

Challenge 2: Global Health care

EEE-CS Specialist Team Final Report

Authors: Yashvir Sangha, Andy Jing, Xinyao Qian, Huiyan Xiao, Waheeb Ameen Hassen, Jeremy Lo Ying Ping, Kelly Ding, Rafael Asensio Delgado, Xiang Long, Thowhid Ahmed, Chenyao Li

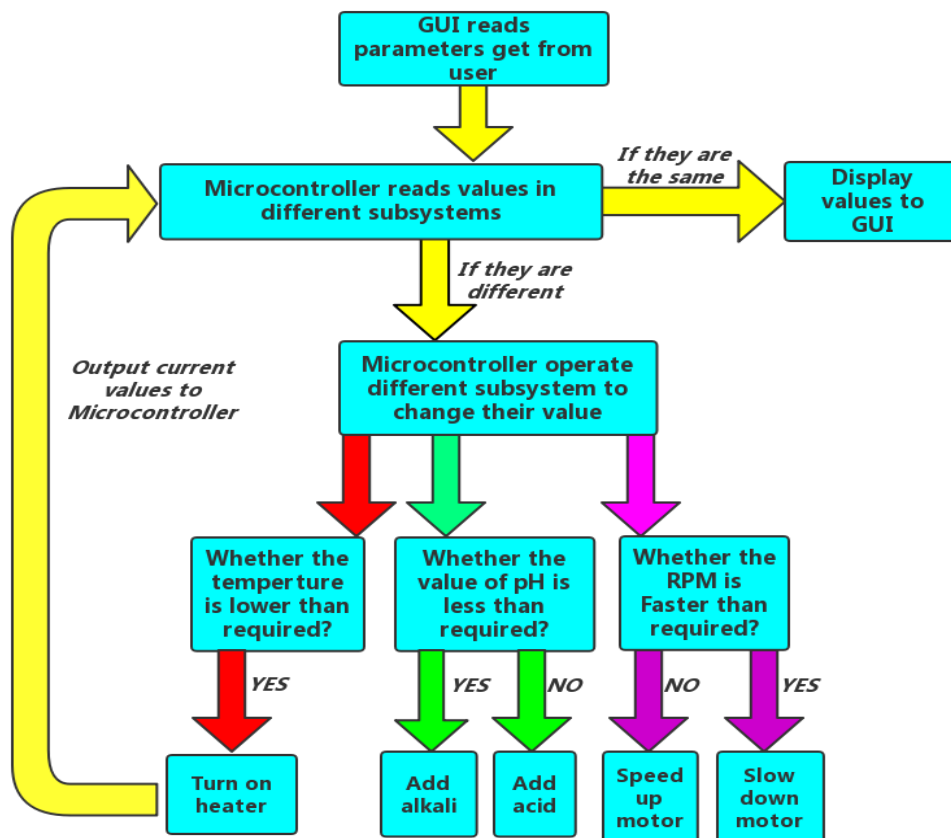
1 Introduction	2
2 Subsystem Descriptions	4
2.1 Temperature Control System	4
2.1.1 Temperature Sensing Subsystem	4
2.1.2 Heating Subsystem	4
2.2 pH subsystem = pHsensor subsystem + pumps control subsystem	6
Description	6
2.2.1 pH Probe Subsystem	7
2.2.2 Pumps Control Subsystem	7
2.2.3 pH Subsystem circuit overview	
2.3 Stirring Subsystem = RPMsensor subsystem + RPMcontrol subsystem	8
2.3.1 RPM Calculation	9
2.3.2. Speed Control	9
2.4 User Interface	10
2.4.1 User input data	10
2.4.2 Display Data	11
2.4.3 User Interface Example Result	11
2.4.4 User Interface application in real word problem	11
3 Overall System Integration and Summary	12
4 Appendices	
4.1 User Interface code for Processing	13
4.2 User Interface code for Energeria	16
4.3 Appendix: pH	20
4.4 Appendix: PH Subsystem Testing Results	23
4.5 Appendix: Temperature Sensing Subsystem	25
4.6 Appendix: Heating Subsystem	26
4.7 Appendix: Temperature Control System	28
4.8 Appendix: Stirring Subsystem and Results	28
4.9 Appendix: Stirring Code	32

1 Introduction

Section Author: Andy Jing

Tuberculosis (TB) is a deadly disease; Uganda's TB rate was 253 per 100,000 people in 2017, compared to 159 per 100,000 in 2015 (1).¹ Also, the ratio between newly reported TB patients and those who were treated is greater than five, which means the rate of treating people cannot catch up to the rate at which they are being diagnosed. Therefore, in Engineering Challenges 2, we set out to work in an interdisciplinary team in order to design and build a bioreactor model that could scale so as to be used in a TB vaccine production facility in Uganda. During the project, we were not only concerned about technical problems and challenges, but also paid attention to the impact that we could make in the local community, for example the pollution made by producing vaccines.

Our CS-EEE team specialised in building a bioreactor control system which contains four main subsystems: the pH system, the temperature control system, the stirring system, and the user interface. This bioreactor can change the pH, temperature and stirring speed of solution to any value the user requires and the following chart concisely shows the relation between different subsystems.



Firstly, the user needs to enter the parameters they want through the GUI, and then the microcontroller will receive these values and compare it with the readings obtained from each subsystem. If they are the same then the value will be displayed on the GUI. If not, the microcontroller will run code written by team members in each subsystem to make readings match the target values.

¹ <http://www.tbonline.info/posts/2017/8/29/tb-prevalence-rises-60-uganda-survey/>

This bioreactor can detect many different indexes of solution in a container, among the whole project the only place that needs the cooperation of our bioreactor is the vaccine producing department. For the production of the TB vaccine it is important that all of the subsystems are working in their correct ranges, otherwise it will lead to ineffective vaccines which cannot be distributed to the public potentially resulting in loss of life.

If we use this kind of bioreactor in Mbarara, Uganda, one thing we need to be concerned about is the power supply; these two areas may sometimes lack mains electricity. Our device will not work without a sufficient and stable power supply, therefore such a power supply is necessary. The other thing that concerns us is that there may be a shortage of well-educated employees in the local area, which means a large amount of money may need to be spent on educating local people or bringing in foreign specialised workers from abroad. Most importantly, the price of the vaccine may be too high for those middle income and low income families, since Uganda is a very underdeveloped country. Therefore even if the vaccine we make can effectively cure and prevent the TB, but this will not change the local health situation to a higher level.

The corresponding group 7 for CEGE-ME have decided to use a hydroelectric dam as a power source for the vaccine plant. The location of the dam could potentially lead affect rural communities and surrounding wildlife which are dependant on the river flow.

In order to provide a reliable source of power, the CEGE-ME team 7 have decided to use a dam. This could potentially lead to affecting rural communities and surrounding wildlife dependant on the river flow.

2 Subsystem Descriptions

2.1 Temperature Control System

Section authors: Jeremy Lo Ying Ping, Waheeb Ameen Hassen

The role of the temperature control system is to maintain a constant temperature environment for biochemical reactions within the bioreactor model as settable by the end user. It consists of a temperature sensing subsystem and a heating subsystem which work in tandem; temperature sensing data is used to control the state of the heating subsystem. The circuit for this subsystem can be found in appendix 4.8 (Fig. 4.8.1).

Maintaining such a constant temperature environment is crucial for many biochemical reactions, especially within the context of a tuberculosis vaccine, as it can affect the rate of reaction and ability to push the reaction to completion. In extreme circumstances, uncontrolled high temperatures could induce the denaturisation of enzymes and other proteins in the bioreactor.

It was required of the temperature control subsystem that 10-20°C water from some inlet source be heated to and maintained at some user settable target temperature in the range of 25°C to 35°C to within $\pm 0.5^\circ\text{C}$. In order to achieve the design specification requirements, the bioreactor contents is to be heated should the temperature as measured by the temperature sensing subsystem be lower than the target temperature.

2.1.1 Temperature Sensing Subsystem

The temperature sensing subsystem finds the temperature of the bioreactor contents from measured data. This is important as temperature values are not only used by the heating subsystem but also by the pH control system. The temperature control system is required to be accurate to within $\pm 0.5^\circ\text{C}$, temperature values found by the temperature sensing subsystem must be accurate to within this range as well.

The temperature sensing subsystem circuit (Fig. 4.6.1) is composed of a $10\text{k}\Omega$ NTC (negative temperature coefficient) thermistor (ND06P00103K) in series with a $10\text{k}\Omega$ resistor which act together as a potential divider. The circuit is powered using the 3.3V DC supply of the MSP432 microcontroller. The microcontroller is then used to measure the potential difference across the $10\text{k}\Omega$ resistor from which a temperature may be calculated.

On the microcontroller, there is an analogue-to-digital converter which takes pin voltages as a proportion of the 3.3V DC supply and quantises these values in the range 0 to 1023; thus, to calculate the voltage in code, we can read the analogue value of the pin and multiply it by . The 3.3V DC supply is divided between the thermistor and the resistor in the share of their resistances, so it is possible to calculate the voltage across the thermistor and thus the resistance of the thermistor at any given time.

$$V_{\text{thermistor}} = V_{\text{total}} - V_{\text{resistor}}$$
$$R_{\text{thermistor}} = \frac{R_{\text{resistor}} \cdot V_{\text{thermistor}}}{V_{\text{total}} - V_{\text{thermistor}}}$$

The Steinhart-Hart equation may be then used to calculate the temperature of a thermistor from the resistance of the thermistor, where, A, B and C are the Steinhart-Hart equation coefficients calculated with three datasheet resistance-temperature pairs as shown in appendix 4.6 (Fig. 4.6.2).

$$T = \frac{1}{A + B \ln R + C (\ln R)^3}$$

2.1.2 Heating Subsystem

The purpose of the heating subsystem is to heat the contents of the bioreactor to within $\pm 0.5^\circ\text{C}$ of a target temperature set by the user through the user interface.

A $3\ \Omega$ heater with a maximum power rating of 30W was available to be used in the heating subsystem; however, the maximum current output of all pins on the microcontroller is only 48 mA, which is insufficient to power the heater. To overcome this problem, a MOSFET controlled by the microcontroller is used to connect the heater to a 12V, 4A battery power source capable of supplying 48W of power. The microcontroller can therefore switch the heater on or off by writing a high or low output to the pin respectively. Both the source pin of the MOSFET and the negative terminal of the 12V battery are connected to a common ground on the microcontroller. The circuit diagram for the heating subsystem can be found in Appendix 4.7 (Fig. 4.7.1).

