INTEGRATED TERRARIUM REPORT

GROUP Q

ELEANOR ROBERTS, JAMES BURGESS, ANA HERNANDEZ, WERNER LANDMAN

Contents

STATEMENT OF CONTRIBUTIONS

This statement confirms each individual's contribution to the project.

Ana Hernandez:
- Humidity & Temperature subsystem
- Code combination and write up
- Adding Wi-Fi to code
- Testing
- Video

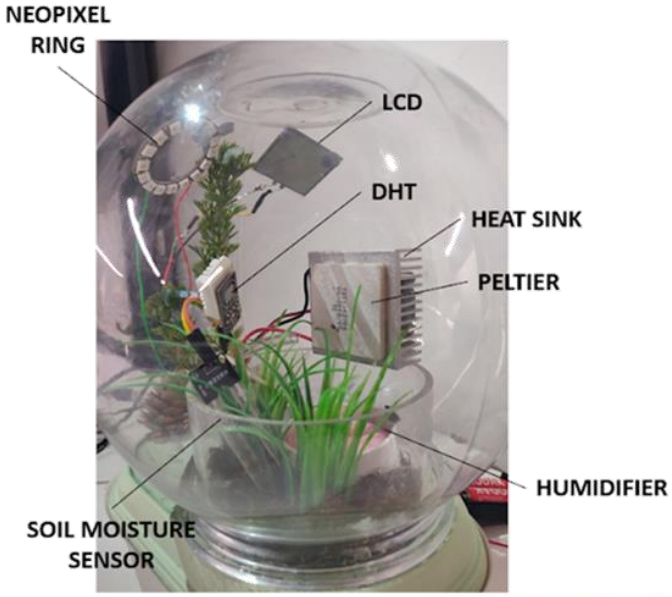
Eleanor Roberts:
- Soil moisture subsystem
- Building terrarium
- Report write up
- Circuit diagrams

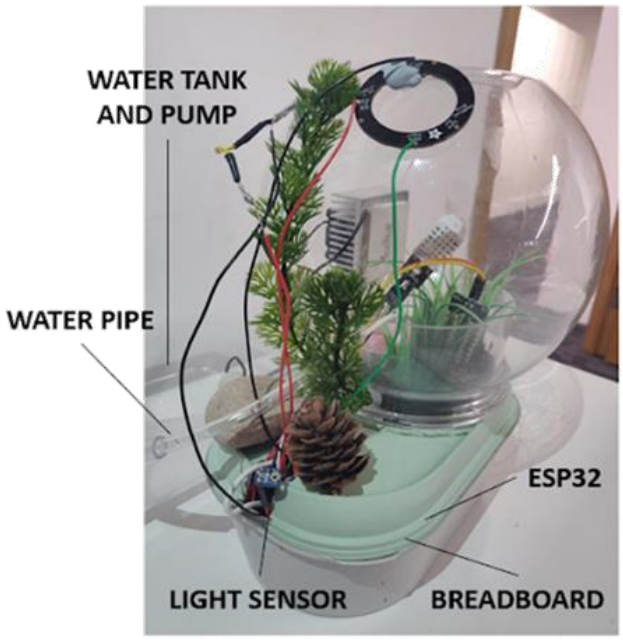
James Burgess:
- Light subsystem
- Building terrarium
- Testing
- Limitations write up

Werner Landman:
- Temperature subsystem
- App and write up

PROTOTYPE

The final terrarium can be seen below. The bulk of the circuitry is contained within the lower part of the terrarium for neatness and to protect from any water sources. The wires for the components within the terrarium are fed through a hole in the side to keep them neat. The soil moisture sensor rests in the soil with the water pump and tank are located behind the base of the terrarium and are fed into the terrarium through a hole in order to keep the pipe securely above the coffee plant. The humidifier sits in a pool of water below the Peltier. The Peltier, heat sink and fan are located centrally within the terrarium. The fan faces outwards so it does not blow directly onto the humidity and temperature sensor and to encourage air circulation throughout the whole terrarium, rather than straight onto the coffee plant, which could cause a dramatic temperature change which would be detrimental to the plant. The temperature sensor and humidity sensor is also located as close to the plant as possible in order to get an accurate reading. The neopixel ring is located near the top of the terrarium to mimic the way the sun would hit a coffee plant, they like light to semi-shady conditions. The LCD is on the outside of the terrarium to make it easy to demonstrate the light subsystem working, ideally the whole terrarium would be made of LCD shutter glass. Additionally, the light sensor is located just outside of the terrarium to make it easily demonstrable.





CONTROL APP

The proposed application would be able to control pre-programmed logic and display sensor values of the integrated terrarium sub-systems. Design and logic features are done through the MIT App Inventor 2 programme. Testing of said logic is through the MIT app companion. Thus, the creation process consisted of two application versions (for systematic improvement of functions through testing and feedback). Ideally, the proposed app would display the values from the various sensors connected to the ESP32, display the current functionality status of each of the sub-systems and allow interaction with actuators of the integrated system.

VERSION 1 - SIMPLISTIC

Each sub-system would have three information displays: current, target and action. Current describes the value from the sensor (the information source would be similar to the serial monitor function for the Arduino). Target is the optimum value that the system tries to maintain (static value that is not subject to change). Action displays what the specific sub-system is currently doing e.g. temperature is heating or moisture is idle. This function is just to show that a sub-system is working and removes the needs for individual light emitting diodes for each part. From testing with the Arduino serial monitor, it is possible to display specific information. For the ESP32, the web viewer user interface will enable any display of the desired values through creating “mini webpages” via Wi-Fi. To reduce the processing power required for this application, all displayed values would refresh every 2 minutes.

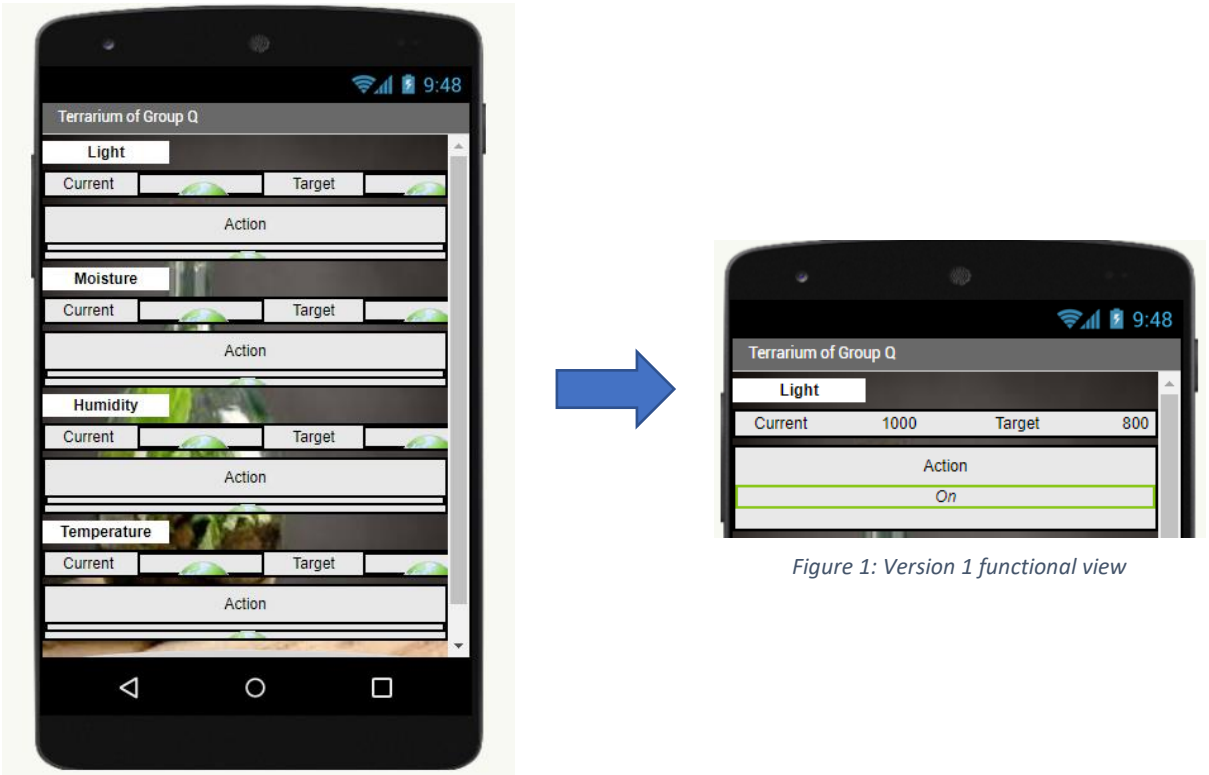
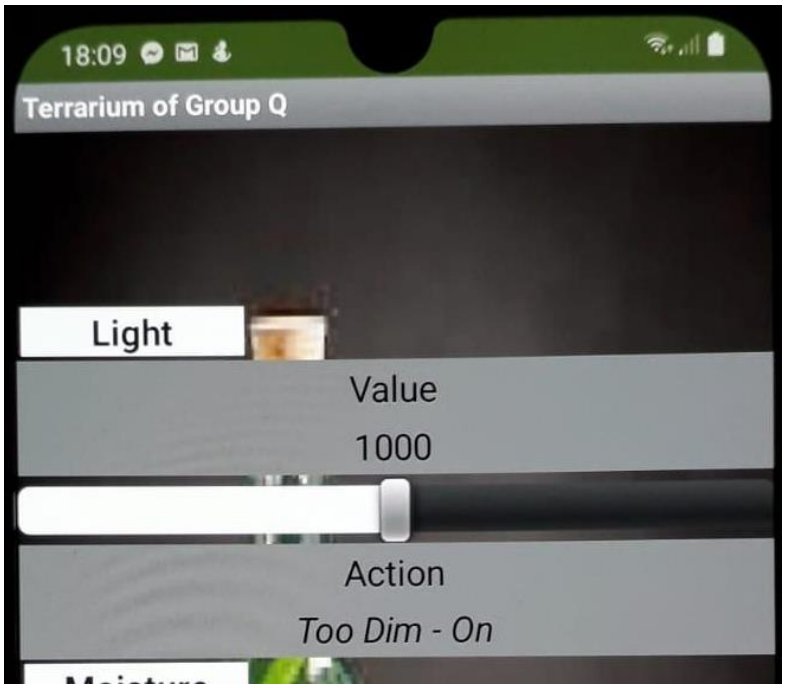


