog or digital, and how much they cost in dollars. This survey was done in 2015; there might be a slight change in prices as the years progress. The DS18B20 temperature sensor is a viable solution because it is cheaper, has accuracy of ±0.5°C and makes use of the one wire protocol that allows ease of increasing the number of sensor nodes. The DHT22 humidity sensor is a viable solution for monitoring humidity due to the ± 2% RH accuracy.

2.3.3 Components of a sensor node

A sensor node is made up of a processing unit, a sensing unit, a transceiver unit, and a power unit. Sensing units are composed of sensors, which collect the raw data, and analog-to-digital

converters (ADCs), which convert the analog-to-digital signals produced by the sensors and feed the results to the processing unit. The processing unit normally contains a storage unit and manages the procedures that ensure that the sensor node collaborates with the sensors and with the other nodes to carry out the assigned sensing tasks. A transceiver unit is responsible for connecting the node to the network, while the power unit provides power to the sensor nodes and the entire sensor network [53]. Table 2-4 below shows the different embedded platforms that can be used as processing units for sensor networks.

Table 2-4: Comparison between Galileo, Raspberry Pi & Arduino UNO and C8051F20X (Mote family) based on [35]**.**

	Intel Galileo	**Raspberry pi (model B)**	**Arduino Uno Rev3**	**C8051F20X**
Board dimensions	100mm X 70mm	85.60mm X 53.4 mm	68.6mm X 53.4 mm	200mm X 180mm
Processor	Intel quark X100-single core	Broadcom BCM 2835-single core	ATmega 328 single core	8051
Architecture	Intel Pentium	ARM 1176	Advanced RISC	C IP-51
Speed	400MHz	700MHz	16MHz	25MHz
Analog I/O	Up to 6 analog inputs with 12-bit resolution.	17 General purpose input output (GPIO) pins	Up to 6 analog inputs with 12-bit resolution	Up to 8 analog inputs with 8-bit resolution
TWI/I^2C	Yes	No	Yes	Yes
Serial data (UART)	Yes	Yes	Yes	Yes
Digital I/O	14 digital I/)	8 GPIO	14 digital I/O	8 GPIO
Video support	No	HDMI – 1080p	No	No

The raspberry pi is a viable solution, because it supports Linux, is cheaper, has a built in support for DS18B20 temperature sensor chosen in section 2.3.2 above, supports the DHT22 humidity sensor chosen above and there is a lot of documentation available.

2.4 Prediction Models

Various prediction models are discussed in this section, a vaiable model for the implementing the ICTP is chosen at the and of the section.

2.4.1 Linear Regression

Linear regression consists of finding the best fit in a set of points by drawing the best fitting line, which is called a regression line. Given some information of a dependent variable at certain values of the independent variable x, the aim is to find the best estimate of the dependent variable at a new value, x'. Linear regression works by establishing a relationship between the dependent variable and one or more independent variables, using a best fit straight line. It is represented by an equation $Y = b + c * X + e$ where b is the intercept, c is the slope of the line and e is the error term. This equation can be used to predict the value of target variables based on given predictor variable(s). There are two types of linear regressions: simple linear regression, which has one independent variable, and multiple linear regressions, which have more than one independent variable [54].

In linear regression, the underlying function has to be linear, an assumption about the input data is made, and the least-squares method is used to fit a straight line, where the observed data is calculated by minimizing the sum of the squares of the vertical deviations from each data point to the line. Linear regression is a model where the output is a linear combination of the inputs. Linear regression is mostly popular because of its simplicity in implementation and interpretability. Its main drawback is its limited flexibility; if the relationship between input and output is not a linear function, the model will give poor and incorrect predictions [54].

2.4.2 Logistic Regression

Logistic regression is a regression model where the dependent variable is categorical, i.e. binary 0/1, True/False, Yes/No in nature. Logistic regression analyses a dataset that determines an outcome with one or more independent variables. The goal of logistic regression is to find the best fitting model to describe the relationship between the two characteristics of interest and the independent variables or values to be predicted. Logistic regression chooses parameters that maximize the likelihood of observing the sample values [55].

2.4.3 A Gaussian Process Regression

A Gaussian process regression (GPR) is an even finer approach instead of claiming f(x) relates to some specific models or function (e.g. *f(x) = mx + c),* a GPR can represent f(x) by learning the

nature of the data and relationship between the points in the data. Observations in a GPR occur in a continuous domain and the distribution is a joint distribution of all those (infinitely many) random variables; as such, it is a distribution over functions with a continuous domain [56]. All the points in this input space are associated with a normally distributed random variable and every finite collection of those random variables has a multivariate normal distribution. GPRs are used to perform non-parametric models and can be applied for both regression and classification problems [57].

GPR is used for regression problems.

A GPR is characterized by specifying the mean m(x) and covariance k (x, x') functions. Given the mean and covariance function for the GPR, an estimate of the function it represents can be seen. For any subset X consisting of k of these random variables, we can construct the vector μ of their expectations and the matrix Σ of the covariance between variables. Then the joint distribution of X is given by equation 1:

$$P(X \quad x) \quad \frac{}{(2\pi)^{k/2}|\Sigma|)} \quad (-1/2(x \ \ \mu)^T \quad ^1(x \ \ \mu)) \quad \ldots \text{(eq. 1)}$$

We can index a single random variable by x. The value of a particular random variable x then represents the value of some function f at the location x. For example, a GPR may consist of a collection of random variables that represent the function f, where f (x, y) is the temperature at a given location in 2-dimensional space. If the random variables are indexed by $(x_i, x_j), then\ f(x_i)$ represents the temperature at location (x_i, x_j) [58].

GPR is the viable solution for this research because GPR can represent f(x) by learning the nature of the data and relationship between the points in the data.

2.5 Chapter summary

This chapter involved reviewing relevant theories and trying to investigate component and tool choices that would lead in to the design. The chapter covers cooling techniques: immersion cooling and air cooling where immersion cooling is chosen as a viable solution. Research is done on sensor networks this includes evolution of sensor networks, types of sensors and components

of a sensor node, a wired sensor network is chosen and the DS18B20 temperature sensor and DHT222 humid sensor are the chosen sensors. The Raspberry Pi is chosen because it has a built in support for the DS18B20 temperature sensor and supports the DHT22 humidity sensor. The chapter looks at different prediction models: linear regression, logistic regression, polynomial regression and Gaussian regression, where the GPR is chosen as a viable solution for this study. The chapter provides literature on different software models, namely spiral, built and fix, waterfall, prototype and incremental model. The prototype model was chosen as a viable solution for this project. The chapter concludes by a section on heat transfer with a focus on convection. The methodology that is followed in this research is discussed in the next chapter.

3 Thesis overview

This chapter describes the thesis overview. The aim of this chapter is to give the reader insight into the development of the two subsystems, viz., the Environmental Monitoring System (EMS), a system that monitors the temperature and humidity for an immersion cooled environment and the Immersion Cooling Temperature Predictor (ICTP), a model that predicts the temperature at locations that are not covered by sensors. Together the ICTP and EMS make up a full system.

3.1 Introduction

The project has two objectives, as indicated in Chapter 1: Firstly, to implement a prototype of an EMS, a system that can monitor the temperature and humidity, and secondly, to implement a software program (the ICTP) that predicts temperature in locations that are not covered by sensors. The temperature monitoring component of the EMS will be used to gather data to build the model. The EMS and ICTP together make up a full system. The methodology of the system comprises various phases of research and development, which are listed in Figure 3-1 below:

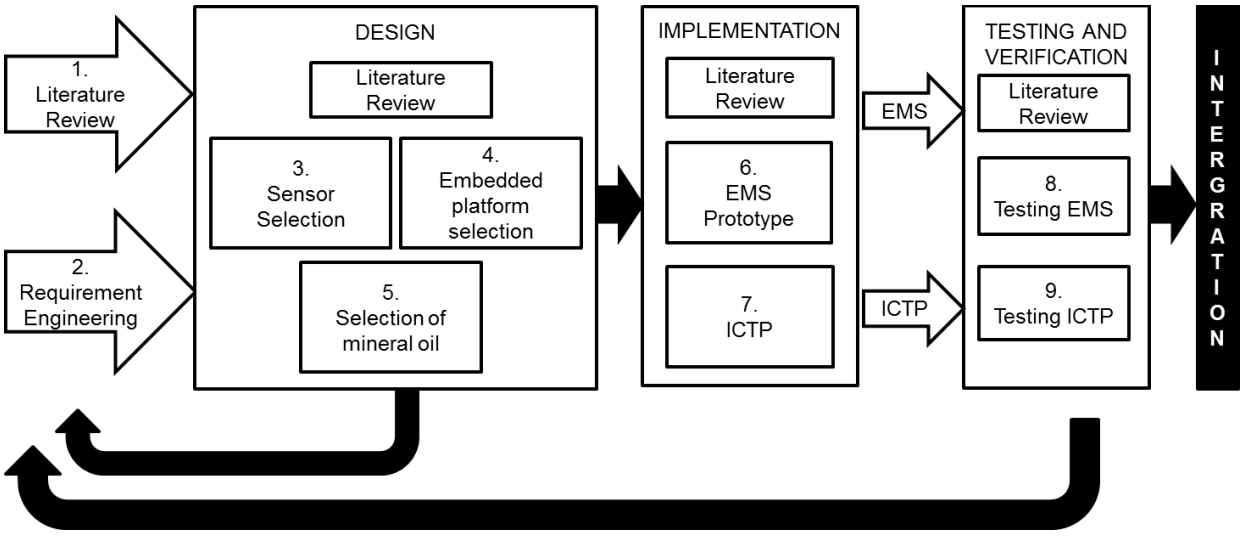


Figure 3-1: Methodology followed in the research

The thesis is implemented in the following steps as indicated below. Each of these will be discussed in further detail in the subsequent sections:

- Step 1 – Literature review: This project started with a literature review to gain insight into what is available and how the concepts of the research have evolved over time.

- Step 2 – Requirement engineering: The stakeholder requirements (in this case, the requirements of the MeerKAT SDP) team were clearly defined to ensure that all parties had a clear understanding of the project and that mutual agreements were reached regarding the project goals.
- Step 3 – Sensor selection: An investigation of sensor options was performed according to a criteria list provided by the stakeholders. The most effective sensors were made, and orders were placed.
- Step 4 – Embedded platform selection: After a thorough review of embedded platforms, a platform that satisfies the set criteria was selected; this included sensor interfacing hardware development and wiring.
- Step 5 – Selection of mineral oil: The SDP team, the stakeholders of this project, had preselected the mineral oil that would be used for this project. Ideally, it should be properly investigated which is the best mineral oil to use, but this is beyond the scope of this project.
- Step 6 – Design and implementation of the EMS: The EMS was designed and implemented according to the specified requirements, which are outlined in Chapter 4.
- Step 7 – Implementing the ICTP: The ICTP was designed and implemented; data was collected using the EMS.
- Step 8 – Testing EMS: The EMS was designed and implemented, using the selected sensors and the embedded platform.
- Step 9 – Testing and verification of EMS and ICTP: Both subsystems were tested and verified individually and then integrated.

The method followed in this thesis is illustrated in Figure 25. The literature review and the requirements are inputs to the entire process. The process is broken down into various stages: design, implementation, testing and verification. Each of these started with a review of the literature. Using the inputs, the EMS was designed and implemented. Using the EMS prototype, the ICTP simulator was built. The outputs of the design and implementation were the two subsystems, the EMS prototype and the ICTP; these subsystems were analysed, tested and verified individually. If the performance metrics were not met adequately for each of the subsystems, they were refined. Once the subsystems were operating adequately, they were

integrated. The design process was compared with the requirements before continuing with implementation, to ensure that the needs of the stakeholders had been met. The results from testing and verification were also compared with the requirements to ensure that the needs of stakeholder had been met.

3.1 Step 1: Literature Review

The process began by researching the problem and evaluating previous solutions. The main aim of the literature review was to gain insight into how the solutions to the problem had evolved over time. It allowed us to evaluate technologies that could be used to develop EMS, to avoid redundancy by creating a system that had already been developed, and to save time by focusing on the aspects that needed more work.

3.2 Step 2: Requirement Engineering

The Systems Engineering (SE) approach to was used to gather requirements for the system. SE is an interdisciplinary approach used to realize successful engineering systems[59].

The stakeholders for this project are the SDP Team represented by Simon Ratcliffe who is the Technical Lead for the team. The six requirements of the SDP monitoring system are tabulated in table 3-1 below:

Table 3-1: High level requirements of the system

Requirement	**Explanation**
Monitoring	The EMS shall monitor environmental characteristics of the system
Control	The EMS shall control the environmental characteristics of the system
Reporting	The EMS shall report on the state of the environment
Generate Alarms	The EMS shall raise alarms when necessary
Application Interface	The EMS shall have a Karoo Array Telescope Control Protocol(KATCP) interface that will monitor the environment in application level
Prediction	The EMS shall have a prediction functionality that predicts temperature in locations where there are no physical sensors.

A list of requirements was recorded using Innoslate, a requirement management tool that allows multiple people to work on the same project and export the project to an MS Word document.

Meetings were held with the stakeholders to ensure that the requirements are adequately satisfied. Once the stakeholders were satisfied with the requirements, a System Specification document, in other words, a document that best describes the user requirements, was signed by Simon Ratcliffe [60]. The requirement document for the SDP is available at [20].

3.3 Step 3: Sensor Selection

Sensors were analysed according to the type of communication bus they support, the power supply range, temperature range, accuracy, whether they support analog or digital, and the cost of the sensors. The results of this analysis are presented in Table 3 in section 2.2 of Chapter 2. The preferred sensor must be:

- Digital
- The most accurate amongst the chosen sensors
- The cheapest amongst the chosen sensors
- Easily scalable

Using the results from Table 3, the DS18B20 temperature sensor and DHT22 humidity sensor were chosen.

3.4 Step 4: Embedded Platform Selection

This phase explains the procedure that was used to analyse different embedded systems to enable us to choose one that would be suitable for this particular problem. All the embedded systems were qualified according to the qualifying factors listed below. The humidity and temperature sensor were chosen in Phase 3; this meant that the embedded system had to support these two sensors. The qualifying factors had been specified by the SDP. Three embedded systems were chosen, the Raspberry Pi, Intel Galileo and Arduino. Table 2 in the literature review is a comparison of these three boards. A board that did not meet these requirements was disqualified. The platform must:

- Support the DS18B20 temperature sensor and the DHT22 humidity sensor
- Allow scalability of the DS18B20 temperature sensors and be able to connect the sensor network with a minimum of 60 sensors because the temperature sensor network would be used to gather data for the ICTP; the minimum requirement was 60 sensors.
- Have Linux support
- Have Python support
- Have Networking support – Ethernet cable
- Support all five qualifying factors above and be the cheapest

The Raspberry Pi embedded platform was chosen because it supports all five qualifying factors above and because it was the cheapest.

3.5 Step 5: Selection of Mineral Oil

The stakeholder had preselected Indy Pharmex 68 BP [61] as the mineral oil to be used as a cooling method for this project. A literature reviews to understand the oil was completed. The type of mineral oil used was not within the scope of this project, and no further mineral oils were investigated or tested. The Indy Pharmex BP mineral oil was thus used for this project.

3.6 Step 6: EMS Prototype

The EMS was designed and implemented, noting that it would be used to gather data for the ICTP. The temperature sensors used had to be scalable to ensure that sufficient temperature readings were achieved. The entire process is discussed in Chapter 4 of this dissertation.

3.7 Step 7: Constructing an ICTP

Gaussian Process Regression (GPR) was used to model the ICTP, GPR is chosen because it produces a simple, accurate model that produces uncertainty and prediction; GPR is also flexible to incorporate physical setup of your experiment through the kernel. GPR and the implementation of the model is further discussed in Chapter 5 of this study.

3.8 Step 8: Testing the EMS

During Phase 8, the EMS was tested and validated. The aim was to test if the system could function without human intervention, whether it was an effective capturing system, and whether it was able to handle errors. Table 3-2 summarises the testing methods that were used to test the EMS.

Table 3-2: Methods used for testing the EMS

Test	**Experiment**	**Validate result**
Test accuracy of the sensor	Define two points A and B, 10cm apart. Point A is the location of the sensor and Point B is the location of the thermometer. The distance chosen is 10cm because that is the distance between the sensor locations in the scaffold illustrated in Figure 30. Record temperature readings from the sensor and the thermometer.	If the temperature for point A coincides with the temperature at point B then the sensor is accurate.
Test if the system can function without human intervention	Leave the system capturing for a week and check for failures	A plot demonstrating data captured for a week with no disturbance.
Test if the system is fault tolerant	Demonstrate what happens when a sensor or two is deleted. Demonstrate how the system handles this.	A visual illustrating how the system handles this error.
Test the capturing system	Demonstrate that the system can capture data.	An image of the file.

3.9 Step 9: Testing the ICTP

The ICTP was tested to understand the accuracy of the model and its ability to identify the locations of hotspots. Table 3-3 tabulates the testing methods that were used to test the EMS.

Table 3-3: Methods used for testing the ICTP

Test	Experiment	Validate results
Test the quality of the model	Obtain a prediction of temperature and uncertainty.	An illustration to show that measurements are within the predicted uncertainty.
Test that the model can predict temperature at an unknown location	Measure temperature at a place that was not tested	A plot demonstrating that the unknown value is within error bars predicted.

3.10 Chapter summary

In this chapter the methodology followed in this research was discussed, the methodology is divided in 9 steps namely literature review, requirement engineering, sensor selection, embedded platform selection, selection of mineral oil, EMS prototype, ICTP model, testing EMS and testing ICTP. Various test methods for the EMS and ICTP were clearly defined. The implementation of the EMS is discussed in the following chapter.

4 Design and Implementation of the EMS

This chapter describes the design and development of the Environmental Monitoring System (EMS) and uses a series of diagrams and photographs to explain the design and implementation of the EMS.

4.1 EMS design

In order to provide monitoring for the SDP imager, a system that monitors the temperature, and humidity was designed and implemented. An oil-cooled environment was built at a small scale to gain understanding on the technique, a prototype is built to monitor the environment in the built oil-cooled environment. In this section the design of the EMS is discussed, this includes goals, resources, user requirements, design approach and design decisions.

4.1.1 Goals

The first goal of the EMS was to monitor the temperature of the oil-cooled environment. The temperature sensor allows us to identify hot spots; these are very common in oil-cooled environments, especially when there is no circulation in the oil. A hotspot in this project was defined as a stagnant area that reached a temperature above 40 degrees; this was done because, if the temperature is stagnant at 40 degrees or higher, this might damage the Printed Circuit Boards (PCB).

The second goal of the EMS was to monitor the humidity of the oil-cooled environment. The humidity sensor informs us about high humidity levels that might cause corrosion and short-circuits on the PCB.

The third goal was data logging; the EMS was required to log data in a format that could be accessed by the users. Data logging was also important in collecting data to model the ICTP. The temperature readings obtained by the EMS had to be in a format that could be easily used during the implementation of the ICTP.

4.1.2 Resources

The EMS was implemented using hardware made available by SKA SA. Table 4-1 below shows the relevant resources, along with a short description of each resource.

Table 4-1: Table of resources provided by the SKA-SA and a description of each resource

Resource	Quantity	Description
Raspberry pi 2 model B	2	A credit sized computer that plugs into a TV or a monitor with a Linux operating system
Intel Galileo Board	1	A microcontroller board from Intel designed to be hardware and software compatible with the Arduino shield ecosystem
Grove Base Shield	1	A shield that allows users to add grove modules to the Arduino or Intel Galileo Board
Grove temperature sensor	1	A temperature sensor that uses a thermistor to measure ambient temperature
DHT22 Humidity sensor	2	A low-cost digital temperature and humidity sensor
DS18B20 temperature sensor	60	A low-cost digital temperature sensor
Jumper wires		An electronic wire used to interconnect the electric components
Laing DDC-3.2 PWM Pump	1	A High-Performance water cooling pump

4.1.3 Users

The direct users of the EMS were members of the SDP team, which needed access to the information provided by the EMS. The long-term goal of this system was for it to be integrated into the MeerKAT system. This meant that there would be more users of the system, such as the operators, for instance. The operators are the people who operate the telescope and need information from the system such as data logging. In addition, the Control and Monitoring (CAM) Team is responsible for controlling and monitoring the MeerKAT system, so they too would be using the system. With all the users identified, requirements for the system are outlined in the next section.

4.1.4 User Requirements

The key stakeholders for this project are the SDP Team represented by Simon Ratcliffe who is the Technical Lead for the team. A list of requirements was recorded using Innoslate, a

requirement management tool that allows multiple people to work on the same project and export the project to an MS Word document. Meetings were held with the stakeholders to ensure that the requirements are adequately satisfied. Once the stakeholders were satisfied with the requirements, a System Specification document, in other words, a document that best describes the user requirements, was signed by Simon Ratcliffe. The requirement document for the SDP is available at [20]. The requirements and users for this system are outlined in Table 4-2.

Table 4-2: Functional requirements for the EMS

Requirement	**Function**	**Users**
Monitor temperature	Report on changes in temperature every minute	SDP Team Operators CAM Team
Monitor humidity	Report on changes in humidity every 5 minutes	SDP Team Operators CAM Team
Data logging	Store temperature and humidity	SDP Team Operators CAM Team
Fault detection	The EMS shall report if one of the sensors is no longer working	SDP Team Operators CAM Team

4.1.5 Design Approach

The design approach used was evolutionary prototyping, an iterative process that allows for user input throughout the development process [62]. The advantage of this approach is that it begins with a basic prototype, which can be incorporated into the final system. The prototype is presented to the users who provide feedback. The developer then uses this feedback to design the next prototype, which is built on the previous prototype. This process continues until the final prototype fulfils all of the user requirements. The final prototype is the final system, unlike "throw-away" prototypes, where small prototypes are used to test out ideas but ultimately are not used as part of the final system.

