Challenge 2: Bioreactor Control System

CS-EEE Specialist Team Final Report

Authors: Keyu Ji, Helen Zhang, Ediz Cinbas, Annaelle Mansuy, Jash Shah, Akhilesh Sivananthan, Wenhao Teng, Yiwei Wang

Referencing coversheet

The questions below will help you check if you have correctly referenced any external sources you may have used when creating your report. External sources includes the teaching material provided to you. If you do not follow these referencing guidelines then you may commit academic misconduct and you may be penalized.

- What referencing style have you used?
  - IEEE referencing style
- Have you included references for any external sources (including the teaching material provided to you) that you have used?
  - Yes
- Where you have specifically referred to an external source (including the teaching material provided to you) in your report have you included in an in-text citation?
  - Yes
- Where you have directly quoted material from an external source (including the teaching material provided to you), have you used quotation marks ("…") and provided an in-text citation?
  - Yes

If you have answered no to any of the questions above please consider reviewing your work and checking that you have not commit academic misconduct.

AI Tools

"AI Tools are more widespread that you might have realized. They can be really useful and help to make our writing clearer and more understandable. However as with all tools, they are only useful when we know how to use them. Part of this process is understanding where we are using them and being transparent about how and when we are using them."

- Which AI tools have you used when writing? This includes spelling and grammar checkers as well as text predictors.

  Grammarly
- How have you used them? For example was it throughout the document? Or in a specific paragraph or section?

  In the summary of system paragraph
- If you had to add settings or a prompt please add that below?

Introduction

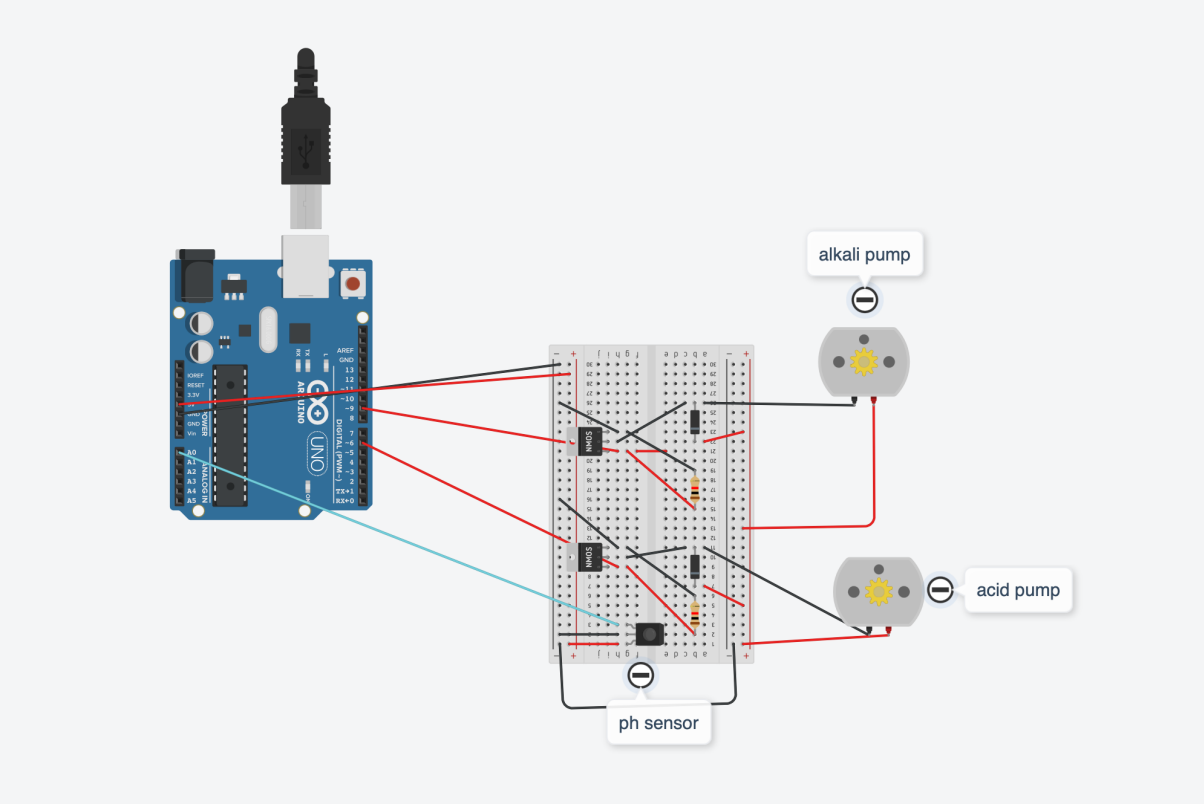
Mycobacterium tuberculosis (TB) is a bacterial disease that affects the lungs. BCG (Bacillus Calmette–Guérin) is a vaccine made to treat TB. Despite being preventable and curable, TB kills 2 million people a year [1]. Therefore, we propose a bioreactor in Uganda to produce BCG and reduce TB infections around the area. BCG is developed in an environment that requires specific temperatures, acidity, and solution mixing. As a result, a precise control system is needed to facilitate these temperatures, acidity, and mixing processes. The optimal values for these three criteria are 6.7 pH (in the range of 6.5 pH to 7.2 pH), 37 °C (35), and 200 rpm [1] from our initial findings, but this may differ in the following practice. The machine needs to be capable of being monitored and adjusted so that the scientists in charge can change these values for different batches and optimize the production process further.

Since we are CS-EEE collaborated, EEE focuses on constructing subsystem circuits and Nucleo coding; CS then focuses on the connection of the user platform using ESP32. Specifically, inside EEE, Helen and Alisa are in charge of the pH subsystem; Akhilesh is responsible for the stirring subsystem, and Wenhao is in charge of the heating subsystem. For the CS group, Annaelle and Jash focus on the part on the communication between Uno and ESP. Moreover, Ediz and Fradrake work together on the communication between ESP32 and the cloud, plus data logging and visualization.

Here is the design specification for this bioreactor system. To start off, it is assumed that the inlet water temperature is in the range of 10-20 degrees Celsius. Subsequently:
The temperature control system must maintain the temperature at a set point in the range of 25-35°C to within± 0.5°C of the set point [2].
The stirring subsystem must maintain the stirring speed at a set point in the range of 500-1500 RPM within ±20 RPM of the set point [2].
The pH subsystem must maintain the pH value at a set point in the sensing range 3-7. By default, this subsystem should maintain the pH at an optimum 5 [2].

Subsystem Designs

2.1 pH Subsystem



Shown in figure 2.1.1 is the wiring configuration where the Nucleo microcontroller (represented by an Arduino Uno board but using the same pinout configuration as the Nucleo) is connected to both the pH sensor and the pH pumps which function as control elements.

