Imperial College London

EE3 Summer Group Project

Imperial College London

Department of Electrical and Electronic Engineering

Monitoring Infection Thermometer Design History File (DHF)

Author(s):
Sajid Ali
Yuze Du
Constantinos Kourris
Thomas Bishop
Xiangwei Jing

Graduate Supervisor:
Daryl Ma

Department Supervisor:
Professor Anthony G. Constantinides

June 25, 2020

Acknowledgements

Throughout the prototyping phase we received regular input from professor Anthony G. Constantinides - Imperial College London. He developed the algorithms from which our design is based and also provided valuable assistance when it came to resolving issue associated with implementation and optimisation. We could also like to give thanks to Daryl Ma who attended all of our weekly meetings and helped to structure the project to ensure we could meet our deadlines.

Abstract

The purpose of this document is to serve as a record of the work done and decisions made during the prototyping phase of a medical device created during the Summer term of 2020. This document is also a starting point for groups who would like to develop this product further, or as reference to existing work in this field. The first three chapters explains the client brief and the subsequent design specification our team derived. It goes on to give justifications of the design specifications we settled on along with a verification matrix that shows how well our final prototype met these requirements. The remaining chapters explain how the project was modularised. A chapter is dedicated to the contributions of each team. In depth explanations are given for the design rationale along with the technical challenges of implementation and design verification. Finally, the document is formatted to comply to the U.S. Food and Drug Administration requirements around design control of medical devices. Certain sections are shortened because of lack of time and convenience.

Contents

1 Introduction **7**

2 Design Input Documentation **8**

2.1 Client Requirements . . . 8

2.2 Product Specifications . . . 8

2.3 System Risk Analysis (hazards and source identification) . . . 8

3 Design Output Documentation **9**

3.1 Product and Process Design Outputs (drawings and instructions) . . . 9

3.2 Design Verification . . . 10

4 Hardware **11**

4.1 Experiment Design . . . 11

4.1.1 Before the experiment . . . 11

4.1.2 Water Bath Experiment . . . 11

4.1.3 Body Test . . . 14

4.2 Dual-heat-flux Method . . . 14

4.3 2-sensor Arrangement . . . 16

4.3.1 Theory . . . 16

4.3.2 Implementation . . . 16

4.4 3-sensor Arrangement . . . 17

4.4.1 Theory . . . 17

4.4.2 Implementation . . . 18

4.5 4-sensor Arrangement 1 (Kitamura Method) [Final Solution] . . . 18

4.5.1 Theory . . . 18

4.5.2 Implementation . . . 20

4.6 4-sensor Arrangement 2 . . . 22

4.6.1 Theory . . . 22

4.6.2 Implementation . . . 22

4.7 Sensor arrangement Choice . . . 23

4.8 Other Details . . . 23

5 Algorithm **24**

5.1 2-Sensor Mathematical Model . . . 24

5.2 4-Sensor Mathematical Model . . . 26

5.3 Machine Learning as a Prediction Model . . . 30

6 Base Station Application **33**
6.1 Preliminary Designs 33
6.2 v0.1: Application Framework 33
6.2.1 What is a Framework? 33
6.2.2 Native Applications 35
6.2.3 Why Flutter? 35
6.2.4 File Structure Hierarchy? 36
6.3 v0.2: Bluetooth Communication 36
6.3.1 Why Bluetooth? 36
6.3.2 Implementation 37
6.4 v0.3: Firebase Authentication and Database 38
6.4.1 Database as a Service 40
6.4.2 Firebase and Flutter 40
6.5 v0.4: UI Overhaul 42
6.5.1 Connection to Algorithm API 42
6.5.2 Event Manager 43
6.5.3 Adding Graphs 43

7 Recommended Updates **45**
7.1 Custom PCB and Use of 3D printer 45
7.2 Sensor Arrangement 45
7.3 Train Model with More Data 45

List of Figures

3.1 System Overview . . . 9

4.1 Overview of the hardware design . . . 12
4.2 Overview of the hardware usage . . . 12
4.3 Sensor Check . . . 13
4.4 Water bath Experiment . . . 13
4.5 Heating Coil . . . 13
4.6 Complete Experiment . . . 13
4.7 Body Test . . . 14
4.8 Core Body Temperature . . . 15
4.9 Dual-heat-flux method . . . 15
4.10 2 sensor-arrangement . . . 16
4.11 T1 vs T2 (2 sensor-arrangement) . . . 17
4.12 3-sensor arrangement . . . 18
4.13 4-sensor arrangement 1 . . . 19
4.14 4-sensor arrangement 1 Improved . . . 19
4.15 Plots of 4 temperatures in the 3rd test of 37^oC . . . 20
4.16 Plots of 4 temperatures in the body test . . . 21
4.17 4-sensor arrangement 2 . . . 22

5.1 Model of the 2 sensor hardware implementation . . . 24
5.2 Simulating the temperature measurements of the probe . . . 26
5.3 Predicting Tc on the simulated environment . . . 27
5.4 Model of the 4-sensor hardware implementation . . . 27
5.5 Simulated temperatures using the 4-sensor models . . . 29
5.6 Predicted Tc on simulation values and the effect of post processing . . . 29
5.7 Overview of how the final predictive model operates . . . 30
5.8 Data collected from hardware measurements . . . 31
5.9 Predicted Tc from real data using a neural network and post processing . . . 31

6.1 Preliminary Design 1 . . . 34
6.2 Preliminary Design 2 . . . 34
6.3 Preliminary Design 3 . . . 34
6.4 Flutter Framework Icon: flutter is a UI framework for creating mobile multi-platform mobile applications. . . . 35
6.5 Flutter File Structure v0.1 . . . 36
6.6 *ActiveConnection* and *Bluetooth* Class Overview . . . 37
6.7 Establishing a Bluetooth Connection . . . 38
6.8 Receiving Data . . . 39

6.9 Bluetooth Connection UI 39
6.10 Flutter File Structure v0.2 40
6.11 Flutter File Structure v0.3 42
6.12 Final Application User Interface 44
6.13 Flutter File Structure v0.4 44

7.1 Flutter File Structure 45

List of Tables

4.1 Time cost to reach linear region ($T_1 \propto T_2$) 17
4.2 K value in different water-bath temperature (3-sensor arrangement) 18
4.3 Final Results of Arrangement 1 20
4.4 Body Test Results of Arrangement 1 21
4.5 Actual λ values 22
4.6 Final Results of Arrangement 2 23

Chapter 1

Introduction

Core Body Temperature (CBT) refers to the temperature of the body's internal organs, such as the heart, liver, brain and blood [1]. Monitoring CBT is a necessary tool of diagnosis in modern medicine. Accurate measurements are necessary in a variety of scenarios such as detecting fevers, hypothermia, ITU management etc.

Current commercially available non-invasive methods for measuring CBT include tympanic and Axillary/Forehead thermometers which give readings of up to 1.3C less than the actual CBT[2], not to mention that these solutions cannot give continuous accurate readings and most cannot store data for future analysis. The only way to currently achieve continuous clinically accurate readings is to use a rectal thermometer. But this is inconvenient for uses outside of surgery for obvious reasons.

We created a non-invasive method of measuring CBT through the use of a skin patch which measures relative heatflux to estimate the CBT. Similar methods have been attempted using zero/duel heatflux methods. But most suffer from ambient temperature disturbance such as when it is too cold[3].

We hope that the system we have created can aid further development in this field and highlight some of the technical challenges that require new and novel solutions to overcome.

Chapter 2

Design Input Documentation

2.1 Client Requirements

Our brief was to create a device which could **detect feverish symptoms** quickly and efficiently, while being **non-invasive** and relatively inexpensive. We were also asked to ensure that **continuous readings** could be taken with all the **data available for download** by the user.

Our starting point was a **family of novel estimation algorithms** used to estimate Core Body Temperature (CBT) using skin temperature readings.

2.2 Product Specifications

From the aforementioned product requirements we derived a product specification which we believe satisfies the design criteria with justifications given for each specification.

Specification 1: ***Mobile Application Capable of Two Way Communication with Device***
The reliance of heavy processing on the part of our proprietary algorithm necessitates the creation of an application which can handle this along with being the system through which the user interacts with the device.

Specification 2: ***Result Latency within 3 minutes***
Having continuous up to date readings in crucial in early diagnosis.

Specification 3: ***Achieve Clinical Accuracy (within 0.2C)***
The unique selling point of the device is that it can achieve a clinically accurate result while being non-invasive.

2.3 System Risk Analysis (hazards and source identification)

Growing awareness of the collection of user data on a mass scale has prompted many legislators to pass laws to protect the data sovereignty of individuals. This awareness is only likely to grow as more data is collected and more uses for it are discovered. The final product we develop must ensure that users are protected from any sensitive information leaking. To this end transparency about how information is collected and used is crucial, as well as giving the user the ability to delete data permanently is they so desire.

Ensuring that the final product is manufactured in a way that is as sustainable as possible will reduce its environmental footprint and give the product resilience to future environmental regulation. To this end the sources of greatest environmental degradation, such as the battery, should be considered. For example, if the sensor can be minimised sufficiently, a passive power-source such as skin heat might be feasible.

Chapter 3

Design Output Documentation

3.1 Product and Process Design Outputs (drawings and instructions)

An overview of the entire system if shown in Figure 3.1. Each of these systems are explained in the subsequent chapters. But to summarise:

- The micro-controller on the sensor patch reads the temperature values from the many thermometers after regular intervals.
- These values are stored in a buffer and once full are formatted then transmitted to the base-station.
- The base-station application queries the algorithm API with the data it receives and displays the result of the calculation to the user.
- The application also pulls historic data from its backend and displays these to the user in an intuitive manner.

All the code implementation can be found at the following github repo link: https://github.com/Sajid-Ali-Imperial/Thermomet-Group-Project/commit/1f57bbe7362f5665d6bb36118c69933518ad4c26

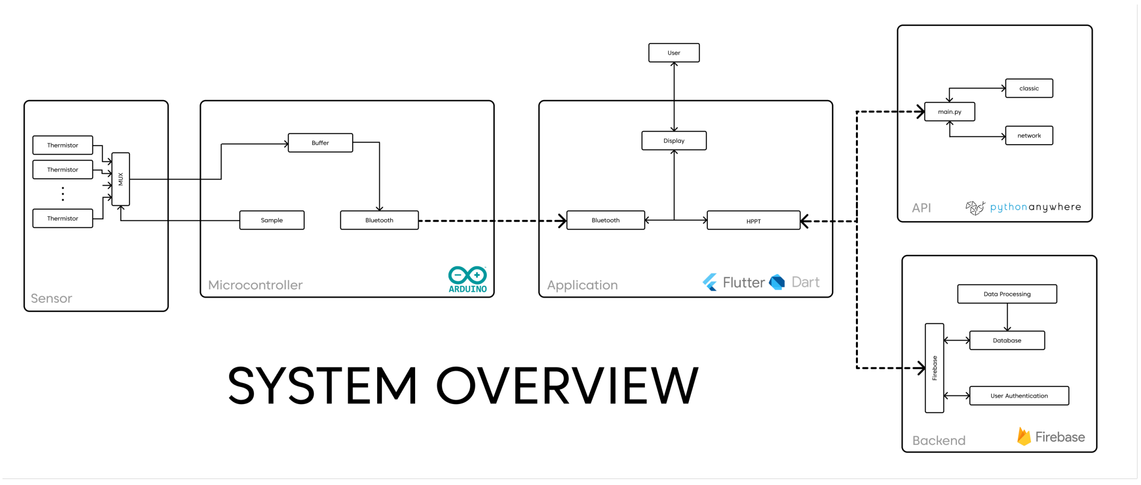


Figure 3.1: System Overview

3.2 Design Verification

Specification	**Target**	**Actual**
Communication	Two Way	Two Way
Result Latency	180s	165s
Clinical Accuracy	0.2C	0.2C

Chapter 4

Hardware

The overview of the hardware design is shown in figure 4.1. The whole device contains two parts, arm cuff and base, where probe is built in arm cuff and base device contains power units, processor unit and data transmission unit. Figure 4.2 showed the overview of the hardware usage. The strap is designed to be adjustable, which allows measurement done on different parts of the body (e.g. chest cavily, legs, arm). We did measurements on arms for convinience.

4.1 Experiment Design

4.1.1 Before the experiment

1. Sensor temperature check. Leave sensors in room temperature for 3-5 minutes before reusing them.
2. Water bath temperature check. In water bath experiment, water bath temperature is adjusted to 37 ^oC for normal cases and 40 ^oC for extreme cases.
3. Breadboard and Microprocessor connection check.

4.1.2 Water Bath Experiment

In waterbath experiment, the purpose is using adjustable water bath to simulate the core body environment and find the difference between estimated temperature and actual temperature. The converging temperature and converging time heavily depend on environmental temperature.

Complete Experiment The Temperature sensors used in this experiment are DS18B20 Programmable Resolution 1-Wire Digital thermalmeter. And figure 4.3 shows that we test sensors by comparing the results with that of standard thermalmeter to ensure the functionality of every sensor. After comparing the results in 10 different place, the difference is well under $0.05 o_C$, which confirmed the reliability of the sensor. Figure 4.4 shows the scratch of the experiment design and figure 4.6 shows the actual implementation of water bath experiment. A pan was used as container of water bath. Furthermalre, in order to make the experimental results more reliable, all the experiments are repeated at least 3 times.

Temperature Control: Challenges The major challenge for water bath experiment is to control the temperature of the water. A 2cm long heater shown in Figure 4.5 is put underneath the pan and is used to constantly heat the water during the experiment. More specifically, a PWM wave is applied to the heater, then, by controlling the duty cycle, the heater could heat the water to certain temperature. In practice, the target temperature of the heater is designed to be 0.2 ^oC higher than the required one considering the heat lost. After monitoring the temperature of water bath with a standard thermalmeter shown in Figure 4.3, the error for this temperature control system is $\pm$ 0.1^oC.

