Challenge 2: Global Healthcare

CS-EEE Specialist Team 13 Final Report

Authors: Edgar Tsang, Joshua Liao, Amber Du, Anna Talantova, Lucy Cui, Taiqiao Li, Scott Taylor, Peter Ling, Chi Zhou, Haoyu Gu

1 Introduction

Tuberculosis is a disease caused by the bacteria Mycobacterium Tuberculosis [1] and in 2021, caused over 10 million people to fall ill [2]. However, Tuberculosis is a preventable disease by the administration of a vaccine, more specifically, the BCG (Bacillus Calmette-Guérin) vaccine. As of 2021, there has been a shortage of BCG vaccines due to "the closure of BCG production plants, reduction of production, and the withdrawal of the BCG Connaught strain" [3]. Shortages reduce the uptake of the BCG vaccine and increase the risks of infection and consequent death from the disease.

Our problem space regards the production of the BCG vaccine. The BCG vaccine involves the cultivation of a weakened version of the bacteria Mycobacterium Bovis, a close relative of Mycobacterium Tuberculosis [4]. The weakened bacteria "triggers an immune response within the body", developing the antigens against the disease [5], reducing the risk of infection. The country within the problem space is Uganda, a country within the highest 30 TB/HIV burden countries [6]. People with an HIV infection have an increased risk of sickness from TB, with "TB being one of the leading causes of death among people living with HIV" [7]. As such, greater uptake of the BCG vaccine is required in Uganda due to increased risk of fatal sickness from combined infection of the two diseases. The problem presented is a need for a production facility of the vaccine in Uganda to facilitate this increased uptake.

Our task was to design, develop and test the circuitry, software and interfaces required to remotely monitor and control the parameters of a small-scale bioreactor system, simulating a larger system and production facility in Uganda for the vaccine. This system must be able to provide the ideal environment for the cultivation of bacteria and the factors involved in growth must be monitored and adjustable. The factors we chose to monitor within this system were Temperature, pH, and Stirring RPM (Revolutions per minute).

Initially, we began designing our system with the following characteristics:

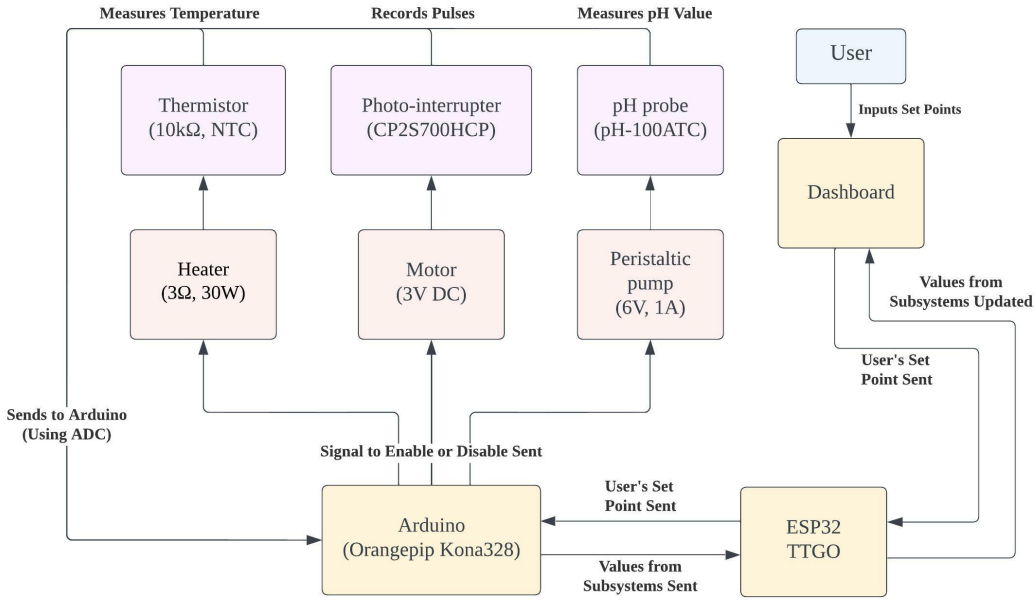
- Monitors pre-set factors.
- Continually updates and maintains system conditions.
- Stays within desired user range.
- Creates a desirable environment for cell culture.

	Component(s)	Subsystem
1	Heater {3Ω, 30W} + NTC Thermistor {10kΩ NTC}	Temperature
2	Motor {3V DC} + CP2S700HCP Photo-Interrupter (RPM)	Stirring RPM
3	Peristaltic Pumps {6V, 1A} + PH-100ATC pH Probe (pH) + MCP 6022 op-amp + npn BJT (ZTX450) transistor	pH
4	Arduino Orangepip Kona328 + ESP32 TTGO + Bi-Directional Level Shifter	Network Connectivity
5	Power Supply {12V, 7AH rechargeable lead acid battery}	Power

Furthermore, the design specification provided for this Bioreactor requires the temperature to be maintained "at a set point in the range **25-35° C** to within **± 0.5°** C of the set point.", the stirring speed to be maintained "at a set point in the range **500-1500 RPM** within **± 20 RPM** of the set point", and the pH to be maintained "at a setpoint in the sensing range **3-7**" with a default pH "at an **optimum 5**". To regard the bioreactor as successful, the bioreactor should follow this specification and work within the provided

ranges. Likewise, the bioreactor should also send data in real-time to a dashboard so the system information can be viewed by those managing the bioreactor.

In the larger context of our integrated system, the physical bioreactor components serve to monitor and maintain their respective factors and keep it in accordance with the predetermined ranges. The Arduino serves to control these physical components centrally and communicates with the ESP32 which serves as a gateway for network connectivity, connecting the system to the internet. A database was established on a cloud platform to store the data and desired ranges. This permits for the retrieval and storage of data from the sensors and conversely, the retrieval and storage of ranges. A dashboard was also established for human monitoring of the subsystems and changes to the subsystems. The relation between components is highlighted in Figure 1 below.



