Challenge 2: Bioreactor

CS-EEE Specialist Team Final Report

Authors:

Mathis Laurent - pH Hardware and Report Lead

Ibraheem Siddiqui - pH Software

Anthony Hipperson - Heating Hardware

Sameer Kurbanov - Stirring Software, Website, and Data Logging Software

Brandon Wong - Stirring Hardware

Hao Xuan Ng - Connectivity Hardware, Hardware Integration Lead

Ioan Steffan Thomas - Connectivity Software, Software Integration Lead

Ceto Kim - Heating Software

1. Introduction

Section Authors: Mathis Laurent, Hao Xuan Ng

This project focuses on the design, development, and integration of hardware, software, and communication interfaces to enable real-time monitoring and precise control of parameters in a small-scale bioreactor system. The bioreactor is designed to simulate and optimize conditions for biological processes, particularly the production of the BCG vaccine, which has a 70–80% success rate in preventing tuberculosis. In Uganda, tuberculosis remains a significant public health issue, affecting approximately 94,000 people in 2024, with 3,700 deaths reported that year. The bioreactor aims to contribute to a large-scale vaccination program by providing controlled conditions for vaccine production.

At a **high level**, the system architecture is centered around an **Arduino Uno**, which acts as the "motherboard" for managing communication between the subsystems. Each subsystem, responsible for a specific parameter (pH, temperature, stirring), is implemented on a breadboard and interfaces with the Arduino through sensors and actuators. The **ESP32 module** plays a critical role in enabling **connectivity and data handling** by acting as the intermediary between the Arduino and the user interface, which is accessed via a web-based dashboard. Additionally, an intelligent adjustor connects the heating and pH subsystems, dynamically adjusting pH levels based on temperature changes to enhance precision in the culture medium.

The system operates as follows:

- Each subsystem (pH, heating, stirring) measures current values through sensors connected to the Arduino Uno. These values are processed and sent to the ESP32 over a serial interface (I^2C).
- The ESP32 acts as a gateway, encoding data in JSON format and transmitting it to a web server using WebSocket communication. This enables **real-time data visualization** and control via an interactive GUI.
- The GUI allows the user to set target values for each parameter. These targets are sent back to the ESP32, which communicates them to the Arduino Uno.
- The Arduino compares the target values with the current values and computes the necessary adjustments. These corrections are implemented by activating actuators, such as motors, heaters, and pumps, in each subsystem.

The system we are creating cosnsists of 2 microcontrollers, the Arduino and ESP32. The Arduino controls the Heating, pH and stirring subsystems. Information from the 3 major subsystems is collected by the Arduino, and then transferred to the ESP32, which then transmit this information to the cloud, where the data is logged and visualized on graphs in real time. This represents the Connectivity subsystems.

The **system specifications** emphasize modularity and scalability. Each subsystem is designed to function independently while maintaining seamless integration with the central control system. The ESP32 ensures that connectivity between the hardware and software components is robust, enabling **remote monitoring and control**.

The functional diagram highlights the flow of data and control signals between components. It shows how the Arduino Uno retrieves sensor data, calculates corrective actions, and interfaces with the ESP32 for data logging and visualization. The GUI serves as a unified platform for system operation, ensuring that operators have real-time insight into the bioreactor's status.

This high-level design integrates intelligent adjustments, precise control, and remote accessibility, ensuring efficient communication across all subsystems. The project addresses the

challenges of managing interconnected hardware and software in resource-constrained environments, supporting a scalable and effective bioreactor system for vaccine production.

soul red green

ID	Requirement	Design Parameters	Components	Interfaces
1	Measure RPM of motor	0 – 1300 RPM	Arduino Uno, Optical Sensor	Pulses/interrupts
2	Measure pH of reservoir	3 – 7 pH	Arduino Uno, pH Sensor	Analogue input
3	Measure temperature of solution	25 – 35 °C	Arduino Uno, Thermistor	Analogue Input
4	Adjust RPM of motor	0 – 1300 RPM	Arduino Uno, Motor	Digital Output
5	Adjust pH of reservoir	3 – 7 pH	Arduino Uno, 2x Peristaltic Pumps	Digital output
6	Adjust temperature of solution	25 – 35 °C	Arduino Uno, MOSFET Transistor	Digital Output
7	Send pH, temperature, and RPM values to Arduino	3.000–7.000 pH, 0–1300 RPM, 25.00–35.00 °C	Arduino Uno, ESP32, Level Shifter	I^2C
8	Send current pH, temperature, and RPM values	3.000–7.000 pH, 0–1300 RPM, 25.00–35.00 °C	Webserver, browser and website files	HTTP WebSocket
9	Get target pH, temperature, and RPM values	3.000–7.000 pH, 0–1300 RPM, 25.00–35.00 °C	Webserver, browser and website files	HTTP WebSocket and JSON
10	Internet connected dashboard to display graphs and get targets	3.000-7.000 pH 0-1300 RPM, 20.00-35.00 °C	Webserver/host Browser on computer	HTTP WebSocket and JSON

2. Subsystem Designs

2.1. Heating system

Section Authors: Anthony Hipperson, Ceto Kim

Heating Subsystem Overview

The heating subsystem is designed to maintain a precise and adjustable water temperature within the bioreactor, ensuring optimal conditions for biological processes. The main components of this subsystem include a thermistor, MOSFET, power supply, and Arduino Uno. The thermistor leverages temperature-dependent resistance properties to measure the system temperature accurately. A MOSFET (Metal-Oxide-Semiconductor Field-Effect Transistor) acts as a switch to regulate the power supplied to the heating element, providing precise control of the temperature. The system is powered by a 12V supply, ensuring adequate energy for both

the heater and other components, while the Arduino Uno generates Pulse Width Modulation (PWM) signals to adjust heating intensity based on feedback from the thermistor.

