Challenge 2: Global Health care

CS-EEE 02 Specialist Team Final Report

Authors: Qiren Dong, Maciej Kowalski, Grace Lau, Yujie Peng, Nerea Sainz de la Maza Melon, Aaryaman Sharma, Raven Valencia, Zhiyu Wang, Jian Yang

1 Introduction 2

2 Subsystem Descriptions 2

2.1 Connectivity subsystem 2

2.2 Stirring subsystem 4

2.3 Temperature subsystem… 6

2.4 pH subsystem………………………………………………………………………………………………………………7

3 Overall System Integration and Summary 8

4 Bibliography…………………………………………………………………………………………………….…9

5 Appendices 9

# Introduction

**Section Authors:** Zhiyu Wang (report task lead), Maciej Kowalski,

According to the World Health Organization, Tuberculosis (TB) is “the second leading infectious killer after COVID-19 (above HIV/AIDS)”, causing as many as 1.5 million deaths in 2020.[1] To combat the disease, a TB vaccine production plant in Uganda (since 25% of the deaths are in the African Region [2]) is being designed. Our task, as CS/EEE students, was to design a control system for a bioreactor - a key component in the production of the vaccine. The system needs to meet the following specifications:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Function** | **Range​** | **Accuracy** |
| **Parameter control** | Maintain a set temperature | 25-35°C​ | 0.5°C​ |
| Stir at a constant speed | 500-1500 RPM​ | 20RPM​ |
| Maintain the desired pH | 3-7​ | 0.5​ |
| **User interface** | Allow remote adjustment of the setpoints | | |
| Log and send data to an external server | | |

Table 1: Parameter control specification

 These characteristics are essential to supervise and maintain favourable growing conditions in the bioreactor, as well as allow to adapt the environment should such necessity arise. Below is a high-level block diagram of our proposed system:

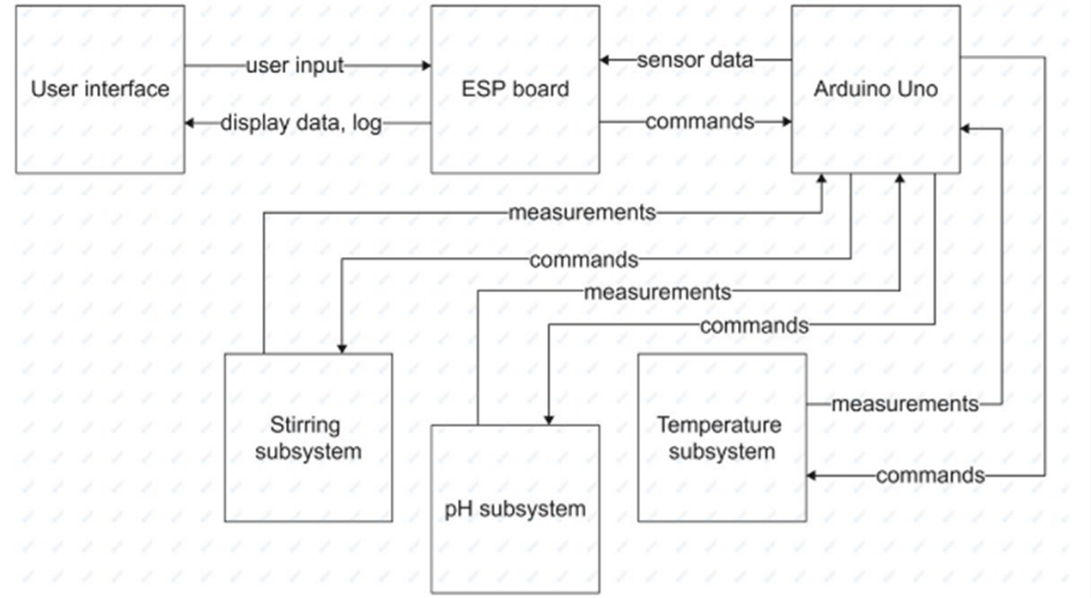


Figure 1: The block diagram of the system

The core element is Arduino Uno, which receives data from the subsystems, interprets it and commands the actuators to make the necessary corrections. The systems employs an ESP board to establish a two way-communication between the Arduino Uno and an the User Interface (UI).

# Subsystem Descriptions

## Connectivity subsystem

**Section Authors:** Nerea Sainz de la Maza Melon (Subsystem lead), Aaryaman Sharma

The primary goal of the subsystem is to link the UI to the sensor subsystems.

The specification for this subsystem can be broken down as follows:

UI:

* Allow user to edit ideal set points for pH, temperature and stirring (RPM).
* Monitor the above parameters in real-time and generate dynamic graphs for each.
* Enable logging of parameters for error detection etc.

Handling Sensor Data:

* Collect sensor data at regular intervals and send it to the UI for display to the user.
* Creating and deploying an encoding scheme to parse the analogue sensor data to a microcontroller and converting into a human readable form for the UI.