[Figure 1, Diagram showing full connectivity of systems]

Following on from Figure 1, to test the system, it should successfully take the desired ranges as an input on the dashboard from the user and utilise them to control the pH and temperature of the solution, establishing an environment suitable for cell cultivation. The subsystems should also be capable of relaying data for their related measurements to the database which can be monitored on the dashboard, allowing for conditions to be monitored in real-time. Each subsystem should be fully functioning and stay within the range provided, while communications between systems should also be fully functioning, measurable through accurate graphs on the dashboard.

Holistically, the system we have designed should fulfil the assigned task when implemented, as it permits monitoring and adjusting of factors related to creating a desirable environment for the bacteria to grow. Furthermore, the conditions of the bioreactor can be maintained within the values listed in the design specification if set by the user and would be a fully automated process capable of running continuously until stopped. In terms of the larger context of a Ugandan vaccine production facility, this design would allow for remote monitoring of the processes and environment within the facility in real-time, while all changes to setpoints can also be done remotely and will have an immediate impact on the system. This opens to wider implications of unmanned facilities and external control of the production environment.

2 Heating Subsystem

2.1.1 Description of the Heating subsystem:

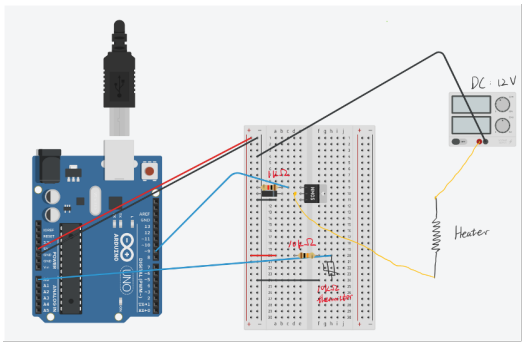
The heating subsystem was designed to constantly monitor the temperature of the water and maintain the temperature of the water at a set point. This set point is set to be 30° by default. However, it can be adjusted by the user through the user interface.

2.1.2 Specification of the Heating subsystem:

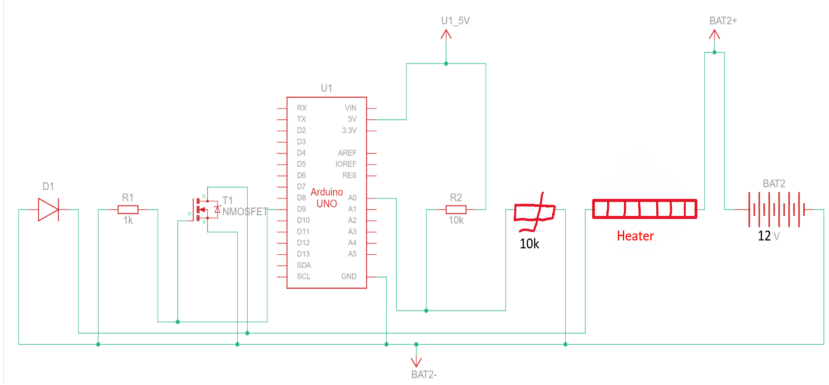
The temperature of the water would be maintained at a set point within the range of 25° to 35°. It was further required that the temperature would be maintained at within ± 0.5° of this pre-determined set point.

2.1.3 Design of the Heating subsystem:

The heating subsystem consisted of a 3Ω, 30W heating element and a 10kΩ thermistor that detects the temperature of the water. A transistor (IRLB8748 MOSFET) is used to switch the heater on and off as a solid-state switch. The circuit diagram and schematics of the heating subsystem is shown in Figures 2 and 3.



[Figure 2, Circuit Diagram of the heating system]



[Figure 3, Schematic of the heating system]

Initially, the thermistor was connected in series to a 10kΩ resistor to create a potential divider circuit. 10kΩ was chosen so that small changes in the voltage across the thermistor can be detected. We then recorded the voltage reading of the thermistor at its corresponding temperature and used excel to find the formula that relates the voltage across the thermistor and the temperature. The formula (1) as seen below allows us to directly convert the voltage reading of the thermistor into its corresponding temperature.

$$\text{Temperature} = -0.089 * \text{Voltage} * 1023.0 / 5.0 + 69.081 \qquad \text{Eq. (1)}$$

For maintenance of the temperature (flowchart in the appendix {1}), the system was operated such that if at any point, temperature is below the set point, the heater is switched on; if the temperature is above the set point, the temperature is switched off. Carry over heating due to the remaining heat in the heating element was also considered. After further observation, the range by which the solution was heated over the set point was within the acceptable range of 0.5°, allowing us to ignore it in our system.

3 Stirring Subsystem

3.1.1 Description of the Stirring subsystem:

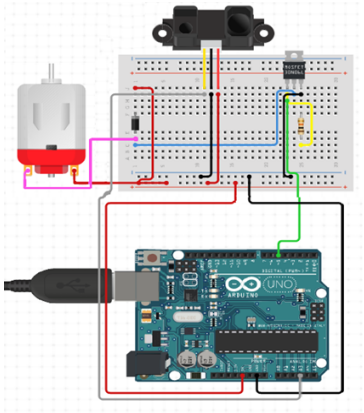
The stirring subsystem composes of a stirring rod, a CP2S700HCP photo-interrupter and a 3V DC motor with a maximum current of 1 A. Both the photo-interrupter and the motor are connected to an Arduino board using jumper cables. Stirring of the solution allows for equal distribution of the other factors such as temperature and pH, keeping the solution identical throughout the container. The speed of stirring is realized by measuring RPM from the photo-interrupter and its speed is influenced by changing the current passing through the motor.

3.1.2 Specification of the Stirring subsystem:

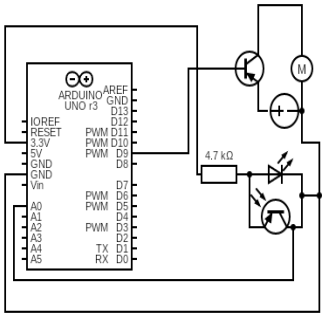
The stirring speed should be within a given range: 500RPM – 1500RPM. The required speed should be controllable by the user via a dashboard.