Figure 1: Version 1 functional view

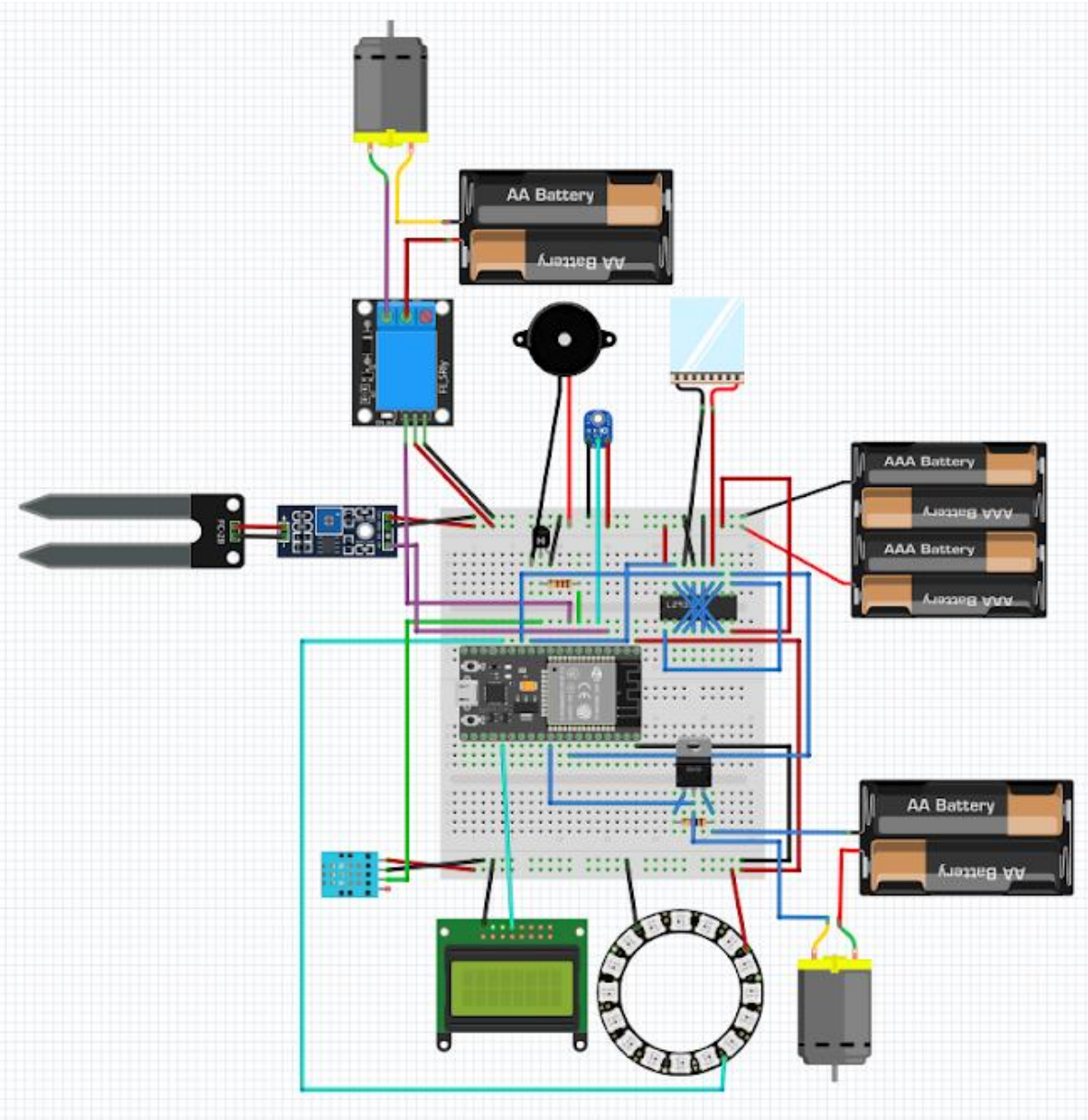
VERSION 2 - REFINED

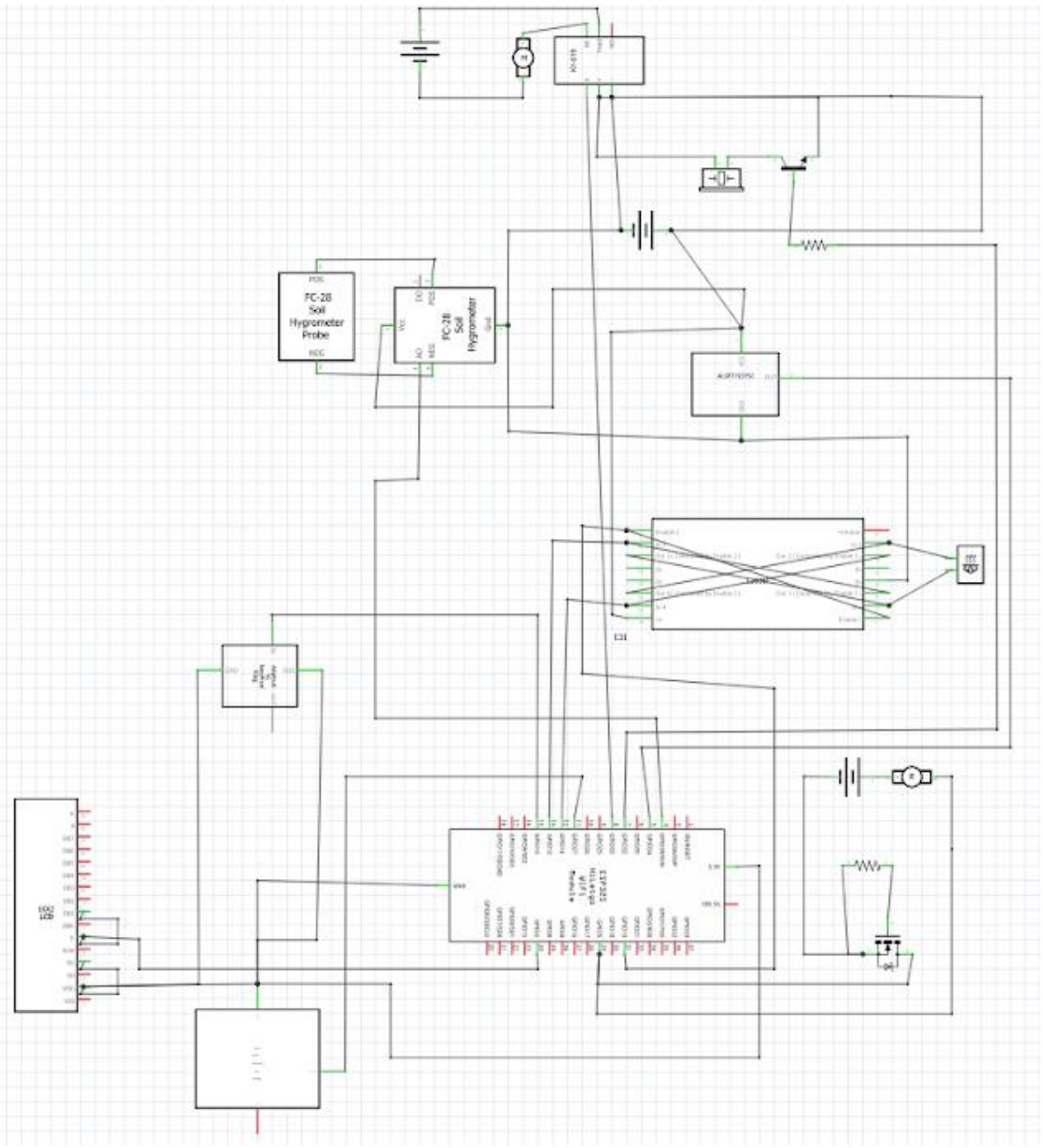
The communication between the app and ESP32 will remain the same. The user interface is different for this version. Based on feedback from group members, the interface should be more simplified. Another condition is the need for user interaction. Thus, version two of the app is controlled by both the user's input or through the integrated system. This means that the app displays readings through a text block and specifies it through a specified slider bar position. If the user interrupts this process, through manual input to a specified value, the app will either increase or decrease the current reading to the set value. Note that the manual input values is limited to values that cannot damage the system. After achieving the target, normal function will ensue to maintain pre-programmed target values.



INTEGRATED CIRCUIT

The integrated circuit diagram and pin out circuit can be seen below; it has been developed using fritzing and displays which components are connected to each pin of the esp32. The water pump and humidifier could not be represented on fritzing so other components have been used in their place; the water pump and fan are displayed as DC motors and the donut humidifier is displayed as a buzzer.

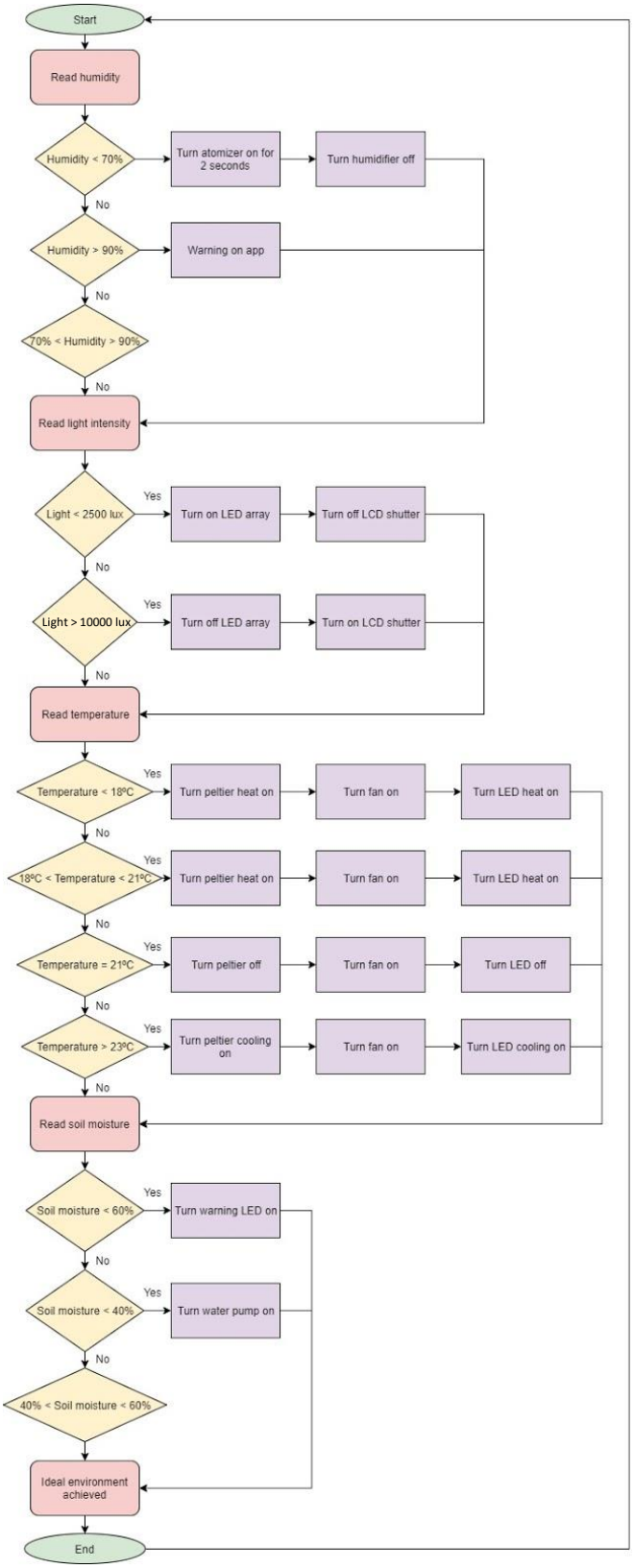



FC-28
Soil
Hygrometer
Probe
FC-28
Soil
Hygrometer

INTEGRATED CODE

FLOW DIAGRAM

The flow chart seen below describes the sequence of actions which are executed by the code. The code runs through each subsystem and applies the required corrective actions in turn before running through each subsystem again, this occurs on a loop.



