Challenge 2: Global Health care

EEE-CS Specialist Team 24 Final Report

Authors: Z. Alkalza, S. Kaklin, Y. L. Ho, F. Calderon, A. Fawzy, A. Augugliaro, N. Soni, R. Shah, R. Wong, S. Sethi, J. Halse

1 Introduction

Author: Ziad Alkalza

Tuberculosis is currently the second deadliest disease in terms of death count with over 1.6 million deaths [1], according to the World Health Organisation. That means that we are faced with a very serious pandemic that needs to be treated. Well developed countries are to an extent able to contain this disease. However, in other places this can be difficult because of many reasons like poverty and poor infrastructure.

The challenge is to model a fully functioning plant which creates a vaccine for tuberculosis in a small village in Uganda called Mbarara. Part of our job is to also see how this can be implemented to suit the infrastructure and nature of a country like Uganda, and find ways more sustainable that will improve the way of life in the area.

To create a vaccine, the bacteria need to be placed in a bioreactor which contains a specific solution that will allow the bacteria to grow, so it can then be harvested and used. It also needs to be placed in a controlled environment that will aid in producing the optimum yield.

Our specialist team worked on creating a control system that makes sure the bacteria is in the right environment. This is done using a microcontroller, which makes proportional changes to the output according to the input fed to it by sensors.

The system (as seen in figure 1) will consist of 4 subsystems each with a specific task:

1. Temperature
2. pH
3. Stirring
4. User Interface

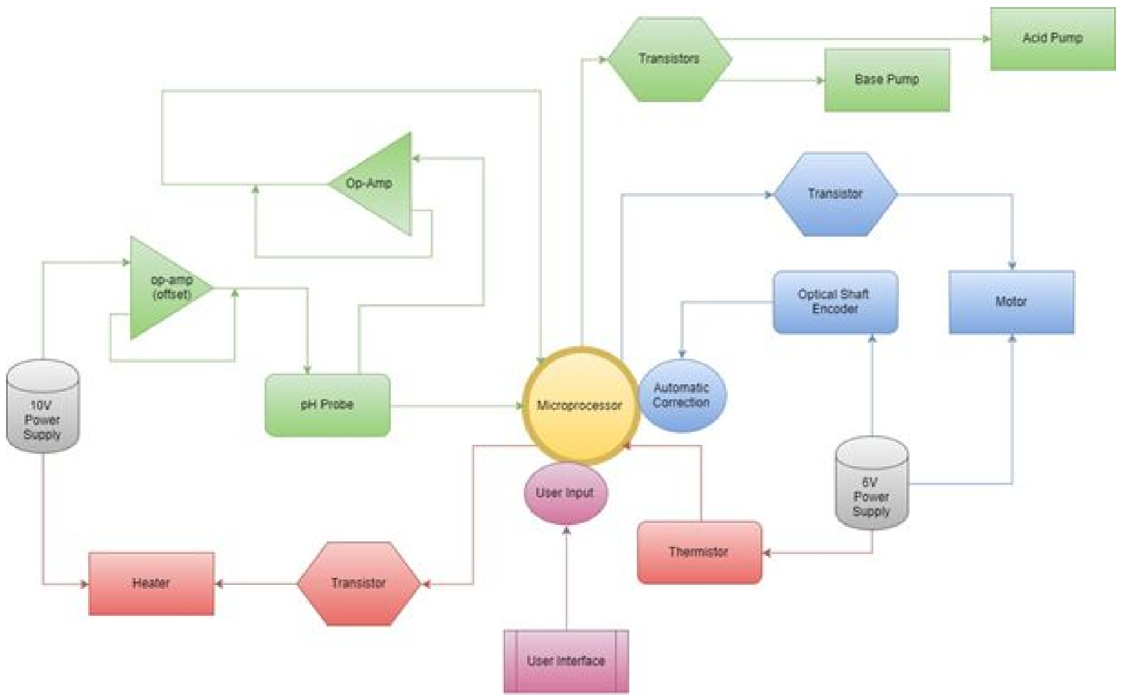


Figure 1:Block Diagram of the whole system

Each subsystem has its own parameter that needs to be controlled in the Bioreactor. These parameters need to be of a specific value or range, which are specified by the Chemical and Biomedical engineering specialist teams.

These parameters are:

- Temperature
- pH of Solution
- Stirring speed of Solution

Specifications for each subsystem	
Heating	Optimum=32°*C (min=30°C max=35°C)*
Mixing	260 ± 20 RPM
pH	Optimum=7 (minimum=6.2)

If these parameters fell above or below the range, the vaccine may not grow correctly and will not be effective. That means it is crucial for them to be constantly controlled with high accuracy, or else the production of the vaccine will fail.

Now we will get into more detail for each subsystem.

[1]https://www.npr.org/sections/goatsandsoda/2014/09/16/347727459/which-contagious-diseases-are-the-deadliest

2 Subsystem Descriptions

2.1 Heating Subsystem:

Authors: A.Augugliaro and N.Soni

The purpose of the heating system is to provide heat to the bioreactor in order to create an acceptable environment for the development of the vaccine. The system consists of a simple circuit which provides heat to a certain volume of water and then that heat energy is used as a source in the bioreactor.

