Challenge 2: Global Health Care

EEE-CS Specialist Team 12 Final Report

Authors: Haonan Shen, Yuxuan Li, Vladislav Repinskiy, Vincent Tan, Minh Nguyen Nhat, Adamos Hadjivasiliou, Rui Chen, Zhizhe Xu, Zhang Bang, Samuel Bouilloud

1. Introduction

Authors: Minh Nguyen Nhat, Samuel Bouilloud

The overall goal of the project is tackling TB disease infection in areas where western medicine treatment is unavailable. As TB is the 9th leading causes of death in the world, it is crucial that we address the issue of TB thriving in areas where this is most likely to occur. Namely, places with low social conditions and lack of effective healthcare system.

Places like Africa, where 14% of the Earth's population only produces less than 0.1% of the world's vaccines, is the most in need of the solution that this project intends to tackle. The main idea is to construct a facility that can produce TB vaccines locally, in order to address the health needs of the local community. This facility should be able to go through the whole process of vaccine manufacturing using as many local resources as possible and remaining sustainable. The project also needs to help infrastructural and socio-economic development in the area, while being aware of the local cultural and ethical contexts that might affect its success.

A bioreactor will be housed in this facility which produces the vaccines that will be distributed to the locals. Our EEE-CS specialist team was specifically working on the controls for the bioreactor, including temperature, pH level and stirring control, all of which integrated into one user interface. We split into four sub-teams, each responsible for designing and implementing one subsystem.

The heating subsystem consists of two main parts: temperature sensor (10kΩ NTC Thermistor ND06P00103K) to determine the substance's temperature, and a heater to change it as desired (3Ω, 30W heater element). Our goal here was to design a circuit and a computer program capable of reading the analogue value from the sensor and convert it into temperature in units of Celsius. Based on this measurement, our computer should be able to determine whether our bioreactor needs more heating or not, and adjust the written voltage value to the heater accordingly. All communication between the built system and computer is done through the launchpad (MSP430).

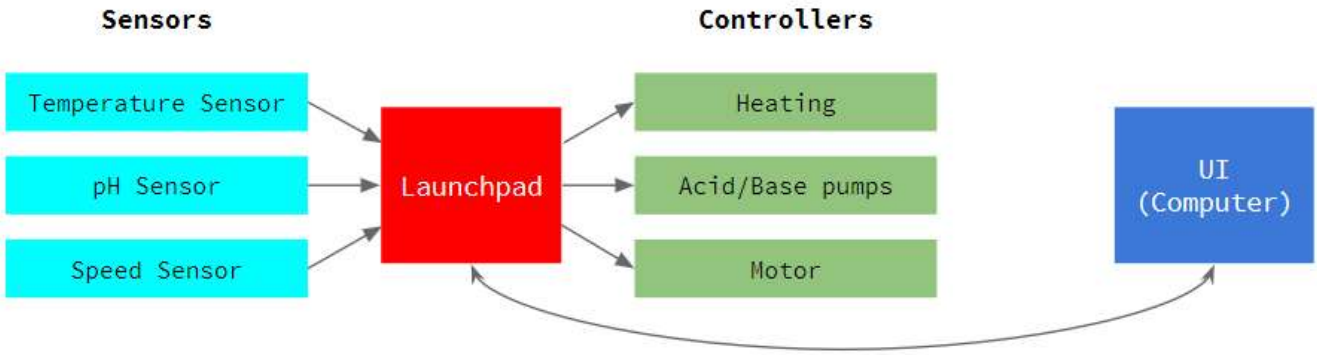
Stirring is one of the others subsystem, responsible for mixing the liquid found inside the reactor. The design uses a motor to stir, which is powered through the two 3V DC motor drive pins. We can use the launchpad and our code to vary the analogue value written to the motor drive pins, thus changing the speed of the motor's rotation. Monitoring system is a rotating bow-tie that breaks a beam of light twice per rotation. Feedback is then, given by a photo-interrupter (CP2S700HCP) that measures the time elapsed between two interruptions and displays the rotational speed. We can use this information to know the current speed of the motor and set it to a certain speed.

Controlling pH level of the substance is also done through two-way communication with the computer. Briefly, a PH-100ATC Probe is used to measure the pH level of the liquid and through the launchpad, it sends information to our computer where our code processes it and displays it. We are also able to adjust the pH value ourselves by using two 6V, 1A peristaltic pumps that pumps the wanted substance into the reactor: acid if the solution is too basic and vice versa.

All the three subsystems are integrated into one bioreactor using different pins, all connected to one microcontroller. The whole integrated system can be controlled from a User Interface that receives data from each of the three subsystems and displays them on a real-time graph. It can also be used to write values to the different pins of the launchpad, thus not only reading but also adjusting pH values, motor speed and temperature.

Our design is a crucial part of the project since this system enables the bioreactor to produce the desired vaccines under required conditions. Without this system, chemical or biomedical engineers would have no control over the production of medicines for local communities, although developing the region's healthcare is supposed to be the primary goal of this project. The EEE-CS specialist team's mission is to provide every electrical and computational background needed.