CODE

The final code with Wi-Fi connectivity for maintaining the ideal environment for an arabica coffee plant can be seen below. See the comments for details.

```
// Load Wi-Fi library
#include <WiFi.h>
//Network details
const char* ssid = "Terrarium";
const char* password = "123456789";

// Set web server port number to 80
WiFiServer server(80);
// Variable to store the HTTP request
String header;

//Light

#include <Adafruit_NeoPixel.h>
#define PIN 13
#define NUM_LIGHTS 16
#define LightsensorPin 34
#define LCDPin 2

const int Min_Light = 1023;
const int Max_Light = 3595;

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LIGHTS, PIN, NEO_GRB + NEO_KHZ800);

//Humidity & Temperature

#include <DHT.h>//Include DHT sensor library
#define DHTPIN 27 //Define what pin the sensors digital output is connected to
#define DHTTYPE DHT22 //Define type of DHT sensor
DHT dht (DHTPIN, DHTTYPE);

const int Min_Humidity = 70;
const int Max_Humidity = 90;

#define AtomizerTransistorPin 32
#define MOSFETPin 5
#define PowerHBridgePin 19

const int Min_Temperature = 20;
const int Max_Temperature = 25;

//Soil Moisture

#define WaterPumpRelayPin 33                  //Sets digital pin 33 to the relay module
#define SoilMoistureSensorPin 39              //Sets Analog pin to the soil moisture sensor

const int IdealDrySoil = 3320;               //Sets the upper threshold for ideal soil moisture
const int IdealWetSoil = 2980;                //Sets the lower threshold for ideal soil moisture

void setup() {

  //WIFI

  Serial.begin(115200);

  Serial.print("Setting AP (Access Point)…");


  IPAddress IP = WiFi.soft.AP();
  Serial.print("AP IP address: ");
  Serial.println(IP);

  server.begin();
```

```
  // Humidity
  dht.begin();
  pinMode (32, OUTPUT); //Set atomizer as output pin 5
  digitalWrite(32, LOW); //Start the system with the atomizer turned off

  //Temperature
  pinMode(MOSFETPin, OUTPUT);
  pinMode(12, OUTPUT); //Peltier
  pinMode(14, OUTPUT); //Peltier
  digitalWrite(19, HIGH);//set this pin to HIGH to enable HBridge
  digitalWrite(5, LOW); //Set the MOSFETPin, and therefore the fan, to be off
  digitalWrite (14, LOW); //Set the peltier to start off
  digitalWrite (12, LOW); //Set the peliter to start off

  //Light
  pinMode(LCDPin, OUTPUT);
  pinMode (LightsensorPin, INPUT);
  strip.begin();
  strip.show(); //Initialize all pixels to off

  //Soil Moisture
  pinMode(SoilMoistureSensorPin, INPUT);
  pinMode(WaterPumpRelayPin, OUTPUT);
  digitalWrite(WaterPumpRelayPin, HIGH);

}

void loop() {

  //WIFI
  WiFiClient client = server.available(); // Listen for incoming clients
  if (client) { // If a new client connects,
    Serial.println("New Client."); // print a message out in the serial port
    String currentLine = ""; // make a String to hold incoming data from the client
    while (client.connected()) { // loop while the client's connected


      //Humidity

      float Humidity = dht.readHumidity();              //Read humidity
      Serial.print("Humidity: ");               //Set the display of humidity readings
      Serial.print(Humidity);
      Serial.println("%");
      Serial.println(" ");                      //Line between values

      if (Humidity < Min_Humidity) {                  //Set condition for humidity being below the minimum
        Serial.println("Humidity too low");
        digitalWrite(32, HIGH);                 //If humidity is below minimum turn atomizer on
        Serial.println("Atomizer on");
        Serial.println(" ");                      //Line between values
        delay(1500);
        digitalWrite (32, LOW);                   // Turn atomizer off
      }
      if (Humidity > Max_Humidity) {                      //Set condition for humidity being below the maxi
        Serial.println("Humidity too high");
        Serial.println(" ");
        digitalWrite(32, LOW);                    //Make atomizer be off
      }
      if (Min_Humidity < Humidity > Max_Humidity) {
        digitalWrite (32, LOW);                   //Atomizer off
        Serial.println("Humidity OK");
        Serial.println(" ");                     //Line between values
      }

      //Temperature

      float temperature = dht.readTemperature();
      Serial.print("Temperature: ");
      Serial.print(temperature);
      Serial.println("°C");
      Serial.println(" ");
```

```
  if (temperature < Min_Temperature)
  {
    Serial.println ("Too cold");
    digitalWrite(5, HIGH);              //Turn fan on
    digitalWrite(12, LOW);              //Heats heat sink side
    digitalWrite(14, HIGH);              //Cools 'exposed side'
    Serial.println("Heating");
    Serial.println(" ");                            //Line between values
  }
  if (Min_Temperature < temperature > Max_Temperature)
  {
    digitalWrite(5, LOW);                //Turn fan off
    digitalWrite(12, LOW);              //Peltier off
    digitalWrite(14, LOW);               //Peltier off
    Serial.println("Temperature OK");
    Serial.println(" ");                      //Line between values
  }

  if (temperature > Max_Temperature)
  {
    Serial.println ("Too hot");
    digitalWrite(15, HIGH);               //Turn fan on
    digitalWrite(12, HIGH);             //Cools heat sink side
    digitalWrite(14, LOW);               //Heats 'exposed side'
    Serial.println ("Cooling");
    Serial.println(" ");                            //Line between values

  }

  //Light

  int sensorValue = analogRead(37);             // read the input from analog pin 0:
  int Light = analogRead(LightsensorPin);           // defines light as input value

  float voltage = Light*(5.0 /4096.0);   // convert sensor value into voltage, (voltage)/(no. of bytes)
  float microamps = (voltage / 1000.0) * 1000000.0; // convert value into microamps so Lux value can be read from graph (I=V/R)
  float LUX = (microamps / 0.5);                   // converting light current value into illuminance (LUX)


     //Soil Moisture

     float MoistureLevel = analogRead (SoilMoistureSensorPin);       // Reading value from soil moisture sensor

     //Prints the value read by sensor i.e. moisture level, in the serial monitor window
     Serial.print ("Moisture Level: ");
     Serial.println (MoistureLevel);

     if (MoistureLevel < IdealWetSoil) {                          // MoistureLevel drops below 675, pump is off, moisture level is too wet
       Serial.println("Too moist");
       digitalWrite (WaterPumpRelayPin, HIGH);                     // 5V power goes to the relay module breaking the circuit = Water Pump off
     }

     else if (MoistureLevel > IdealDrySoil) {                    // MoistureLevel goes above 787, pump on, moisture level is too dry
       Serial.println ("Too dry");
       digitalWrite (WaterPumpRelayPin, LOW);                    // 0V power goes to the relay module, thus connecting the circuit = Water Pump on
       Serial.println ("Water Pump On");
       Serial.println(" ");                                       //Line between values
     }

     else if (MoistureLevel < IdealDrySoil) {                    // MoistureLevel drops below 787, pump off, moisture level is within ideal range
       digitalWrite(WaterPumpRelayPin, HIGH);                     // 5V power goes to the relay module, thus breaking the circuit
       Serial.println ("Moisture OK");
       Serial.println(" ");                     //Line between values
     }

     ////////////////////////////////////////////////////////////////WIFI

// Variables

int  mintemp = Min_Temperature;
int  currenttemp = temperature;
int  maxtemp = Max_Temperature;
int  minlight = Min_Light;
int  currentlight = Light;
int  maxlight = Max_Light;
int  minhumidity = Min_Humidity;
int  currenthumidity = Humidity;
int  maxhumidity = Max_Humidity;
int  minsoilmoisture = IdealWetSoil;
int  currentsoilmoisture = MoistureLevel;
int  maxsoilmoisture = IdealDrySoil;
*/
```

```
Serial.print("Light Intensity: ");             // prints prior to sensor value
Serial.print(LUX);                             // print out the value calculated
Serial.println(" lux");                          // prints units of value read
Serial.println(" ");                           //Line between values

uint32_t low = strip.Color(0, 0, 0);           //set colour of neopixel ring for low
uint32_t high = strip.Color(255, 0, 255);      //set colour of neopixel ring for high


if (Light < Min_Light) {                              // condition if light value less
  Serial.println("Too dim");

  for ( int i = 0; i < NUM_LIGHTS; i++) {
    strip.setPixelColor(i, high);
    strip.show();                              // Turn on Led light source
  }
  digitalWrite(LCDPin, LOW);            //Set LCD to be off
  Serial.println ("Light on");
  Serial.println(" ");                      //Line between values
}


if (Light >= Min_Light && Light <= Max_Light) {      // condition if light value in between optimal level

  for ( int i = 0; i < NUM_LIGHTS; i++) {
    strip.setPixelColor(i, low);            // Turn off Led light source
    strip.show();
  }
  digitalWrite(LCDPin, LOW);                  // Turn off LCD shutter
  Serial.println("Light Intensity OK");
  Serial.println(" ");                      //Line between values

}

if (Light > Max_Light) {                                        // condition if light value above 921, (10,000 lux)
  Serial.println ("Too bright");
  digitalWrite(LCDPin, HIGH);                // Turn on LCD Shutter
  Serial.println ("LCD On");
  Serial.println(" ");                           //Line between values
  for ( int i = 0; i < NUM_LIGHTS; i++) {
    strip.setPixelColor(i, low);
    strip.show();
  }                          // Turn off Led light source
}

WiFiClient client = server.available();   // Listen for incoming clients

if (client) {                             // If a new client connects,
  Serial.println("New Client.");          // print a message out in the serial port
  String currentLine = "";                // make a String to hold incoming data from the client
  while (client.connected()) {            // loop while the client's connected
    if (client.available()) {             // if there's bytes to read from the client,
      char c = client.read();             // read a byte, then
      Serial.write(c);                    // print it out the serial monitor
      header += c;
      if (c == '\n') {                    // if the byte is a newline character
        // if the current line is blank, you got two newline characters in a row.
        // that's the end of the client HTTP request, so send a response:
        if (currentLine.length() == 0) {
          // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
          // and a content-type so the client knows what's coming, then a blank line:
          client.println("HTTP/1.1 200 OK");
          client.println("Content-type:text/html");
          client.println("Connection: close");
          client.println();



          // Display the HTML web page
          client.println("<!DOCTYPE html><html>");
          client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
          client.println("<link rel=\"icon\" href=\"data:,\">");
          // CSS to style the page and labels
          // Feel free to change the background-color and font-size attributes to fit your preferences
          client.println("<style>html {display: inline-block; margin: 0px auto; text-align: left;}");
          client.println("label.OK { float:right;background-color: #66d05e; border: none; color: white;}");
          client.println("label.alarm {float:right; background-color: #fc00ff;color: black;}</style></head>");
```