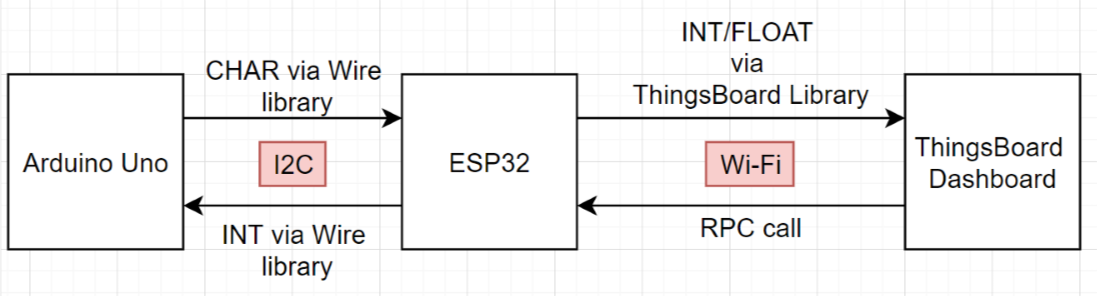
**Design**

Figure 2: Protocols and the flow of data between each agent

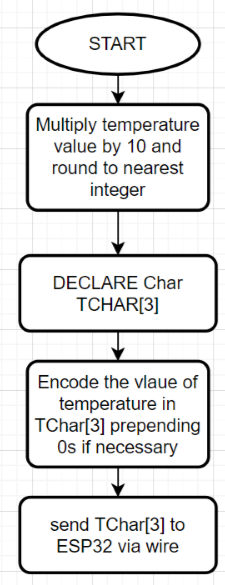
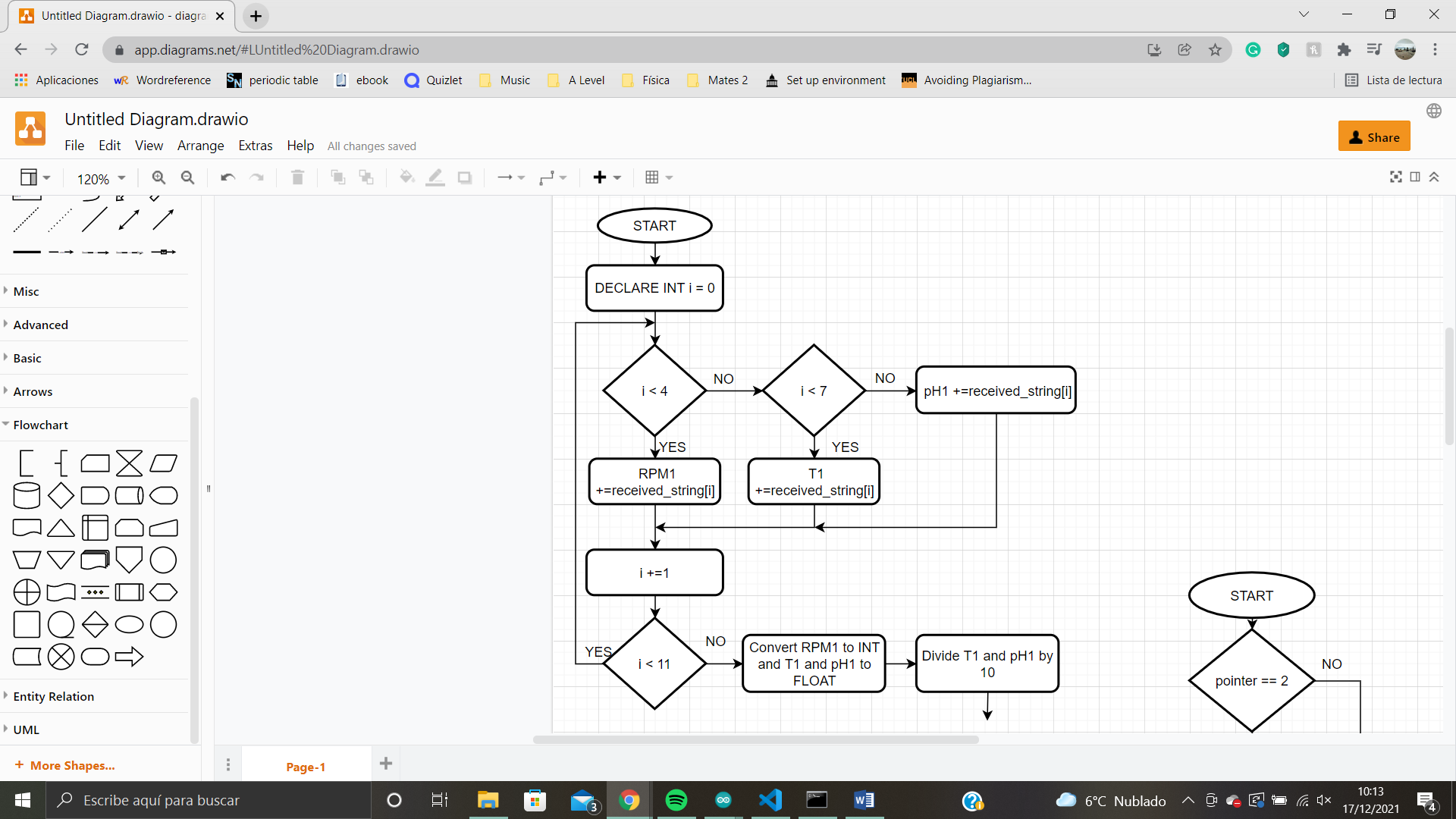
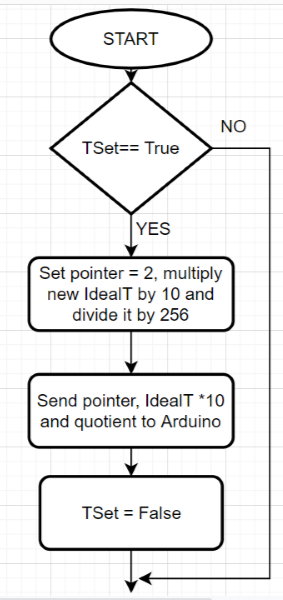
Figure 2 shows the bi-directional communication between the sensors and the UI. In terms of wiring, Since the Arduino Uno (5V) and the ESP32 (3.3V) are connected through a bi-directional level shifter to regulate the voltage. The wiring scheme provided in the Moodle page was followed [3]. Upon receiving the raw data, Arduino Uno converts it into standard units. In the case of temperature and pH, the values are floats. Before transmitting them, they are multiplied by 10 and rounded to the nearest decimal place as one decimal place is sufficient accuracy given the design specification. Figure 3 shows the encoding of temperature. Other parameters are sent analogously. RPM is encoded and sent first, it must be a 4-digit number. Then temperature is sent as a 3-digit number and finally pH is sent as a 4-digit number. This size and order is always kept the same - there is no need for extra bits to flag what each value represents. The wire library was used to send the values as per the example code on Moodle [3]. For full code, see Appendix A.    
 The ESP32 receives the 11-bit character array and stores it as a string, once again following [3]. It then iterates through the values in the string and stores them in their respective variables, as illustrated by figure 4. For full code check appendix B. Data arrives from Arduino Uno to ESP32 practically instantaneously. Finally, every 50ms the data is sent to ThingsBoard to be displayed and logged (with a few seconds’ delay because of the Wi-fi connection).   
 When a user inputs a set value via ThingsBoard a remote procedure call (RPC) is created in the dashboard. The ESP32 is connected to Wi-Fi [3] and subscribed to this dashboard using the code provided in [4]. This means that once an RPC call is received, the appropriate callback is called. The callbacks have a flag, indicating if they’ve been called. If they’ve been called, the variable relative to the changed set point is formatted so that it can be sent to the Arduino Uno. It is sent as an integer. Given that Wire.write() can only send 2 bytes, the value of the new ideal set points is divided by 256 and the result of the division is sent to the Arduino Uno alongside a pointer indicating which set point the sent data is referring to (1 for RPM, 2 for temperature, 3 for pH) and the value of the new set point as this would be equal to the remainder of dividing it by 256. Figure 5 shows the example encoding (temperature). For full code check appendix c. Once the value is received by the Arduino Uno, it calculates the new set point (figure 6).

Figure 4: Decoding of string sent by Arduino Uno to ESP32

Figure 3: Encoding of temperature before it is sent to the ESP32

**Validation**

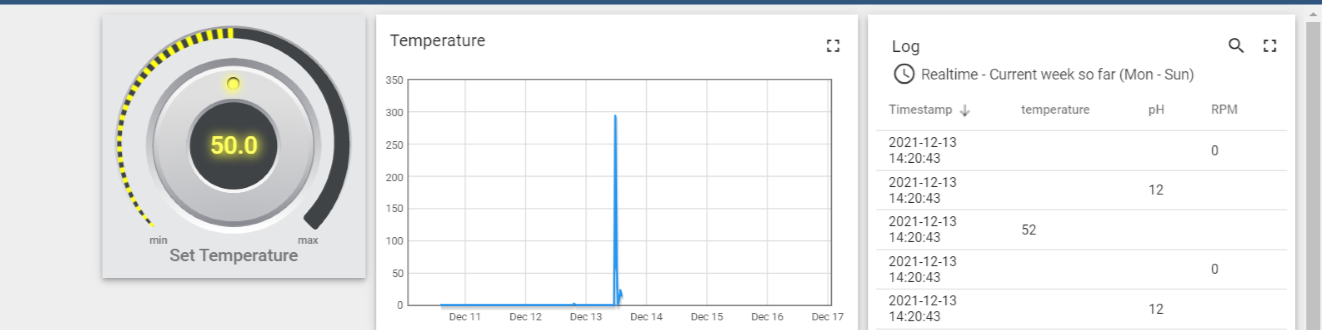
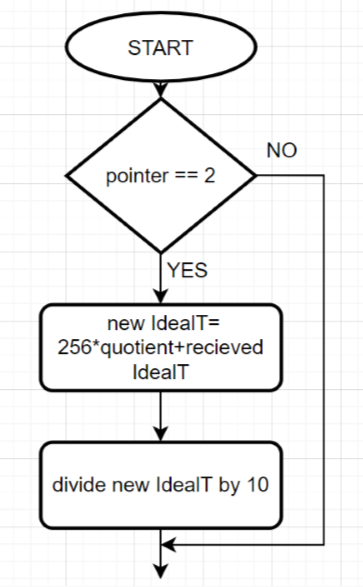
ThingsBoard facilitates the change of set these parameters set points via user inputs in the form of virtual control knobs as shown in the Figure 7. The RPC call is sent once the button is released meaning the user can easily control and set the set value to the one that is desired. The graph in figure 7 shows data that was sent in one week, which includes test data and actual data from the bioreactor sub systems. As for the log, it shows data that was received when the temperature and pH subsystems where connected. It was consistent with all the test messages and the data read by Arduino.