A pH subsystem consisting of a pH probe and electrically driven pumps are provided for measuring and maintaining a required pH range, which is optimal for biological reactions involved in the bioprocess. Alkali/acidic solutions can be supplied to the bioreactor as required to maintain the pH of the medium at a desired range. This provides the favorable conditions for growth and productivity of biological entities that are important in bioprocesses, like vaccine production for instance.

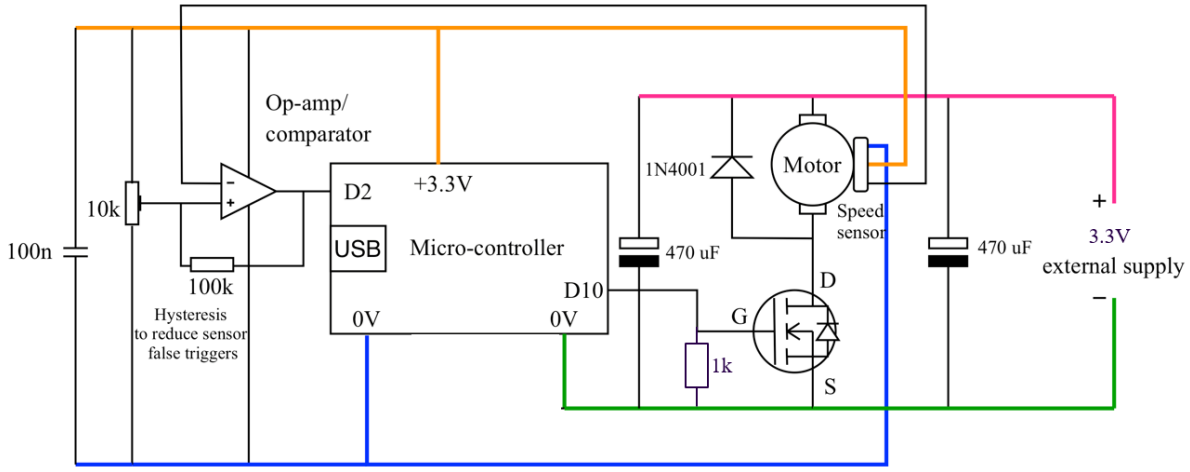
The STM32 Nucleo board, which is a microcontroller, controls the pH subsystem. This component reads the data that is being obtained from the pH probe that is constantly measuring the pH of the solution. When it discovers that the pH is not at the desired pH level, it uses the data value to control a pump that modifies the solution and brings the pH to the desired pH level.
The pH probe, a device that measures pH of a solution, consists of a glass tubing that measures pH in of the variation of the voltage with the pH of the experimental solution. The circuit board recognizes this voltage to measure the pH value.
Two dose pumps are connected to the Nucleo on the board to alter the pH level. One pump is for adding base to increase the pH level and the other pump is for adding an acid to decrease the pH level.
These pumps are powered by an external power source or battery of 6V. The pH probe is attached to the analogPin(A1) of the Nucleo. The pH probe sends a signal voltage (The signal voltage is corresponding to the pH level of the liquid) to the micro controller.

For clarification, black wires are used to connect to ground and the red wires are used for pump signal and 3.3V output from the Nucleo board.
Last but not the least, a feedback control loop as a highlight of the Nucleo coding (PI: Proportional – Integral) would be implemented in the Nucleo to maintain pH at the desired set point. The Nucleo would send a signal to the appropriate pump if the pH level were to deviate from the target value set by the user to add acid or base to the solution to bring the pH back to the set point. The code for the pH subsystem PI control can be viewed in Appendix C.

The table below shows the calibration data for the pH probe.

Test	PH Analog reading	Actual Value (pH)
PH measurement	153	4
PH measurement	581	7
PH measurement	999	10

2.2 Stirring Subsystem



Shown in figure 2.2.1 above, is the wiring configuration for the stirring subsystem where the Nucleo microcontroller is connected to the speed sensor and stirrer motor, with several additional components

1. Purpose:

Mixing (stirring) the solution in the cup at a set speed (RPM).

2. System components：

Power supply: 3.3V DC Power Supply and 3.3V from Microcontroller (STM32)
MOSFET: Controls the gate voltage to regulate the source and drain currents to maintain the speed of the motor at a desired value (RPM).
Monitor: The speed sensor is the component used to measure the speed of the stirrer and how it changes over time.
Motor: It is connected to a blade (or propeller) via a gearbox and shaft, and this blade is spun by the motor to mix the solution.

3. Working process:

a. Measure the frequency of the speed sensor pulses generated from the speed sensor signal getting amplified by the operational amplifier when the blade of the motor passes the sensor, triggering it to send a signal (represented by "*speedsensorpin*" in our code) to the microcontroller (pin D2).
b. The speed (RPM) of the motor is then calculated using the formula Speed = frequency*15. This formula was derived from the calibration of the thermistor by recording the analog

output at different temperatures, plotting a line of best fit and then forming an equation relating the analog output to the temperature of the solution.

c. The error (RPM*e*) is the difference between the setpoint and the current temperature. *RPMe = setpointStir - RPM*

d. PI control, as the name implies, uses proportional and integral terms to control the output. They can be calculated using the set values with the above measurements. The output of PI control was implemented using the code in Appendix D.

e. Finally, the value of the PWM signal is calculated. If the power is greater than the upper limit of the set range, then take the maximum value of the set range (255), and if the power calculation results in the required power being within the set range, then send the resulting power to the heater. However, if the power required is less than or equal to 0, it sets the power to 0. The code for the stirrer subsystem PI control can be viewed in Appendix D.