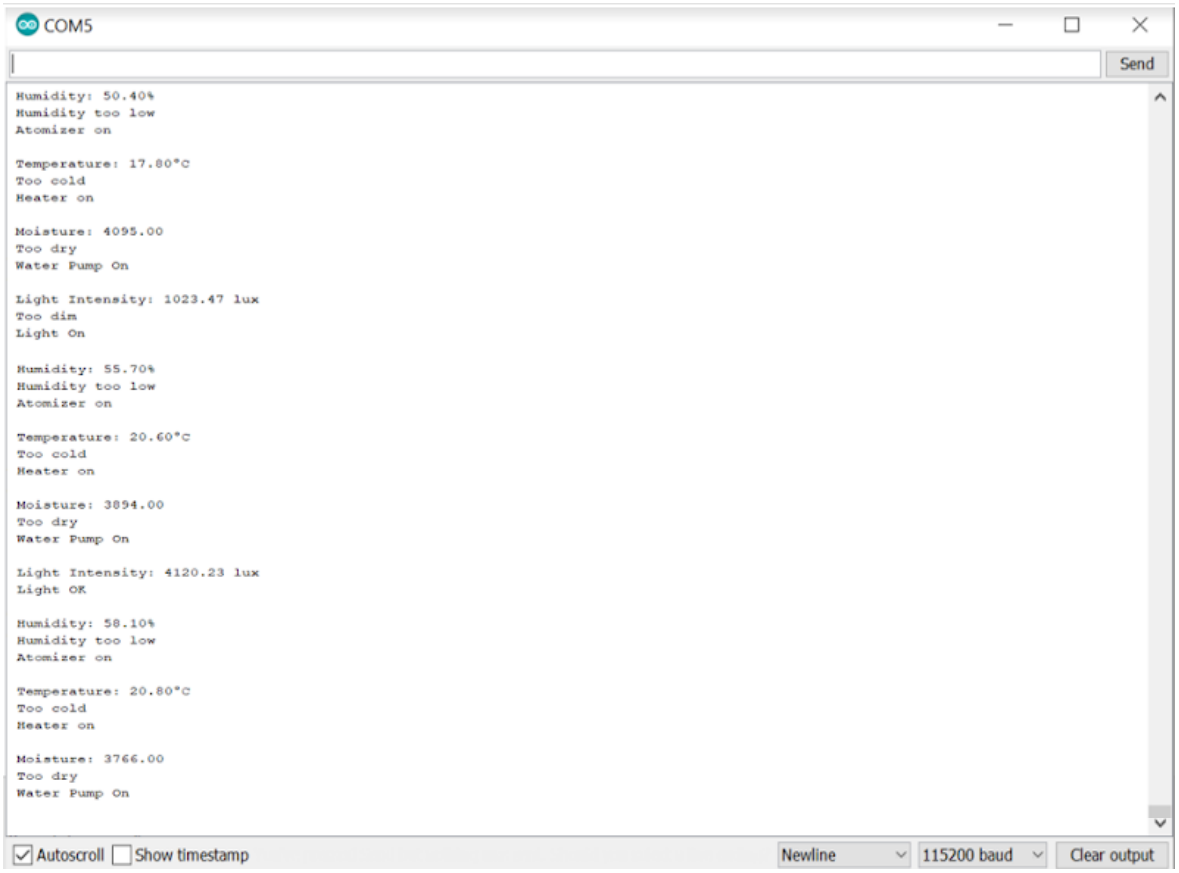
```
// Web Page Heading
client.println("<body><h1>ESP32 Terrarium Monitor</h1>");

// Display current values
//centers text on page and limits width of block
client.println("<div style='width:375px; margin:auto;'>");

//this block prints a slider with max min and current value for temperature
client.println("<div>");
client.print(mintemp);
client.print("<input name='Temperature' type='range' min='");
client.print(mintemp);
client.print("' max='");
client.print(maxtemp);
client.print("' value='");
client.print(currenttemp);
   client.print("' />");
   client.print(maxtemp);
   client.print("<label for='Temperature' ");
   if (currenttemp > maxtemp || currenttemp < mintemp) {
     client.print("class='alarm' >");
     client.print("WARNING ");
   } else {
     client.print("class='OK' >");
   }
   client.print("Temperature (C) = ");
   client.print(currenttemp);
   client.print("</label></div><br/><br/>");


   //this block prints a slider with max min and current value for Light level
   client.println("<div>");
   client.print(minlight);
   client.print("<input  name='Light' type='range' min='");
   client.print(minlight);
   client.print("' max='");
   client.print(maxlight);
   client.print("' value='");
   client.print(currentlight);
   client.print("' />");
   client.print(maxlight);
   client.print("<label for='Light' ");
   if (currentlight > maxlight || currentlight < minlight) {
     client.print("class='alarm' >");
     client.print("WARNING ");
   } else {
     client.print("class='OK' >");
   }
   client.print("Light level = ");
   client.print(currentlight);
   client.print("</label></div><br/><br/>");

   //this block prints a slider with max min and current value for Humidity level
   client.println("<div>");
   client.print(minhumidity);
   client.print("<input  name='Humidity' type='range' min='");
   client.print(minhumidity);
   client.print("' max='");
   client.print(maxhumidity);
   client.print("' value='");
   client.print(currenthumidity);
```

```
            client.print("' />");
            client.print(maxhumidity);
            client.print("<label for='Humidity' ");
            if (currenthumidity > maxhumidity || currenthumidity < minhumidity) {
              client.print("class='alarm' >");
              client.print("WARNING ");
            } else {
              client.print("class='OK' >");
            }
            client.print("Humidity level = ");
            client.print(currenthumidity);
            client.print("</label></div><br/><br/>");


            //this block prints a slider with max min and current value for soil moisture
            client.println("<div>");
            client.print(minsoilmoisture);
            client.print("<input  name='Humidity' type='range' min='");
            client.print(minsoilmoisture);
            client.print("' max='");
            client.print(maxsoilmoisture);
            client.print("' value='");
            client.print(currentsoilmoisture);
            client.print("' />");
            client.print(maxsoilmoisture);
            client.print("<label for='Humidity' ");
            if (currentsoilmoisture > maxsoilmoisture || currentsoilmoisture < minsoilmoisture) {
              client.print("class='alarm' >");
              client.print("WARNING ");
            } else {
              client.print("class='OK' >");
            }
            client.print("Soil Moisture level = ");
            client.print(currentsoilmoisture);
            client.print("</label></div><br/><br/>");



            client.print("</div>");
            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
}
}
}
}
```

VIDEO

The video shows the working terrarium and the it's subsystems. The video begins by showing the components outside of the terrarium, the humidity, light and soil moisture subsystems all work together. However, when we put the components into the terrarium, we could not get the subsystems to work together, only individually as can be seen in the later section of the video. The temperature sensor is reading values, as can be seen in the section of the serial monitor below, however due to power requirement issues there is no footage of the fan working.

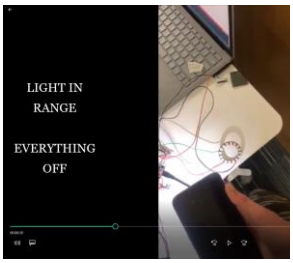
SERIAL MONITOR



STORYBOARD

You can see that the humidity reads 50.40%, which is too low, therefore the atomiser turns on.

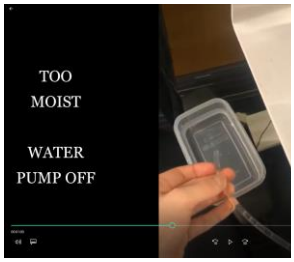
With a torch shining on the light sensor the conditions are too bright (12574.67 lux) the neopixel ring is off and the LCD shutter glass is on.



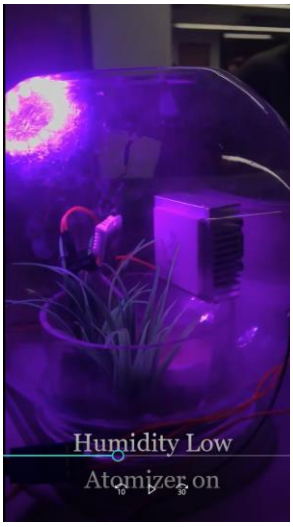
With the torch a small distance away, the light conditions are ideal (4120.23 lux) so the neopixel ring is off and the LCD shutter glass is off.

With the torch far away, the conditions are too dark (1023.47 lux) so the neopixel ring is on and the LCD shutter glass is off.

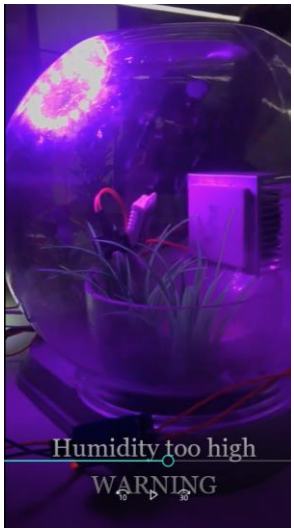
The soil moisture is being read as 4095 when it is in air, this is too dry, therefore the water pump turns on.



The soil moisture is read as 1233 when it is water, this is too moist, so the water pump turns off.



The humidity is read as too low; therefore, the atomizer is on.

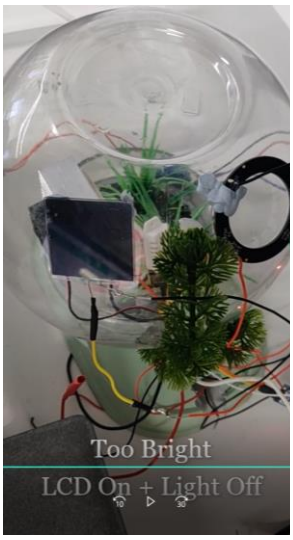


The humidity is read as too high, this creates a warning on the app.

The humidity is ideal, so the atomiser is off.

It is too dark, the neopixel ring is on and the LCD shutter glass is off.

The light is ideal so the LCD shutter glass and neopixel ring are off.