Since the heating element has a maximum power consumption rating of 30W, pulse-width modulation (PWM) control is used to ensure the maximum average voltage of the system would remain just below 8V and bring the maximum power supplied to the heater by the battery down to also be 30W.

Two heating logic approaches were tested during this project to see which would adhere to the specifications best. The first more naïve approach was to switch the heater on if the current temperature was below the set temperature and off once it reached the set temperature; however, in development and testing, the heater was found to remain hot for a prolonged period after the target temperature was reached, causing significant overshooting of the target temperature and a sawtooth effect as the bioreactor contents heats and cools. This can be seen in Fig. 4.7.2 in appendix 4.7 .

To minimise this effect therefore, the heater power is tapered off as the temperature asymptotically nears the target temperature using PWM control. When the temperature is less than the start of the taper, which was found to be 1.7°C below the target temperature in testing, PWM is used so as to supply the maximum heater power rating to the heater. Between 1.7°C and 0.2°C below the target temperature, the power supplied to the heater is linearly tapered down to zero. Above 0.2°C below the target temperature, the heater is completely switched off. A more detailed flowchart with this logic may be found in appendix 4.7 (Fig. 4.7.3). The results of this approach are also displayed in appendix 4.7 (Fig 4.7.4) when the target temperature was set to 34.5 degrees. These results indicate that the heating system is able to operate within the aforementioned specification for the overall system.

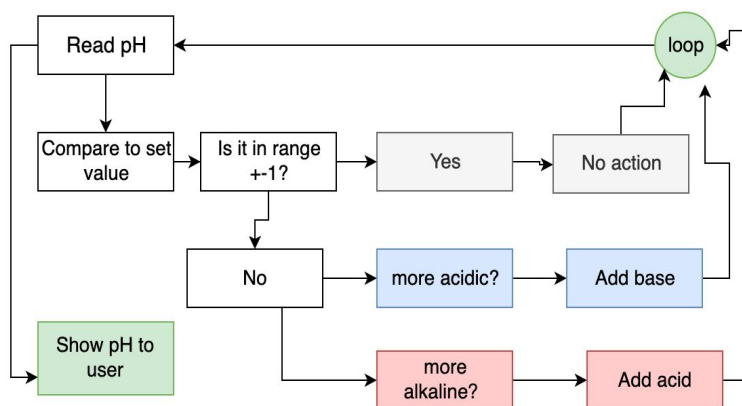
2.2 pH subsystem = pHsensor subsystem + pumps control subsystem

Description

Authors: Huiyan Xiao

The pH subsystem is used to monitor and control the PH of the liquid according to the needs of the user. The pH is calculated through a transfer function shown in Figure 4.4.1.

There is a sensor to monitoring the PH of the liquid and will send pH value to the UI, the system will automatically adjust the PH to match the requirement given by the user. Two pumps will be used to adjust the pH of t.he liquid, one contains acid, and the other contains base. The system will choose to let the pumps working or not in order to adjust the pH.



The temperature can be obtained from the temperature sensor, pass by the codes related to the temperature subsystem. As shown in Figure 2.3, After calculation, we can read the current pH value of the fluid. It will be passed to the UI, and the user can do actions about it. Our program includes if statements, which can judge whether the current pH value within the acceptable range. If not, it will turn on a related pump, and if right, we turn off all the pumps.

Figure 2.2 : Overview of pH subsystem workflow.

2.2.1 pH Probe Subsystem

Author: Xinyao Qian

This part is about designing a circuit that allows the microcontroller to read the current pH value from the pH electrode.

There are two challenges we are facing:

1) The pH electrode initially gives an output voltage, which is a bipolar signal. However, the microcontroller we are using (MSP432P401R) operates on a single supply from the laptop, the signal will have to be shifted to give a unipolar system.

To solve the problem, we designed a circuit with op-amp U1, shown in Figure 2.3.1.1, the voltage divider provides a voltage of 1.6V across R1, due to the high impedance input of the op-amp, the voltage input of the op-amp is also 1.6V. The op-amp U1 is used as a voltage follower, providing 1.6V DC offset to the pH electrode output signal.

2)The pH probe has a high output impedance, while the microcontroller only allows low impedance input.

To give a low impedance output, another op-amp U2 is used, as shown in Figure 2.3.1.2, the output of the op-amp is connected to the microcontroller, giving a low impedance input to the microcontroller while maintaining the shifted signal unchanged.

Then we can integrate these two op-amps, the voltage output of the pH electrode will be level shifted, and the low impedance signal will be read by the microcontroller for further calculations.

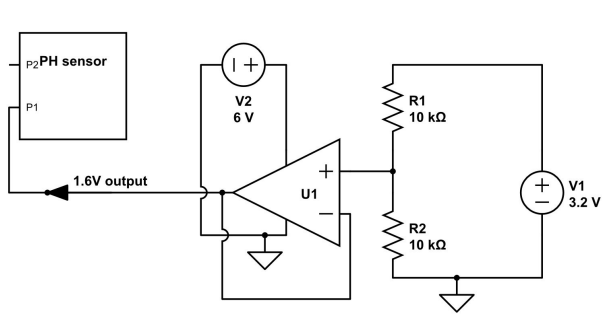


Figure 2.2.1.1 : Op-amp U1 used as a voltage follower to provide 1.6V DC offset to the output of pH probe.

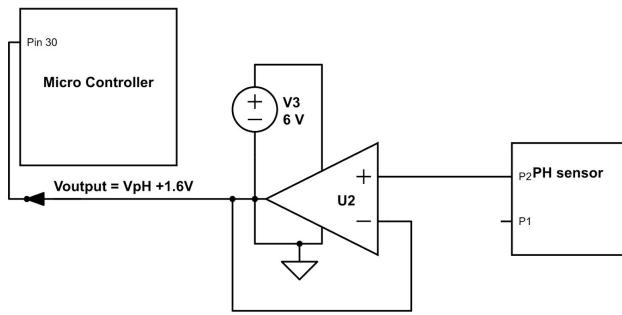


Figure 2.2.1.2 : Op-amp U2 used as a voltage follower to reduce the high impedance of the pH probe.

2.2.2 Pumps Control Subsystem

Author: Xinyao Qian

This part is about designing a circuit that allows the microcontroller to control the pumps. The pumps draw a large current which exceeds the current limit of the microcontroller, so the pumps are powered by external voltage supply.

The 7030BL transistors are used as a switch to control each pump independently, as shown in Figure 2.3.2.1. The microcontroller will control the gates of the transistors. However, the pumps are running too fast with a given power supply of 6V, giving a sharp change of pH when they are on. To control the speed of the pumps, we used PWM (Pulse Width Modulation). “analogWrite()” is used to give a 48.8% duty cycle. As a result, the pumps operated at an optimal speed.

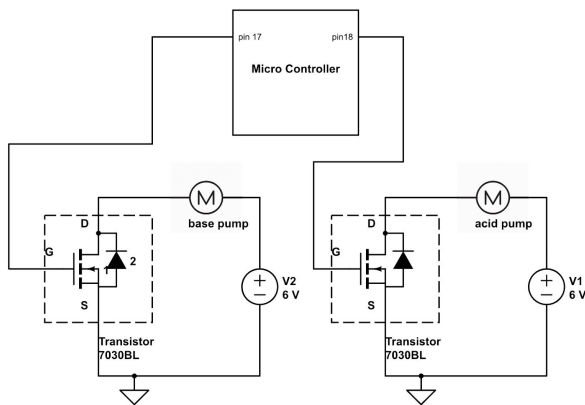


Figure 2.2.2.1 : Transistors controlled by the microcontroller that are used to turn on/off the solution pumps with external power supply.

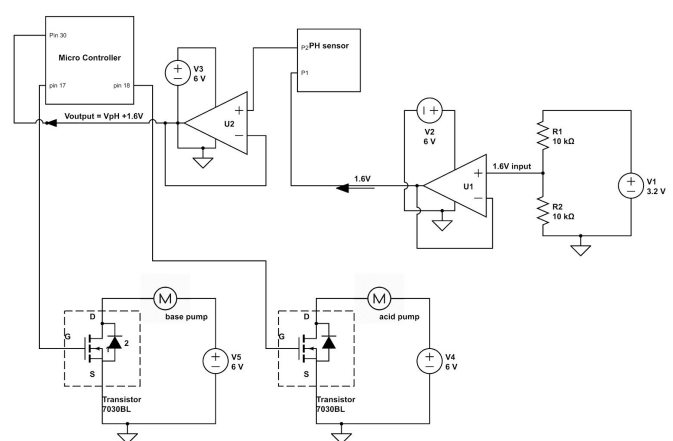


Figure 2.2.3.1 : Completely Integrated pH subsystem

2.2.3 pH Subsystem circuit overview

Author: Huiyan Xiao

Finally, the pH electrode system and pumps control system are combined, as shown in Figure 2.3.3.1, the pH subsystem can complete the workflow in Figure 2.3 with a given temperature input. Our overall results meet the specification requirements after testing. The testing results are shown in appendix.

2.3 Stirring Subsystem = RPMsensor subsystem + RPMcontrol subsystem

Authors: Thowhid Ahmed, Xiang Long and Rafael Asensio Delgado

Description:

The stirring subsystem is to allow users to monitor and adjust stirring speed through the UI. The system's specifications allow the user to input a stirring speed between 500 and 1500 RPM, maintaining that velocity within a ± 20 RPM range.

This subsystem is composed of two main parts: monitoring and adjusting the speed. The speed is monitored by using a photo interrupter, a component that emits an infrared light signal and calculates the distance to an object based on its reflection by outputting a value of voltage. The speed of the motor is varied by inputting a voltage through an external power supply and controlling it with the use of a microcontroller and a transistor. If the value read is beyond the specified range, then the microcontroller will either increase or decrease the voltage supplied to the motor (as seen in Fig. 2.3.2). Both the motor and the photo interrupter are powered by a three volt power supply.