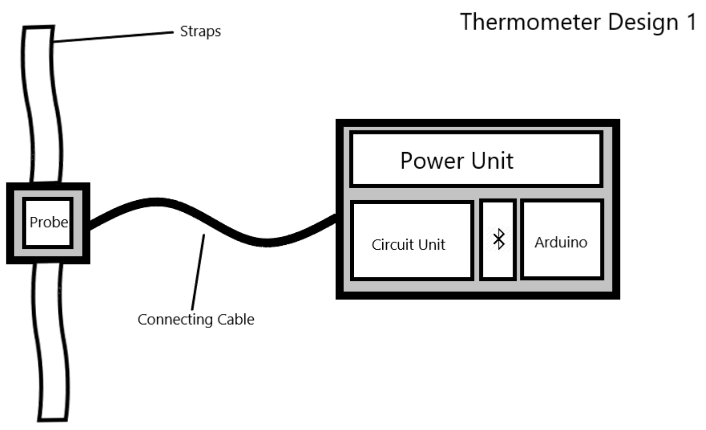


Figure 4.1: Overview of the hardware design

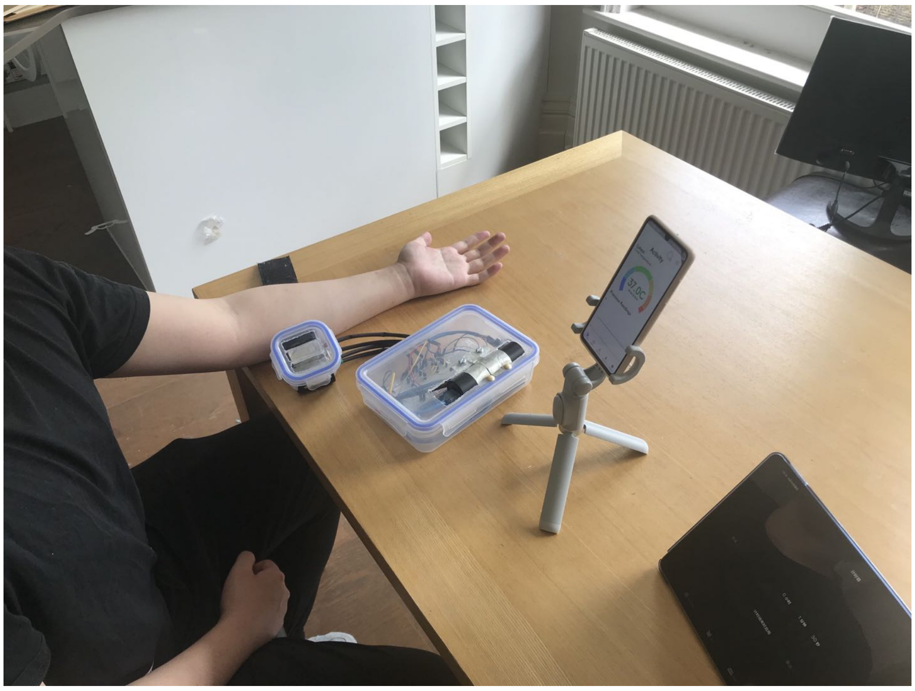


Figure 4.2: Overview of the hardware usage

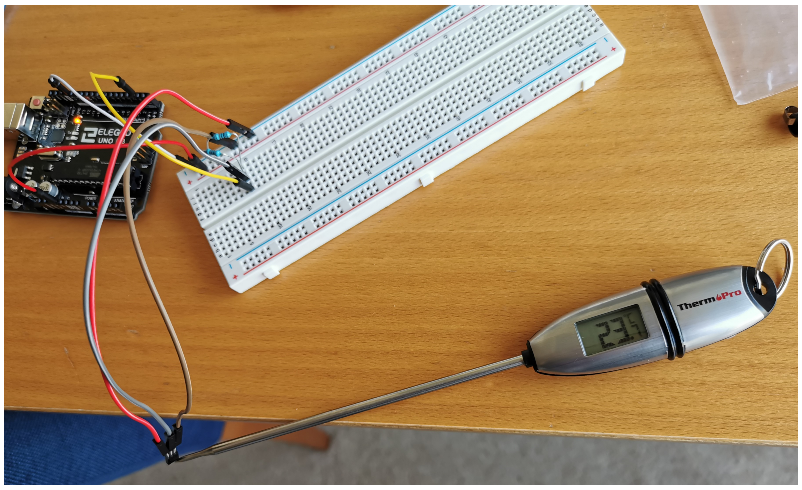


Figure 4.3: Sensor Check

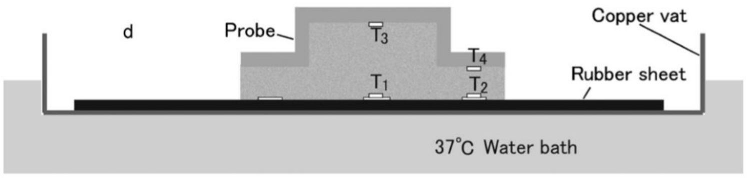


Figure 4.4: Water bath Experiment

Figure 4.5: Heating Coil

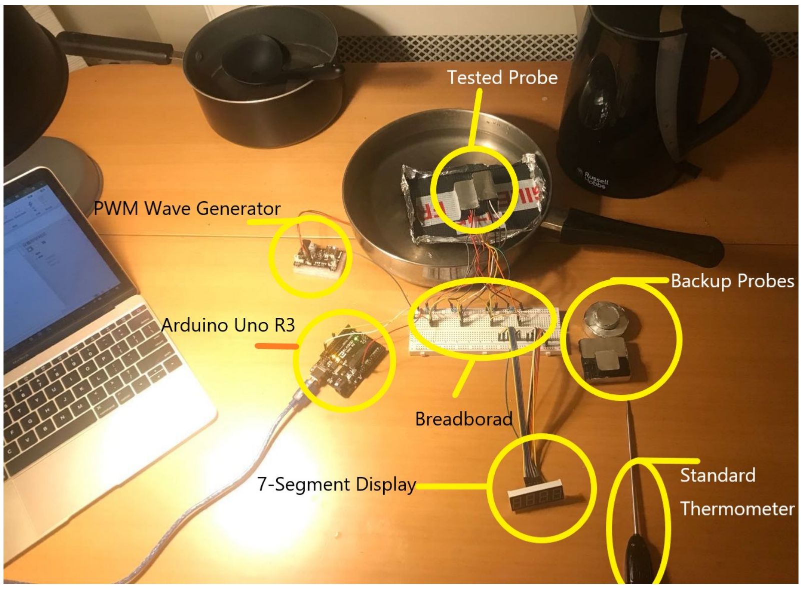
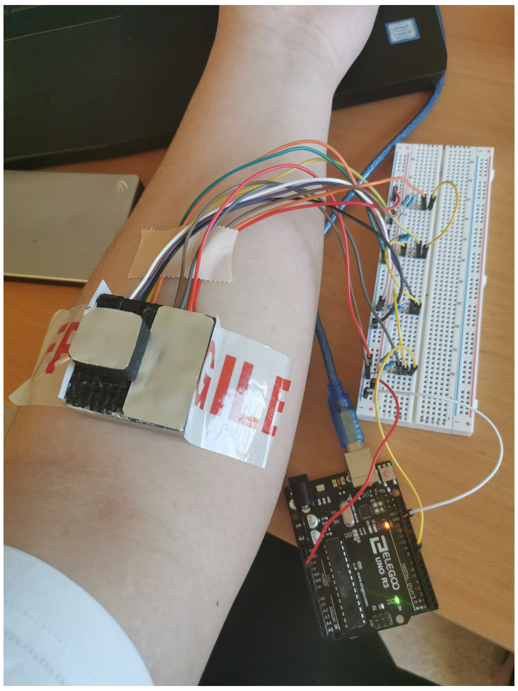


Figure 4.6: Complete Experiment


(a) Measurement on arm

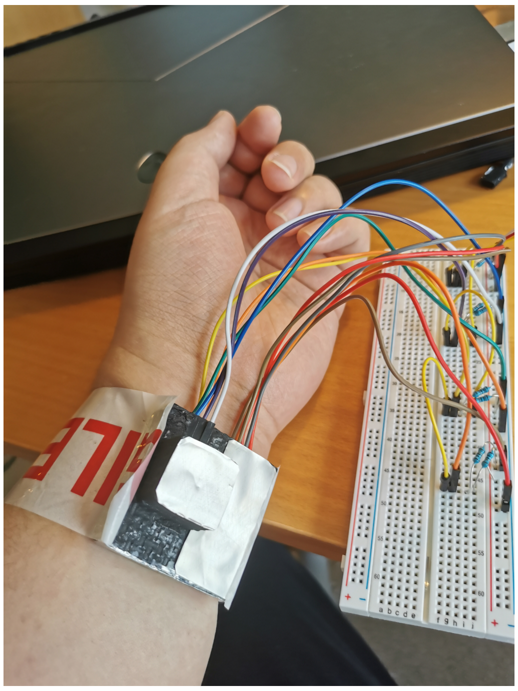

(b) Measurement on wrist

Figure 4.7: Body Test

4.1.3 Body Test

Experiment Notifications

1. Due to the principles of the kitamura Arrangement, please hold the side of the probe rather than the top of it.
2. As the rubber inside the probe is water-absorbing, and the thermal-resistance will change after absorbing water. So please leave the probe away from water.
3. As, for both tracks, we assume the thermal-resistance between skin and core body are the same, so make sure the two sensor are on the flat plane on your body .

As shown in figure 4.7, we also did tests on body to checked the availability of the probes. In order to make sure the results are reliable, we chose different part on body to apply the tests. Arm wrist,chests has different thicknesses of fat, and different core body temperature (We did test in both warm and cold environments the theoretical distributions are shown in figure 4.8.[4] While doing the test, we measure the temperature of mouth cavity as a comparison of the results and mouth cavity temperature is 1.0 1.5^oC lower than deep body temperature under chest cavity in normal situation. However,as the mouth cavity has no certain relation with body temperature, this test could only prove that the results are in reasonable range regardless of position of skin. As some of our attemption failed in previous water bath experiment. Only the final sensor arrangement was tested on body. And results and be found in table 4.4.

4.2 Dual-heat-flux Method

Dual-heat-flux Method (shown in figure 4.9 is one of the most widely-used method to measure deep-body temperature. The principle for this method is having two track of heat transmission and simulate them as electrical circuit, where heat media as resistor and temperature as electrical potential. In that case the unknown temperature could be acquired by rearranging the equation 4.1 and 4.2. In the following part, we will introduce alternative solution for Single-heat-flux Method (2-sensor arrangement) and Dual-heat-flux method (3 and 4-sensor arrangements).

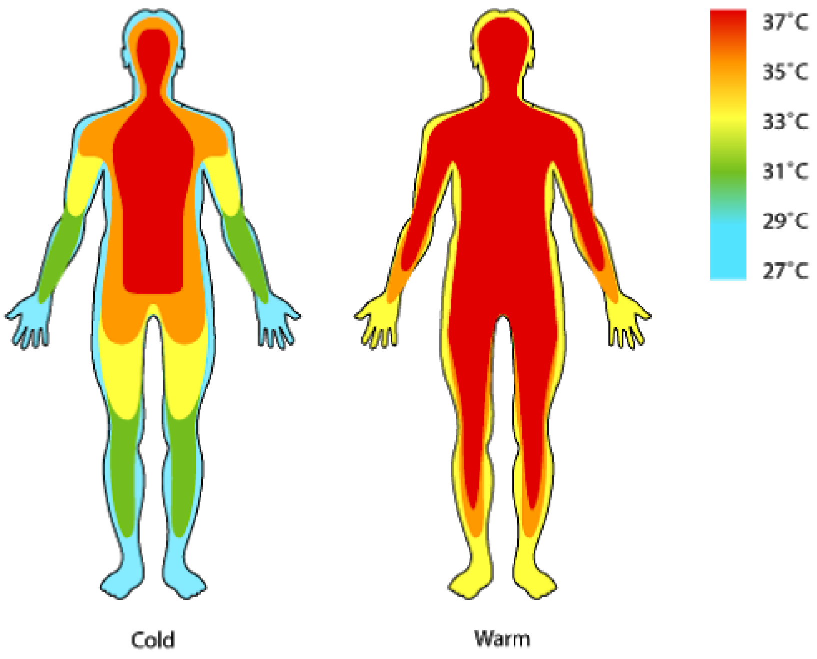


Figure 4.8: Core Body Temperature

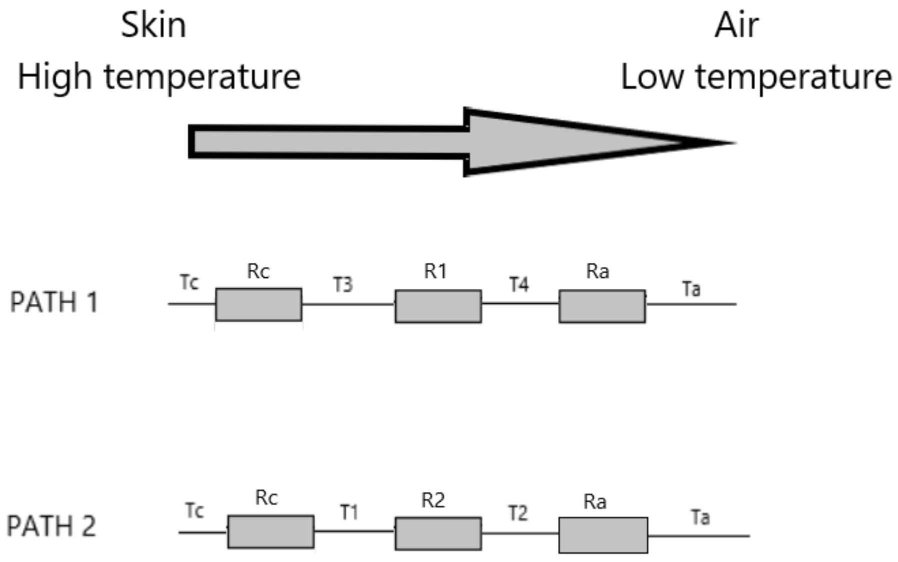


Figure 4.9: Dual-heat-flux method

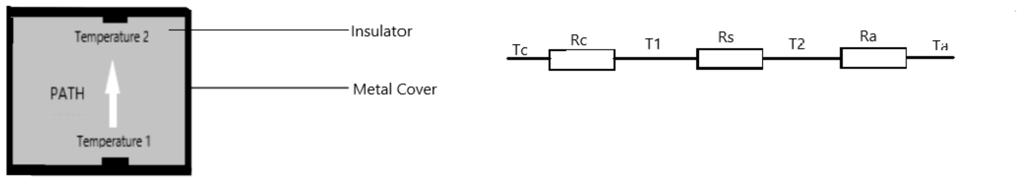


Figure 4.10: 2 sensor-arrangement

$$\frac{T_c - T_1}{R_c} = \frac{T_1 - T_2}{R_2} \tag{4.1}$$

$$\frac{T_c - T_3}{R_c} = \frac{T_3 - T_4}{R_1} \tag{4.2}$$

4.3 2-sensor Arrangement

The scratch in Figure 4.10 showed the arrangement of 2 sensors,which is the minimum-sensor arrangement that could possibly work. And the idea and implementation guidance are provided by prof. Constantinides.[5]

4.3.1 Theory

From basic equation of heat flux method mentioned in previous section:

$$T_2 = T_1 - \frac{R_s}{R_c} * (T_c - T_1) \tag{4.3}$$

Rearrange

$$T_1(n) = a * T_2(n) + b * T_c(n) \tag{4.4}$$

where

$$a = \frac{1}{1 + \frac{R_s}{R_c}} \tag{4.5}$$

$$b = \frac{\frac{R_s}{R_c}}{1 + \frac{R_s}{R_c}} \tag{4.6}$$

We shall assume that the changes in Tc over some short period of time are not significant enough to be considered, thereby rendering the second term b*Tc(n) in equation 4.4 a constant, unknown but over the period of observation we assume it to be constant.