The motivation for using evolutionary prototyping for this project was that the EMS had to start collecting data before the Imager was complete. The EMS and the Imager were supposed to work automatically for very long periods, because they would be deployed at the SKA site in the

Northern Cape. Although there are technicians at the site, there are no engineers; the system would thus be left unattended for long periods. To ensure the success of the project, various prototypes had to be created to test the functionality before deploying it on site.

4.1.6 Design Decisions

The requirements were collected to gain a better understanding of the functionality of the monitoring system and the users' expectations. The temperature and humidity sensors had to be physically connected to a microcontroller, a Raspberry Pi in the case of this project. Figure 4-1 illustrates the overall overview of the EMS. The Raspberry Pi is responsible for processing readings and producing an output of the readings via Ethernet through secure shell (ssh) to a computer. The computer does the display. Users use the IP address and gain access to the system through ssh, where they can view readings. Users can also access the hdf5 file for the day in which they are interested. The temperature readings are the most important form of storage because they will be used for the ICTP.

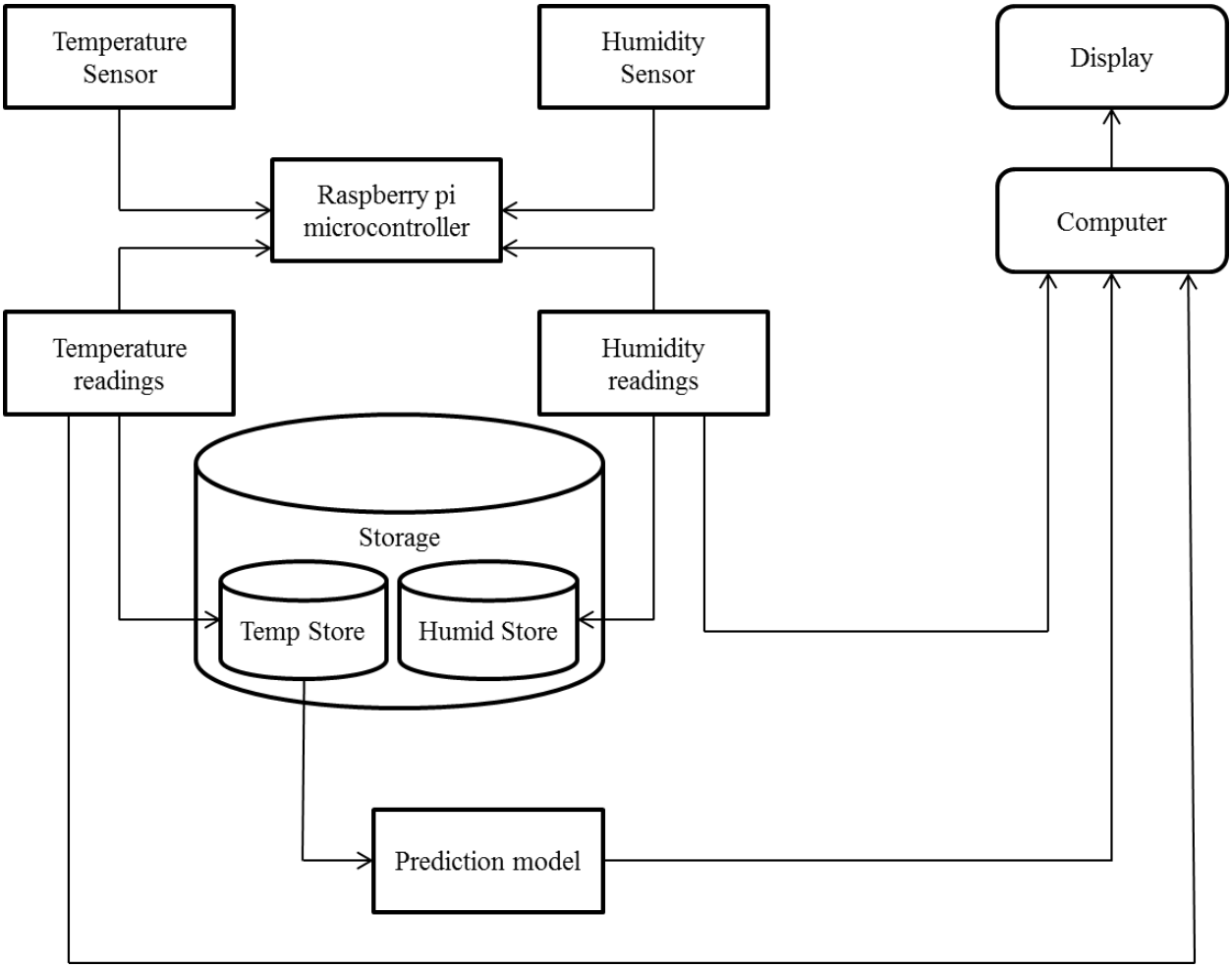


Figure 4-1: Design overview of the prototype system for monitoring the immersion cooling system.

4.2 Implementing the EMS

During the design and implementation of EMS, evolutionary prototyping was used where a series of prototypes were implemented; Sections 4.2.1 through 4.2.3 describe the prototypes as they evolved over time.

4.2.1 EMS Prototype 1

The first prototype was a temperature monitoring systems implemented using the Grove temperature sensor and the Intel Galileo board. The Intel Galileo used Arduino IDE, and the code was written in C++, the exemplary code provided on the Intel website was modified to satisfy our requirements. This system took temperature readings and displayed them on an LCD.

The temperature readings were not stored to a file. Figure 4-2 provides an illustration of prototype 1.

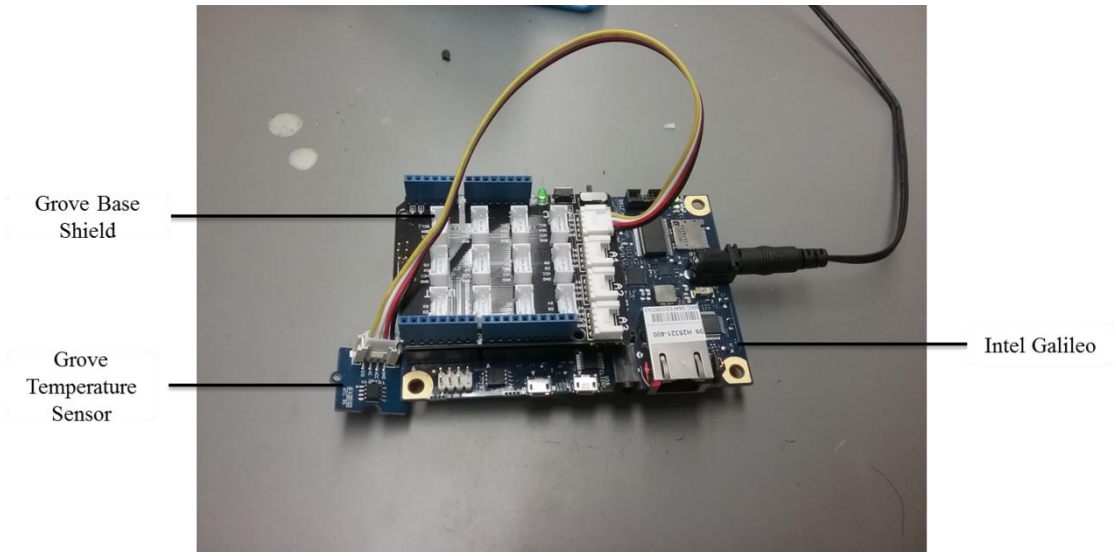


Figure 4-2: Intel Galileo board with Grove sensor connected for prototype 1

The greatest challenge with this prototype was its scalability;a minimum of 60 sensors connected to the chosen microcontroller was needed. The scalability option for this prototype was to use a Grove base shield that connected six sensors at a time; therefore more shields would be needed to get to 60 sensors. The cost of the shields was high, considering that a cheaper option is required. The operating system was also a challenge: the requirement was Linux but the Intel Galileo uses Yocto built Linux image, which is not documented very well. The Intel Galileo moreover had trouble supporting the DS18B20 temperature sensor.

4.2.2 EMS Prototype 2

From Prototype 1, it was evident that a different embedded system was needed, as it had to support a higher number of sensors. For Prototype 2, the Raspberry Pi was used; this ran Linux, supported Python and was well documented – and thus was more advantageous. In this prototype, 20 temperature sensors were connected to the Raspberry Pi using the 1-wire bus. A tub was filled with oil; the sensors captured the temperature readings. Since one of the objectives of this thesis was to ascertain the location of a hotspot, there is need for a heat source. The Tegrak1 board was used as the heat source. Figure 4-3 illustrates prototype 2.

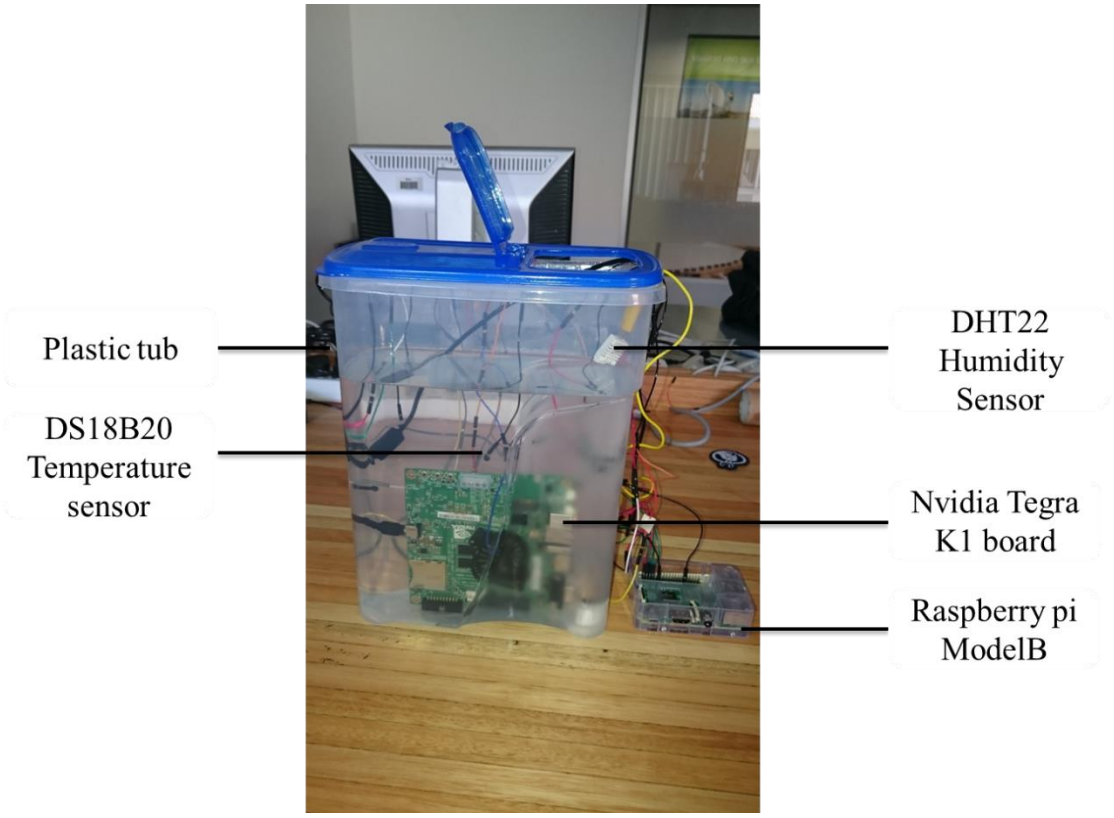


Figure 4-3: An illustration of prototype 2

The challenges with this prototype were that the sensors did not have fixed positions; they were scattered all over the tub, so it was difficult to locate the actual position of each sensor. However, this was an important goal, as highlighted in Section 4.1.1 above. Also, the tegrak1 board was not generating enough heat. A structure that would hold the sensors in place was needed so that it would be easy to identify and assign coordinates for each sensor. Furthermore a heat source that was smaller than the tegrak1 board was needed so that the exact location of the hotspot would be known. There was need for circulation of the oil, thus a pump was needed to circulate the oil to get different variations in temperature. The tub used was not large enough to accommodate 60 sensors; thus a bigger tub was needed, this motivated for a third prototype.

4.2.3 EMS Prototype 3

For Prototype 3, a bigger tub was used; a structure in a form of a scaffold that would hold the sensors in place to identify the exact coordinates of each sensor was needed. There were two options in building the scaffold either build the scaffold with PVC pipes, which were easy to

connect, or build a scaffold using wood. The challenge with using PVC pipes was that they were too thick and their thickness disrupted the flow of the oil. As a result, a 3-dimensional wooden scaffold was built, illustrated in Figure 4-4, the complete scaffold with temperature sensors is illustrated in Figure 4-6, the sensors are 10cm apart; the connectors were designed by means of Thingiverse software, which is used to create 3D printable objects. The connectors were printed with Ultimaker 2, a 3D printer, illustrated in Figure 4-5. The Ultimaker 2 is a 3D printer that allows complex designs of 3D prints. The Direct Digital Control (DDC) pump was used to circulate the oil in the tub to create variations in temperature. DDC pumps are normally used to circulate water, in this study they were used to circulate oil. They have a maximum flow rate of 1000 litres per hour [63]. The pump also has pulse width modulation (PWM) to control the flow by turning the voltage on and off. Figure 4-7 illustrates the approved prototype, this is an improvement from prototype 3 with a bigger tub, a 3D scaffold for the exact locations for each sensor and a radiator for heat exchange.

Figure 4-4: Scaffold with connectors

Figure 4-5: Ultimaker2 3d printer

A plastic tub with dimension length = 59cm, Breadth = 50cm and Height = 34cm was used.

The scaffold that supported the sensors in the mineral oil environment had the following dimensions: Length = 40cm, Breadth = 20cm and Height = 30cm. The temperature sensors were spaced ten centimetres apart. A 20W resistor with the following dimensions was used as a heat source: Length = 6cm, Breadth = 1cm and Height = 6cm.

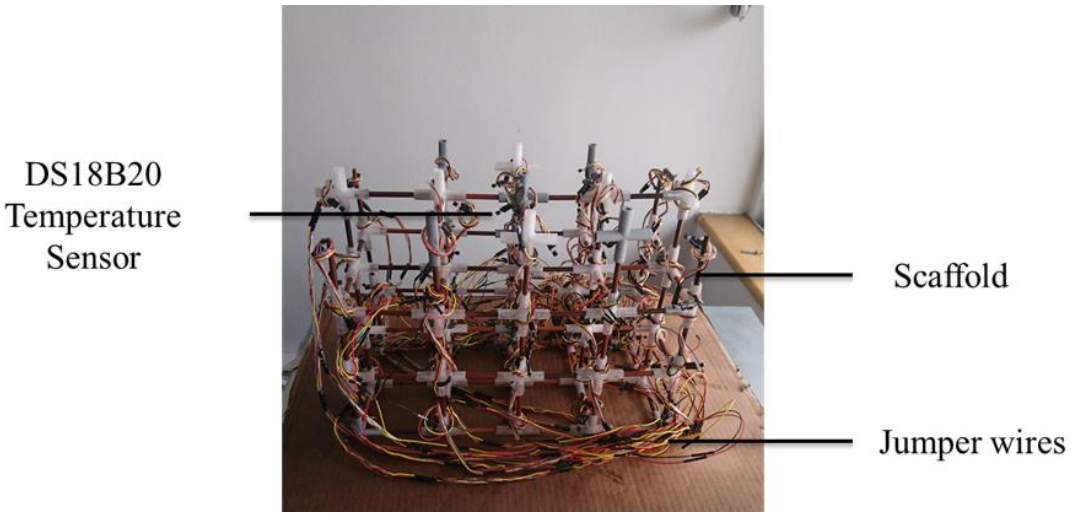


Figure 4-6: A scaffold with sensors

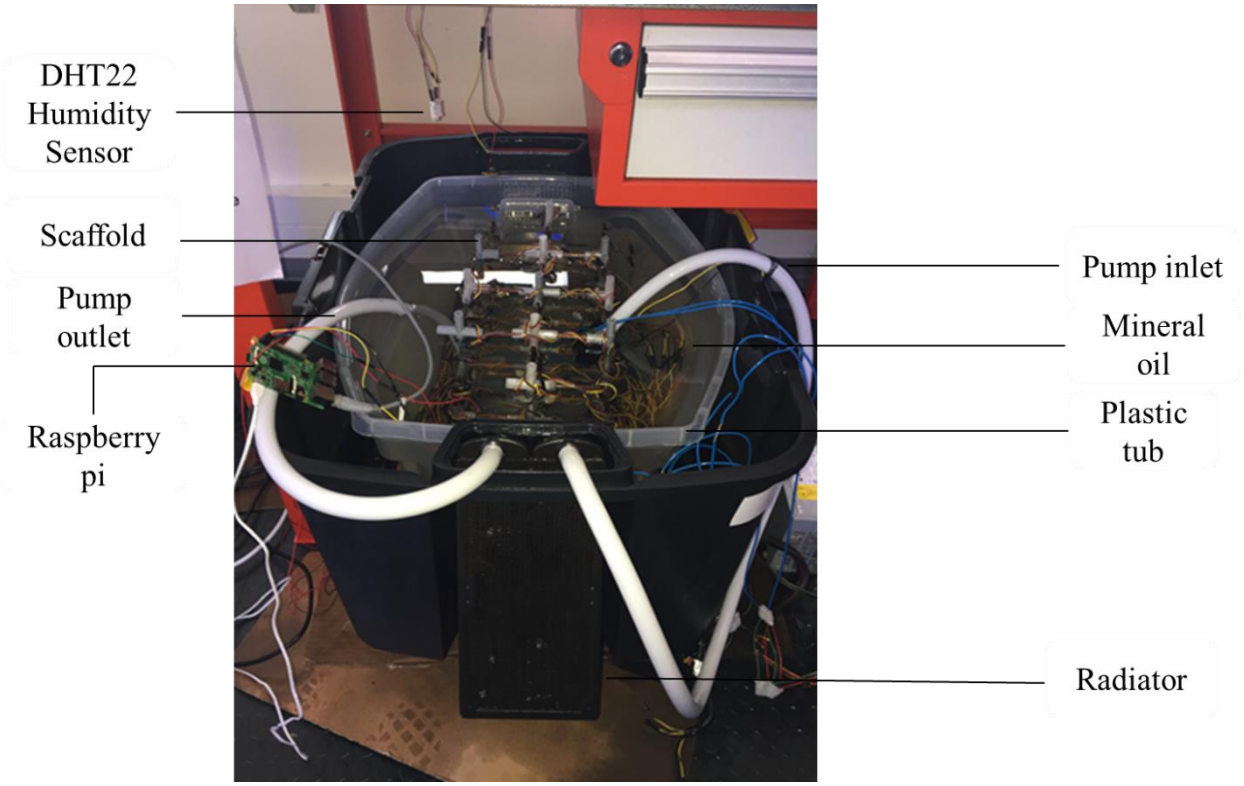


Figure 4-7: An illustration of prototype 3

This section further describes how temperature, humidity and data storage were achieved in prototype 3 since this prototype satisfied the user requirements. Table 4-3 illustrates the improvement from prototype 1 to prototype 3.

Table 4-3: Comparison between three prototypes

Element	**Prototype 1**	**Prototype 2**	**Prototype 3**
No of sensors	4 sensors	20 sensors	60 sensors
Heat source	None	Tegra k1, challenge is the size and the heat produced is not enough	20W resistor
Cable length	10cm	20cm	100m
Pump support	No	No	Yes
Support structure for sensors	No	No	Yes
Size of tub	No tub was used	Could not fit 60 sensors	Could fit 60 sensors

Temperature Monitoring

The DS18B20 temperature sensor was chosen for this project. It communicates over a 1-wire bus, which is a communication data bus system developed by Dallas Semiconductors. This data bus provides low-speed data in the form of signalling power over a single signal wire. It is similar to I2C, though it has a longer range and lower data rates. It is used to communicate with inexpensive devices like humidity sensors, thermometers and other one-wire sensors over a long range. The main advantage is that each device has a unique address, which makes it possible to identify each sensor uniquely. This was very useful for this project, since each sensor would have x, y, and z coordinates attached to its unique identity.

There DS18B20 thermal sensor could be connected to the raspberry pi in two ways, viz. the 1-wire bus or the I2C. The 1-wire bus has one disadvantage, the 1-wire data link acts as a very long antenna, catching interference. The 1-wire bus on the Raspberry Pi is not implemented on hardware, but only as software on GPIO4. GPIO pins on Raspberry Pi are directly connected to the CPU. Interferences caught on this bus are transported to Broadcom SoC, which could be destroyed easily if the data line cable length is longer than 20cm, destroying the SoC destroys the Raspberry Pi.

A minimum of 60 sensors were needed to gather data for the ICTP, the cables were longer than 20cm. Due to this, the I2C to 1-wire reduction was used. I2C was implemented on the hardware and not the software. The reduction allowed the I2C pins on the Raspberry to communicate via 1-wire. With this approach, the cables could be 100m long. The Integrated circuit known as the I2C to 1-wire converter called the DS2482S-100 was used to convert I2c to 1-wire. The converter is unfortunately sold in the SO-8 package, which is almost un-solderable. A schematic on Eagle Cad was thus designed. A board was printed using the Computer Numerical Control (CNC) milling machine, illustrated on Figure 4-8 below.

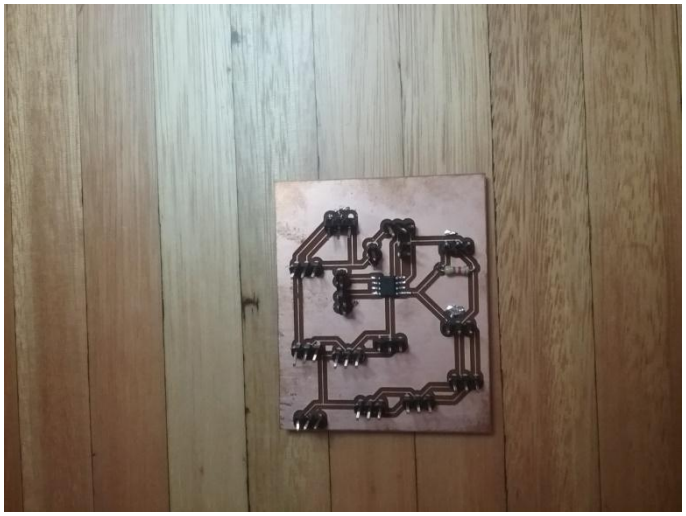


Figure 4-8: DS2482S-100 I2c to 1-wire converter board using CNC Machine

Humidity Monitoring

The DHT22 was connected to the Raspberry Pi; readings were collected using a Python script and a library called DHT22, which was open source and could be used by any developer. Humidity readings were stored in an hdf5 file.