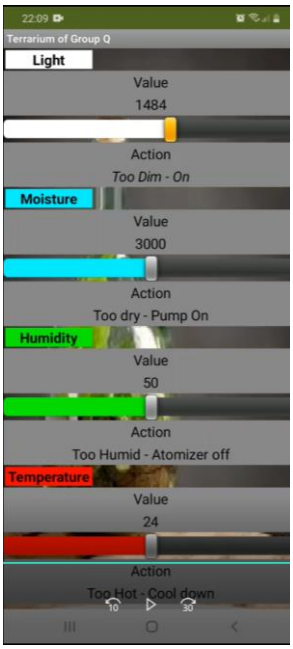


It is too bright, so the LCD shutter glass is on and the neopixel ring is off.

The soil is too dry, so the water pump is on.

The soil is too moist which creates a warning in the app.

The soil moisture is within the ideal range so the water pump is off.



This is the control app which demonstrates the warning when soil moisture is too moist, or the humidity is too high.

LIMITATIONS

Throughout the integration process of all of the individual sub-system errors were experienced within;

TESTING

Assembled terrarium circuitry and code required an energy intensive testing process. One of the major problems we had with the terrarium was meeting the power requirements for all of the components. Multiple tests were conducted of the circuitry configuration alongside modifications within the code, which in turn tested the strength of components. In the first configuration we powered the neopixel ring on the 5V source. This did not work as the power was shared amongst too many components. In the second configuration we wired the neopixel to the 3V source, which also did not work. In the third configuration we wired the DHT to the 6V source, which worked, however the Peltier did not work and thus we assumed the H-bridge lacked power. The fourth configuration we tested was removing the Peltier, this resulted in all the other components working.

COMPONENT INTEGRITY

Issues were experienced with the DHT sensor, this component was extremely fragile, which required multiple re-purchasing. Components reliability consistently affected testing scenarios providing no information regarding sensor readings, this is down to the component integrity which can associated with the investment in cheap components, which inherently affected the terrarium outputs. Another key issue we had was the esp32 which burst due to overloading and had to be changed for a new one. This was particularly difficult as there was little way of knowing when the esp32 was about to be fried in order to prevent it.

WIRING INTEGRITY

Original wiring of individual subassemblies was not adequate for terrarium construction. Re-wiring of components was necessary which led to a poor consistency regarding component reliability and sensor readings.

CHALLENGES

POWER REQUIREMENTS

Throughout the integration process of all of the individual sub-system errors were experienced with power requirements of components due to the transfer from an Arduino board to an ESP32. The power capabilities of the ESP32 are significantly less only providing the fully assembled circuit with a 3.3v power supply. The challenge was to overcome the lack of power by introducing an external power supply of 5v that ensured the components which relied on a higher power source were satisfied, ensuring that the sensor output becoming reliable.

DIFFERING RESOLUTIONS

Due to the difference in resolution between the esp32 and Arduino pins the values for the light and soil moisture needed to be altered accordingly. This increase in resolution encouraged the re-evaluation of our threshold values set for controlling the soil and light readings.

Initially the ideal dry soil reading was 787, this was changed to 3320, and the ideal wet soil reading was 675, this was changed to 2980. The original light values that corresponded to the ideal range of light intensity were 256 < Ideal Light < 921, which were then altered to manage the change in

resolution to 1023 < Ideal Light < 3595. Due to this change in resolution the calculations to calculate lux values also required altering due to the change in resolution.

TERRARIUM CONSTRUCTION

Integrating the circuitry into the shell of the terrarium proved difficult, as the attachment of wires to the breadboard was weak. This created challenges when it came to recording the entire subassembly. Underestimated the size of components and accessibility to terrarium environment, this created issues when it came to effectively test the terrarium conditions.

REASSIGNMENT OF PINS

As individual subassemblies all utilised similar pin numbers on the Arduino board, pin numbers within individuals code had to be adapted. With the transfer to the esp32 assigning pins became increasingly difficult as each pin had very specific capabilities, which at times was hard to work out for each component and its requirements.

DEBUGGING

As for the code, there were problems initially in combining the code, the code could not just be copied from each individual subsystem into a combined code as this introduced syntax errors which required debugging. Debugging in the end was not a huge problem but definitely caused some teething issues initially when we combined the code.

OUTSTANDING PROBLEMS

TEMPERATURE SUBSYSTEM

The power requirements for the temperature subassembly are an outstanding problem. The fan required a large amount of power, which when trying to run with the Peltier, was not possible. The Peltier was able to work for a while but only at the expense of other subsystems within the terrarium.

CONCLUSION

In conclusion, we have managed to build a functioning terrarium. However, the integrity of the components and wiring make this highly temperamental. Additionally, we have built a terrarium to house the components and the arabica coffee plant which functions well; the ideal soil moisture, light and humidity can be maintained, with the temperature monitored, but not adjusted to keep it within the ideal range.

APPENDICES – INDIVIDUAL REPORTS

DM401

Individual Subsystem Report

Ana Hernandez

201612089

Table of Contents

Introduction

DM401's individual project consisted on creating an interactive system that sensors and controls the climate conditions of an enclosed small vessel for it to be suitable for the planting of a chosen coffee.

For the early stage of this project, each participant of the group was assigned to study in depth one of the variables to analyse. These variables were Temperature, Light, Humidity and Soil Moisture. Humidity is the variable we will be investigating in this individual report.

Objective: Creating a system which monitors and regulates the humidity of its enclosed environment to be adequate for planting arabicas pacas coffee.

Subsystem Specification

After brainstorming a few dwarf coffee species, we decided for the pacas variety of arabicas coffee.The plantation of this coffee originally started in El Salvador and then expanded within other countries like Brazil and Costa Rica. As a dwarf coffee species, it matched our requisites for a small terrarium.

To create an appropriate environment for arabicas pacas coffee, climate conditions similar to the ones in El Salvador, Brazil or Costa Rica were explored. Humidity in these areas tend to oscillate between 60% and 80%, so these were set as the desired values for our subsystem to achieve.

Using Arduino Uno a code was programmed to continuously read the humidity of its surroundings and check if this value was within our desired range. If it was, then no actions were to take place. Whereas, if the humidity was measured to be below our previously set "minimum humidity", the system should turn on an atomizer until these levels were reached. Actions for humidity being above the set "maximum humidity" were not set as this belongs to phase 2 of this project.

Schematic Block Diagram

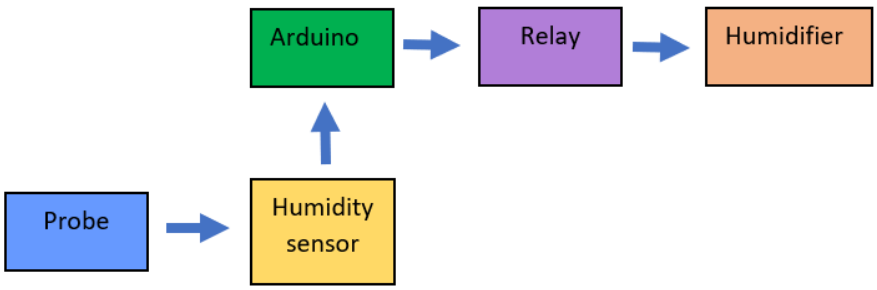


Figure 1- Block Diagram

Components

To choose the appropriate components, the required actions to carry out were determined.

Situation	Required Action
Humidity higher than specifications	Humidify
Humidity within specifications	Keep monitoring
Humidity lower than specifications	Phase 2 (no action)

The chosen components to accomplish these tasks were a DHT11 sensor, to detect the level of humidity and a donut atomizer to provide more humidity when needed.

DHT11

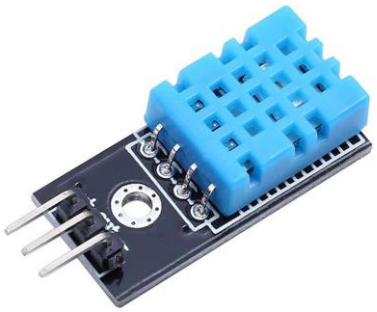


Figure 2- DHT11 sensor

DHT11 is a temperature and humidity sensor compatible with Arduino. These sensors feature a calibrated digital signal output with the temperature and humidity sensors. Its technology ensures a high reliability, good performance, small size and low power required. When choosing sensors, this type was considered fit for purpose for the project especially due to its small size and its quality-price ratio.

- Power input: 5V
- Humidity range: 20-90%RH with ±5ºC HR error
- Sampling rate is once every second (1Hz).

Donut atomizer

Figure 3- Donut humidifier

Similarly, for the donut atomizer, our requirements were small size and good efficiency. Although there are smaller humidifiers, the donut atomizer was recommended to us by some of the project technicians as its price was very low but its performance and size were fit for the project.

- Power Input: DC 5V 500mA
- Capacity:<1L
- Water-shortage Power-off Protection:Yes
- Mist Output: 30ml/h

NPN Transistor

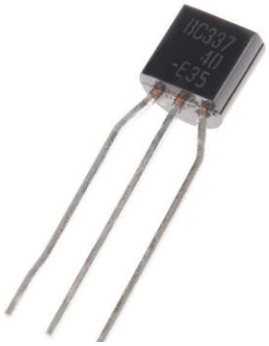


Figure 4- Transistor NPN

For the atomizer to be turned on and off, a transistor is needed. This component will conduct current across the collector-emitter path only when voltage is applied to the base (middle pin), which is when our atomizer is meant to turn on. When no base voltage is present, the switch is off. Basically, this performs as a switch to the atomizer.

Resistor 120Ω

Figure 5- Resistor 120Ω

To limit the maximum base current of the NPN transistor, a resistor is needed. This will connect the base of the transistor to the digital output of the arduino.