(Further details on each subsystem found below)



2. Subsystem Descriptions

2.1 - Heating System

Authors: Yuxuan Li, Haonan Shen

Description: The entire heating system consists of two parts: temperature capturing and temperature control. The main goal is to make the heater turn on automatically when the detected temperature is below the target level (e.g. 30°C). Otherwise, the heater is turned off. The launchpad then, can upload the ambient temperature level to the computer every second so the user can read the real-time value on the screen.

Equipment used: MSP430 launchpad, one 3Ω 30W heater element, one 10kΩ NTC thermistor, one FDP 7030BL transistor, 12V DC power supply, one LED, one 10Ω resistor, one 220Ω resistor, jumper wires. *(All circuit design provided in appendices)*

2.1.1 - Temperature capture - *Author: Yuxuan Li*

The general idea is to use a voltage divider to detect the potential difference across the thermistor, read the voltage value at pin 2 using *analogRead* function, then convert the data to degrees Celsius with the Steinhart-Hart equation.
The resistance of the thermistor is about 10kΩ at 25°C, so connect a 10kΩ resistor in series. As the resistance of thermistor changes if the temperature is changed, the voltage at the end of the thermistor which connect to pin 2 is also changing.
According to this algorithm (Steinhart-Hart equation) we can get the temperature in units of Celsius:

```
12 float ThermistorVal = analogRead(2);
13 float ThermistorTempC;
14 ThermistorTempC = logf(10000.0 * ((1023.0 / (ThermistorVal) - 1)));
15 ThermistorTempC = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * ThermistorTempC
16                  * ThermistorTempC ))* ThermistorTempC );
17 ThermistorTempC = ThermistorTempC-273.15;
```

Where *analogRead(2)* is what maps the voltage of pin 2 with respect to the ground on a scale of 0 to 1023, where 0 means 0V and 1023 means 3.3V.

Testing and calibration (see figure below):
We place a separate thermometer into the liquid, and check if the reading of the thermometer corresponds with the output sent through the launchpad to the screen. If not, we just change the codes accordingly to calibrate. A specific way of calibration can be: if the value measured by our thermistor is smaller than the actual temperature, we add a number(eg.20) to the *rawADC* in the code, where *rawADC* is *"ThermistorVal"* in line 12 as shown in the image below. If the measurement is bigger, we subtract a number.
Then we continue this calibration procedure until the temperature captured is accurate.

Figure - Code to calibrate the heating system

```
float ThermistorVal = analogRead(2);
ThermistorVal = ThermistorVal + 20;
float ThermistorTempC;
ThermistorTempC = logf(10000.0 * ((1023.0 / (ThermistorVal) - 1)));
ThermistorTempC = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * ThermistorTempC
                  * ThermistorTempC ))* ThermistorTempC );
ThermistorTempC = ThermistorTempC-273.15;
```

2.1.2 - Temperature control - *Author: Yuxuan Li*

The heater is turned off when the detected temperature is greater than the target temperature (30°C). Otherwise, the heater is turned on until the temperature reaches the target level.
The operation voltage of the heater is 9V but a power supply of 12V is given. So, we use Pulse Width Modulation (PWM) to adjust the power of the heater. According to the calculation below, we can get the pulse width:

$W = V^2/R*t \quad (V_1)^2/R*t_1 = (V_2)^2/R*t_2 \quad (V_1)^2/(V_2)^2 = t_2/t_1 \quad 12^2/9^2 = t_2/t_1 \quad \mathbf{t_2/t_1 = 16/9}$

This means, if the heater is driven by a 12V power supply and stop 7ms after working 9ms, it will function as it is connected to a 9V power supply.

We use a MOSFET to control the high voltage circuit as follow: we connect the gate of the transistor FDP 7030BL to the microcontroller via a certain pin (pin 5), the drain to the positive electrode of 12V power supply and the source to the heater. If the temperature returned by the thermistor is under 30°C, the pin connected to the ground will be output a high voltage (3.3V) and gate will let current flow from the drain to the source, making the heater work. If the temperature is above the bound, the output of pin 5 will be digital low (0V), which means that the gate will be open, so no current will flow through the heater to operate it (see figure below). Notice: in the high voltage circuit, the current must flow into the MOSFET at pin D and flow out at pin S. Otherwise you may damage the MSP430 or other components.

```
const int ctrl = 5;
if (ThermistorTempC < 30)
{digitalWrite(ctrl, HIGH);
delay(9);
digitalWrite(ctrl, LOW);
delay(7);}
else{digitalWrite(ctrl, LOW);}
```

Figure - Code for the Launchpad: If the temperature is below 30ºC, the heater is turned on, otherwise it is turned off.

(The tests for the heating subsystem can be found in the appendices)

2.2 - pH system

Authors: Bang Zhang, Rui Chen, Samuel Bouilloud

Description: The pH subsystem includes two main parts: probe and pumps. The pH probe works as a data collection sensor, which reads the pH value of the given solution every second. Hence, the data is sent to the Launchpad via a circuit which consists of an amplifier and a potential divider. The data is analysed through the codes, specially designed for pumps driving. The pumps ensure that the pH value remains stable by pumping acid or alkali into the solution.