3.1.3 Design of the Stirring subsystem:

The Arduino board takes an input of the photo-interrupter from pin A0 and measures the number of pulses per unit of time. When a wave is reflected by the gear, it signals a pulse indicating half a rotation has occurred as there are two gears.



[Figure 4, Circuit Diagram of the stirring system]



[Figure 5, Schematic of stirring system]

As shown in Figure 4 and 5, to control the speed of motor, a transistor is connected in series with a DC motor, utilizing PWM output from Arduino to control the current flow in the circuit of motor. Since the output of PWM is 8 bits, we choose a value between 0- 255 to output. The speed of the motor is calibrated to get a relation between the output on PWM and the motor's actual speed. Through experimentation, the value of output which takes response to 500RPM and 1500RPM respectively can be obtained (60,98). One thing that should be further considered is after collecting data on the speed with corresponding output between output value =50-110, the fitting function (2) below is obtained. It us used as the transformation of data input to RPM output.

$$RPM = 1000000/(2*(time1+time2)) * 60 \qquad \text{Eq. (2)}$$

Where time1 is the first pulse and time2 is the second pulse

When the subsystem is started, it provides an output of 80, resulting in an RPM roughly at 1000. While the system is running, RPM is constantly calculated. When it falls below 980, the output is increased to maintain the RPM in the range. Vice versa, if RPM is above 1020, the output is decreased. The required speed can be controlled by the user via a dashboard, changing the PWM output indirectly, in turn leading to different currents and motor speeds. The process is also shown in a flowchart in the appendix {2} alongside the code {3}.

4 pH Subsystem

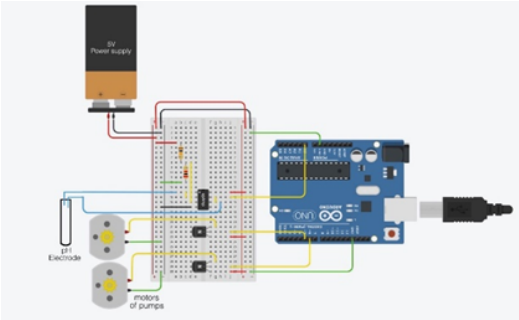
4.1.1 Description of the pH subsystem:

The pH system was designed to be able to adjust and maintain the constant pH of the environment. In the program, the default pH was set to 5, however, it can be manually adjusted by the user using the interface created by our team.

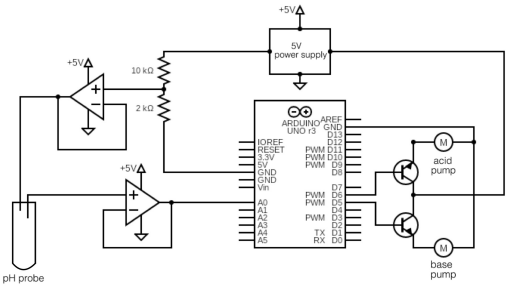
4.1.2 Specification of the pH subsystem:

In the Arduino code, the pH value is set to be a range to allow for an uncertainty of ± 0.5. The pH probe constantly records and checks the pH values. If it is below the set value, the pump with the alkaline solution will be switched on and start pumping the alkaline solution until it reaches the set value. If the value of pH is above the user input, pump with acidic solution will be switched on to pump in acid until the value reaches user input. The process - a closed feedback loop - then repeats so the program runs continually. The Arduino code is shown in Appendix {4} along with the flowchart representing this process {5}.

4.1.3 Design of the pH subsystem:



[Figure 6, Circuit Diagram of the pH system]

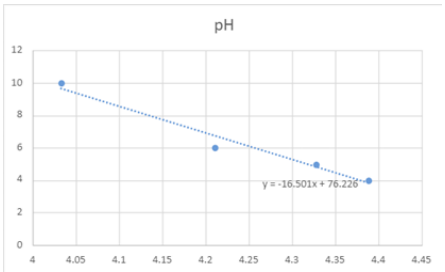


[Figure 7, Schematic of pH system]

In pH system, two op-amps are used to control the input of the voltage from the electrodes to the Arduino UNO. The first op-amp offset is set to around 0.5 V to make all values of voltage positive as the output voltage of the pH probe is bipolar. The second op-amp amplifies the small current due to high impedance of the electrodes. Two transistors, which act as switches, are used to control the motors of the pumps. The schematic diagram and circuit diagram are shown in Figures 6 and 7 above. For calibration, readings from serial monitor, and measurements of real pH values were taken for four different solutions. Then the readings were mapped from 0-1023 to 0-5 to convert it to voltage. The data, in Appendix {6}, plots the graph of pH values against voltage. The graph is a straight line as shown in Figure 8. Then an equation (3) is generated to convert serial monitor readings to actual pH values:

$$pH = -16.501 \times Voltage + 76.226 \quad \text{Eq. (3)}$$

The range of pH values taken is from 4 to 10.

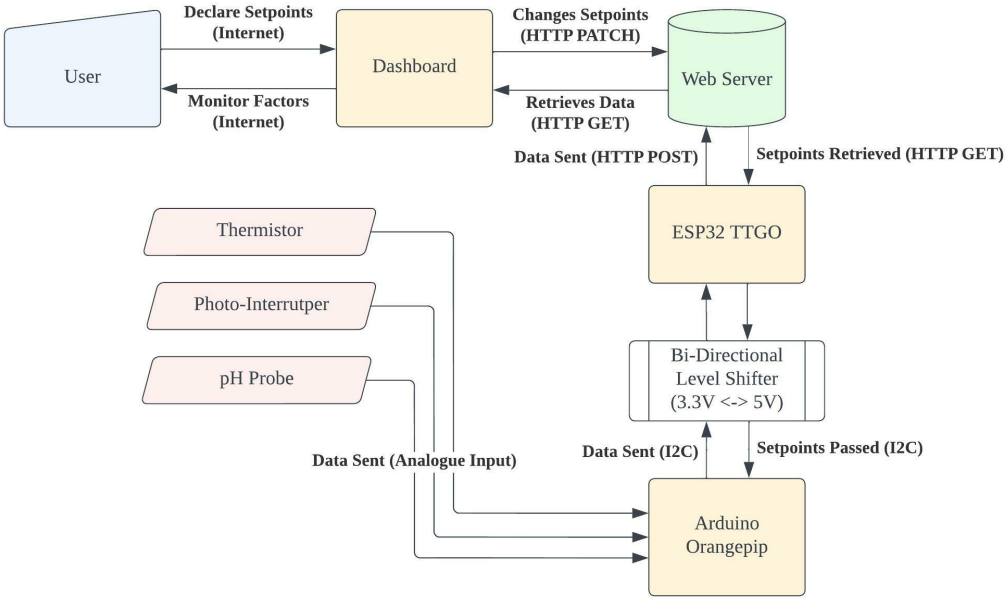


[Figure 8, pH (voltage) graph]