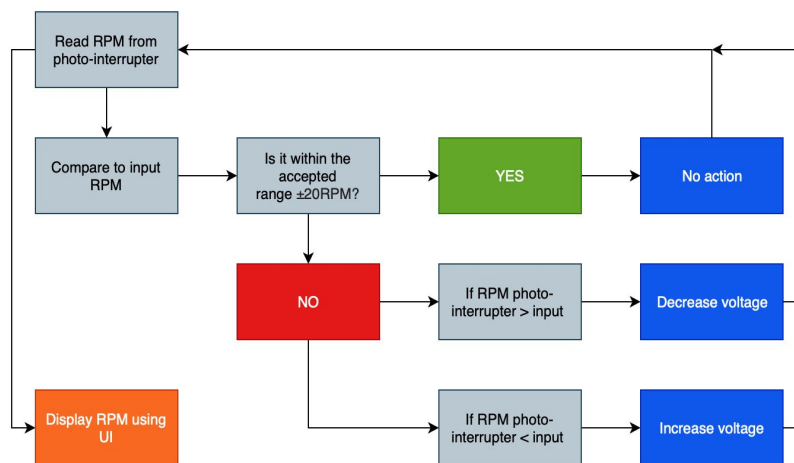
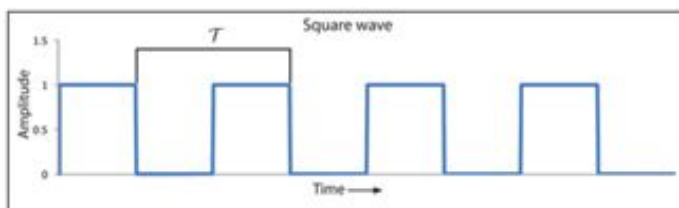


Figure 2.3.2 - Stirring subsystem workflow

2.3.1 RPM Calculation

To be able to calculate the RPM, one piece of equipment we utilised was a CP2S700HCP photo-interrupter. A photo-interrupter can be broken down into two components, the LED and a photodetector. The LED fires a beam upwards and the spinning motor blade will cut the beam periodically. The photodetector will measure how much of the beam is reflected and we can use this as an optical switch. The output of the photo-interrupter will generate a square wave in which we can calculate the RPM by working out the period of the wave, calculating the frequency and then converting that into an RPM.



RPM Calculation

Let T be the period, F be the frequency and R be rpm.

$$F = (1/T)/2$$

$$R = F * 60$$

We divide the frequency by two due to the fact that the motor blade had two fins, so each time a peak would be generated on the square wave, it would only be half a revolution so the frequency was two times greater than what it should've been.

Upon carrying out testing from the RPM calculations, it became apparent that the values didn't match the tachometer (Fig 4.9.7). To visualize the issue, we analysed the raw output values of the frequency and from the results of Fig 4.9.4 we identified that the frequency being calculated was not consistent. The issue was pinpointed down to the photo-interrupter returning noisy data. To denoise the data, we omitted any frequency value over 1000Hz and from Fig 4.9.5 and with further testing with the tachometer (Fig 4.9.8) it is clear that the frequency returned was more reasonable.

2.3.2. Speed Control

The speed control subsystem is fairly basic in principle. The user inputs a value through the UI in RPM, which is then converted to a value between 1 and 255 to adjust the voltage supplied to the motor (4.10). Since we want to use a range of different RPM speeds, we use pulse-width modulation to allow us to reach any RPM we want. These values for the voltage were calibrated in order to match the desired RPM. The motor was set to have an operating speed of 700 RPM, so that its velocity changes faster when started, since at some of the lower values (500 - 600 RPM) the motor had trouble starting from rest.

After the operating speed is applied, the microcontroller compares the value read from the photo interrupter to that inputted by the user. If this value is higher the microcontroller will decrease the voltage supplied to the motor (through the transistor). If the value is lower, the voltage supplied will increase. This is the main function of the transistor on our circuit (see Fig. 2.5.2).

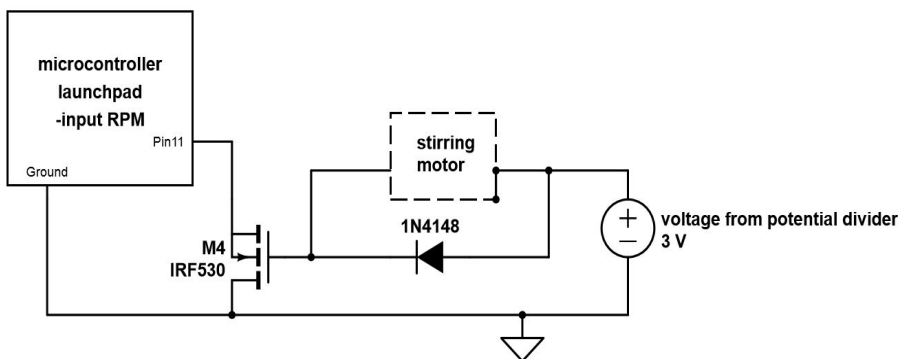


Fig 2.5.2

Moreover, when the motor rotates it generates a back emf, opposite in direction to the supplied voltage. If the current generated by this emf goes back to the transistor this component could be damaged, which is why we decided to put a diode in parallel with the motor, with the cathode in the direction of the voltage supplied (see Fig. 2.5.2).

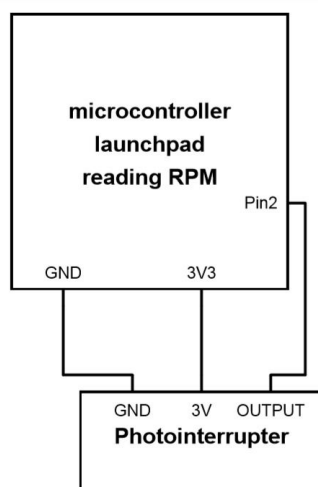


Fig 2.5.3

2.4 User Interface

Author: Kelly Ding

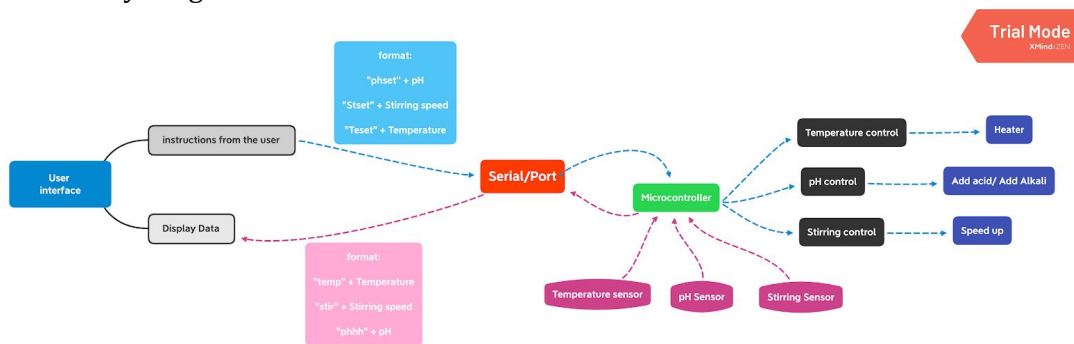


Figure 2.4 Overall design of the UI.

The user interface is there in Figure 2.4. to allow user to input the temperature, stirring speed and pH value they want the bioreactor to present. It is implemented based on serial communication. The user interface consists of 2 parts: User input and data output.

The user interface takes input from the user and output the user-input settings to the serial. Then the microcontroller pick up the strings from the serial and according to this data, the microcontroller will send instructions to different micro pins according to the first 5 characters in the string. The microcontroller collects data from the sensors and format the data into a string which its initial 5 characters indicate that which category the data belongs to (temperature, stirring or pH). The user interface picks up string from the serial and then figure out the category of the data. Then it displays the data in the graph.

(Code for this part can be found at Appendix 4.2 and 4.3)

2.4.1 User input data

Author: Kelly Ding

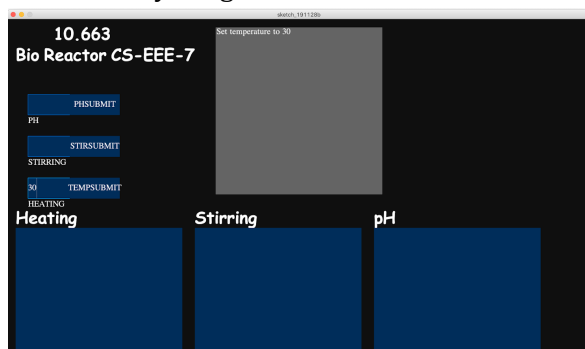


Figure 2.4.1 Input Data

The user interface screen allow user to input data in the text field provided on the screen, the user click the box next to the text field when the data is ready to be submitted. Then the data will be displayed on the console message area on the screen.

At the same time, the data will be formatted into a string with the specific heading which indicates the category of the data and this string is written into the serial. Then the microprocessor picks this message up from the serial and find out the category of the data and send the instructions to the pin on the microcontroller based on the data received. The string heading of different category is the following: setting temperature - "Tset", setting ph - "phset" and setting stirring speed - "stset".

For example, the user wants to set the temperature to 30 °C. Then he types the value 30 into the text area for temperature and click the submission button next to it. Then a text message “Set temperature to 30” will display on the console screen. At the same time, the microcontroller sends the string “Treset30” to the console. (As figure 2.4.1 shown above.)

The microcontroller picks up the string “Treset30” and by substring comparison, it could find out that the user wants the temperature to be set to 30 degrees. After comparing the 30 °C with the current temperature(say 20 °C), then the microcontroller will turn on the heater and heat up the water until the water reaches the 30 °C .

2.4.2 Display Data

Author: Kelly Ding

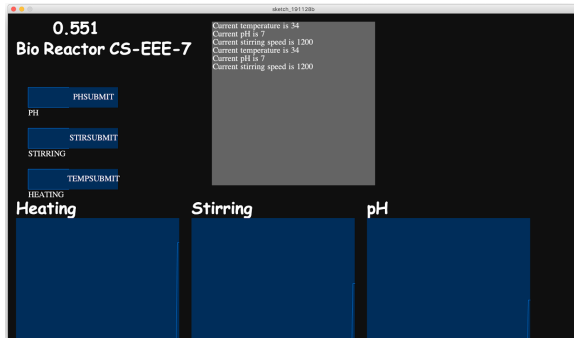


Figure 2.4.2 Read Data

For every cycle the microcontroller is working, it fetches data from the microcontroller and write it to the serial with headings indicate the category of the data. Then the computer gets data from the serial and display the data both in the form of text and graph. The string heading of different category is the following: temperature - “temp”, setting ph - “phhh” and stirring speed - “stir”.

For example, the current temperature in the system is 34 °C so the microcontroller will send the string “temp34” to the serial. When the computer picks up the string “temp34” from the string and it knows that the current temperature in the system is 34 °C , so it displays the value onto the graph for temperature and display the message “Current temperature is 34” on the console message area. (As figure 2.4.2 shown above.)

2.4.3 User Interface Example Result

Author: Kelly Ding

The graph on the right is an example of a graph which the user interface can generate from bioreactor data.