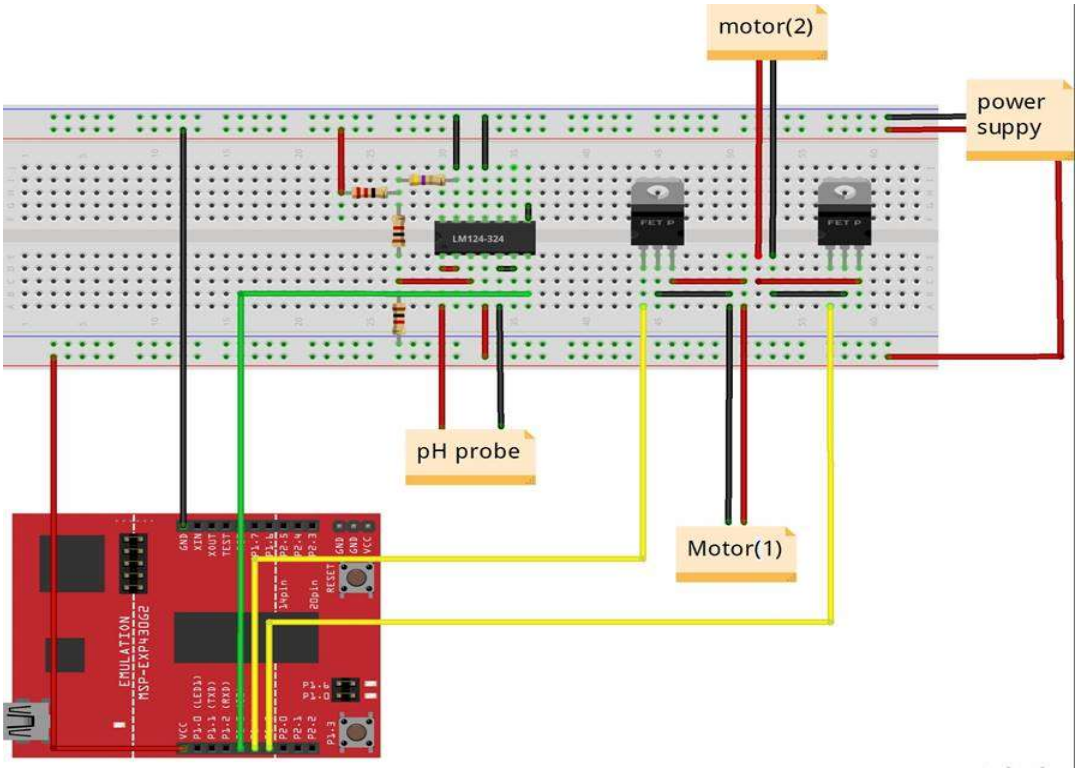
Specifications: Two 6V 1A peristaltic motor pumps, PH-100ATC pH probe, two FDP 7030BL transistors, 12V supply, LM124 quad-operational amplifier, resistors, wires

The circuit design

The pH probe is connected to the data collection circuit via a BNC adapter. Initially, the form of the reading is voltage and the signal is transmitted to an amplifier, which acts as a buffer for the potential divider circuit to ensure that the voltage supply is constant. The other connection, connected to the microcontroller, allows the potential reading of the probe, reaching the microcontroller. Due to the high-impedance of the probe itself, the voltage drop on microcontroller would be very tiny without the op-amp.

In case of the control circuit, each pump is separated from the power supply using two transistors. The two gates are connected to specific pins on the launchpad, while the drain and the source are connected to negative and positive terminals of the power supply. When a high voltage is written from the microcontroller to the pins, the circuit of the pump is closed and absorb acid or base from each bottle to adjust the pH value of the solution to a certain level. When the pH of the solution matches the required pH, the circuits are reopened, and pumps stop working (see figure 1 below).

Figure 1 - The circuit used for the pH subsystem



The code for the pH control

To calculate the current pH of the solution, we use a transfer function (see figure 2). This function takes as input the *AnalogRead* from the pH probe and calculate the pH value from it. To do it, it also uses the voltage output of the standard solution (read when pH = 7), the current temperature (measured by heating system) and some constants, like Faraday's constant or universal gas constant.

The transfer function of the pH electrode is:

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

where

- 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 is the Faraday constant = $9.6485309*10^4$ C mol^{-1},
- R is the universal gas constant = 8.314510 J K^{-1} mol^{-1}
- T is the temperature in Kelvin

```
#define Faraday 9.6485309*pow(10,4)
#define R 8.314510
#define ln10 2.302585093

// read the probe value, the electric potential at pH measuring electrode
// multiply by 0.0029 because it is  2.9 mV per unit of analogRead
pHX = calcX(analogRead(probe) * (2.9/100000));

float calcX(double Ex)
{
  float X;
  float pHS = 7;
  float Es = 190*2.9/100000;

  X = pHS + (((Es - Ex)*Faraday)/(R*T*ln10)); //apply transfer function

  return X;
}
```

Figure 2 - Transfer function of the pH electrode

After measuring the current pH value, we need to act accordingly to set this value to our wanted value. To do this we use the acid and alkali pumps. Indeed, we know that the optimum pH for our bioreactor is 5, so let's take a lower limit (4.7) and an upper limit (5.3). With all this information we can code the loop of our Launchpad for the pH control. If the pH exceeds our high limit, the acid pump is activated; and if it goes below our low limit, the alkali pump is activated. These pumps are activated during 1sec as we can see with "delay(1000)". In addition, we have a limit of activation of the pumps to avoid overflow, which is reseted after some time (see code below in figure 3).