Operational Mechanism

The heating subsystem operates by maintaining a predefined water temperature set by the user, adjustable between 25°C and 35°C via Serial input. This temperature range is ideal for biological processes, preventing overheating or insufficient heating. The thermistor continuously measures the water temperature and sends this data to the Arduino Uno for processing. If the measured temperature is below the set point, the Arduino sends a PWM signal to the MOSFET, activating the heater. The PWM duty cycle is dynamically adjusted based on the difference between the current temperature and the set point, allowing efficient heat output control. The system incorporates safety measures to prevent overheating, ceasing heating once the temperature reaches the upper limit of the set range.

Performance and Precision

The subsystem was tested for its ability to maintain a temperature within ±0.5°C of the set point. Results demonstrated that the system reliably achieved this precision under steady-state conditions. Additionally, the control logic prevented significant overshooting or fluctuations, ensuring consistent heating suitable for sensitive biological cultures. The energy efficiency of the system was evident during prolonged operation, with minimal deviations from the target temperature.

Integration with Other Subsystems

Integration with the other subsystems was a critical aspect of the design. The heating subsystem works seamlessly with the stirring system, ensuring uniform temperature distribution across the bioreactor. Furthermore, it complements the pH control subsystem by maintaining consistent conditions that facilitate accurate pH adjustments. Together, the integrated systems ensure the bioreactor operates as a cohesive unit, with no interference between the subsystems, thereby maintaining stable and reliable performance essential for biological processes.

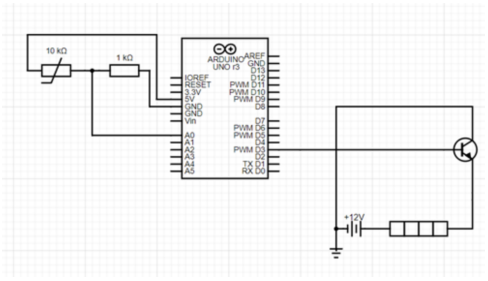

Figure 1: Heating system circuit overview.

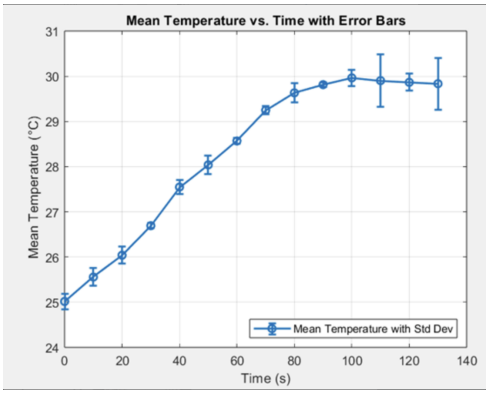

Figure 2: Behaviour of temperature versus time in the Bioreactor with standard deviation.

2.2. pH system

Section Authors: Mathis Laurent, Ibraheem Siddiqui

pH Subsystem Overview:

The pH subsystem automates the regulation of pH within the bioreactor, maintaining a target range of 3.0–7.0 with a precision of ±0.2. This system ensures a stable environment for bioreactor processes, using a pH sensor, Arduino Uno, two peristaltic pumps, and a breadboard circuit

for power and control.

pH Measurement and Signal Processing

The subsystem begins with the pH sensor, which measures the acidity or alkalinity of the bioreactor solution. Analog voltage signals from the sensor are sent to the Arduino Uno, where they are converted to pH values using Faraday's formula. The sensor connects to the Arduino through three wires: one for power (5V), one for ground (GND), and one for signal input (A0), ensuring accurate pH readings.

Initial Control Approach: PI Logic

Initially, PI (Proportional-Integral) control was used in the software for precise pump activation. The system sampled 40 pH values at 20-millisecond intervals and calculated their average to stabilize readings. When the pH fell outside the target range, the Arduino activated the appropriate pump using PWM (Pulse Width Modulation), allowing gradual adjustments. However, the system faced instability due to the logarithmic nature of pH and fluctuations near the target range.

Updated Control Approach: Bang-Bang Logic

To overcome these issues, the control system was updated to use a bang-bang approach, simplifying the logic. Pumps now activate in short bursts with a two-second delay to allow thorough mixing. This approach avoids overshooting and maintains stable operation, making it better suited for the non-linear pH dynamics.

Role of the Arduino and Control Signals

The Arduino Uno acts as the central controller, determining whether to activate the acid or alkaline pump based on sensor readings. If the pH is too high, the acid pump is activated; if too low, the alkaline pump adjusts the pH. Control signals are sent via digital pins 12 and 13, which connect to transistors on the breadboard.

Breadboard and Pump Control

The breadboard serves as the interface between the Arduino and pumps, powered by an external 6V supply. Each pump connects to the breadboard through a transistor, which switches current flow. Transistors ensure the pumps are powered only when needed and shield the Arduino from high currents. When activated, current flows from the 6V supply, through the pump, and into the transistor, enabling operation.

Performance and Validation

Testing confirmed the subsystem could maintain pH within ± 0.2 of the target range. Buffer solutions validated the sensor's accuracy, while the pumps effectively adjusted pH levels. The breadboard and transistors ensured reliable switching and control, and the bang-bang logic provided stable adjustments without overshooting.

Integration with Other Subsystems

The subsystem integrates seamlessly with the stirring and heating subsystems. The stirring subsystem ensures even distribution of acid or base during adjustments, preventing localized imbalances. The heating subsystem maintains consistent temperature, supporting accurate pH readings. Additionally, implementing an intelligent adjustor between the heating and pH subsystems could further enhance precision by dynamically adjusting pH based on temperature variations. Though minimal, this improvement would benefit sensitive biological processes significantly.