To do the calculations, we were given an initial range of the temperature for the liquid inside the bioreactors, which was between 25^{o} to 35^{o} Celsius. In this range, it was assumed that this would create perfect conditions to have a high yield in developing the vaccinations in a short amount of time. As we needed a set temperature to have a conclusive system, we had meetings with the Biochemical Engineers and decided that the optimum temperature would be 32^{o}. Then the Electrical and Computer engineers based the circuit and the program controlling the circuit around this temperature.

The heating system consists of three main components: The thermistor to monitor the temperature, Microcontroller and The Actuator- a load connected to a 12 Volts power supply and to the output of a MOSFET transistor, which takes the microcontroller signal as input. All three work in conjunction to each other to make a perfect heating system. The thermistor gives in input to the Microcontroller about the temperature of the liquid in the reactor, as soon as the temperature falls below 30^{o}, the circuit gates are closed, so that the resistance controlled by the actuator increases and causes the liquid to heat up.

2.1.1 Temperature Monitoring

Authors: A.Augugliaro and N Soni

For this first section of the subsystem, we have the thermistor(connected to the circuit using a pair of jumper cables) in series with a fixed 1k Ohm resistor; connecting the microcontroller across the thermistor in this manner creates a voltage divider circuit which allows us to convert the initial resistance into a voltage. The circuit must be connected to a 6 Volts supply from the battery(Figure 1,appendix 1). In the potential divider circuit as the, temperature across the thermistor increases the potential difference across the series resistor increases. This increase in potential difference is measured and input into the microcontroller.

2.1.2 Microcontroller

Author: N.Soni

The device we used to control the heating system was a MSP430 microcontroller, which was connected to the heating circuit. The controller was used to take temperature inputs from the circuit and keep a record of them. As soon as the value exceeded the range, a signal would be sent to close the gates through the controller and stop the heating process. This was achieved using the ADC (analogue to digital converter) pins on the board. These pins take voltage inputs and convert it into a digital signal which can be understood by the program.

A software was designed to process all this information. As the microcontroller can be controlled using the Energia which is based on the C/C++ language. The system took inputs from the serial input pin, in the form of a digital signal. These voltage values were then converted into degree Celsius using the Steinhart-Hart equation. Firstly to ensure that the program took correct voltage inputs, we used a digital multimeter to measure the voltage between the resistors in the circuit and then compared it to the values being output by the computer. During this process, we used a common constant to convert from voltage to degrees Celsius. However, once the circuit was accurately completed, we plotted a graph of 5 voltage readings versus the temperature of the water, measured using a thermometer to calibrate the system, in order for it to take in accurate real time inputs.

Besides this the Microcontroller also has ground pin which were used to ground the circuit in order for it to be complete. This is used as a common reference points as all the components of the circuit are connected to the same ground pin.

2.1.3 Heating Actuator

Author: A. Augugliaro

In this section two things were fundamental: the first one was to read the datasheet of the particular transistor we were using, because each transistor has a different gate order; the second one was to use the 12 volts power supply. In fact, this subsystem is the one that requires the highest amount of power, which is the product of voltage and current; we used a voltage of 10 V, putting a resistor in series with the battery to make the voltage drop, and a current of 3 Amps. As the microcontroller can draw a maximum current of 48 mA, an external power supply is needed, which is controlled by the transistor, as it acts like a switch which makes it possible to control the flow of current. Notice that the second input of the transistor, which has 3 gates, is connected to ground. To make it clearer, the load(heater) has the power supply attached at one terminal(it doesn't matter which one) and the output of the transistor at the other(figure 2, appendix 1).

2.1.4 Heating Results/Validation

Authors: A. Augugliaro and N Soni

We managed to meet the requirements, but not before solving an arduous issue: the heater kept on dissipating heat even after it had been turned off. Consequently, the temperature rose before the maximum limit by at least 1-2 degrees Celsius, which meant that the value being input into the bioreactor would be inaccurate. This problem was found in one of our test cycles where we placed a thermometer in the beaker containing the water and the thermistor. During the test we realised that even when the gates are closed at 32^{O}, the temperature of the water continued to rise which went against our hypothesis of the water cooling down too fast after the heater is switched off. In order to overcome this problem we ran many test cycles to find the optimum temperature at which, even after the system is turned off, the temperature of the remains to be at 32^{O} for the longest period of time.

All these values had a possibility of being changed at any time in the project as we had to coordinated with the needs of Biochemical engineers about the ideal conditions required to make the vaccination. For example the ideal temperature required was changed from 32^{o} to 50^{o}. This was because at this temperature the yield of the yeast was the highest.

As a future improvement for the heating system, we also decided to insulate the heaters in order to prevent heat loss and consequently use less power in heating. Also to have to take the resistance of the wires and connecting cables/crocodile clips, into account when making calculation about the resistor being used and the current in the potential divider circuit. This would help to have a more efficient system, which will be cost effective. This will help to reduce the cost of the final vaccinations being made, which will help them to be accessible by a wider range of the population.

2.2 pH Subsystem

Authors: Rowan Shah & Y. L. Ho

The pH subsystem is responsible for monitoring and controlling the pH value of the media in the bioreactor. The subsystem should maintain the pH at optimal 7 according to our biochemical engineers and send the measured pH value to the user interface which keeps a history of the pH value by displaying a graph.