System Storage

The MeerKAT telescope system uses the Highly Definable File version 5 (HDF5) format to store data. Once the EMS was completed, it would be integrated into the SDP, a subsystem of the MeerKAT system and it would also need to use the HDF5 file system for storage. The HDF5 format was thus adopted for the EMS, to maintain consistency with the rest of the MeerKAT system design. The HDF format was developed by the National Centre for Supercomputing. The

development has since been taken up by the HDF Group, which is responsible for the latest version of HDF and its accompanying technologies called HDF5. The HDF5 data model, file format, API, library, and tools are all open source and can be used free of charge.

The HDF format allows users to define the hierarchical data format of their choice, which is stored in one file. The format allows for the creation of HDF5 groups and datasets. A group is a structure containing HDF5 objects, which could be other groups or datasets; each group also contains a set of metadata, which describes that group. The user can define the contents of a group. A dataset is a multi-dimensional array of data along with metadata. HDF5 files have some useful features; they can be compressed with a range of algorithms. This compression can be performed on subsets of the file or chunks of a dataset. This allows access to parts of the file without having to decompress the entire file.

A hierarchical data format that was suitable for our system was designed. The data was arranged into groups: the temperature readings were stored in a separate group as were the humidity readings. Inside each group, was the dataset for each sensor; as stated in the previous sections, a unique ID identified the DS18B20. Each sensor had a dataset containing a 2-dimensional array, with one column being the temperature and the other being the time. HDF5 also allows the user to create attributes; attributes are small datasets that can be used to describe the nature or intended usage of the object or dataset. Each dataset has the x, y and z locations of the sensor attached as attributes on the file.

4.2.4 Software Implementation

Python is the scripting language of choice used by MeerKAT engineers. To maintain consistency with the system, the Python scripting language was adopted for the implementation of this project.

Python is an interpreted language that is interoperable with many other languages and runs on major operating systems, with many wrappers for popular languages, such as MATLAB, Java, C++ and CUDA; this makes it very useful for interfacing with diverse systems. However, Python is slow for large numerical calculations. There are libraries, which can mitigate this problem for scientific applications. The library used by MeerKAT engineers is Scipy. The most fundamental

part of Scipy regarding efficiency is Numpy, which implements N-dimensional array manipulation in C. This allows for efficient calculation when using standard Numpy types; the arrays can contain arbitrary objects; it also allows for easy integration with the database [64].

Figure 4-9 illustrates the Unified Modelling Language (UML) diagram for the different classes and methods that exist within those classes.

Diagram A represents the temperature class and the functions in the temperature class, the code is available on Annexure A; the temperature class has 3 methods:

- Temp_raw () converts the raw temperature readings from the raspberry pi. It is the standard conversion method used for the DS18B20.
- Get_Temp () converts the temperature form the raspberry pi to the preferred method.
- Read_Temp () function is where the temperature data is read and stored.

The temperature data is stored in the HDF5 file; the file is divided into groups. The TempVal group has several datasets; each sensor has a unique dataset divided into two columns, one for the temperature values and one for the time the temperature was read. This is useful for monitoring and training the ICTP. The code is available in Appendix A of this dissertation.

Diagram B illustrates the class that was used to record the humidity data, the code is available on Annexure B. The class has two functions:

- get_ht () is the function that collects readings for the physical humidity sensor; the humidity data is converted in this function. Sections of this code use standard code for the DHT22 humidity sensor they were modified to be specific to the project.
- The humidity () function is where the humidity data is stored in the HDF5 file. The code is available in Appendix B of this dissertation.

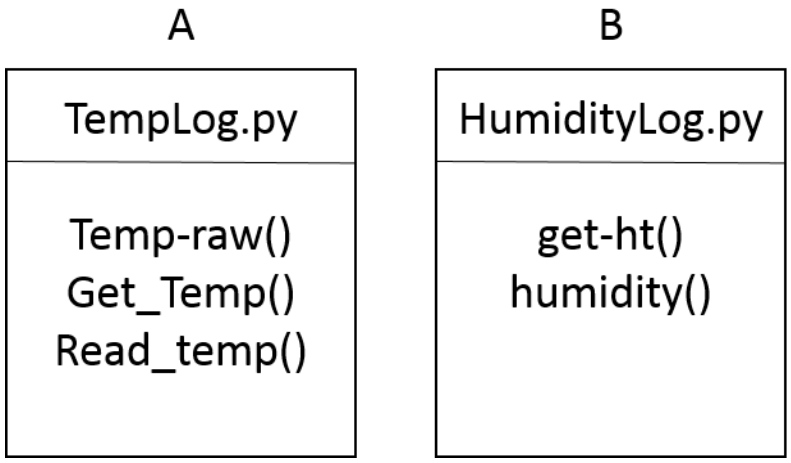


Figure 4-9: UML diagram for the temperature and humidity sensors

4.3 Chapter summary

During this chapter the design and development of the EMS was discussed where goals, users, user requirements, design approach and design decisions were specified. Furthermore, the chapter reports on the implementation of the EMS. Evolutionary prototyping was used, the chapter reports on the evolution of prototype 1 to prototype 3. The chapter concludes with software implementation.

5 Design and Implementation of the ICTP

The aim of this chapter is to explain the method followed during the design and implementation of the Immersion Cooling Temperature Predictor (ICTP). The aim of the ICTP is to predict the temperature in a location where there is no physical sensor. This is an estimation problem and can be solved in various ways; the method chosen is the Gaussian Process Regression (GPR) because it produces a simple, accurate model that produces predictions and uncertainty; GPR is also flexible to incorporate the physical setup of the experiment. This chapter details the GPR used, the basis behind using it, the choice of covariance function and the training of hyperparameters [65].

5.1 Designing the ICTP

The goal of the ICTP is to predict temperature in locations where there is no physical temperature sensor. The ICTP model was built using GPR.

The aim of Gaussian Process Regression (GPR) is to estimate an output value at a previously unseen location (the test point), given known measurements at sensor locations (the training set containing N points, i N). Let the measured values be stacked in an N-dimensional vector y and K be the NxN covariance matrix as a function of x and θ. The output value is then estimated by the posterior mean

$\bar{}$ = E [$y_*|X$ = … (eq. 2)

Where is an N-dimensional column vector with ith element given by C (x).

Let k_{ij} be the element of the covariance matrix and is the covariance function that describes how correlated the fitted function is between any two locations in space given by $(x_i;\ x_j)$. The covariance function is also known as the kernel.

The uncertainty of is represented by the posterior variance

E [(y $\bar{}$)^2| X, θ, x_*] = C (x) … (eq. 3)

Note that the uncertainty does not depend on the actual sensor measurements of y. This implies that the accuracy of the ICTP is determined only by the sensor accuracy and the placement of the sensors and is therefore known before it is switched on. Additionally, the accuracy can be improved by changing the sensor locations. The first step to performing GPR was to collect the data and to choose the prior distribution. A prior distribution involved selecting a form for the mean and covariance function. The prior mean and covariance represent the physical setup if the system, thus if the data did not represent the physical system well, then the model would perform poorly and predictions are incorrect. In the context of our problem, thus the GPR was used effectively to combine discrete sensor data into a continuous model that estimated the temperature of an unknown location. Additionally, the GPR model supplied information about the uncertainty of the predictions made.

5.1.1 Choosing a Covariance Function

Covariance functions encode the form of the function that is being modelled, e.g. the smoothness of the function being modelled. For the purpose of our study a covariance function explains how temperature function correlates at various points in space. Consider two input points or locations x and $x_j$with corresponding observed values y_i and y_j. If the inputs x_i and x_j are close to each other, a stationary and isotropic covariance function is used, the expectation is that y_i and y will be close as well.

The first step was to select a covariance function, also known as a kernel. Some kernels were smooth for modelling physical environments, such as the squared exponential and quadratic kernels. The Matérn class with ν = 5/2 was adopted for this project, as it is recommended for modelling many physical phenomena and interpolating temperature sensors [66]. The kernel contained hyperparameters that needed to be learned from the data; the hyperparameters were adjusted to fit the kernel to the specified dataset. The covariance function between inputs x and x_j is given by $C\left(x_i;\, x_j\right)$. If the covariance is isotropic, it is only a function of the Euclidean distance $r = \left\|x_i - x_j\right\|$ between the inputs and can be instead written as $C(r)$.

The equation for the Matérn kernel given by

$C(r)$ $\left\{ \right.$ $\left(1+\frac{\sqrt{\quad}}{\quad}\right)$ $\left(-\frac{\sqrt{\quad}}{\quad}\right)$ ... (eq. 4)

The Matérn kernel contains two hyperparameters: the lengthscale L and signal variance $\sigma_f{}^2$, however you can add a third parameter that represents the measurement noise of the sensor known as the noise variance $\sigma_n{}^2$. The lengthscale determines how close two points have to be to influence each other significantly and describes how smooth a function is. A small lengthscale value means that function values can change quickly; large values characterize functions that change slowly, as illustrated in Figure 5-1 below.

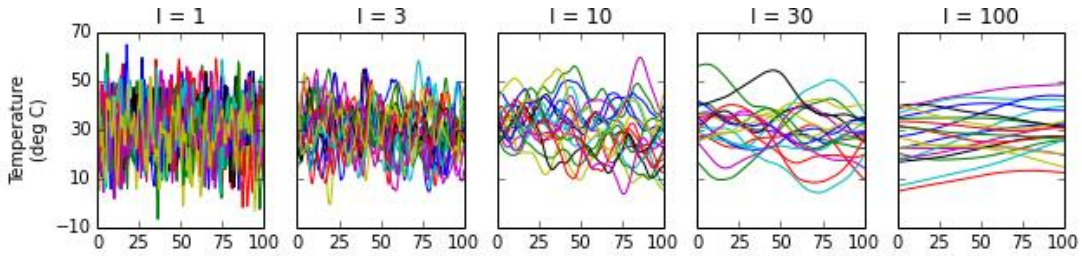


Figure 5-1: An illustration of the effect of a lengthscale on Temperature data

The signal variance $\sigma_f{}^2$ constrains the amount by which a function can change for a given L and separation between points in the domain. A small value of $\sigma_f{}^2$ means that the function is closer to the mean, whereas a larger signal variance means there is more variation from the mean. A large signal variance is not preferred, because it allows the model to chase outliers. The noise variance $\sigma_n{}^2$ is the measurement noise of the sensor that is uncorrelated, also known as the additive white noise.

5.1.2 Learning Hyperparameters

Learning the hyperparameters for a GPR particularizes the chosen covariance function to a given data set. This is called the training step in the GPR. Learning the best hyperparameters is an optimization problem in the 3-dimensional $\theta = (\sigma_f{}^2, \sigma_n{}^2, L)$ hyperparameter space. Maximizing the evidence gives the best set of hyperparameters for the GPR model.

Let K be theNxN covariance matrix, with elements given by $(x$ $)$where 1<= i, j<= N

The log likelihood of the measurements, given the locations and model parameters, is given by

$$\log p(y|X,\theta) = -\frac{1}{2} y^T K^{-1} y - \frac{1}{2}\log|K| - \frac{n}{2}\log(2\pi) \quad \text{... (eq. 5)}$$

where K is a function of

X is the set of N training inputs – that is the set of locations at which sensor readings are taken. We define y as the set of training outputs, i.e. the set of temperature values recorded by the sensor at a specific location. Maximizing the evidence gives the best set of hyperparameters for the GPR model.

5.2 Implementing ICTP

ICTP was implemented using GPy, a GPR framework written in Python, from the Sheffield machine learning group [67]. The most common approach in GP is to have a default mean of zero [71]. Following this approach, we declared our mean as zero and trained the hyperparameters ($\sigma_f^2, \sigma_n^2, L$). However, both the L and σ_f^2were too large and did not match the changes across the spatial dimensions. The high lengthscale and variance were affected by declaring a mean of zero. This approach works for datasets that have negative values; however, since we never had a temperature of zero or below zero, this approach did not work for our problem.

On the next iteration we assumed the 95% percentile: the confidence interval is from 10-50 degrees, given a mean of 30 degrees. The mean was calculated using various datasets. The calculated mean was used to fix the mean of the model $\sigma = 10$, making $\sigma_f^2 = 100$, we followed the similar approach for the noise variance (σ_n^2) based on expected sensor ranges provided on the datasheet [36]. It was necessary for us to fix the parameters due to the number of temperature sensors and because we did not have enough data points.

The lengthscale (L)was the only parameter left to learn: we trained on different datasets and found 18cm as the lowest lengthscale in the tub (dimensions of the tub in cm were 59X50X34) as specified in Section 4.2.3. We wanted the shortest lengthscale that matched the changes across spatial dimensions. We tried different lengthscales for L_x, L_y, L_z but all were around 18; there

was not much difference. Once all the parameters were obtained, we stopped training the parameters and fixed them according to the outcomes. The disadvantage of this approach was that it restricted the chosen model. The next step involved training the strength of the model and testing the accuracy of the model so that we could start with predictions. The information about the strength of the model is discussed in Section 6.3.

The following approach that was followed for the experiment involved the series of four steps shown below:

1. First, find the range of the data and fix the variance according to the range
2. Find the mean of the data and fix the mean according to the calculated mean
3. Fix the noise variance according to the datasheet of the sensor used
4. Train the lengthscale using multiple datasets until you find the one that matches changes across special dimensions.

We deduced that training fewer hyperparameters resulted in a more accurate model, results in Section 6.3.

5.3 Chapter Summary

The chapter discusses on the method followed during the design and implementation of the Immersion Cooling Temperature Predictor (ICTP). The chapter explains what GPR is and how it was used, the basis behind using it, the choice of covariance function and the training of hyperparameters. The next chapter outlines the results for the Environmental Monitoring System (EMS) and Immersion Cooling Temperature Predictor (ICTP).

6 Results

The aim of this chapter is to present results for this study. The chapter is divided in three sections as follows: Section 6.1 provides analysis of results for the Environmental Monitoring Systems (EMS), Section 6.2 presents a series of experiments test whether a hotspot can be detected in an immersion cooled environment and Section 6.3 provides analysis of results for the Immersion Cooling Temperature Predictor (ICTP). Testing of these subsystems follows the testing outlined in Section 3.8 and 3.9 (see Chapter 3).

6.1 EMS Results

This chapter reports on the testing of the EMS, following the approach described for step 8 of the methodology (see Section 3.8).

6.1.1 Test 1: Accuracy of the sensor in the coolant

The purpose of this experiment was to test the accuracy of the sensor in the coolant.

6.1.1.1 Experimental setup:

This experiment made use of one digital probe and a sensitive handheld digital thermometer. The aim of this experiment was to test the accuracy of the temperature sensors. This experiment set out to investigate the following research questions:

- Does the temperature from the thermometer coincide with the temperature from the temperature sensor?

The experiment was done by placing a sensor at point A and a handheld thermometer at point B; thereafter, a measurement was taken. Then the coolant was reset to ambient, the positions A and B were swapped, and the test was repeated to see whether the thermometer and the sensor gave the same results.

6.1.1.2 Control:

Ensure that the thermometer used ranges up to at least 100 degrees Celsius and that the accuracy of the thermometer is 0.5 degrees. The tub used for this experiment has the same dimensions as those specified in section 4.2.3.

6.1.1.3 Results:

The temperature of the sensor and the temperature of the thermometers coincided. This experiment was done using one temperature sensor; the assumption was that the results would be the same for all the other temperature sensors.

6.1.2 Test 2: System's ability to function without human intervention

The second test was to see if the system could maintain correct operation for long periods, correct operation is running and visually reporting errors; this was an important test, since the system would be left at the Karoo site without any human intervention for lengthy periods. To test the system, we ran a 20-hour experiment to test if the system can remain stable for long periods. According to Figure 6-1, the system was stable, also at level 1(z=10) there is a dead sensor, the system visualised this making it easy for the user/ operator to know.

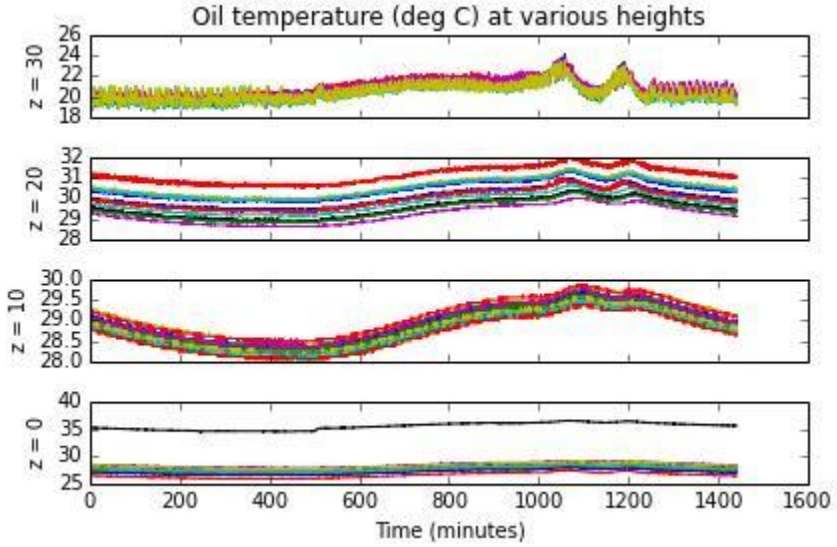


Figure 6-1: A 23-hour experiment to test the robustness of the EMS

6.1.3 Test 3: Fault tolerance of the system

The aim of the third test was to investigate how the system handled faults; in particular to see what happens when one of the sensors was removed while the system is running. Would the entire system shut down? Does the system continue running with the rest of the sensors? Would it give information about the missing sensor?

To test this, one sensor was removed from the setup; as illustrated in Figure 6-2, the system continued to run. The system also provided information about the missing sensor, informing the user about the unique identity of the sensor. This made it easy to identify and locate the missing sensor. The system was designed such that the x, y and z coordinates of the sensors were stored in a configuration file, making it easy for the user to know exactly which sensor was missing. Figure 6-3 is an example of the configuration file: in the case of sensor 28.37EC46060000, which was reported missing by the system, the operator/technician would go to the configuration file and determine that the sensor was at coordinates x=20, y=0 and z=20. The operator would then go to the specified location.

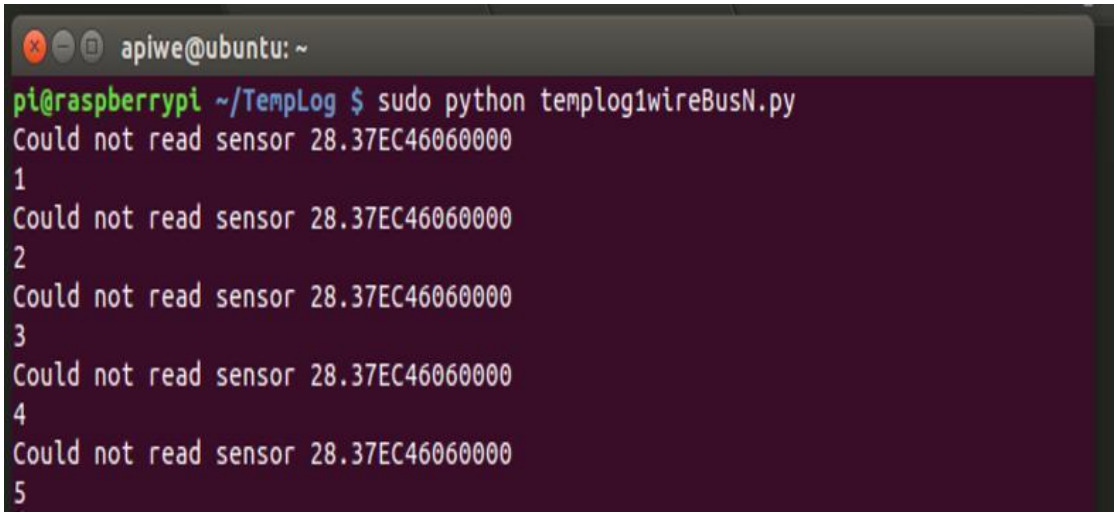


Figure 6-2: An experiment where a sensor was removed to test the fault tolerance of the system.

Figure 6-3: The configuration file, where the individual locations of the sensors are stored.

6.1.4 Test 4: Data capturing system

In this test, it was investigated whether the data capturing system captured data to an HDF5 file, the data included the time and the x, y and z coordinates as attributes in the file.

As mentioned in the section 4.2.3, the data was stored using HDF5 files. The data in the file was represented in groups, and inside the groups were datasets; each sensor had its own dataset and inside the dataset were the temperature values and the timestamp. An attribute was attached to each sensor, and these attributes contained the x, y and z coordinates of the sensor. Figure 6-4 provides an illustration of the hdf5 file. This was useful in testing the model to predict the temperature in a specific location. A better image is available on appendix E of this study.