Equation used for pH calculation in the Arduino code:

$$pH(X) = pH(S) + \frac{(E_S - E_X)F}{RT\ln(10)} \tag{1}$$

See constants and variable in appendix

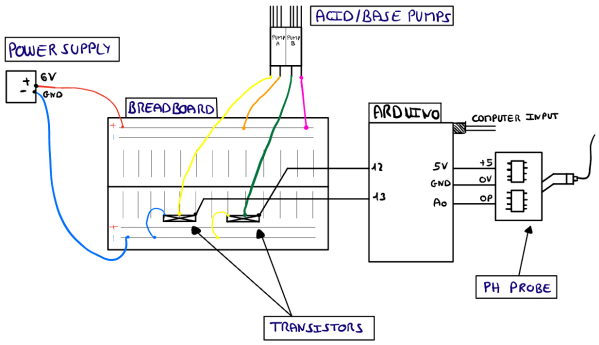

Figure 3: Behaviour of temperature versus time in the Bioreactor.

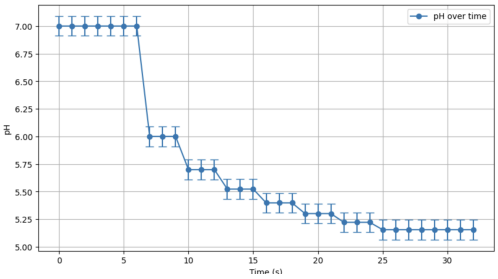

Figure 4: Graph showing the pH of the substrate against time, with error bars representing the standard deviation. The initial pH is 7, with a target of 5 and the pump stops activating when the pH is less than 0.2 away from the target pH.

2.3. Stirring system

Section Authors: Sameer Kurbanov, Brandon Wong

Purpose of the Stirring System

The purpose of the stirring system is to ensure thorough mixing of the culture medium (e.g., water) used to suspend yeast, thereby facilitating an even distribution of nutrients, gases, and heat. This uniformity creates an optimal environment for yeast growth, reducing the risk of cell death and maximizing vaccine production efficiency. Stirring also enhances pH adjustments by rapidly distributing alkali or acid introduced by the pH subsystem, improving the effectiveness and speed of neutralization reactions. Additionally, the stirring subsystem promotes temperature uniformity within the medium by reducing localized hot and cold spots, optimizing heat transfer between the heater (from the heating subsystem) and the culture medium. This process improves heat transfer efficiency, as described by Fourier's Law [21], which states that the rate of heat transfer through a material is proportional to the temperature gradient between two points.

System Components and Control Mechanism

The system integrates a motor-driven stirrer shaft to create turbulence in the medium. A microcontroller generates PWM (pulse-width modulation) signals in the range of [0, 1023] (the usual range for PWM signals is [0, 255], however one of the Arduino pins was configured to support 10-bit PWM signals for the sake of increased accuracy) to adjust motor speed via a MOSFET, while a Hall sensor provides feedback through a PI control loop to maintain consistent stirring performance. Together, these components ensure precise and reliable mixing, which is critical for the bioreactor's operation.

Importance of pH Regulation

Extreme pH conditions can denature proteins critical to yeast growth, such as glycolytic enzymes like pyruvate kinase (which become ineffective under adverse conditions). This enzyme is essential for energy production during glycolysis [22] (the first stage of aerobic respiration). Since yeast survival depends on energy derived from respiration, maintaining optimal pH is paramount. Additionally, aerobic respiration in yeast produces CO_2, which reduces pH within the culture medium [23]. This further highlights the stirring subsystem's role in aiding pH regulation.

PI Controller for Motor Regulation

The stirring subsystem is controlled by an Arduino Uno programmed in C++, using a PI controller for precise and responsive motor speed regulation. The controller ensures robust performance despite disturbances like variations in viscosity. Motor speed is adjusted through PWM signals, with the proportional component of the controller adjusting the PWM signal in response to the difference between current and target RPM. This mechanism allows both macro and micro adjustments, ensuring the system remains stable under varying conditions, without overshooting or undershooting the target RPM [24, 25].

Role of the Integral Component

The integral component of the PI controller enhances stability by accumulating the difference between the target and current RPM over time, improving responsiveness to steady-state errors caused by factors like changes in viscosity or heat. While a P controller alone may lead to higher average RPM errors, a PI controller balances speed and precision, making it more suitable for yeast growth where consistent RPM is critical [26, 27]. Moreover, the PI controller is preferred over a bang-bang control system, which oscillates between extreme states, leading to unwanted fluctuations [28]. The PI controller's smoother adjustments are better suited to the bioreactor's operational needs.

PWM Signal Calculation

When the Arduino calculates the PWM signal for the motor, it first computes the RPM error (difference between the current and target RPM). Next, it calculates the integral error (sum of RPM errors over the past three seconds). These values are used to compute the proportional and integral terms of the PI formula, as shown in Equations 2, 3, and 4.

$$V_m = 200 \cdot (K_p + K_i) \tag{2}$$

$$K_p = \frac{2 \cdot \zeta \cdot \omega_n}{\omega_n - 1} \cdot \frac{1}{K_v} \cdot rpmError \tag{3}$$

$$K_i = \frac{\omega_n^2}{K_v \cdot \omega_n} \cdot integralError \tag{4}$$

In these equations:

- ζ represents the damping ratio (set to 1 for critical damping).
- ω_n is the natural frequency, motor response time to changes in voltage.
- K_v is the voltage constant, representing RPM per volt of input.
- V_m represents the PWM signal magnitude, determining the control signal sent to the motor to adjust its speed.
- K_p is the proportional term of the PI controller, which calculates a control response based on the current error between the target RPM and the actual RPM. This term helps achieve immediate adjustments to reduce the error.
- K_i is the integral term of the PI controller, which accounts for the accumulated error (integral error) over time.

All constants were calculated/calibrated through trial and error testing. These constants are key for the PI control system and ensure that the PWM signal is scaled for a range of [0, 1023], suitable for precise motor adjustments [24, 25].

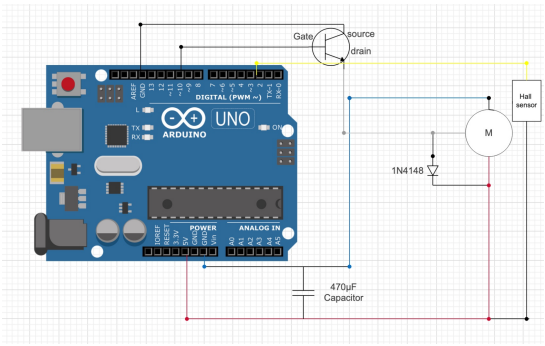

Figure 5: Circuit diagram of stirring system.