2.2.2 pH calibration

Author: Y. L. Ho

During calibration, we were provided vinegar and water as testing solutions. We first measured the pH of both water and vinegar by a pH meter. We then inserted the pH probe connected to the circuit into these two solutions separately. The microcontroller received the raw ADC value and the code translate the value into electric potential at the pH-measuring electrode (using the formula: electric potential = raw ADC value * 3 / 1023). After obtaining the referencing electric potential value which is about 400mV, we can compute the pH of the solution by the formula: pH = 7 + ((0.4 – electric potential at pH probe)) * F / (RT In(10)). However, we discovered both the electric potential and pH value measured by the pH probe fluctuated at about ± 0.5 and the accuracy was affected. Thus, we decided to divide the original measuring time interval into 10 subintervals. Every 10 measured values were stored temporarily in the microcontroller and only the average of these 10 values was used in controlling the pumps and shown in the user interface. The average values appeared to be more stable and a lot closer to the value shown on the pH meter. Since temperature also has a slight effect on measuring pH, we also took it into consideration later on and improved the accuracy further. The relationship between electric potential, pH value and temperature is shown in Appendix 2 Figure 1.

2.2.3 pH Results/Validation

Author: Y. L. Ho

We managed to maintain the pH value of the solution within optimal 7 ± 0.2. The tolerance was adopted since we experienced a problem: the pumps would be turned on and off repeatedly if there was no tolerance. For instance, let the measured pH of the solution be 6.95 which is slightly below 7. The base pump will be turned on since 6.95 is strictly smaller than 7. However, it is likely that we add more base than needed for the solution to reach a pH of 7. The solution will have a pH greater than 7 and the acid pump is turned on this time. Then once again, the base pump is turned on when the pH drops below 7. This cycle repeats and keeps on adding more base and acid. It is not only a waste of resources but also seriously affect the quality and efficiency of the vaccine production since the media for producing vaccine is being diluted constantly.

We came up with two ideas to deal with this issue. The first one was trying to calculate the amount of acid or base needed for the solution pH to reach 7, and then control the amount of acid and base released by the pumps precisely. However, we did not find an easy solution to precisely control the pump. Instead, we chose to adopt a tolerance for pH, so not more acid and base is added when the measured pH falls in its range. This method was effective in solving the problem. The value ± 0.2 for tolerance is chosen after several experiments to make the pumps more stable while still maintaining a pH around 7. A mixture of vinegar and water, and a pH meter were used to prepare the experiment solution. Moreover, the pH value was measured every 5 seconds to make sure the solution has time to be stirred well and has uniform pH.

2.3 Stirring Subsystem = Motor Driver + Motor Speed Sensor

Authors: Szymon Kaklin, Ahmed Adeeb Fawzy, Ryan Wong

The stirring subsystem is responsible for maintaining a constant stirring speed within the bioreactor, determined by user input. The system can be subdivided into two main parts: the motor and speed sensor parts. The stirring system must be capable of providing 260 ± 20 RPM, displaying the current RPM to the user, as well as keeping a log of the data.

Figure 2 (see appendix 3) shows a simple block diagram of the full stirring subsystem, with the motor part in red, and the speed sensor and user input in blue. Figure 3 shows the completed circuit design.

2.3.1 Motor Driver

The motor section was the largest for the stirring subsystem, and consisted of a 3V DC motor with a maximum current of 1A, 6V battery with 7AH, a diode to protect the motor, and a transistor. Once the circuit was complete (figure 3), the code for this section had to be written in such a way that the motor would be able to respond to user input. The input is provided in RPM, for example “260”, and is then converted to an 8-bit output which is sent to the transistor gate. As the battery is 6V and the motor is 3V, we decided to set a maximum output of 128 for the transistor in the code, essentially halving the previous maximum output. This safeguards the motor.

The conversion from 8-bit to RPM values was done by linearly mapping the values from 0 to 255 to various RPM values. Through our testing, we found that an output of 128 to the transistor corresponded to approximately 3000 RPM. As RPM is assumed to be directly proportional to the 8-bit output value, we were able to map all values with the function presented in figure 4 (appendix 3). This value between 0 and 255 dictates the speed of the motor through a technique called pulse width modulation. The transistor acts like a switch, turning the motor driver part of the circuit on and off at various speeds to change the RPM output of the motor.

During testing we found several problems with this method. For one, there was no way to tell if a user RPM input would accurately match the 8-bit output value sent to the transistor. Secondly, it became obvious that the accuracy of the model when the motor would be submerged into liquid would be off. For example, when the motor was in water, the 8-bit output values would not match the RPM values accurately. This is why we implemented a self correcting ‘if’ loop in the code, to make sure the RPM of the motor would correct itself, even if the liquid in the bioreactor were to become more dense. This will be discussed in the next section. This meant that even if the RPM were to be affected by other external factors, or if the initial output value does not match the RPM defined by the user, the system should correct itself.

2.3.2 Motor Speed Sensor

For the speed sensor, a photo-interrupter was used alongside the code to calculate the motor speed. The sensor is made up of two parts, a disc and a photodetector. The disc is attached to the motor and therefore rotates at the same speed as the motor. It also has two slits on it, this means that there will be two interruptions of light reaching the photodetector per revolution of the motor. Therefore, with regards to the code used, will result in two changes of state. The photodetector determines the change by counting either the number of rising or falling edges, which is determined by if light is incident on the sensor or not. this data is then processed to produce the RPM of the motor. For RPM to begin measuring, the photo - interrupter must complete two full cycles, as such, autocorrection begins a bit late.