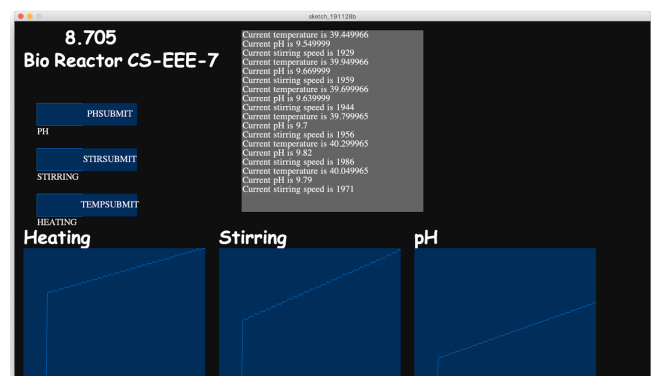


Figure 2.4.3 Overall Result

2.4.4 User Interface application in real word problems

Author: Kelly Ding

The user interface allows scientists to configure system parameters without modifying the circuit, code or architecture the bioreactor. The user interface allows the scientists to change the environment inside the bioreactor easily, read precise data from the system and observe real-time changes of the conditions inside the system, which can speed up their investigation and research processes.

3 Overall System Integration and Summary

Our team was unable to integrate all of the systems together, however, we were able to get each individual system working within the required criteria. The error in integrating the systems together was due to the UI system being unable to fetch and send strings correctly leading to overflow meaning incorrect data was displayed when looking at the real time results in the UI.

The stirring subsystem individually was able to reach the target RPM and maintain ± 20 RPM in 4.07 seconds (for 1500 RPM) reaching an average accuracy of 99.69% over the three trials at 500, 1000 and 1500 RPM (seen in figures 4.9.1, 4.9.2 and 4.9.3). The system was able to maintain a higher accuracy at 500 and 1000 RPM (100%) compared to 1500 RPM (99.07%) due to friction in the motor which increased at higher RPM values causing fluctuations in the RPM values, could be fixed by tapering the voltage on approach to the target RPM. When integrated into the overall system the stirring system took significantly longer to reach the target RPM as the stirring system was required to wait for the UI response.

The heating subsystem individually was able to hit the given target temperature and maintain that temperature. A target temperature of 34.5°C was met within 72.5 seconds (seen in figure 4.7.4) from a starting temperature of roughly 31.3°C and maintain within the $\pm 0.5^{\circ}\text{C}$ as specified in the bioreactor requirements therefore, the heater had 100% accuracy after it has reached the target temperature. When integrated into the overall system heating didn't work as intended as the temperature didn't rise fast enough to be considered functional. This was due to parallelisation issues within the UI code interfering with the voltage output to the heater. This was due to the code for the subsystems not being written to be run parallel to each other from the ground up.

The pH subsystem individually was able to meet the target pH given and maintain that pH through changing temperature (as temperature influences pH) and stirring (as the addition of acid or base may not have been mixed, therefore, not picked up by the pH probe). The pH system was able to hit a target pH of 4 from a pH of 7 within 30 seconds (whilst stirring is operating and temperature is constant). As the systems were not integrated correctly in time we were not able to test the pH system whilst changing the temperature and stirring speed to see how these changes impact the pH system.

Overall, as we did not manage to integrate the system something to consider for further projects would be to coordinate more together before we start creating our systems. The key reason the systems did not integrate was due the code, to avoid this in future we should we should consider agreeing between the subsystems how the code will be written e.g. naming conventions, units, data types etc. However, all system did manage to work correctly independently and with a possibly another session with the equipment could be working.

4 Appendix

4.1 Appendix: User Interface code for Processing

```
import controlP5.*;
import processing.serial.*;

PFont font1;
PFont font2;
ControlP5 cp5;

Timer startTimer;
float time;

Serial myport;

Textarea console;
Chart heatingChart;
Chart StirringChart;
Chart pHChart;

void setup(){
  size(1400,800);
  font1 = loadFont("ComicSansMS-Bold-40.vlw");
  font2 = loadFont("Serif-20.vlw");
  myport = new Serial(this, "/dev/cu.usbmodemM43210051", 9600);
  myport.bufferUntil('\n');
  startTimer = new Timer(0);
  cp5 = new ControlP5(this);
  textFont(font1);

  // Input text field and submission button
  cp5.addTextfield("pH").setPosition(50,180).setSize(200,50).setAutoClear(false).setFont(font2);
  cp5.addTextfield("Stirring").setPosition(50,280).setSize(200,50).setAutoClear(false).setFont(font2);
  cp5.addTextfield("Heating").setPosition(50,380).setSize(200,50).setAutoClear(false).setFont(font2);
  cp5.addButton("pHSubmit").setPosition(150,180).setSize(120,50).setFont(font2);
  cp5.addButton("StirSubmit").setPosition(150,280).setSize(120,50).setFont(font2);
  cp5.addButton("TempSubmit").setPosition(150,380).setSize(120,50).setFont(font2);

  //console
  console = cp5.addTextarea("console")
    .setPosition(500, 20)
    .setSize(400, 400)
    .setCaptionLabel("")
    .setFont(font2)
    .setColorBackground(100)
    .scroll(1)
    .hideScrollbar();

  //heating chart
  heatingChart = cp5.addChart("temperatureChart")
    .setPosition(20, 500)
    .setSize(400, 400)
    .setRange(0, 40) //nead change
    .setView(Chart.LINE)
```

```

        .setCaptionLabel("")
        .setStrokeWeight(10);
heatingChart.addDataSet("Heating");
heatingChart.setData("Heating", new float[100]);

//stirring chart
StirringChart = cp5.addChart("StirringChart")
    .setPosition(450, 500)
    .setSize(400, 400)
    .setRange(0, 3000) // nead change
    .setView(Chart.LINE)
    .setCaptionLabel("")
    .setStrokeWeight(10);
StirringChart.addDataSet("Stirring");
StirringChart.setData("Stirring", new float[100]);

//stirring chart
pHChart = cp5.addChart("pHChart")
    .setPosition(880, 500)
    .setSize(400, 400)
    .setRange(0, 14) //nead change
    .setView(Chart.LINE)
    .setCaptionLabel("")
    .setStrokeWeight(1.5)
    .setColorCaptionLabel(0);
pHChart.addDataSet("pH");
pHChart.setData("pH", new float[100]);
}

void addConsoleMsg (String newMsg) {
    String s = console.getText();
    s += newMsg + "\n";
    console.setText(s);
}

void draw(){
    background(15);
    text("Bio Reactor CS-EEE-7",20,100);
    startTimer.countup();
    text(startTimer.getTime(),100,50);
    text("Heating", 20, 490);
    text("Stirring", 450, 490);
    text("pH", 880, 490);

    getdata();
}

void pHSubmit(){
    String text = cp5.get(Textfield.class,"pH").getText();
    myport.write("phset"+ text + "\n");
    addConsoleMsg("Set pH to " + text);
}

void StirSubmit(){
    String text = cp5.get(Textfield.class,"Stirring").getText();
    myport.write("Stset"+text + "\n");
    addConsoleMsg("Set Stirring to " + text);
}

```

```

void TempSubmit(){
    String text = cp5.get(Textfield.class,"Heating").getText();
    myport.write("Treset"+text+"\n");
    addConsoleMsg("Set temperature to " + text);
}

void getdata(){
    if(myport.available()>0){
        String data = trim(myport.readStringUntil('\n'));
        println(data);
        if(data!= null && data.length()>4){
            if(data.substring(0,4).equals("temp")){
                addConsoleMsg("Current temperature is " + data.substring(4));
                heatingChart.push("Heating", float(data.substring(4)));
            }else if(data.substring(0,4).equals("phhh")){
                addConsoleMsg("Current pH is " + data.substring(4));
                pHChart.push("pH", float(data.substring(4)));
            }else if(data.substring(0,4).equals("stir")){
                addConsoleMsg("Current stirring speed is " + data.substring(4));
                StirringChart.push("Stirring", float(data.substring(4)));
            }
        }
    }
}

void serialEvent(Serial myport) {
    if(myport.available()>0){
        String data = trim(myport.readStringUntil('\n'));
        println(data);
        if(data!= null && data.length()>4){
            if(data.substring(0,4).equals("temp")){
                addConsoleMsg("Current temperature is " + data.substring(4));
                heatingChart.push("Heating", float(data.substring(4)));
            }else if(data.substring(0,4).equals("phhh")){
                addConsoleMsg("Current pH is " + data.substring(4));
                pHChart.push("pH", float(data.substring(4)));
            }else if(data.substring(0,4).equals("stir")){
                addConsoleMsg("Current stirring speed is " + data.substring(4));
                StirringChart.push("Stirring", float(data.substring(4)));
            }
        }
    }
    myport.write("1\n");
}