Figure 3 - Code for activation of the pH pumps

```
void loop() {
  pHX = calcX(analogRead(probe) * (2.9/100000));
  float pHLow = 4.7;
  float pHHigh = 5.3;
  int pumpLim = 10;//number of pump activation possible to prevent overflow
  int pumpAct = 0; //number of pump activation

  if(pumpAct <= pumpLim) {
    if (pHX > pHHigh) {
      digitalWrite(pumpA, HIGH); //turn on acid pump
      pumpAct = pumpAct + 1;
    }
    else if (pHX < pHLow) {
      digitalWrite(pumpB, HIGH); //turn on base pump
      pumpAct = pumpAct + 1;
    }
  }
  delay(1000);
  digitalWrite(pumpA, LOW); //turn both off
  digitalWrite(pumpB, LOW);
}
```

(The tests for the pH subsystem can be found in the appendices)

2.3 - Stirring System

Authors: Vlad Repinskiy, Vincent Tan

Description of subsystem

The stirring subsystem consists of a 3V DC electric motor and a CP2S700HCP photo-interrupter. These elements are united into one subsystem to perform the following task: Stir the solution at a desired RPM. To assess if our system is working correctly we have the following specifications to test it against:

1. The stirring frequency should be within the range of 500 to 1500 rpm.
2. We should be able to log RPM and input and output voltage data
3. Should allow user to input desired rpm (in collaboration with UI subsystem)

System Design

Based on the technical requirements, we broke the solution down into these steps:

- Build a circuit for the motor, the sensor and then integrate them together (circuit diagram in Appendix 3)
- Write the code to calculate RPM and output the appropriate voltage to the motor based on desired RPM

(Circuit diagram can be found in appendices)

Detailed principles of operation

We used a photo-interrupter to measure RPM. First, we tried to determine what kind of data we receive from a photo-interrupter in order to understand how we can use it to calculate RPM. After testing we concluded that numbers retrieved from the sensor are proportional to the distance of the object the light is reflected off. This means that when a motor blade passes above the photoresistor the output value read from it drops. We've also established that the reading is highly dependent on the light settings in the room where the photo sensor operates. So, if we were to construct such system in the real-world settings we would need to ensure a constant light source for the sensor. Our solution to this was to have a 3 second initialisation period that will adapt the sensor to the current light setting and will allow the sensor to work flexibly in different environments. During this initialisation period a constant voltage is applied to the motor to ensure a constant RPM, and the midpoint between the highest and lowest values recorded would be used as the deciding value whether the light was reflected. From that, a counter every loop is used to calculate the RPM every 500ms.

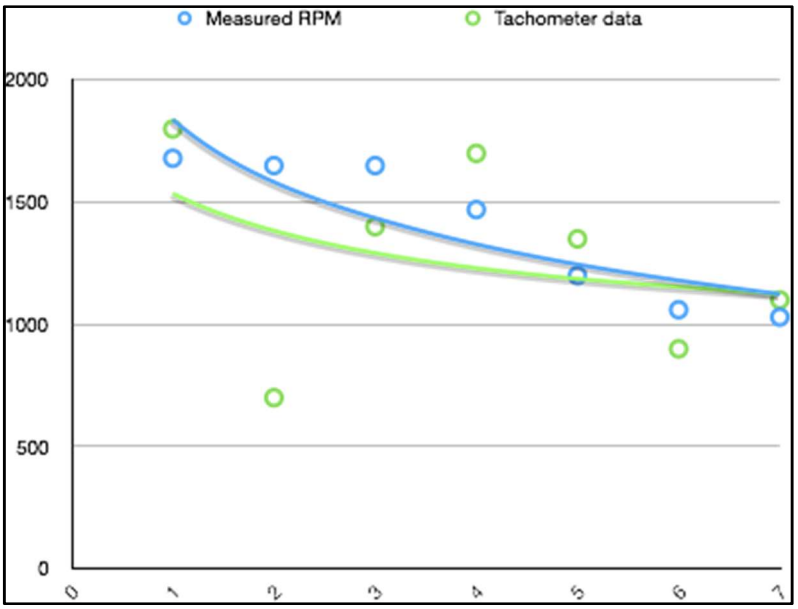
The difference between the calculated RPM and desired RPM is multiplied by a fixed constant, and added or subtracted to the motor *AnalogWrite* value accordingly.

Testing method

We are able to test our system using a tachometer, so we can confirm that we're calculating correct values of RPM and that system meets our specifications. Using tachometer might be problematic as the tachometer we were provided with utilises the same principle as the photo interrupter, and when measuring RPM with it we have to make sure that we're not pointing it

in a direction of a photosensor, as it might interfere with the values we're reading from the sensor.