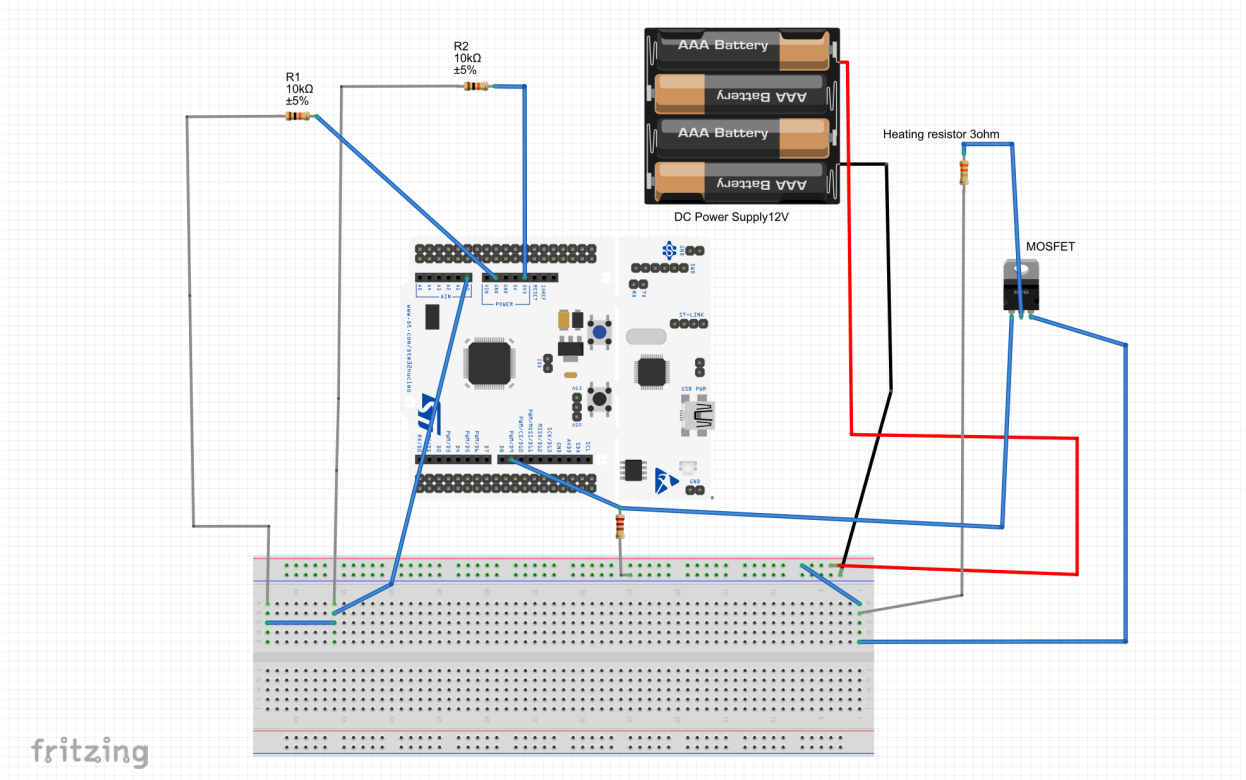
The table below shows the calibration data for the speed sensor.

Test	Calculated speed (RPM)	Actual Speed(rpm)
stirring measurement	441	418
stirring measurement	798	769
stirring measurement	1236	1211

2.3 Heating Subsystem

The heating subsystem consists of a thermistor to measure the temperature and a heater to increase the temperature of the solution. The temperature of the solution is measured using a potential divider consisting of a 10 kilo-Ohm thermistor in series with a 10 kilo-Ohm resistor.

Fig 2.3.1 below shows the wiring configuration of the heating subsystem.



1. Purpose:

Heating the solution in the cup to the set value as fast as possible and remains stable within a certain range. (27 degree for our setup)

2. System components：

Power supply: 12V DC Power Supply and 3.3V from Microcontroller (STM32)

MOSFET: Control the gate voltage to regulate the source and drain currents to maintain the power of the heating resistor within the set range. (0-25W in our setup)

Monitor: Also called thermistor in the above, represented by a 10-kilo-ohm resistor, used to detect the temperature and how it changes over time.

Heater: (Represented by a 3-ohm resistor).

3. Working process:

a. Read the value of temperature from thermistor and sent the analog value (represented by "*tempsensorpin*" in our code) to microcontroller (from pin A0)

a. Then temperature value will be calculated by formula *T= tempsensorpin*0.0842-19.3675* in our laptop. This formula was derived from the calibration of the thermistor by recording the analog output at different temperatures, plotting a line of best fit and then forming an equation relating the analog output to the temperature of the solution.

b. The temperature is obtained to calculate the difference between it and the set value, which is known as the error (*Te*) where *Te = setpointTemp - T ,* with *setpointTemp* being the setpoint (desired value) for the temperature.

c. PI control, as the name implies, uses proportional and integral terms to control the output. They can be calculated using the set values with the above measurements.

d. Finally, the heating power is calculated, if the power is greater than the set range, then take the maximum value of the set range (25W), and if the power calculation results in the required power being within the set range, then send the resulting power to the heater. However, if the power required is less than or equal to 0, it sets the power to 0. The code for the heating subsystem PI control can be viewed in Appendix E. A MATLAB simulation of the heating subsystem is provided in Appendix F.

The table below shows a sample of the calibration data for the thermistor. The full range of data collected can be viewed in Appendix 3.

Test	Thermistor Analog reading	Actual Value(°C)
1	594	30.6
2	628	33.4
3	671	37.6

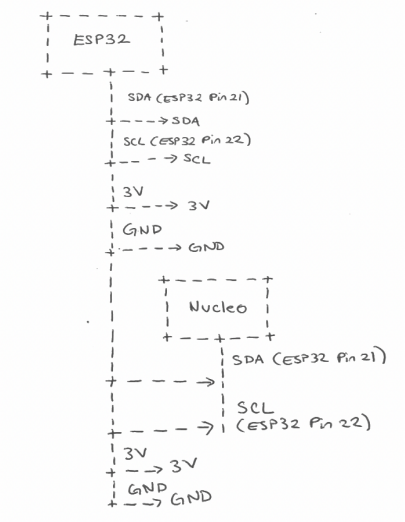
2.4 Communication between Nucleo and ESP32

The purpose of this subsystem is to facilitate the communication of data between the Nucleo board and ESP32 to exchange data comprising of inputs and outputs. The data exchanged comes from the outputs from the individual subsystems with temperature (˚C) from the heating subsystem, stirring speed (RPM) from the stirring subsystem, and the pH from the pH subsystem which is sent to the ESP32. The ESP32 allows for connectivity, and data sent to the ESP32 will be sent to a server displayed on a user-interface. Data can also be sent back to the Nucleo board from the ESP32 if the values are modified on the user-interface to control the subsystems. To put this in context, the data communicated to the ESP32 and back allows for remote control of the subsystems when engineering personnel are not situated in Uganda where the bioreactor is located.

To implement communication between the ESP32 and the Nucleo board, an I2C protocol is used, sending and receiving information through the SCL and SDA pins [3] with the ESP32 on "master mode" and Nucleo on "slave mode" [4]. An I2C protocol is chosen for this design due to its

simplicity in wiring so it makes it less complex for engineers to build and allows for communication between multiple devices if needed [3]. There are four connection wires with the GND connected to GND, 3V to 3V and the SCL and SDA of the Nucleo are connected to the ESP32's 22 and 21 pins [3]. The code works with the use of the standard 'Wire' library where in every set interval of time a request for data is made, and the data is sent across [3]. Refer to Appendix B for more information on the code. **Figure 2.4.1** shows a hand-drawn diagram of the connections between the ESP32 and Nucleo board.