```

4.2 Appendix User Interface code for Energeria

```
int user_ph = 7;

//stirring group constants
#define MOTOR 11 //LED 1 ON IEP BOARD
#define SQ 2 //Square Wave Pin
#define SIZE 50
int desiredVal = 600;
int previousVal = 0;
int startTime = millis();//time at a peak
float t = 0;//period
float frequency = 0;
float rpm = 0;
float adjustedRPM = 0;
float averageRPM = 0;
float rpms[SIZE] = {0, 0, 0, 0, 0};
int count = 0;
int previousRPM = 0;
int time_high;
int time_low;
int starting = 1;

//ph constants
const int PHin = 30;// to get voltage passed the op-amp
const int base = 17; //
const int acid= 18;
int counter_add_acid = 0;
int counter_add_base = 0;
int lag = 0;

//heating constants
static const uint8_t THERMISTOR = A4;
static const uint8_t HEATER = 31;
double V_total = 3.3;
double R_resistor = 9955;
double SH_A = 1.278808880E-3;
double SH_B = 2.171660200E-4;
double SH_C = 0.9603609520E-7;
static const int heater_max_pwm = 170;
static const double heater_start_taper = 1.7;
static const double heater_end_taper = 0.2;

double target_temperature = 10;

void setup() {
  Serial.begin(9600);
  while(!Serial){}
  //stirring setup
  pinMode(MOTOR, OUTPUT);
  pinMode(SQ, INPUT_PULLUP);
  adjustedRPM = 500;
  //ph setup
  pinMode(acid, OUTPUT);
  pinMode(base, OUTPUT);
  pinMode(PHin, INPUT);
  //heating setup
  pinMode(HEATER, OUTPUT);
```



```

}

void heating(){
    double temperature = get_temperature();

    int heater_pwm = 0;

    if (temperature < target_temperature - heater_end_taper) {

        heater_pwm = temperature < target_temperature - heater_start_taper ?
            heater_max_pwm : map(temperature * 100, (target_temperature - heater_start_taper) * 100, (target_temperature -
heater_end_taper) * 100, heater_max_pwm, 0);
    }

    analogWrite(HEATER, heater_pwm);
}

double get_temperature(){
    double V_thermistor = V_total - analogRead(THERMISTOR) * V_total / 1023.0;
    double R_thermistor = (R_resistor * V_thermistor) / (V_total - V_thermistor);
    double ln_R = log(R_thermistor);
    float T = 1/(SH_A + SH_B * ln_R + SH_C * ln_R * ln_R * ln_R) - 273.15;
    return T;
}

float get_rpm(){
    return averageRPM;
}

int setRPM(int value){
    if ((value >= 500) || (value <=1500)){
        desiredVal = value;
    }
    return desiredVal;
}

float get_ph(){
    // constant values
    const int pH = 7;    // pH of standard solution
    const float Es = 1.67;    // electric potential at reference or standard electrode
    const float F = 9.6485309*10000;    // faraday constant
    const float R = 8.314510;    // the universal gas constant
    const float ln10 = 2.30258509;

    // initialise
    float ReadPH = 0.0;
    double Temp = 0.0;
    float pH_with_tp = 0.0; // PH calculated with known temperature
    float new_readPH = 0.0;

    // read the Voltage from the pins:
    ReadPH = analogRead(PHIn); // since the op-amp shifted the signal by 512mV, we did the reverse.
    new_readPH = map(ReadPH, 0, 1023, 0, 3350);
    new_readPH = new_readPH / 1000;
    Temp = get_temperature();    // get the temperature value from tp sensor
    // print the results to the serial monitor:
    pH_with_tp = (((Es - new_readPH) * F) / (R * (Temp + 273.15) * ln10)) + pH;
    return pH_with_tp;
}

```

```

void setPH(int ph)
{
    int PHinput = ph; /* User input PH needed */ // initialise the PH that was given by the user
    String str = "phin" + String(PHinput) + "\n";
    Serial.write(str.c_str());
    int PHmax = PHinput + 1;
    int PHmin = PHinput - 1;

    int currentPH = get_ph();

    if (currentPH > PHmax && lag <= 2) // wait 3 seconds to see if the value is continuously greater than we expected
    {
        lag += 1;
    }
    else if (currentPH > PHmax && lag > 2) //the solution need acid
    {
        if (counter_add_acid < 500)
        {
            counter_add_acid += 200;
        }
        analogWrite(acid, counter_add_acid);
        digitalWrite(base, LOW);
    }
    else if (currentPH < PHmin && lag <= 2) // wait 3 seconds to see if the value is continuously lower than we
    expected
    {
        lag += 1;
    }
    else if (currentPH < PHmin && lag > 2) // the solution need base
    {
        if (counter_add_base < 500)
        {
            counter_add_base += 200;
        }
        analogWrite(base, counter_add_base);
        digitalWrite(acid, LOW);
    }
    else // we are in acceptable range
    {
        digitalWrite(acid, LOW);
        digitalWrite(base, LOW);
        counter_add_acid = 0;
        counter_add_base = 0;
        lag = 0;
    }
    // delay(100); // wait 100 milliseconds before the next loop
}

void loop() {
    // get user data
    String s = Serial.readString();
    Serial.println(s);
    if (s[0] == 'T'&&s[1] == 'e'&&s[2] == 's'&&s[3] == 'e'&&s[4] == 't'){
        String temp = s.substring(5);
        target_temperature = temp.toFloat();
        Serial.print(temp);
    } else if (s[0] == 'S'&&s[1] == 't'&&s[2] == 's'&&s[3] == 'e'&&s[4] == 't'){
        String stir = s.substring(5);
        setRPM(stir.toInt());
    }
}

```

```

    Serial.print(stir);
} else if (s[0] == 'p' && s[1] == 'h' && s[2] == 's' && s[3] == 'e' && s[4] == 't'){
    String ph = s.substring(5);
    user_ph = ph.toInt();
    setPH(user_ph);
    Serial.print(ph);
}

heating();
speedup();
setPH(user_ph);
// get system data
float t = get_temperature();
String st = String("temp") + String(t) + String("\n");
Serial.write(st.c_str());
float t2 = get_rpm();
String st2 = String("stir") + String(t2) + String("\n");
Serial.write(st2.c_str());
float t3 = get_ph();
String st3 = String("phhh") + String(t3) + String("\n");
Serial.write(st3.c_str());
}

//stirring
//analogue read maps volatages 0-3.3v into integer values between 0-1023. Analogue to Digital Converter
int removeNoise(int rpmVal, int prevRPM){
    if (rpmVal < 1500){
        return rpmVal;
    }
    return prevRPM;
}

float getAverage() {
    rpms[count] = rpm;
    count++;
    if (count >= SIZE) {
        count = 0;
    }
    float sum = 0;
    for (int i = 0; (i <= SIZE); i++) {
        sum = sum + rpms[i];
    }
    return sum;
}

void startRoutine(){
    while(millis() - startTime < 10000){
        analogWrite(MOTOR, map(700, 500, 1500, 1, 255));
    }
    starting = 0;
}

void speedup(){
    /*
    if (starting){
        startRoutine();
    }*/
    time_high = pulseIn(SQ, HIGH);
    time_low = pulseIn(SQ, LOW);
    t = (time_high + time_low)/2;

```

```

rpm = removeNoise((300000/t)*60, previousRPM);
averageRPM = getAverage() / SIZE;
previousRPM = (300000/t)*60;

if (rpm > (desiredVal)) {
    adjustedRPM -= 5;
}
if (rpm < (desiredVal)) {
    adjustedRPM += 5;
}
analogWrite(MOTOR, map(adjustedRPM, 0, 2000, 1, 255));
}

```

4.3 Appendix: pH

Author: H. Xiao, X.Qian

The transfer function of the pH electrode is:

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

where

- $\text{pH}(X)$ = pH of unknown solution(X)
- $\text{pH}(S)$ = 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 = $9.6485309 \times 10^4 \text{ C mol}^{-1}$,
- R is the universal gas constant = $8.314510 \text{ J K}^{-1} \text{ mol}^{-1}$
- T is the temperature in Kelvin

Figure 4.4.1 : Transfer function of the pH

Codes for subsystem PH

```

/*
This one is for the pH part
Pin 33 (P5.1) to P1 to get value from pH sensor
Pin 17 (P5.7) to L4 to control base pump
Pin 19 (P2.5) to L3 to control acid pump
*/

const int PHin = 30; // to get voltage passed the op-amp
const int base = 17; //
const int acid = 18; //
    // value read from the pot

void setup() {
    // initialise serial communications at 9600 bps:
    Serial.begin(9600);
    pinMode(acid, OUTPUT);
    pinMode(base, OUTPUT);
    pinMode(PHin, INPUT);
}

```

```

int counter_add_acid = 0;
int counter_add_base = 0;
int lag = 0;

//to set up a loop

int getPH()
{
    // constant values
    const int pHs = 7;    //pH of standard solution
    const float Es = 1.67; // electric potential at reference or standard electrode
    const float F = 9.6485309*10000; // faraday constant
    const float R = 8.314510; // the universal gas constant
    const float ln10 = 2.30258509;

    // initialise
    float ReadPH = 0.0;
    double Temp = 0.0;
    float pH_with_tp = 0.0; //PH calculated with known temperature
    float new_readPH = 0.0;

    // read the Voltage from the pins:
    ReadPH = analogRead(PHIn); //since the op-amp shifted the signal by 512mV, we did the reverse.
    new_readPH = map(ReadPH, 0, 1023, 0, 3350);
    new_readPH = new_readPH / 1000;
    Serial.println("new_readPH");
    Serial.println(new_readPH);

    Temp = 25.0; /* getTemperature(); */ // get the temperature value from tp sensor
    // print the results to the serial monitor:

    Serial.println("Temperature:");
    Serial.println(Temp);

    pH_with_tp = (((Es - new_readPH) * F) / (R * (Temp + 273.15) * ln10)) + pHs;

    Serial.println("PHValue:");
    Serial.println(pH_with_tp);
    return pH_with_tp;
}

void setPH()
{
    int PHinput = 7 /* User input PH needed */; // initialise the PH that was given by the user
    int PHmax = PHinput + 1;
    int PHmin = PHinput - 1;

    int currentPH = getPH();

    if (currentPH > PHmax && lag <= 10) // wait 3 seconds to see if the value is continuously greater than we expected
    {
        lag += 1;
    }
    else if (currentPH > PHmax && lag > 10) //the solution need acid
    {
        if (counter_add_acid < 500)
        {
            counter_add_acid += 50;
        }
        analogWrite(acid, counter_add_acid);
        digitalWrite(base, LOW);
    }
    else if (currentPH < PHmin && lag <= 10) // wait 3 seconds to see if the value is continuously lower than we expected
    {
        lag += 1;
    }
}

```

```

}
else if (currentPH < PHmin && lag > 10)    // the solution need base
{
    if (counter_add_base < 500)
    {
        counter_add_base += 50;
    }
    analogWrite(base, counter_add_base);
    digitalWrite(acid, LOW);
}
else    // we are in acceptable range
{
    digitalWrite(acid, LOW);
    digitalWrite(base, LOW);
    counter_add_acid = 0;
    counter_add_base = 0;
    lag = 0;
}
delay(100);    // wait 100 milliseconds before the next loop
}

void loop()
{
    setPH();
}

```

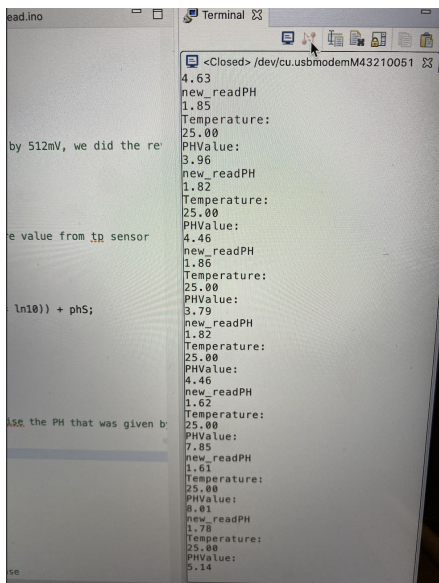
4.4 Appendix: PH Subsystem Testing Results

Author: H. Xiao

We use the IEP test board to simulate the PH change at the very first. The PH voltage can be changed by turning the button. For the temperature, we input different temperature values manually in the code. The result matched the expectation.