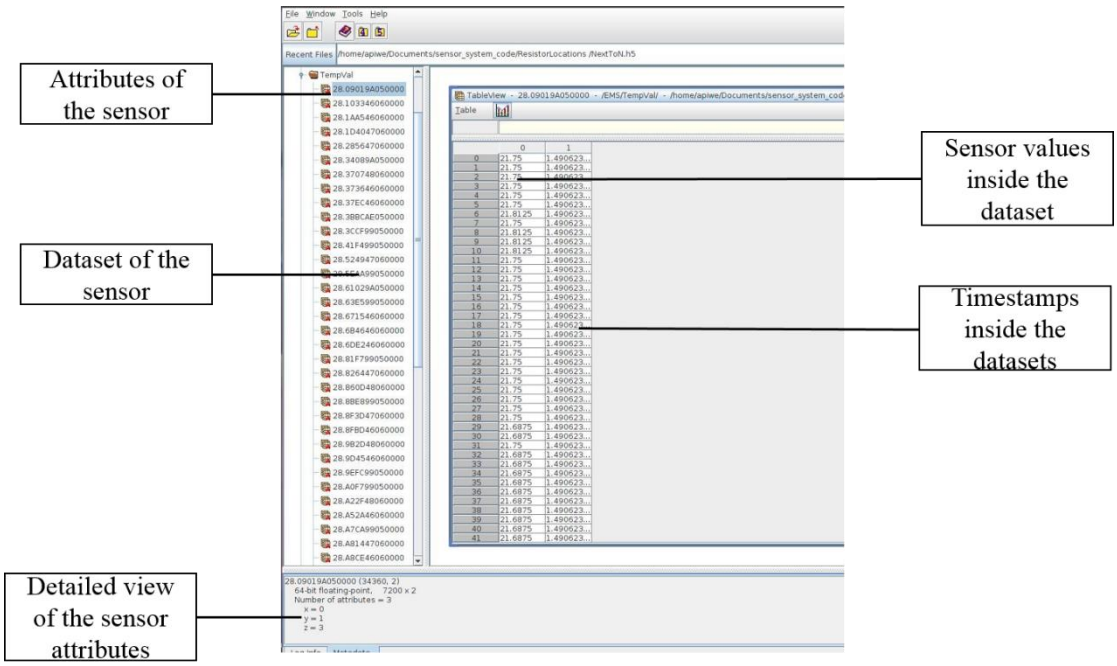


Figure 6-4: A representation of the HDF5 file with datasets and attributes in each file.

6.2 EMS and immersion cooling

The aim of this section is to experiment whether hotspots can be detected in an immersion cooled environment. A hotspot is defined as a location in a contained environment, where the temperature is higher than the standard temperature in the rest of the environment. The section thus presents a series of experiments to demonstrate the behaviour of the oil, depending on where the hotspot was placed.

The experiments were done by means of the EMS setup described in Section 4.2.3. An illustration of the physical representation of the scaffold is available in Figure 4-6 and the finalized representation is available in Figure 4-7. Appendix D is a configuration file with the unique sensor ID and x,y and z locations of each sensor. The EMS was used to capture and store temperature values for these experiments. The experiments mimic a real world setup, where there is a hotspot in a vat or in an immersion-cooled container. The 20W resistor was used as the heat source for all of these experiments; the dimensions of the resistor are available in Section 4.2.3. for the purposes of a clear demonstration. A single sensor was selected as the target sensor; this is the sensor that would be observed at a particular time. The target sensor is at location x=10, y=20 and z=10, all the values are in centimetres. The heat source (resistor)

could be below, above or next to the target sensor, depending on the type of experiment and the objective/aim of the experiment. The 20W resistor will be referred to as the heat source and the target sensor, as described above, is the sensor that is currently observed. The measurements were collected over an hour: initially, there was no heat source; after 15 minutes, the heat source was switched on and allowed to run for 30 minutes; it was turned off at the 45-minute mark. The time (1 hour) for the experiment was arbitrarily selected. The sensor sampling program automatically captured readings every 8 seconds. Initially, samples were taken once a minute, but due to a rapid change in the temperature of the oil, it was reduced to 8 seconds.

6.2.1 Test 1: Testing how heat disperses

The aim of this experiment was to understand how the heat dispersed within the container. The aim of the experiment was to answer these questions:

- Once the heat source is switched on, do all the temperature sensors heat up at the same time?
- Do they cool down at the same time when the heat is turned off?

Knowing whether the sensors heated up at the same time helped us detect whether the heat was dissipating in the oil or not. For instance, the heat could move in one direction and not in another. This might be caused by the location of the heat source relative to the sensors, which were not affected by the heat source.

Results:

Figure 6-5 illustrates the results of the experiment; the heat source was placed at layer two (z=20). According to the top layer sensors heat up when the heat source is switched on and cool down once switched off roughly the same time. Level 3(z=30) takes up to 1 minute to respond, colder levels take up to 10 minutes to respond, this is not clearly shown on Figure 41, because the change is minimal. According to Figure 6-5, the heat moved to the top layer. The temperature of all the sensors at the top layer increased once the heat source had been switched on and that it only decreased once the heat had been switched off.

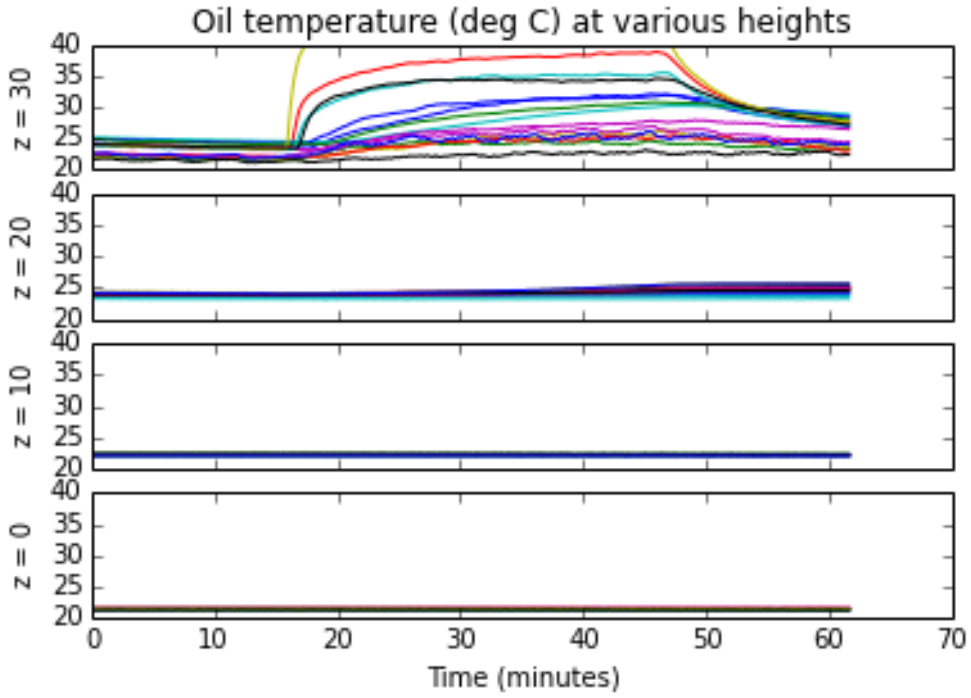


Figure 6-5: Measurement of the mineral oil temperature at heights varying from 0cm to 10cm

Figure 6-5 illustrates the spread of heat vertically while Figure 6-6 illustrates the spread of heat horizontally. The location of the resistor is indicated by the black box and is placed next to the target sensor (x=10, y=20cm and z=10cm. Interestingly the temperature of the second layer (z=20) is not affected as much as the third layer (z=30) by the heat. The heat flows to third layer neglecting the second layer. The temperature of the location above the target sensor (the sensor at location x=10, y=20cm and z=30cm) is approximately 5℃ higher than the temperature at the location where the heat source is placed. The temperature at the layer below the heat source (z=0) does not show any effect, this is expected based on the example of the boliling water. Colder oil moves to the bottom, also it might not be affected because the experiment was conducted for an hour, it might be that the oil needed more time to transfer the heat. Transfer of heat depends also on the viscocity of the oil and the rate at which the oil tranfers the heat.

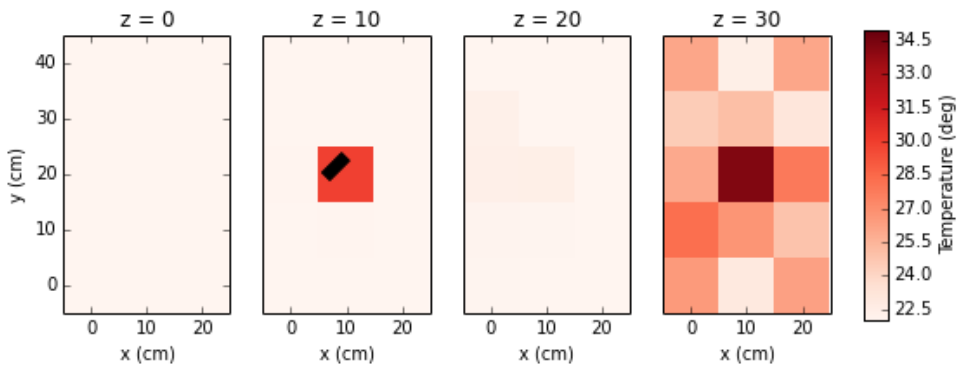


Figure 6-6: An experiment of heat transfer using a 20W resistor

6.2.2 Test 2: Detecting the presence of a hotspot

The aim of this experiment was to detect the effect of the heat source. To illustrate this; the heat source was placed in different locations to test how it affected the other sensors around it: first below the target sensor (layer zero), then above (layer three) and lastly next to the sensor (layer one). The aim of this experiment was to check how far from a sensor and the heat source needed to be before it was detected and how far it needed to be before it was no longer detected.

Results when heat source was placed below,above and next to target sensor

Heat source placed below the target sensor

According to Figure 6-7, the hotspot was minimally perceived by the target sensor: the target sensor is represented by the purple bump, and the effect of the hotspot is shown on the layer above the target sensor, which is represented by the light blue bump. This experiment, like the previous experiment, illustrated that heat moved to the top, and had < 0,1% impact on the other layers, even though the heat was coming from the layer below the sensor. There were minor changes in the temperature of the oil shown in other layers roughly <0.01 degrees centigrade, but very clear changes in the top layer roughly >10 degrees centigrade. The temperature at the bottom of the vat changes slowly, within a band of 0.5 degrees centigrade which is quantised to a value of the resolution of the sensor

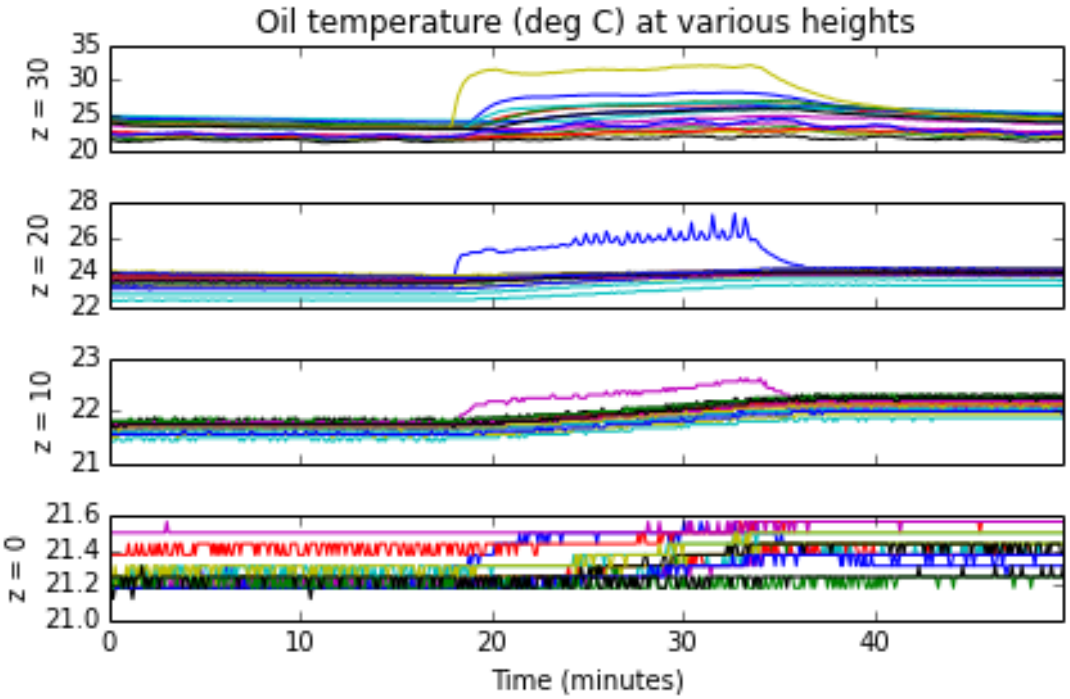


Figure 6-7: The measurement results of the heat source being placed below the target sensor

Heat source placed above the target sensor

Figure 6-8 illustrates the heat source placed above the target sensor. According to Figure 44, heat travelled to the top from layer two, where the heat source was placed. There was a slight increase in temperature in the layer where the heat source was placed, of about five degrees; the highest temperature effect was represented in the top layer.

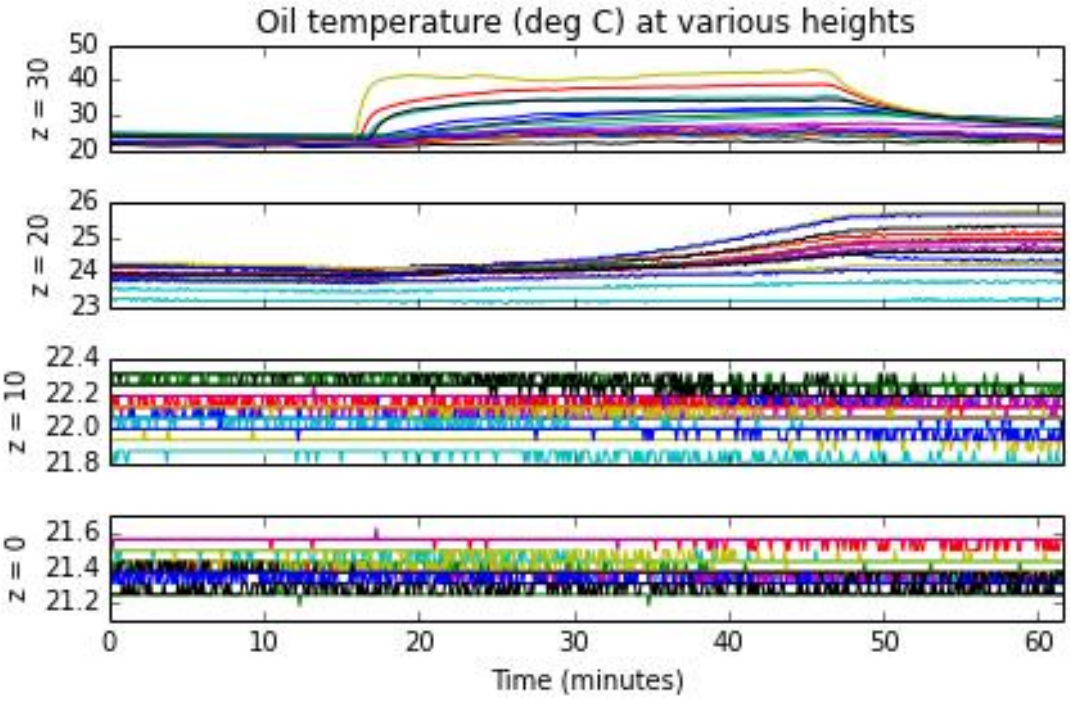


Figure 6-8: The measurement results of the heat source being placed above the target sensor

Heat source placed next to the target sensor

Figure 6-9 illustrates the heat source placed next to the target sensor. According to the illustration, the hotspot was clearly represented, as illustrated by the purple bump in the graph. This is the location of the target sensor (x=10, y=20 and z=10). The hotspot caused the temperature of the target sensor to increase; the heat source was 20 mm from the target sensor. As soon as the hotspot was moved to 30mm from the sensor, the effect of the heat was no longer visible on the sensor, but it was only visible on the sensors in the top layer. The green bump in the top layer is the sensor that is directly above the target sensor (x=10, y=20 and z=30).

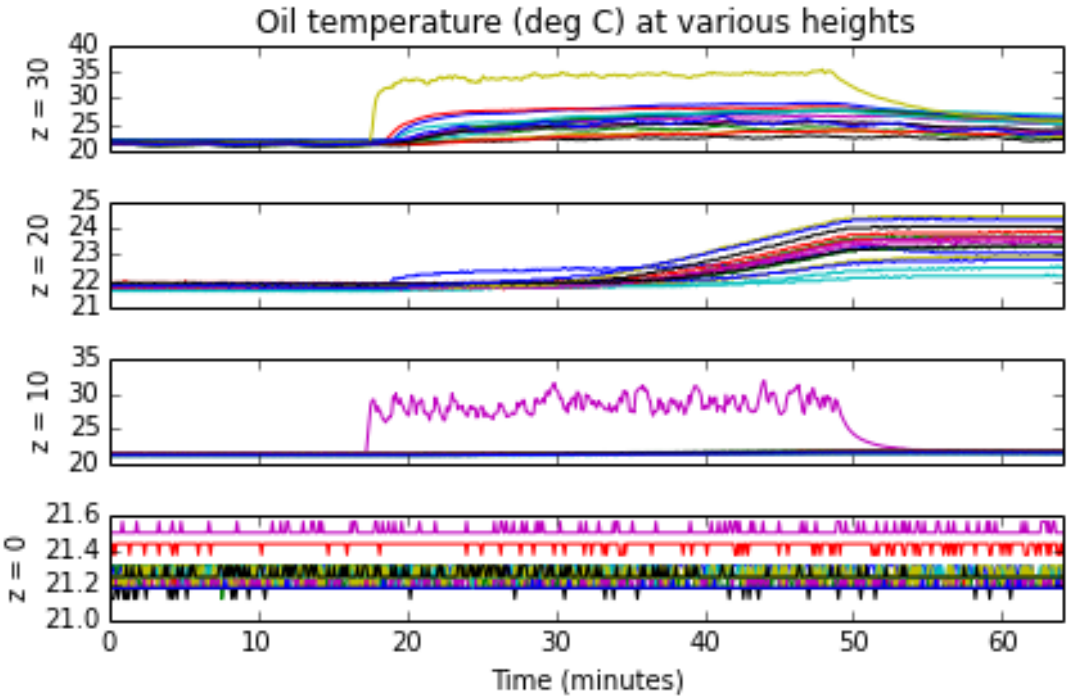


Figure 6-9: The measurement results of the heat source being placed next to the target sensor

6.2.3 Test 3: Detecting the time it takes for a hotspot to be detected

The aim of this experiment was to determine how long it took the heat to spread in the oil. The following questions were thus asked:

- How long does it take for heat to increase once the resistor is switched on?
- How long does it take for heat to decrease once the resistor is switched off?

Results:

According to Figure 6-9 above, the heat source was switched on after 15 minutes; according to the samples, it took a few seconds for the heat to become visible on the target sensor. The top layer reacted similarly to the target sensor. Layer two started to respond 25 minutes after the heat source had been turned on. The time was also affected by how much heat the heat source was giving off, i.e. a resistor with 10 Watts would take longer to be detected than a resistor with 20 Watts.

6.2.4 Test 4: Determine if a single overheating source can be detected among multiple other sources at nominal temperature

If there were multiple warm spots and one hot spot, would the hotspot be detected or would an ambient temperature be established, making the hotspot undetectable? What were the spaces between heat sources, and between sensors or heat sources for this to work?

Experiment setup:

Three 10W resistors connected in parallel were switched on 5 minutes after the experiment had been started and left to heat up the tub. The resistors were placed at z = 10. After 31 minutes, a 20W resistor placed at z = 2 was switched on, to tests if the effect of the heat from this resistor would be seen or not.

Results:

Figure 6-10 illustrates the results of Test 4: the nominal temperature was 25º Celsius, and when the resistor was switched on, this temperature increased after 3 minutes. This effect like other experiments was visible at the top layer, indicated by the green bump, this is the sensor above the target sensor. This meant that the hotspot was detected, even though there was already an ambient temperature established. As soon as the resistor was switched on, it increased the ambient temperature. The time it takes for heat to disperse is also dependent on the viscosity of the oil and how quickly the fluid can direct heat away. This would require extensive heat transfer theory and this is beyond the scope of this thesis.

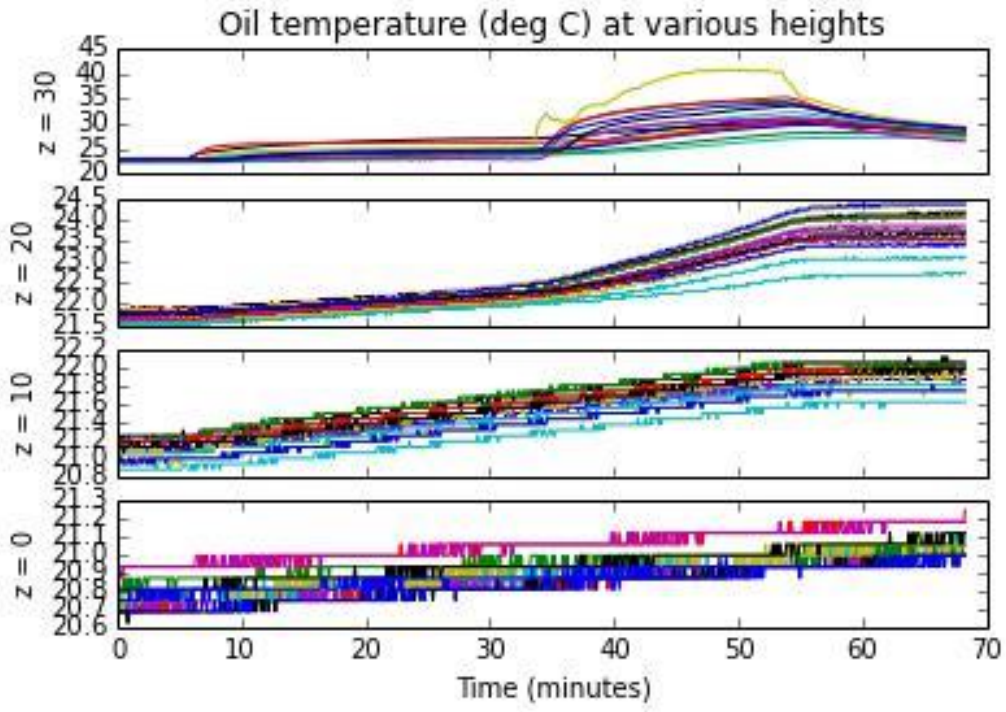


Figure 6-10: Oil temperature at various heights

6.2.5 Test 5: Investigation to determine the extent that the pump affects the temperature of the vat

Experiment setup:

A sensor at coordinate x =10, y=20 z=10 was identified as the target sensor as stated above; the heat source was placed next to the sensor, below the sensor and above the sensor. A pump inlet was placed at level 3 (z=30) and an outlet at level 0 (z=0). The pump pumped oil from level 3, the oil went through the radiator to the outlet and was released at level 0, an illustration of the setup is in Figure 4-7 in Section 4.2.3.