Figure 2.4.1:



Technical Testing: The success of this subsystem is confirmed by checking if a message is received in the serial monitor of the Arduino IDE with a print statement. After testing the communication between the Nucleo to the ESP32, it is confirmed that the Nucleo, when connected to the right pins, can successfully send messages to the ESP32.

2.5 Connectivity and User-Interface

Since the bioreactor will be placed in Uganda, and the scientists on the project could be located elsewhere (possibly the United Kingdom), the system needs to be controllable remotely in order to allow for quick access to components even without physical personnel present. It is also unreasonable to assume the scientists will be familiar with the programs used to implement the system, so a simple GUI over the cloud needs to be implemented to allow for easy access to the bioreactor controls. Thus, there exists a need for connectivity to the internet and a graphical user interface, which can be achieved using the ESP32's Wi-Fi capabilities and an online IOT (Internet of Things) management software like Thingsboard.

The bioreactor is a crucial part of vaccine production and is capable of destroying itself by heating up too much or supplying more than the maximum voltage of the stirring motor if there was to be wrong instructions sent to it. Therefore, checks had to be placed within the software in order to only set the temperature, pH, and stirring speed variables to values within expectations. See Appendix A1 for how this was implemented using max and min values.

To send values to the cloud, a lightweight protocol needed to be used in order to communicate to the Thingsboard client. The industry standard and the recommended protocol of Thingsboard is MQTT (Message Queuing Telemetry Transport), which is an efficient publish-subscribe protocol.

[5] This means values can be synced and linked together to allow for Thingsboard to change values on the ESP32 and vice-versa. Thingsboard also has a valuable dashboard feature which visualises telemetry on a GUI that can be accessed by people given permission to (Diagram 2.5.1). This makes the system secure and convenient. Seen in Diagram 2.5.2, the system relies on MQTT protocol to establish the communication. The connection is implemented in code using the Thingsboard library's call back functions seen in Appendix A2.

Diagram 2.5.1

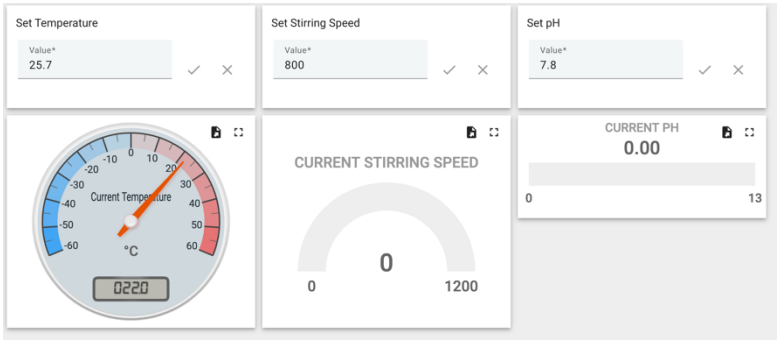
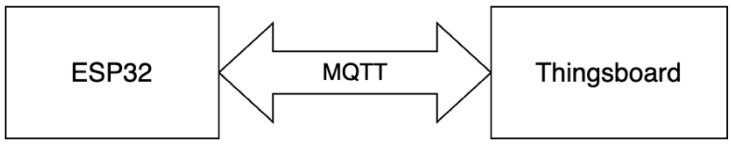


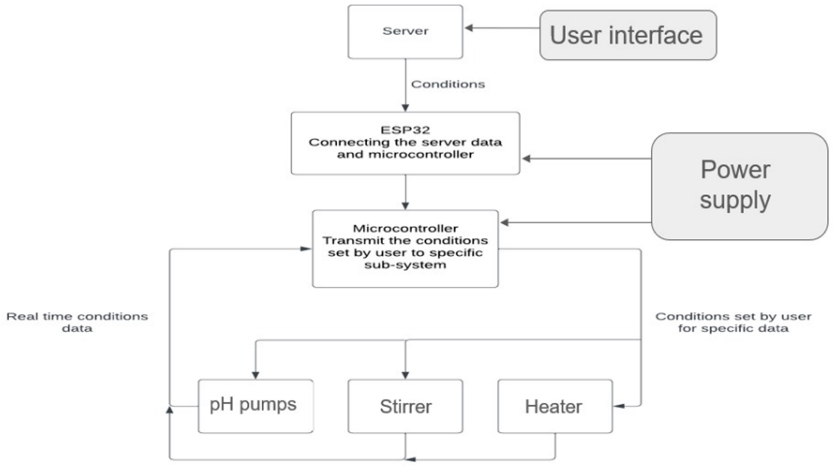
Diagram 2.5.2



Thingsboard syncs the required values over to the variables on the ESP32 which are then sent over to the other systems using I2C. The subsystems can also send back telemetry to Thingsboard by setting the variables on the ESP32 like the 'curTemp' which is the current temperature reading from the Nucleo board. With this design, the purpose of communicating data over the cloud and creating a GUI to control the bioreactor systems is achieved. The implementation of the system proved to be reliable and quick after rigorous testing.

Overall System Integration and Summary

All the subsystems are integrated into one efficient system that allows the user to interact with the UI, on Thingsboard, from any place in the world with just the help of the internet. This is the block diagram of the entire system:



Once the user interacts with the UI, the data reaches the three subsystems through two microprocessors, the ESP32 and the Nucleo. The subsystems then act as per the commands received from the Nucleo. As the factors in the subsystems change, the data is sent back to the Thingsboard, through the same two microprocessors, which then updates the UI with the real-time data received by the ESP32.

When critically evaluating the heating subsystem, the team met a problem that the temperature increased too slowly, which may be insufficient for creating a suitable environment for the vaccine to grow. The root of this problem was that the PWM signal being sent to the MOSFET controlling the heater was only set to half (125) of its full potential value (255) in order to not reach or exceed the maximum power rating of the heater (30 W). This problem can be overcome by increasing the PWM to a higher value, allowing the MOSFET to pass a larger current through the heater. In addition to that, the heater itself can be switched to one with a higher maximum power rating, so that the full value of the PWM signal can be used, allowing the MOSFET to pass the largest possible current through the heater and increasing the rate of heating to its maximum value. “Selecting elements with superior thermal conductivity and quicker response times is key. “[6] These improvements can sharply cut down the time needed to hit desired heat levels, enhancing productivity overall.