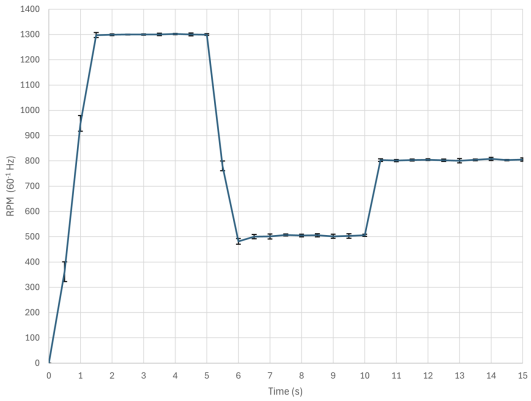

Figure 6: Graph showing the mean RPM after 5 trials of the stirrer against time, with error bars representing the standard deviation. The initial RPM is 0, with a target of 1300 RPM. After 5 seconds, the target RPM is changed to 500, and after another 5 seconds, it is changed to 800. The standard deviation is extremely low, making the error bars barely visible, indicating high precision and stability. The system stabilizes effectively after reaching each target RPM, taking only about 0.4-1.4 seconds to reach desired RPM depending on the intensity of the chang

2.4. Communication between Arduino Uno, ESP32 and Cloud

Section Author: Ioan Steffan Thomas, Mathis Laurent
Data Encoding:

The data is encoded as a maximum of 15 characters per transmission. The first 4 characters represent the pH value (a single unit followed by 3 decimal places). The next character is a comma. The following 4 characters represent the temperature value (2 digits followed by 2 decimal places). Another comma separates this value from the next 4 characters, which directly represent an integer representation of the RPM. The final character is a newline. For example, the string "4500,2750,500" represents pH 4.5, temperature 27.5°C, and RPM 500. For the pH and temperature values, only the integer sections are required, while only one digit is required for RPM.

The ESP32 and Arduino Uno communicate using the I^2C protocol [8]. The ESP32 acts as the controller (reader/writer), while the Arduino operates as a peripheral (sender/receiver). The Arduino Wire library [9] is used to establish the I^2C connection. Since the Arduino operates at 5V and the ESP32 operates at 3.3V, a level shifter is required to facilitate their connection.

The pin connections, routed through the level shifter, include the following: the SCL (clock) pin on the ESP is connected to the Arduino's SCL pin, the SDA (data line) pin on the ESP is connected to the Arduino's SDA pin, and the GND (ground) pin on both microcontrollers is connected. Data is sent from the ESP to the website every 100ms, while any changes made on the UI are sent back to the ESP.

The ESP connects to the Internet using WiFi, utilising either an enterprise access point

(EAP) or a standard wireless access point (WAP). A WebSocket is used to enable two-way communication between the ESP and the website. This is implemented using the WebSocket protocol RFC 6455 [10] and the Arduino WebSockets library [11]. The WebSocket port is set to 3000, and the host IP corresponds to the website's IP.

The data transmitted between the ESP and the website is encoded in JSON, following RFC 8259 [12]. The ArduinoJSON library [13] is used for encoding the payload, which includes the fields 'heat,' 'rpm,' and 'temp'. Each field is a 4-digit value encoded as float, float, and integer, respectively. Data transmission across the Internet is handled via HTTP, adhering to RFC 2616 [14].

Design Rationale:
The data encoding is limited to 15 bytes, allowing the I^2C protocol to be used effectively (within the 32-byte limit imposed by the Wire library). Using the ESP32 as the I^2C controller reduces the workload on the Arduino. The encoding format accommodates the required number of significant digits for each subsystem's supported performance levels and is designed to be simple and computationally efficient.

A 100ms delay is implemented to optimize resource consumption. Instead of transmitting data every loop cycle, data is sent at regular intervals, reducing the load on the Arduino. WiFi connectivity simplifies the system's setup and reduces the wiring needed, enabling flexibility in the bioreactor's placement. The WebSocket protocol was selected for its use of TCP/IP, guaranteeing reliable data delivery. HTTP is used for data transfer due to its widespread support, and JSON was chosen for its standardized format and the availability of libraries for encoding and decoding.

This approach ensures reliable, efficient, and flexible communication between the ESP32, Arduino, and the website, supporting the bioreactor's functionality while minimizing resource usage.

2.5. Data logging and visualization

Section Author: Ioan Steffan Thomas
The UI/Dashboard subsystem allows users to monitor and control the bioreactor through a web interface. Built with TypeScript [15], for type safety, and React [16] (using the Next.js framework [17]), the frontend ensures scalability and performance. Lucide-React [18] components are used for icons, while custom components (e.g., buttons) and Tailwind CSS [19] are employed for styling. Data is visualized using Recharts [20], which displays up to 240 data points in a dot-to-dot style with timestamps and exact values.
The dashboard includes input fields for users to set parameters (e.g., temperature, pH), which are sent to the ESP32 for control. The frontend communicates with the ESP32 and Arduino subsystems to display sensor data and adjust bioreactor settings in real time.

Design Rationale: TypeScript provides type safety, which aids system robustness. React's component-based structure allows for efficient UI development, and Tailwind CSS enables rapid styling. Recharts is utilized for dynamic data visualization, while Lucide-React ensures consistent iconography.

The UI is fully integrated with the ESP32 and Arduino, receiving data from the bioreactor and sending control commands. Testing includes unit, integration, and user tests to ensure functionality and performance.