Results:

Figures 6-11, 6-12 and 6-13 show the measured results for Test 5 with the use of a pump. There was a significant difference: even though the heat also moved to the top layer, with the use of the pump, other sensors also started to show an increase in temperature. A comparison of Figure 6-11 and Figure 6-7 shows that, in both instances, the heat source was placed next to the sensor; the pump eliminated the hotspot, and therefore there is no sign of the hotspot. Also the

temperature at layer 0 (z=0) increased from 21℃ without the pump to 27℃ with the pump. One way to avoid the heat being in the top layer could thus be circulating the oil. The pump can further be improved by placing the pump in a good position as well as increasing the flow rate of the pump. The best practice would be to place the inlet of the pump at the top layer and the outlet at the bottom layer. However, experimenting with the flow rate is beyond the scope of this project.

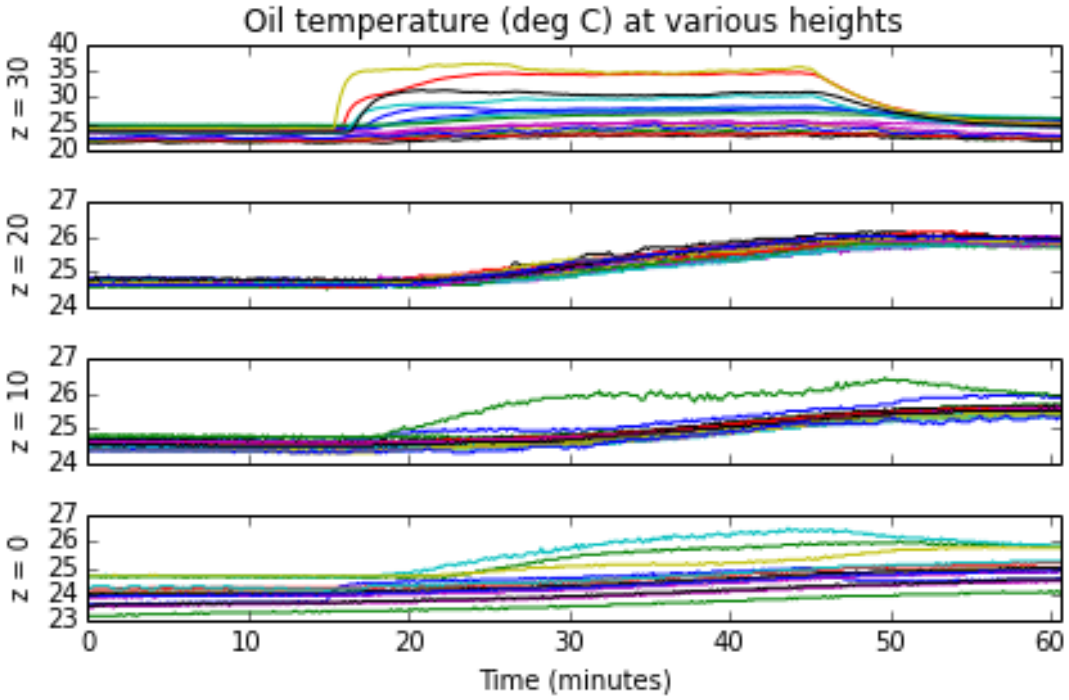


Figure 6-11: The measurement results of the heat source being placed below the target sensor with a pump running

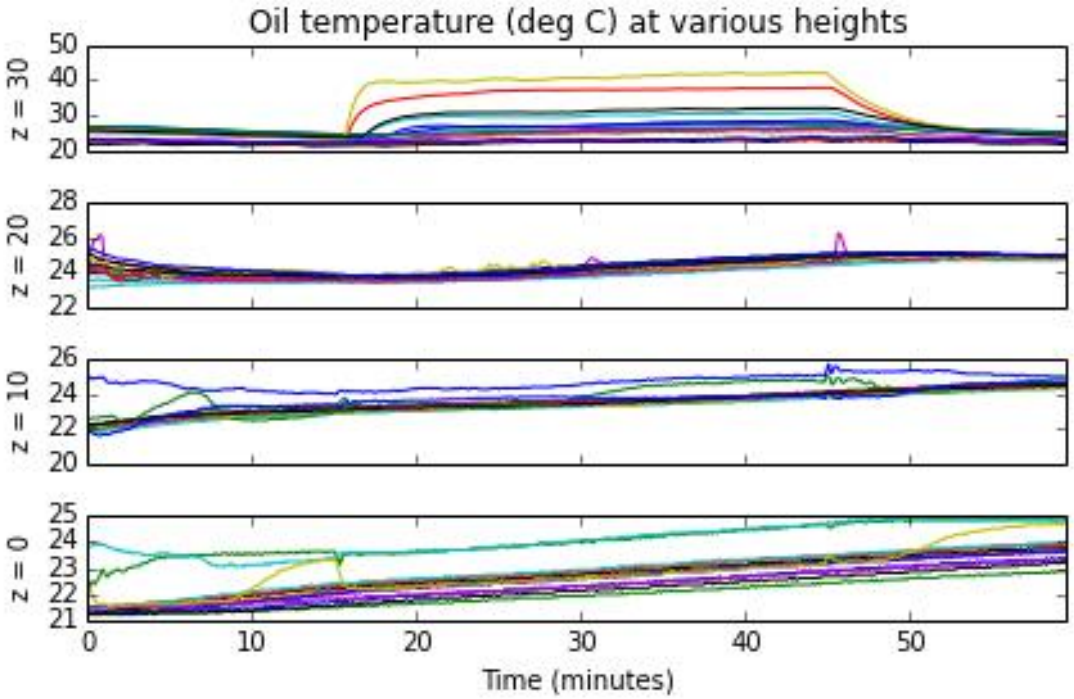


Figure 6-12: The measurement results of the heat source being placed above the target sensor with a pump running

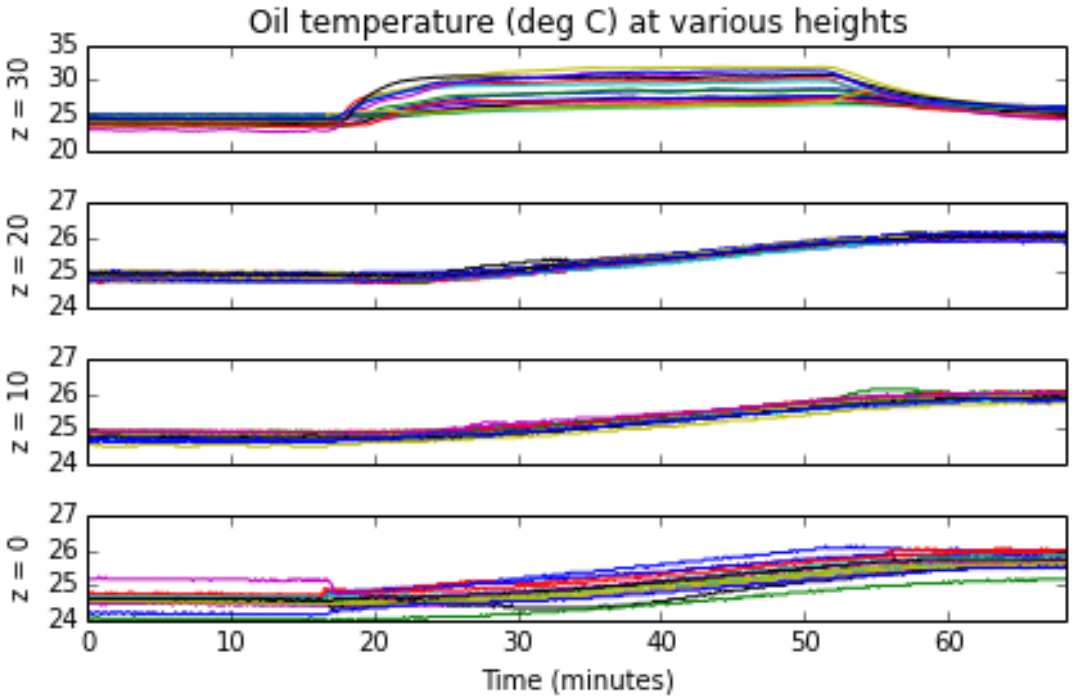


Figure 6-13: The measurement results of the heat source being placed next to the target sensor with a pump running

6.2.6 Conclusion from experiments

It was determined that the heat moved to the top layer; In all our experiments, the temperature sensors all had an initially colder temperature. The question that was, why did the heat not affect layer two on its way to layer three? This might be caused by convection, hot oil rises and colder oil moves to the bottom, it might also be that if the experiments were recorded longer than 1 hour there could have been a change in layer 2(z=20) and possibly layer 0 (z=0). This might also be affected by the grid arrangement and possibly by the wires in the oil, as they might be directing the flow of the oil. There is a small change in temperature in layer two, but not as much as layer three. The sensor above the target sensor has a higher temperature than all the other sensors. These questions might be answered by means of Computational Fluid Dynamics (CFD), but this is beyond the scope of this dissertation. An alternative would be to place sensors at the top layer, this will not provide information about the temperature of the hotspot but will provide information about the location of a hotspot and an indication that there is a hotspot. To get the actual temperature the on-board temperature sensors might be used for critical processing.

6.3 ICTP Results

To test the strength of the ICTP, the data was equally split into two sets, i.e. training and testing, and only a single timestamp was used. Figure 6-14 illustrates the results that were obtained from the test. In the Figure, a 4X3 grid: the x coordinates apply to the column and z coordinates apply to the row, and the y coordinates are contained within each plot. The markers are at the locations of the sensor and are split into two groups, the crosses represent the training data while the dots represent test data. The 2 sigma confidence region ranged between 10 degrees and 15 degrees, which was very large, indicating that the predictions are uncertain. The line is the GPR posterior mean based on the training data, and the grey area is the posterior confidence region. Most of the data was within the confidence region, except for the location where a physical hotspot was placed. It was also deduced that the range increased mostly in the corners, because there were no values that the model could use when interpolating.

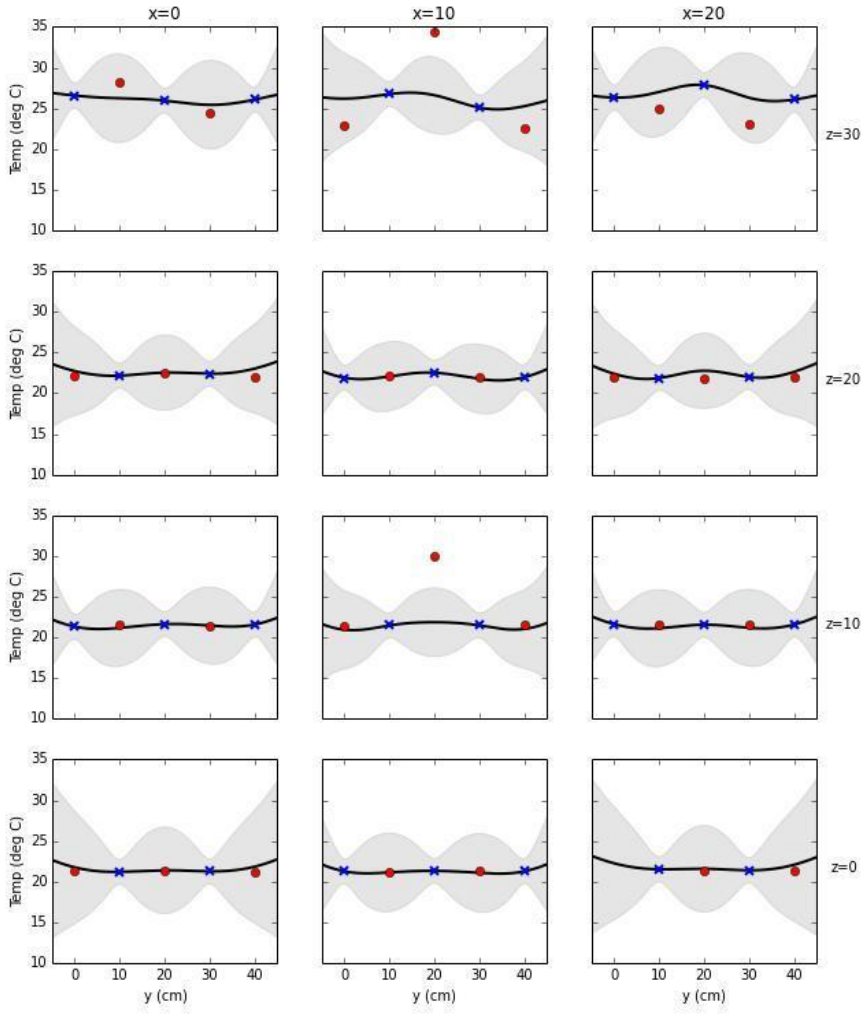


Figure 6-14: An experiment to test the strength of the ICTP where data was equally split into two sets

The error for ICTP is given by

… (eq. 6)

The predicted test value is given by the posterior mean evaluated at the location of the test sensor. Table 6-1 outlines the mean, standard deviation and predicted standard deviation of the error. The mean error provides information about the direction of the error, whether the error is biased or not. According to Table 6-1, the mean error is significantly small meaning that the

IICTP is not biased. The standard deviation of the error also known as root mean square (rms) error is 2.6 which means the temperature are mostly within 5 degrees. The ICTP can provide a prediction about the rms error by combining the posterior variances at the location of the test sensor.

Table 6-1: ICTP error

Error Type	**Value**
Mean of the error	0.0083
Standard deviation error	2.562
Predicted standard deviation for the error	2.439

To reduce the temperature range, an experiment was conducted, where all the data was used, because in the previous experiment half of the data had been removed; it would perform better with more data used. To further test the model, the 59 sensors were used as training and one sensor was left out in a challenging location next to a hotspot. The model had to predict the temperature of the missing sensor. Figure 6-15 illustrates the results of this experiment the model was trained using 59 sensors and had to predict the temperature of one sensor, x coordinates apply per column, z coordinates apply per row and y coordinates are contained within each plot where the dot illustrates the missing sensor. The sensor was within the confidence region.

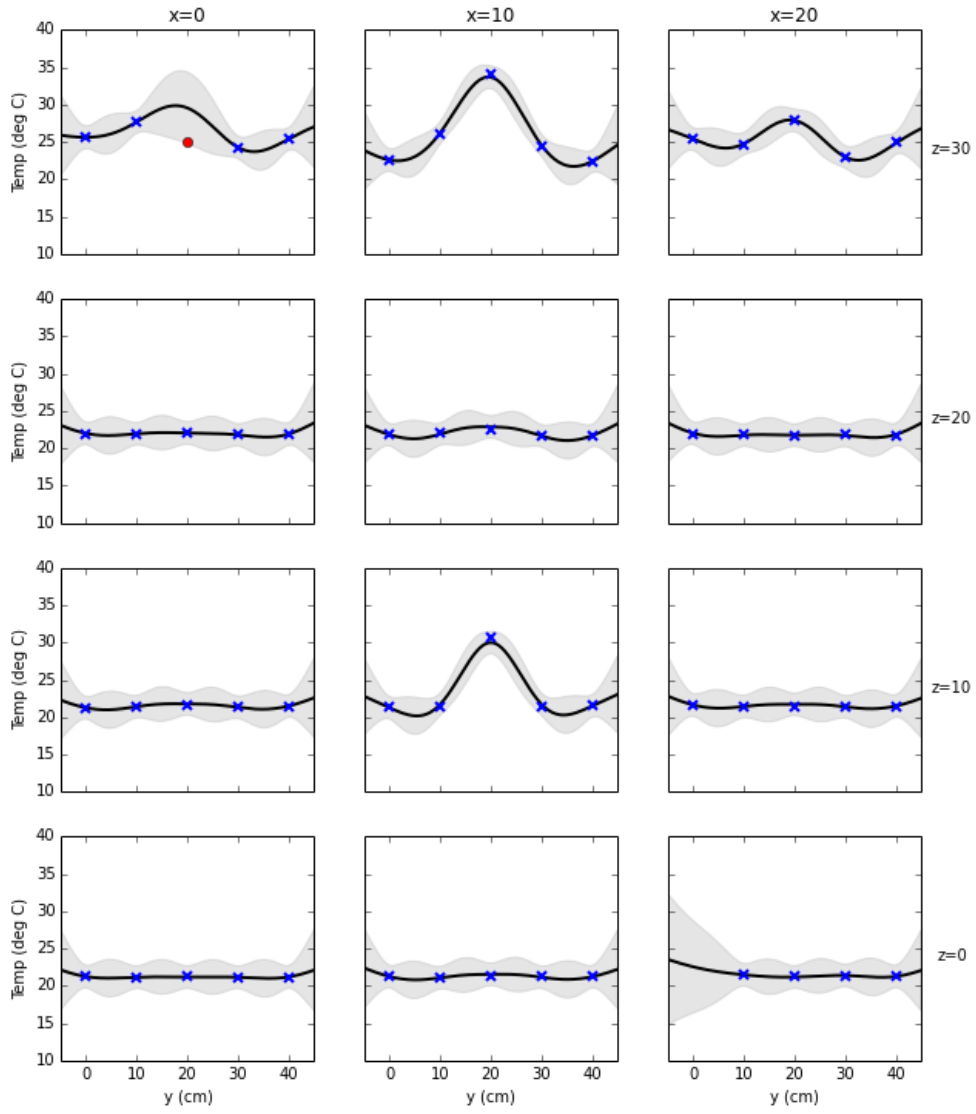


Figure 6-15: An experiment to reduce the temperature range by using all the data.

Chapter summary

The chapter presents result for this study, the chapter is divided in three sections: section 6.1 presents an analysis of results for the Environmental Monitoring Systems (EMS). Four tests were done to validate the EMS, a test to validate the accuracy of the sensor, the ability of the system to function without human intervention, a fault tolerance test and lastly a test to validate that the data capturing system. Section 6.2 presents a series of experiments whether a hotspot can be detected in an immersion cooled environment. The section also presents a series of experiments to demonstrate the behaviour of the oil, depending on where the hotspot was placed. According to the experiments it is possible to detect hotspots in an immersion cooled environment with limitations outlined in the section. Section 6.3 presents results for the Immersion Cooling Temperature Predictor (ICTP), the error of the ICTP is also presented in the section.

7 Conclusion and future work

In this project, an immersion cooling environment was implemented at a small scale. A prototype of an EMS for the MeerKAT imager was designed and implemented; the SDP imager will thus have a system that is able to report on the health of the system. A model, referred to as the ICTP, which can predict the temperature in locations where there is no temperature sensor, was implemented using Gaussian Process Regression (GPR). The EMS and the ICTP model are not specific to the immersion-cooled environment, however; they can be used in air and water cooling systems. The ICTP and EMS together form a system. A list of requirements was recorded for the system using Innoslate, a requirement management tool that allows multiple people to work on the same project and export the project to an MS Word document. Meetings were held with the stakeholders to ensure that the requirements are adequately satisfied. The key stakeholders for this system are the Science Data Processor Team (SDP). Once the stakeholders were satisfied with the requirements, a System Specification document, in other words, a document that best describes the user requirements, was signed by Simon Ratcliffe the technical lead for the SDP team.

A prototype of an EMS is implemented; this system monitors the temperature and humidity of the immersion-cooled environment and stores the data in a file that is easily accessible to the team. The design approach used was evolutionary prototyping, an iterative process that allows for user input throughout the development process, EMS was developed in a series of three prototypes (Section 4.2.1, 4.2.2, 4.2.3). Prototype 3 satisfied the user requirements. Python is the scripting language of choice used by MeerKAT engineers. To maintain consistency with the system, the Python scripting language was adopted for the implementation of this project. The direct users for the systems are members of the SDP team, which need information provided by the EMS. The long-term goal of this system is for it to be integrated into the MeerKAT system. This means that there would be more users of the system, such as the operators which will require data logging from the system. The operators are the people who operate the telescope. In addition, the Control and Monitoring (CAM) Team is responsible for controlling and monitoring the MeerKAT system, so they too would be using the system.

Various experiments were done to test whether hotspots can be detected in immersion cooled environment. The experiments were done by means of the EMS setup described in Section 4.2.3. The EMS was used to capture and store temperature values for these experiments. The experiments mimic a real world setup, where there is a hotspot in a vat or in an immersion-cooled container. The 20W resistor was used as the heat source for all of these experiments; the dimensions of the resistor are available in Section 4.2.3. For the purposes of a clear demonstration, a single sensor was selected as the target sensor, which would be observed at a particular time. The heat source (resistor) could be below, above or next to the target sensor, depending on the type of experiment and the objective/aim of the experiment. One of the experiments include an investigation to determine the extent that the pump affects the temperature of the vat. The Direct Digital Control (DDC) pump was used to circulate the oil in the tub to create variations in temperature. The pump also has pulse width modulation (PWM) to control the flow by turning the voltage on and off, more details about the experiments are on Section 6.2. It was determined that the heat moved to the top layer; all our experiments, the temperature sensors all had an initially colder temperature. The question that asked was, why did the heat not move uniformly to the top layer? This might be affected by the grid arrangement and possibly by the wires in the oil, as they might be directing the flow of the oil. This question might be answered by means of Computational Fluid Dynamics (CFD), but this is beyond the scope of this dissertation. Also the effect of the pump was very minimal; this might be increased by using the PWM to increase the flow rate of the pump.