Pin Level Schematic

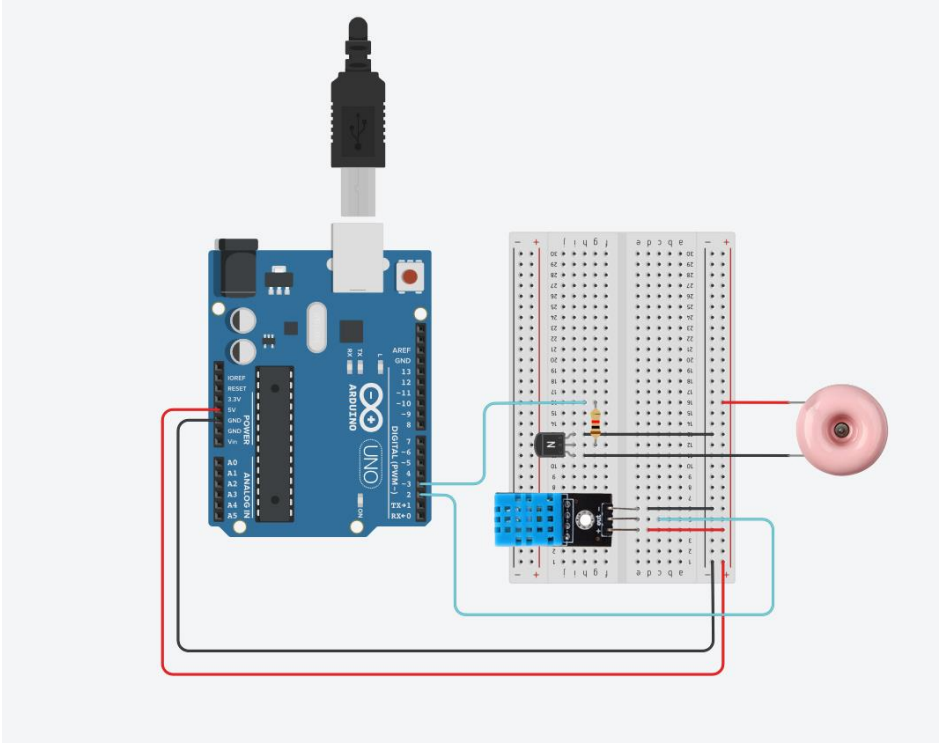


Figure 6-Pin Level Schematic

As shown in the Pin Level Schematic diagram shown in Figure 6, all the components are connected to the Arduino in the following way:

Arduino 5V -	Atomizer + Humidity sensor +
Arduino GND-	Transistor - Humidity sensor -
DIG OUTPUT 2-	Humidity sensor DATA
DIG OUTPUT 3-	Transistor BASE (Through resistor)

Circuit Diagram

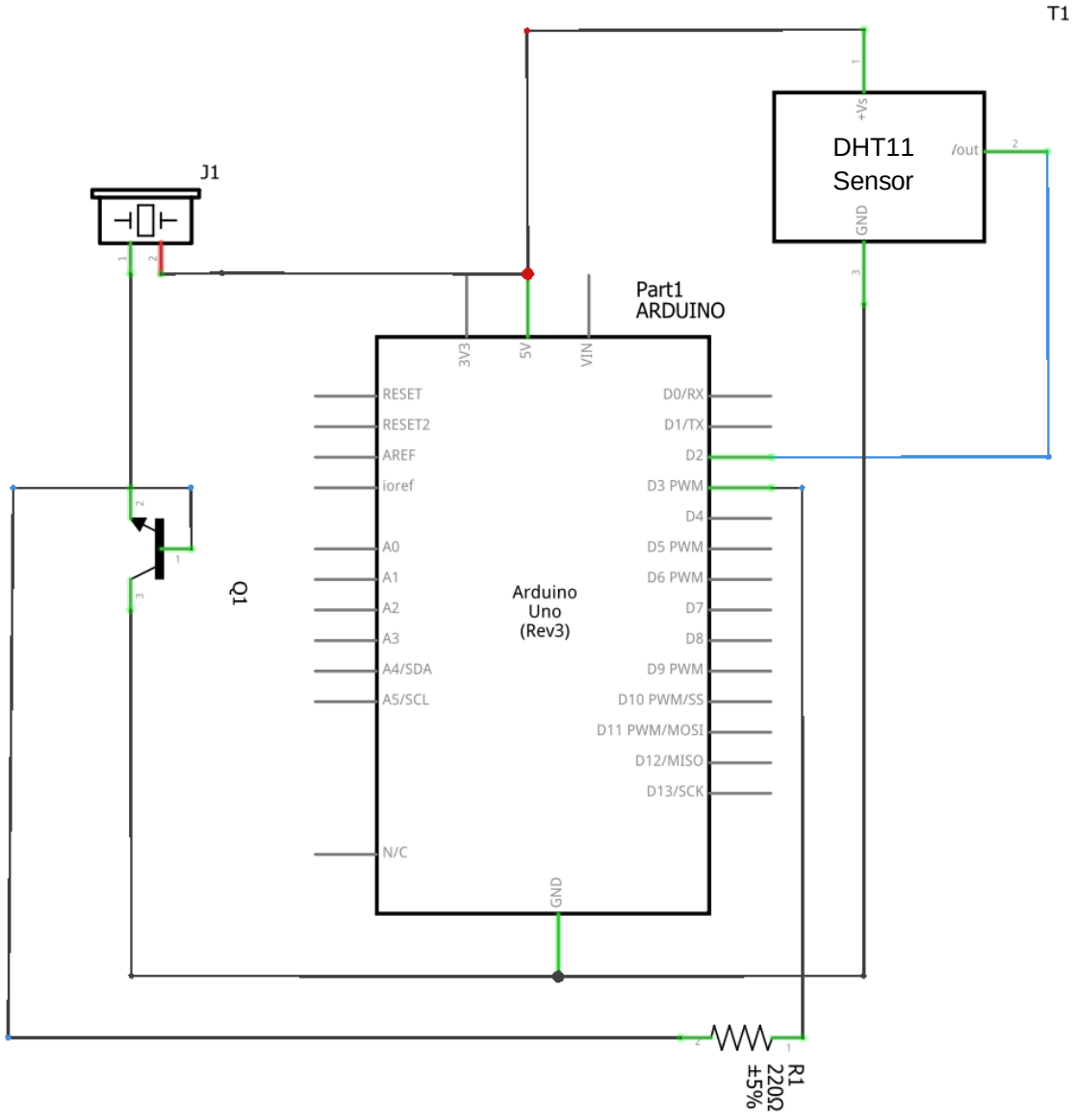


Figure 7- Circuit Diagram

Flowchart

Initially, the flowchart shown in Figure 8 was our planned behaviour for the code. After testing the draft code and realising it wasn't as effective as planned due to delays and processing times of the sensor, the flowchart was re-designed (Figure 9) and so was the code. (Further explanation on the failures of the code are discussed in the testing section).

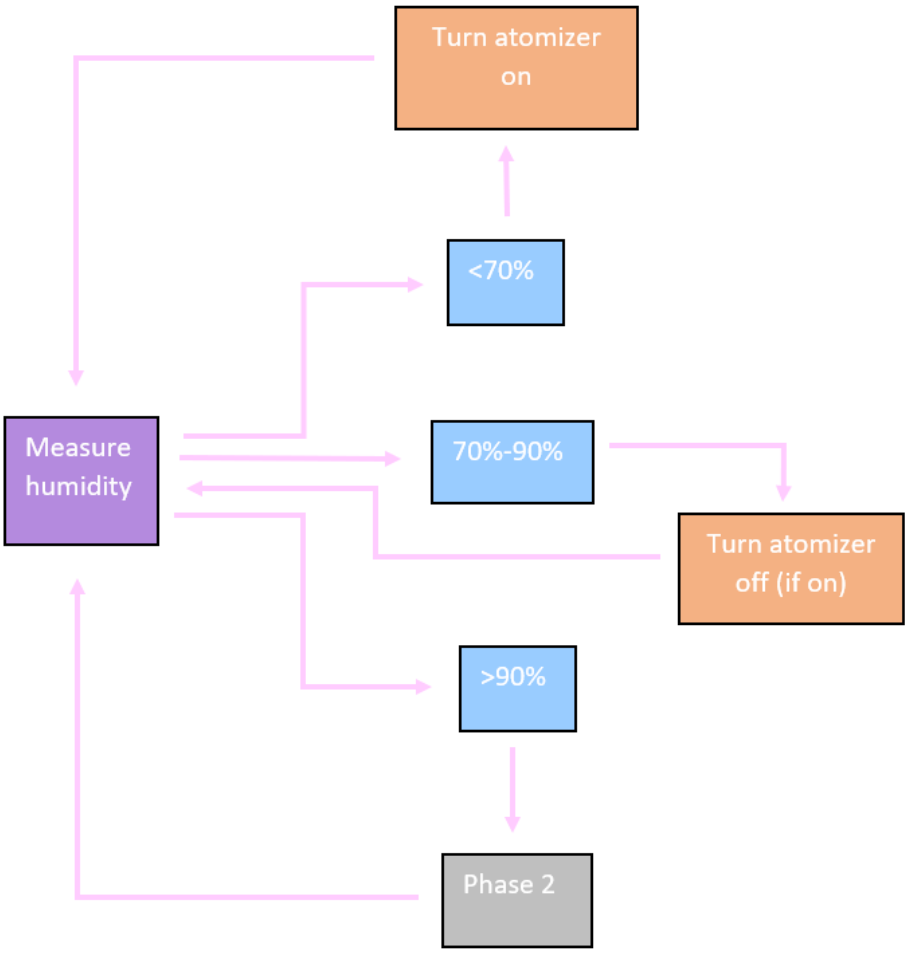


Figure 8- Flowchart 1

This flowchart describes a system that turns on the atomizer for a period of time that will last as long as the sensor reads any value below 70. Processing times and delays weren't taking into consideration when this flowchart was designed.

Redesigned flowchart

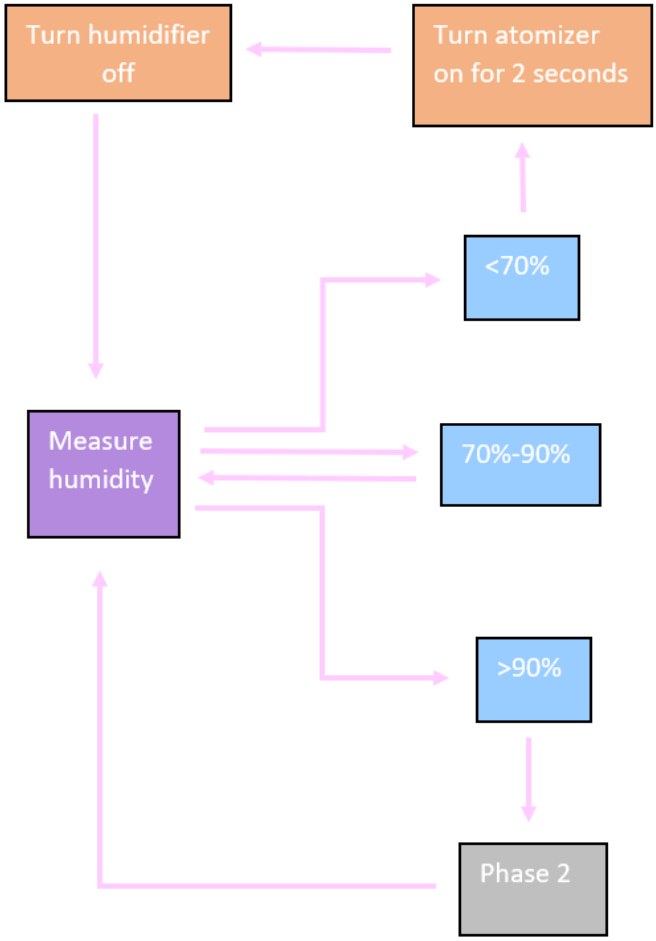


Figure 9- Flowchart 2

This flowchart, unlike the previous one, would only turn the atomizer on (if required) for 2 seconds and then would measure the humidity again, only depending on this second read it will atomize for another two seconds or not. This way we avoid humidifying over the maximum humidity desired. When humidity is below the minimum by a relatively high amount, the humidifier will be turned on for two seconds for a few loops in a row. As this wasn't tested in the terrarium that will be used for the full project, numerical data and calculations of the humidity variation of the system provided by the donut atomizer per second were not taken, as these will depend on the volume and shape of the vessel we use.