Figure 5: Encoding of temperature received from RPC call

Figure 7: Thingsboard dashboard (temperature and log)

## Stirring Subsystem

**Section author:** Raven Valencia (subsystem lead)

Its purpose is to stir the contents of the bioreactor so that the temperature and pH is distributed evenly throughout the liquid. In order to meet the specifications, the subsystem needs to identify the current RPM of the motor and be subsequently able to increase or decrease the power supplied to adjust the speed.

Figure 6: Calculating the new temperature setpoint

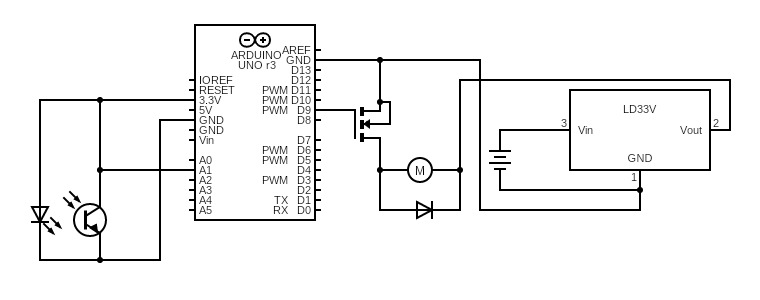
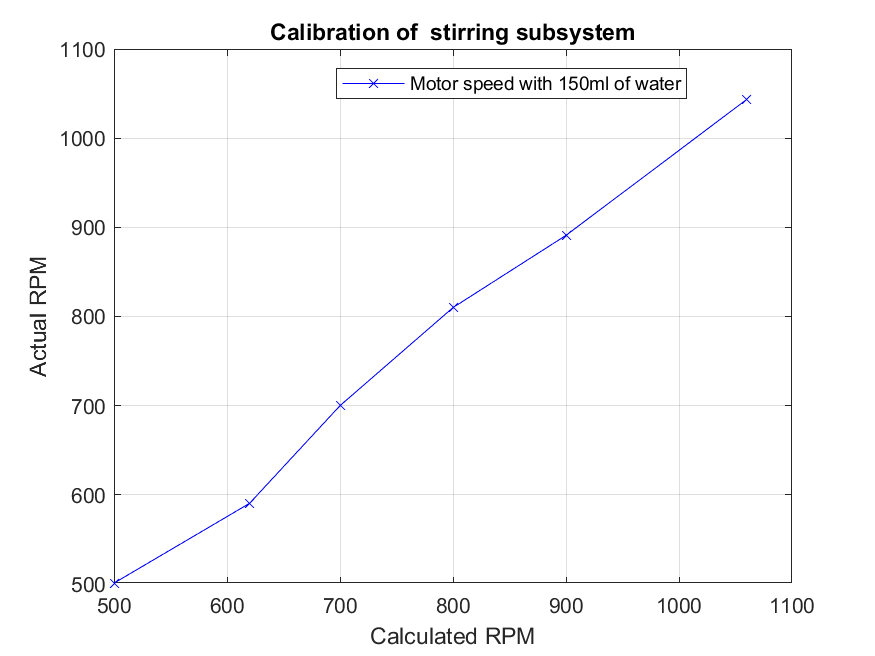
**Design**

Figure 8 shows the proposed circuit diagram. The subsystem consists of a motor and an infrared (IR) emitter detecting reflection caused by the fins on the rotor of the motor. The part on the left of the Arduino UNO represents the photo-interrupter found on the stirring subsystem. Whenever a fin passes over the sensor, it generates a pulse, which leads to 2 pulses per revolution to be generated (there are two fins). The circuit on the right is used to control the motor. The diode is connected in parallel to the motor to block any back voltage generated by the motor, which would damage the MOSFET. [5]

Figure 8: The stirring subsystem schematic

Since there are a few unpredictable sources of disturbance, such as the level of friction or varying load (in the form of solution being stirred), a feedback control system was implemented. The Arduino continuously measures the time between every second pulse and by taking an average of 150 revolutions, calculates the current RPM, incrementally decreasing or increasing the power output if RPM too far from ideal.

**Calibration & validation**

To establish the ground truth, a tachometer was used to measure the actual rpm of the motor and compare against the calculated RPM. As seen in figure 9, the calculated RPM is within 20 of the actual RPM, except for the calculated RPM of 620, where actual RPM was 591. This shows that the calculated RPM is within an acceptable range of 20RPM of the actual value.

Figures and show the process of reaching the desired RPM. The stirring speed gradually increases until it reached the setpoint after ~30s. Once the setpoint is reached, the subsystem controls power to maintain the setpoint RPM, seen especially well when the setpoint is 800: there is only one point where the RPM is above 820. The stirring speed fluctuated significantly at some points due to the motor slowing down due to friction then beginning to increase in speed again, most evidently at t=55 seconds on figure \_, when it got a 100RPM over the setpoint. Sometimes the system causes the power in the motor to increase despite the motor only slowing down temporarily.

