Challenge 2: Global Health care

CS-EEE Specialist Team Final Report

Authors: Sebastian Khorasan-Shiri, Xiao Sun, Abby Tjandra, Jinjiang Nie, Vasu Jha, Aryan Jain, Ulk Gerguri, Olivier Gaillard, Lukas Iskenderoglu, Hsien Jun Ong, Chubing Long

Table of Contents

1 Introduction

Section Authors: A. Jain

Tuberculosis (TB) is a bacterial infection that typically affects the lungs but can also affect other parts of the body. It is spread through the air when a person with TB of the lungs or throat coughs, sneezes, or talks. If left untreated, TB can be deadly. The TB vaccine, Bacillus Calmette–Guérin (BCG) vaccine, is part of the routine childhood vaccination schedule and is given to infants at birth in Uganda. As of 2020, around 196 out of 100,000 people have TB in Uganda. [1] The World Health Organisation recommends the use of the BCG vaccine as a protective measure against TB in countries with a high prevalence of the disease. Therefore, building a TB vaccine plant in Uganda will greatly benefit the people. However, producing vaccines in Uganda could be challenging due to limited healthcare infrastructure, public funding, human resources, technological expertise, and energy resources.

This project of producing the BCG vaccine was split into three parts, managed by specialised groups. CS-EEE had to create a control system for the bioreactor whilst the other two teams had to design and implement the reactor and select the location and provide resources such as gas and power. The bioreactor had three subsystems: heating, pH and stirring. We had to design and test these components and remotely monitor and control them via a user interface. A bioreactor is a device or system that is used to facilitate the growth of cells or microorganisms, or to support the biological conversion of raw materials into useful products.

The subsystems were operated with Arduino and controlled via the dashboard by connecting the Arduino to the cloud with the ESP32. The set points were configured by the user on the dashboard and must be within certain ranges. The specifications were that the heating subsystem would maintain the temperature of the bioreactor between 25-35°C within ±0.5°C of the set point, the pH subsystem would maintain the pH level of the chemicals in the bioreactor between 3 – 7 and the stirring subsystem would control the stirring rate of the bioreactor between 500-1500 RPM within ±20 RPM of the set point.

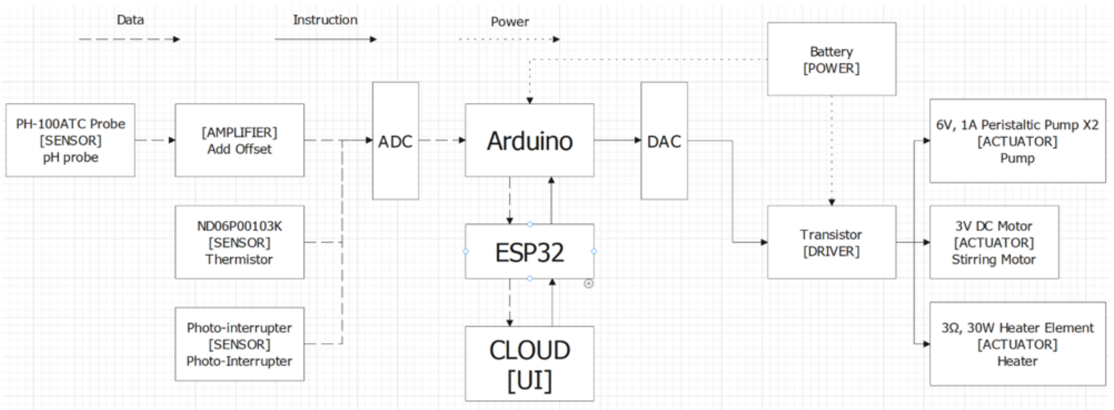

Data
Instruction
Power
Battery [POWER]
PH-100ATC Probe [SENSOR] pH probe
[AMPLIFIER] Add Offset
ADC
Arduino
DAC
6V, 1A Peristaltic Pump X2 [ACTUATOR] Pump
ND06P00103K [SENSOR] Thermistor
ESP32
Transistor [DRIVER]
3V DC Motor [ACTUATOR] Stirring Motor
Photo-interrupter [SENSOR] Photo-Interrupter
CLOUD [UI]
3Ω, 30W Heater Element [ACTUATOR] Heater

Figure 1 Complete system

Figure 1 shows the system parameters which were remotely monitored and controlled by the user and provide the UI with real time data that gets logged for quality control.

In a bioreactor, gene-modified yeast was usually used to produce vaccine. The growth rate of yeast was determined by pH, temperature, and other factors. We controlled these factors to the most ideal condition for the yeast to grow fast. The stirring subsystem would ensure that the glucose [or acid or alkaline added by the pH subsystem] was mixed evenly in the liquid. With the GUI, anyone could easily manipulate the settings without writing any code. Engineers could offer remote updates to the code or alter them via the internet since the entire system was connected to Wi-Fi. Also, using simple and cheap components in the subsystem (like a microcontroller) plus the nature of the control logic behind the codes enabled maintenance at little cost, low power consumption and smaller environmental impact. All of this saved money for the government, which meant that the TB vaccine was more likely to be taken by more people (through both private and public health-centers, as there are significant non-private health institutions in Uganda).

2 Subsystems

2.1 Heating Subsystem

Section Authors: T. A. Tjandra and J. Nie

2.1.1 Description of Heating Subsystem

The purpose of this subsystem was to measure the current temperature of a liquid, if the temperature was below the setpoint set by the user, the heater would heat the liquid until it reached the desired temperature. In the production of TB vaccines in Uganda, the temperature at which the vaccine was manufactured and stored was very important. A temperature too cold or too hot could spoil the vaccine, making it ineffective or even harmful to humans.