The sensor is an important part of the circuit because not only is it required to control the RPM it is also responsible for auto-correcting the RPM. Figure 5 shows the if loop in the code used to auto-correct the 8-bit output. This means that even if the initial user set RPM input does not agree with the one measured by the speed sensor, the 8-bit output to the transistor will be varied accordingly, so that the detected RPM matches the one set by the user. Figure 6 shows the oscillation of the RPM around the required value of 260. Unfortunately, the range within which the RPM oscillated was closer to ±40 than the specified ±20. This is most likely due to the way in which the RPM reading and the auto-correcting code are refreshed. The auto correction is applied every 0.5 seconds, however the RPM updates upon its completed cycle, therefore the two code loops are essentially at a random phase relative to each other at any point in time. This is what may cause the seemingly random changes in RPM.

The circuit, as shown in figure 2, worked hand in hand with the code to drive the motor. While varying the RPM of the motor as desired with the programs mentioned, there were a couple things to consider: The delay and the change in the 8-bit output value in each cycle of the photo interrupter. As the rotations are measured continuously, there will be a rounding error of 1 rotation. This error can be minimized by using a longer delay, allowing time for more rotations each time the code loop runs. However, as the delay is increased, the time interval between each increase/decrease in the 8-bit output will be longer and the time taken for the RPM desired by the user to be reached will also increase. Therefore, to ensure a good balance between speed and accuracy, we used a delay of 500 milliseconds, and ran several tests to see how much to increase the 8-bit after each cycle. We ended up with increasing the 8-bit output by 2 each cycle.

3 User-Interface Subsystem

Author: F. Calderon

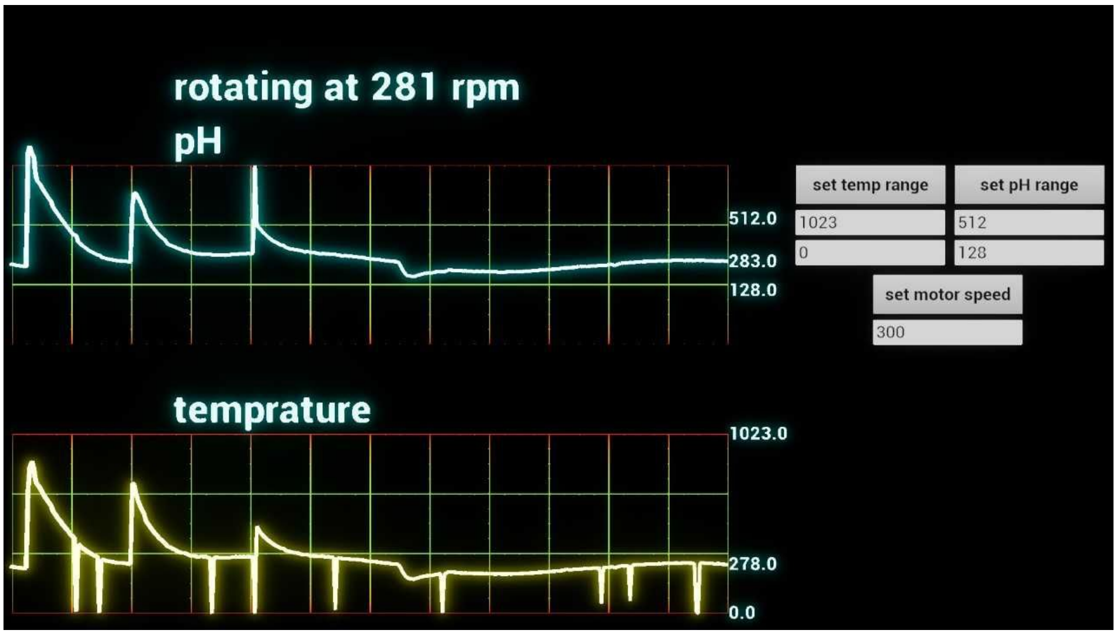
The UI subsystem team had to work on a User to Computer interface each team could use in the goal of accomplishing their goal: controlling the pH of a solution, it temperature and the speed of the stirrer.

First of all, we wanted to make an easy to use User Interface (UI) that looks simple and pleasant. After some researches, we started to use the software Processing coding with Java language. We then learned how to design boxes on the UI, create buttons, plot a graph with several points and lines to link each point together. But after a moment we had some problems trying to connect Processing to Energia's launchpad. We couldn't display the data we received from the microcontroller on the UI. We then tried to use a different language for Processing, Python, but in the end, we had the same problem.

Then, one of our team member had the idea to use another software he masters: Unreal Engine. Even though this software main goal is to create video games, it's different and wide range of options were enough to create a good UI for a non-video game project. The difference between Processing and Unreal is that to create a UI, on Processing we will code in Java whereas on Unreal we don't use a proper code language. We had to use a Blueprints Visual Scripting (BVS). A BVS is a gameplay scripting system created with C++. It goal is to defined object-oriented classes or objects without a single line of code. Using this system allowed us to work quicker and more effectively. People from other sub teams could easily understand the code thanks to the BVS design. Each step of the code is represented by a box. Hence, to make the code work, we had to connect each box together with a coloured line. Making a UI code understandable by each member of the team was very useful in the extent to which they could know how the UI they use work and how it was built.

Our UI could display in a simple way data received from each subsystem of the bioreactor and plot them on a graph. On the UI, we designed two slot for pH values graph and heating values graph. Then three slots to control pH, heating and stirring values so we could easily change them.
The Launchpad sends data over the usb/serial port which is received by the UI.
The data is a string in format temp,pH,rpm. The string is split into 3 values which are sent to the display objects and rendered as the graphs.
The ui contains a menu which can be used to set the desired range of the temperature and pH.
In the screenshots the range was for the test data not the final system.