Figure 9

Chart, line chart

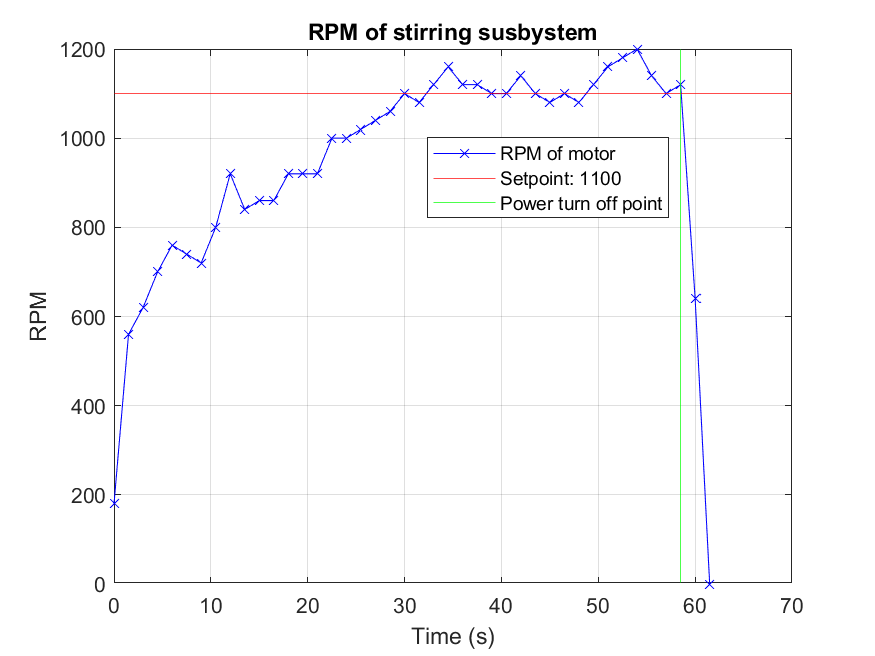
Description automatically generated

Figure 11

Figure 10

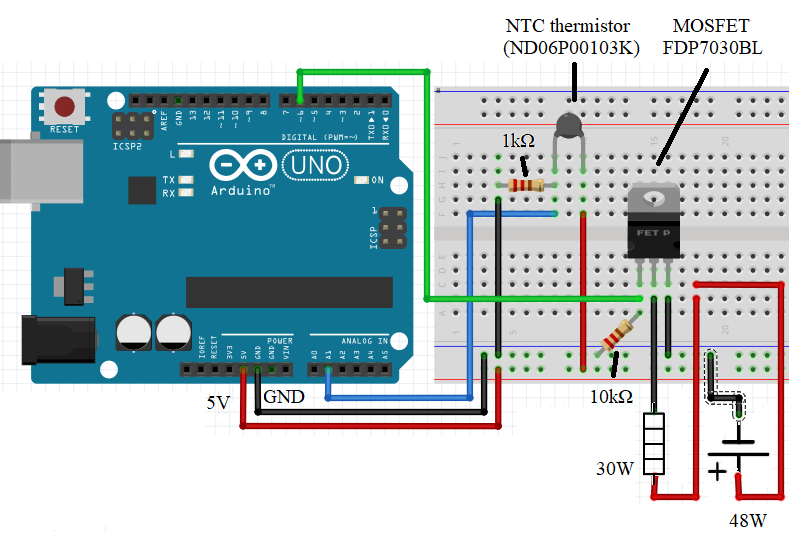
## Temperature subsystem

Authors: Jian Yang ,Yujie Peng, Maciej Kowalski

To maintain a preferred temperature of the bioreactor, the subsystem needs to:

* Measure a temperature-dependent observable (TDO) and interpret it accordingly;
* Apply an adjustable amount of power in the form of heat.

Since the specified temperature range 25-35°C is above the assumed water inlet temperature (15-20°C), there was no need to include a cooling element.

**Design**  
Figure shows the proposed design. Arduino registers the change in voltage across the thermistor (immersed in water) and from that can infer the temperature. If it’s too low, a varying amount of power can be delivered using the MOSFET[[1]](#footnote-2) and an analogue Pin (using Pulse Width Modulation, PWM).

Preliminary tests have shown a significant thermal inertia of the system (increasing temperature by up to 10 degrees after the power has been switched off), thus a proportional control system was adopted.

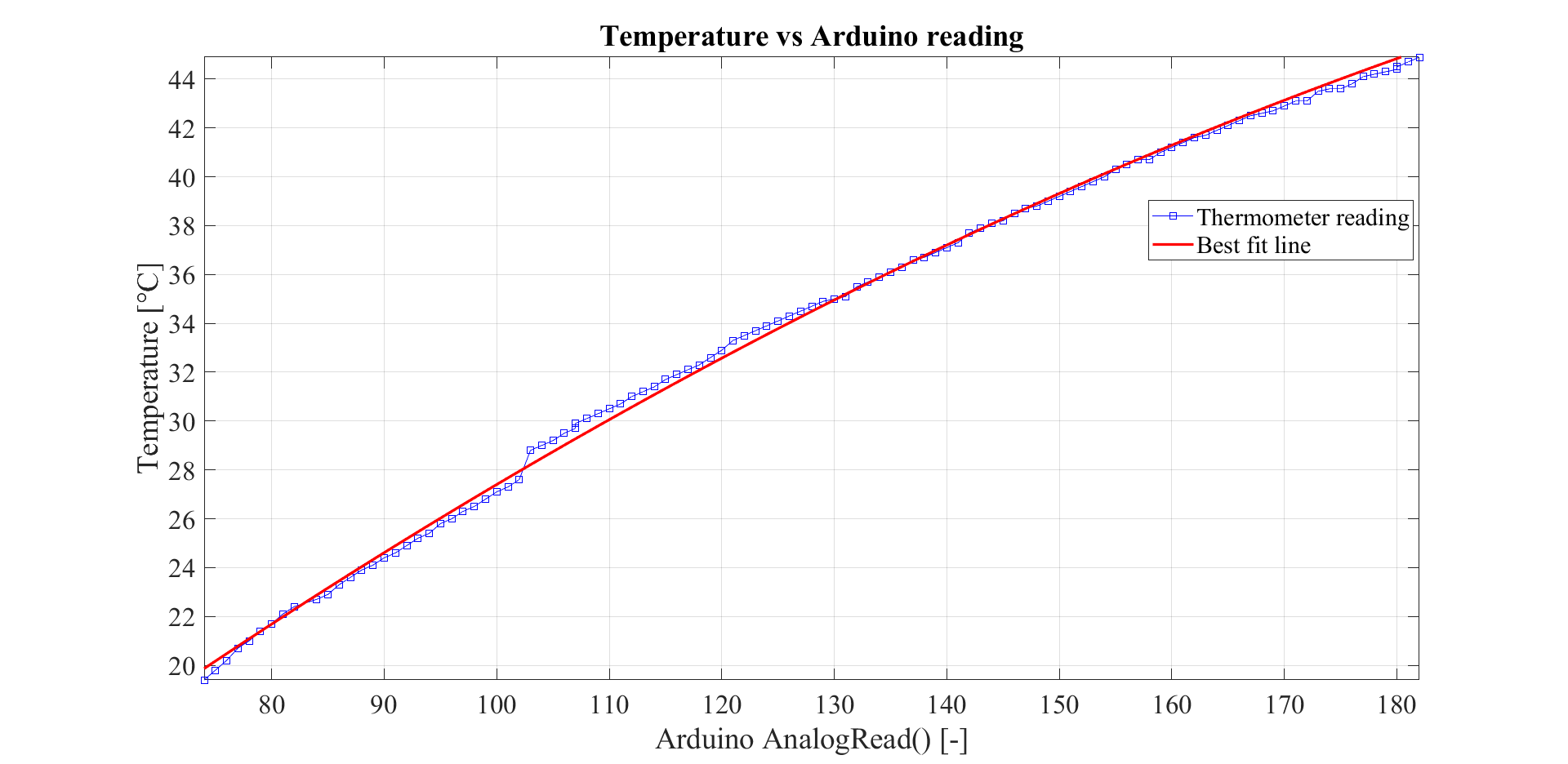
The equation for the PWM value (range 0-255) driving the heater is: . Having considered that the power supply is larger than necessary, conservative values for the constant and the parameter a were chosen. The initial value of *a* is 45, and if the temperature does not increase after 10s, *a* is increased until the power is sufficient. Then after the set temperature is achieved, *a* is reset to its initial value.

Figure 12: Temperature subsystem’s final design. For earlier versions, see Appendix E

**Calibration**

Although the TDO is resistance, the actual observation in this design is the reading of voltage, quantised by Arduino to assume an integer value between 0-1023. Hence, the most direct way of calibration is to measure how this value changes along with temperature. That method is simple and allows to mitigate any rounding errors or oversimplifications of the system.