Then we used real solutions with different PH to test.

We set the temperature to 25 and tried a solution with a PH of 4. Then we got a value without logic.



The screenshot shows an Arduino IDE window with a terminal output. The terminal displays a series of sensor readings. The first few lines show a pH value of 4.63, followed by a new read of 1.85. Subsequent lines show a temperature of 25.00 and a pH value of 3.96. This pattern repeats with a new read of 1.82, temperature of 25.00, and pH value of 3.96. The terminal output continues with a new read of 1.86, temperature of 25.00, and pH value of 3.79. This is followed by a new read of 1.82, temperature of 25.00, and pH value of 4.46. The terminal output then shows a new read of 1.62, temperature of 25.00, and pH value of 7.85. This is followed by a new read of 1.61, temperature of 25.00, and pH value of 8.01. The terminal output ends with a new read of 1.78, temperature of 25.00, and pH value of 5.14.

As the picture shows, the PHValues make no sense.

In order to get the real PH of the liquid, we used PH-meter to test our solution.



As the picture shows, the real PH of the liquid is 4.27.

```

Temperature:
25.00
PHValue:
3.96
new_readPH
1.85
Temperature:
25.00
PHValue:
3.96
new_readPH
1.86
Temperature:
25.00
PHValue:
3.79
new_readPH
1.86
Temperature:
25.00
PHValue:
3.79
new_readPH
1.85
Temperature:
25.00
PHValue:
3.96
new_readPH
1.85
Temperature:
25.00
PHValue:
3.96
new_readPH
1.85
Temperature:
25.00

```

Overall, the reading is quite close to the real PH value, and the result meets the requirements.

4.5 Appendix: Temperature Sensing Subsystem

Figure 4.6.1: temperature sensing circuit.

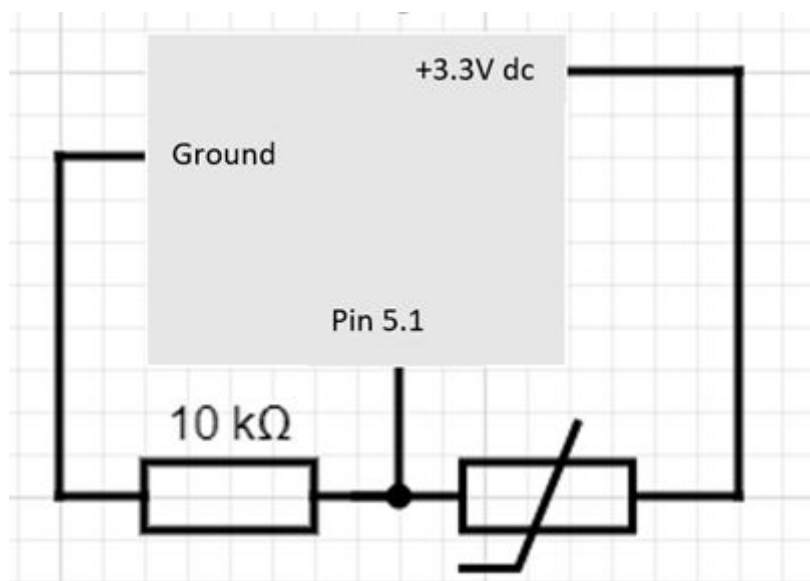
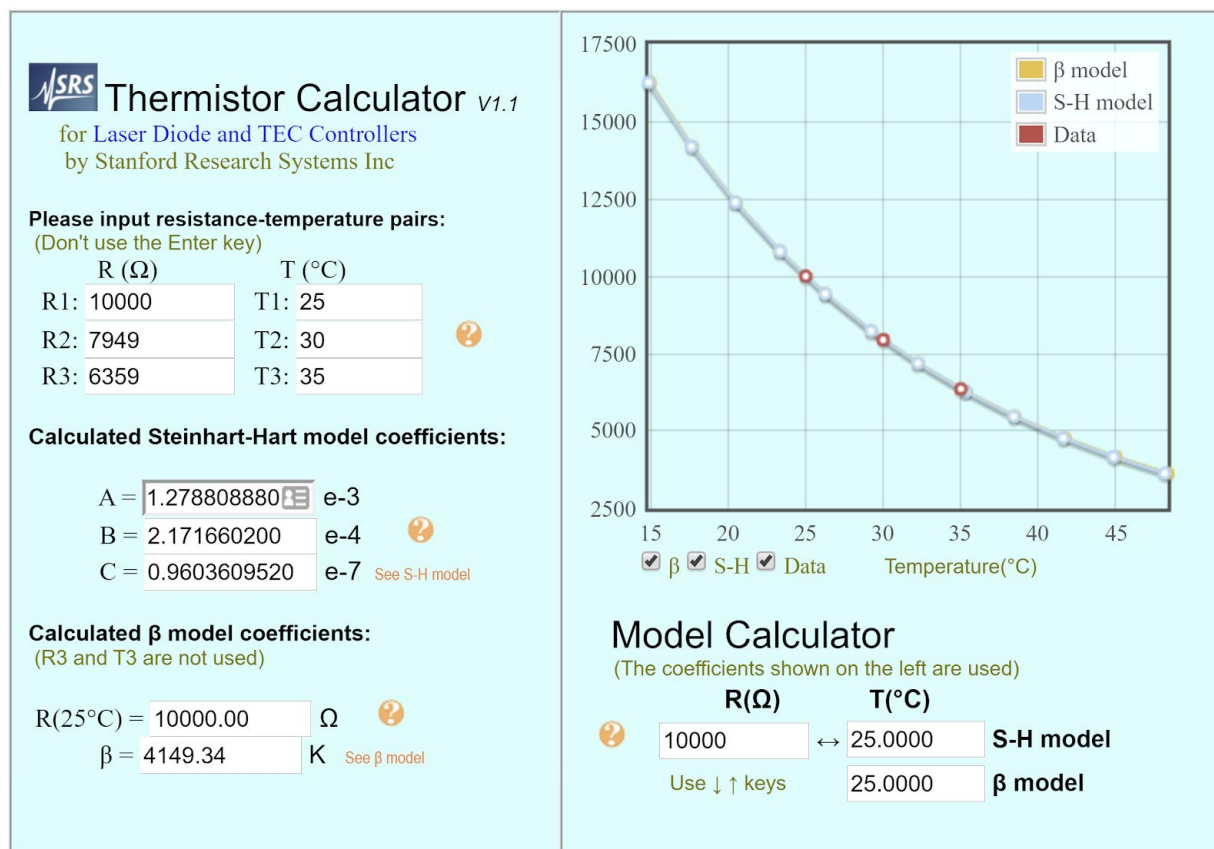


Figure 4.6.2: Steinhart-Hart equation coefficient calculations.



<https://www.thinksrs.com/downloads/programs/therm%20calc/ntccalibrator/ntccalculator.html>

4.6 Appendix: Heating Subsystem

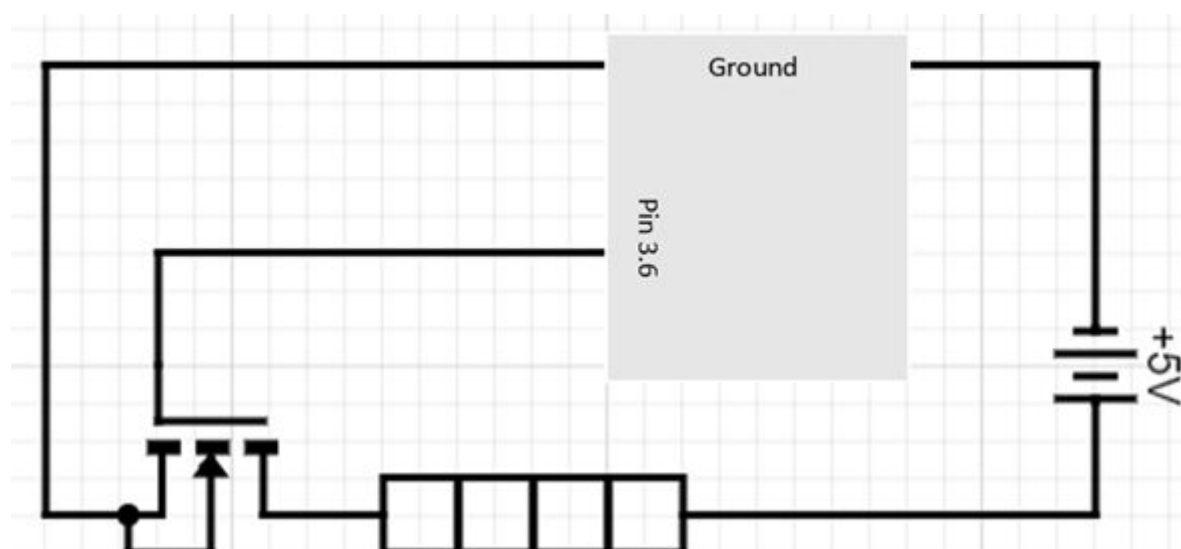


Figure 4.6.1: Heating subsystem circuit

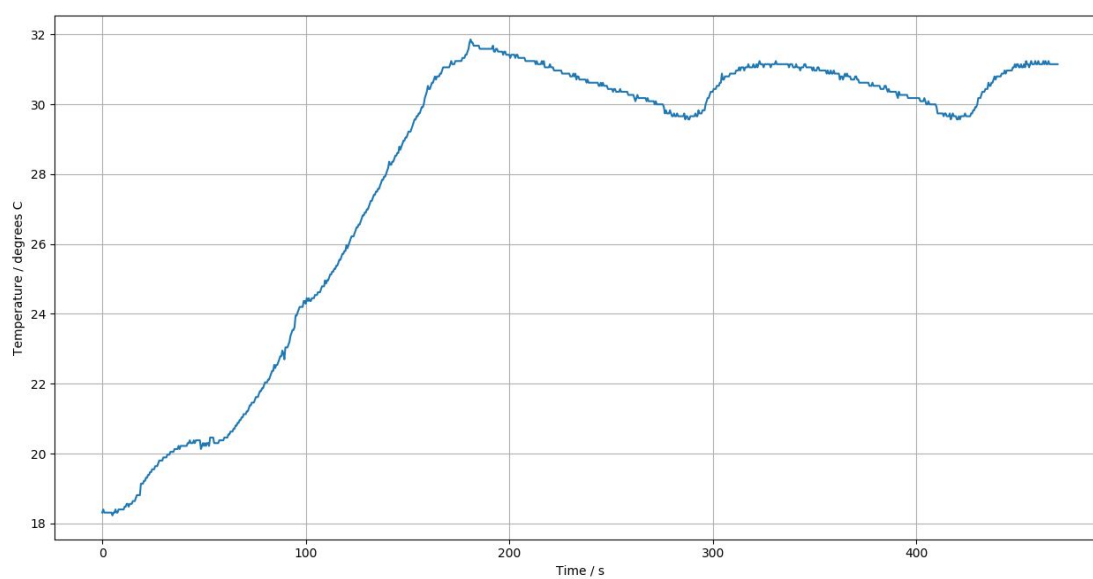


Figure 4.6.2: heating to a target temperature of 30°C using a more naive approach to the heating logic.