Under these assumptions equation 4.4 describes the linear equation $y = m * x + C$. Equations with gradient and interceptions could solve core body temperature.

4.3.2 Implementation

The theory for 2-sensor plan is the data from two sensors are proportional to each other after several minutes assuming core body temperature are constant. Equations with gradient and interceptions could solve core-body temperature. However, the problem is the uncertainty in time cost to get to linear region for different environment temperature, so we gave up. The plots of T1 vs T2 are shown in figure 4.11 and time cost to reach linear region is shown in table 4.1

Set Temperature(^oC)	Test number	Time for T1-T2 plot become linear (second)
37	1	74.45
	2	56.85
	3	80.44
40	1	90.87
	2	100.56
	3	104.11

Table 4.1: Time cost to reach linear region ($T_1 \propto T_2$)

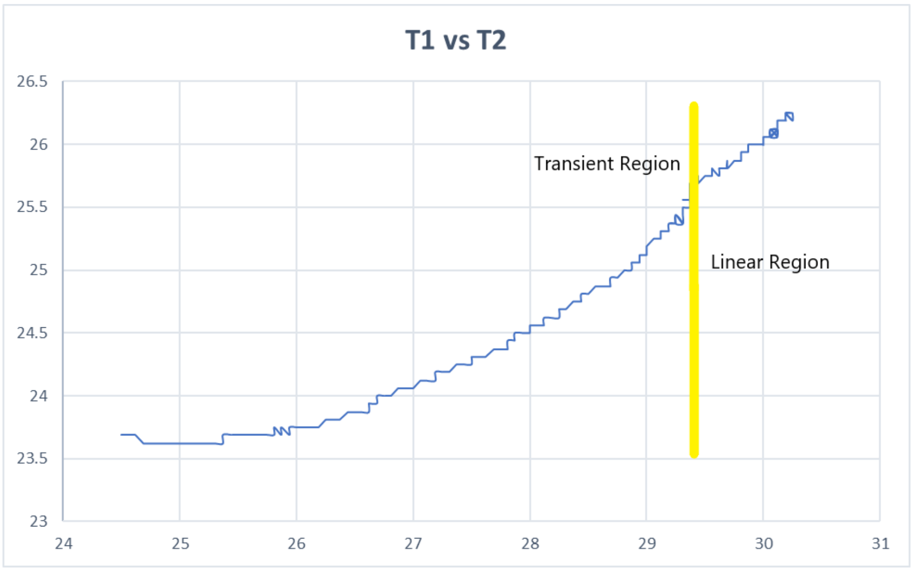


Figure 4.11: T1 vs T2 (2 sensor-arrangement)

4.4 3-sensor Arrangement

The scratch in Figure 4.12 showed the arrangement of 3 sensors based on an alternative version of dual-heat-flux method. This method was inspired by Professor Constantinides's idea.[6].

4.4.1 Theory

The basic equation is similar to equation 4.1 and 4.2

$$\frac{T_c - T_1}{R_c} = \frac{T_1 - T_3}{R_2} \tag{4.7}$$

$$\frac{T_c - T_2}{R_c} = \frac{T_2 - T_3}{R_1} \tag{4.8}$$

By rearranging previously mentioned equation 4.7 and 4.8, the equation for Core body temperature is

$$T_c = T_2 + \frac{(T_2 - T_1) * (T_2 - T_3)}{K * (T_1 - T_3) - T_2 + T_3} \tag{4.9}$$

Where

$$K = \frac{R_1}{R_2} = \frac{(T_c - T_2)(T_1 - T_3)}{(T_c - T_1)(T_2 - T_3)} \tag{4.10}$$

As K is only determined by characteristics (material, structure and etc.) of the probe. So we assume $T_c = 37^oC$ and find the value of K. Use this K value as a constant to measure other Core body temperature.

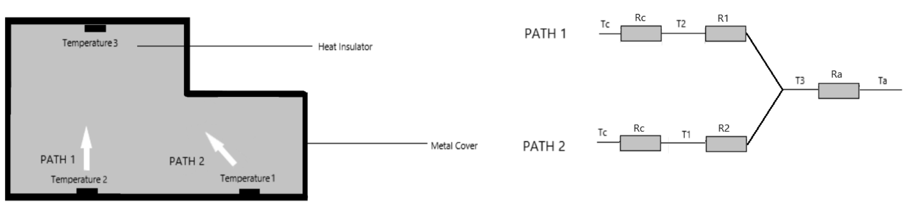


Figure 4.12: 3-sensor arrangement

Set Temperature(^oC)	Test number	K-value
37	1	1.574
	2	1.572
	3	1.577
40	1	1.711
	2	1.724
	3	1.715

Table 4.2: K value in different water-bath temperature (3-sensor arrangement)

4.4.2 Implementation

Calculate K Measure temperatures from 3 sensors and substitute $Tc = 37^oC$ into equation 4.10. The results are shown in table 4.2

Consistency Test in 40^oC In theory, the value of K should not change with core body temperature, ambient temperature and other external factors. So we increase the temperature of water bath to 40^oC and repeated tests. However, thermal-expansion becomes a problem, the length R1 expand in y-axis while R2 expand in both, which changes R1 over R2.

4.5 4-sensor Arrangement 1 (Kitamura Method) [Final Solution]

This arrangement was firstly developed by Kei-Ichiro Kitamura in his publication[7], it provides a non-invasive method for deep-body temperature measurement. Scratch in figure 4.13 showed the arrangement of sensors in Kitamura Arrangement. But in practice, this arrangement cause a problem, which is the two track are too close to each other and might have impact on each other. So a improved scratch is shown in figure 4.14

4.5.1 Theory

Arrangement 1 is based on Kitamura Arrangement. Similarly by rearranging equation 4.1 and 4.2. The equation for Core body temperature is

$$T_c = T_3 + \frac{(T_3 - T_1)(T_3 - T_4)}{K(T_1 - T_2) - (T3 - T4)} \tag{4.11}$$

Where

$$K = \frac{R_1}{R_2} = \frac{(T_c - T_1)(T_3 - T_4)}{(T_c - T_3)(T_1 - T_2)} \tag{4.12}$$

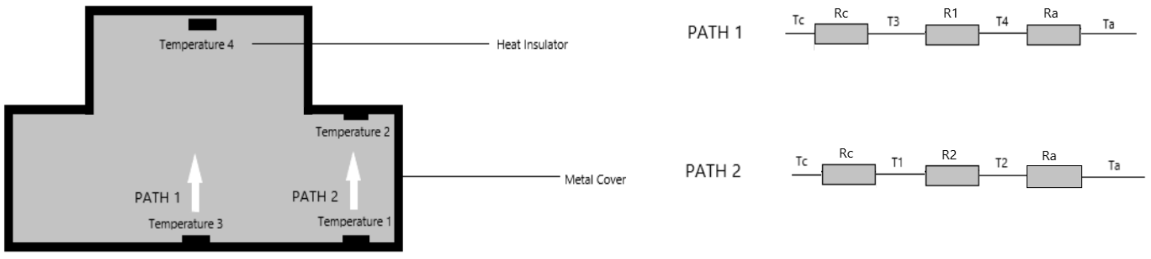


Figure 4.13: 4-sensor arrangement 1

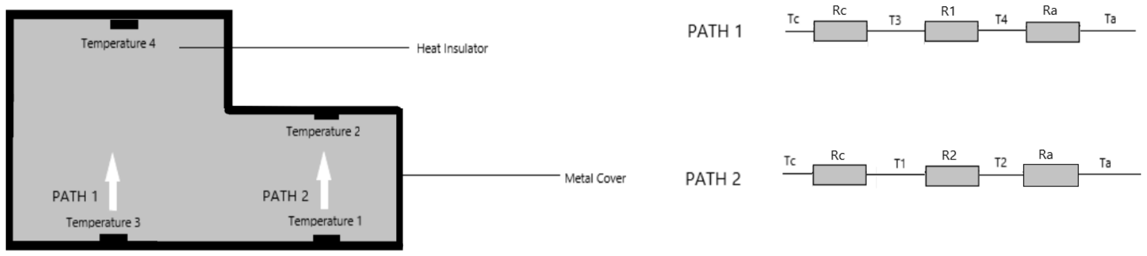


Figure 4.14: 4-sensor arrangement 1 Improved

Set Temperature(^oC)	Test number	Test Results(^oC)	Error(^oC)	Error Pencentage(%)
37	1	36.64	0.36	0.972
	2	37.08	0.08	0.216
	3	37.13	0.13	0.351
40	1	40.02	0.02	0.050
	2	39.46	0.54	1.350
	3	40.14	0.14	0.350

Table 4.3: Final Results of Arrangement 1

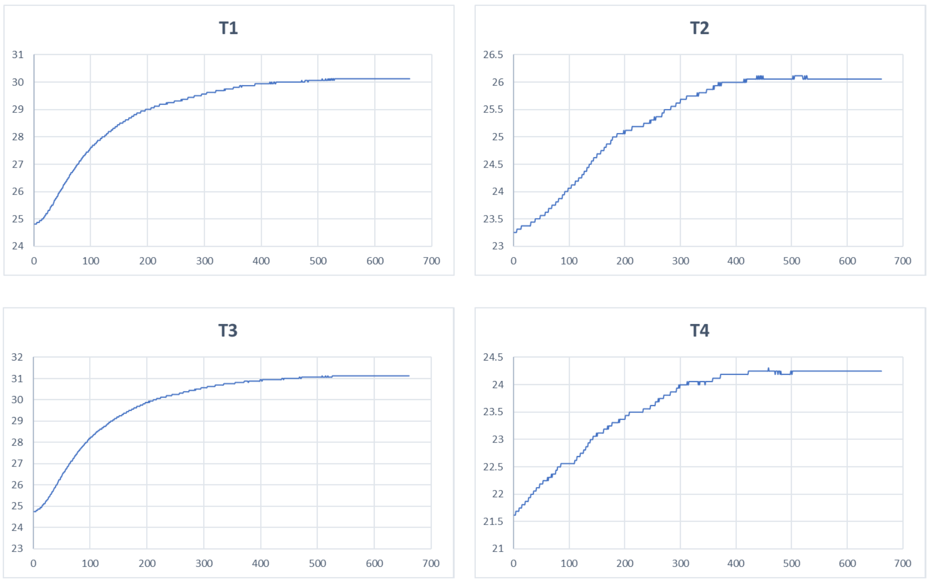


Figure 4.15: Plots of 4 temperatures in the 3rd test of 37^oC

Similar to previous arrangement, as K is only determined by characteristics (material, structure and etc.) of the probe. So we assume $T_c = 37^oC$ and find the value of K. Use this K value as a constant to measure other Core body temperature.

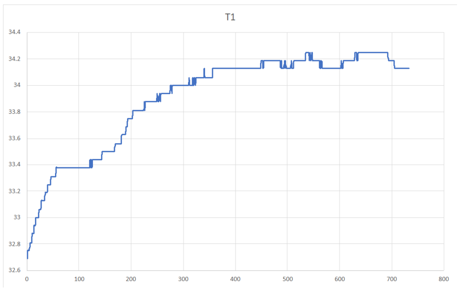
4.5.2 Implementation

Calculate K Collect data from 4 sensors and substitute $Tc = 37^oC$ into equation 4.12.

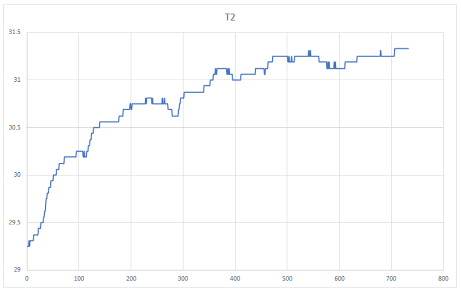
Consistency Test in 40^oC In theory, the value of K should not change with core body temperature, ambient temperature and other external factors. In this step, the temperature of the water bath will be set to 40^oC . In the 3 experiments, the values of K converges to 1.88, 1.87, 1.88 respectively. Thus, consistency of K is confirmed.

Core Body Temperature Measurement In this step, we set the water bath temperature, namely core body temperature, to 40 and 37 ^oC. Experiments will be done for 3 times for each temperature. The final results of all the 6 tests are shown in Table 4.3. Also, the plots of 4 temperatures in the 3rd test of 37^oC water bath are shown in figure 4.15. From the plot, the trend of temperatures can be approximated as an exponential equation.

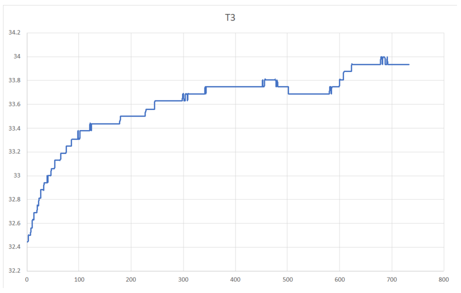
Body test Results of body tests in chest cavity shown in figure 4.16 are less regular, which is caused by complicated processes inside body, such as blood flow.