To achieve this, thermometer was placed in the container to provide the ground truth, and the Arduino was programmed to display the value of the analogRead(). The heater was turned on on full power and the whole scene filmed. The data obtained from the footage is presented in figure 13: (a screenshot of the film is available in Appendix F)



The region is shifted slightly, probably indicating a temporary disturbance in the system. Nevertheless, the best fit line does not differ from the datapoint by more than 0.5°C at any point

Figure 13

**Validation**

When the subsystem was fully assembled during the Team Demonstration, a testing T­­ideal was set to 29.0°C. Initially (T~23.0°C), nothing happened. After about 30s, when *a*  was increased enough, temperature started to increase, however the temperature shown by Arduino lagged behind the reading of the thermometer by up to 0.5°C. Finally, the temperature settled with the thermometer showing ~29.2°C – well within the specified accuracy of ±0.5°C. However, that was the temperature at the bottom of the container. At the top it reached up to 31°C. Since the solution was not stirred, the discrepancy between different parts was greater than the permitted range. The test should have been performed in tandem with the stirring subsystem. Nevertheless, that also revealed the flaw in the calibration – it was also performed with no stirring, most probably biasing the process towards a lower temperature – thermometer was at the bottom, whereas thermistor is in the middle of the container (Appendix F). This is supported by the temperature lagging behind the ground truth during the heating phase.

## pH subsystem

**Section Authors:** Grace Lau Yong En (subsystem lead), Qiren Dong

The pH subsystem consists of a pH probe and two pH pumps to add acid or alkali into the solution to correct the pH.

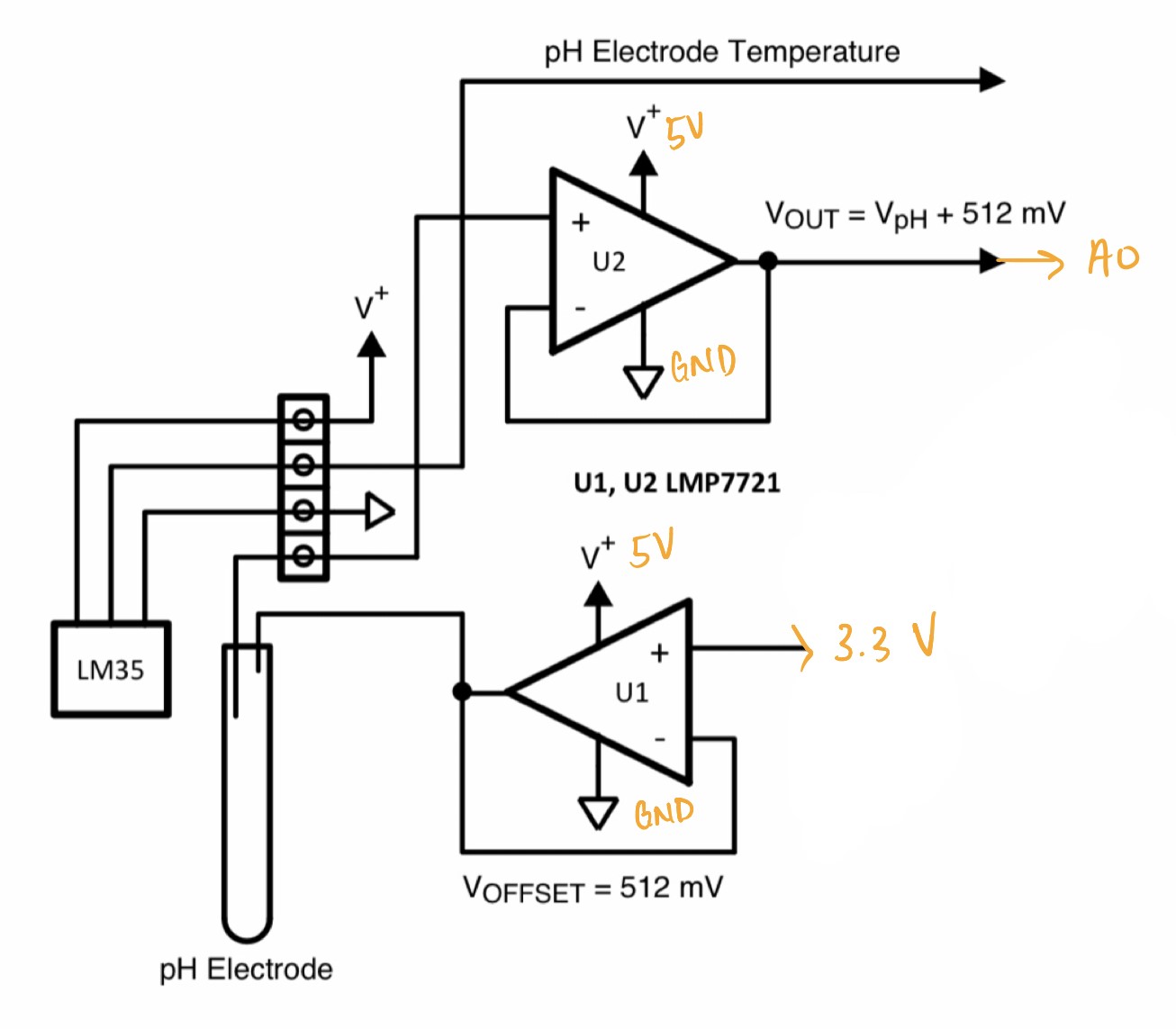
**Design:**

The calibration of the pH probe took far longer than expected, hence the part to correct the pH has not been completed. At the end of this section, a list of actions than would have been done to achieve the aim, can be found.

Figure 14 shows the circuit for pH probe, adapted from [\_]. Since the probe produces a bipolar output, U1 shifts the signal by 3.3V to make it readable to Arduino Uno. U2 serves as a high impedance buffer. LM35 in this design represents an independent temperature sensor, which in this case is replaced by the temperature subsystem (See section 2.3)

**Calibration**

 The probe was calibrated by checking the Arduino reading against 4 buffer solutions of known pH. A linear fit produced the following equation:



**Validation**

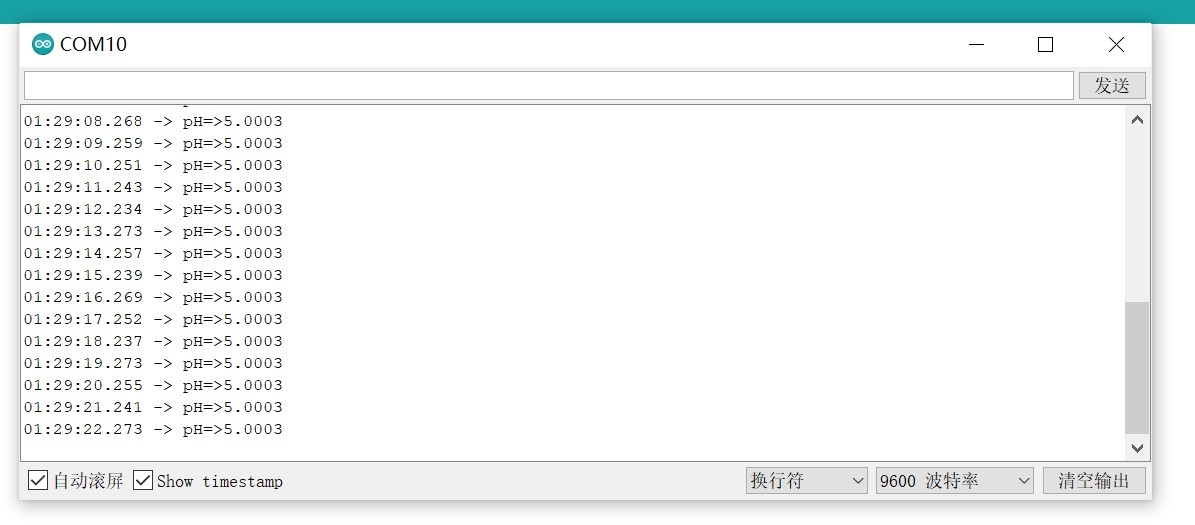
In each trial, the pH value in the computer was printed to confirm the results. The pH subsystem worked properly for the buffer solutions, although it was not tested much against unknown solutions. Figure 15 shows the Serial print produced during the Team Demonstration against the solution of pH=5

Figure 14

Figure 15

# Overall System Integration and Summary

For simplicity of connectivity, the original idea for integration was to connect al the subsystems to one Arduino Uno. Regarding, pH and temperature, no changes were needed and the subsystems worked the same in isolation. Connectivity also proved fully functional managing all three parameters. However, the Team Demonstration showed a problem with the stirring subsystem. Initially, the code included the function delay() while taking measurements. That had to be altered as to not affect the other subsystems. Unfortunately, the alternative code (Appendix ) provided was not able to replicate the results – RPM was always the maximum possible.