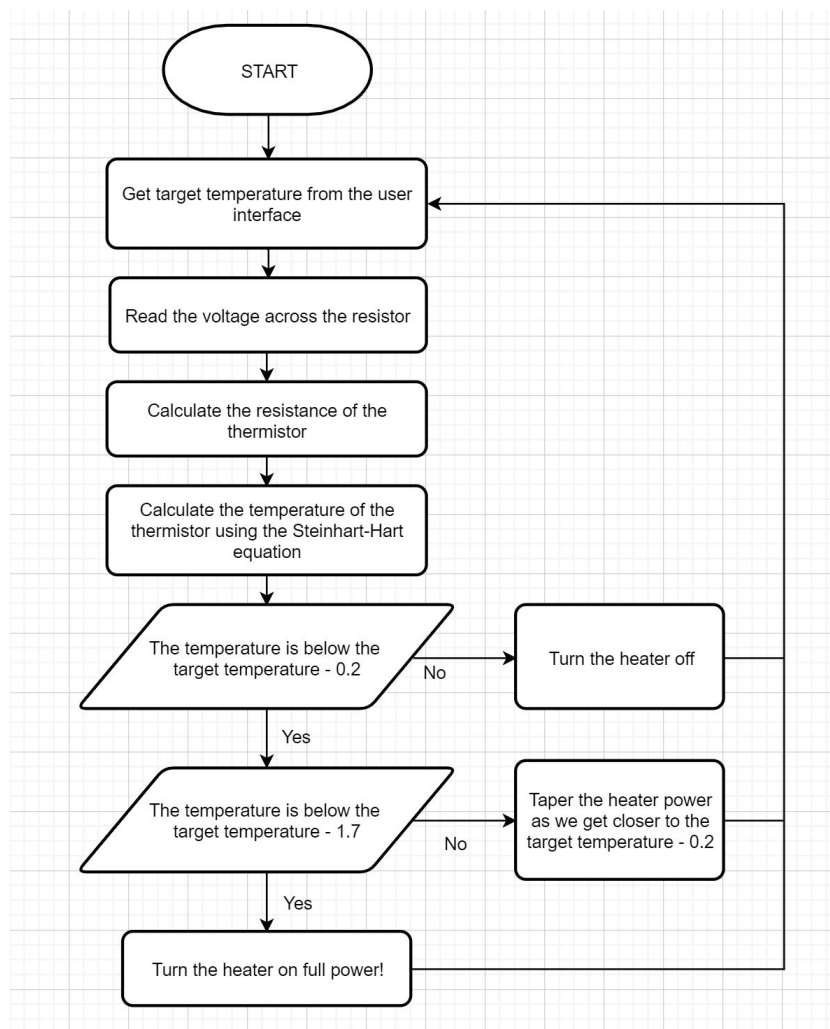


Figure 4.6.3: improved heating logic flowchart.

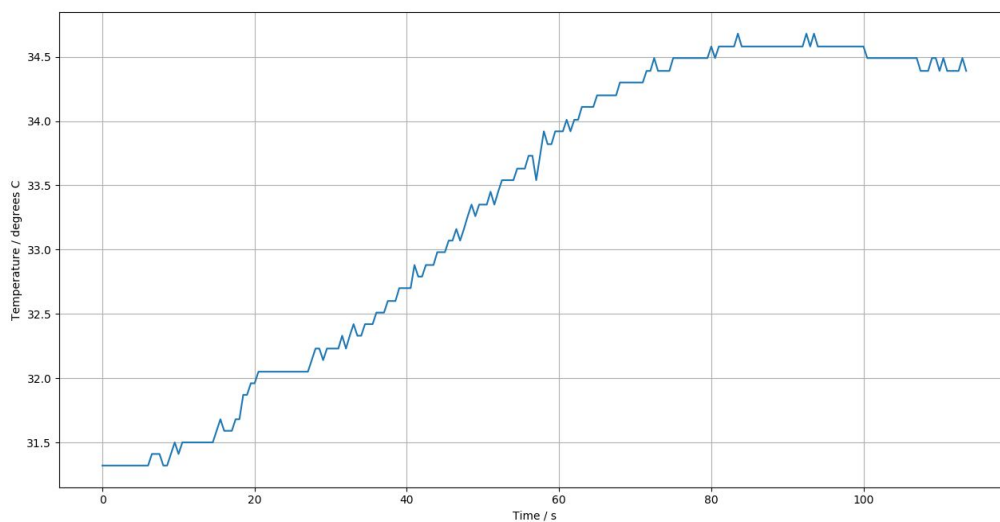


Figure 4.6.4: heating to a target temperature of 34.5°C with an improved heating strategy.

4.7 Appendix: Temperature Control System

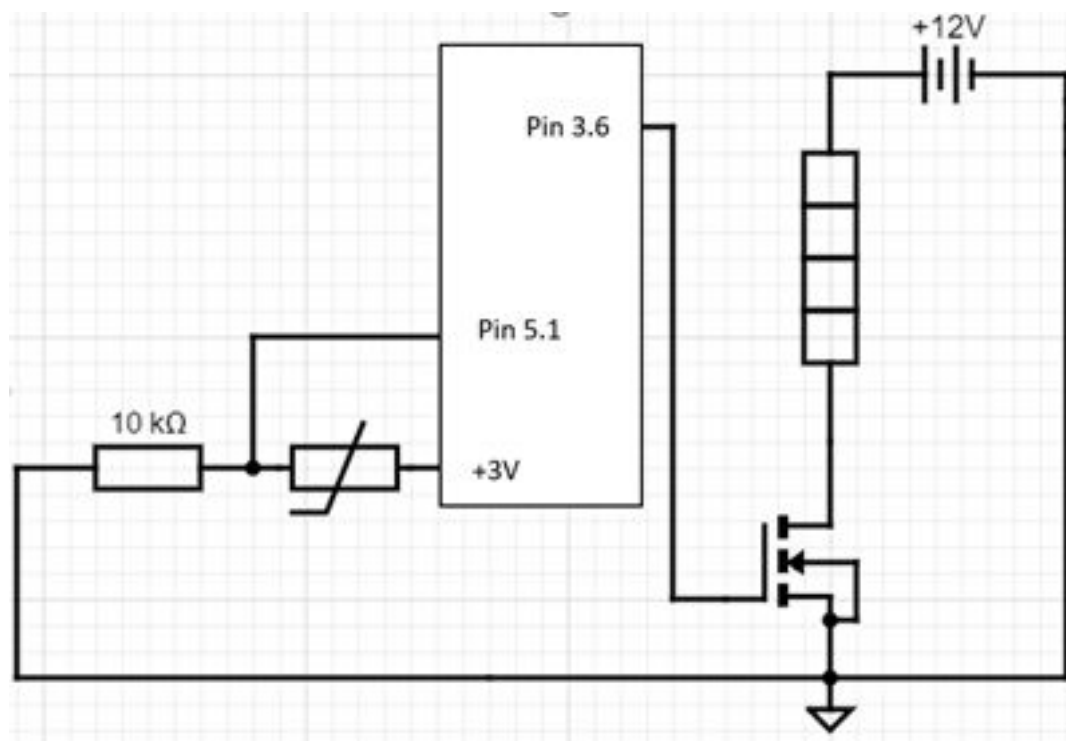


Figure 4.7.1: the overall temperature control system circuit.

4.8 Appendix: Stirring Subsystem and Results

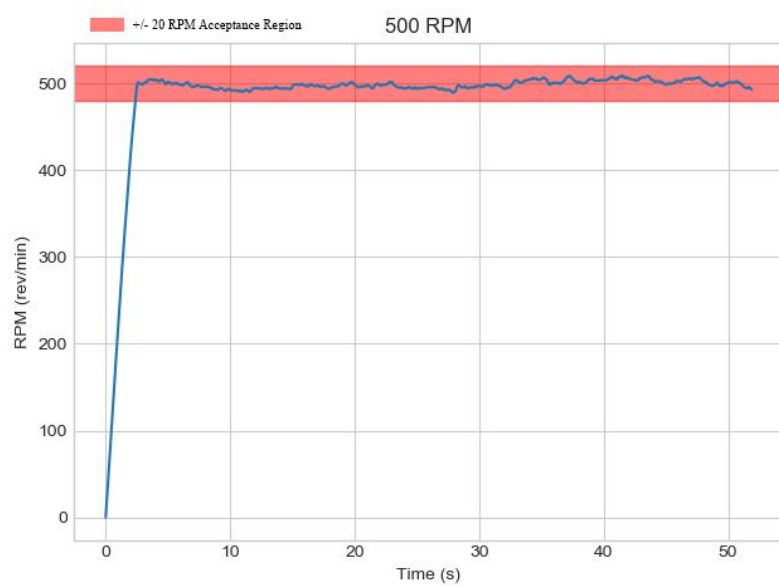


Figure 4.8.1 : Stirring maintaining 500RPM, staying 100% within the acceptance region.

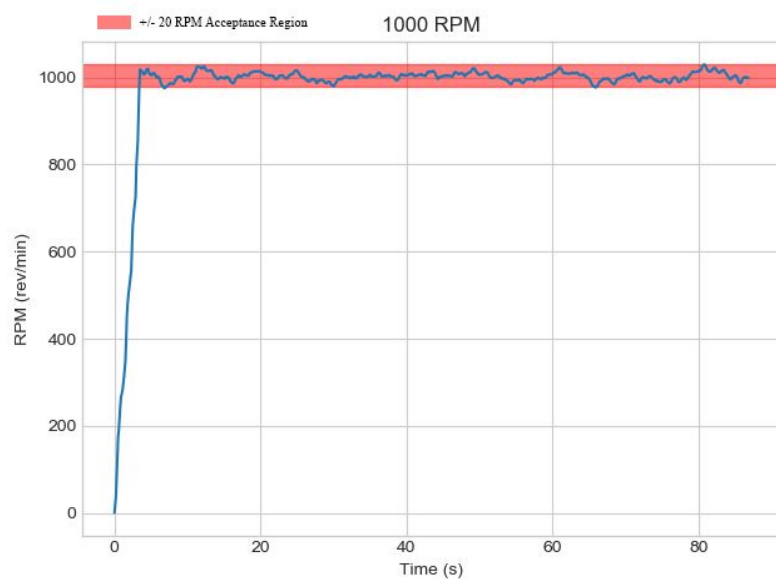


Figure 4.8.2 : Stirring maintaining 1000RPM, staying 100% within the acceptance region.