2.1.2 Specification of Heating Subsystem

The heating subsystem consisted of a 30W, 3Ω heater and a 10kΩ thermistor. The heating subsystem should maintain the temperature within a ±0.5°C from the temperature setpoint. The system would be powered using the 12V DC power supply. An IRLB8748 MOSFET was used which acted as a switch. The voltage and current for the heater were too high for a BJT transistor, therefore using MOSFET was more stable. Since the thermistor had a resistance of 10kΩ, a 10kΩ resistor needed to be connected to the thermistor. Meanwhile the heater was connected to a 10kΩ resistor and a 1N4148 diode which would balance the difference in charge in the MOSFET. The user must be able to decide their own desired temperature setpoint between 25°C and 35°C.

2.1.3 Design of Heating Subsystem

Thermistor Calibration

The calibration of the thermistor was done by measuring the resistance of a range of temperatures. 6 temperatures were used ranging from -1°C to 57.3°C. The Steinhart-Hart equation was used to calculate the temperature from the resistance: $T = \frac{1}{A+B\ln R+C\,(\ln R)^3}$. [2] The MATLAB codes in the appendix was used to calculate the constants A, B and C. From the temperature and resistance that was measured, three different values of A, B and C are calculated, and the average was taken. [4.1.1]

Heater Calibration

The calibration of the heater element was done by changing the PWM number written from the analog pin. The statements "on" or "off" were controlled by the comparison between the desired temperature from cloud and the actual temperature measured by the thermistor. If the actual temperature was over the desired temperature-0.5°C, the analog pin would write 0 turning off the heater. This was done as the heater would keep heating for a while after it was switched off. In addition, 230 was used instead of 255, because 255 provides too much power for the heater, and 230 makes the power lower. This ensured the temperature to not go too high. Figure 2 shows the pseudocode for the heating subsystem which was then converted into an

```
DECLARE Vo : INTEGER
DECLARE logR2 : REAL
DECLARE R2: REAL
DECLARE T : REAL
DECLARE temperatureSetPoint : REAL
DECLARE Heater : INTEGER
CONSTANT R1 ← 10000
CONSTANT A ← 2.8107e-04
CONSTANT B ←  3.8214e-04
CONSTANT C ← -5.5210e-07

//takes output voltage from thermistor and converts it to temperature in ° C
FUNCTION Temperature(Vo:integer) RETURNS real
      R2 ← R1*(1023/Vo-1)
      logR2 ← log(R2)
      RETURN (1.0 / (c1 + c2*logR2 + c3*logR2*logR2*logR2))-273.15
ENDFUNCTION

T ← Temperature(Vo)
WHILE T < 100
      IF T >= temperatureSetPoint-0.5
            THEN
                  Heater ← 0 //switch off heater
            ELSE
                  Heater ← 1  //switch on heater
      ENDIF
      SLEEP 1000ms //delay for 1 second
ENDWHILE
```

Figure 2: Pseudocode for Heating Subsystem

Arduino program which is shown in the appendix. [4.1.2]
The circuit diagram of the heating subsystem is shown below:

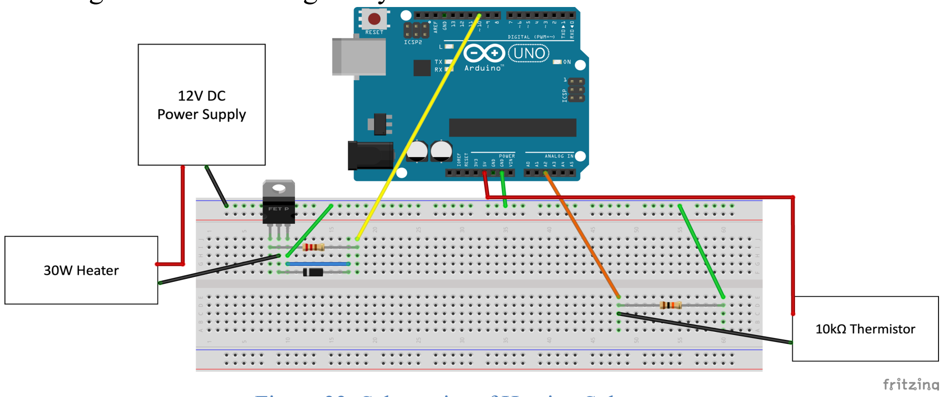


Figure 33: Schematics of Heating Subsystem

2.1.4 Results of Heating Subsystem

To test the thermistor, a thermometer was used when the thermistor was measuring the temperature of water. By comparing the temperature measured by the thermometer and the value displayed by the Arduino, it was determined that the error of thermistor measurement was controlled to within 0.1°C. Furthermore, to test the whole heating subsystem, different desired temperatures were set in each attempt. The results showed that the temperature of water after the heater is turned off and cooled down is within the desired temperature±0.5°C. When the temperature of water is lower than the desired temperature-0.5°C, the heater element would be turned on again to heat the water, thus keeping the temperature within the desired temperature ±0.5°C. Overall, due to the tests, the heating subsystem achieved the goal which was controlling the temperature within a ±0.5°C from the temperature setpoint. However, during the demonstration, the heating subsystem failed to connect with the UI therefore the temperature setpoint was manually adjusted from the Arduino codes instead of from the dashboard. Aside from connectivity issues the subsystem was able to run according to the specifications.

2.2 pH Subsystem

Section Authors: Xiao Sun

2.2.1 Description of pH Subsystem

The pH subsystem monitored the pH value in the bioreactor and maintained the pH value within a certain range to ensure the ideal growth rate of microorganism and thus maximise the efficiency of the bioreactor. Therefore, the bioreactor could help improve Uganda's medical service in terms of TB vaccination rate.

2.2.2 Specification of pH Subsystem

The subsystem was made up of a sensor [PH-100ATC Probe], an op-amp [MCP6022], a transistor [ZTX450] and two actuators [6V 1A peristaltic pump], the components worked together with the Arduino and the IOT subsystem to control the pH value. Also, it needed to upload the data to the cloud and execute order from the cloud to adjust the pH value to a certain value.