3. Overall System Integration and Summary

The developed bioreactor control system successfully integrates multiple subsystems—pH regulation, temperature control, stirring, and data connectivity—ensuring optimal conditions for yeast growth and vaccine production. The system's performance aligns closely with the specified design parameters: maintaining stable RPM within a range of 500–1300 RPM, a temperature range of 25–35°C with ±0.5°C precision, and pH stability within a range of 3–7 with a tolerance of ±0.2.

Testing revealed that the integration of hardware (Arduino Uno, ESP32, sensors, and actuators) and software (PI control, bang-bang control, and real-time communication protocols) produced reliable results during isolated testing. The subsystems function cohesively in principle, with the stirring subsystem uniformly mixing the medium to support even nutrient and gas distribution, the heating subsystem maintaining consistent temperature across the medium, and the pH subsystem effectively adjusting acid or base levels to stabilize the environment.

The system also incorporates data visualization and control via a web interface, providing real-time feedback and enabling precise user interaction. This feature ensures accessibility and allows for dynamic adjustments to the bioreactor's parameters based on monitored data.

While individual components worked well during tests, an unexpected software error occurred during the final integration and demo. Despite thorough debugging attempts, the error persisted, preventing the system from functioning as intended during the demonstration. Refining the integration and identifying the root cause of this issue would be critical for achieving full operational reliability in future iterations.

Potential improvements include further calibration of sensors and optimization of control algorithms to enhance response times during parameter changes. Incorporating an intelligent adjustor to account for interdependencies between temperature and pH could also improve precision and reliability.

Overall, the bioreactor system demonstrates strong potential to reliably control the required conditions for biological processes and aligns with the overarching goal of facilitating efficient vaccine production in a controlled environment. Despite setbacks during the demo, the design shows promise for scalability and adaptability for use in larger manufacturing contexts with further refinement.

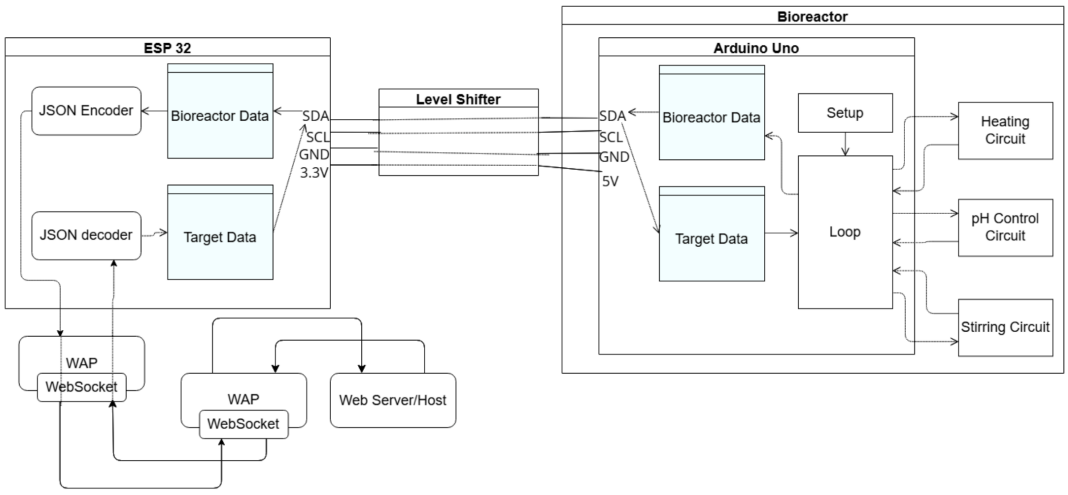

Figure 7: Overall design of the bioreactor, consisting of 2 microcontrollers and the 3 major subsystems.

4. Appendices

References

[1] U.S. Agency for International Development, "Uganda Tuberculosis Roadmap Overview, Fiscal Year 2024," 2024.

[2] "BCG Vaccine (TB vaccine)," Vaccine Knowledge Project. Accessed: Dec. 12, 2024. [Online]. Available: https://vaccineknowledge.ox.ac.uk/bcg-vaccineKey-vaccine-facts

[3] fmarquis, "Control Peristaltic Pump With TA7291P and an Arduino," https://www.instructables.com/Control-peristaltic-pump-with-TA7291P-and-an-Ardui/, Instructables, accessed Oct. 2, 2024.

[4] Texas Instruments, "AN-1852 Designing With pH Electrodes," https://www.ti.com/lit/an/snoa529a/snoa529a.pdf, Texas Instruments, Sep. 2008.

[5] I. Fette, Google Inc., A. Melnikov, Isode Ltd, "The WebSocket Protocol (RFC 6455)," Available: `https://datatracker.ietf.org/doc/html/rfc6455`, IETF, Dec. 2011.

[6] T. Bray, Ed, Textuality, "The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259)," Available: `https://www.rfc-editor.org/rfc/rfc8259`, IETF, Dec. 2017.

[7] R. Fielding, UC Irvine, J. Gettys, Compaq/W3C, J. Mogul, Compaq, H. Frystyk, W3C/MIT, L. Masinter, Xerox, P. Leach, Microsoft, T. Berners-Lee, W3C/MIT, "Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616)," Available: `https://www.rfc-editor.org/rfc/rfc2616`, IETF, June 1999.

Communication between Arduino Uno, ESP32 and Cloud references:

[8] Wikipedia, "I^{2}C," https://en.wikipedia.org/wiki/I%C2%B2C, Wikipedia, accessed Dec. 12, 2024.

[9] Arduino, "Wire Library - Communication Between Devices," https://docs.arduino.cc/learn/communication/wire/, Arduino Documentation, accessed Dec. 12, 2024.

[10] IETF, "The WebSocket Protocol (RFC 6455)," https://datatracker.ietf.org/doc/html/rfc6455, IETF, Dec. 2011.

[11] Links2004, "arduinoWebSockets Library," https://github.com/Links2004/arduinoWebSockets, GitHub, accessed Dec. 12, 2024.

[12] IETF, "The JavaScript Object Notation (JSON) Data Interchange Format (RFC 8259)," https://www.rfc-editor.org/rfc/rfc8259, IETF, Dec. 2017.