5 Network Connectivity

5.1.1 Overview of Network Connectivity:

In our design, we utilise an Arduino, ESP32, Webserver and Dashboard to connect the user with the individual subsystems, allowing remote monitoring and control of the subsystems. Figure 9 below highlights the connections between the constituents and indicates what protocols and communication buses are used.



[Figure 9, Block Diagram of Network Connectivity]

5.2 Communication between Arduino and ESP32

5.2.1 Description of the Communication between Arduino and ESP32:

The Arduino is used to both control the subsystems and relay data from the subsystems to the ESP32. The ESP32 then handles the data received but also sends the set points from the webserver to the Arduino, upon which the operation set points of the subsystems are changed if necessary.

5.2.2 Specification of the Communication between Arduino and ESP32:

Communication between the two devices must be uninterrupted. The Arduino should update the ESP32 with real-time values from the sensors. The ESP32 should update the set points in the Arduino based on the User's input.

5.2.3 Connection between Arduino and ESP32:

The ESP32 and Arduino are connected using the I2C (Inter-Integrated Circuit) serial communication bus. Each device is connected by four cables to a level shifter which reduces the voltage so the 5V Arduino does not damage the 3.3V ESP32. The four cables are the SDA, the SCL, ground and power. The protocol uses the SDA (serial data line) cables to transmit data between the devices and the SCL (serial clock line) is used to enable the data to be sent synchronously. In our system the ESP32 was set up as the master and an Arduino for each subsystem were slaves. The pins used on the ESP32 were manually assigned: SDA was 21 and SCL was 22. Each Arduino had a slave address: temperature controller Arduino was 9, RPM controller was 8 and PH controller was 7. In the code for the ESP32 and Arduinos, the Wire library was used to allow for I2C

communication. Furthermore, due to an inability to send double values through the Wire methods, we used the method dtostrf, a Double to String conversion, to send the data as a string. This string was then directly passed to the webserver.

A UART connection was also considered, however, there were difficulties in establishing a successful connection between the two devices. Furthermore, accounting for our use of 3 Arduinos, one for each subsystem, I2C supports multiple Master/Slave setups which is an advantage over UART. Our current configuration for the Master/Slave setup is by using a parallel connection to utilise the same level shifter to send data to the ESP32.

5.3 Communication between ESP32 and Cloud

5.3.1 Description of the Communication between ESP32 and Cloud:

Our design includes a webserver that stores readings and user set points. The ESP32 communicates with the webserver over an internet connection and updates the dashboard with values from the Arduinos, whilst set points are relayed back to the ESP32 to be used on the Arduinos.

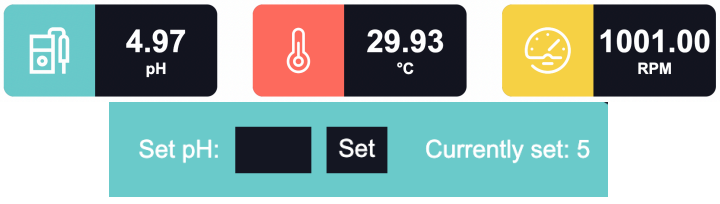
5.3.2 Specification of the Communication between ESP32 and Cloud:

Our requirement for the ESP32 and Cloud was for real time updates to be uploaded and displayed on the dashboard, whilst the desired set points can be assigned remotely.

5.3.3 Connection between ESP32 and Cloud:

For the backend, a web server built on Node.js using Express, a web application framework, was established. This backend exposes a REST API, handling GET and PATCH requests for settings data and GET, POST and DELETE requests for logging data. These endpoints allow clients to communicate with the server and interact with the data we store in an authenticated MongoDB cloud database. The backend is hosted and deployed via the service Heroku for reliable uptime.

The front end is a webpage responsible for displaying the dashboard interface as seen in the appendix {7}. It immediately presents the latest readings for our subsystems at the top (Figure 10) and follows with line graphs of their respective historical data. The webpage is created from HTML, CSS using flexbox for layout and vanilla JS. Every second, a GET request is made to the data endpoint, querying for new data to update the state of the page; the response is JSON data such as that displayed in Figure 8. When a user changes the settings for a subsystem via the input boxes (Figure 11), a PATCH request is made to the settings endpoint to update the latest settings with the JSON data in the request's body.



[Figure 10, Values as displayed on Dashboard]

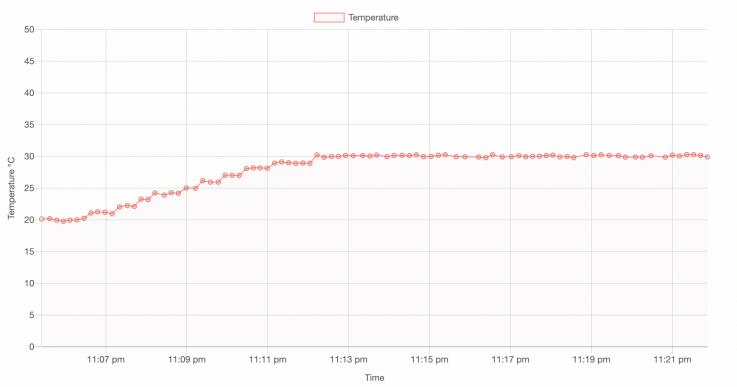
[Figure 11, Input Box on Dashboard]

The ESP32 receives real-time data from the Arduino and then makes a POST request with the new data in the body as JSON to the data endpoint. Every 5 seconds, the ESP32 makes a GET request to the settings endpoint and receives the latest settings in a JSON format. If the settings received have changed since the last check, the ESP32 would communicate this back to the subsystems.