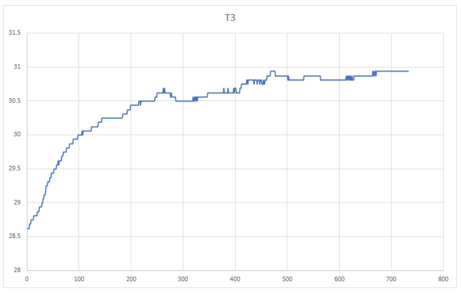
(a) T1



(b) T2



(c) T3



(d) T4

Figure 4.16: Plots of 4 temperatures in the body test

Position of measurement	Warm Environment Test Results(^oC)	Cold Environment Test Results(^oC))
Wrist	33.64	28.36
	34.08	29.08
	34.13	29.13
Arm	36.02	32.02
	36.46	32.54
	36.14	32.14
Chest	37.12	37.21
	37.40	37.15
	37.11	37.24

Table 4.4: Body Test Results of Arrangement 1

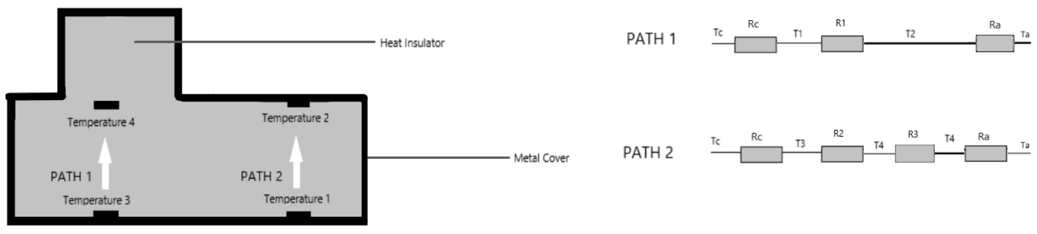


Figure 4.17: 4-sensor arrangement 2

Set Temperature(^oC)	37			40		
Test Number	1	2	3	1	2	3
λ	1.00	0.99	1.01	1.00	1.01	0.99

Table 4.5: Actual λ values

4.6 4-sensor Arrangement 2

The scratch in Figure 4.17 showed the arrangement of sensors in alternative solution with 4-sensors. This solution is also based on dual-heat-flux method, provided by prof. Anthony G.Constantinides.[8]

4.6.1 Theory

Arrangement 2 is to calculate the core body temperature by equation 4.13. In this method T_3 and T_4 are in same distance to skin so $R_1 = R_2$. Therefore, K=1.

$$T_c = T_1 + \frac{(T_1 - T_3)(T_1 - T_2)}{T_3 - T_4 - T_1 + T_2} \tag{4.13}$$

Before using equation 4.13, we are supposed to make two assumptions. The first one is R_c in Path 1 and 2 are the same, which is also necessary for Arrangement 1. The second assumption is $R_1 = R_2$. However the second assumption may not be true for the probe we use. Therefore, we should firstly define $\lambda = \frac{R_1}{R_2}$ and make sure it equals to 1. However, if the second assumption failed, then the original equation becomes

$$T_c = T_1 + \frac{(T_1 - T_3)(T_1 - T_2)}{\lambda(T_3 - T_4) - T_1 + T_2} \tag{4.14}$$

Where

$$\lambda = \frac{R_1}{R_2} = \frac{(T_c - T_3)(T_1 - T_2)}{(T_c - T_1)(T_3 - T_4)} \tag{4.15}$$

4.6.2 Implementation

Calculate the value of λ in 40 and 37^oC As shown in table 4.5, the value of λ varies between 0.99 to 1.01, so for the probe we made, assumption 2 is proven correct.

Core Body Temperature Measurement Similar to Arrangement 1, the measurement will be done in 37 and 40^oC and, for each temperature, we applied 3 experiments.

Set Temperature(^oC)	Test number	Test Results(^oC)	Error(^oC)	Error Percentage(%)
37	1	37.40	0.40	1.081
	2	34.75	2.25	6.081
	3	40.06	3.06	8.270
40	1	39.50	0.50	1.250
	2	41.80	1.80	4.500
	3	39.74	0.26	0.650

Table 4.6: Final Results of Arrangement 2

4.7 Sensor arrangement Choice

By comparing the results acquired from 4 arrangements above, it is obvious that the 4-sensor arrangements are more realistic solutions. The reason for that is K for Arrangement 1 and λ for Arrangement 2 has different tolerance for error. For instance, a 0.01 difference in λ could cause 1-2 ^oC difference in final results. However, as seen in 4-sensor Arrangement 1, 0.01 difference in K only caused 0.5^oC drop in final results. So considering the accuracy and robust, 4-sensor Arrangement 1 should be the final choice.

4.8 Other Details

For the choice of data transmission, we used Bluetooth because it's more stable and unlikely get interrupted by internet connection problem. For internal material, we used rubber as heat insulator because temperature trends cannot be analyzed if it change so fast. And the aluminium cover outside the probe could reduce heat radiation.

Chapter 5

Algorithm

Introduction The algorithm for predicting core body temperature was developed while closely communicating with the hardware team. While different design implementations were considered and experimented with, mathematical models were created to indicate how reliable the predictions for each method could be.

The major concepts considered were 2, 3, and 4 sensor configurations. In this section we will outline the operation of each one and why we opted to move forward with the 4-Sensor implementation.

Finally, we will show the formulae behind our simulations and how machine learning was introduced to the prediction model.

5.1 2-Sensor Mathematical Model

For all implementations work began by modelling the hardware as a thermal resistance circuit as seen in Figure 5.1.

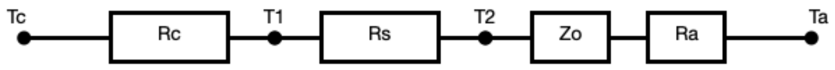


Figure 5.1: Model of the 2 sensor hardware implementation

Tc resembles the core body temperature, Ta the ambient temperature, and T1 and T2 the temperature measurements of the two sensors in the design. Rc resembles the thermal resistance from the core body to the surface of the skin, and Rs the thermal resistance inside the probe between the 2 temperature sensors. We assume Zo to be of the form $\frac{Ro}{1+s\tau_0}$, where τ_0 is an unknown time constant, to model any thermal mass within the probe, while Ra is the thermal resistance of the probe to the ambient temperature.

The values of the thermal resistances and impedances are unknown to us, therefore, any solution of Tc must be in terms of T1, T2, and possibly Ta.

By doing circuit analysis on the model in Figure 5.1, we can arrive to the equation,

$$\frac{Tc - T1}{Rc} = \frac{Tc - Ta}{Rc + Rs + Zo + Ra} = \frac{T1 - T2}{Rs} \tag{5.1}$$

By breaking down Equation 5.1 and solving for for T1 and T2, we obtain the two equations for the temperatures.

$$T1 = Tc - \frac{Rc\,(Tc - Ta)}{Rc + Rs + Zo + Ra} \tag{5.2}$$

$$T2 = T1 - \frac{Rs\,(Tc - T1)}{Rc} \tag{5.3}$$

To find a solution for Tc, we start by rewriting Equation 5.3 as

$$T1 = aT2 + bTc \tag{5.4}$$

where $a = \frac{1}{1+k_c}$ and $b = \frac{k_c}{1+k_c}$, and $k_c = \frac{Rs}{Rc}$. We now point out that since Rs and Rc are both constants, so are a and b. We now consider that both T1 and T2 are functions of time, determined by the heat flow within the probe, and dependant on Tc. The core body temperature, Tc, can also vary with time, however since the body's function is to keep Tc stable, rate of change is low. Therefore we will make the assumption that the change is Tc, over a short period of time, is insignificant and thus a constant.

This enables us to view Equation 5.4 as an equation of a straight line with a gradient of a and y-intercept of bTc. What we desire is Tc, which is obtainable through experimental data.

In our implementation, measurements of T1 and T2 were made simultaneously, with an interval of $1.5s$ between each measurement. All data points could then be plotted on a graph of T1 against T2. Our implemented algorithm grouped the data points using a moving window of length 5 and obtained the line of best fit using the least squared method. By finding the best fit line, the gradient could be obtained which will then be used to calculate kc as initially shown in Equation 5.4. Finally, by also obtaining the y-intercept of the best fit line, and using kc, Tc can be found.

The decision for the window length is crucial for the accuracy of the predictions. As mentioned previously, the assumptions made are dependant on the small variation of Tc over the measurement period, and thus a large window length will lead to unreliable predictions. Too small of a size can lead to volatile predictions that are largely affected by probe and other noise.

Initial testing of the algorithm was done on simulations of the hardware, which were derived from the mathematical analysis of the model and the previously shown equations.

We start by expanding Zo in Equation 5.2 and rewriting it to obtain the equation

$$T1 = Tc - H_1(Tc - Ta) \tag{5.5}$$

$$H1 = \frac{Rc(1 + s\tau_0)}{(Rs + Rc + Ro + Ra)(1 + s\tau_1)}$$

$$\tau_1 = \frac{Rs + Rc + Ra}{Rs + Rc + Ro + Ra}\tau_0$$

The reason for rewriting the equation in this way is because the equations so far have been in continuous time, but sampling is done in discrete time. To build the simulator we first need to transform the equations in discrete time by applying Z-Transform. This format enables for simpler calculations, and by applying the Z-Transform on Equation 5.5 we obtain

$$T1(z) = Tc(z) - G_1(z)(Tc(z) - Ta(z)) \tag{5.6}$$

$$G_1(z) = \frac{g_1(1 - \rho_0 z^{-1})}{1 - \rho_1 z^{-1}}, \quad g_1(z) = \frac{Rc(1 - \rho_1)}{(Ro + Rc + Rs + Ra)(1 - \rho_0)}$$

$$\rho_0 = e^{-T/\tau_0}, \quad \rho_1 = e^{-T/\tau_1}$$

where T is the period or the time interval between each measurement. After some final manipulations and we arrive to the discrete time equation for T1

$$\begin{aligned} T1(n) =& \rho_1 T1(n-1) + Tc(n) - \rho_1 Tc(n-1) - g_1(Tc(n) - Ta(n)) \\ &+ \rho_0 g_1(Tc(n-1) - T_a(n-1)) \end{aligned} \tag{5.7}$$

By using the same method but starting from the equation

$$T2 = Ta + \frac{Zo + Ra}{Rc + Rs + Zo + Ra}(Tc - Ta) \tag{5.8}$$

we can arrive to the discrete time equation for T2

$$\begin{aligned} T2(n) =& \rho_3 T2(n-1) + Ta(n) - \rho_3 Ta(n-1) + g_2(Tc(n) - Ta(n)) \\ & - \rho_2 g_2(Tc(n-1) - T_a(n-1)) \end{aligned} \tag{5.9}$$

$$g_2(z) = \frac{(Ro + Ra)(1 - \rho_3)}{(Ro + Rc + Rs + Ra)(1 - \rho_2)}$$

$$\rho_2 = e^{-T/\tau_2}, \quad \rho_3 = e^{-T/\tau_3}$$

$$\tau_2 = \tau_0 \frac{Ra}{Ro + Ra}, \quad \tau_3 = \tau_0 \frac{Rc + Rs + Ra}{Ro + Rc + Rs + Ra}$$

By implementing Equations 5.7 and 5.9 within python we can visualise the output of the simulation models in Figure 5.2. Noise of the range $+/-0.1$ degrees Celsius was added to the temperatures T1 and T2 in order to simulate the accuracy of the probe of the same magnitude.

By then using the prediction algorithm described above we can obtain a set of predictions on the simulated values. The predictions can be seen on Figure 5.3. The predictions do not reach the desired accuracy of 0.2^oC any time before 400 iterations, which indicates that any implementation of such a system will take long to saturate and perhaps not perform any better than existing competitors in the market. The large spikes are expected to have occurred due to the additive noise of the probes, and from the inaccuracy created if Tc changes too sharply within the window used to estimate Tc.

Even though some of the spikes can be filtered out using post-processing techniques, this implementation does not seem to offer as optimal results as would be desired. Especially if we consider that the simulation model used is far more optimal and less noisy than the real data the hardware will produce.

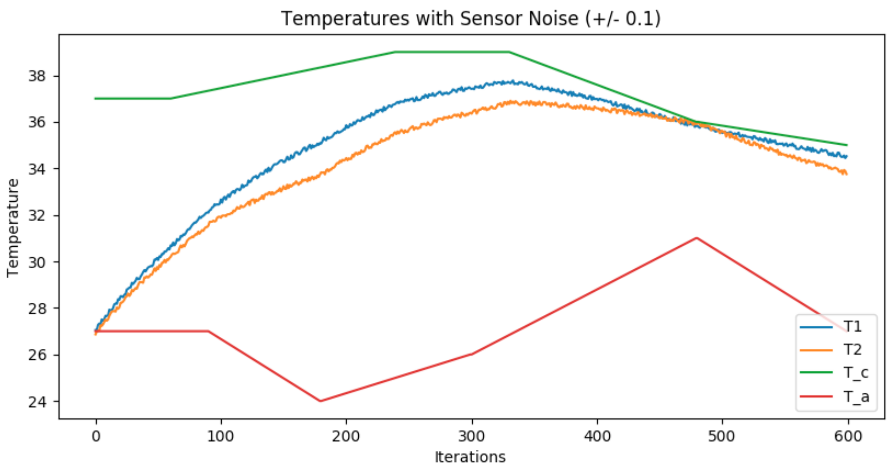


Figure 5.2: Simulating the temperature measurements of the probe

5.2 4-Sensor Mathematical Model

The final implementation that was considered was a 4-Sensor implementation with a setup as shown by the model in Figure 5.4. This setup can be viewed as 2 parallel 2-Sensor model circuits in one. The only difference is that one of the paths does not have another thermal impedance after

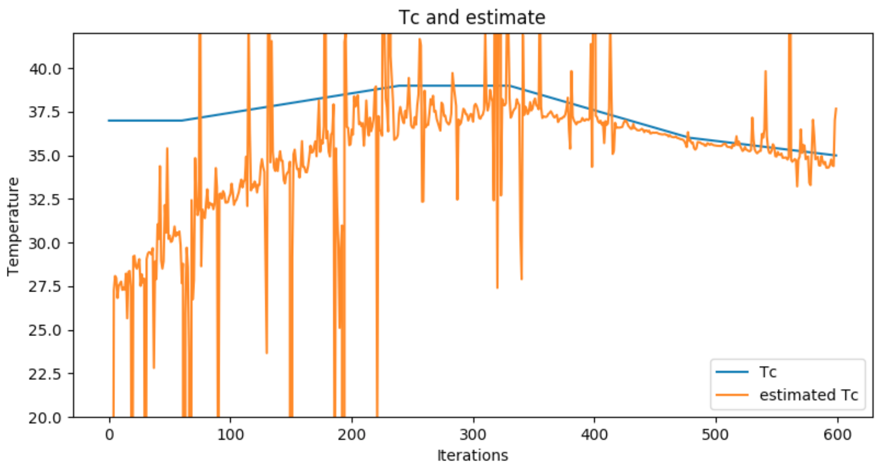


Figure 5.3: Predicting Tc on the simulated environment

the second temperature sensor (Rs). The promise of viewing the circuit in this manner is that we can improve on our previous findings by comparing the two paths with different heat flows and get a more reliable estimation of Tc, while also bypass any assumptions and estimates that were required in the 2-Sensor method.