Arduino Code

To start programming our code it was recommended to us to commence using tinkercad. Due to the fact that tinkercad does not have humidity sensors, atomizers or fans as components, the humidity sensor was replaced with a temperature sensor and the atomizer with a buzzer. At this stage it wasn't of big importance, as the task was to conclude how to create the adequate code and circuit connections for our components, which worked in a similar way as their replacements. Once the basic code was obtained from tinkercad, this was added into arduino. To complete this code, some arrangements and upgrades had to be made:

1) A DHT sensor library: to provide extra functionalities.
2) **Serial**.print: added to display the humidities that were collected.
3) float: Data type for floating-point numbers, a number that has a decimal point.
4) delay: delaying the program to improve simulation performance and to determine atomizers period of time on.

Final code

```
#include "DHT.h" //Include the DHT Sensor library
#define DHTPIN 2 // Define what pin the sensors digital output is connected to
#define DHTTYPE DHT11   // Define type of DHT sensor
DHT dht(DHTPIN, DHTTYPE);
const int MIN_HUMIDITY = 70; //Set a minimum humidity constant
const int MAX_HUMIDITY = 90; //Set a maximum humidity constant

void setup()
{

  Serial.begin(9600);
  Serial.println("DHT11 Humidity sensor\n\n");

  dht.begin();

  pinMode(3,OUTPUT);//Set atomizer as output pin 3
  digitalWrite(3,LOW);//Start the system with the atomizer pin turned off
}

void loop()
{
  float h = dht.readHumidity(); //Read humidity
  //Set the display of humidity readings
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.println(" %\t");

  if (h < MIN_HUMIDITY) { //Set condition for humidity being below the minimum required
    digitalWrite(3, HIGH);//If humidity is below minimum turn atomizer pin on
    delay(2000);//Leave atomizer on for 2 seconds
  } else {
    digitalWrite(3, LOW);//If atomizer is not below minimum keep the atomizer off
  }
  digitalWrite (3,LOW);//Turn atomizer off (if was it was on)

  delay(2000); // Delay a little bit to improve simulation performance
```

Figure 10- Final code

Testing the subsystem

Before testing our code with the physical components, our main components were tested individually to make sure these were working in good conditions.

Testing the sensor

```
#include "DHT.h"
#define DHTPIN 2     // what pin we're connected to
#define DHTTYPE DHT11   // DHT 11

DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600);
  Serial.println("DHT11 test!");

  dht.begin();
}

void loop() {

  float h = dht.readHumidity();
  Serial.print("Humidity: ");
  Serial.print(h);
  Serial.print(" %");

}
```

Figure 12- Sensor test code

```
DHT11 Humidity sensor


Humidity: 41.00 %
Humidity: 41.00 %
Humidity: 41.00 %
Humidity: 42.00 %
Humidity: 43.00 %
Humidity: 43.00 %
Humidity: 43.00 %
Humidity: 44.00 %
Humidity: 44.00 %
```

Figure 11- Results sensor test

Using the code shown in Figure 12, and connecting the breadboard as shown in Figure 13, the sensor read the humidity shown in Figure 11. This meant that our DHT11 sensor was effectively reading humidity, and therefore fit to test our full code.

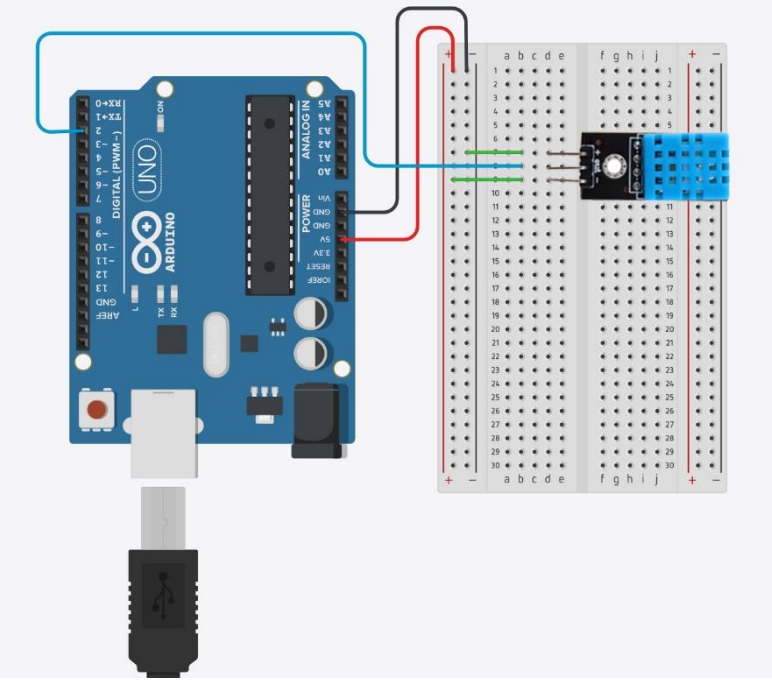


Figure 13- TinkerCAD sensor test

Testing the atomizer

```
#define ATOMIZERPIN 3

void setup() {
  // put your setup code here, to run once:
    pinMode (3, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:


  digitalWrite(3, HIGH);
  delay (1000);
  digitalWrite (3, LOW);

}
```

Figure 15- Atomizer test code

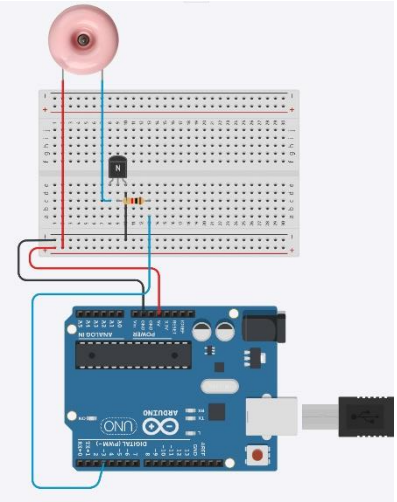


Figure 14- TinkerCAD Atomizer test

Using the code shown in Figure 15 and connecting the atomizer as shown in Figure 14, the atomizer turned on and off successfully. Therefore, both of our main components were fit for purpose.

Testing the final code

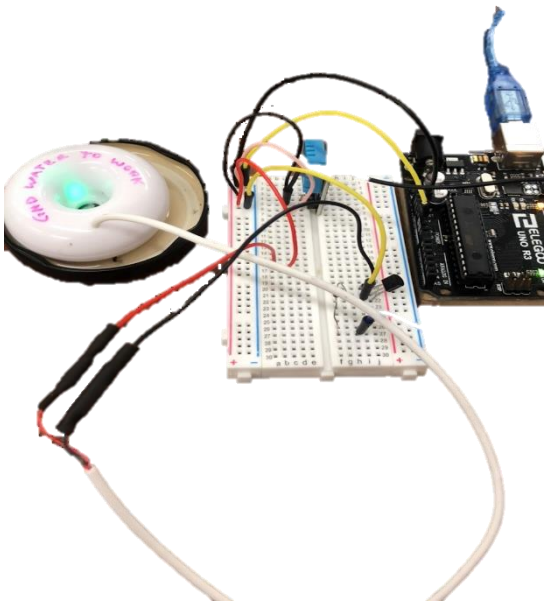


Figure 16- Final test

After testing the initial drafts of what were thought to be the final code, failures in the system kept arising. At first, the idea was to turn on the atomizer for as long as needed to reach the desired humidity. The coding for this was very complex, as a lot of factors conditioned this action. First, the humidifier would produce water vapour, but there would be a delay from that action until this mist would reach near the sensor to be detected. After that, the sensor itself takes an approximate 250ms to read humidity. Moreover, the readings may be between one and two seconds old, due to the sampling speed of sensor (1Hz). Therefore, due to the delay, by the time the system read the desired humidity, the real humidity of the system was above the maximum required.

After changing and testing this code several times, it was concluded that the best approach was to change the behaviour of the code. A redesign of the flowchart was carried out (as mentioned above) and the code was altered to follow its behaviour (shown in Code section).

Limitations

Throughout the project, many problems were encountered. Due to the sampling rate of the sensor and the assumed small capacity of our vessel for the terrarium, many tests and arrangements had to be done. Another perspective of the subsystem's behaviour had to be contemplated. Also, due to the uncertainty of our terrarium volume, it was difficult to make the code perfect for phase 2 of the project, but this can easily be adjusted when these values are known.

Conclusions

In conclusion, the objective for this individual project was successfully achieved. The objective was to create a system which monitors and regulates the humidity of its enclosed environment to be adequate for planting arabicas pacas coffee.

I have found this project to be very complete and enriching. With the application of some electronics, learning the basics of coding and testing, it has seemed to put into practice different topics we have learnt throughout the last years.

In terms of the coding, for the second part of this project the code will have to involve the 3 different variables codes in the same loop. For all these codes to work simultaneously, some arranges will have to be made to the individual codes, especially because they are all dependant of each other. Also, "delays" are effective in the humidity subsystem code, but when all subsystems join to one, this will mean stopping the loop for every variable, not just one. This will cause problems for all 4 variables to work together, as some subsystem require a speedier processing time than others. To replace the delays in each individual subsystem, the code will use the millis () function, a command that returns the number of milliseconds since the board started running its current sketch, to turn things on and off without delaying the rest of the process. The adjustment of the code will look something like this:

```
unsigned long currentMillis = millis();
if (currentMillis - previousMillis >= interval) {
  // save the last time you turned on atomizer
  previousMillis = currentMillis;

  // if the atomizer is off turn it on and vice-versa:
  if (atomizerState == LOW) {
    atomizerState = HIGH;
  } else {
    atomizerState = LOW;
```

Figure 17- Upgraded future code

In addition to this, when we are aware of the volume of our terrarium, measurements on how the humidity levels vary with the atomizer being on for one second will be made. This way, if the humidity is below by a relatively high amount, the atomizer can be programmed to humidify for a longer amount of time, to reach our desired range faster.

Table of Figures

DESIGNING A SUBSYSTEM TO CONTROL THE SOIL MOISTURE OF A COFEE PLANT

DM401 INDIVIDUAL ASSIGNMENT

ELEANOR ROBERTS
201613214 | 4TH YEAR PDE

CONTENTS

INTRODUCTION

SUBSYSTEM SPECIFICATION

Coffea Arabica is an evergreen [1] shrub or small tree [2] which is native to humid rainforests and blooms only when it receives enough water [3]. Figure 1 shows the ideal soil moisture for a variety of trees and shrubs and, despite not listing coffea arabica specifically, we can assume coffea arabica requires a similar soil moisture to other evergreen shrubs or small trees. For example, rhododendrons are evergreen shrubs or small trees which grow best in areas of high rainfall [4], due to their similarities with the coffea arabica plant they can be used as a guideline for ideal soil moisture. Figure 1 shows that rhododendrons require 41-60% soil moisture, therefore, any soil moisture value within this range will be acceptable.