[13] ArduinoJson, "ArduinoJson Library Documentation," https://arduinojson.org/, ArduinoJson, accessed Dec. 12, 2024.

[14] IETF, "Hypertext Transfer Protocol – HTTP/1.1 (RFC 2616)," https://www.rfc-editor.org/rfc/rfc2616, IETF, June 1999.

Data logging and visualization:

[15] Microsoft, "TypeScript - JavaScript with syntax for types," *TypeScript*, Available: `https://www.typescriptlang.org/`, accessed: Dec. 12, 2024.

[16] Meta, "React - A JavaScript library for building user interfaces," https://react.dev/, Meta, accessed Dec. 12, 2024.

[17] Vercel, "Next.js - The React Framework," https://nextjs.org/, Vercel, accessed Dec. 12, 2024.

[18] Lucide, "Lucide-React - Icon Library for React," https://lucide.dev/guide/packages/lucide-react, Lucide, accessed Dec. 12, 2024.

[19] Tailwind Labs, "Tailwind CSS - A utility-first CSS framework," https://tailwindcss.com/, Tailwind Labs, accessed Dec. 12, 2024.

[20] Recharts, "Recharts - Composable Charts for React," https://recharts.org/en-US/, Recharts, accessed Dec. 12, 2024.

Stirring System sources:

[21] Byju's, "Fourier's Law," https://byjus.com/physics/fouriers-law/, accessed Dec. 12, 2024.

[22] FEBS, "Glycolysis Step," https://febs.onlinelibrary.wiley.com/..., accessed Dec. 12, 2024.

[23] ScienceDirect, "Aerobic Respiration in Yeast," https://www.sciencedirect.com/..., accessed Dec. 12, 2024.

[24] APMonitor, "Proportional-Integral Control," https://apmonitor.com/..., accessed Dec. 12, 2024.

[25] GeeksforGeeks, "Proportional-Integral Controller in Control System," https://www.geeksforgeeks.org/..., accessed Dec. 12, 2024.

[26] LibreTexts, "Proportional-Integral-Derivative (PID) Control," https://eng.libretexts.org/..., accessed Dec. 12, 2024.

[27] GeeksforGeeks, "Proportional Controller in Control System," https://www.geeksforgeeks.org/..., accessed Dec. 12, 2024.

[28] Control.com, "Closed Loop Control - On/Off Control," https://control.com/..., accessed Dec. 12, 2024.

Constants and variables:

- $pH(X)$ = pH of unknown solution (X)
- $pH(S)$ = pH of standard solution = 7
- E_S = Electric potential at reference or standard electrode
- E_X = Electric potential at pH-measuring electrode
- F = Faraday constant = $9.6485309 \cdot 10^4\, Cmol^{-1}$
- R = Universal gas constant = $8.314510\, JK^{-1}mol^{-1}$
- T = Temperature in Kelvin

Data:

Table 1	Temperature	Time
	24.9	0
	25.68	10
	25.99	20
	26.63	30
	27.37	40
	28.01	50
	28.616	60
	29.227	70
	29.57	80
	29.89	90
	29.87	100
	29.98	110
	30.134	120
	29.774	130

Table 3	Temperature	Time
	25.267	0
	25.766	10
	25.82	20
	26.642	30
	27.437	40
	28.31	50
	28.495	60
	29.156	70
	29.791	80
	29.81	90
	30.068	100
	30.012	110
	29.623	120
	30.45	130

Table 5	Temperture	Time
	24.843	0
	25.448	10
	25.924	20
	26.697	30
	27.639	40
	27.878	50
	28.616	60
	29.196	70
	29.379	80
	29.839	90
	29.697	100
	29.12	110
	29.873	120
	30.37	130

Table 2	Temperature	Time
	25.12	0
	25.274	10
	26.231	20
	26.764	30
	27.525	40
	27.805	50
	28.509	60
	29.376	70
	29.904	80
	29.759	90
	30.091	100
	30.4	110
	29.799	120
	29.13	130

Table 4	Temperature	Time
	24.952	0
	25.618	10
	26.239	20
	26.726	30
	27.77	40
	28.189	50
	28.605	60
	29.291	70
	29.52	80
	29.777	90
	30.092	100
	30.34	110
	29.93	120
	29.45	130

Mean Table	Temperature	Time
	25.017	0
	25.557	10
	26.04	20
	26.693	30
	27.549	40
	28.039	50
	28.568	60
	29.249	70
	29.633	80
	29.814	90
	29.964	100
	29.904	110
	29.866	120
	29.835	130

Figure 8: Data points of the heating subsystem.

Table 2: RPM Measurement Converted to Hertz Versus Time (see plot in the Stirring System section).

Time (s)	347	393	303	363	401	Avg	RPM	Error
0.5	347	393	303	363	401	361.4	39.33	
1	952	923	912	963	990	929.6	31.33	
1.5	1310	1306	1293	1291	1288	1297.6	9.76	
2	1296	1295	1302	1298	1303	1298.8	3.56	
2.5	1299	1299	1302	1301	1299	1300	1.41	
3	1298	1297	1299	1303	1301	1299.6	2.41	
3.5	1306	1295	1301	1295	1303	1300	4.90	
4	1298	1303	1301	1302	1304	1301.6	2.30	
4.5	1306	1294	1295	1302	1305	1300.4	5.59	
5	1299	1301	1295	1298	1304	1299.4	3.36	
5.5	782	793	754	769	802	780	19.07	
6	489	485	470	469	494	481.4	11.33	
6.5	501	505	510	496	489	500.2	8.11	
7	505	497	487	513	502	500.8	9.65	
7.5	510	502	507	504	511	506.4	5.13	
8	503	506	497	511	506	504.6	6.06	
8.5	504	512	505	497	511	505.8	7.86	
9	505	508	489	499	507	501.6	9.33	
9.5	502	487	508	511	505	502.6	4.04	
10	507	506	510	499	506	505.6	5.59	
10.5	800	803	809	795	807	801.2	4.32	
11	802	804	795	799	806	801.2	3.84	
11.5	808	804	801	798	805	803.2	3.36	
12	804	809	800	803	806	804.4	4.83	
12.5	795	808	802	803	805	800.4	8.79	
13	803	810	786	802	801	800.4	3.91	
13.5	806	803	799	804	808	804	5.90	
14	805	808	801	817	807	806.6	2.59	
14.5	802	803	806	804	799	802.8	6.57	
15	815	801	803	807	798	804.8	6.57	