6 Overall System Integration and Summary

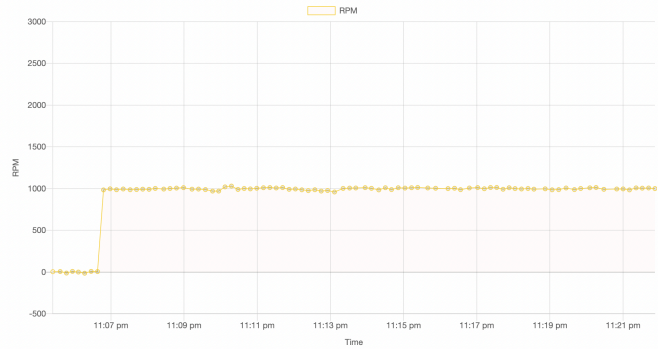
6.1.1 Results from System

In terms of the subsystems, their integration into the system can be measured by the data produced and displayed on the dashboard, as this should be updated in real-time according to the conditions in the subsystems. We tested system as a whole and gained the following data in the graphs below.



[Figure 12, Readings from Dashboard - Temperature Subsystem]

As can be observed in Figure 12, the steady increase from the room temperature 20° water to our desired temperature of 30° in 5 minutes and maintenance of the temperature at 30° thereafter indicates that our temperature subsystem is fully functioning. Variation of the temperature stays within 0.5° of 30°, confirming that the subsystem meets the design specification.

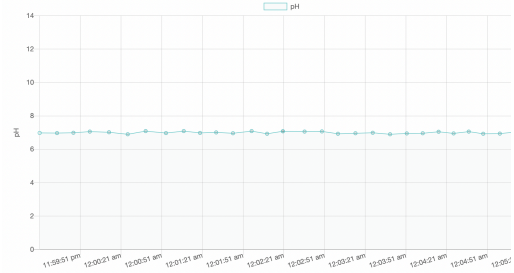


[Figure 13, Readings from Dashboard - Stirring Subsystem]

Figure 13 indicates that the stirring subsystem meets the design specification. While there was an initial pause due to a disconnection, the RPM was then kept at 1000 RPM, fluctuating within 20 RPM.

In testing the pH system, one acidic solution of known pH value of 5 and water were tested. The graphs are shown in Figure 14 and Figure 15 respectively.

[Figure 14, pH of acidic solution versus time]



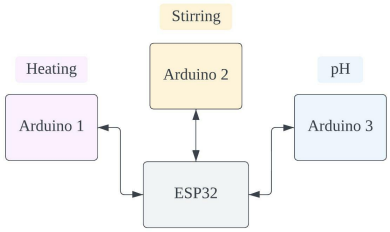
[Figure 15, pH of water versus time]

From Figure 14, it can be observed that the value of pH reaches 5 in around 4 minutes and remains constant with insignificant fluctuations around the set point. From the graph, the maximum fluctuation which occurred at 11:13 a.m. is around 0.5 which is acceptable by the specification. From Figure 15, the value of pH fluctuates around 7, which should be the pH value of water, with uncertainty within 0.5. Therefore, the pH probe is viable. However, further testing of the system cannot be taken because one of the pumps is broken and acidic and alkaline solutions are not given.

The graphs in Figures 12, 13, 14, and 15 are taken directly from our dashboard, and the data comes directly from the subsystems and are updated in real-time. This indicates that the network we established between the ESP32, Cloud and Arduino is successful. The ESP32 is communicating with both the webserver and Arduino, while both the back end of the webserver and front end of the webpage are successful - as can be seen in Figures 8 and 9 of the connectivity subsystem - in processing, retrieving, and storing the data within this bioreactor system.

6.1.2 System Integration

It must be noted that while each subsystem worked individually, issues arose getting the subsystems to work simultaneously consistently. While we were able to get the system fully integrated and working, this was inconsistent and had to be fixed after running for a longer period. We were also unable to contain the system within one container and instead separated the subsystems into multiple containers. These were major reasons behind why three Arduinos were used as opposed to one in our original design. Another major reason was that we had the ability to do so due to our choice of I2C communication between the Arduinos and ESP32 as opposed to UART. It allows for an ease of debugging and testing of the subsystems and leads to a distributed system with the Arduinos (Figure 16), which is beneficial as maintenance of the subsystems can be accomplished without affecting the other subsystems. However, consideration must be made towards the extra work and wiring required which increases the time required for setup.



[Figure 16, Distributed System]

Correspondingly, one way our design could be improved is through improving the systems to account for overshoots, such as with the heating system. While the overshoot can be ignored in our design, if the viable range was smaller, the overshoot would likely be past the range. To reduce this error, a PID (proportional-integral-derivative) control algorithm [8], an algorithm commonly used in the industry, can be implemented. While greater tuning will be required to set up this algorithm, the overshoot can be reduced, and the factors can be controlled to a greater precision.

6.1.3 Conclusion

Ultimately, our designed small-scale bioreactor meets the design specifications and can be implemented successfully. Our use of a dashboard on the web allows for remote monitoring and control, allowing the vaccine production environment maintenance to be fully automated. In terms of the larger context of a production facility in Uganda, our small-scale bioreactor design and suggested improvement, will be a reliable system for a larger bioreactor system. With a distributed implementation and PID control algorithm, the vaccine production system is easily scalable and highly precise with the opportunity to add more factors to be monitored and controlled.

7 References

[1]
"TB profile," *worldhealthorg.shinyapps.io*. [Online]. Available: https://worldhealthorg.shinyapps.io/tb_profiles/?_inputs_&lan=%22EN%22&entity_type=%22group%22&group_code=%22global%22

[2]
World Health Organization, "Tuberculosis," *www.who.int*, 2020. [Online]. Available: https://www.who.int/health-topics/tuberculosis#tab=tab_1

[3]
N. Mayor, C. Fankhauser, V. Sangar, and H. Mostafid, "Management of NMIBC during BCG shortage and COVID -19," *Trends in Urology & Men's Health*, vol. 12, no. 1, pp. 7–11, Jan. 2021, doi: 10.1002/tre.783.