2.2.3 Design of pH Subsystem

<u>pH Probe Calibration</u>

The pH probe sometimes outputted a negative voltage which can't be read by the Arduino board. After reading the guidance made by TI [3] , it was decided that a 3.3V offset should be added to the signal by using an op-amp [4] since there was a 3.3V output on Arduino. Finally, a PH Buffer Standard Solution was used to establish the equation between voltage and pH value [$pH = \frac{V-1.8}{0.2}$] using MATLAB's polyfit() function [4.2.1].

<u>Pump Calibration</u>

It was inappropriate to use bang-bang control in this subsystem since it was a waste of acid and alkaline [precious in Uganda], which will increase the cost of the vaccine and the power consumption of the bioreactor. Plus, adding to much liquid into the bioreactor will eventually exceed its maximum capacity. Therefore, another equation was established to control the added amount [$x = \frac{V_{total}*(c_{now}-c_{target})}{c_{base}+c_{target}}$ or $\frac{V_{total}*(c_{now}-c_{target})}{c_{target}-c_{target}}$ ml]. Two 100ohm resistors were used to protect the collector pins of the transistor and give a fixed output to them via the PWM pins on the Arduino. The rate of the water flowing out of the pump per second was measured under that circumstance and established another equation [$delay = \frac{1000x}{0.00714}$ ms]. After the delay the pump would be turned off to give precise amount of chemicals into the bioreactor. [4.2.2]

<u>Error Calibration</u>

Still, a problem occurred as the pH probe's output voltage kept fluctuating. Therefore, a capacitor was used to stabilize it, however, it failed (possibly due to circuit problems). Eventually this problem was fixed by taking several values and average them in the code.

2.2.4 Results of pH Subsystem

In the first test the DC power supply and DMM were used to test if the op-amp was working as expected [actual offset is 3.2V no matter the input voltage]. The sensing and actuating part was tested separately by assigning given target pH and current pH values to the Arduino & breadboard to see if either pump run idle and using the IEP test board to give simulated voltage output and test if the correct pH was presented on serial monitor. Finally, the subsystem was tested with the ESP32 attached to the Arduino to see whether the data could be presented on the UI.

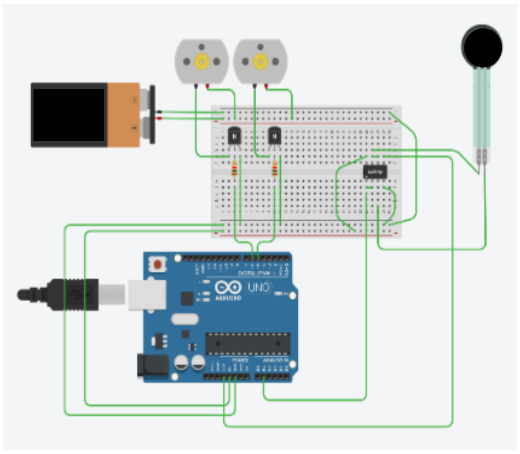


Figure 4

2.3 Stirring Subsystem

Section Authors: Sebastian Khosravani-Shiri

2.3.1 Description

To produce a Tuberculosis vaccine, the ingredients must be stirred to react fully. This sub-system was designated to controlling a spinning motor that stirs the mixture. The motor's RPM was decided by the user of the system through the UI, and the RPM was maintained by the system through a correctional procedure that loops forever.

2.3.2 Specification

The exact specifications for this subsystem:

- The user must be able to set an RPM between 500 and 1500.
- The real RPM of the motor must be maintained within plus or minus 20 RPM of the set point.

Both requirements were functional.

2.3.3 Design

In design, one must carry out requirement analysis before high level and detailed design. The first requirement meant that there must be a loop that increases the RPM of the motor until it reached a variable set by the user. The Arduino must communicate with the motor to alter the RPM. For this to happen, it must be able to translate from real RPM values to a Pulse Width Modulation value between 0 and 255 which could be placed in the AnalogWrite function. The second requirement meant that there must be a sensor collecting values and communicating with the Arduino. This communication involved translation of raw values to interpretable ones, and then varying the output voltage until an acceptable reading was collected by the sensor.

On to the high-level, detailed, and top-down design:
The first step to achieving the requirements was to design the entire sub-system. Here is the block diagram of the sub-system created:

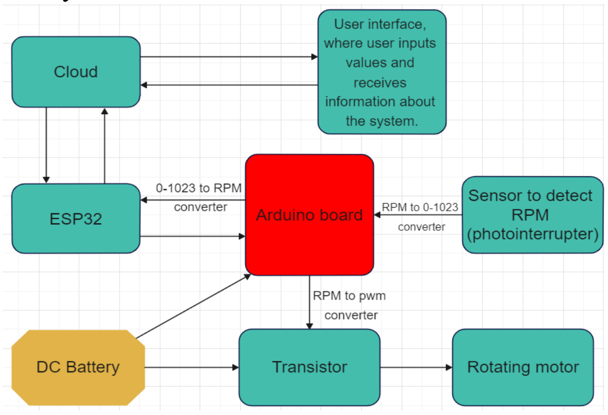
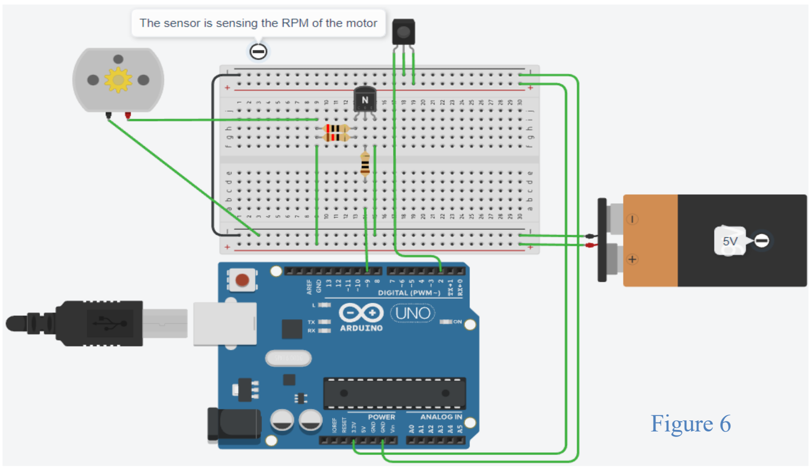