When a range is set the ui sends a string down the serial port which is read and interpreted by the Launchpad and used to set its internal variables for the ranges.



set temp range	set pH range
1023	512
0	128

set motor speed: 300

4 Overall System Integration and Summary

Author: Sehej Sethi

Heating:
The heating system had the ability to heat the solution to the desired temperature. While saying this, the system was not completely accurate. For example, the way the heating system was implemented meant that the temperature of the solution usually surpassed the anticipated temperature. This was because the heater would heat the solution until it reached the temperature required rather than turning off a few degrees below the desired temperature.

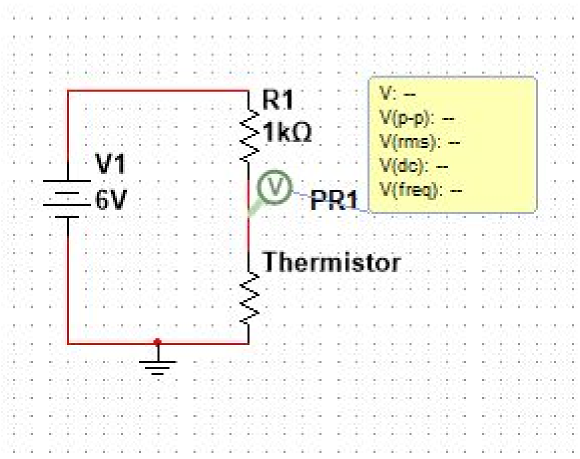
Stirring:
The stirring system functioned at a range of speeds allowing the solution to be mixed thoroughly. The motor seemed to work well, producing accurate results, at 1000 revolutions per minute (rpm) to speeds of around 3000 rpm. On the other hand, the stirring system had trouble working at speeds less than 1000 rpm which may be a problem for the overall system. For example, a recommended speed for the system to work at is 260 rpm but this speed had trouble being maintained at times.

pH Level:
Determining the pH level of the solution proved to be a challenging task. This subsystem, provided an approximation of the pH level of the solution, with a range of +_ 1 from the true value. This meant that the system was not completely accurate. Additionally, this caused the pumps, which were used to optimize the pH level of the solution, to also be erroneous. Unfortunately, this meant it was very hard to maintain a constant and preferred level of pH.

Complete System Summary:
In general, most parts of the system did work at varying levels of success. As some of the elements of the system are not completely accurate, it may lead to unwanted results.