[4]
"BCG vaccine | medicine," *Encyclopedia Britannica*. [Online]. Available: https://www.britannica.com/science/BCG-vaccine

[5]
World Health Organization, "How do vaccines work?," *www.who.int*, Dec. 08, 2020. [Online]. Available: https://www.who.int/news-room/feature-stories/detail/how-do-vaccines-work#:~:text=How%20vaccines%20help

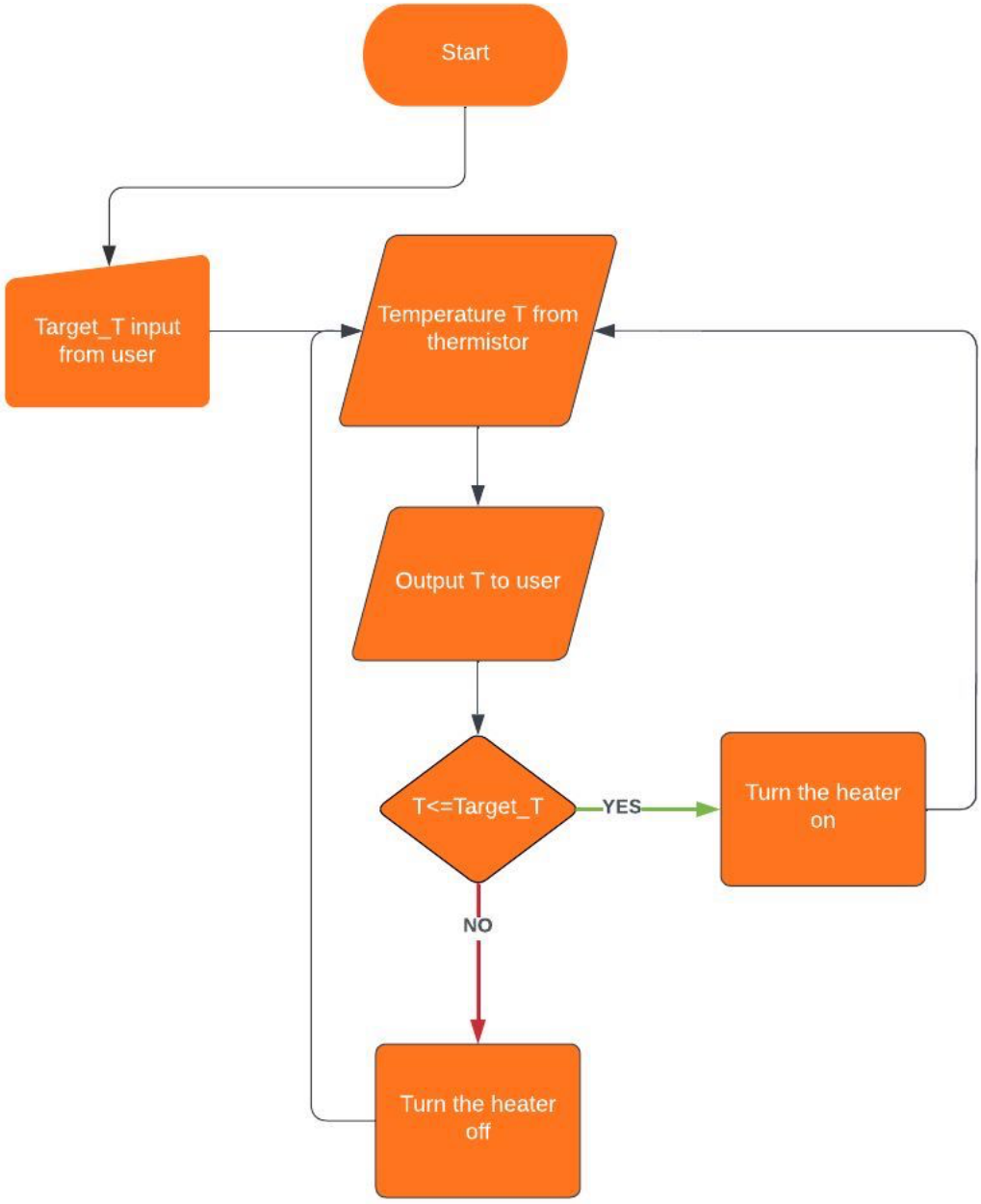
[6]
"High burden TB countries - 2018 lists - TBFacts," *TBFacts*, 2018. [Online]. Available: https://tbfacts.org/high-burden-tb/

[7]
CDCTB, "TB & HIV Coinfection," *Centers for Disease Control and Prevention*, Sep. 01, 2022. [Online]. Available: https://www.cdc.gov/tb/topic/basics/tbhivcoinfection.htm#:~:text=People%20living%20with%20HIV%20are