Figure 5

A breadboard circuit must be designed to achieve create the first piece of this sub-system. The voltage supplied to the motor must be changed to vary the RPM. This meant a transistor was necessary. The transistor used was a ZTX 450, which is an NPN transistor [5]. The base leg (middle) should be connected to the microcontroller via a resistor, to protect the circuit. The collector leg was connected to the DC battery and motor via a resistor, to protect the circuit. The current was too high for a microcontroller to power the circuit, so an external DC battery was necessary. The emitter leg was connected to the ground.
After creating a first draft circuit, it was connected to a microcontroller and a test Arduino code was uploaded. The pwm input was varied to make sure that the RPM was changing.
A problem occurred: the RPM was not changing, and it was far too high. Using the tachometer, it was shown that the RPM was way out of range (above 1500) and not changing regardless of pwm input in the AnalogWrite function. After altering the pwm from 0 to 255 and nothing changing, it was apparent that this was a circuit error. The ground of the microcontroller and the ground of the DC battery were not connected. After connecting the two, the RPM came down to the correct area, and varied with the pwm input.
The final problem was to choose suitable resistors. The exact resistance required in series with the collector leg was the resistance where the motor starts spinning just below the minimum RPM of 500. By varying this resistance, it was found that the prime resistance for this condition is 7.5 ohms. This resistance was achieved with a 12-ohm and 20-ohm resistor in parallel. The resistance between the base leg and microcontroller could be anything, so 10 ohms was used.
Using the stated improvements, a circuit was designed:



I decided to use pin 2 for sensing instead of A0 because the sensor was not giving stable readings when connected to A0.

Figure 6

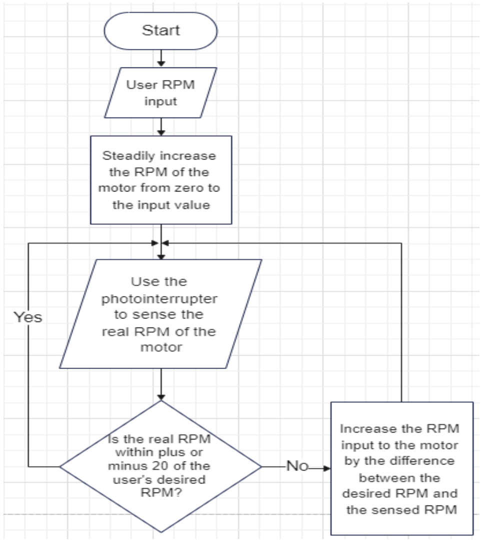
The next step was calibration. In other words, finding the relationship between the pwm input and the RPM, and then the relationship between the sensor value and the RPM. A table with three columns was created, labelled RPM, sensor value and pwm input. The pwm input was the independent variable. Using Arduino code, the pwm was varied and a tachometer detected the real RPM of the motor, and AnalogRead gave a sensor value. The table of collected data was put into MATLAB in three different arrays. MATLAB was used to find a line of best fit for both relationships:

The MATLAB code and graphs are in the appendix [4.3.1].

- Pwm=0.13116*RPM+40.927
- RPM=-37.481*sensorvalue+13190

The final step was to create a code using the linear relationships.
The requirements of the code:

- Ramp the RPM of the motor from 0 to the desired RPM.
- Sense the real RPM and correct it to the desired RPM.



Start
User RPM input
Steadily increase the RPM of the motor from zero to the input value
Use the photointerrupter to sense the real RPM of the motor
Is the real RPM within plus or minus 20 of the user's desired RPM?
Yes
No
Increase the RPM input to the motor by the difference between the desired RPM and the sensed RPM

Figure 77

A flowchart describing the process of the code was created. The Arduino code itself is placed in the appendix [4.3.2].

The biggest problem here was the initial inaccuracy of the sensor. When running the whole subsystem, the sensor gave wild readings until gradually reaching the correct values. To solve this, a sensor warm-up process was added in the code: the sensor would run 7 waste readings, by the end of which, it was detecting correct values. This was done with a for loop, and only happened for the first loop of the system.

2.3.4 Results

The whole subsystem (code and circuit) was connected and tested on the laptop and Arduino used in calibration. The subsystem met the requirements on this laptop.
The final step was to transfer to and test the subsystem on the group's main microcontroller and laptop. It was found that the ramp up process was correct, but the correction process failed. After investigation, this problem was identified: the relationship between sensor output and the real RPM had changed when placed on a new microcontroller and laptop. Thus, the relationship used in the code did not work as the code was correcting the RPM to incorrect values. This problem was found at the end of the last session, so a second sensor calibration could not be carried out. Therefore, the sub-system correction failed on demonstration, and only the first specification was met.

2.4 Communication between Arduino and ESP32

Section Authors: U. Gerguri

2.4.1 Description of the Communication Subsystem

The Communication subsystem between the Arduino and the ESP32 was responsible for enabling data transfer between the two devices. It utilized a two-way I2C connection to do that. The aim of the communication subsystem between the Arduino and the ESP32 was sending values from the ESP32 to the Arduino and vice versa.

2.4.2 Specification of the Communication Subsystem

This subsystem was made up of the Arduino, ESP32 and a level shifter. Via the I2C protocol, setpoints are sent from the UI to the ESP32 which in turn relayed the data to the Arduino. Simultaneously the Arduino read the data from its 3 subsystems and sent the values back to the ESP32 to be sent over the internet to the UI to be displayed.

2.4.3 Design of the Communication Subsystem

The ESP32 is the master device, and the Arduino is the slave device. The ESP32 would be sending three setpoints (Temperature, pH and RPM) to the Arduino, and the Arduino would send the current values of the subsystems (Temperature, pH and RPM).