During the testing process, the team also found out that there was too much load on the ESP32, which resulted in the ESP32 heating up. The issue turned out to be a wrong wire connection between the ESP32 and Nucleo which led to higher voltage than required flowing through the ESP32 which was causing it to heat up. To resolve this issue, the team disconnected the ESP32 and Nucleo from the system, rewired the entire system, and made sure the connections were correctly done.

The final system was flawless and was able to meet the requirements. The user was able to control the three subsystems through the UI and was able to see real-time data received from the three subsystems which meant that the integration of the entire system was successful.

In conclusion, the aim of this project was to build a working bioreactor that also allows for remote handling to produce vaccines that will help reduce the spread of TB in Uganda. The project has succeeded in doing this by building a working prototype with the aforementioned design that fulfils the requirements adequately.

References

[1] M. A. Horwitz, G. Harth, B. J. Dillon, and S. Maslesa-Galic', "Recombinant bacillus calmette-guerin (BCG) vaccines expressing the mycobacterium tuberculosis 30-KDA major secretory protein induce greater protective immunity against tuberculosis than conventional BCG vaccines in a highly susceptible animal model," Proceedings of the National Academy of Sciences of the United States of America, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC17665/ (accessed Dec. 13, 2023).

[2] Wang, R. (2023). *Connected Bioreactor Design Specification*. Lecture.

[3] Ardunio.cc, "A Guide to Arduino & the I2C Protocol (Two Wire) | Arduino Documentation," *docs.arduino.cc*. https://docs.arduino.cc/learn/communication/wire (accessed Dec. 13, 2023).

[4] Prernaajitg, "I2C Communication Protocol," *GeeksforGeeks*, Feb. 01, 2021. https://www.geeksforgeeks.org/i2c-communication-protocol/ (accessed Dec. 13, 2023).

[5] "What is MQTT," Amazon, https://aws.amazon.com/what-is/mqtt/#:~:text=MQTT%20is%20a%20standards%2Dbased,constrained%20network%20with%20limited%20bandwidth

[6] Jones, "Efficient Heating Solutions," International Journal of Applied Engineering Research, 2021.

[7] Thingsboard, "How to connect esp32 Dev Kit V1 to ThingsBoard?," ThingsBoard, https://thingsboard.io/docs/pe/devices-library/esp32-dev-kit-v1/ (accessed Dec. 13, 2023).

Appendices

Appendix A: ESP32 Code Snippets

A1:

```
// Settings for temperature
constexpr uint16_t SET_TEMP_MAX_C = 350U;
constexpr uint16_t SET_TEMP_MIN_C = 150U;
volatile uint16_t setTemp = 250U;

constexpr uint16_t CUR_TEMP_MAX_C = 500U;
constexpr uint16_t CUR_TEMP_MIN_C = 0U;
volatile uint16_t curTemp = 250U;
// Settings for PH
constexpr uint16_t SET_PH_MAX = 90U;
constexpr uint16_t SET_PH_MIN = 20U;
volatile uint16_t setPH = 70U;

constexpr uint16_t CUR_PH_MAX = 130U;
constexpr uint16_t CUR_PH_MIN = 0U;
volatile uint16_t curPH = 70U;
// Settings for Stir
constexpr uint16_t SET_STIR_MAX = 1200U;
constexpr uint16_t SET_STIR_MIN = 0U;
volatile uint16_t setStir = 400U;

constexpr uint16_t CUR_STIR_MAX = 1200U;
constexpr uint16_t CUR_STIR_MIN = 0U;
volatile uint16_t curStir = 400U;
```

```
        if (newTemp >= SET_TEMP_MIN_C && newTemp <= SET_TEMP_MAX_C) {
          setTemp = newTemp;
          Serial.print("Desired temperature set to: ");
          Serial.println(newTemp);
        }
```

A2:

```
const Shared_Attribute_Callback attributes_callback(&processSharedAttributes, SHARED_ATTRIBUTES_LIST.cbegin(), SHARED_ATTRIBUTES_LIST.cend());
const Attribute_Request_Callback attribute_shared_request_callback(&processSharedAttributes, SHARED_ATTRIBUTES_LIST.cbegin(), SHARED_ATTRIBUTES_LIST.cend());
```