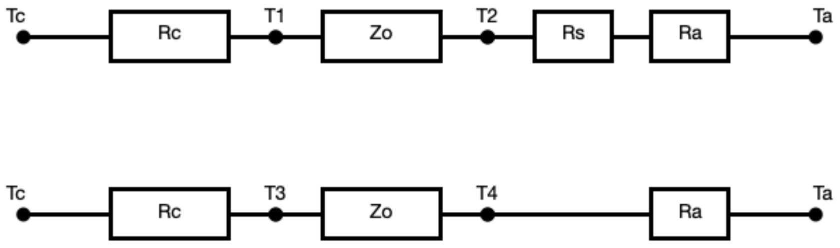


Figure 5.4: Model of the 4-sensor hardware implementation

As we have done previously, we analyse the model to extract the equations

$$\frac{T_c - T_1}{R_c} = \frac{T_1 - T_2}{Z_o} \tag{5.10}$$

and,

$$\frac{T_c - T_3}{R_c} = \frac{T_3 - T_4}{Z_o} \tag{5.11}$$

T1, T2, T3, and T4 are the temperatures to be measured by the temperature sensors. with T1 and T3 both in contact with the skin. Zo models the thermal impedance of the insulator between the two sensors while also accounting for any thermal mass within the probe. $Zo = \frac{Ro}{1+s\tau_o}$. Rc represents the thermal resistance from the core body to the surface of the skin, Rs the resistance of the additional insulator placed on the first channel, and Ra the impedance from the probe to the ambient temperature, as seen previously.

The impedances are again unknown so any equation to calculate Tc should not include them. Any method of determining the impedances and involving them in the calculation will most likely lead to inaccuracies in a commercial product as different probes may have slightly different impedances from the ones that are initially measured. Using Equations 5.10, and 5.11 we can arrive to

$$T_c = T_1 + \frac{(T_1 - T_3)(T_1 - T_4)}{T_3 - T_4 - T_1 + T_2} \tag{5.12}$$

For the simulation side, we need to formulate equations for each of T1 up to T4. These are:

$$T1 = Tc - h_1(Tc - Ta), \quad h_1 = \frac{Rc}{Rc + Zo + Rs + Ra} \tag{5.13}$$

$$T2 = Ta + h_2(Tc - Ta), \quad h_2 = \frac{Ra + Rs}{Rc + Zo + Rs + Ra} \tag{5.14}$$

$$T3 = Tc - h_3(Tc - Ta), \quad h_3 = \frac{Rc}{Rc + Zo + Ra} \tag{5.15}$$

$$T4 = Ta + h_4(Tc - Ta), \quad h_4 = \frac{Ra}{Rc + Zo + Ra} \tag{5.16}$$

By substituting $Z_o = \frac{R_o}{1+s\tau_o}$ to model any thermal mass within the probe, we replace h_1 in equation 5.13 with H_1, where

$$H_1 = \frac{Rc(1 + s\tau_0)}{(Ro + Rc + Rs + Ra)(1 + s\tau_1)} \tag{5.17}$$

$$\tau_1 = \frac{\tau_o(Rc + Ra)}{Rc + Ro + Rs + Ra}$$

and similarly for T2, T3, and T4. As we have done in the previous models, we now apply the z-transform to obtain the equation in the discrete time domain.

$$T1(z) = Tc(z) - G_1(z)(Tc(z) - Ta(z)) \tag{5.18}$$

$$G_1(z) = \frac{g_1(1 - \rho_0 z^{-1})}{1 - \rho_1 z^{-1}}, \quad g_1 = \frac{Rc(1 - \rho_1)}{(Ro + Rc + Rs + Ra)(1 - \rho_0)}$$

$$\rho_0 = e^{-T/\tau_0}, \quad \rho_1 = e^{-T/\tau_1}$$

where T is the period or the time interval between each set of measurements.
By manipulating Equation 5.18 we obtain the final form of the discrete-time equation

$$\begin{aligned} T1(n) =& \rho_1 T1(n-1) + Tc(n) - \rho_1 Tc(n-1) - g_1(Tc(n) - Ta(n)) \\ &+ \rho_0 g_1(Tc(n-1) - Ta(n-1)) \end{aligned} \tag{5.19}$$

By solving in the same way for the other temperatures we obtain

$$\begin{aligned} T2(n) =& \rho_1 T2(n-1) + Ta(n) - \rho_1 Ta(n-1) + g_2(Tc(n) - Ta(n)) \\ &- \rho_0 g_2(Tc(n-1) - Ta(n-1)) \end{aligned} \tag{5.20}$$

$$\begin{aligned} T3(n) =& \rho_2 T1(n-1) + Tc(n) - \rho_2 Tc(n-1) - g_3(Tc(n) - Ta(n)) \\ &+ \rho_0 g_3(Tc(n-1) - Ta(n-1)) \end{aligned} \tag{5.21}$$

$$\begin{aligned} T4(n) =& \rho_2 T1(n-1) + Ta(n) - \rho_2 Tc(n-1) + g_4(Tc(n) - Ta(n)) \\ &- \rho_0 g_4(Tc(n-1) - Ta(n-1)) \end{aligned} \tag{5.22}$$

Equations 5.19-5.22 are implemented within python to simulate the temperature change as would be recorded from the hardware from the point it would be applied on a patient. The starting value for all the temperatures is set to be the same as the ambient temperature.

Ambient temperature and core body temperature are generated either using constant values over the whole simulated period or linearly varying values. The resistance values have to be manually entered as constants since these are unknown in the hardware. These values can be altered to better resemble the values that are directly measured in the hardware.

Additive random noise of the range $+/-0.1^oC$ was added to the simulated values to resemble the inaccuracy of the temperature sensors of the same magnitude. Figure 5.5 shows the results from the implemented simulation.

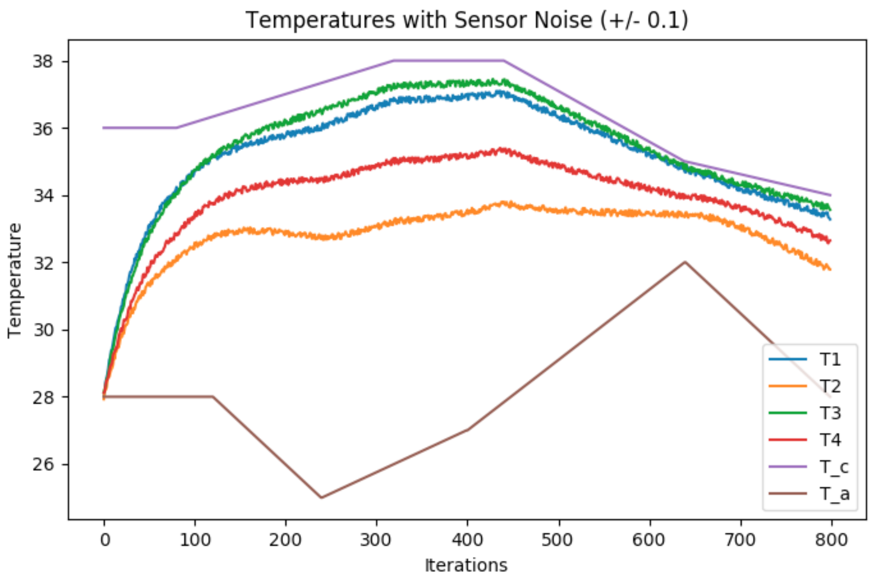


Figure 5.5: Simulated temperatures using the 4-sensor models

By implementing Equation 5.12 on the simulated values we can obtain an estimate for Tc. The result of the predictions can be seen on Figure 5.6.

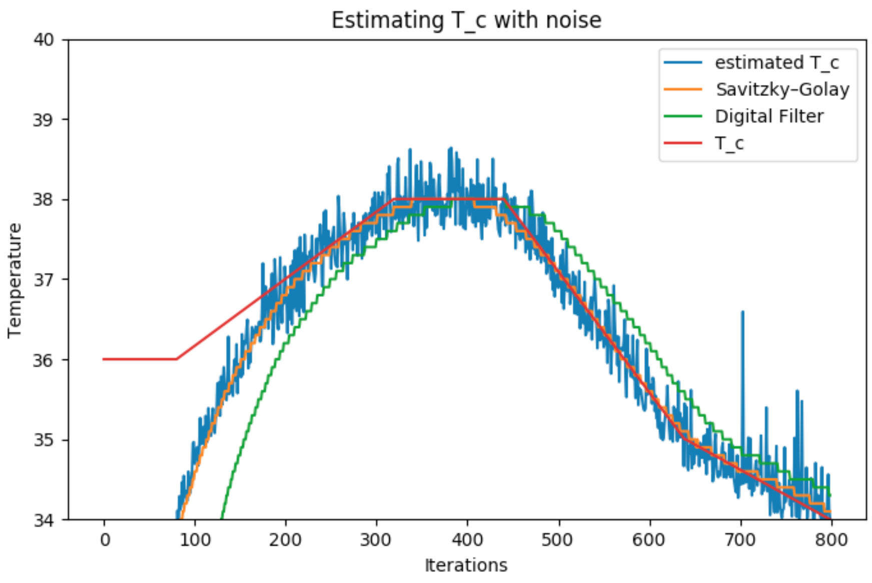


Figure 5.6: Predicted Tc on simulation values and the effect of post processing

The blue line indicates the raw output of the equation which shows that it is heavily influenced by the added noise. In order to obtain more accurate measurements post processing is introduced to smoothen the results.

The most efficient was a Savitzky–Golay filter which uses the method of least squares to approximate sections of the graph with a polynomial of predetermined degree. The size of the window will have different implications on the application. A big window will lead to smoother and more accurate results, however it will lead to a greater delay for the output as more values need to be collected to apply the filter.

As seen from Figure 5.6, the targeted accuracy is achieved near 200 iterations, half what was obtained in the 2-sensor model. The results are much less volatile which will make for a better model.

Even though the results obtained from the simulations were satisfactory, when using real data collected from the hardware, the results were much worse. The desired accuracy was not achieved and the prediction took longer to saturate than the simulation.

A different approach was needed if we were to produce more promising results.

5.3 Machine Learning as a Prediction Model

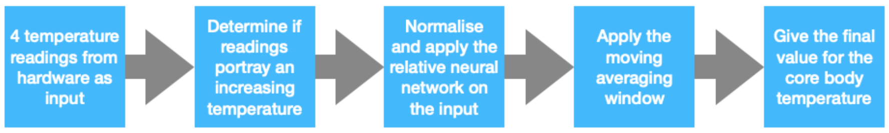

4 temperature readings from hardware as input
Determine if readings portray an increasing temperature
Normalise and apply the relative neural network on the input
Apply the moving averaging window
Give the final value for the core body temperature

Figure 5.7: Overview of how the final predictive model operates

The motivation behind using machine learning was that the previous analysis has shown that the 4-Sensor model leads to sufficient inputs to allow a reliable enough prediction of the core body temperature. The only limitation were inaccuracies produced in the real data that the mathematical formulae did not take into account.

A neural network could be trained on real data collected by the hardware to create a model capable of making accurate predictions on experimental measurements. The limitation of this implementation is risk of over-fitting training data which will lead to good results on the training sets but poor results on new ones. This can be combated by keeping the epochs to a relatively small number and use out of sample data for validation.

Testing proved very promising. Many different architectures were considered which led to the optimal neural network architecture consisting of the input layer of size 4, three hidden layers of size 16, 32, and 128 equivalently, and a single output neuron. The activation functions used were sigmoid, relu, and relu respectively.

A crucial part necessary for training the network is normalisation. After normalisation is introduced, the effect of over-fitting is reduced, and the accuracy of the predictions increased substantially. Normalisation seeks to reduce inconsistencies between different inputs due to their magnitude, and therefore leads to a more responsive model.

Another effect that seemed to hold back the model was that the network was better fitting the saturated (flat) part of the data, worsening the predictions made in the transient portion of the measurements. Multiple methods were tested to provide a solution such as using less saturated data to train the network. The best results were obtained when 2 distinct neural networks were trained. One was trained on the transient portion of the data while the other on the saturated portion. Predictions were made with both the models, one for when the inputs are still changing and one for when the inputs are stable.

Figure 5.8 shows the data obtained from real hardware measurements. There is a noticeable difference from the data in the simulations created in previous sections, and different data sets can be quite different between themselves as well. This is why the static algorithm had a difficult time producing accurate predictions.

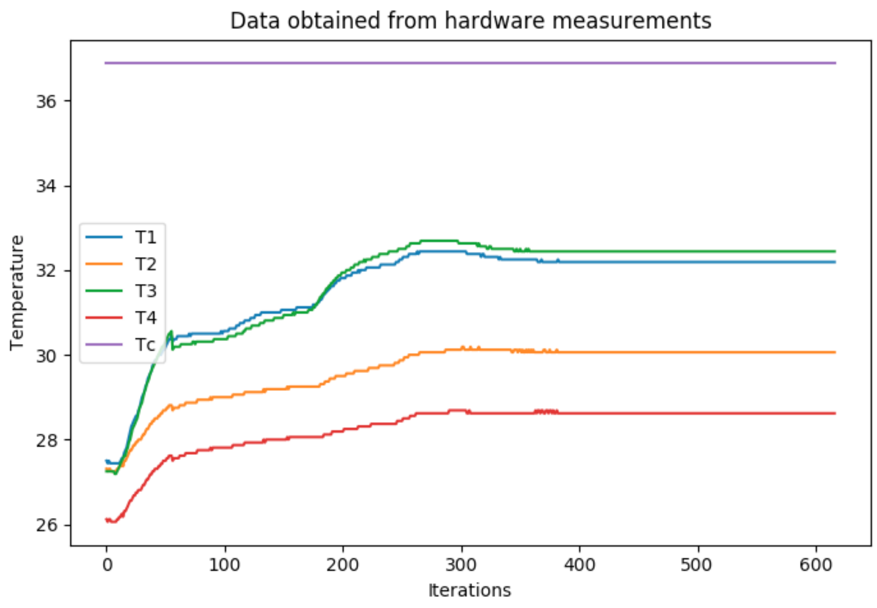


Figure 5.8: Data collected from hardware measurements

Running the neural network on the test data produced the predictions shown on Figure 5.9. The accuracy remains within the 0.2^oC threshold, and the predictions saturate instantaneously. The post processing introduced in this instance is a moving average of the last 3 values. The post-processing used was simplified compare with the ones used in the 4-sensor method as already the neural network is quite computationally intense and we did not want to introduce even more latency in the measurements. The results are also rounded to the nearest tenth since any higher precision will lead to unnecessary detail.