Test Results and Graph



On this graph we've plotted values of RPM measured with a tachometer and a photosensor. As can be seen from the graph, the results are not accurate, which we believe to be caused by an accuracy of tachometer. During this test we determined target value of RPM to be 1000. And as can be seen from the graph - the general trend is that the RPM value approaches 1000. This graph also reflects another feature of our code - the output voltage value is adjusted proportionally to the difference between measured RPM and desired RPM, which makes the the change in RPM more precise. At last, despite we were not completely satisfied with the results of our tests, it is generally the case that the subsystem was fitting the specifications we've set for it.

Real world application

There are a few changes that could be done in order to improve our subsystem's performance. First, having a constant light background is the key. This could be done either by keeping the sensor in a dark room and shining a laser at it, or keeping it in a room with fixed lighting. In addition, we could have used several photosensors and averaged their values to minimize the rate of error. There is also a chance that the spinning arm that reflects the beam of light is not optimally reflective.

The motor that we used for this challenge was not powerful enough such that a change in the density of the liquid would affect its RPM. If we had a more powerful motor that would be unaffected by the medium it is in, we could better calculate what voltage has to be applied to achieve the desired RPM.

2.5 - User Interface

Authors: Minh Nguyen Nhat, Adamos Hadjivasiliou, Samuel Bouilloud

The user interface of our bioreactor is the first contact of the user with the system. The main goal is to display real-time data of the different values measured: temperature, pH and rotations per minute (rpm) of the stirring system.

To code this User Interface, we used Processing IDE, a very useful and simple language to create display windows. It is also commonly used with microcontrollers (e.g. our Launchpad or Arduino) due to the facility of communication between them.

The code behind this Interface

First of all, we need to establish a communication from the microcontroller to our computer to transmit the values measured. This is done using the Serial Port of communication through the USB cable binding the two components. For the Launchpad side, the code consists of writing the values in this port at every "Loop" (the code is repeated while the board is connected), under the format of a String "<Temperature>|<Speed>|<pH>".

For the Processing side (computer side - see figure), we need to read the serial port for each line sent, doing all verification necessaries to see if we received the data complete. We have now a String with the data in it (variable input in the code), so we need to split it into three different parts, using the '|' as a separator. Finally, we can store these values into variables, converting them to Float (a numerical value).

```
// If this is the first cycle, we ignore it to avoid partial data
if (first_cycle) {
  first_cycle = false;
  return;
}

// Collect the data if available and store it in input
if (myPort.available() > 0)
{
  input = myPort.readStringUntil('\n');
}
else {
  return;
}

// If the value is Null, we ignore it
if(input == null) {
  return;
}

// Split the different parts of the input in an array
String[] input_parts = split(input,"|");

// Check if all the parts of the data were collected. If not, we ignore it
if (input_parts.length != 3) {
  return;
}

// Store all the values in different variables, converting them to Float values
Float temperature = Float.parseFloat(input_parts[0]);
Float speed = Float.parseFloat(input_parts[1]);
Float pH = Float.parseFloat(input_parts[2]);
```

Figure 1 - Processing code for catching the input values

After receiving the values measured, we need to display them to the user. The way of display chosen is graphs because we can see the evolution of the value on it. To do it, we use one function to each graph. These functions create a white background for the graph, the scale labels for it and, at each loop, a line representing the value (from the bottom to the right scale point). As we go, the lines go more and more to the right and the graph is constructed. On the other hand, for precision, we display the exact value just underneath (see figure 2 below).