Appendix B: ESP32 Complete Code

Adapted from Thingsboard Documentation [7]

```
#define THINGSBOARD_ENABLE_PROGMEM 0
#include <WiFi.h>
#include <Arduino_MQTT_Client.h>
#include <ThingsBoard.h>
#include <Wire.h>

// Wires for ESP to Nucleo
#define SLAVE_ADDR 9
#define I2C_SDA 21
#define I2C_SCL 22

constexpr char WIFI_SSID[] = "Ediz IPhone";
constexpr char WIFI_PASSWORD[] = "12345678";

constexpr char TOKEN[] = "edizESP";
constexpr char THINGSBOARD_SERVER[] = "thingsboard.cloud";

constexpr uint16_t THINGSBOARD_PORT = 1883U;
constexpr uint32_t MAX_MESSAGE_SIZE = 1024U;

constexpr uint32_t SERIAL_DEBUG_BAUD = 115200U;


// Initialize underlying clients
WiFiClient wifiClient;
Arduino_MQTT_Client mqttClient(wifiClient);
ThingsBoard tb(mqttClient, MAX_MESSAGE_SIZE);

// Attribute names for attribute request and attribute updates functionality
constexpr char setTempAttr[] = "setTemp";
constexpr char setPHAttr[] = "setPH";
constexpr char setStirAttr[] = "setStir";

volatile bool attributesChanged = false;
uint32_t previousAttributeChange;

// Settings for temperature
constexpr uint16_t SET_TEMP_MAX_C = 350U;
constexpr uint16_t SET_TEMP_MIN_C = 150U;
volatile uint16_t setTemp = 250U;

constexpr uint16_t CUR_TEMP_MAX_C = 500U;
constexpr uint16_t CUR_TEMP_MIN_C = 0U;
volatile uint16_t curTemp = 250U;
// Settings for PH
constexpr uint16_t SET_PH_MAX = 90U;
constexpr uint16_t SET_PH_MIN = 20U;
```

```
volatile uint16_t setPH = 70U;

constexpr uint16_t CUR_PH_MAX = 130U;
constexpr uint16_t CUR_PH_MIN = 0U;
volatile uint16_t curPH = 70U;
// Settings for Stir
constexpr uint16_t SET_STIR_MAX = 1200U;
constexpr uint16_t SET_STIR_MIN = 0U;
volatile uint16_t setStir = 400U;

constexpr uint16_t CUR_STIR_MAX = 1200U;
constexpr uint16_t CUR_STIR_MIN = 0U;
volatile uint16_t curStir = 400U;


// For telemetry
constexpr int16_t telemetrySendInterval = 2000U;
uint32_t previousDataSend;

// List of shared attributes for subscribing to their updates
constexpr std::array<const char *, 3U> SHARED_ATTRIBUTES_LIST = {
  setTempAttr,
  setPHAttr,
  setStirAttr
};


// WI-FI functions
void InitWiFi() {
  Serial.println("Connecting to AP ...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to AP");
}
const bool reconnect() {
  const wl_status_t status = WiFi.status();
  if (status == WL_CONNECTED) {
    return true;
  }
  InitWiFi();
  return true;
}


// Thingsboard functions

/// @brief Update callback that will be called as soon as one of the provided shared attributes changes value,
/// if none are provided we subscribe to any shared attribute change instead
/// @param data Data containing the shared attributes that were changed and their current value
```

```
void processSharedAttributes(const Shared_Attribute_Data &data) {
  for (auto it = data.begin(); it != data.end(); ++it) {
    if (strcmp(it->key().c_str(), setTempAttr) == 0) {
      const uint16_t newTemp = (int)(it->value().as<float>() * 10);
      if (newTemp >= SET_TEMP_MIN_C && newTemp <= SET_TEMP_MAX_C) {
        setTemp = newTemp;
        Serial.print("Desired temperature set to: ");
        Serial.println(newTemp);
      }
    }
    if (strcmp(it->key().c_str(), setPHAttr) == 0) {
      const uint16_t newPH = (int)(it->value().as<float>() * 10);
      if (newPH >= SET_PH_MIN && newPH <= SET_PH_MAX) {
        setPH = newPH;
        Serial.print("Desired PH set to: ");
        Serial.println(newPH);
      }
    }
    if (strcmp(it->key().c_str(), setStirAttr) == 0) {
      const uint16_t newStir = it->value().as<uint16_t>();
      if (newStir >= SET_STIR_MIN && newStir <= SET_STIR_MAX) {
        setStir = newStir;
        Serial.print("Desired stirring speed set to: ");
        Serial.println(newStir);
      }
    }
  }
  attributesChanged = true;
}

void processClientAttributes(const Shared_Attribute_Data &data) {
  for (auto it = data.begin(); it != data.end(); ++it) {
  }
}

const Shared_Attribute_Callback attributes_callback(&processSharedAttributes,
SHARED_ATTRIBUTES_LIST.cbegin(), SHARED_ATTRIBUTES_LIST.cend());
const Attribute_Request_Callback attribute_shared_request_callback(&processSharedAttributes,
SHARED_ATTRIBUTES_LIST.cbegin(), SHARED_ATTRIBUTES_LIST.cend());

void setup() {
  // Initalize serial connection for debugging
  Serial.begin(SERIAL_DEBUG_BAUD);


  delay(1000);
  InitWiFi();

  // // Initialise I2C
  Wire.begin(I2C_SDA, I2C_SCL);
  Serial.begin(115200);
```

```
}

void loop() {
  delay(10);

  if (!reconnect()) {
    return;
  }

  if (!tb.connected()) {
    // Connect to the ThingsBoard
    Serial.print("Connecting to: ");
    Serial.print(THINGSBOARD_SERVER);
    Serial.print(" with token ");
    Serial.println(TOKEN);
    if (!tb.connect(THINGSBOARD_SERVER, TOKEN, THINGSBOARD_PORT)) {
      Serial.println("Failed to connect");
      return;
    }
    // Sending a MAC address as an attribute
    tb.sendAttributeData("macAddress", WiFi.macAddress().c_str());

    Serial.println("Subscribing for RPC...");

    if (!tb.Shared_Attributes_Subscribe(attributes_callback)) {
      Serial.println("Failed to subscribe for shared attribute updates");
      return;
    }

    Serial.println("Subscribe done");

    // Request current states of shared attributes
    if (!tb.Shared_Attributes_Request(attribute_shared_request_callback)) {
      Serial.println("Failed to request for shared attributes");
      return;
    }
  }

  if (attributesChanged) {
    attributesChanged = false;
    previousAttributeChange = millis();

    tb.sendAttributeData(setTempAttr, setTemp / 10);
    tb.sendAttributeData(setPHAttr, setPH / 10);
    tb.sendAttributeData(setStirAttr, setStir);
  }


  // Sending telemetry every telemetrySendInterval time
  if (millis() - previousDataSend > telemetrySendInterval) {
    previousDataSend = millis();
```

```
  tb.sendTelemetryData("currentTemperature", curTemp / 10);
  tb.sendTelemetryData("currentPH", curPH / 10);
  tb.sendTelemetryData("currentStirring", curStir);

  tb.sendAttributeData("rssi", WiFi.RSSI());
  tb.sendAttributeData("channel", WiFi.channel());
  tb.sendAttributeData("bssid", WiFi.BSSIDstr().c_str());
  tb.sendAttributeData("localIp", WiFi.localIP().toString().c_str());
  tb.sendAttributeData("ssid", WiFi.SSID().c_str());
}

tb.loop();

// Creating a string to send
String DataSend = "";
DataSend += setTemp;
DataSend += setPH;
if (setStir < 1000){
  DataSend += "0";
  DataSend += setStir;
}else{
  DataSend += setStir;
}
DataSend += " ";

Wire.requestFrom(SLAVE_ADDR, 10);
String received_string = "";
while (Wire.available() > 1) {
  char c = Wire.read();
  received_string += c;
}

Serial.println(received_string);
curTemp = received_string.substring(0, 3).toInt();
curPH = received_string.substring(3, 2).toInt();
curStir = received_string.substring(5, 4).toInt();

Wire.beginTransmission(SLAVE_ADDR);
Wire.write(DataSend.c_str());
Wire.endTransmission();

delay(500);

}
```