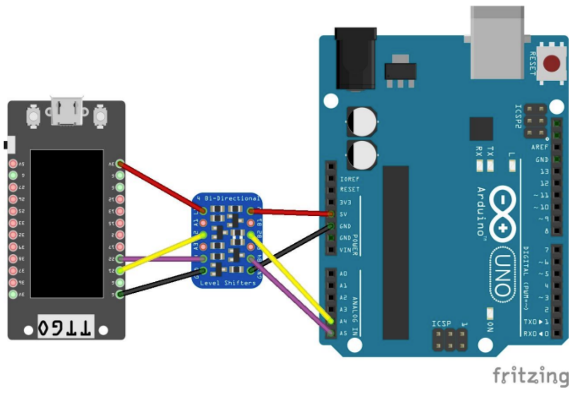


Figure 8

2.5 Communication between ESP and Cloud, and User Interface

Section Authors: O. Gaillard, L. Iskenderoglu

2.5.1 Description of Connectivity Subsystem

The purpose of this system was to connect the Arduino (all other subsystems) to the Cloud as a two-way connection. This allowed the bioreactor to be controlled from anywhere to match the specific parameters for the TB vaccine production parameters.

2.5.2 Specification of Connectivity Subsystem

The aim of the Connectivity system was to connect the Arduino to Wi-Fi, to allow the logging of data from all other subsystems. It also provided a user interface that allowed monitoring of data and allowing set points to be changed in real time. This was achieved by connecting an ESP32 to the Arduino which was used as a connection point.

2.5.3 Design of Connectivity Subsystem

Arduino IoT Cloud was used instead of the suggested ThingsBoard platform, since it has a better dashboard design creation and allows abstraction of MQTT protocol using its API. It also has several security features that make it safer to use as control system for the bioreactor.
The ESP connected 6 variables from the Arduino to the Cloud, each subsystem having a setpoint and a data value.

2.5.4 ESP

The ESP32 booted up, connected to eduroam Wi-Fi, connected to IoT Cloud, started the I2C protocol. The program then split into 2 threads:
One thread looped every 5 seconds, it got sensor data from Arduino and sent setpoints to Arduino. The ESP32 (The master) sent the setpoints via the I2C protocol, the Arduino (The slave) received these setpoints, these setpoints were then sent to each of the subsystems (Temperature, pH, RPM) and set.

While the other thread looped every 10 secs, sensor data was sent to the Cloud and whenever a setpoint was changed, the Cloud sent the new setpoints to the ESP32. This was done using the linked variables properties of the Arduino IoT Cloud API.

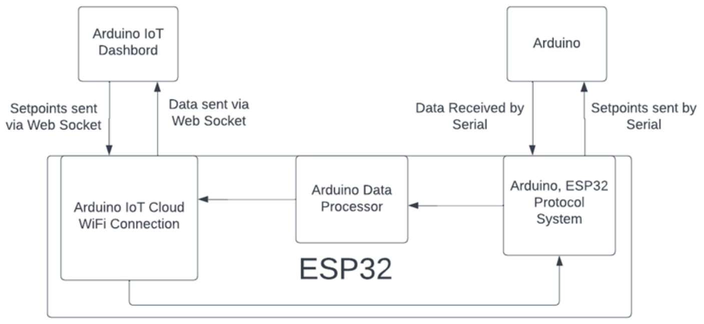


Figure 9: ESP32 Data Flow

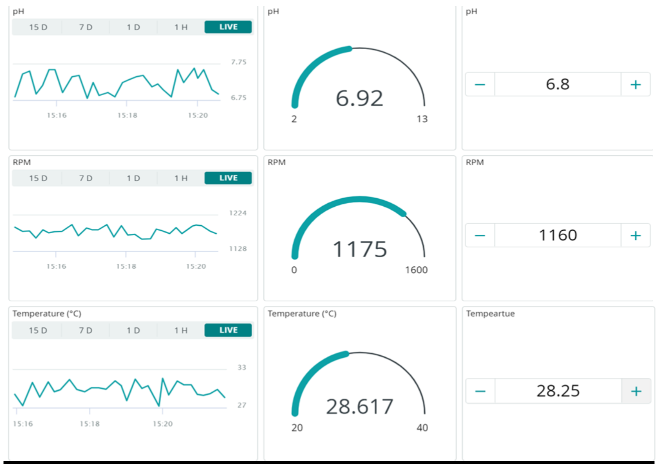


Figure 10: Dashboard UI

3 Overall System Integration and Summary

Section Authors: V. Jha, H. J. Ong

For the testing aspect of the system, we utilised three different types of data; namely, normal, extreme and abnormal. Normal data being comfortably inside the input range of each subsystem; extreme being on the edge of the input range; abnormal being outside the range. After the subsystem code was integrated with the hardware and connected through the circuit, unit testing on each subsystem was carried out. Initially, all the subsystems were working on one Arduino each, but for our integration process, all were connected to a single Arduino which was further connected to a single laptop where the data would be logged and set points for each subsystem could be inputted. Firstly, the code was tested on the Arduino IDE using print statements to check the sensor outputs on hard-coded setpoints and later was it integrated to the UI.

3.1.1 pH result summary

The testing for this subsystem worked really well, a range of input pH setpoints (from 0-14) gave appropriate results in the serial monitor and also the right output pH was displayed in a test with the UI. Accordingly, the peristaltic pumps were activated to change and eventually stabilise the pH. There was little uncertainty in the system.

Input	Data Type	Expected Result	Actual Result
4	Normal	4	4 ± 0.3
5	Normal	5	5 ± 0.1
6	Normal	6	6 ± 0.4
0	Abnormal	No value inputted	No output results
H	Abnormal	No value inputted	No output results
3	Extreme	3	3 ± 0.3
7	Extreme	7	7 ± 0.3

3.1.2 Heating result summary

Input	Data Type	Expected Result	Actual Result
27	Normal	27 ± 0.5	27 ± 0.5
30	Normal	30 ± 0.5	30 ± 0.4
33	Normal	33 ± 0.5	33 ± 0.5
120	Abnormal	No value inputted	No output results
C	Abnormal	No value inputted	No output results
25	Extreme	25 ± 0.5	25 ± 0.3
35	Extreme	35 ± 0.5	35 ± 0.5

The testing for this subsystem worked well, most of the allowed temperature range was input; and as expected, the heater heated until the desired temperature and switched off. However, if the setpoint was a bit far off from the room temperature, it took a while to get to that value. The uncertainty in this case was well within requirement specification limits (±0.3-0.5).