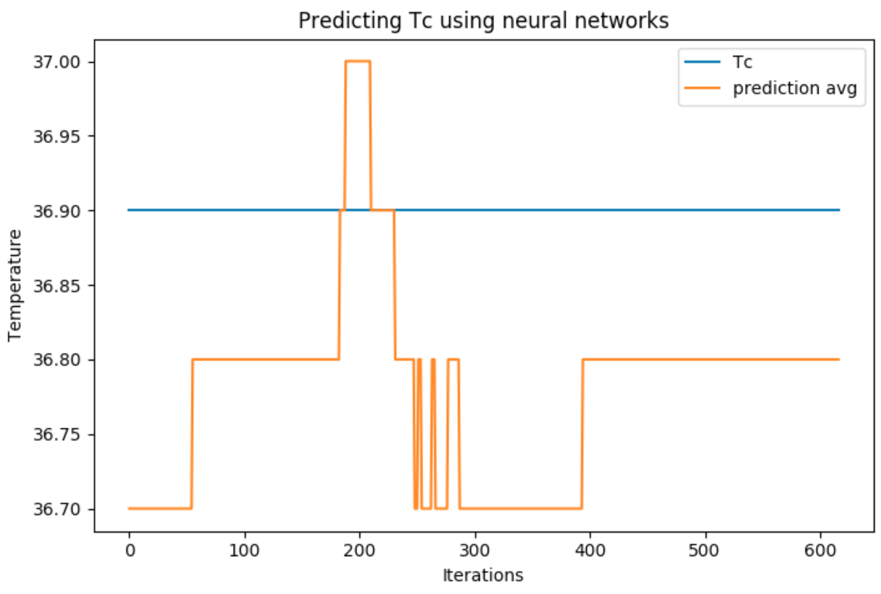


Figure 5.9: Predicted Tc from real data using a neural network and post processing

Further validation was made on data measured from a person not used in the training set to further support the accuracy of the produced network. Even though there is evidence to suggest that the implemented neural network produces reliable results, there are a few more steps necessary before final commercial release in terms of the prediction algorithm that were not possible to be made due to current circumstances. More more measurements should be made on a wider range of test subjects in a wider range of scenarios to train and validate on, to ensure as high reliability as possible.

Chapter 6

Base Station Application

The following sections will explain the design and implementations of the various sections of the base-station application. The sections flow in a chronological fashion in terms of the timing of their respective implementation. The key changes are explained in the text and further explanations are included in the appendix for your convenience. Code snippets which are of particular importance as it pertains to the overall functions of some subsystems are included for your viewing. The reader is encouraged to view the entire Github repo to run the code as well as to gain a deeper understanding of the exact implementation.

NOTE:
Source code can be view using the github link in *Design Output Documentation* under folder named */App Source Code*.
Run the project by reading through the following tutorial : https://flutter.dev/docs/get-started/codelab.

6.1 Preliminary Designs

From the design specification generated (see input documentation) preliminary designs for the application user interface (UI) could be put together. This, it was hoped, would give an insight into the technologies required for the actual implementation and therefore enable a more informed decision when it came to selecting a specific framework for developing the application.

To start with three designs were generated. Each design would contain three renders; a login page, an activity page (where live readings could be seen), and a history page. These encapsulated the core features that the application would have to implement.

After a meeting with the entire group, discussing the ergonomics, visuals and implementation details with the rest of the team, it was decided that *Preliminary Design 1* would be pursued further. The main reason for this choice was the clean aesthetics and colour theme which we believe would fit in well with contemporary UI designs.

6.2 v0.1: Application Framework

After settling on a design for the application, we began focusing on the selection of tools that would enable us to create the final application. Many considerations have to be taken into account when making these decisions. These are explored in the following section.

6.2.1 What is a Framework?

A framework, or software framework, is a platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform. For example, a framework may include predefined classes and functions that can be used to process input, manage hardware devices, and interact with system software. This streamlines the development process since programmers don't need to reinvent the wheel each time they develop a new application. [9].

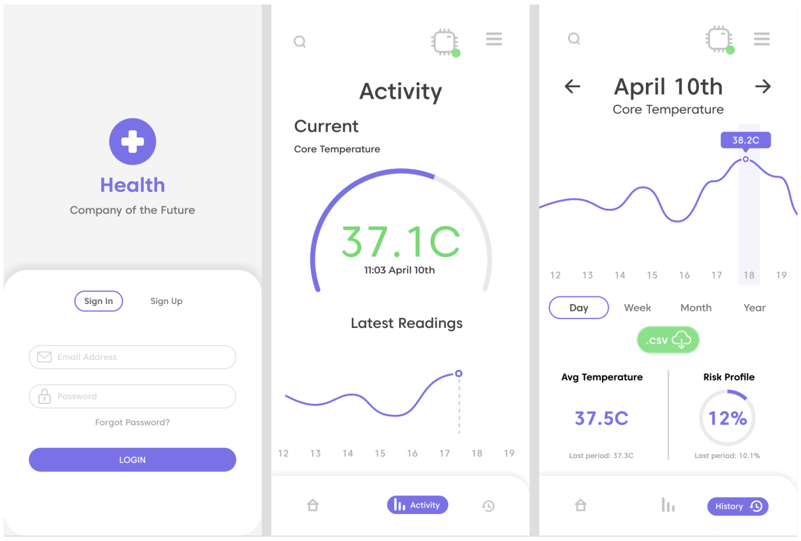


Figure 6.1: Preliminary Design 1

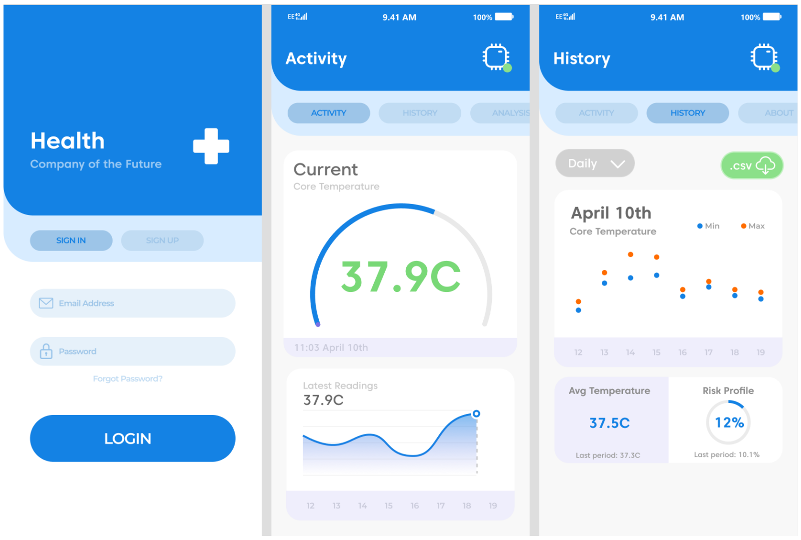


Figure 6.2: Preliminary Design 2

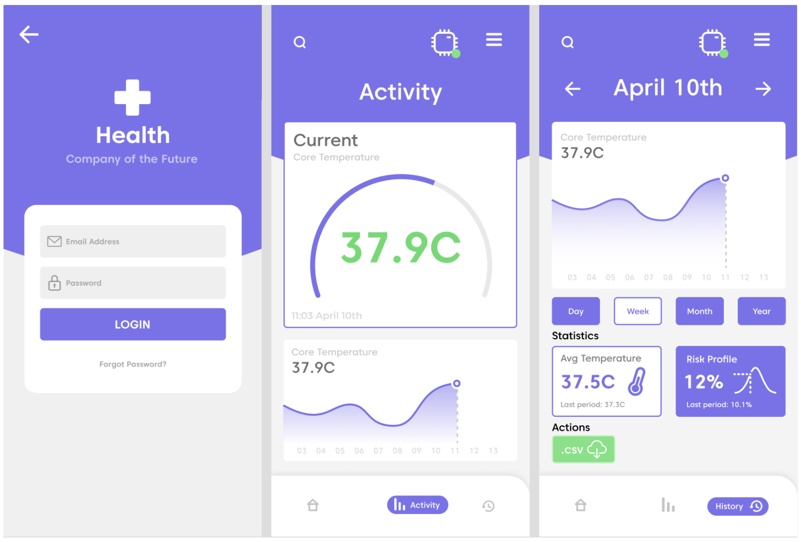


Figure 6.3: Preliminary Design 3

Figure 6.4: Flutter Framework Icon: flutter is a UI framework for creating mobile multi-platform mobile applications.

Using a framework that is optimised for mobile development will be useful in creating complex UI elements and save time debugging and testing.

6.2.2 Native Applications

Since the device we are designing will communicate over Bluetooth to a mobile device the operating systems we must design for are Android and IOS.

A Native code refers to an app written using the languages and IDEs directly supported and recommended by the operating system maintainer. In the case of Android this is an app written in Java/Kotlin created in AndroidStudio, and for IOS this is an app written in ObjectiveC created in Swift.

What this means is that creating a native application which works on all mobile devices will require the creation of two applications which must be simultaneously updated, tested, and optimised for.

6.2.3 Why Flutter?

Writing a native application would severely hamper our ability to add all the features we would like in the time available to us. An alternative method of creating an application is done by leveraging something known as a Cross-Platform Native Framework. This Framework acts as an intermediary between your designs and the device and can compile your designs to many different operating systems. This has the main advantage of enabling your app to be contained within a single code base with the ability to interact with native code if so desired. It does come at a cost. The main one being that there now exists a layer of abstraction between your code and the device which its running on. This comes with a performance penalty as well requiring extra boilerplate code to interact with device sensors when compare to native code.

This is where Flutter comes in. Flutter is an open-source UI software development kit created by Google.

Flutter consists of two important parts [10]:

- An SDK (Software Development Kit): a collection of tool that are going to help you develop your application. This includes tools to compile your code into native machine code (code for iOS and Android).
- A Framework (UI library based on widgets): A collection of reusable UI elements that you can personalize for your own needs.

On top of all this, Flutter is said to integrate well with other Google services such as Firebase [11] which will act as our backend, has a wide array of available plugins, compiles quickly (which is good for testing) and is written in Dart (an easy to pick up Object-Orientated-Programming Language). Its is for these reasons we decided to use Flutter as the tool to create the mobile application.

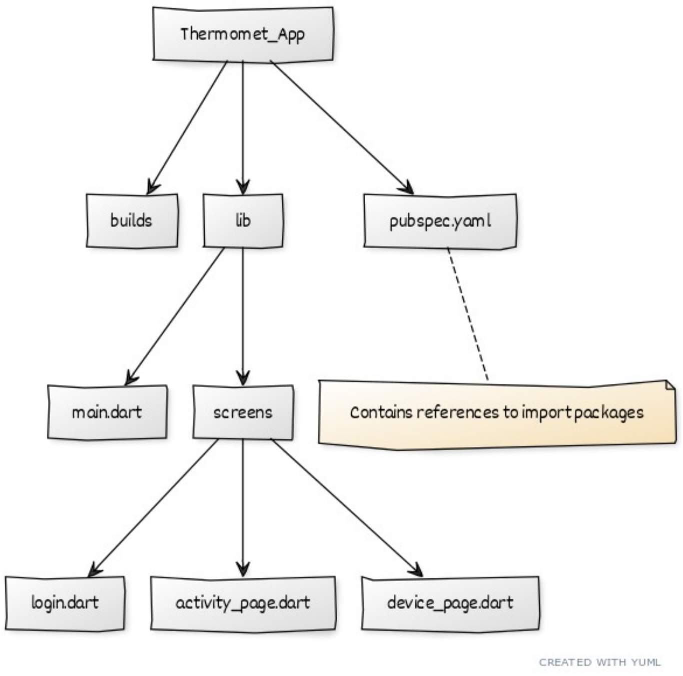


Figure 6.5: Flutter File Structure v0.1

6.2.4 File Structure Hierarchy?

The general file structure of the application is show in Figure 6.5. Some folders have been purposefully omitted because of their lack of relevance when it comes to developing the application.

Generally speaking, all the classes files are placed in the lib folder. The main.dart file contains the entry point of the application UI. The adjacent screens folder contains the UI layouts for the pages. An import files is the pubspec.yaml file located in the top right of the figure. It contains all the references to any 3rd party installed packages and assets, along with any version numbers and other dependencies. Compilation errors may occur if the version numbers of the packages are incompatible.

6.3 v0.2: Bluetooth Communication

Communicating with the sensor was the first priority of the application. Once data could be reliably transferred between the base-station and the physical sensor; only then would further development be possible.

6.3.1 Why Bluetooth?

Physically attaching a cord between the user's mobile device and the sensor could better optimise power consumption and weight by only using the battery of the mobile device, but would be impractical for long term continuous reading, not to mention the wear and tear associated with physical cords.

Its clear that a wireless protocol would be a better solution. The main contenders for the choice of communication protocol are WiFi and Bluetooth/BLE.

Here are some key comparison information we considered [12]:

- Bluetooth is has high penetration in various end-user devices including healthcare, sport and fitness.
- Bluetooth is optimized to transport very large amounts of very small data packets.
- WiFi has a data rate up to an order of magnitude greater than Bluetooth.