In conclusion, the implementation was partially successful. The temperature subsystem was able to roughly read and maintain a constant temperature, but because of a fault in calibration, the accuracy was not high enough. Fortunately, that issue can be easily fixed by improving the calibration – the system is otherwise functional.

For pH subsystem, the part that has been completed appears to work fine – although more measurements are needed to provide certainty. Moreover, further work is required to include the function of correcting pH.

In regards to the stirring subsystem, only the integration part was unsuccessful, and even if the function delay() proved indispensable, there can be always a second Arduino connected to ESP32 via a second bus. That would only require a slight change in the connectivity subsystem code (adding a new slave address), hence giving hope for the issue to be quickly fixed.

# Bibliography

[1] ["Tuberculosis"](https://web.archive.org/web/20130617193438/http:/www.who.int/mediacentre/factsheets/who104/en/print.html). World Health Organization (WHO). 2002. Archived from [the original](https://www.who.int/mediacentre/factsheets/who104/en/print.html) on 17 June 2013.

[2] <https://www.afro.who.int/health-topics/tuberculosis-tb>

[3] <https://moodle.ucl.ac.uk/mod/folder/view.php?id=3269276>

[4] <https://moodle.ucl.ac.uk/mod/folder/view.php?id=3269276>

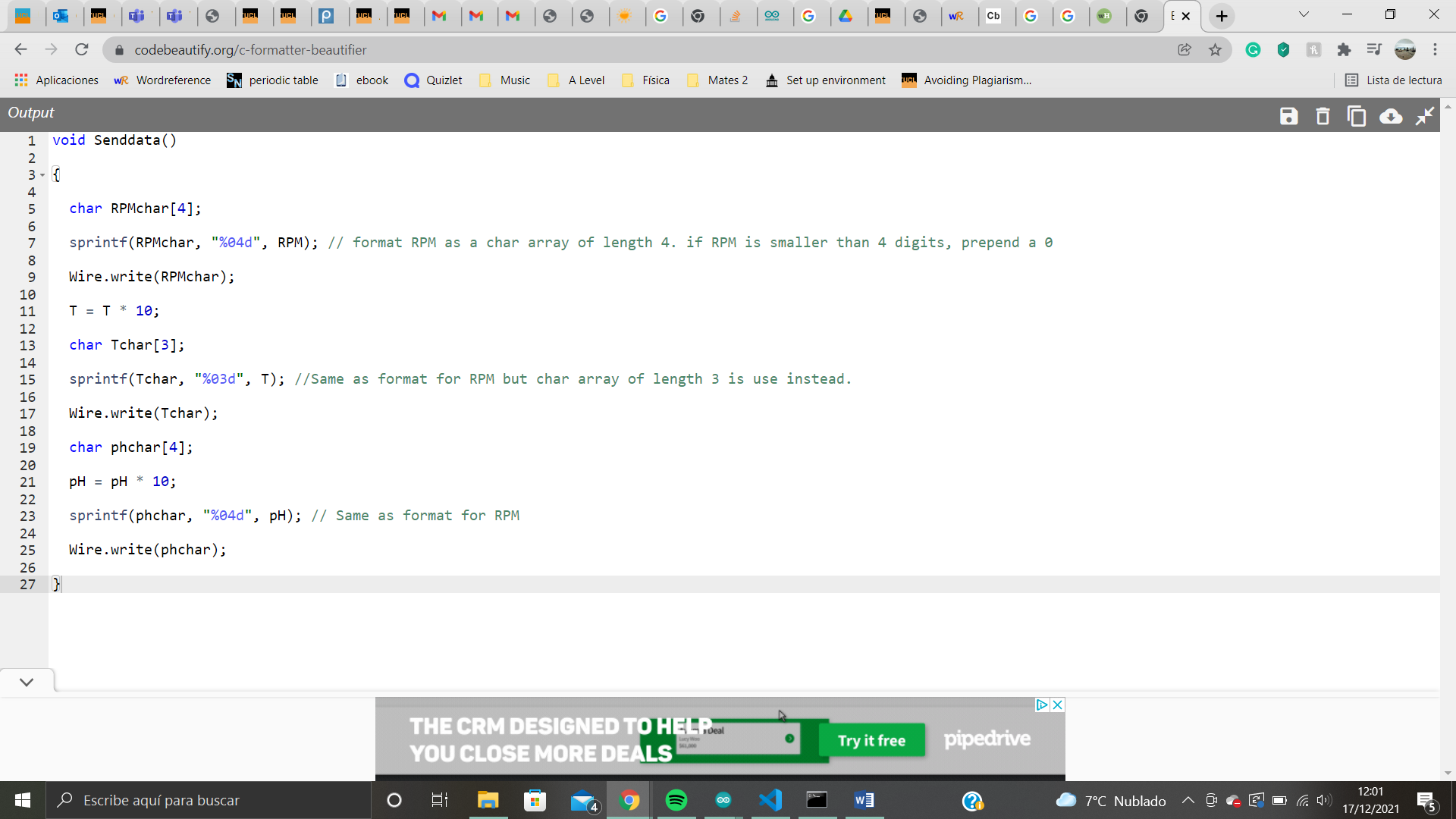
[5] <https://itp.nyu.edu/physcomp/labs/motors-and-transistors/using-a-transistor-to-control-high-current-loads-with-an-arduino/>