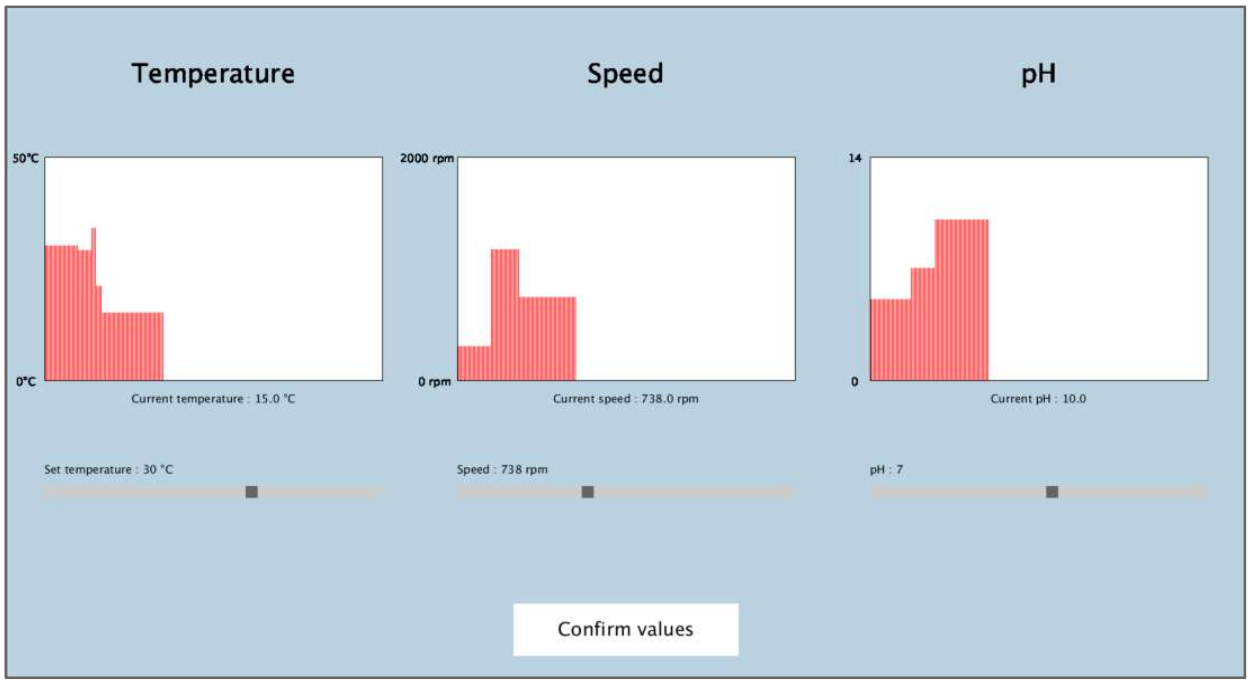


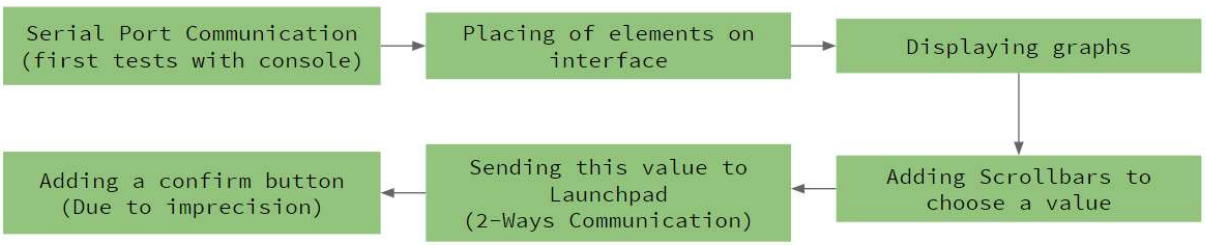
Figure 2- Our User Interface

So now that we have all the display necessary, the next step is to add a way for the user to choose the values he wants for our bioreactor. For that we use three scrollbars (for each value we need to choose), implementing a class "Scrollbar" in the code. This class contains a function to get the value so that we know what we are choosing (display of it just on top), and send the right value to the Launchpad. Because of the imprecision of these sliders, we have a confirm button, so that the user clicks when he is ready.

The last step of this User Interface is to add a way to send the chosen values back to the Launchpad. We also use the Serial Port to do it, it is called a Two-Ways Communication. Indeed, when the button is pressed, all the values are written in it. From computer to microcontroller, we can't send String so we send the values byte after byte. The temperature and the pH 'fit' into one byte, because they are smaller than 255; but we must split the speed between thousands/hundreds/dozens/units. On the Launchpad part of the code, we read this port byte after byte when an input is detected, to then recompose the chosen values and use them for the three subsystems of our bioreactor, as target temperature/speed/pH.

Design Cycle / Testing:

Finally, this is how we coded our User Interface, adding each element one at the time. So, we can think of a design cycle like this:



While adding elements, we were always testing everything at each step. As we didn't have the other subsystems, we tested it with a Launchpad simulating it, sending random values for temperature, pH and stirring speed.
After all these steps done, the User Interface was ready, and the next goal was to connect it to the whole system, to get a proper functional bioreactor.

3. Overall System Integration and Summary

After properly testing each subsystem with its own code and circuit design separately, the next task was to integrate the parts into one complete system. We placed all three subcomponents - heater with thermistor, pH probe and pump pipes, stirring motor - into one vessel that acts as our bioreactor. Here our team had to be sure that the subparts are placed correctly so they fit in the vessel and that they don't interfere with each other; the pH probe and heaters should not block the rotating wheel of the motor.

All circuits, electrical components are placed on one breadboard that should be supplied with 12V from the power supply. All subparts that require voltage to operate can then be powered through the bread board's positive and negative connections. We use one MSP launchpad to control and read everything from the system, which means we had to make sure the pin connections of each part is consistent with pins we use in our code to read from or write to. About the computer program, all code is integrated into one file that makes the launchpad operate the system. Here, it is essential that we check there are no clashes between variable names of different components. To make the user interface capable of controlling the system, we established a two-way communication between the Processing IDE and the Energia code.

This complete system described above is our final "product". It should be fully capable of controlling the environments, under which the vaccines are produced. We have the user interface connected with two-way communication to the launchpad. This enabled us to read real-time data from the bioreactor. Information about the temperature of the reactor, rotation speed of the motor and pH value of the solution were all displayed on the graph for the user. Knowing about the exact conditions, we could use the control bars to set each of the values as desired. When the control bar for temperature was moved, the heater turned on (or off) accordingly, which resulted in the thermometer giving feedback and displaying different values on the real-time graph. Same applied to stirring and pH; this effective structure enabled us to control all these systems while constantly receiving feedback.

Although the system initially seemed to be fine, constant testing showed us that it is not working consistently every time. The sensors, especially the photo-interrupter in the motor was returning inaccurate data for displaying, thus hindering us from having total control. The pH probe was also inconsistent and sometimes displayed random values. We tested it using water, which returned a pH value between 5 and 8; also, our team was unable to use acid every time

for testing which also resulted in some unexpected inaccuracies. The heating system was what worked most consistently; the thermistor showed quite accurate data, and the heater also turned on according to our commands.