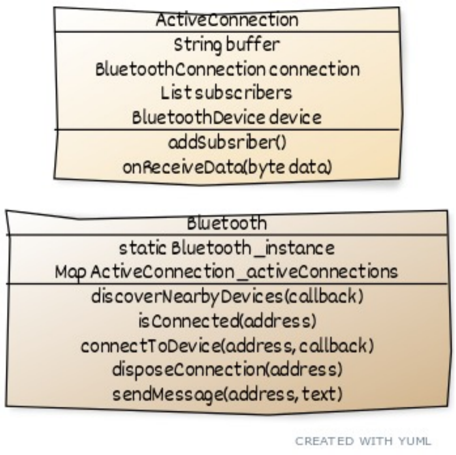

ActiveConnection
String buffer
BluetoothConnection connection
List subscribers
BluetoothDevice device
addSubsriber()
onReceiveData(byte data)
Bluetooth
static Bluetooth _instance
Map ActiveConnection _activeConnections
discoverNearbyDevices(callback)
isConnected(address)
connectToDevice(address, callback)
disposeConnection(address)
sendMessage(address, text)
CREATED WITH YUML

Figure 6.6: *ActiveConnection* and *Bluetooth* Class Overview

- WiFi cannot independently establish a connection with a device.

We decided to go with Bluetooth because its lightweight protocol can communicate directly to a device without having to go through a large network which would increase latency. Moreover, low power consumption and reliability is more important than the raw data rate because the data packets that will be transferred are relatively small but transmitted frequently.

6.3.2 Implementation

Interacting with any sensor on a mobile devices requires the use of native code to access the sensor libraries. Fortunately, there are many packages provided by Flutter which implement the necessary interface to communicate with most sensors. Using the Flutter package manager we imported the package named ***flutterBluetoothSerial 0.2.2***. This package provides basic functions that will allow the application to access the on board Bluetooth sensor.

Using the aforementioned package, we proceeded to create a custom class called *Bluetooth*. This class will handle all the connection maintenance and troubleshooting in a single easy to access place. This script is the channel through which the entire application talks to the device, hence it uses of a singleton pattern[13]. The script can scan for devices and pair to any discovered device. It can also send and receive data from the device in byte format. Another class called *ActiveConnection* was created to hold the *BluetoothConnection* object along with a buffer and a few other useful pieces of information. The main purpose of the ActiveConnetion class is to format the received data into the correct expected format and handle errors associated with missing byte and or corrupted data. Once a valid message is received, the data is formatted and forwarded to all the subscribers to the connection. Figure **??** shows the main properties and methods of ActiveConnection and Bluetooth class. Note that the Bluetooth class has a static instance and contains a list of ActiveConnections.
Figure 6.7 shows the steps to establish a connection with a Bluetooth enabled device.

- The method 'onSearchDevice()' calls discoverNearbyDevices()
- Search occurs for 4 seconds and fills the map 'discoveryResult' with all the discovered devices.
- Once a devices is selected 'connectDevice()' calls 'connectToDevice()'
- Once the selected devices has established a connection, a conformation message is transmitted.

The code below is the method inside the *Bluetooth* class which is responsible for establishing a connection with a device. The key point to note is that every new connection is added to the list of *activeConnections*.

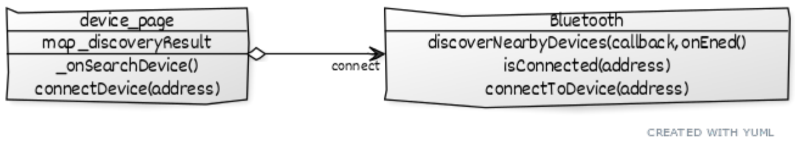


Figure 6.7: Establishing a Bluetooth Connection

```
/// Established a connection with a device so messages can be transferred.
  ///
  /// Note: a live connection can only be established with devices which have been
    discovered and bonded with.
  Future<void> connectToDevice(String address, Function _onDatReceived) async{
    // close connection if one already exists
    if(_activeConnections.containsKey(address)){
      await _activeConnections[address].connection.finish();
    }

    try {
      // connect to device
      BluetoothConnection _connection = await BluetoothConnection.toAddress(address
    );
      // create new active connection
      _activeConnections[address] = ActiveConnection(connection: _connection,
    terminatePhrase: terminatePhrase, device: _results[address].device);
      // add callback as subscriber to ActiveConnection
      _activeConnections[address].addSubsriber(_onDatReceived);
      // set activeConnection.onReceiveData as 'listen' callback for
    BluetoothConnection
      _activeConnections[address].connection.input.listen(_activeConnections[
    address].onReceivedData).onDone(() {print('Disconnected by remote request');});
    }
    catch (error) {
      print(error);
    }
  }
```

Now that a device has been paired, a system for receiving and processing data was required. This system is explained below and summarised in Figure 6.8.

- The *BluetoothConnection* class is defined in the imported package. It receives data from the connection port and forwards it to its callback method - which in our case is *onReceiveData(data)* method in the *ActiveConnection)* class.
- The 'ActiveConnection' class receives data from the 'BluetoothConnection' and stores the bytes in a buffer.
- Once the 'endToken' of a message is received the relevant data is packaged and forwarded to all the subscribers to the connection. Note, the *Bluetooth* class contains a list of all the ActiveConnections.
- Currently there is only a single subscriber to the connection. The 'ReceiveDataHandler' class will implement the algorithm to estimate the CBT. But at the moment it prints the data is receives.

After implementing the Bluetooth communication the project file structure looked like Figure 6.10.

6.4 v0.3: Firebase Authentication and Database

Part of the design specification required that the final product provide the user the ability to view and download their historic data. This means the measurements that are collected need to be stored somewhere. A trivial solution would be to store all the data locally on the base station.

Our team decided to save the data to a database that can be accessed by the user through a set of authentication credentials. We chose this path for the following reasons:

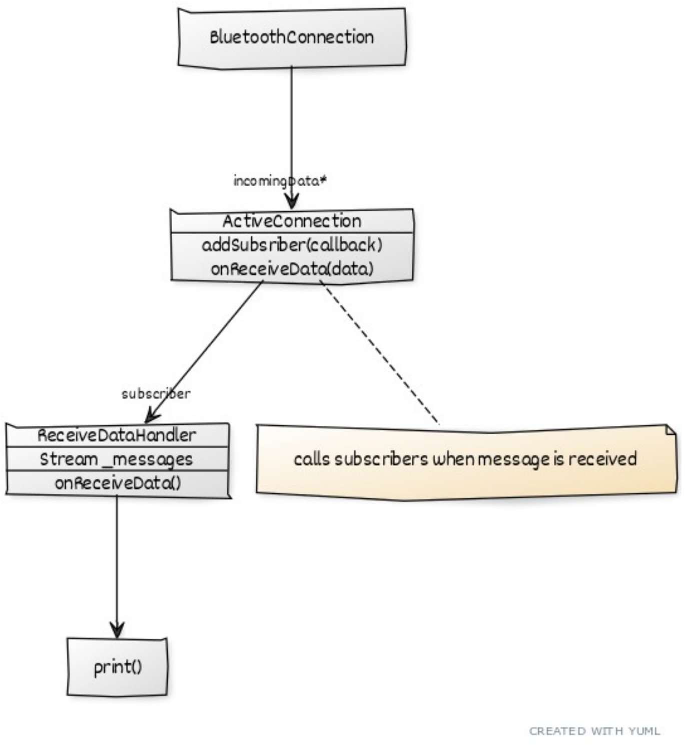
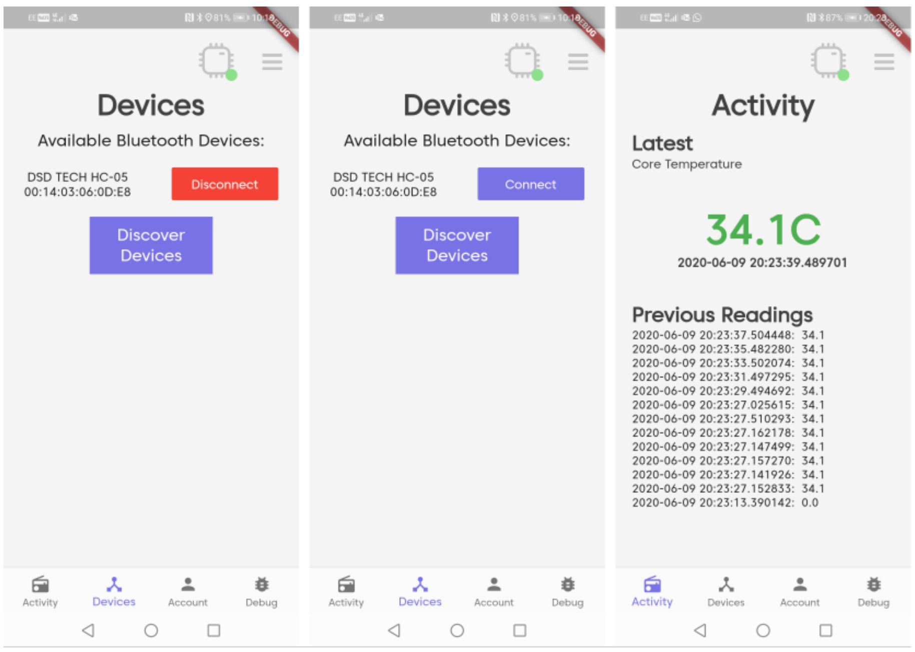


Figure 6.8: Receiving Data



```
2020-06-09 20:23:39.489701
```

Previous Readings

```
2020-06-09 20:23:37.504448:  34.1
2020-06-09 20:23:35.482280:  34.1
2020-06-09 20:23:33.502074:  34.1
2020-06-09 20:23:31.497295:  34.1
2020-06-09 20:23:29.494692:  34.1
2020-06-09 20:23:27.025615:  34.1
2020-06-09 20:23:27.510293:  34.1
2020-06-09 20:23:27.162178:  34.1
2020-06-09 20:23:27.147499:  34.1
2020-06-09 20:23:27.157270:  34.1
2020-06-09 20:23:27.141926:  34.1
2020-06-09 20:23:27.152833:  34.1
2020-06-09 20:23:13.390142:  0.0
```

Figure 6.9: Bluetooth Connection UI

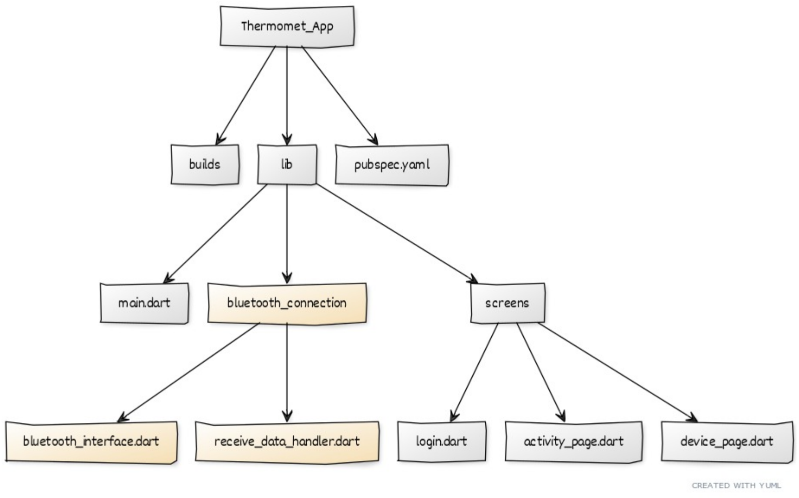


Figure 6.10: Flutter File Structure v0.2

- Reduces the number of read/write cycles required on the local devices, which can be CPU intensive.
- Reduces the memory footprint of the application.
- Allows collection of user data to perform more complex analysis which may not be possible on low powered mobile. devices.

6.4.1 Database as a Service

Increase of creating and managing a database that we would have to create from scratch we opted to use *Firebase*, a *Database as a Service*[14] provided by Google. The reason for using Firebase stems from the need for data security and scalability. Firebase fulfils both of these by providing database as a service. This means Firebase will handle all the back-end complexity of saving data, authenticating users, and scaling as the user-count increases without us having to change anything. This gives us the ability to focus on creating a better user experience and optimising other aspects of the device.

6.4.2 Firebase and Flutter

As mentioned previously, one of the prime motivators for using Flutter as the design framework was its ability to easily integrate with Firebase.

Integrating Flutter with Firebase is done through the following steps [11]:

- Import the packages ***firebaseAuth*** and ***cloudFirestore*** into your *pubspec.yaml* file under *dependencies:*.
- Generate a *private key* (json file with authentication data) from your Firebase dashboard and add to Flutter project.
- A static instance of the class *FirebaseAuth* and *Firestore* is now available to use in the project.

With Firestore (a NoSQL database), and FirebaseAuth (a user authentication service) installed, we were able to write scripts in our Flutter application which could login/register a user, and pull/push data to our database. The main methods which implement this functionality can be found below. The key thing to note is that we are using static instances of the FirebaseAuth and Firestore classes. This means that only a single user can be logged in at a time, and that any update to the user credentials will update all instances.

```
/// Contains instance of ‘FirebaseAuth‘.
///
/// Subscribe the ‘user‘ stream to get updates on authentication state.
/// Also contains all Sign In/Out/Up.
class AuthenticateService{

  final FirebaseAuth _auth = FirebaseAuth.instance;

  // sign in with email and password, returns user object with credentials
  Future signInWithEmailAndPassword(String email, String password) async {
    try {
      AuthResult result = await _auth.signInWithEmailAndPassword(email: email,
    password: password );
      FirebaseUser user = result.user;
      return user;
    } catch(error) {
      print(error.toString());
      return null;
    }
  }

}
```

```
// Collection reference
CollectionReference userDataReference = Firestore.instance.collection('
    thermomet_user_data');

/// Override the user data
Future<void> updateUserData(Map<String, dynamic> data) async{
  return await userDataReference.document(uid).setData(data);
}

/// Pull the user data from ‘Firestore‘.
Future<void> _pullUserData() async{
  var doc = await userDataReference.document(uid).get();
  _cachedDoc = doc.data;
}
```

Now that we can login and logout its important that the application UI state responds to changes in the user authentication state. To accomplish this the root widget of the application was edited to update when the *FirebaseAuth.instance.user* object changed.

This is implemented in the following code:

```
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    // listen to the user object
    return StreamProvider<FirebaseUser>.value(
      value: AuthenticateService.instance.user,
      initialData: null,
      child: MaterialApp(
        title: 'Flutter Demo',
        theme: ThemeData(
          //fontFamily: 'Caros',
          fontFamily: 'Kollektif',
        ),
        home: AuthenticationParent(),
      ),
    );
  }
}
```

Essentially what is happening is that the application is re-built whenever the state of the *user* changes. If there is no user, then the login page is shown. If a valid user is logged in then the activity page is shown and all the relevant user data is pulled from the server.