The ICTP was implemented to reduce the cost of the sensor nodes deployed by reducing the number of sensor nodes; this was done by implementing a model that is able to predict values in locations where there is no physical sensor. A model that predicts the temperature in locations where there is no temperature sensor is implemented, using GPR. The first step to GPR was to select a covariance function, also known as a kernel. The Matérn class with $\nu = 5/2$ was adopted for this project, as it is recommended for modelling many physical phenomena and interpolating temperature sensors. The kernel contained hyperparameters that needed to be learned from the data, the lengthscale L, signal variance $\sigma_f{}^2$and noise variance $\sigma_n{}^2$. The lengthscale determines how close two points have to be to influence each other significantly and describes how smooth a function is. The signal variance $\sigma_f{}^2$ constrains the amount by which a function can change for a

given L and separation between points in the domain (see Section 5.1.1) . The noise variance$\sigma_n{}^2$is the measurement noise of the sensor that is uncorrelated, also known as the additive white noise.

ICTP was implemented using GPy, a GPR framework written in Python, from the Sheffield machine learning group. Learning the hyperparameters and adapting them to the kernel is called training in GPR. We assumed the 95% percentile: the confidence interval is from 10-50 degrees, given a mean of 30 degrees. The mean was calculated using various datasets. The calculated mean was used to fix the mean of the model $\sigma = 10$, making $\sigma_f{}^2 = 100$, we followed the similar approach for the noise variance ($\sigma_n{}^2$) based on expected sensor ranges provided on the datasheet. The lengthscale (L) was the only parameter left to learn: we trained on different datasets and found 18cm as the lowest lengthscale in the tub (dimensions of the tub in cm were 59X50X34) as specified in Section 4.2.3. We wanted the shortest lengthscale that matched the changes across spatial dimensions. To test the strength of the ICTP, the data was equally split into two sets, i.e. training and testing, and only a single timestamp was used. Most of the data was within the confidence region, except for the location where a physical hotspot was placed. We also deduced that the range increased mostly in the corners, because there were no values that the model could use when interpolating.

7.1 Research questions

The project aimed to answer the following research questions.

- Question 1: What implications should be considered when implementing an immersion-cooled environment?
- Question 2: Can the EMS be used to ensure sufficient monitoring?
- Question 3: Can a model be implemented that can predict temperature in locations where there is no physical sensor?

7.1.1 Question 1: What should be considered when implementing an immersion-cooled environment?

The main difference between immersion cooling and other cooling techniques is the use of directly immersing electronic components into a liquid in order to cool them down or to direct heat away from the electronics. The following considerations are relevant:

- Only non-conductive liquids can be used for immersion cooling. Prior to implementing the technique, decisions must be taken about the specific type of liquid to be used. For the purposes of this research, pharmaceutical mineral oil was, which has similar properties to baby oil [70].
- Decisions must be made about the type of container; any type of container can be used, depending on the size (how many litres). For the purposes of this research, plastic was used since this was a prototype. Any metal can be used too, such as stainless steel, aluminium, brass or copper; however, certain materials need to be replaced sooner than others, it might need to conduct research about the type of metal that does not rust easily.
- The type of servers that will be used is important, because hard drives cannot be dipped in oil; the alternative is solid state drives.
- The arrangement of the electrical components, this includes servers, switches, power connectors, Ethernet cables and all the electronic components that will be used.
- Circulation is also important in implementation as seen in the experiments done in Section 6.2. A smaller pump was used, however any pump can be used depending on the amount of circulation needed. Also the flow rate of the pump can be increased.
- One disadvantage of immersion cooling is that it is messy, when implementing such a system a mechanism on how to keep the area clean is necessary.

7.1.2 Question 2: Can the EMS be used to ensure sufficient monitoring?

The EMS can be used to monitor the health of a system as seen in tests done in section 6.1; sensor networks help detecting temperature in certain location. The system developed monitors the humidity and temperature and also provides information about missing sensors and location of the missing sensors.

7.1.3 Question 3: Can a model be implemented to predict temperature in locations where there is no physical sensor?

Yes within certain ranges between the heat source and the sensor, as soon as the hotspot is 30mm from the sensor the effect of the heat is no longer visible on the sensor, it works for situations where the sensor is less than 30mm away from the hotspot as seen in Section 6.2 . This is true for an immersion cooled environment,it might be different in air cooling. An alternative would be to place sensors at the top layer, this will not provide information about the temperature of the hotspot but will provide information about the location of a hotspot, particularly x and y locations. To get the actual temperature, the on-board temperature sensors might be used for critical processing. The ICTP implemented in this project predicts temperature in locations where there is no temperature sensor as seen from the results in Section 6.3. The model developed can further be used to implement a hotspot detector.

7.2 Future work

The EMS and ICTP will be added into the MeerKAT system and created as KATCP devices so that they can be monitored on the local system. The system needs to be improved so that it is KATCP complaint. Using the EMS and the ICTP a detector can be built, a system that detects the exact location of a hotspot in an immersion cooled environment. The EMS will be used for sensing and capturing and the predictor will be used for predicting temperature. The ICTP is not programmed to give the exact temperature on the sensor, a neural network might be used for this. The flow rate of the pump can be increased and the PWM can be integrated into the system so that operators can control the flow of the oil. This will also save power used to run the pump because the pump can be switched on and off depending on the load.

Bibliography

[1] C. Schollar and S. Blyth, “RFI Monitoring for the MeerKAT Radio Telescope Supervisors :,” no. February, 2015.

[2] T. Ska, T. Ska, B. Bang, N. Zealand, S. Africa, and I. Ocean, “The Square Kilometre Array Fact sheet for journalists,” *SKA Fact Sheet*, pp. 3–4, 2011.

[3] A. Performance, “Memo 132,” no. January, 2011.

[4] R. G. Edgar *et al.*, “Enabling a high throughput real time data pipeline for a large radio telescope array with GPUs,” *Comput. Phys. Commun.*, vol. 181, no. 10, pp. 1707–1714, 2010.

[5] J. W. Romein, “An efficient work-distribution strategy for gridding radio-telescope data on GPUs,” *Proc. 26th ACM Int. Conf. Supercomput. - ICS ’12*, p. 321, 2012.

[6] P. D. Spdo *et al.*, “Memo 130 SKA Phase 1 : Preliminary System Description,” *Instrumentation*, no. November, pp. 1–25, 2010.

[7] M. A. Garrett, “Square Kilometre Array : a concept design for Phase 1,” pp. 1–8, 2010.

[8] P. J. Hall, “Power Considerations for the Square Kilometre Array (SKA) Radio Telescope 3 . Special Requirements for SKA Power Systems,” 2013.

[9] F. Haddock, “Introduction to Radio Astronomy,” *Proc. IRE*, vol. 43, no. 1, pp. 273–12, 1958.

[10] B. F. Burke and F. Graham-Smith, *An Introduction to Radio Astronomy*. Cambridge University Press, 2010.

[11] T. L. Wilson, K. Rohlfs, and S. Hüttemeister, “Fundamentals of Antenna Theory,” 2013, pp. 137–164.

[12] A. R. Thompson, J. M. Moran, G. W. Swenson, and Jr., *Interferometry and Synthesis in Radio Astronomy*, vol. 20. John Wiley & Sons, 2008.

[13] T. L. Wilson, K. Rohlfs, and S. Hüttemeister, *Tools of Radio Astronomy*. Berlin,

Heidelberg: Springer Berlin Heidelberg, 2013.

[14] B. D. Asabere, M. Gaylard, C. Horellou, H. Winkler, and T. Jarrett, "Radio astronomy in Africa: the case of Ghana," p. 6, Mar. 2015.

[15] D. B. Davidson, "Potential technological spin-offs from MeerKAT and the South African Square Kilometre Array bid," *S. Afr. J. Sci.*, vol. 108, no. 1/2, p. 3 Pages, Jan. 2012.

[16] Government of South Africa, "Astronomy Geographic Advantage Act," vol. 516, no. 666, pp. 1–3, 2008.

[17] J. L. Jonas, "MeerKAT—The South African Array With Composite Dishes and Wide-Band Single Pixel Feeds," *Proc. IEEE*, vol. 97, no. 8, pp. 1522–1530, Aug. 2009.

[18] R. S. Booth, W. J. G. de Blok, J. L. Jonas, and B. Fanaroff, "MeerKAT Key Project Science, Specifications, and Proposals," Oct. 2009.

[19] SKA, "African SKA precursor inaugurates its first antenna in the Karoo - SKA Telescope." [Online]. Available: https://skatelescope.org/news/african-ska-precursor-inaugurates-first-antenna-karoo/. [Accessed: 07-Jun-2017].

[20] T. Bennett, T. Abbott, and W. Esterhuyse, "NRF Science Processor Design Document M1400-000-003," 2015.

[21] A. Beloglazov and R. Buyya, "Energy Efficient Resource Management in Virtualized Cloud Data Centers," in *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010, pp. 826–831.

[22] G. A. Brady, N. Kapur, J. L. Summers, and H. M. Thompson, "A case study and critical assessment in calculating power usage effectiveness for a data centre," *Energy Convers. Manag.*, vol. 76, pp. 155–161, 2013.

[23] D. Prucnal, "Doing more with less:Cooling computers with oil pays off."

[24] C. D. Patel, R. Sharma, C. E. Bash, and A. Beitelmal, "Thermal considerations in cooling large scale high compute density data centers," *Intersoc. Conf. Therm. Thermomechanical Phenom. Electron. Syst. ITHERM*, vol. 2002–Janua, pp. 767–776, 2002.

[25] Tidewater Mechanical, “Geothermal Heat Pumps | Tidewater Mechanical,” 2016. [Online]. Available: http://tidewatermechanical.com/geothermal-heat-pumps/. [Accessed: 07-Jun-2017].

[26] xdadevelopers, “Nvidia Unveils Kepler-Based Tegra K1 with 192 CUDA Cores and Optional 64-Bit Denver Processor!” [Online]. Available: https://www.xda-developers.com/nvidia-unveils-kepler-based-tegra-k1-with-192-cuda-cores-and-optional-64-bit-denver-processor/. [Accessed: 07-Jun-2017].

[27] S. V. Patankar, “Airflow and Cooling in a Data Center,” *J. Heat Transfer*, vol. 132, no. 7, p. 73001, 2010.

[28] J. Moore, J. S. Chase, and P. Ranganathan, “Weatherman: Automated, Online and Predictive Thermal Mapping and Management for Data Centers,” *2006 IEEE Int. Conf. Auton. Comput.*, pp. 155–164, 2006.

[29] Sanity Technology, “Data Centre cooling - Sanity Technology,” 2013. [Online]. Available: http://www.sanitytechnology.com.au/blog/p/data-centre-cooling. [Accessed: 07-Jun-2017].

[30] R. C. Chu, R. E. Simons, M. J. Ellsworth, R. R. Schmidt, and V. Cozzolino, “Review of cooling technologies for computer products,” *IEEE Trans. Device Mater. Reliab.*, vol. 4, no. 4, pp. 568–585, 2004.

[31] “Green Revolution Cooling’s Four Rack Carnotjet Installation,” 2011. [Online]. Available: https://wn.com/green_revolution_cooling’s_four_rack_carnotjet_installation. [Accessed: 07-Jun-2017].

[32] N. Kaur and S. Monga, “Comparsions of Wired and Wireless Networks: A Review,” *Int. J. Adv. Eng. Technol.*, vol. V, no. II, pp. 34–35, 2014.

[33] C. Y. Chong and S. P. Kumar, “Sensor networks: Evolution, opportunities, and challenges,” *Proc. IEEE*, vol. 91, no. 8, pp. 1247–1256, 2003.

[34] L. Kong *et al.*, “Data Loss and Reconstruction in Wireless Sensor Networks,” *IEEE Trans. Parallel Distrib. Syst.*, pp. 1–1, 2013.

[35] L. Jie, H. Ghayvat, and S. C. Mukhopadhyay, “Introducing Intel Galileo as a development platform of smart sensor: Evolution, opportunities and challenges,” *Proc. 2015 10th IEEE Conf. Ind. Electron. Appl. ICIEA 2015*, pp. 1797–1802, 2015.

[36] Maxim, “Datasheet DS18B20,” vol. 92, pp. 18–21, 2015.

[37] “Temperature and humidity module AM2302 Product Manual.”

[38] “Smart Humidity and Temperature sensor TH02 Library – Charles’s Blog.” [Online]. Available: https://hallard.me/th02-library/. [Accessed: 07-Jun-2017].

[39] a Gaddam and W. F. Esmael, “Designing a Wireless Sensors Network for Monitoring and Predicting Droughts,” *Proc. 8th Int. Conf. Sens. Technol.*, pp. 2–4, 2014.

[40] HopeRF electronic, “Digital I2C Humidity and Temperature Sensor,” pp. 1–26, 2013.

[41] Sensirion The Sensor Company, “Datasheet SHT1x Humidity and Temperature Sensor,” no. September, pp. 1–11, 2008.

[42] “Humidity and Temperature Sensor - SHT15 Breakout [SHT15] ID: 1638 - $41.95 : Adafruit Industries, Unique & fun DIY electronics and kits.” [Online]. Available: https://www.adafruit.com/product/1638. [Accessed: 07-Jun-2017].

[43] “Pololu - HIH-4030 Humidity Sensor Carrier.” [Online]. Available: https://www.pololu.com/product/1643. [Accessed: 07-Jun-2017].

[44] “Grove -Temperature Sensor User Manual.”

[45] I. I. Honeywell, “HIH-4000 Humidity sensor,” p. 6, 2010.

[46] L. Self-heating and L. I. Output, “LM35 Precision Centigrade Temperature Sensors,” 2013.

[47] “LM35 Temperature Sensor: 3 Steps.” [Online]. Available: http://www.instructables.com/id/LM35-Temperature-Sensor/. [Accessed: 07-Jun-2017].

[48] “TMP100 Temperature Sensor (SKU:TOY0045) - DFRobot Electronic Product Wiki and Tutorial: Arduino and Robot Wiki-DFRobot.com.” [Online]. Available:

https://www.dfrobot.com/wiki/index.php/TMP100_Temperature_Sensor_(SKU:TOY0045). [Accessed: 07-Jun-2017].

[49] D. Uk, "Temperature Sensor DHT 11 Humidity & Temperature Sensor," *DHT11 Datasheet*, p. 9, 2010.

[50] Tmp10x Temperature Sensor with I2Cand SMBus Interface with Alert Function in SOT-23 Package, "TMP10x Temperature Sensor," 2002.

[51] G. Description and K. Specifications, "LM75 Digital Temperature Sensor and Thermal watchdog with Two-Wire Interface Digital Temperature Sensor and Thermal watchdog with Two-Wire Interface," no. April, pp. 1–17, 2001.

[52] "LM75." [Online]. Available: http://tspares-bd.com/index.php?route=product/product&product_id=1053. [Accessed: 07-Jun-2017].

[53] M. J. McGrath and C. N. Scanaill, *Sensor Technologies: Healthcare, Wellness and Environmental Applications*. Apress, 2013.

[54] C. K. I. Williams, "Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond," in *Learning in Graphical Models*, Dordrecht: Springer Netherlands, 1998, pp. 599–621.

[55] Scott Menard, *Applied Logistic Regression Analysis - Scott Menard*, Second Edition. Sage Publications, 2002.

[56] C. K. I. Williams, "Prediction with Gaussian Processes: From Linear Regression to Linear Prediction and Beyond," in *Learning in Graphical Models*, Dordrecht: Springer Netherlands, 1998, pp. 599–621.

[57] A. G. Wilson and R. P. Adams, "Gaussian Process Kernels for Pattern Discovery and Extrapolation," vol. 28, 2013.

[58] P. Erickson, M. Cline, N. Tirpankar, and T. Henderson, "Gaussian processes for multi-sensor environmental monitoring," *IEEE Int. Conf. Multisens. Fusion Integr. Intell. Syst.*, vol. 2015–Octob, pp. 208–213, 2015.

[59] B. June, “Guide to the Systems Engineering Body of Knowledge (SEBoK) v1.4 - Full,” 2015.

[60] M. Type and R. Specification, “MeerKAT Science Processor Requirements Specification,” no. May, pp. 1–80, 2013.

[61] F.-G. Banica, *Chemical Sensors and Biosensors: Fundamentals and Applications*. John Wiley & Sons, 2012.

[62] S. Mcconnell, “CxOne Best Practice Further Reading,” *IEEE Softw.*, 1996.

[63] EK-XTOP Manufacturers, “DDC Pump,” vol. 2, 2016.

[64] D. Kuhlman, “A Python Book: Beginning Python, Advanced Python, and Python Exercises,” pp. 1–227, 2009.

[65] C. E. Rasmussen and C. K. I. Williams, *Gaussian processes for machine learning.*, vol. 14, no. 2. 2004.

[66] P. Erickson, M. Cline, N. Tirpankar, and T. Henderson, “Gaussian processes for multi-sensor environmental monitoring,” in *2015 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2015, pp. 208–213.

[67] SherifieldMachineLearningGroup, “GPy by SheffieldML.” [Online]. Available: https://sheffieldml.github.io/GPy/.

[68] M. Ebden, “Gaussian Processes: A Quick Introduction,” no. August, 2015.

[69] “Introduction to Gaussian Processes.”

[70] Indy oil South Africa, “Product description,” *Notes*, pp. 23–24, 2005.

Appendix A: Code for Temperature Monitoring System

```
import os, glob, sys #import all python modules for the
import numpy as np
import h5py
import pylab as pl
import datetime
import time
from time import strftime
from time import sleep
import sys
import pdb
import ConfigParser
import timeit
sys.stderr = open('/tmp/err2.log','w')
devices = open("/mnt/1wire/28.AD7146060000/temperature","r")
sensor1 = "28.9D4546060000"
sensor2 = "28.A81447060000"
sensor3 = "28.FBF299050000"
sensor4 = "28.285647060000"
sensor5 =  "28.8F3D47060000"
sensor6 = "28.A52A46060000"
sensor7 = "28.B9C646060000"
sensor8 = "28.103346060000"
sensor9 = "28.DAC547060000"
sensor10 = "28.F21047060000"
sensor11 = "28.6DE246060000"
sensor12 = "28.DFB946060000"
sensor13 = "28.524947060000"
sensor14 = "28.82FE99050000"
sensor15 = "28.C2CA46060000"
sensor16 = "28.3BBCAE050000"
sensor17 = "28.D12446060000"
sensor18 = "28.373646060000"
sensor19 = "28.F2BA99050000"
sensor20 = "28.1D4047060000"
sensor21 = "28.8BE899050000"
sensor22 = "28.A0F799050000"
sensor23 = "28.3CCF99050000"
sensor24 = "28.E4E099050000"
sensor25 = "28.CC8847060000"
sensor26 = "28.9B2D48060000"
sensor27 = "28.CCF499050000"
sensor28 = "28.E9C499050000"
sensor29 = "28.CFDD46060000"
sensor30 = "28.37EC46060000"
sensor31 = "28.F63A46060000"
sensor32 = "28.8FBD46060000"
sensor33 = "28.F70A9A050000"
sensor34 = "28.EEC647060000"
sensor35 = "28.826447060000"
sensor36 = "28.860D48060000"
sensor37 = "28.671546060000"
```

```
sensor38 = "28.61029A050000"
sensor39 = "28.B71E9A050000"
sensor40 = "28.BA4047060000"
sensor41 = "28.A8CE46060000"
sensor42 = "28.81F799050000"
sensor43 = "28.D61748060000"
sensor44 = "28.63E599050000"
sensor45 = "28.E86547060000"
sensor46 = "28.9EFC99050000"
sensor47 = "28.370748060000"
sensor48 = "28.09019A050000"
sensor49 = "28.FE7E47060000"
sensor50 = "28.1AA546060000"
sensor51 = "28.41F499050000"
sensor52 = "28.A22F48060000"
sensor53 = "28.A7CA99050000"
sensor54 = "28.B57146060000"
sensor55 = "28.ACC046060000"
sensor56 = "28.6B4646060000"
sensor57 = "28.E06046060000"
sensor58 = "28.DD1047060000"
sensor59 = "28.34089A050000"
sensor60 = "28.85B846060000"
#pdb.set_trace()
config = ConfigParser.ConfigParser() #declare the variable config
config.read("/home/pi/TempLog/locationsFinal.cfg")
template = "/mnt/1wire/uncached/[PLACEHOLDER]/latesttemp"
devices =
[sensor1,sensor2,sensor3,sensor4,sensor5,sensor6,sensor7,sensor8,sensor9,s
ensor10,sensor11,sensor12,sensor13,sensor14,sensor15,sensor16,sensor17,sen
sor18,sensor19,sensor20,sensor21,sensor22,sensor23,sensor24,sensor25,senso
r26,sensor27,sensor28,sensor29,sensor30,sensor31,sensor32,sensor33,sensor3
4,sensor35,sensor36,sensor37,sensor38,sensor39,sensor40,sensor41,sensor42,
sensor43,sensor44,sensor45,sensor46,sensor47,sensor48,sensor49,sensor50,se
nsor51,sensor52,sensor53,sensor54,sensor55,sensor56,sensor57,sensor58,sens
or59,sensor60]
def temp_raw(sensor_name) :
f = open(sensor_name, 'r')
t = f.read()
f.close()
return t
#print ("%f seconds" % (time.time() - start_time))    #
#print timeit(temp_raw)
def get_temp_c(sensor_name):
    t = temp_raw(sensor_name)
    temperature = float(t)
    return temperature
def read_temp1(): #creating a method called read_temp()
    #time = datetime.datetime.now().time() #declaring the variable time
    #pdb.set_trace()
    path = "/home/pi/TempLog/" +
    datetime.datetime.strftime(datetime.datetime.now(), '%Y/%m') #
    declaring the variable path
    filename = datetime.datetime.strftime(datetime.datetime.now(),"%d.h5")
```

```
location = "%s/%s"%(path,filename) #declaring the variable location
#print filename
today = datetime.date.today().day #declaring the variable today
if not os.path.exists(path): #creating a conditional statement that
creates a directory if it does not exist
os.makedirs(path) #creating a new directory in the path specified
f = h5py.File(location,'w') #creating an HDF5 file with the name
called by the time and the day of the week
group = f.create_group("EMS")
group1  = group.create_group("ConfigVal")
group2  = group.create_group("TempVal")
group3  = group.create_group("HumidVal")
group4   = group.create_group("PowerVal")
group5 = group.create_group("Timestamps")
#print "here"

group5.create_dataset("t", (7200,),dtype = np.float64)
for i in devices: #creating a for loop that loops through the devices
#grp = f.create_group("sensor")
group2.create_dataset(i, (7200,2,),dtype = np.float64)#creating a
dataset inside a file which creates a space for 144 32 bit floats in
the dataset called i(which is the unique id of the sensor)
#print i
if config.has_section(i):
f['EMS']['TempVal'][i].attrs['x']= config.get(i,'x')#create attribute
f['EMS']['TempVal'][i].attrs['y']= config.get(i,'y')
f['EMS']['TempVal'][i].attrs['z']= config.get(i,'z')
else:
print "%s not in config file"%i
#print("attributes created")
#pdb.set_trace()
count = 0
#start_time = time.time()
while True:
#pdb.set_trace()
#try:
#print "loop"

if today != datetime.datetime.today().day:#conditional statement to
check if the date is not today
start_time = time.time()
f.close()#closing the file
file you create
path = "/home/pi/TempLog/" +
datetime.datetime.strftime(datetime.datetime.now(), '%Y/%m') #
declaring the variable path
#filename = time #declaring the variable filename
filename = datetime.datetime.strftime(datetime.datetime.now(),"%d.h5")
location = "%s/%s"%(path,filename) #declaring the variable location
today = datetime.datetime.today().day #Must set today to new day,
otherwise this if statement will always be true and your code will
crash
count = 0 #setting the count to zero so that we can increment starting
from 1
```

```
if not os.path.exists(path):#creating a conditional statement that
creates a directory if it does not exist
os.makedirs(path) #creating a new directory in the path specified
f = h5py.File(location,'w')#creating an HDF5 file with the name called
by the time and the day of the week
group = f.create_group("EMS")
group1  = group.create_group("ConfigVal")
group2  = group.create_group("TempVal")
group3  = group.create_group("HumidVal")
group4   = group.create_group("PowerVal")
group5 = group.create_group("Timestamps")
group5.create_dataset("t", (7200,),dtype = np.float32)

#f["EMS/Timestamps"]["t"] = time.time()
for i in devices:

group2.create_dataset(i, (7200,2,),dtype = np.float64)#creating a
dataset inside a file which creates a space for 144 32 bit floats in
the dataset called i(which is the unique id of the sensor)
#print e
list1 = config.sections()
list2 = f['EMS']['TempVal'].keys()
if [item for item in list1 if item in list2]:
f['EMS']['TempVal'][i].attrs['x']= config.get(i,'x')#create attribute
f['EMS']['TempVal'][i].attrs['y']= config.get(i,'y')
f['EMS']['TempVal'][i].attrs['z']= config.get(i,'z')
#print i
print ("%f seconds to create file" % (time.time() - start_time))
#pdb.set_trace()
#f[i].attrs['x'] = 3
#time.sleep (1)#this is a way to determine when/how long the code
should run
#pdb.set_trace()
#for i in f["EMS/Timestamps"]:
#f["EMS/Timestamps"]["t"] = time.time()

os.system('echo 1 > /mnt/1wire/simultaneous/temperature')
sample_time = time.time()
time.sleep (1)
for i in devices: #creating a for loop that loops through the devices
#start_time = time.time()
#print("we are inside the for loop")
sensor_name = template.replace("[PLACEHOLDER]",i)
try:
temp_c = get_temp_c(sensor_name) #calling the string replace
method,which replaces the unique id everytime the for loop runs
except IOError:
print "Could not read sensor %s"%i
else:
f["EMS/TempVal"][i][count,0] = temp_c
f["EMS/TempVal"][i][count,1] = sample_time

#print i
f.flush()
```

```
    #print ("%f seconds to read sensor %s" % (time.time() - start_time,
    sensor_name))
    #print("we have entered the data")
    #print("we are outside the loop")

    count+=1 #incrementing the count
    print count
    f.flush()
```
read_temp1()

