Heating subsystem

For the heating subsystem we first have the ideal temperature that is recorded by the user in the user interface. This is stored in the server and the ESP 32 can access this and send the ideal temperature is sent to the Arduino. The Arduino gets the current temperature from the thermistor. We are using a 10 kOhm thermistor that will be submerged within the solution so that it can detect the temperature changes. With a change in temperature, there is a change in resistance. Positive Temperature Coefficient (PTC) thermistors will in increase in resistance when the temperature increases, however Negative Temperature Coefficient (NTC) thermistors will decrease in resistance when the temperature increases and vice versa. We will need to get the temperature resistance graph for this specific thermistor so that we can get the temperature for the current resistance using that graph. If this graph for this thermistor does not exist, then we will have to make the graph ourselves by seeing how the resistance of the thermistor varies with temperature.

By using this thermistor, we can send the temperature back to the user to display on the user interface. The temperature is also sent back to the Arduino so that it can send the correct instruction to vary the voltage so that we can achieve the ideal temperature. This is known as negative feedback. The instruction for varying the voltage goes to the supply and the power supply varies the power that goes to the heating element. This will vary the temperature. For this we use a 3Ohm, a max of 30W thermistor.

In summary the thermistor can detect the temperature using variance in resistance and the heating element will vary the temperature.

pH subsystem

For the pH subsystem, we get the current pH from the pH probe. The pH probe is inserted into the solution and so can detect the level of pH and send it to the Arduino for analysis. The way that the pH probe does this is that it records a voltage from the hydrogen ions and the pH meter that converts voltage to a pH value. This can be done by finding the difference between the pH probe’s electrode and the internal electrode.

Then the Arduino reads the pH and sends back an instruction to the pH pump so that it can pump more of an acid or an alkali to the solution to change the solution. So, for example if the ideal pH is 7 (neutral) and the current pH is 3, then we would need to pump alkaline from the bottles to the solution to neutralise it. So, we would keep pumping alkaline until the correct pH is reached where the Arduino would stop sending the pump command.

One of the problems that can occur is that the bottles between the alkaline and acid could get mixed up. So, we would need for the pump to be separate for the acid and alkaline bottles. Also, the command sent from the Arduino will need to specify if the solution needs alkaline or acid so that the correct pump can be used.

Connectivity and User Interface

For this the user interface will take the commands for the ideal temperature/pH/stirring RPM of the user, so that this can be sent to the server for the ESP32 to read. The user interface will also display the current temperature/pH/stirring RPM of the equipment. This can be shown in multiple ways for the user such as a line graph that can show the variation in the temperature. There will be an input section for the user so that they can change either the temperature, pH or the stirring RPM. This data will then be sent to the server via a certain internet protocol such as HTTP or HTTPS if we want the message to be secure. We have decided to use Things Board for this to make the setup for the UI easier and the connection to the server simpler and we don’t need to worry about the communication protocol between the server and the user interface.

The ideal values for temperature/pH/stirring must also be stored on the server in an organised fashion. So, what we can do is store the values in an SQL database. This means that we can do an SQL query to get the data and SQL update to change the database. However, one of the issues that can arise is that there might be an attempt to access the same table while another process is trying to modify it. To overcome this, we can introduce a method like putting the commands into a queue and executing them one at a time.

Then the server needs to be able to talk to the ESP 32 and there are multiple protocols that we can utilise to allow for the communication. One good example is the MQTT (MQ telemetry transport) protocol which is a lightweight internet protocol for IoT devices. This is typically used for low-bandwidth devices like ESP 32’s so that it makes it easier for the devices to talk. This probably makes it more preferrable to more of the heavy-duty WIFI protocols. In this protocol there is a subscriber and publisher for the sending and receiving of data.

Then we need to allow for the data to be sent from the ESP 32 to the Arduino. The two devices will relate to a wire so will be using wired protocols. One we have considered is the I2C (I to C) protocol that allows for two-way serial communication between the Arduino and the ESP 32. Here there is a bus with the controller being the ESP32 and the reader being Arduino. However, since there are two devices talking to each other, the reader and the controller need to switch to allow for both to communicate.

Flowchart

For the heating subsystem, first we get the needed temperature from the server. Then we calculate the resistance needed with that ideal temperature. Then using this data, we calculate the proportion of voltage across the R5 pin. Then we finally compare the real time temperature and the ideal temperature needed. If it is less, then a PWM signal is released from the Arduino. This modulation signal allows for the heater to get a signal to increase the temperature. And LED is also turned on to signal that the heater is on. Then at the end, the voltage is sent back to the Arduino in a negative feedback loop. Once the desired heat is reached the Arduino will stop sending the PWM signal.