[6] temp : <http://www.farnell.com/datasheets/245431.pdf>

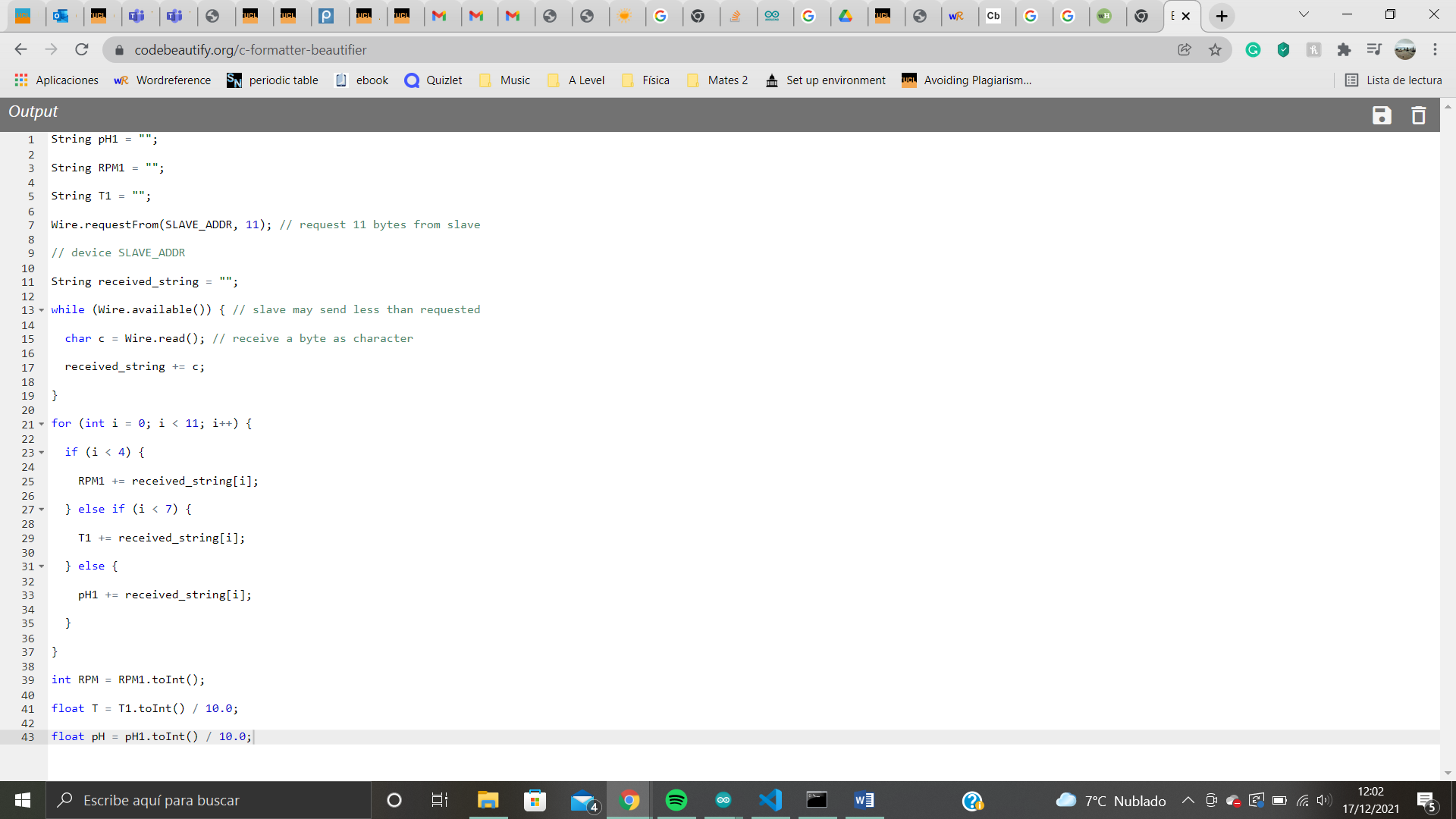
[7] <https://moodle.ucl.ac.uk/pluginfile.php/4362739/mod_resource/content/1/ELEC0002_Arduino%20Uno_Lab%20Script_Updated.pdf>

# Appendices

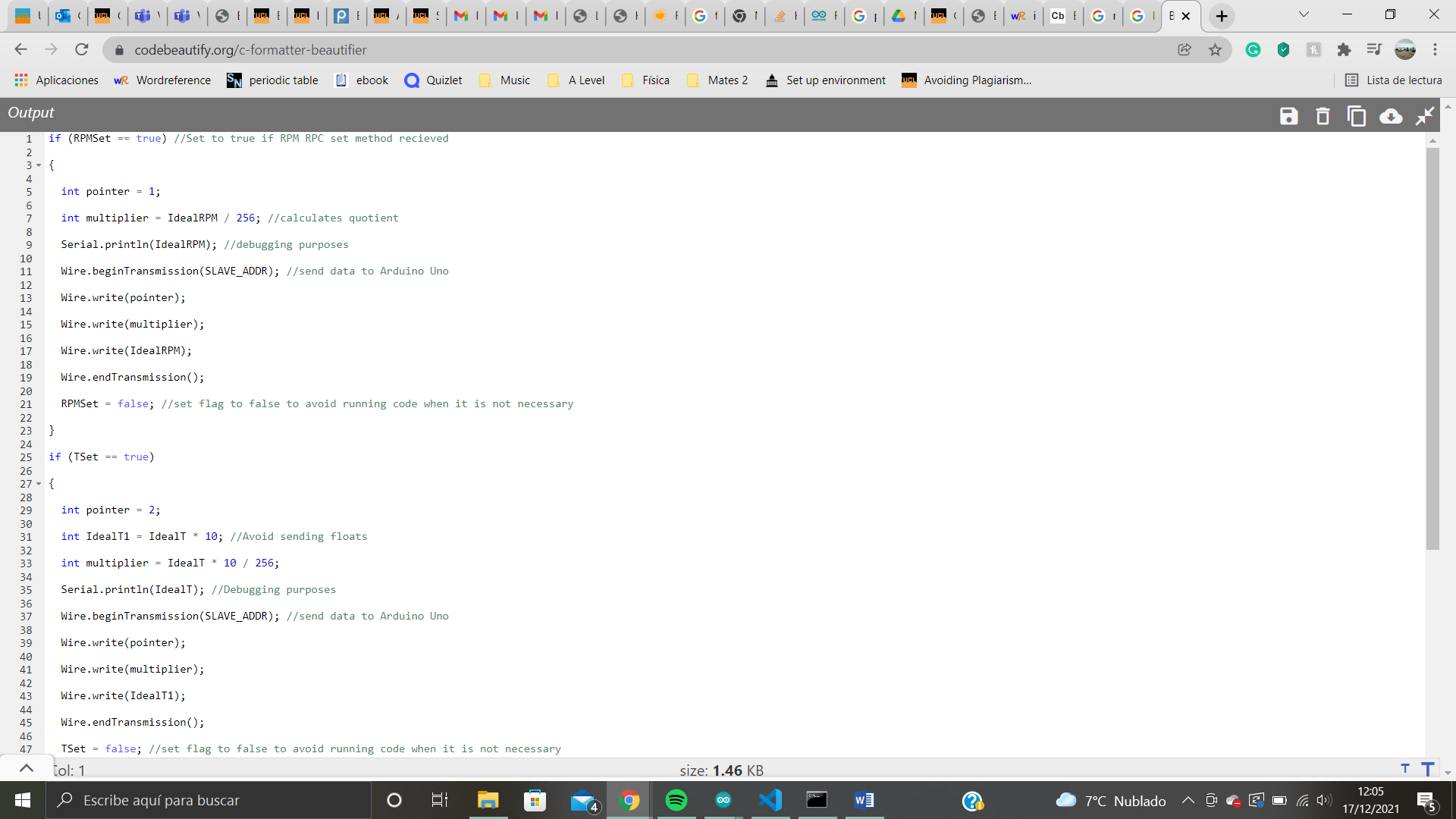
Appendix A:

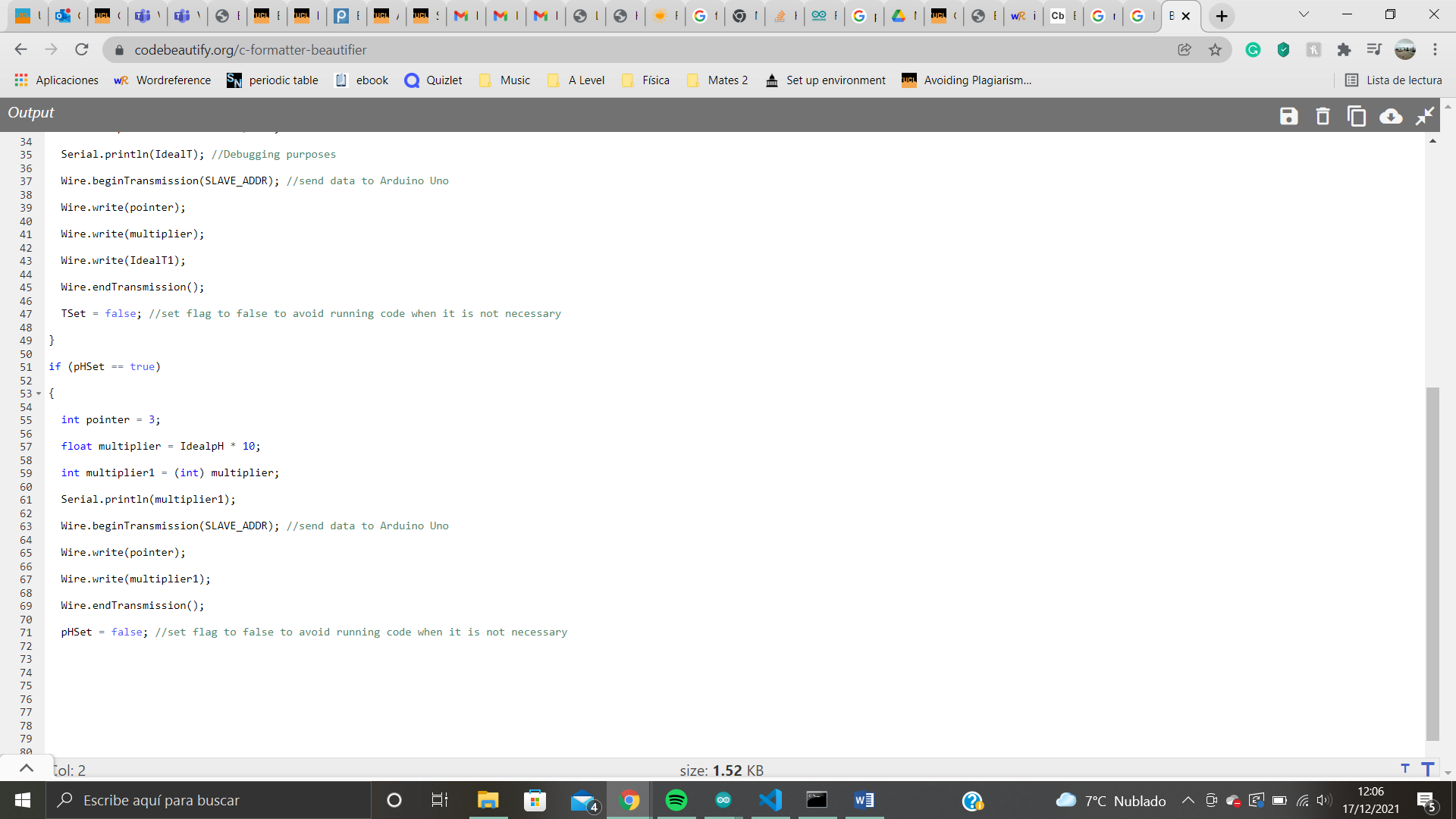


Appendix B:

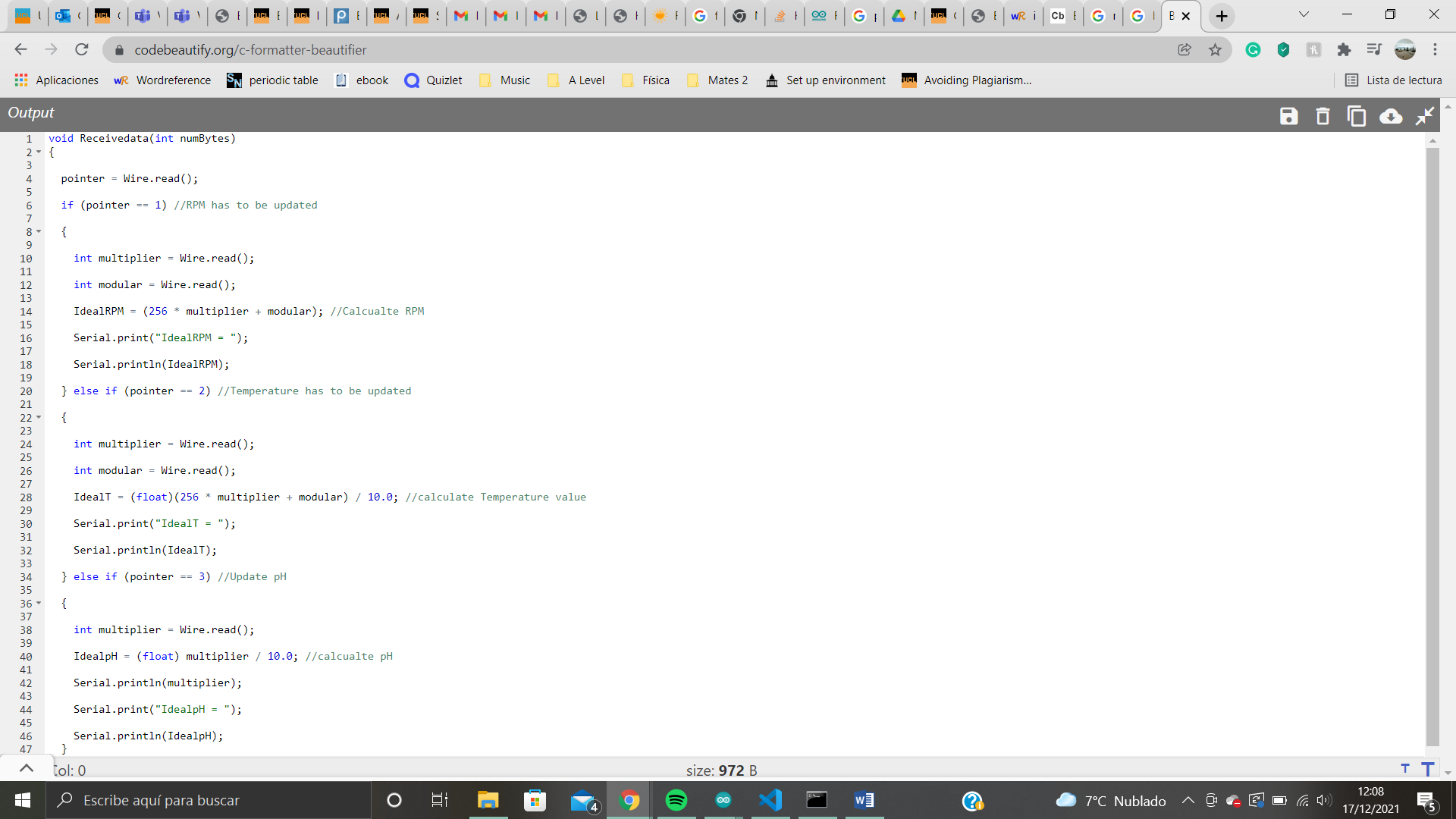


Appendix C





Appendix D



Appendix E

Initially, the circuit did not include the pull-down resistor connecting gate to ground. Later on, the temperature didn’t stop rising even though the power was switched off. Moreover, the transistor was heating up up to 70°C. It turned out to be damaged. After replacing it and adding the pull-down resistor, there were no further issues.

Appendix F



Screenshot of the calibration footage. As can be seen, the thermometer and the thermistor were on different levels in the container, which most probably contributed to the fault discussed in the main body

1. Circuit adapted from [6] [↑](#footnote-ref-2)