Appendix C: pH subsystem code

```
//pH subsystem variables
double dtpH = 20;
double kppH = 10;
double kipH = 0.351;
double IntegpH = 0;
double pHe;
double prevpHe;
double ProppH;
double OutputpH;
double setpointpH;
long currTimepH;
long prevTimepH = 0;
int pHsensorval;
int pHpumpinterval;
int pumpTimer;
double pH;

#define pHpumpA 5
#define pHpumpB 6
#define pHsensorpin A1

void setup() {
	pinMode(pHpumpA, OUTPUT);
	pinMode(pHpumpB, OUTPUT);
	pinMode(pHsensorpin, INPUT);
}

void loop() {
	currTimepH = millis();
	setpointpH = 10;

	//pH subsystem code
	 if ((currTimepH - prevTimepH) >= dtpH * 1000) {
	  pHsensorval = analogRead(pHsensorpin);
	  pH = pHsensorval * 0.00709 + 2.879;
	  pHe = setpointpH - pH;
	  ProppH = (kppH * pHe);
	  IntegpH = IntegpH + kipH * 0.5 * (prevpHe + pHe) * dtpH;
	  OutputpH = ProppH + IntegpH;
	  if (pHe > 0) {
	   pHpumpinterval = abs(round(OutputpH * 416.67));
	   pumpTimer = millis();
	   if (pumpTimer != 0 && millis() - pumpTimer >= pHpumpinterval) {
	    analogWrite(pHpumpB, 0);
	   }
	   else {
	    analogWrite(pHpumpB, 250);
	   }
	  }
	  else if (pHe < 0) {
	   pHpumpinterval = abs(round(OutputpH * 416.67));
```

```
    pumpTimer = millis();
    if (pumpTimer != 0 && millis() - pumpTimer >= pHpumpinterval) {
     analogWrite(pHpumpA, 0);
    }
    else {
     analogWrite(pHpumpA, 250);
    }
   }
   else{
    analogWrite(pHpumpB, 0);
    analogWrite(pHpumpA, 0);
   }
   prevTimepH = currTimepH;
 }
}
```

Appendix D: Stirring subsystem code

```
//Stirring subsystem variables
double dtStir = 0.01;
double kpStir = 10;
double kiStir = 0.351;
double IntegStir = 0;
double RPMe;
double prevRPMe = 0.0;
double PropStir;
double OutputStir;
double setpointStir;
long currTimeStir;
long prevTimeStir = 0;
double RPM;
float freq;
int controlStir = 1;
long pulseT, prevpulseT;
int PowerStir;

#define speedsensorpin 2
#define motorpin 11

void setup() {
pinMode(speedsensorpin, INPUT);
 pinMode(motorpin, OUTPUT);
 pinMode(13, OUTPUT);
 attachInterrupt(digitalPinToInterrupt(speedsensorpin), freqcount, CHANGE);
 Serial.begin(250000); // Use serial monitor to monitor speed, motor voltage etc
 analogWriteResolution(10); // 10-bit PWM - TCCR1A = 0b00000011; for Uno/Nano
 analogWriteFrequency(8000); //8 kHz PWM -  TCCR1B = 0b00000001; for Uno/Nano
 analogWrite(motorpin, 0);
}

void loop() {
currTimeStir = micros();
setpointStir = 800;
```

```
//Stirring subsystem code
 if ((currTimeStir - prevTimeStir) >= dtStir*1000000) {
  RPM = freq*15; //RPM
  if (currTimeStir-pulseT>5e6) {
   RPM = 0; // Indicate zero measured speed if no pulses detected for 0.5 s
  }
  RPMe = setpointStir - RPM;
  PropStir = (kpStir * RPMe);
  IntegStir = IntegStir + kiStir * 0.5 * (prevRPMe + RPMe) * dtStir * controlStir;
  OutputStir = PropStir + IntegStir;
  PowerStir = round(OutputStir * (1020 / 12000));
  if (PowerStir < 0) {
   PowerStir = 0;
   controlStir = 0;
  }
  else if (PowerStir > 1020) {
   PowerStir = 1020;
   controlStir = 0;
  }
  else {
   PowerStir = PowerStir;
   controlStir = 1;
  }
  analogWrite(motorpin,200);
  prevTimeStir = currTimeStir;
 }
 Serial.println(T);
 Serial.println(pH);
 Serial.println(RPM);
}

void freqcount() {
 pulseT = micros();
 if(pulseT-prevpulseT>9375){
 freq = 5e5/float(pulseT-prevpulseT)+freq/2;}
 prevpulseT = pulseT;
 digitalWrite(13,!digitalRead(13));
}
```

Appendix E: Heating subsystem code

```
//Heating subsystem variables
double dtTemp = 0.1;
double kpTemp = 10;
double kiTemp = 0.351;
double IntegTemp = 0;
double Te;
double prevTe = 0.0;
double PropTemp;
double OutputTemp;
double setpointTemp;
long currTimeTemp;
```

```
long prevTimeTemp = 0;
double T;
int PowerTemp;
int controlTemp = 1;

#define tempsensorpin A0
#define Heaterpin 9

void setup() {
        pinMode(tempsensorpin, INPUT);
        pinMode(Heaterpin, OUTPUT);
}

void loop() {
currTimeTemp = millis();

setpointTemp = 27;

//Heating subsystem code
 if ((currTimeTemp - prevTimeTemp) >= dtTemp*1000) {
   int tempsensorval = analogRead(tempsensorpin);
   T = tempsensorval*0.0842 - 19.3675;
   Te = setpointTemp - T;
   PropTemp = (kpTemp * Te);
   IntegTemp = IntegTemp + kiTemp * 0.5 * (prevTe + Te) * dtTemp * controlTemp;
   OutputTemp = PropTemp + IntegTemp;
   PowerTemp = round(OutputTemp * (125 / 30));
   if (PowerTemp < 0) {
    PowerTemp = 0; controlTemp = 0;
   }
   else if (PowerTemp > 25) {
    PowerTemp = 25; controlTemp = 0;
   }
   else {
    PowerTemp = PowerTemp;
        controlTemp = 1;
   }
   analogWrite(Heaterpin,PowerTemp);
   prevTimeTemp = currTimeTemp;
 }
}
```