Appendix B: Code for humidity sensor

```
import pigpio
import os
from time import sleep
import DHT22
import os, glob, sys #import all python modules for the
import numpy as np
import h5py
import pylab as pl
import datetime
import time
from time import strftime
import pdb
import sys

sys.stderr = open('/tmp/err.log','w')

pi = pigpio.pi()
dht22 = DHT22.sensor(pi,04)
dht22.trigger()
sleep(3)
#pdb.set_trace()
#os.chdir('DHT22')
def get_ht():
    dht22.trigger()
    humidity = dht22.humidity()
    temperature = dht22.temperature()
    return humidity,temperature
    print humidity
print ("The GPIO pins are enabled")


#pdb.set_trace()
def humidity(): #creating a method called read_temp()
    print ("I am inside the method")
    #time = datetime.datetime.now().time() #declaring the variable time
    path             =                          "/home/pi/"             +
datetime.datetime.strftime(datetime.datetime.now(),  '%Y/%m')  #  declaring
the variable path
        #filename = time #declaring the variable filename
    filename = datetime.datetime.strftime(datetime.datetime.now(),"%d.h5")
    location = "%s/%s"%(path,filename) #declaring the variable location
         #print filenam
    today = datetime.date.today().day #declaring the variable today
    if  not  os.path.exists(path): #creating  a  conditional  statement  that
creates a directory if it does not exist
                os.makedirs(path) #creating  a  new  directory  in  the  path
specified
    f  =  h5py.File(location,'w')  #creating  an  HDF5  file  with  the  name
called by the time and the day of the week
    #creating a for loop that loops through the devices
```

```
    f.create_dataset("dht22",  (144,3,),dtype  =  np.float32)#creating  a
dataset inside a file which creates a space for 144 32 bit floats in the
dataset called i(which is the unique id of the sensor)
    count = 0
    pdb.set_trace()
    while True: #an infinite while loop
        #hum, temperature = get_ht()
        #pdb.set_trace()
        if  today  !=  datetime.datetime.today().day:#conditional  statement
to check if the date is not today
            f.close()#closing the file
                #ADDED BY CHRIS -- You need to figure out the new path for
each new file you create
            path           =                        "/home/pi/"           +
datetime.datetime.strftime(datetime.datetime.now(),   '%Y/%m')#   declaring
the variable path
                #filename = time #declaring the variable filename
            filename                                                       =
datetime.datetime.strftime(datetime.datetime.now(),"%d.h5")
            location  =  "%s/%s"%(path,filename)  #declaring  the  variable
location
            today = datetime.datetime.today().day #Must set today to new
day, otherwise this if statement will always be true and your code will
crash
                #END ADDED BY CHRIS
            #pdb.set_trace()
            count = 0 #setting the count to zero so that we can increment
starting from 1
            if not os.path.exists(path):#creating a conditional statement
that creates a directory if it does not exist
                os.makedirs(path) #creating a new directory in the path
specified
            f  =  h5py.File(location,'w')#creating  an  HDF5  file  with  the
name called by the time and the day of the week
            f.create_dataset("dht22",  (144,3,),dtype  =  np.int32)#creating
a dataset inside a file which creates a space for 144 32 bit floats in the
dataset called i(which is the unique id of the sensor)
            #f['dht22'] = [hum]

            sleep (0.1)#this is a way to determine when/how long the code
should run
        #pdb.set_trace()

        #for  i  in  values:  #creating  a  for  loop  that  loops  through  the
devices
        humid = get_ht()#calling the string replace method,which replaces
the unique id everytime the for loop runs
        # print f[humid]
        #pdb.set_trace()
        print ("The I have started entering data to the file")

        f['dht22'][count,0] = humid[0]
        f['dht22'][count,1] = humid[1]
        f['dht22'][count,1] = time.time()
```

```
        f.flush()

        #print humid

        print ("I am about to sleep")
        sleep (0.1)
        count+=1 #incrementing the count
        print count

humidity()
```

Appendix C: Code for the Immersion Cooling Temperature Predictor

```
import GPy
#GPy.plotting.change_plotting_library('plotly')
#import matplotlib.pyplot as plt
import numpy as np
#import plotly.plotly as py
# Learn about API authentication here: https://plot.ly/python/getting-
started
# Find your api_key here: https://plot.ly/settings/api
import h5py
import ConfigParser
from IPython.display import display
import matplotlib.pyplot as plt
import plotly.plotly as py
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import pylab as pb
from matplotlib.backends.backend_pdf import PdfPages

get_ipython().magic(u'pylab inline')

f = h5py.File("/home/apiwe/Documents/sensor_system_code/ResistorLocations
/tworesistorsnopumpF.h5", "r+")
config = ConfigParser.ConfigParser() #declare the variable config
config.read("/home/apiwe/Documents/sensor_system_code/locationsFinal.cfg")
#read from the file
data_x1 = []
data_y1 = []
data_t = []
#1100 was an interesting time
start_time = 300
num_times = 1#the number of samples in time used
time_step = 60 # the number of steps, how far way point are from each
other
#time in the file or dataset
for   i,keyv   in   enumerate(f['EMS']['TempVal'].keys()):#keyv   sensor
name,enumerate numbers
    #print f['EMS']['TempVal'][keyv].attrs.items()
    h5_data = f['EMS/TempVal'][keyv]
    x = 10 * float(h5_data.attrs['x'])
    if x == 30.:
        x = 20.
    y = 10 * float(h5_data.attrs['y'])
    z = 10 * float(h5_data.attrs['z'])
    time_indices = slice(start_time, start_time + num_times * time_step,
time_step)
    temp = h5_data[time_indices, 0]
    temp_times = h5_data[time_indices, 1]
#    relative_times = temp_times - temp_times[-1]
    relative_times = temp_times - temp_times[len(temp_times) // 2]
    if not all(temp):#throw out any zero
        continue
```

```
    #print x,y,z,temp
    for t in relative_times:
        data_x1.append((x, y, z, t))
    #coordinates.append(add)
    data_y1.append(temp)
    data_t.append(temp_times)
x1 = np.atleast_2d(data_x1) # turn into cm
y1= np.atleast_2d(np.ravel(data_y1)).T #
#print x1, np.atleast_2d(data_y1)
#print x1
train_indices = []
test_indices = []
for i, (x,y,z,t) in enumerate(x1):
    #print t
    add = (x+y+z)/10
    if (add % 2 == 1):
        train_indices.append(i)
    else:
        test_indices.append(i)
train_y = y1[train_indices]
train_x = x1[train_indices]
test_x  = x1[test_indices]
test_y  = y1[test_indices]
#define mean function SAUPEC
mf = GPy.core.Mapping(4, 1)
mf.f = lambda x: 30
mf.update_gradients = lambda a,b: None
# define kernel
#ker = GPy.kern.Matern52(3,ARD=True) + GPy.kern.White(3)
#ker = GPy.kern.RBF(input_dim=3, variance=1., lengthscale=1.)
#ker = GPy.kern.RatQuad(input_dim=3, variance=1., lengthscale=1.)
ker  =  GPy.kern.Matern52(input_dim=4,  variance=100.,  lengthscale=18.0,
ARD=True) # + GPy.kern.White(0.15)
# create simple GP model
m       =       GPy.models.GPRegression(train_x,       train_y,       ker,
mean_function=mf)#training data
#m['.*lengthscale']  =  3.How  to  set  parameters,you  can  replace  the
lengthscale with var to get variance
#print m[''] see all the parameters in the model
m.unconstrain('')
#m.constrain_fixed(variance=50, warning=True, trigger_parent=True)
m['Gaussian_noise.variance'].constrain_fixed(0.3)
m['Mat52.variance'].constrain_fixed(100)
m['Mat52.lengthscale'].constrain_fixed((18, 18, 18, 100))
#m['Mat52.lengthscale'].constrain_bounded(10, 10000)
print m['Mat52.lengthscale']
#m.optimize()
# Train and test points
#testing the performance of the predictor, compare prodiction to actual
value
sensor_x, sensor_y, sensor_z, sensor_t = train_x.T
sensor_test_x, sensor_test_y, sensor_test_z, sensor_test_t = test_x.T
def  plot_1d_slice(x=None,  y=None,  z=None,  t=0,  xlim=None,  ylim=None,
zlim=None, ax=None, sigma_factor=2):
```

```
    fixed_inputs = []
    slice_samples = 200
    if x is None:
        fixed_inputs = [(1, y), (2, z), (3, t)]
        slice_xaxis = np.linspace(xlim[0], xlim[1], slice_samples)
        slice_coords = np.c_[slice_xaxis,
                             np.tile(y, slice_samples),
                             np.tile(z, slice_samples),
                             np.tile(t, slice_samples)]
        select_xaxis = (sensor_y == y) & (sensor_z == z) & (sensor_t == t)
        train_xaxis = sensor_x[select_xaxis]
        train_yaxis = train_y[select_xaxis]
        select_xaxis = (sensor_test_y == y) & (sensor_test_z == z) &
(sensor_test_t == t)
        test_xaxis = sensor_test_x[select_xaxis]
        test_yaxis = test_y[select_xaxis]
    elif y is None:
        fixed_inputs = [(0, x), (2, z), (3, t)]
        slice_xaxis = np.linspace(ylim[0], ylim[1], slice_samples)
        slice_coords = np.c_[np.tile(x, slice_samples),
                             slice_xaxis,
                             np.tile(z, slice_samples),
                             np.tile(t, slice_samples)]
        select_xaxis = (sensor_x == x) & (sensor_z == z) & (sensor_t == t)
        train_xaxis = sensor_y[select_xaxis]
        train_yaxis = train_y[select_xaxis]
        select_xaxis = (sensor_test_x == x) & (sensor_test_z == z) &
(sensor_test_t == t)
        test_xaxis = sensor_test_y[select_xaxis]
        test_yaxis = test_y[select_xaxis]
    elif z is None:
        fixed_inputs = [(0, x), (1, y), (3, t)]
        slice_xaxis = np.linspace(zlim[0], zlim[1], slice_samples)
        slice_coords = np.c_[np.tile(x, slice_samples),
                             np.tile(y, slice_samples),
                             slice_xaxis,
                             np.tile(t, slice_samples)]
        select_xaxis = (sensor_x == x) & (sensor_y == y) & (sensor_t == t)
        train_xaxis = sensor_z[select_xaxis]
        train_yaxis = train_y[select_xaxis]
        select_xaxis = (sensor_test_x == x) & (sensor_test_y == y) &
(sensor_test_t == t)
        test_xaxis = sensor_test_z[select_xaxis]
        test_yaxis = test_y[select_xaxis]
    slice_temp, slice_temp_std = m.predict(slice_coords)
    slice_yaxis,      slice_yaxis_std      =      slice_temp[:,      0],
np.sqrt(slice_temp_std[:, 0])
#             ax1   =   m.plot(fixed_inputs=fixed_inputs,   plot_data=False,
legend=None)
#     ax1.plot(train_xaxis, train_yaxis, 'kx', markeredgewidth=2)
#     ax1.plot(test_xaxis, test_yaxis, 'ro')
    ax.fill_between(slice_xaxis,    slice_yaxis    -    sigma_factor    *
slice_yaxis_std,
```

```
                    slice_yaxis   +   sigma_factor   *   slice_yaxis_std,
alpha=0.2, color='0.5')
    ax.plot(slice_xaxis, slice_yaxis, 'k', lw=2)
    ax.plot(train_xaxis, train_yaxis, 'bx', markeredgewidth=2)
    ax.plot(test_xaxis, test_yaxis, 'ro')
    ax.set_xlim(slice_xaxis[0], slice_xaxis[-1])
    return ax
fig, ax_grid = subplots(4, 3, sharex=True, sharey=True, figsize=(10, 12))
yaxis_lim = (-5, 45)
# z = 0
ax = plot_1d_slice(x=0, z=0, ylim=yaxis_lim, ax=ax_grid[3, 0])
ax.set_xlabel('y (cm)')
ax.set_ylabel('Temp (deg C)')
ax = plot_1d_slice(x=10, z=0, ylim=yaxis_lim, ax=ax_grid[3, 1])
ax.set_xlabel('y (cm)')
ax = plot_1d_slice(x=20, z=0, ylim=yaxis_lim, ax=ax_grid[3, 2])
ax.set_xlabel('y (cm)')
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=0', rotation='horizontal', labelpad=20)
# z = 10
ax = plot_1d_slice(x=0, z=10, ylim=yaxis_lim, ax=ax_grid[2, 0])
ax.set_ylabel('Temp (deg C)')
ax = plot_1d_slice(x=10, z=10, ylim=yaxis_lim, ax=ax_grid[2, 1])
ax = plot_1d_slice(x=20, z=10, ylim=yaxis_lim, ax=ax_grid[2, 2])
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=10', rotation='horizontal', labelpad=20)
# z = 20
ax = plot_1d_slice(x=0, z=20, ylim=yaxis_lim, ax=ax_grid[1, 0])
ax.set_ylabel('Temp (deg C)')
ax = plot_1d_slice(x=10, z=20, ylim=yaxis_lim, ax=ax_grid[1, 1])
ax = plot_1d_slice(x=20, z=20, ylim=yaxis_lim, ax=ax_grid[1, 2])
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=20', rotation='horizontal', labelpad=20)
# z = 30
ax = plot_1d_slice(x=0, z=30, ylim=yaxis_lim, ax=ax_grid[0, 0])
ax.set_ylabel('Temp (deg C)')
ax.set_title('x=0')
ax = plot_1d_slice(x=10, z=30, ylim=yaxis_lim, ax=ax_grid[0, 1])
ax.set_title('x=10')
ax = plot_1d_slice(x=20, z=30, ylim=yaxis_lim, ax=ax_grid[0, 2])
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=30', rotation='horizontal', labelpad=20)
ax.set_title('x=20')
#ax.figure.savefig("Predicted   values   with   pump   and   two   20W   @380
resistors")

predict_y,predicty_var =m.predict(test_x)
error=predict_y - test_y
print error.mean(),error.std(),np.sqrt(predicty_var.mean())
#biased   or   unbiased,actual   error(measured   uncertainty)   in   terms   of
std,predicted error(predicted uncertainty)

#Test on full dataset
train_y = y1
```

```
train_x = x1

#define mean function SAUPEC
mf = GPy.core.Mapping(4, 1)
mf.f = lambda x: 30
mf.update_gradients = lambda a,b: None
# define kernel
#ker = GPy.kern.Matern52(3,ARD=True) + GPy.kern.White(3)
#ker = GPy.kern.RBF(input_dim=3, variance=1., lengthscale=1.)
#ker = GPy.kern.RatQuad(input_dim=3, variance=1., lengthscale=1.)
ker = GPy.kern.Matern52(input_dim=4, variance=100., lengthscale=5.0,
ARD=True) # + GPy.kern.White(0.15)
# create simple GP model
m = GPy.models.GPRegression(train_x, train_y, ker,
mean_function=mf)#training data
#m['.*lengthscale'] = 3.How to set parameters,you can replace the
lengthscale with var to get variance
#print m[''] see all the parameters in the model
m.unconstrain('')
#m.constrain_fixed(variance=50, warning=True, trigger_parent=True)
m['Gaussian_noise.variance'].constrain_fixed(0.3)
m['Mat52.variance'].constrain_fixed(100)
m['Mat52.lengthscale'].constrain_fixed((18, 18, 18, 1000))
#m['Mat52.lengthscale'].constrain_bounded(1, 10000)
#m.optimize()
print m['Mat52.lengthscale']
test_x = x1[:] + np.array([[5., 5., 0., 0]])
test_x = test_x[(test_x[:, 0] < 21) & (test_x[:, 1] < 41) & (test_x[:, 2]
< 31)]
test_y, test_y_var = m.predict(test_x)
print np.sqrt(test_y_var).mean()

# Train points only
#Performance of predictor on the full 60 sensors
sensor_x, sensor_y, sensor_z, sensor_t =train_x.T
def plot_1d_slice(x=None, y=None, z=None, t=0, xlim=None, ylim=None,
zlim=None, ax=None, sigma_factor=2):
    fixed_inputs = []
    slice_samples = 200
    if x is None:
        fixed_inputs = [(1, y), (2, z), (3, t)]
        slice_xaxis = np.linspace(xlim[0], xlim[1], slice_samples)
        slice_coords = np.c_[slice_xaxis,
                             np.tile(y, slice_samples),
                             np.tile(z, slice_samples),
                             np.tile(t, slice_samples)]
        select_xaxis = (sensor_y == y) & (sensor_z == z) & (sensor_t == t)
        train_xaxis = sensor_x[select_xaxis]
        train_yaxis = train_y[select_xaxis]
    elif y is None:
        fixed_inputs = [(0, x), (2, z), (3, t)]
        slice_xaxis = np.linspace(ylim[0], ylim[1], slice_samples)
        slice_coords = np.c_[np.tile(x, slice_samples),
                             slice_xaxis,
```

```
                            np.tile(z, slice_samples),
                            np.tile(t, slice_samples)]
        select_xaxis = (sensor_x == x) & (sensor_z == z) & (sensor_t == t)
        train_xaxis = sensor_y[select_xaxis]
        train_yaxis = train_y[select_xaxis]
    elif z is None:
        fixed_inputs = [(0, x), (1, y), (3, t)]
        slice_xaxis = np.linspace(zlim[0], zlim[1], slice_samples)
        slice_coords = np.c_[np.tile(x, slice_samples),
                            np.tile(y, slice_samples),
                            slice_xaxis,
                            np.tile(t, slice_samples)]
        select_xaxis = (sensor_x == x) & (sensor_y == y) & (sensor_t == t)
        train_xaxis = sensor_z[select_xaxis]
        train_yaxis = train_y[select_xaxis]
    slice_temp, slice_temp_std = m.predict(slice_coords)
    slice_yaxis,      slice_yaxis_std      =      slice_temp[:,      0],
np.sqrt(slice_temp_std[:, 0])
    ax.fill_between(slice_xaxis,    slice_yaxis    -    sigma_factor    *
slice_yaxis_std,
                    slice_yaxis   +   sigma_factor   *   slice_yaxis_std,
alpha=0.2, color='0.5')
    ax.plot(slice_xaxis, slice_yaxis, 'k', lw=2)
    ax.plot(train_xaxis, train_yaxis, 'bx', markeredgewidth=2)
    ax.set_xlim(slice_xaxis[0], slice_xaxis[-1])
    return ax
# In[60]:

sensor_x, sensor_y, sensor_z, sensor_t = train_x.T
sensor_test_x, sensor_test_y, sensor_test_z, sensor_test_t = test_x.T
def  plot_1d_slice(x=None,  y=None,  z=None,  t=0,  xlim=None,  ylim=None,
zlim=None, ax=None, sigma_factor=2):
    fixed_inputs = []
    slice_samples = 200
    if x is None:
        fixed_inputs = [(1, y), (2, z), (3, t)]
        slice_xaxis = np.linspace(xlim[0], xlim[1], slice_samples)
        slice_coords = np.c_[slice_xaxis,
                            np.tile(y, slice_samples),
                            np.tile(z, slice_samples),
                            np.tile(t, slice_samples)]
        select_xaxis = (sensor_y == y) & (sensor_z == z) & (sensor_t == t)
        train_xaxis = sensor_x[select_xaxis]
        train_yaxis = train_y[select_xaxis]
        select_xaxis  =  (sensor_test_y  ==  y)  &  (sensor_test_z  ==  z)  &
(sensor_test_t == t)
        test_xaxis = sensor_test_x[select_xaxis]
        test_yaxis = test_y[select_xaxis]
    elif y is None:
        fixed_inputs = [(0, x), (2, z), (3, t)]
        slice_xaxis = np.linspace(ylim[0], ylim[1], slice_samples)
        slice_coords = np.c_[np.tile(x, slice_samples),
                            slice_xaxis,
                            np.tile(z, slice_samples),
```

```
                            np.tile(t, slice_samples)]
        select_xaxis = (sensor_x == x) & (sensor_z == z) & (sensor_t == t)
        train_xaxis = sensor_y[select_xaxis]
        train_yaxis = train_y[select_xaxis]
        select_xaxis = (sensor_test_x == x) & (sensor_test_z == z) &
(sensor_test_t == t)
        test_xaxis = sensor_test_y[select_xaxis]
        test_yaxis = test_y[select_xaxis]
    elif z is None:
        fixed_inputs = [(0, x), (1, y), (3, t)]
        slice_xaxis = np.linspace(zlim[0], zlim[1], slice_samples)
        slice_coords = np.c_[np.tile(x, slice_samples),
                             np.tile(y, slice_samples),
                             slice_xaxis,
                             np.tile(t, slice_samples)]
        select_xaxis = (sensor_x == x) & (sensor_y == y) & (sensor_t == t)
        train_xaxis = sensor_z[select_xaxis]
        train_yaxis = train_y[select_xaxis]
        select_xaxis = (sensor_test_x == x) & (sensor_test_y == y) &
(sensor_test_t == t)
        test_xaxis = sensor_test_z[select_xaxis]
        test_yaxis = test_y[select_xaxis]
    slice_temp, slice_temp_std = m.predict(slice_coords)
    slice_yaxis, slice_yaxis_std = slice_temp[:, 0],
np.sqrt(slice_temp_std[:, 0])
#             ax1 = m.plot(fixed_inputs=fixed_inputs, plot_data=False,
legend=None)
#     ax1.plot(train_xaxis, train_yaxis, 'kx', markeredgewidth=2)
#     ax1.plot(test_xaxis, test_yaxis, 'ro')
    ax.fill_between(slice_xaxis, slice_yaxis - sigma_factor *
slice_yaxis_std,
                    slice_yaxis + sigma_factor * slice_yaxis_std,
alpha=0.2, color='0.5')
    ax.plot(slice_xaxis, slice_yaxis, 'k', lw=2)
    ax.plot(train_xaxis, train_yaxis, 'bx', markeredgewidth=2)
    ax.plot(test_xaxis, test_yaxis, 'r^')
    ax.set_xlim(slice_xaxis[0], slice_xaxis[-1])
    return ax

slice_z = 30
temp_levels = np.arange(18, 70., 1.)

xlim = [-5, 25]
ylim = [-5, 45]
slice_samples = 200
slice_xaxis = np.linspace(xlim[0], xlim[1], slice_samples)
slice_yaxis = np.linspace(ylim[0], ylim[1], slice_samples)
slice_cols, slice_rows = np.meshgrid(slice_xaxis, slice_yaxis)
slice_cols, slice_rows = slice_cols.ravel(), slice_rows.ravel()
slice_coords = np.c_[slice_cols,
                     slice_rows,
                     np.tile(slice_z, slice_samples * slice_samples),
                     np.tile(0, slice_samples * slice_samples)]
slice_temp, slice_temp_var = m.predict(slice_coords)
```

```
slice_temp_upper = slice_temp + 2.0 * np.sqrt(slice_temp_var)
slice_temp_lower = slice_temp - 2.0 * np.sqrt(slice_temp_var)
fig, ax = plt.subplots(figsize=(4, 8))
sensor_x2 = x1[x1[:, 2] == 30., 0]
sensor_y2 = x1[x1[:, 2] == 30., 1]
ax.plot(sensor_x2, sensor_y2, 'bo', alpha=0.5)
contset          =          ax.contour(slice_xaxis,          slice_yaxis,
slice_temp.reshape(slice_samples, slice_samples),
                    levels=temp_levels, colors='k')
ax.clabel(contset, inline_spacing=5, fontsize=10, fmt='%2.0f')
resistor = matplotlib.patches.Rectangle((10 - 3, 20 - 0.5), 6., 1.,
angle=0.0, fc='r', ec='r')
ax.add_patch(resistor)
resistor = matplotlib.patches.Rectangle((10 - 3, 0 - 0.5), 6., 1.,
angle=0.0, fc='r', ec='r')
ax.add_patch(resistor)
ax.axis('image')
ax.set_xlabel('x (cm)')
ax.set_ylabel('y (cm)')
ax.set_title('Predicted temperature at z = %d cm' % (slice_z,))

fig, ax_grid = subplots(4, 3, sharex=True, sharey=True, figsize=(10, 12))
yaxis_lim = (-5, 45)
# z = 0
ax = plot_1d_slice(x=0, z=0, ylim=yaxis_lim, ax=ax_grid[3, 0])
ax.set_xlabel('y (cm)')
ax.set_ylabel('Temp (deg C)')
ax = plot_1d_slice(x=10, z=0, ylim=yaxis_lim, ax=ax_grid[3, 1])
ax.set_xlabel('y (cm)')
ax = plot_1d_slice(x=20, z=0, ylim=yaxis_lim, ax=ax_grid[3, 2])
ax.set_xlabel('y (cm)')
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=0', rotation='horizontal', labelpad=20)
# z = 10
ax = plot_1d_slice(x=0, z=10, ylim=yaxis_lim, ax=ax_grid[2, 0])
ax.set_ylabel('Temp (deg C)')
ax = plot_1d_slice(x=10, z=10, ylim=yaxis_lim, ax=ax_grid[2, 1])
ax = plot_1d_slice(x=20, z=10, ylim=yaxis_lim, ax=ax_grid[2, 2])
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=10', rotation='horizontal', labelpad=20)
# z = 20
ax = plot_1d_slice(x=0, z=20, ylim=yaxis_lim, ax=ax_grid[1, 0])
ax.set_ylabel('Temp (deg C)')
ax = plot_1d_slice(x=10, z=20, ylim=yaxis_lim, ax=ax_grid[1, 1])
ax = plot_1d_slice(x=20, z=20, ylim=yaxis_lim, ax=ax_grid[1, 2])
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=20', rotation='horizontal', labelpad=20)
# z = 30
ax = plot_1d_slice(x=0, z=30, ylim=yaxis_lim, ax=ax_grid[0, 0])
ax.set_ylabel('Temp (deg C)')
ax.set_title('x=0')
ax = plot_1d_slice(x=10, z=30, ylim=yaxis_lim, ax=ax_grid[0, 1])
ax.set_title('x=10')
ax = plot_1d_slice(x=20, z=30, ylim=yaxis_lim, ax=ax_grid[0, 2])
```

```
ax.yaxis.set_label_position('right')
ax.set_ylabel('z=30', rotation='horizontal', labelpad=20)
ax.set_title('x=20')
ax.figure.savefig("Actual values")

predict_y, predict_y_var = m.predict(test_x)
np.sqrt(predict_y_var).mean()
```