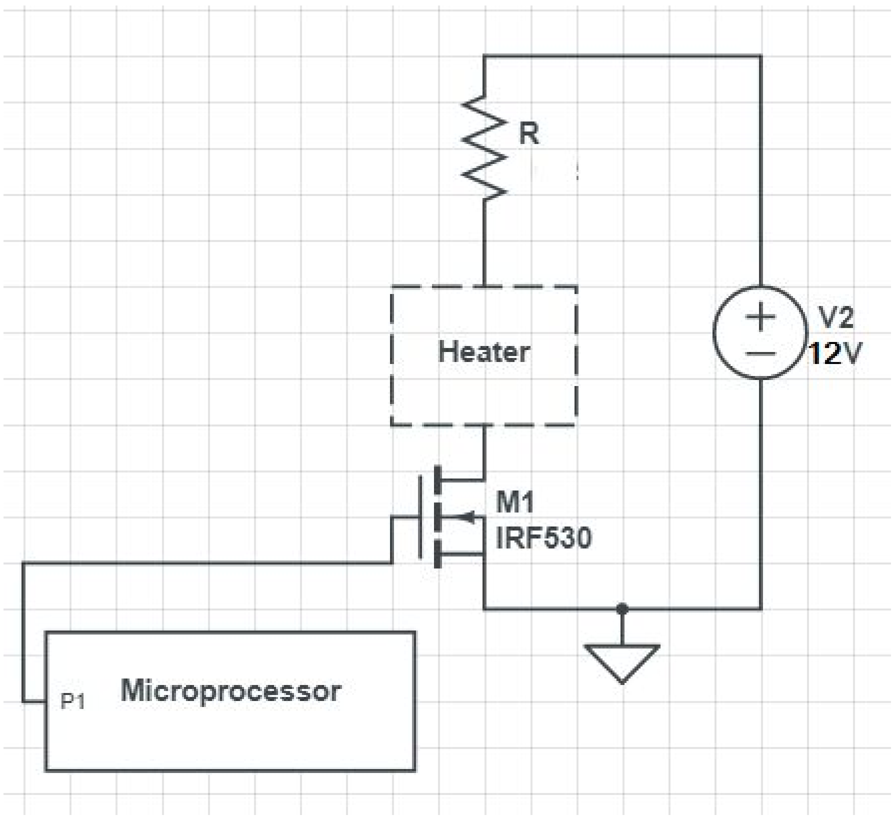
5 Appendices

5.1 Appendix 1



5.2

Figure 1:the voltage divider circuit for the heating sensor



5.3

Figure 2:the circuit for the heating actuator

5.2 Appendix 2

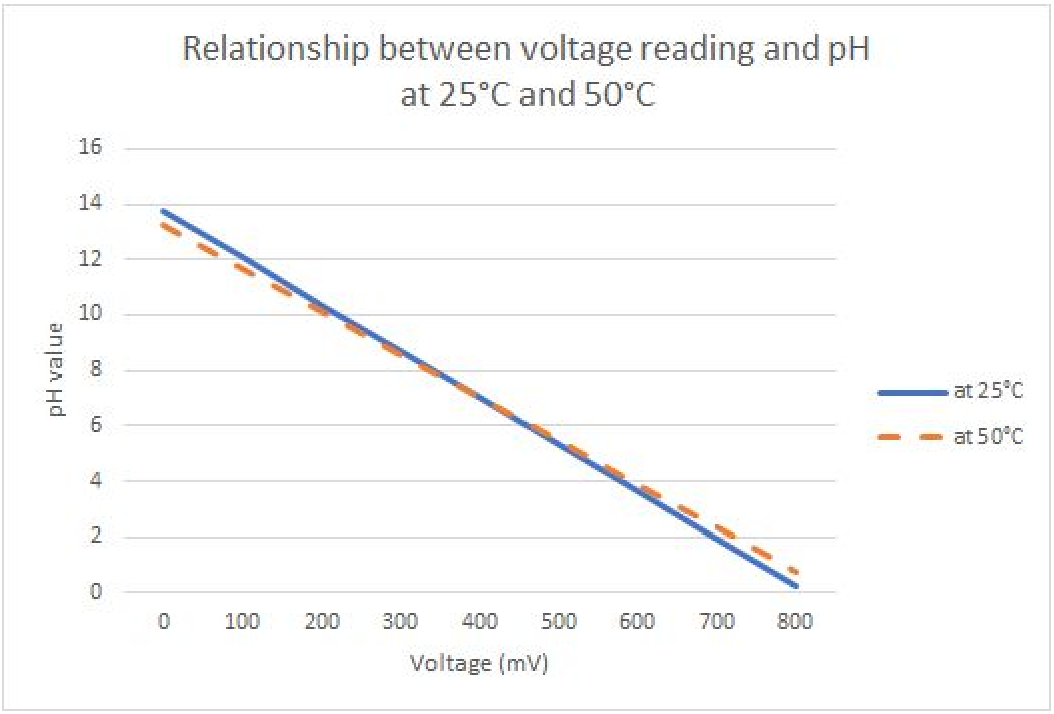


Figure 1. Relationship between electric potential at pH measuring electrode and the pH value at different temperature