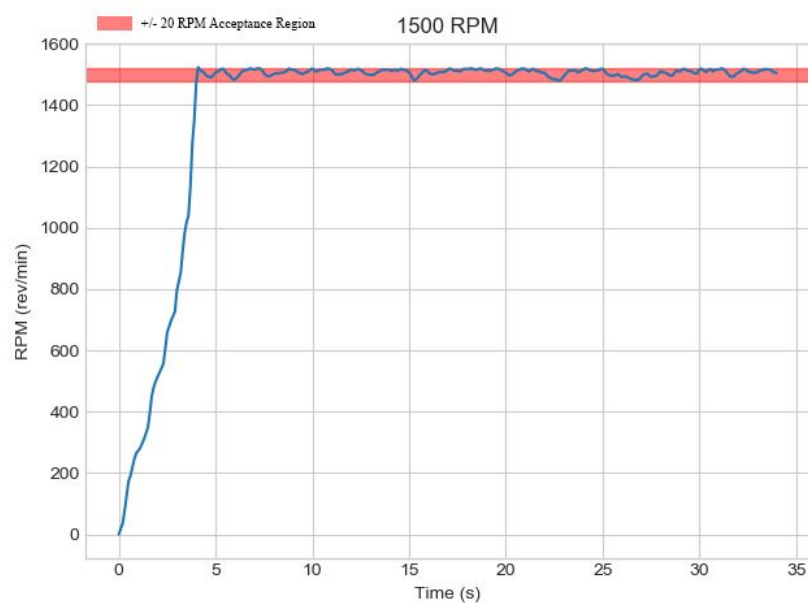


Figure 4.8.3 : Stirring maintaining 1500RPM, staying 99.07% within the acceptance region.

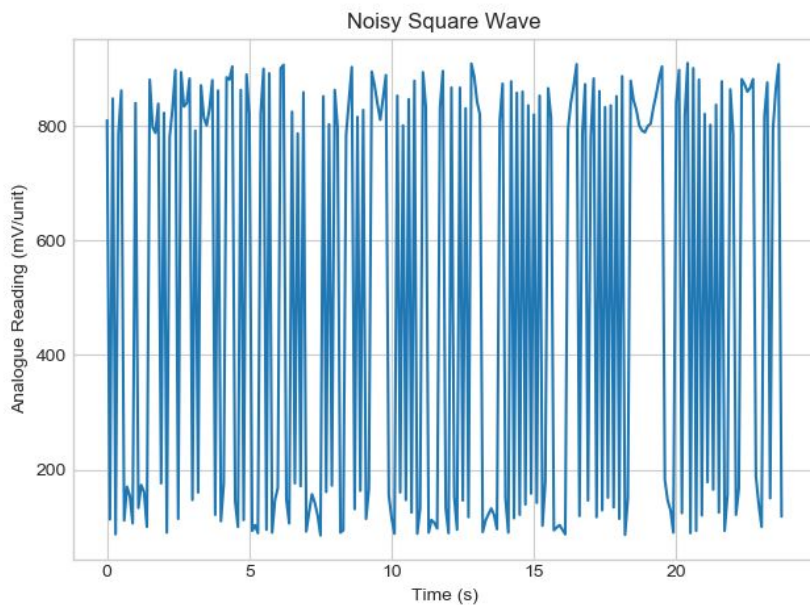


Figure 4.8.4 : Plot of raw output from photo-interrupter.

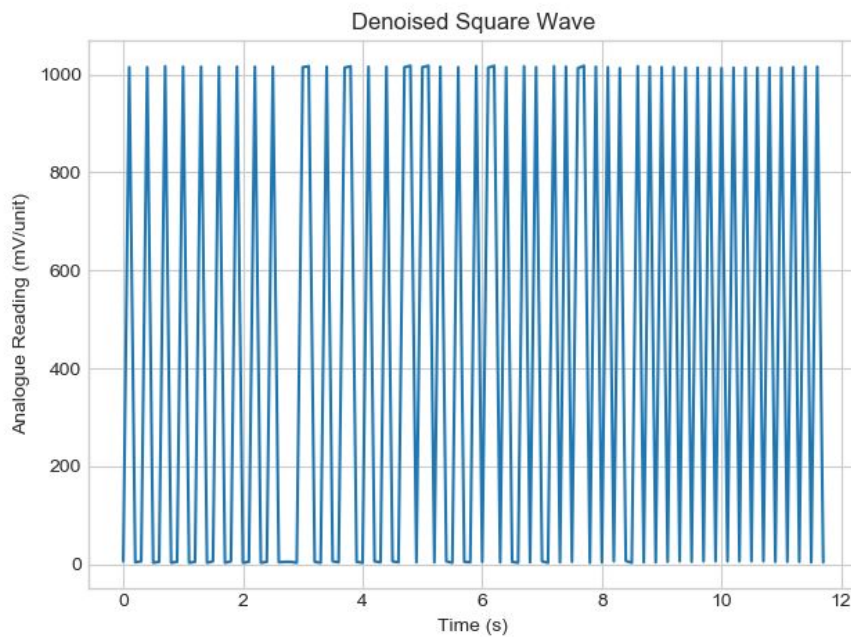


Figure 4.8.5 : Denoised plot of raw output from photo-interrupter.

```
for num in x_axis:
    if float(num) > rpm_upper or float(rpm) < rpm_lower:
        inaccuracy += 1
print("Accuracy", str(((len(x_axis) - inaccuracy)/len(x_axis))*100) + '%')
```

Figure 4.8.6 : Test script used to calculate accuracy of subsystem.

Keep a count of the values that lie outside the bounds defined in the specification. Then work out percentage error.

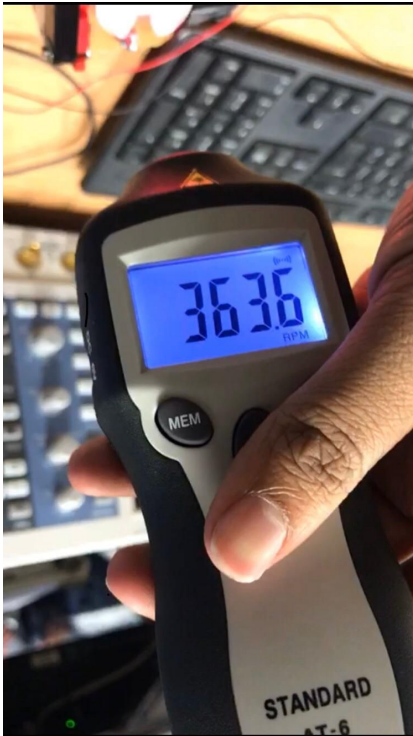


Figure 4.8.7 : Set to 500 RPM but tachometer returning 363.6 (i.e 181.8 RPM)

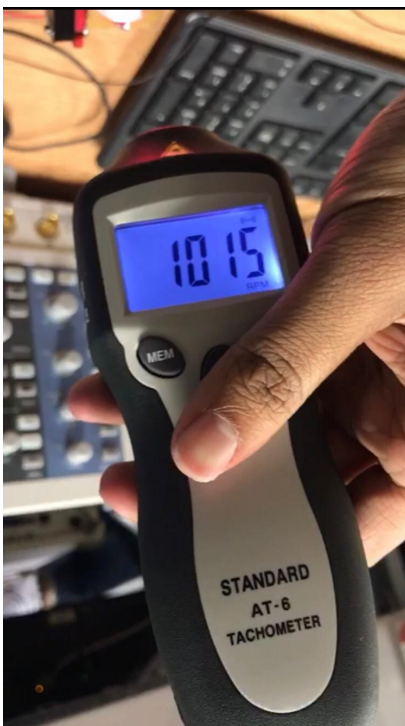


Figure 4.8.8 : Set to 500 RPM and tachometer returning 1015 (i.e 507.5 RPM)

4.9 Appendix: Stirring Code

```
#define MOTOR 11 //Pin to drive 3V Motor
#define SQ 2 //Square Wave Pin
#define SIZE 50
int desiredVal = 700;
int startTime = millis();//time at a peak
float t = 0;//period
float frequency = 0;
float rpm = 0;
float adjustedRPM = 0;
float averageRPM = 0;
float rpms[SIZE] = {0, 0, 0, 0, 0};
int count = 0;
int previousRPM = 0;
int time_high;
int time_low;
int starting = 1;

void setup() {
  Serial.begin(9600);
  pinMode(MOTOR, OUTPUT);
  pinMode(SQ, INPUT_PULLUP);
  adjustedRPM = 500;
}

//analogue read maps volatages 0-3.3v into integer values between 0-1023. Analogue to Digital Converter
int removeNoise(int rpmVal, int prevRPM){
  if (rpmVal < 1500){
    return rpmVal;
  }
  return prevRPM;
}

int setRPM(int value){
  if (((value >= 500) && (value <=1500)) || value == 0){
    desiredVal = value;
  }
  //Serial.println(persist);
  return desiredVal;
}

float getAverage() {
  rpms[count] = rpm;
  count++;
  if (count >= SIZE) {
    count = 0;
  }
  float sum = 0;
  for (int i = 0; (i <= SIZE); i++) {
    sum = sum + rpms[i];
  }
}
```



```
}  
    return sum;  
}
```

```
int getRPM(){  
    return averageRPM;  
}
```

```
void loop() {  
    time_high = pulseIn(SQ, HIGH);  
    time_low = pulseIn(SQ, LOW);  
    t = (time_high + time_low)/2;  
    rpm = removeNoise((300000/t)*60, previousRPM);  
    averageRPM = getAverage() / SIZE;  
    previousRPM = (300000/t)*60;  
  
    if (rpm > (desiredVal)) {  
        adjustedRPM -= 5;  
    }  
    if (rpm < (desiredVal)) {  
        adjustedRPM += 5;  
    }  
  
    analogWrite(MOTOR, map(adjustedRPM, 0, 2000, 1, 255));  
  
}
```