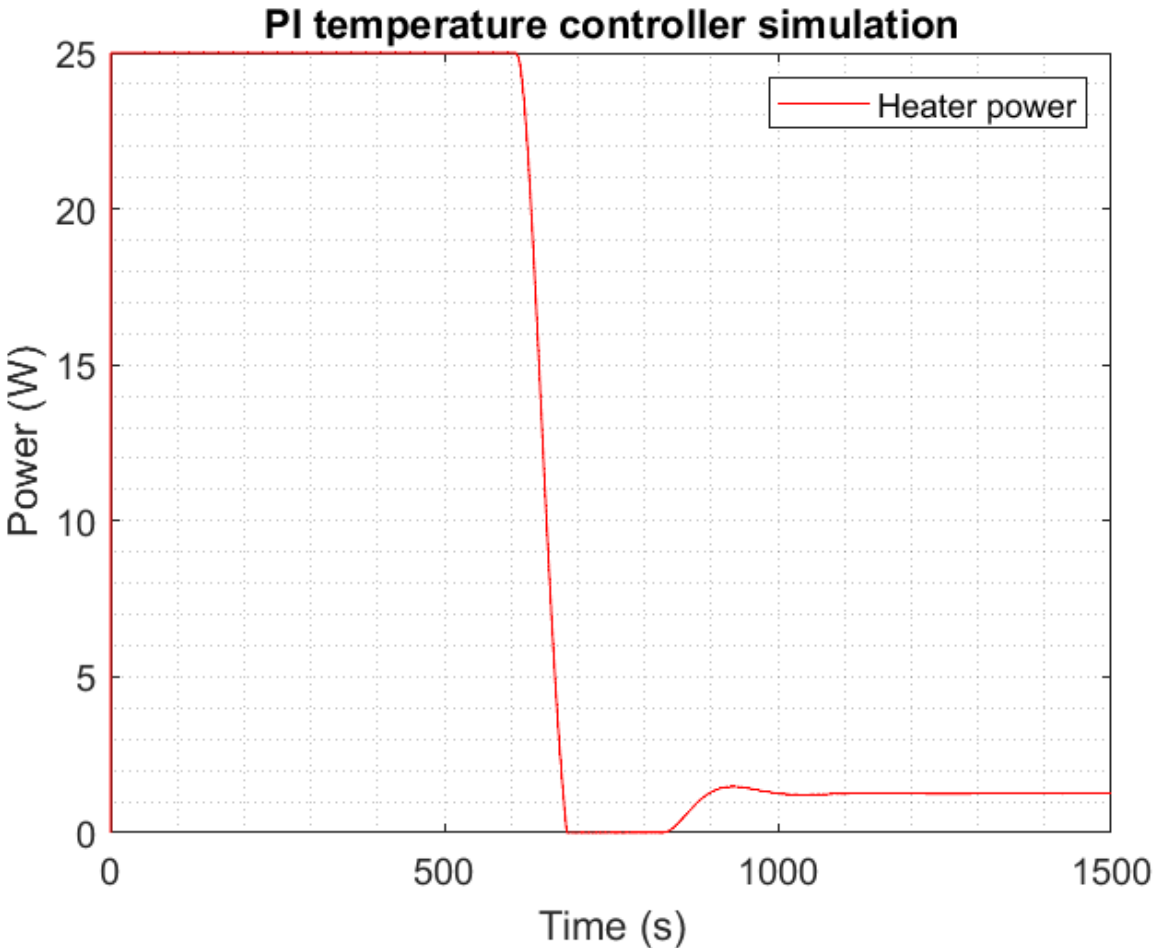
Appendix F: MATLAB simulation of heating subsystem.

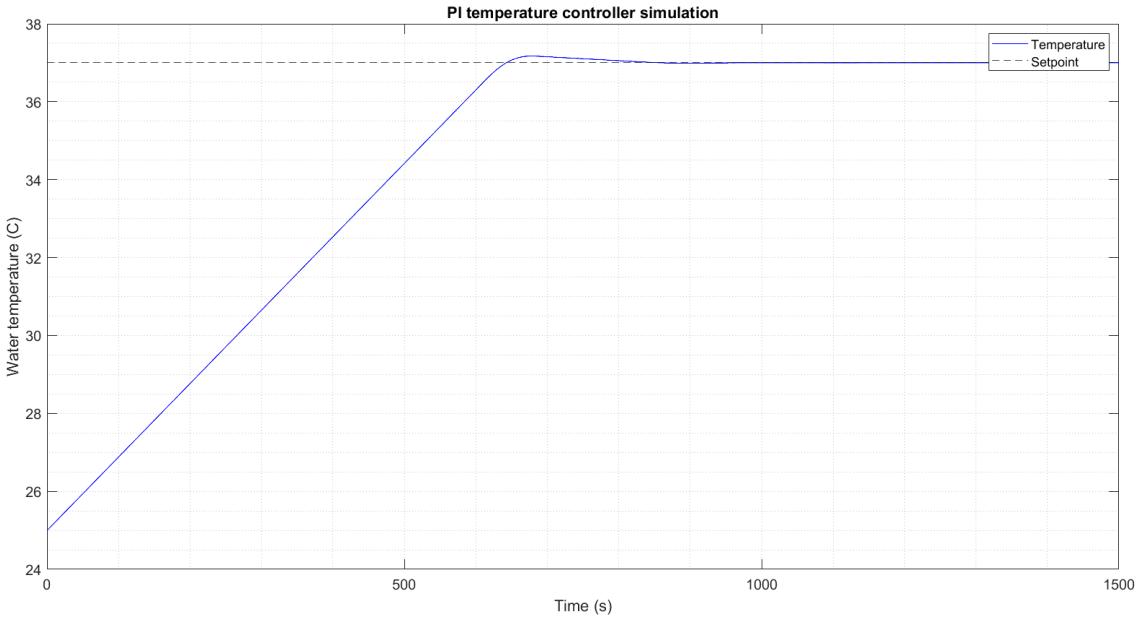
Code:

```
dt = 0.1;
m = 0.3; % m = 0.3kg
c = 4200; % c = 4200 J/kgC
kp = 10;
ki = 0.351;
Integ(1) = 0;
T(1) = 25;
t = 0:dt:1500;
pts = length(t);
```

```
Power(1) = 0;
control = 1;
setpoint = 37;
deltaT(1) = 0;
Output(1) = 0;
Te(1) = setpoint - T(1);
for i = 2:(pts)
T(i) = T(i-1) + deltaT(i-1)-0.0001;
Te(i) = setpoint - T(i);
Prop(i) = kp*Te(i);
Integ(i) = Integ(i-1) + ki*(0.5)*(Te(i-1)+Te(i))*dt*control;
Output(i) = Prop(i) + Integ(i);
Power(i) = Output(i)*(125/30);
if (Power(i) < 0)
Power(i) = 0;
control = 0;
elseif (Power(i) > 25)
Power(i) = 25;
control = 0;
else
Power(i) = Power(i);
control = 1;
end
deltaT(i) = (Power(i)*dt)/(m*c); % change in temperature
end
% Plot result
figure
plot(t,T,'-b')
hold on
yline(setpoint,'--k')
grid minor
title('PI temperature controller simulation')
xlabel('Time (s)')
ylabel('Water temperature (C)')
legend('Temperature','Setpoint')
set(gca,'fontsize',12)
figure(2)
plot(t,Power,'-r')
grid minor
title('PI temperature controller simulation')
xlabel('Time (s)')
ylabel('Water temperature (C)')
legend('Heater power')
set(gca,'fontsize',12)
```

Results:


PI temperature controller simulation
Heater power
Power (W)
Time (s)
0
5
10
15
20
25
0
500
1000
1500


PI temperature controller simulation
Temperature
Setpoint
Water temperature (C)
Time (s)
24
26
28
30
32
34
36
38
0
500
1000
1500