After adding these updates the overall file structure of the project is shown in Figure 6.11.

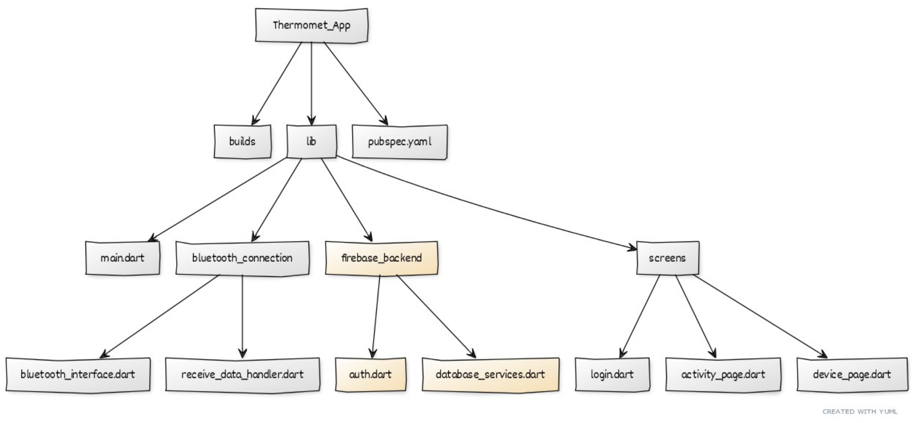


Figure 6.11: Flutter File Structure v0.3

6.5 v0.4: UI Overhaul

Now that all the major system of the application are functioning, namely the Bluetooth communication and database services, it is time to connect the sensor and implement the algorithm to begin generating measurements for the Core Body Temperature (CBT).

6.5.1 Connection to Algorithm API

As the algorithm sections explained, the decision to host the CBT predictor on a web API moves some of the CPU and memory intensive tasks off of the application. Now we updated the *ReceiveDataHandler* which up until now merely printed the received data.

To begin with, a clearly defined format was required for exachanging data between the sensor, application and API. We opted for a Json format because of its wide use in industry, its harmony with the firestore database services, and the existence of widely available libraries to encode and decode. A template of the format is shown below. It consists of a list of readings for each temperature sensor, along with some information relating to the type of algorithm to use.

```
{
    't1' : {
        'item1' : 31.0,
        ...
    },
    't2' : {
        'item1' : 28.37,
        ...
    },
    't3' : {
        'item1' : 30.94,
        ...
    },
    't4' : {
        'item1' : 27.44,
        ...
    },
    'filter': noiseFilter,
    'algorithm': algorithmType,
}
```

To compartmentalise the code we created an class called *algorithmMain* which would handle the API communication. An instance of this class is then stored in *ReceiveDataHandler* which now call the method *computeCoreBodyTemperature()* when it receives raw data readings from the sensor device.

The *algorithmMain* class ensure that the raw data is in the correct format and then communicates with the API over http. Once a result is received (or an error) the result is added to the Stream for all listeners to receive. The updated *ReceiveDataHandler* looks like the following.

```
class ReceiveDataHandler{

  // Instance of algorithm
  Algorithm _alg = Algorithm();

  /// core-body temperature predictions based on received data.
  StreamController<ThermometerData> _tempReadings = StreamController<
    ThermometerData>.broadcast();

  /// This method is called when data is received from a bluetooth device
  void onReceiveData(String name, String address, String message) async{
    // Convert data to a 'Message' data type
    Message newMessage = Message(deviceName: name, address: address, text: message)
    ;

    // Generate prediction
    ThermometerData prediction = await _alg.computeCoreBodyTemperature(message);

    // If valid result is received, inform all subscribers
    if(prediction != null){
      _tempReadings.add(prediction);
    }
  }
}
```

6.5.2 Event Manager

Passing data from parent to children nodes and vice-versa is crucial in all applications, and having a systematised way of achieving this is important in avoiding code clutter and increasing usability. There exist many paradigms to achieve this, but we opted to go with a simple event manager which would hold a list of subscribed methods to a particular event and allow the entire program to access a static *EventManager* class to subscribe to and trigger any number of events [15].

6.5.3 Adding Graphs

Now that all the systems work we moved to update the UI to display the user data. Using the flutter package manager we imported ***charts_flutter 0.9.0***. This package provides an array of graphs and chart templates that can easily be integrated into an application. We refer your to *https://pub.dev/packages/charts_flutter* for more information and examples.

When a new CBT measurement is received, all subscribers to the *ReceiveDataHandler* are updated. The activity page contains a subscriber which holds a list of previously received readings and displays them on a live graph as well as outputting the latest readings - see Figure 6.12.

With these updates integrated the final file structure looks like Figure 6.13.

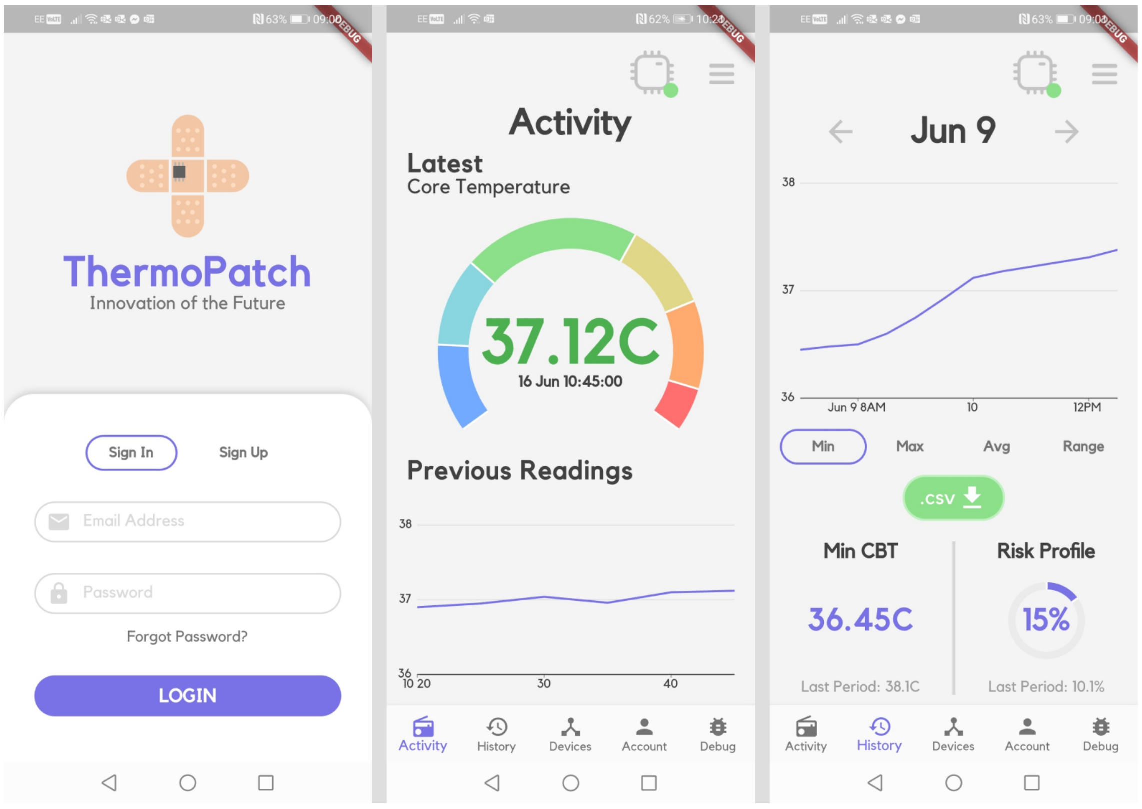


Figure 6.12: Final Application User Interface

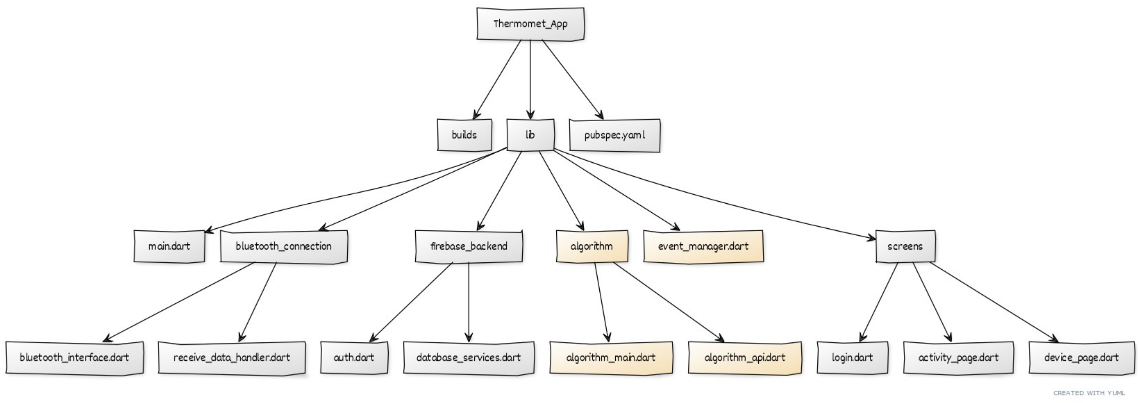


Figure 6.13: Flutter File Structure v0.4

Chapter 7

Recommended Updates

In this section we suggest updates and further improvements which we believe would built on the systems we have created and form a pathway to launching a viable product - Figure 7.1

7.1 Custom PCB and Use of 3D printer

We plan to integrate all internal circuit and microprocessor on one PCB board in the future, which makes the device smaller and possibly wearable. Additionally, more decent industrial design will be achieved with the assistance of 3D modelling and 3D printer.

7.2 Sensor Arrangement

Furthermore, 2 and 3-sensor arrangement will be reconsidered. Additional research including finding materials with less thermo-expansion and using faster-updated sensor will be attempted in the future as it's clearly more power-saving, easily-manufactured, and cost-saving with less sensors in device.

7.3 Train Model with More Data

In data science the maxim of 'more data' is always true. With more data to train the algorithm the prediction error should improve over time. Along with a greater amount of data, a more varied collection would also be desirable. This will allow the network to better account for outliers.

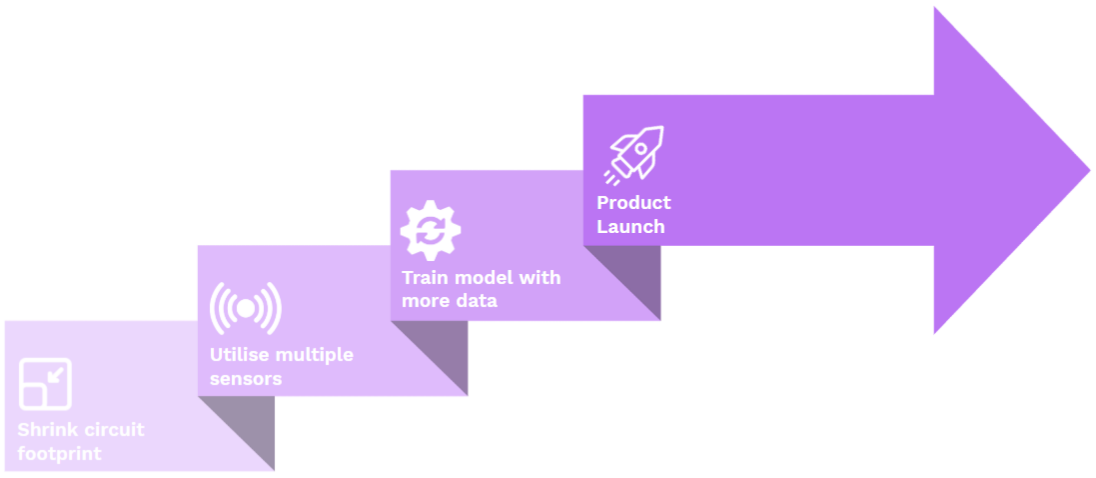


Figure 7.1: Flutter File Structure

Moving beyond a single model, a more optimal solution would involve a network which is optimised for individual users or groups of users. This way differences between individuals and groups can better be identified and provide further diagnostic information for future use.

Bibliography

[1] "Core body temperature v peripheral body temperature," https://www.safework.nsw.gov.au/resource-library/heat-and-environment/working-in-extreme-heat-the-facts/accordians/core-body-temperature#:~:text=Core%20body%20temperature%20is%20the,%2C%20liver%2C%20brain%20and%20blood.

[2] "Fever temperatures: Accuracy and comparison," https://www.mottchildren.org/health-library/tw9223#:~:text=But%20the%20temperature%20readings%20vary,if%20a%20fever%20is%20present.&text=An%20ear%20(tympanic)%20temperature%20is,higher%20than%20an%20oral%20temperature.

[3] R. Rossi, "Clothing for protection against heat and flames," *Protective Clothing*, 2014.

[4] "Core body temperature maintenance," https://www.sciencephoto.com/media/316831/view/core-body-temperature-maintenance.

[5] A. G. Constantinides, "Thermomet: Fewer sensors simplified sensor arrangement," 2020.

[6] ——, "Thermomet: Fewer sensors1 simplified sensor arrangement," 2020.

[7] K.-I. Kitamura, X. Zhu, W. Chen, and T. Nemoto, "Development of a new method for the noninvasive measurement of deep body temperature without a heater," 2010.

[8] A. G. Constantinides, "Core body temperature through heat-flux modelling: Two alternative solutions," 2020.

[9] "Framework definition," https://techterms.com/definition/framework, March 7, 2013.

[10] G. Thomas, "What is flutter and why you should learn it in 2020," https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/, December 12, 2019.

[11] "Add firebase to your flutter app," https://firebase.google.com/docs/flutter/setup, June 19, 2020.

[12] P. Panicker, "Bluetooth vs wifi for the internet of things (iot)," https://www.quicsolv.com/blog/internet-of-things/bluetooth-vs-wifi-comparison-iot-solutions/, April 9, 2020.

[13] "Singleton pattern," https://en.wikipedia.org/wiki/Singleton_pattern, June 20, 2020.

[14] "Cloud firestore," https://firebase.google.com/docs/firestore, June 22, 2020.

[15] "Unity eventmanager with delegate instead of unityevent," https://stackoverflow.com/questions/42034245/unity-eventmanager-with-delegate-instead-of-unityevent, February 22, 2017.