3.1.3 Stirring result summary

Input	Data Type	Expected Result	Actual Result
700	Normal	700 ± 20	700 ± 20 initially but ± 4650 later
1000	Normal	1000 ± 20	1000 ± 20 initially but ± 7820 later
1300	Normal	1300 ± 20	1300 ± 20 initially but ± 2410 later
5000	Abnormal	No value inputted	No output results
S	Abnormal	No value inputted	No output results
500	Extreme	500 ± 20	500 ± 20 initially but ± 3120 later
1500	Extreme	1500 ± 20	1500 ± 20 initially but ± 4890 later

The testing for this subsystem started well but the sensor output and RPM relation changed due to an issue related to the new Arduino and the laptop it was connected to, as mentioned previously in the stirring subsystem part of the report. Overall, the stirrer always reached the setpoint RPM value by starting slow and ramping up, but after that point, it kept failing to stabilise at the RPM and the uncertainty was very large (~±5000). Finally resulting in it increasing its speed over the setpoint till it reached its maximum possible RPM, after which the system stopped working.

3.1.4 Integration result summary

This was the part where we encountered the most difficulty in testing. Even though the connection code between Arduino, ESP32 and cloud was working perfectly in prior testing, there was some problem in connecting the UI to the combined code for all subsystems in the Arduino we were working on. This was a recurring issue, and we weren't able to fix it even when we tested the subsystems in an isolated manner, except for the case of pH, which did work as expected.

3.1.5 Comments on design process and conclusion

The design approach taken for all the subsystems was, in large part, done quite well as, in all cases, the code was able to communicate with the hardware. A part on the design process that could be improved is the connectivity. As we have faced some problems with this aspect of the combined system, combining the codes of the three Arduinos into one should have been prioritised and tested earlier. In the real world, our heating subsystem would also need a cooler since the setpoint could be lower than the room temperature. Similarly, the stirring subsystem would need to be recalibrated so that it works with the main device and microcontroller.

All in all, our bioreactor can be called a partial success in fulfilling the requirements due to all of its components working pretty well in isolation with generally low uncertainty but lacking in the full integration with UI. In the Ugandan context, our bioreactor, after its problems are rectified, is low-cost, power-efficient and requires little training to work with. This is a great solution for Uganda as this makes the vaccine affordable, uses local labour in the long-run and doesn't put much strain on the power grid or the reserves near Mbarara.

4 Appendices

4.1 Heating Subsystem

4.1.1 MATLAB Codes to calculate the constants of the Steinhart-Hart equation

```
L1 = log(33164.56);
L2 = log(11136.36);
L3 = log(2430.13);

Y1 = 1/272.15;
Y2 = 1/294.75;
Y3 = 1/330.45;

g2 = (Y2-Y1)/(L2-L1);
g3 = (Y3-Y1)/(L3-L1);

C1 = ((g3-g2)/(L3-L2))*(1/(L1+L2+L3))
```

```
C1 = 2.4186e-07
```

```
B1 = g2-C1*(L1^2+L1*L2+L2^2)
```

```
B1 = 1.8751e-04
```

```
A1 = Y1-L1*(B1+C1*L1^2)
```

```
A1 = 0.0014
```

```

L12 = log(12834.32);
L22 = log(5066.27);
L32 = log(2430.13);

Y12 = 1/291.65;
Y22 = 1/312.75;
Y32 = 1/330.45;

g22 = (Y22-Y12)/(L22-L12);
g32 = (Y32-Y12)/(L32-L12);

C2 = ((g32-g22)/(L32-L22))*(1/(L12+L22+L32))
```

```
C2 = 3.6688e-07
```

```
B2 = g22-C2*(L12^2+L12*L22+L22^2)
```

```
B2 = 1.5973e-04
```

```
A2 = Y12-L12*(B2+C2*L12^2)
```

```
A2 = 0.0016
```

```
L13 = log(12834.32);
L23 = log(8202.85);
L33 = log(5066.27);

Y13 = 1/291.65;
Y23 = 1/300.25;
Y33 = 1/312.75;

g23 = (Y23-Y13)/(L23-L13);
g33 = (Y33-Y13)/(L33-L13);

C3 = ((g33-g23)/(L33-L23))*(1/(L13+L23+L33))
```

```
C3 = -2.2651e-06
```

```
B3 = g23-C3*(L13^2+L13*L23+L23^2)
```

```
B3 = 7.9917e-04
```

```
A3 = Y13-L13*(B3+C3*L13^2)
```

```
A3 = -0.0022
```

```

A=(A1+A2+A3)/3
```

```
A = 2.8107e-04
```

```
B=(B1+B2+B3)/3
```

```
B = 3.8214e-04
```

```
C=(C1+C2+C3)/3
```

```
C = -5.5210e-07
```