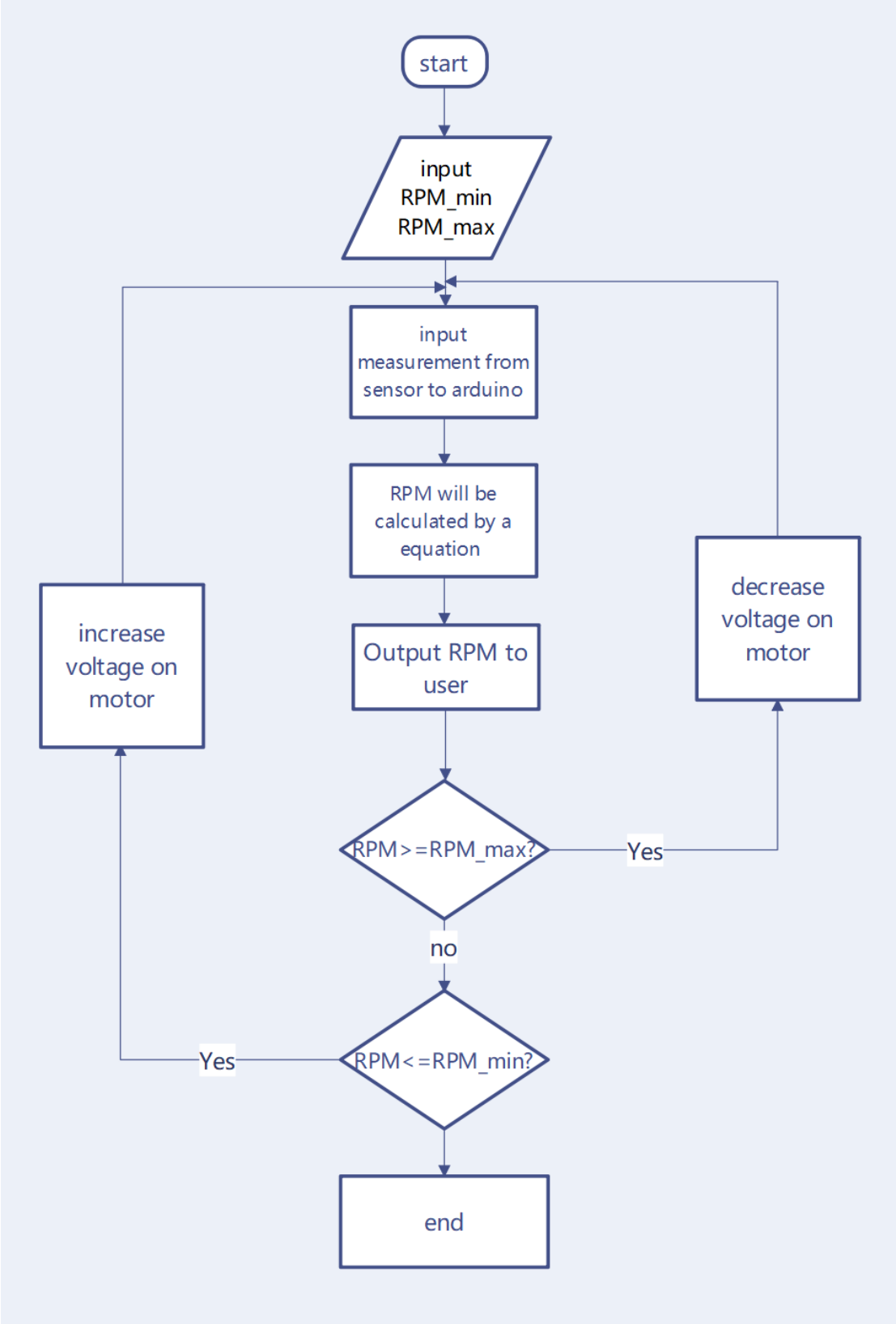
[8]
National Instruments, "PID Theory Explained - National Instruments," *Ni.com*, 2019. https://www.ni.com/en-gb/innovations/white-papers/06/pid-theory-explained.html

8 Appendices

{1} Flowchart for Temperature Subsystem



{2} Stirring Subsystem Flowchart



{3} RPM Arduino Code

```
#define SLAVE_ADDR 8
#include <Wire.h>
#include <string.h>
```

```
const int motorPin = 9;
const int photoPin =A0;
float rps;
int rpm;
float time1;
float time2;
int user = 80;
double range = 1350;
char* endPointer;

void setup()
{Serial.begin(9600);
 pinMode(motorPin, OUTPUT);
 pinMode(photopin, INPUT);
 pinMode(A0, INPUT);
 pinMode(8, OUTPUT);
 Wire.begin(SLAVE_ADDR);
 Wire.onRequest(requestEvent);
 Wire.onReceive(receiveEvent);
}

void requestEvent() {
 char sendRPM[4];
 dtostrf(rpm, 4, 0, sendRPM);
 Serial.println(sendRPM);
 Wire.write(sendRPM);
}

void receiveEvent(int howMany)
{
 char temp = "";
 while(Wire.available()) // loop through all but the last
 {
  char c = Wire.read(); // receive byte as a character
  temp += c;         // print the character
 }
 double temprange = strtod(temp, &endPointer);


 if (temprange !=  NULL) {
  range = temprange;
 }
}

void loop()
{ time1 = pulseIn(echoPin, LOW); // Time continues for not reflecting wave
 time2 = pulseIn(echoPin, HIGH); // Time continues for reflecting wave
```