5.3 Appendix 3

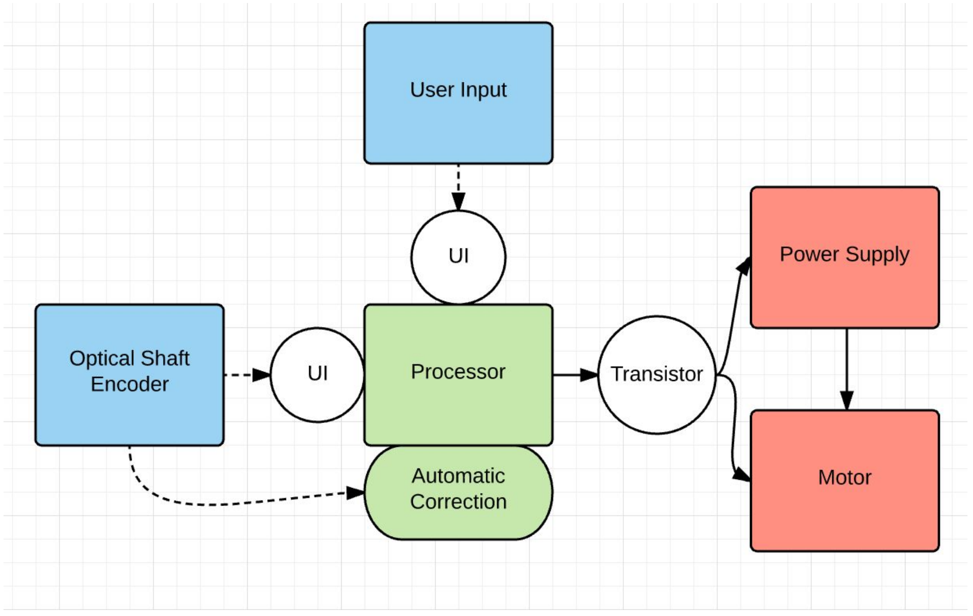


Figure 2. A block diagram of the stirring subsystem

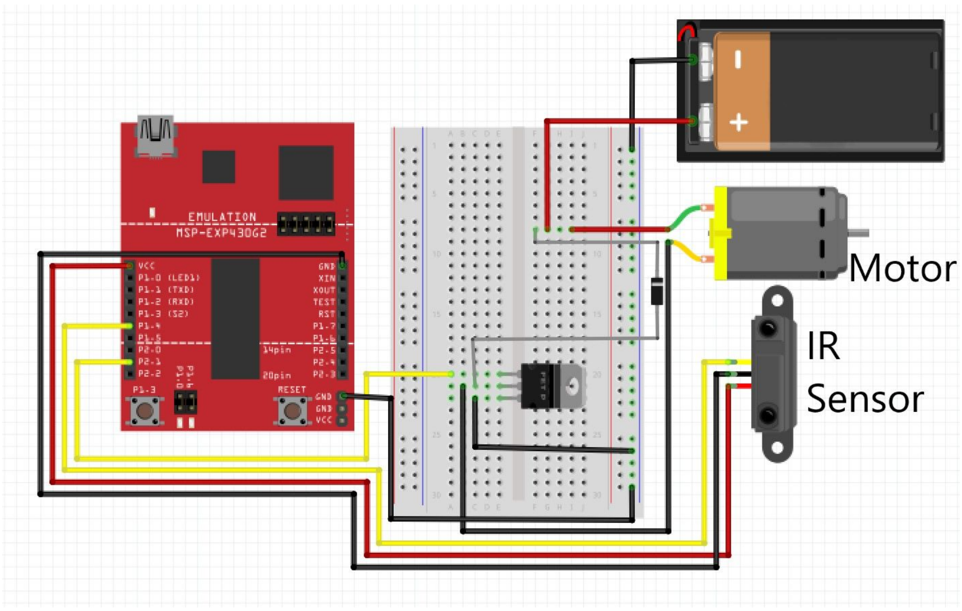


Figure 3. Completed design of the Stirring Circuit

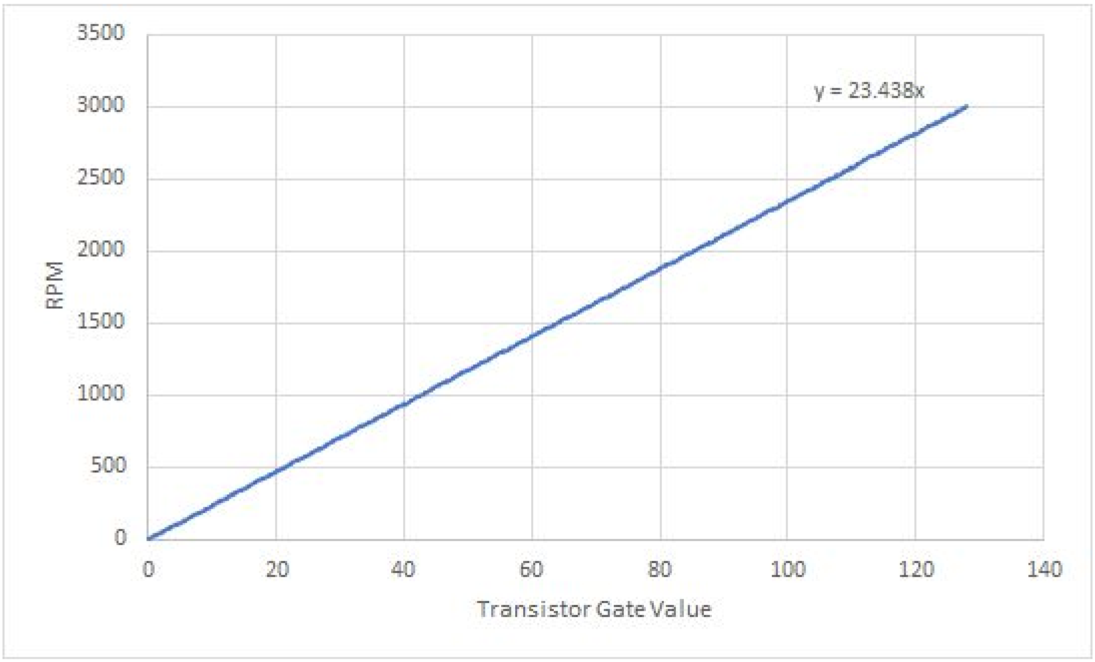


Figure 4. The linear model used to map the 8-bit output to the transistor to RPM values input by the user.

```
 {// Refresh for autocorrecting bit, set to 500 ms.
 if(currentMillis - previousMillis > RPMinterval){
   previousMillis = currentMillis;
// If RPM is more then the user set RPM, decrease 8-bit output by 2
if (rpm > desiredRPM){
  outputValue = outputValue - 2;
}
// If RPM is equal to the user set RPM, do nothing
if (rpm == desiredRPM){
  ;
}
// If RPM is less then the user set RPM, increase 8-bit output by 2
if (rpm < desiredRPM){
  outputValue = outputValue + 2;
}
if (outputValue > 128){
  outputValue = 128;
}
```