Flaws in the 12V supply could also be the reason for unpredictable functioning since our team was only able to use the 6V sometimes during testing. Power loss through the thin cables and imperfect circuit connections could also have contributed to occasional malfunctions.

All in all, although, our system was designed and carried out carefully, subtle faults made it difficult to make it operate perfectly every time.

4. Appendices

4.1 - Appendix 1- Heating Subsystem circuits

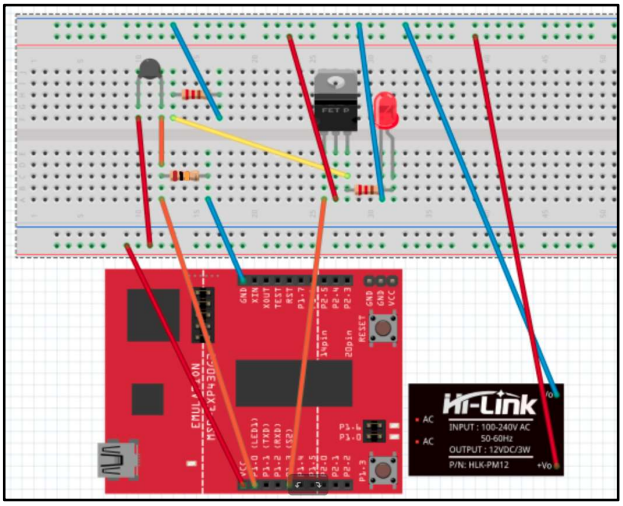


Figure - The circuit diagram for the whole heating system

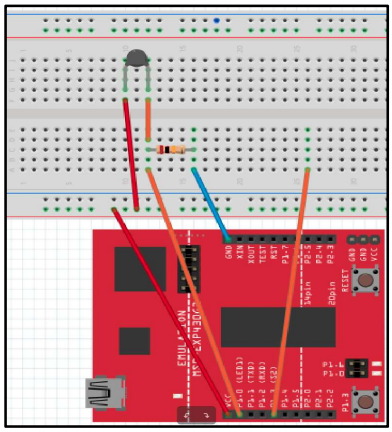


Figure (left) - The circuit diagram for the temperature capture only

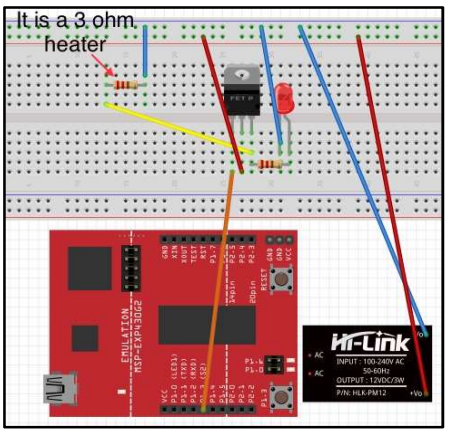


Figure (right) - The circuit diagram for temperature auto-control system

4.2 - Appendix 2 - Testing of the Heating Subsystem

To confirm that the heating system worked well, we had to test it.

Testing of the heater alone: To see if the heater is working, we connect a LED with a resistor, in parallel to the heater. If the system is working (heater on), the LED should light up, otherwise the LED does not light up.

Testing of the whole system: We put the heating system (thermistor and heater) in a beaker of water, and read the temperature on a thermometer besides, always checking that the value printed on the Serial Port (thermistor value), was corresponding. Our required temperature here is 30ºC.

Figure - Heating System results table

Time/s	Temperature/°C	Time/s	Temperature/°C	Time/s	Temperature/°C
0	19.84	50	26.18	100	30.06
10	20.75	60	27.49	110	30.24
20	22.17	70	28.30	120	29.94
30	23.38	80	29.08	130	30.12
40	24.96	90	29.96	140	30.09