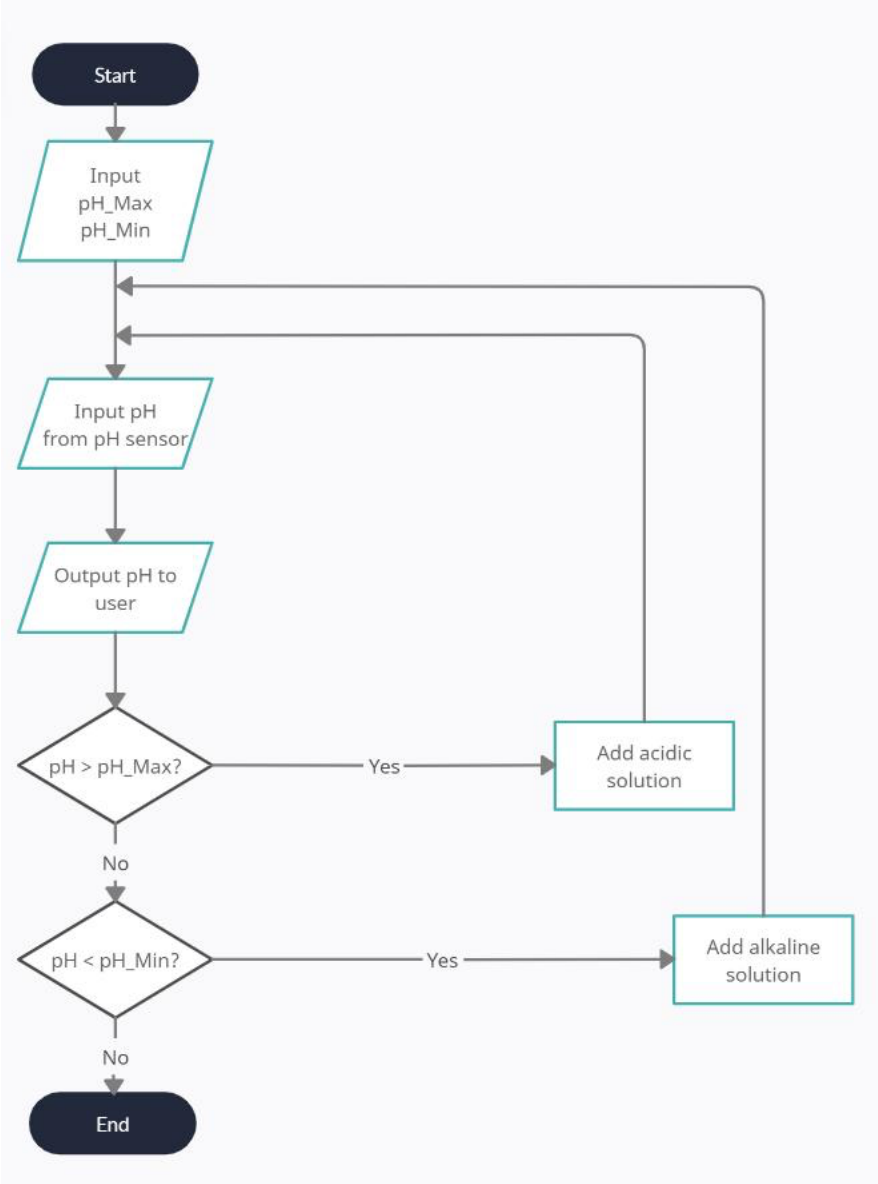
```
  rpm = 60*1000000/(2*(time1+time2)); // formula for calculation
  int input = analogRead(photoPin);
  Serial.print("\nRPM: ");
  Serial.print(rpm);
  if (rpm > 1350){
    user--;
  } //auto-adjustment if speed is higher than expected
  if (rpm < 650) {
    user++;
  } //auto-adjustment if speed is lower than expected
  analogWrite(motorPin,user);
} //control speed from user phase
```

{4} pH Arduino code

```
#define SLAVE_ADDR 7
#include <Wire.h>
#include <string.h>

float pHv;
float pH;

double range = 5;
char* endPointer;

void setup() {
  Serial.begin(9600);

  pinMode(A0, INPUT);
  pinMode(5, OUTPUT);
  pinMode(6, OUTPUT);

  Wire.begin(SLAVE_ADDR);

  Wire.onRequest(requestEvent);
  Wire.onReceive(receiveEvent);

}

void requestEvent() {
  char sendPh[5];
  dtostrf(pH, 5, 2, sendPh);
  Serial.println(sendPh);

  Wire.write(sendPh);

}

void receiveEvent(int howMany)
{
  char temp = "";
  while(Wire.available()) // loop through all but the last
  {
    char c = Wire.read(); // receive byte as a character
    temp += c;         // print the character
  }
  double temprange = strtod(temp, &endPointer);

  if (temprange !=  NULL) {
    range = temprange;
  }
}

void loop() {
  volatile int sensorvalue = analogRead(A0);
  volatile float V = (float)(sensorvalue * 5) / 1023;
  pH = -16.501 * V + 76.226;
  pHv = range;
  if (pH <= pHv + 0.5 && pH >= pHv - 0.5) {
    digitalWrite(5, LOW);
```

```
    digitalWrite(6, LOW);
  } else if (pH > pHv + 0.5) {
    digitalWrite(5, HIGH);
  } else {
    digitalWrite(6, HIGH);
  }
  Serial.print("sensorvalue: ");
  Serial.print(sensorvalue);
  Serial.print(", pH: ");
  Serial.println(pH);
  delay(500);
}
```

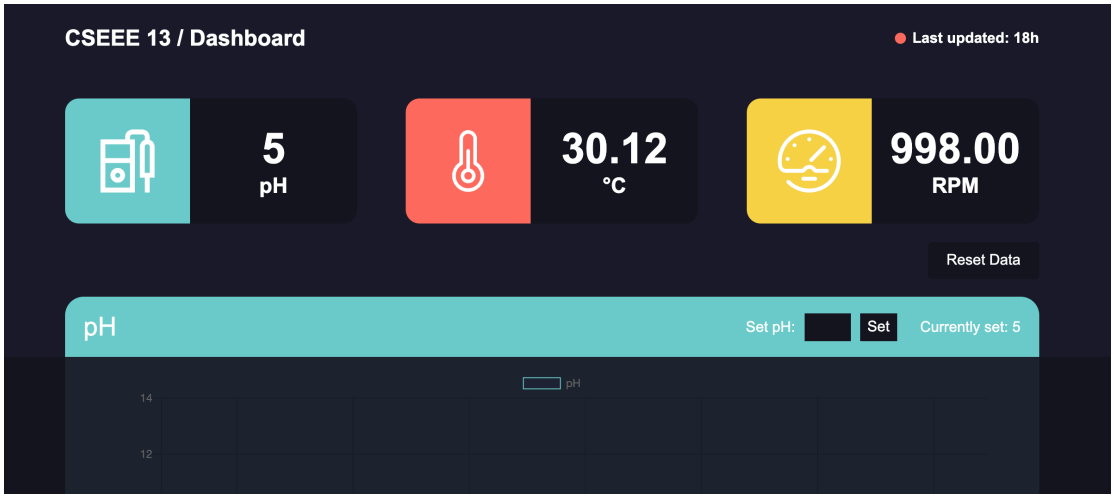
{5} pH Subsystem Flowchart



{6} Table 1, pH and Voltage table

pH value	4	5	6	10
Serial Monitor Reading	897-904	884-889	860-865	821-831
Voltage	4.009-4.058	4.316-4.34	4.119-4.224	4.009-4.058
Average value of voltage	4.389	4.328	4.212	4.034

{7} Dashboard


CSEEE 13 / Dashboard
Last updated: 18h
5
pH
30.12
°C
998.00
RPM
Reset Data
pH
Set pH:
Set
Currently set: 5
pH
14
12