Appendix D: Configuration file for the locations

```
[28.63E599050000]
uri: /home/pi/TempLog/28.63E599050000/temperature
x  : 0
y  : 0
z  : 0

[28.285647060000]
uri: /home/pi/TempLog/28.285647060000/temperature
x     : 0
y     : 1
z     : 0

[28.373646060000]
uri: /home/pi/TempLog/28.373646060000/temperature
x     : 0
y     : 2
z     : 0
[28.8BE899050000]
uri: /home/pi/TempLog/28.8BE899050000/temperature
x     : 0
y     : 3
z     : 0
[28.332148060000]
uri: /home/pi/TempLog/28.332148060000/temperature
x     : 0
y     : 4
z     : 0
[28.3CCF99050000]
uri: /home/pi/TempLog/28.3CCF99050000/temperature
x     : 1
y     : 0
z     : 0
[28.D61748060000]
uri: /home/pi/TempLog/28.D61748060000/temperature
x     : 1
y     : 2
z     : 0
[28.DAC547060000]
uri: /home/pi/TempLog/28.DAC547060000/temperature
x     : 1
y     : 3
z     : 0
[28.826447060000]
uri: /home/pi/TempLog/28.826447060000/temperature
x     : 1
y     : 4
z     : 0
[28.9B2D48060000]
uri: /home/pi/TempLog/28.9B2D48060000/temperature
x     : 2
y     : 0
```

```
z     : 0
[28.B9DE99050000]
uri: /home/pi/TempLog/28.B9DE99050000/temperature
x     : 2
y     : 1
z     : 0
[28.524947060000]
uri: /home/pi/TempLog/28.524947060000/temperature
x     : 2
y     : 2
z     : 0
[28.D08246060000]
uri: /home/pi/TempLog/28.D08246060000/temperature
x     : 2
y     : 3
z     : 0
[28.CFDD46060000]
uri: /home/pi/TempLog/28.CFDD46060000/temperature
x     : 2
y     : 4
z     : 0
[28.13CE46060000]
uri: /home/pi/TempLog/28.13CE46060000/temperature
x     : 0
y     : 0
z     : 1
[28.E6B899050000]
uri: /home/pi/TempLog/28.E6B899050000/temperature
x     : 0
y     : 1
z     : 1
[28.E9C499050000]
uri: /home/pi/TempLog/28.E9C499050000/temperature
x     : 0
y     : 2
z     : 1
[28.CCF499050000]
uri: /home/pi/TempLog/28.CCF499050000/temperature
x     : 0
y     : 3
z     : 1
[28.A8CE46060000]
uri: /home/pi/TempLog/28.A8CE46060000/temperature
x     : 0
y     : 4
z     : 1
[28.1D4047060000]
uri: /home/pi/TempLog/28.1D4047060000/temperature
x     : 1
y     : 0
z     : 1
[28.B71E9A050000]
uri: /home/pi/TempLog/28.B71E9A050000/temperature
x     : 1
```

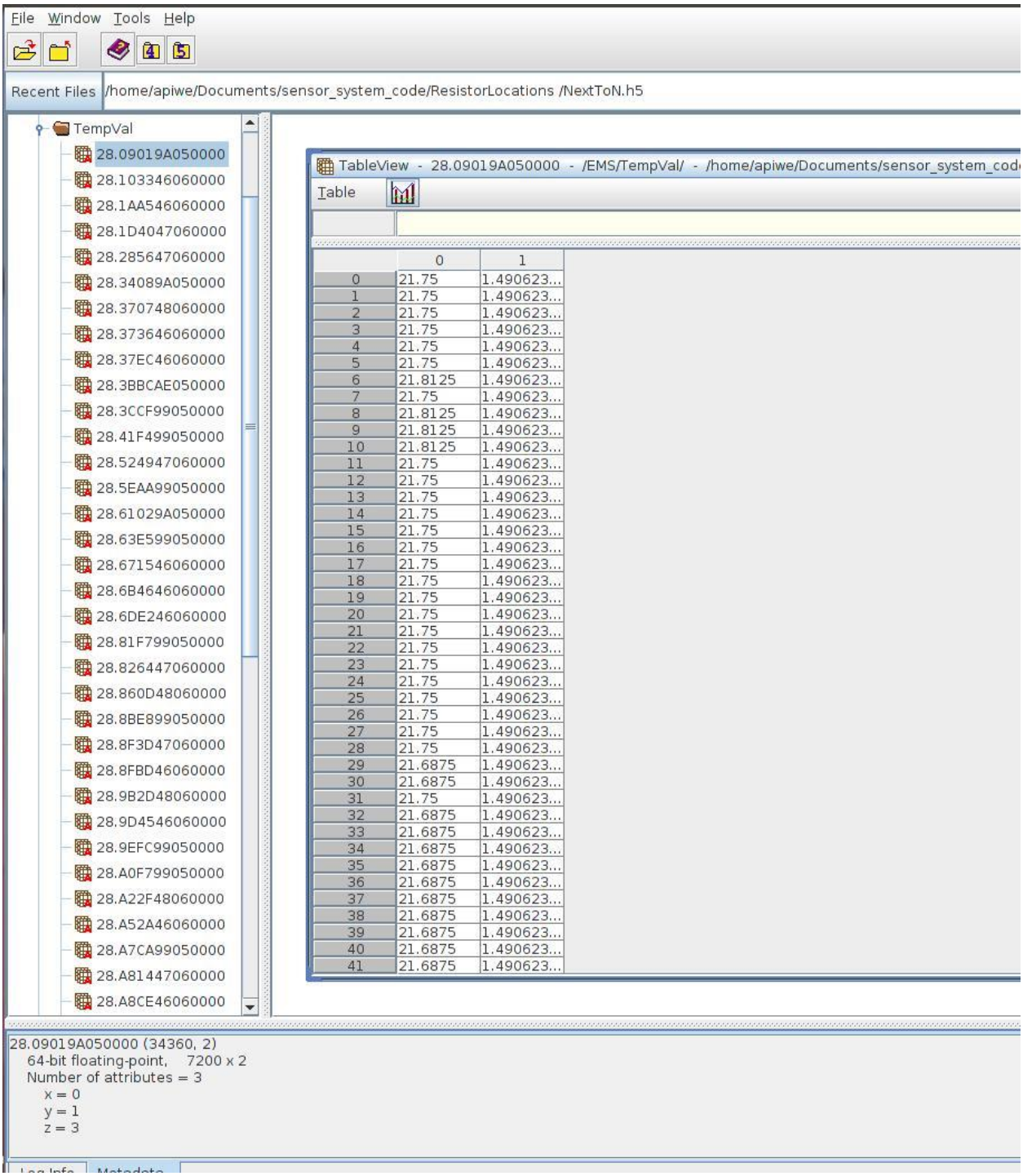
```
y     : 1
z     : 1
[28.1B4A47060000]
uri: /home/pi/TempLog/28.1B4A47060000/temperature
x     : 1
y     : 2
z     : 1
[28.CC8847060000]
uri: /home/pi/TempLog/28.CC8847060000/temperature
x     : 1
y     : 3
z     : 1
[28.C2CA46060000]
uri: /home/pi/TempLog/28.C2CA46060000/temperature
x     : 1
y     : 4
z     : 1

[28.34DF99050000]
uri:  /home/pi/TempLog/28.34DF99050000/temperature  #fixed  but  is  still
questionable.....
x     : 2
y     : 1
z     : 1

[28.829D47060000]
uri: /home/pi/TempLog/28.829D47060000/temperature
x     : 2
y     : 1
z     : 1
[28.DFB946060000]
uri: /home/pi/TempLog/28.DFB946060000/temperature
x     : 2
y     : 2
z     : 1
[28.306E46060000]
uri: /home/pi/TempLog/28.306E46060000/temperature
x     : 2
y     : 3
z     : 1
[28.FBF299050000]
uri: /home/pi/TempLog/28.FBF299050000/temperature
x     : 2
y     : 4
z     : 1
[28.E4E099050000]
uri: /home/pi/TempLog/28.E4E099050000/temperature
x     : 0
y     : 0
z     : 2
[28.DD0747060000]
uri: /home/pi/TempLog/28.DD0747060000/temperature
x     : 0
y     : 1
```

```
z     : 2
[28.A81447060000]
uri: /home/pi/TempLog/28.A81447060000/temperature
x     : 0
y     : 2
z     : 2
[28.61029A050000]
uri: /home/pi/TempLog/28.61029A050000/temperature
x     : 0
y     : 3
z     : 2
[28.E06046060000]
uri: /home/pi/TempLog/28.E06046060000/temperature
x     : 1
y     : 0
z     : 2
[28.D12446060000]
uri: /home/pi/TempLog/28.D12446060000/temperature
x     : 1
y     : 1
z     : 2
[28.BA4047060000]
uri: /home/pi/TempLog/28.BA4047060000/temperature
x     : 1
y     : 2
z     : 2
[28.103346060000]
uri: /home/pi/TempLog/28.103346060000/temperature
x     : 1
y     : 3
z     : 2
[28.37EC46060000]
uri: /home/pi/TempLog/28.37EC46060000/temperature
x     : 2
y     : 0
z     : 2
[28.81F799050000]
uri: /home/pi/TempLog/28.81F799050000/temperature
x     : 2
y     : 1
z     : 2
[28.C6E599050000]
uri: /home/pi/TempLog/28.C6E599050000/temperature
x     : 2
y     : 2
z     : 2
[28.F21047060000]
uri: /home/pi/TempLog/28.F21047060000/temperature
x     : 0
y     : 0
z     : 3
[28.A0F799050000] ----
uri: /home/pi/TempLog/28.A0F799050000/temperature
x     : 0
```

```
y     : 1
z     : 3
[28.0AEB46060000]
uri: /home/pi/TempLog/28.0AEB46060000/temperature
x     : 1
y     : 2
z     : 3
[28.8F3D47060000]
uri: /home/pi/TempLog/28.8F3D47060000/temperature
x     : 3
y     : 2
z     : 3
[28.A7CA99050000]
uri: /home/pi/TempLog/28.A7CA99050000/temperature
x     : 3
y     : 0
z     : 3
[28.931D9A050000]
uri: /home/pi/TempLog/28.931D9A050000/temperature
x     : 3
y     : 1
z     : 3
#start here
[28.F70A9A050000]
uri: /home/pi/TempLog/28.F70A9A050000/temperature
x     : 1
y     : 1
z     : 0
[28.D3169A050000]
uri: /home/pi/TempLog/28.D3169A050000/temperature
x     : 2
y     : 3
z     : 2
[28.F67B46060000]
uri: /home/pi/TempLog/28.F67B46060000/temperature
x     : 2
y     : 4
z     : 2
[28.860D48060000]
uri: /home/pi/TempLog/28.860D48060000/temperature
x     : 1
y     : 4
z     : 2
[28.B9C646060000]
uri: /home/pi/TempLog/28.B9C646060000/temperature
x     : 0
y     : 4
z     : 2
[28.A2E999050000]
uri: /home/pi/TempLog/28.A2E999050000/temperature
x     : 2
y     : 4
z     : 3
[28.3BBCAE050000]
```

```
uri: /home/pi/TempLog/28.3BBCAE050000/temperature
x     : 1
y     : 4
z     : 3
[28.671546060000]
uri: /home/pi/TempLog/28.671546060000/temperature
x     : 0
y     : 4
z     : 3
[28.B0039A050000]
uri: /home/pi/TempLog/28.B0039A050000/temperature
x     : 2
y     : 3
z     : 3
[28.D54447060000]
uri: /home/pi/TempLog/28.D54447060000/temperature
x     : 1
y     : 3
z     : 3
[28.EAE946060000]
uri: /home/pi/TempLog/28.EAE946060000/temperature
x     : 0
y     : 3
z     : 3
[28.F2BA99050000]
uri: /home/pi/TempLog/28.F2BA99050000/temperature
x     : 1
y     : 0
z     : 3
[28.37BCAE050000]
uri: /home/pi/TempLog/28.37BCAE050000/temperature
x     : 0
y     : 2
z     : 3
[28.091D48060000]
uri: /home/pi/TempLog/28.091D48060000/temperature
x     : 1
y     : 1
z     : 3
```

Appendix E: HDF5 file structure used in this study



	0	1
0	21.75	1.490623...
1	21.75	1.490623...
2	21.75	1.490623...
3	21.75	1.490623...
4	21.75	1.490623...
5	21.75	1.490623...
6	21.8125	1.490623...
7	21.75	1.490623...
8	21.8125	1.490623...
9	21.8125	1.490623...
10	21.8125	1.490623...
11	21.75	1.490623...
12	21.75	1.490623...
13	21.75	1.490623...
14	21.75	1.490623...
15	21.75	1.490623...
16	21.75	1.490623...
17	21.75	1.490623...
18	21.75	1.490623...
19	21.75	1.490623...
20	21.75	1.490623...
21	21.75	1.490623...
22	21.75	1.490623...
23	21.75	1.490623...
24	21.75	1.490623...
25	21.75	1.490623...
26	21.75	1.490623...
27	21.75	1.490623...
28	21.75	1.490623...
29	21.6875	1.490623...
30	21.6875	1.490623...
31	21.75	1.490623...
32	21.6875	1.490623...
33	21.6875	1.490623...
34	21.6875	1.490623...
35	21.6875	1.490623...
36	21.6875	1.490623...
37	21.6875	1.490623...
38	21.6875	1.490623...
39	21.6875	1.490623...
40	21.6875	1.490623...
41	21.6875	1.490623...