Code:

ESP code:

```
#include "wifi_connection.h"
#include "data.h"
#include "websocket_client.h"
#include <Wire.h>
```

```
5
6  #define MONITOR_BAUD_RATE 115200
7  #define ARDUINO_ADDRESS 1
8
9  const char* websockets_server = "192.168.137.1"; // Change for IP on LAN, or
       host IP on Internet.
10 const uint16_t websockets_port = 3000; // 3000 is the local host ip address
11
12 Data bioreactor_data;
13 Data target_data;
14
15 void start_serial()
16 {
17   Serial.end();
18   Serial.begin(MONITOR_BAUD_RATE);
19   while (!Serial);
20   delay(1000);
21 }
22
23 void data_setup()
24 {
25   target_data.ph = String(-1);
26   target_data.temp = String(-1);
27   target_data.rpm = String(-1);
28   bioreactor_data.ph = String(-1);
29   bioreactor_data.temp = String(-1);
30   bioreactor_data.rpm = String(-1);
31   Serial.println("Data setup successful.");
32 }
33
34 void wire_setup()
35 {
36   Wire.begin();
37   Serial.println("Wire serial I2C setup successful.");
38 }
39
40 void write_wire_data()
41 {
42   String message = target_data.ph + "," + target_data.temp + "," + target_data.
       rpm + "\n";
43   Wire.beginTransmission(ARDUINO_ADDRESS);
44   for (int i = 0; i < message.length(); i++)
45   {
46     Wire.write(message[i]);
47   }
48   Wire.endTransmission();
49 }
50
51 void read_wire_data()
52 {
53   Wire.requestFrom(ARDUINO_ADDRESS, 15);
54   String data = String("");
55   int count = 0;
56   while (Wire.available())
57   {
58     char c = Wire.read();
59     if (c == ',' || c == '\n')
60     {
61       switch (count++) {
62         case 0:
63           bioreactor_data.ph = data;
64           break;
65         case 1:
```

```
          bioreactor_data.temp = data;
          break;
        case 2:
          bioreactor_data.rpm = data;
          break;
        default:
          break;
      }
      data = String("");
    }
    else
    {
      data += c;
    }
  }
}

float get_bioreactor_ph()
{
  return bioreactor_data.ph.toInt() / pow(10.0, bioreactor_data.ph.length() -
    1.0);
}

float get_bioreactor_temp()
{
  return bioreactor_data.temp.toInt() / pow(10.0, bioreactor_data.temp.length()
     - 2.0);
}

int get_bioreactor_rpm()
{
  return bioreactor_data.rpm.toInt();
}

void set_ph(float ph)
{
  target_data.ph = String((int)round(ph * 1000));
}

void set_temp(float temp)
{
  target_data.temp = String((int)round(temp * 100));
}

void set_rpm(int rpm)
{
  target_data.rpm = String(rpm);
}

void handleIncomingMessage(uint8_t *payload)
{
  DynamicJsonDocument doc(1024);
  deserializeJson(doc, payload);

  if (doc.containsKey("heat"))
  {
    set_temp(doc["heat"]);
  }

  if (doc.containsKey("pH"))
  {
    set_ph(doc["pH"]);
  }
```

```
127
128   if (doc.containsKey("rpm"))
129   {
130     set_rpm(doc["rpm"]);
131   }
132 }
133
134 void setup()
135 {
136   start_serial();
137   data_setup();
138   connect_to_wifi();
139   setupWebSocket(websockets_server, websockets_port);
140   wire_setup();
141
142   Serial.println("Setup successful.");
143   delay(3000);
144 }
145
146 void loop()
147 {
148   maintain_wifi_connection();
149   read_wire_data();
150   write_wire_data();
151
152   webSocket.loop();
153   sendData(bioreactor_data);
154
155   Serial.println("Target pH:   " + target_data.ph);
156   Serial.println("Target Temp: " + target_data.temp);
157   Serial.println("Target RPM:  " + target_data.rpm);
158
159   Serial.println("Bioreactor pH:   " + String(get_bioreactor_ph()));
160   Serial.println("Bioreactor Temp: " + String(get_bioreactor_temp()));
161   Serial.println("Bioreactor RPM:  " + String(get_bioreactor_rpm()));
162
163   delay(100);
164 }
```

Listing 1: ESP32 Communication Code for Bioreactor Subsystems

```
1  #include <ArduinoJson.h>
2
3  #ifndef DATA_H
4  #define DATA_H
5
6  typedef struct Data {
7    String ph;
8    String temp;
9    String rpm;
10 } Data;
11
12 String dataToJson(const Data& data) {
13   DynamicJsonDocument doc(1024);
14   doc["pH"] = data.ph.toFloat() / 1000.0;
15   doc["heat"] = data.temp.toFloat() / 100.0;
16   doc["rpm"] = data.rpm.toInt();
17
18   String jsonString;
19   serializeJson(doc, jsonString);
20   return jsonString;
```

```
}

#endif
```