4.1.2 Arduino Source Codes

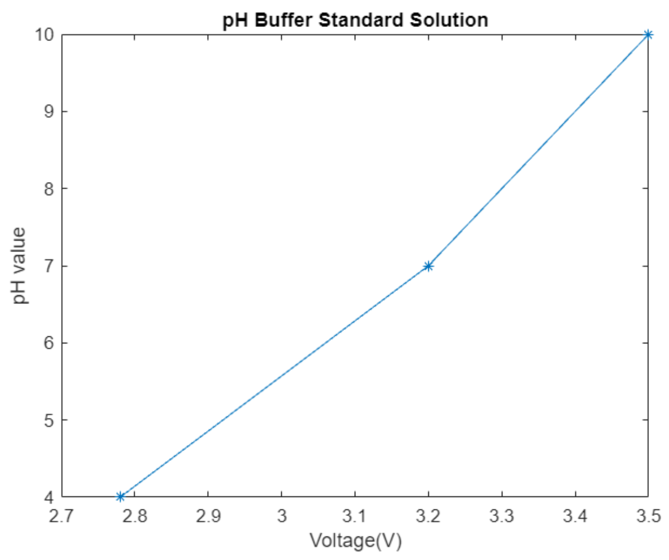
```
//defining thermistor and heater pins
#define ThermistorPin A2
#define HeaterPin 10
//creating variables
int Vo;
float R1 = 10000; //resistor connected to thermistor value
float logR2, R2, T;
float  A =   2.8107e-04,  B  =  3.8214e-04,  C  =  -5.5210e-07;  //constants
obtained during calibration for Steinhart-Hart Equation
float temperatureSetPoint;  //change temperatureSetPoint from dashboard

void setup() {
Serial.begin(9600);
}
float Temperature() {
  Vo = analogRead(ThermistorPin);
  R2 = R1 * (1023.0 / (float)Vo - 1.0); //calculating the resistance of
the thermistor using voltage divider formula
  logR2 = log(R2);
  return  ((1.0  /  (A  +  B*logR2  +  C*logR2*logR2*logR2))-273.15);
//Stenihart-Hart Equation
}
void loop()
{
  // Temperature read
  T = Temperature();
  Serial.print("Temperature: ");
  Serial.print(T);
  Serial.print("ºC");
  // Regulation
  if  (T  >  temperatureSetPoint-0.5)  //temperature  increases  by  around
1.6ºC after switched off
  {
    analogWrite(HeaterPin, 0);
    Serial.print("\nHeater OFF\n");
  }
  else
  {
    analogWrite(HeaterPin, 230);
    Serial.print("\nHeater ON\n");
  }
  delay(1000);
}
```

4.2 pH Subsystem

4.2.1 MATLAB Code for pH & voltage relationship

```
x = [2.78 3.2 3.5];
y = [4 7 10];
plot(x,y,"*-");
xlabel("Voltage(V)");
ylabel("pH value");
title("pH Buffer Standard Solution")
```



```
coefficient = polyfit(x,y,1);
```

4.2.2 Arduino Source Code

```
#define InputPin  A1
#define AcidPin  6
#define BasePin  5

float AddAcidBase(float CurrentpH,float TargetpH,float Volume){
float Amount,Cold,Cnew,Cacid,Cbase,pHacid,pHbase;  //calculate amount of
acid and base to be added
pHacid= 4;
pHbase= 9.2;
Cacid=pow(10,pHacid);// concentration of given acid
Cbase=pow(10,14-pHbase);// concentration of given base
Cnew= pow(10,TargetpH);
Cold= pow(10,CurrentpH);
  if (CurrentpH<TargetpH){
      Amount = Volume *(Cold-Cnew)/(Cbase+Cnew);
  }
  else if(CurrentpH>TargetpH){
      Amount = Volume *(Cnew-Cold)/(Cacid-Cnew);
  }
return Amount;   //in ml
}

int DriveMotor(float CurrentpH,float TargetpH,float Amount)
{ // drive motor to give base and acid
int x;
float time;
x = 100; //the minimum value needed to drive the motor
time = (Amount / 0.00714);  // calculate the time the motor need to be
turned on
  if (CurrentpH<TargetpH){
      analogWrite(BasePin, x);
      delay(time);
      analogWrite(BasePin, 0);
  }
  else if(CurrentpH>TargetpH){
      analogWrite(AcidPin, x);
      delay(time);
      analogWrite(AcidPin, 0);
  }
return 0;
}

int AutoControl(float pH,float pHSetPoint)
{
  if (pH<pHSetPoint)
  {
    return 1;
  }
  else if(pH>pHSetPoint)
  {
    return 2;
  }
  else
  {
    return 3;
  }
}


void setup()
{
  Serial.begin(9600);
  pinMode(AcidPin,OUTPUT);
  pinMode(BasePin,OUTPUT);
  pinMode(InputPin,INPUT);
}

void loop()
{
int temp,Auto,nouse;
float pHValue,pHSetPoint,Volume,Amount,pH;
Volume = 50;   // type in the volume of the liquid later
pHSetPoint = 7;
temp = analogRead(InputPin);
pHValue = 7-((temp/1023*5)-3.2)/0.2; // turning sensor reading to pH value,
maximum reading is pH = 0
pH = pHValue;
Auto = AutoControl(pHValue,pHSetPoint);
if (Auto == 1){
  //message here that will inform user the pH is too low
Amount = AddAcidBase(pHValue,pHSetPoint,Volume);
Volume = Volume + Amount; //accumulate the volume
nouse = DriveMotor(pHValue,pHSetPoint,Amount);
Auto = 0;
}
else if (Auto == 2){
  //message here that will inform user the pH is too high
Amount = AddAcidBase(pHValue,pHSetPoint,Volume);
Volume = Volume + Amount;
nouse = DriveMotor(pHValue,pHSetPoint,Amount);
Auto = 0;
}
else if (Auto == 3){
  //message here that will inform user the pH is just as the suggested
value
Auto = 0;
}
delay(500);
}
```

4.3 Stirring Subsystem

4.3.1 MATLAB Code

```
pwm=[50;100;150;200;250;255];
motorRPM=[0;510;880;1250;1550;1600];
sensor=[350;340;330;316;313;308];

mdl1=fitlm(motorRPM,pwm)
```