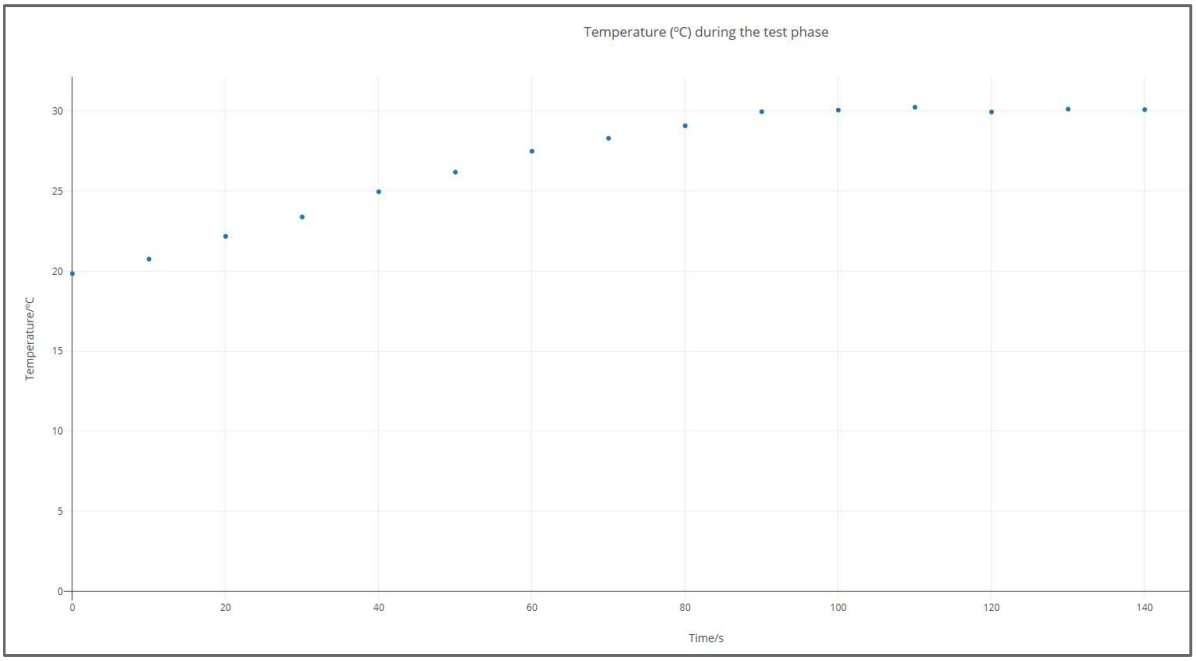


Figure - Graph representation of the tests

We can see that the temperature of solution is raised from 19.84 °C to about 30°C, and kept around 30°C. The heater was on since the start, during approximately 85 seconds. Then the temperature was stabilized. This test showed us that the system was working and that we could integrate it with the others subsystems.

4.3 - Appendix 3 - Testing of the pH Subsystem

To confirm that this subsystem is working, we had to test it. To do it we placed the system (pH probe and pipes from pumps) in a beaker of water, with a pH meter to confirm the values, and started the timer. Our goal here was reaching the pH value 5. For 1 minute, we collected the values from the pH meter, confirming that the ones printed by the Launchpad Serial Port were in adequacy.

Figure - pH System results table

Time/s	pH value	Time/s	pH value	Time/s	pH value
0	7.22	20	5.32	40	5.17
5	6.77	25	5.15	45	5.12
10	6.23	30	5.08	50	5.08
15	5.56	35	5.13	55	5.11

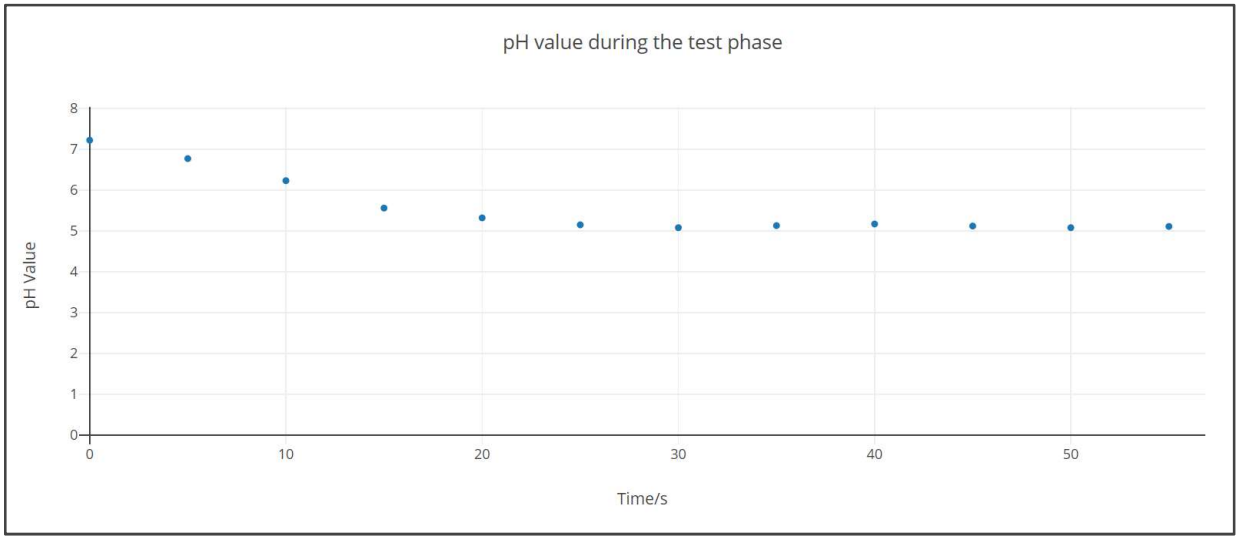


Figure - Graph representation of the tests

With these tests, we can see that the pumps are directly activated, because our starting pH value is 7.22. They keep getting activated for 30 seconds, until the pH is around 5, when it

stabilizes. These results confirmed that the subsystem was really working, and that we could integrate it with the other subsystems.

4.4 - Appendix 4 - Stirring Subsystem Diagram

Circuit diagram for the stirring subsystem. The photo-interrupter part of the circuit and the motor part of the circuit were tested separately first before integrating them into one circuit and breadboard.