Listing 2: Data Handling Code for Bioreactor Subsystems

```
#ifndef WEBSOCKET_CLIENT_H
#define WEBSOCKET_CLIENT_H

#include <WebSocketsClient.h>
#include "data.h"

WebSocketsClient webSocket;

void handleIncomingMessage(uint8_t *payload);

void webSocketEvent(WStype_t type, uint8_t *payload, size_t length)
{
  switch(type)
  {
    case WStype_DISCONNECTED:
      Serial.println("Websocket not connected.");
      break;
    case WStype_CONNECTED:
      Serial.println("Websocket connected.");
      break;
    case WStype_TEXT:
      handleIncomingMessage(payload);
      break;
  }
}

void setupWebSocket(const char* host, uint16_t port)
{
  webSocket.begin(host, port, "/");
  webSocket.onEvent(webSocketEvent);
  webSocket.setReconnectInterval(5000);
  Serial.println("Websocket setup successful.");
}

void sendData(const Data& data)
{
  String jsonString = dataToJson(data);
  webSocket.sendTXT(jsonString);
}

#endif
```

Listing 3: WebSocket Client Code for Bioreactor Subsystems

```
#include <WiFi.h>

#ifndef WIFI_CONNECTION_H
#define WIFI_CONNECTION_H

// Configure for either shared LAN with the interface,
// or for the Internet to connect to a different LAN.
// For example: https://support.microsoft.com/en-us/windows/use-your-windows-pc
    -as-a-mobile-hotspot-c89b0fad-72d5-41e8-f7ea-406ad9036b85
```

```
// Using a Windows hotspot to create a LAN is how this has been used for the demonstration:
// Settings > Network & Internet > Mobile Hotspot > Toggle On/Off
const char *ssid = " 4738";
const char *password = "$8S87o06";

void connect_to_wifi()
{
  int counter = 0;

  Serial.println();
  Serial.print("Connecting to network: ");
  Serial.println(ssid);
  WiFi.disconnect(true);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
    // Reset board after 30 seconds, if not connected.
    if (counter++ >= 60)
    {
      ESP.restart();
    }
  }

  Serial.println(" ");
  Serial.println("WiFi connected.");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void maintain_wifi_connection()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    int counter = 0;

    Serial.println("WiFi connection lost.");
    Serial.print("Attempting to reconnect.");

    WiFi.begin(ssid);
    while (WiFi.status() != WL_CONNECTED)
    {
      delay(500);
      Serial.println(".");
      // Reset board after 15 seconds, if not connected.
      if (counter++ >= 30)
      {
        ESP.restart();
      }
    }
    Serial.println(" ");
  }
}

#endif
```

Listing 4: WiFi Connection Code for Bioreactor Subsystems

Arduino Code:

```
#include "data.h"
#include <Wire.h>
#include <math.h>

#define MONITOR_BAUD_RATE 115200
#define ARDUINO_ADDRESS 1

// Heating globals, constants, and macros.
#define PWM_PIN 3
#define THERMISTOR_PIN A0

const float R1 = 9940;
const float c1 = 0.001129148, c2 = 0.0002347118632678, c3 = 0.0000000876741;
const float kp = 0.3;
const float ki = 12.0;

float set_temperature = 30.0;
float current_temperature = 0.0;
float PID_error = 0.0;
float PID_i = 0.0;
float PID_value = 0.0;

float error, Vheater;
long prevtime, T1;

unsigned long lastUpdateTime = 0;
const unsigned long updateInterval = 200;

unsigned long lastSerialUpdateTime = 0;
const unsigned long serialUpdateInterval = 1500;
// End heating globals, constants, and macros.


// Stirring globals, constants, and macros.
// Located after class definition.
// End stirring globals, constants, and macros.


// pH globals, constants, and macros.
#define pHPin A1
#define acidPumpPin 12
#define basePumpPin 13
#define phS 7 //ph of standard solution
#define Es 1026.393  //Electrical potential at reference or standard electrode
#define F (9.6485309*10000) //Faraday constant
#define R 8.314510 //universal gas constant
#define T_ph 298 //Temperature in Kelvin is assumed to be constant
#define sendPHInterval 100 // Sends the pH every 100 milliseconds
#define pumpCheckInterval 2000 //How often the pump is activated when outside the pH range, to give time for stirring
#define pumpActivationInterval 2500 //The pump is activated for (pumpActivationInterval - pumpCheckInterval) milliseconds

float targetpH = 5; //Initial target pH
float pHHi = 5.20; //NB: Tolerance range
float pHLo = 4.80;
float pH;
bool pumpActivated = false;
// End pH globals, constants, and macros.

Data bioreactor_data;
Data target_data;
```

```
61
62 void start_serial()
63 {
64   Serial.end();
65   Serial.begin(MONITOR_BAUD_RATE);
66   while (!Serial);
67   delay(1000);
68 }
69
70 void data_setup()
71 {
72   target_data.ph = String(7);
73   target_data.temp = String(25);
74   target_data.rpm = String(0);
75   bioreactor_data.ph = String(0);
76   bioreactor_data.temp = String(0);
77   bioreactor_data.rpm = String(0);
78   Serial.println("Data setup successful.");
79 }
80
81 void wire_setup()
82 {
83   Wire.begin(ARDUINO_ADDRESS);
84   Wire.onRequest(requestEvent);
85   Wire.onReceive(receiveEvent);
86   Serial.println("Wire serial I2C setup successful.");
87 }
88
89 void requestEvent()
90 {
91   String message = bioreactor_data.ph + "," + bioreactor_data.temp + "," +
     bioreactor_data.rpm + "\n";
92
93   for (int i = 0; i < message.length(); i++) {
94     Wire.write(message[i]);
95   }
96 }
97
98 // Remaining code for heating, stirring, and pH control continues...
```

Listing 5: Complete Bioreactor Code for Heating, Stirring, and pH Control

```
1 #ifndef DATA_H
2 #define DATA_H
3
4 /* Stores:
5  *   - pH as a String, representing a 4 digit integer (1000 times actual value).
6  *   - Temperature as a String, representing a 4 digit integer (100 times actual
       value).
7  *   - RPM as a String, representing a 4 digit integer (1 times actual value).
8  */
9 typedef struct Data {
10   String ph;
11   String temp;
12   String rpm;
13 } Data;
14
15 #endif
```

Listing 6: Data Structure for Bioreactor Subsystems