Figure 5. The 'If' loop used to auto-correct the output value to match the RPM

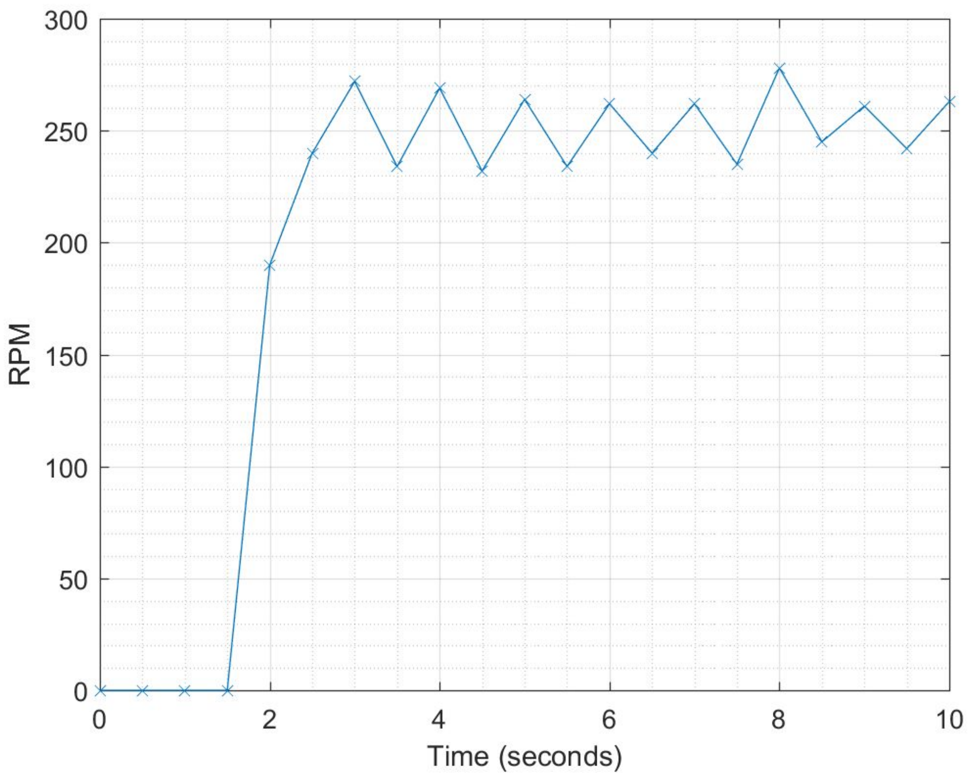


Figure 6. A graph of RPM vs. Time. Readings were done every 0.5 s, as that was the refresh rate in the RPM code. Initial readings are 0 as motor turns on at certain minimum output value. Graph is adjusted in time to show the point at which the correct RPM was reached.

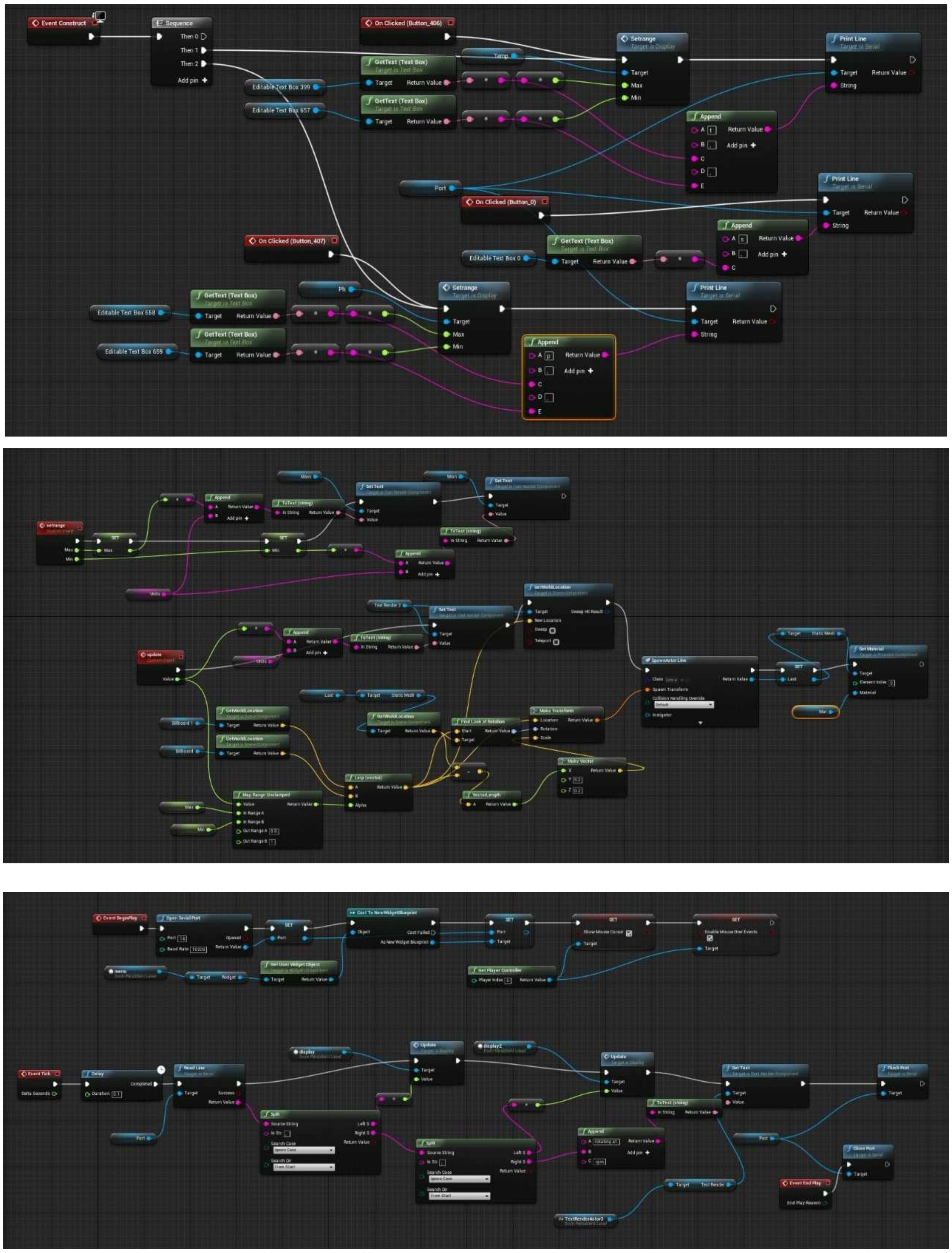


Figure 1: Screenshots of Blueprints Visual Scripting (UI)