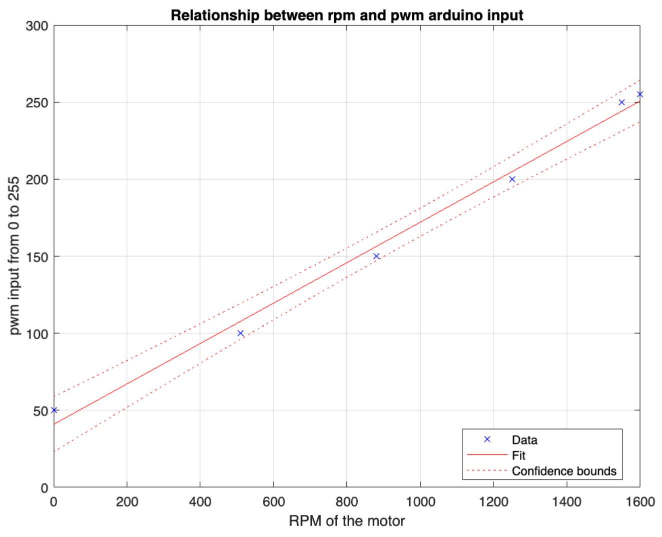
```
mdl1 =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate       SE         tStat       pValue  
                   ________    _________    ______    _________

    (Intercept)     40.927        6.4274    6.3675     0.0031192
    x1             0.13116     0.0057262    22.906    2.1521e-05


Number of observations: 6, Error degrees of freedom: 4
Root Mean Squared Error: 8.04
R-squared: 0.992,  Adjusted R-Squared: 0.991
F-statistic vs. constant model: 525, p-value = 2.15e-05
```

```
figure; plot(mdl1)
grid on
xlabel("RPM of the motor")
ylabel("pwm input from 0 to 255")
title("Relationship between rpm and pwm arduino input")
```



```
mdl2=fitlm(sensor,motorRPM)
```

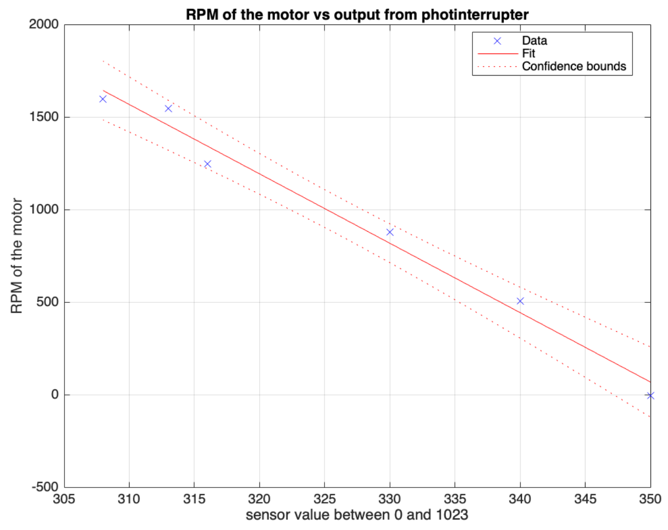
```
mdl2 =
Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
                   Estimate      SE       tStat       pValue
                   ________    ______    _______    __________

    (Intercept)      13190     789.87     16.699    7.5347e-05
    x1             -37.481      2.419    -15.494    0.00010128


Number of observations: 6, Error degrees of freedom: 4
Root Mean Squared Error: 89.9
R-squared: 0.984,  Adjusted R-Squared: 0.98
F-statistic vs. constant model: 240, p-value = 0.000101
```

```
figure; plot(mdl2)
grid on
xlabel("sensor value between 0 and 1023")
ylabel("RPM of the motor")
title("RPM of the motor vs output from photinterrupter")
```



4.3.2 Arduino Source Code

```
#define sensorpin 2
#define outputpin 9

float setRPM, setRPMelec, realRPMsense, realRPM, rampRPM=0, rampRPMelec=0,
desiredRPM=1200;
int ramp=0,c, waste=0, starter=0, inac;

void setup() {
  pinMode(sensorpin, INPUT);
  pinMode(outputpin, OUTPUT);
  Serial.begin(9600);
  setRPM=desiredRPM;
}

void loop() {
  if (starter==0){
    setRPM=desiredRPM;
    starter=1;
  }

  setRPMelec= 0.13116*setRPM + 40.927;

  while (rampRPM!=setRPM) {
    Serial.println("Building up to desired RPM");
    if (rampRPM<setRPM) {
      rampRPMelec=0.13116*rampRPM + 40.927;
      analogWrite(outputpin,rampRPMelec);
      rampRPM+=1;
      delay(10);
      Serial.println(rampRPM);
    }
    else if (rampRPM>setRPM) {
      rampRPMelec=0.13116*rampRPM + 40.927;
      analogWrite(outputpin,rampRPMelec);
      rampRPM-=1;
      delay(10);
      Serial.println(rampRPM);
    }
    Serial.println("Done!");
  }

  delay(3000);
  Serial.println("Sensing...");
  if (waste==0) {
    Serial.println("warming up the sensor...");
    for (c=0;c<7;c++){
      realRPMsense=analogRead(sensorpin);
      realRPM = -37.481*realRPMsense + 13190;
      Serial.println(realRPM);
      delay(3000);
    }
  }

  realRPMsense=analogRead(sensorpin);
  realRPM = -37.481*realRPMsense + 13190;
  Serial.println(realRPM);
  delay(6000);

  inac=realRPM-desiredRPM;

    if (inac>=20) {
      Serial.print("The difference between the real RPM and the desired
RPM: ");
      Serial.println(inac);
      Serial.println("The set RPM will be corrected to reach a suitable
real RPM");
      delay(500);
      for (c=0;c<=inac;c++) {
        setRPM-=1;
        Serial.println(setRPM);
      }
    }
    else if (inac<20 && inac>-20) {}
    else if (inac<=-20) {
      Serial.print("The difference between the real RPM and the desired
RPM: ");
      Serial.println(inac);
      Serial.println("The set RPM will be corrected to reach a suitable
real RPM");
      inac=-1*inac;
      Serial.println(inac);
      delay(500);
      for (c=0;c<=inac;c++) {
        setRPM+=1;
        Serial.println(setRPM);
      }
    }
    Serial.println("Correction completed");
    delay(2000);
  waste=1;
}
```

5 References