The functional requirements of the subsystem are as follows;

- The subsystem must be able to measure the soil moisture of the coffee plant
- The subsystem must be able to compare the actual soil moisture with the required soil moisture
- The subsystem must be able to automatically water the plant until the required soil moisture is achieved

Figure 1: The ideal soil moisture for small trees and shrubs [5].

SCHEMATIC BLOCK DIAGRAM

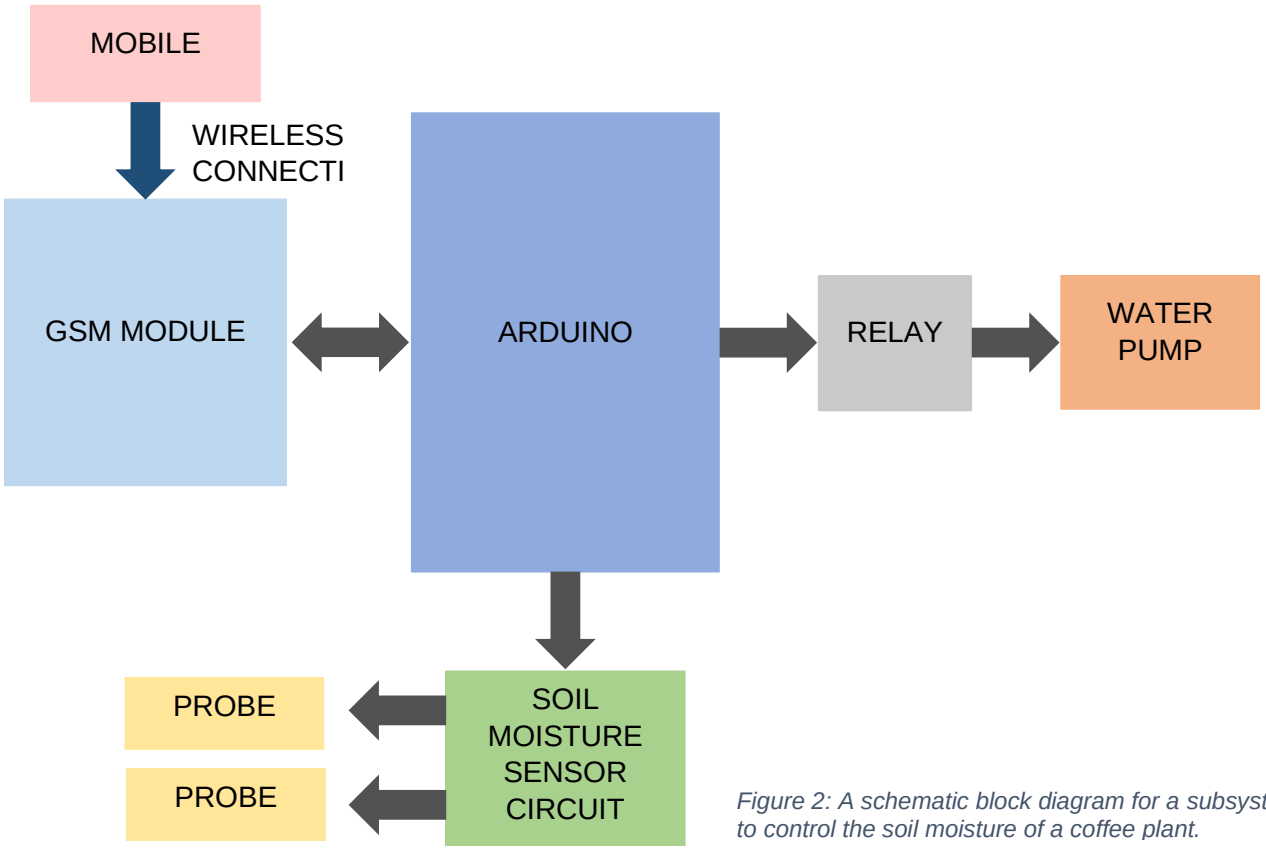
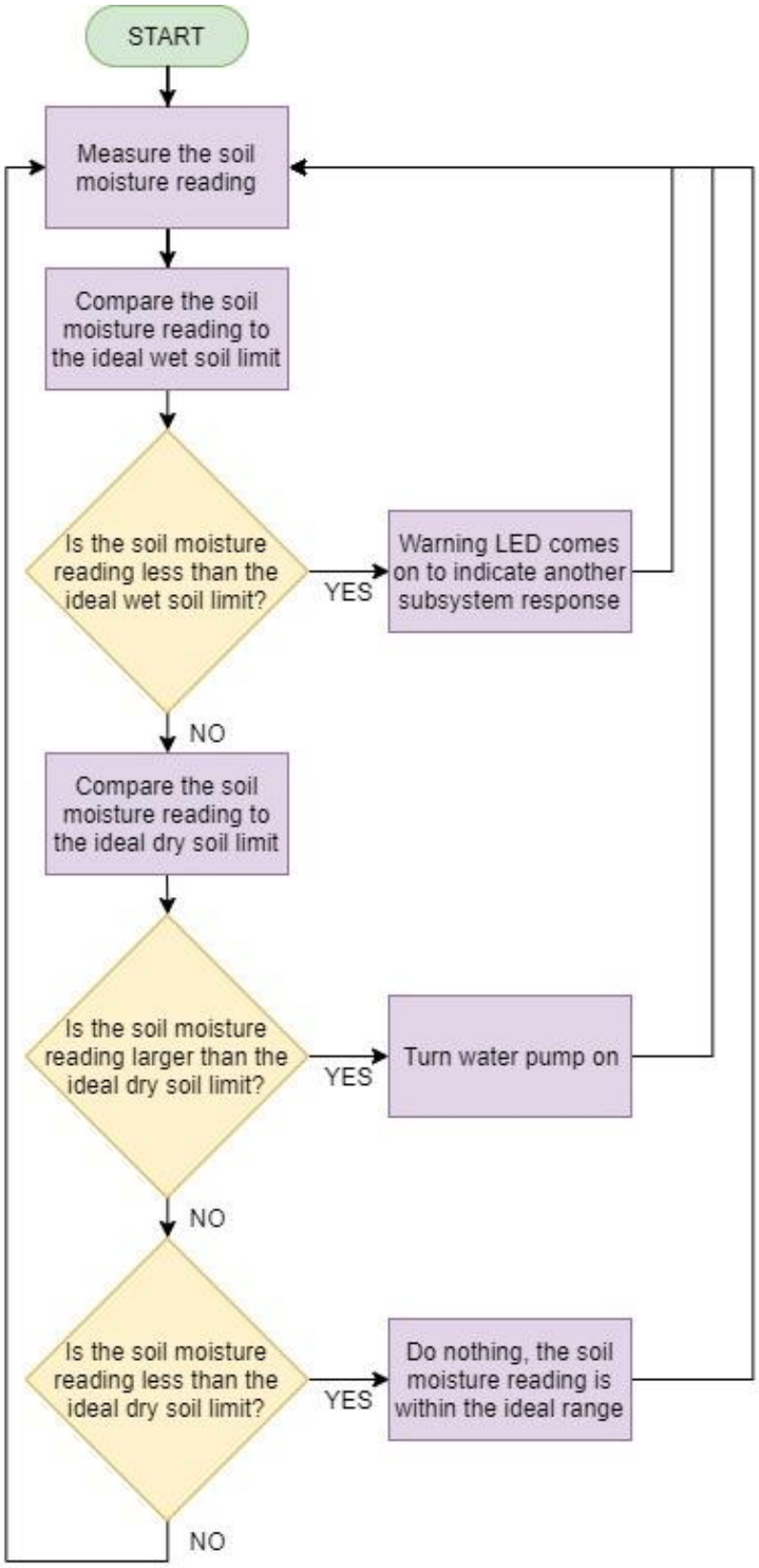


Figure 2: A schematic block diagram for a subsystem to control the soil moisture of a coffee plant.

SUBSYSTEM FLOWCHART

The flowchart below describes the basic logic behind the subsystem inputs and the decisions it will make to ensure the correct output response.



COMPONENT CHOICE

SOIL MOISTURE SENSOR

The requirements for the soil moisture sensor were that its operating voltage should be as low as possible, but definitely below 5V, and that it should come with a soil moisture detection module for ease of installation into the subsystem. It was also important that the sensor should be under £5 and have an estimated delivery time of under 10 days in order to be able to fit the project timeline.

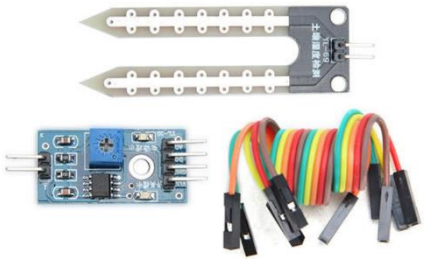


Figure 3: YL-69 Soil moisture sensor and YL-38 Soil moisture detection module from BangGood.co.uk [6].

From the above requirements I decided on the YL-69 soil moisture sensor with an attached YL-38 soil moisture detection module, as seen in figure 3. The sensor has adjustable sensitivity through the potentiometer in the detection module it includes. It has an operating voltage of 3.3V – 5V which is within the ideal range. It also has a power indicator and digital switching output indicator which may be useful when it comes to testing the subsystem. Aside from the technical features of the sensor and detection module, the kit was only £3.96 and had an estimated delivery time of 5-10 days, which fits the budget and timescale of the project.

WATER PUMP

The water pump, like the soil moisture sensor, also needed to run on as low a voltage as possible; because of the safety concerns with using mains voltage, components would be run off the power from the Arduino and any additional batteries. It would also be beneficial if the water pump was submersible to save on the amount of tubing required and the space that requires. As with the soil moisture sensor, the water pump had to be under £5 and delivered within 10 days to fit with the project timeline.

Figure 4: Mini submersible water pump from Ebay [7].

The water pump, as seen in figure 4, was chosen as it was submersible and only required a voltage of 3V – 6V. The water pump was also very small, approximately 50 x 30 x 25 mm, which will reduce the overall space taken up by the subsystem within the terrarium. Additionally, the pump was only £3.45 and had an estimated delivery time of a few days, thus being within budget and suiting the project timescale requirements. It is also notable that the water pump has a flow rate of 80 – 120 L/H, therefore, the pump will work quickly enough so that the subsystem can have a fast response time, increasing the amount of time the coffee plant will have with ideal soil moisture.

RELAY

A relay is needed to act as a switch to turn the water pump on and off according to the soil moisture sensor readings. The relay only requires one channel and should operate on as low a voltage as possible, ideally under 5V, in order to reduce the amount of voltage required in the circuit. Much like the other components, the relay should be below £5 and have an estimated delivery time of under 10 days, to allow enough time to put together the subsystem and test it before the project deadline.

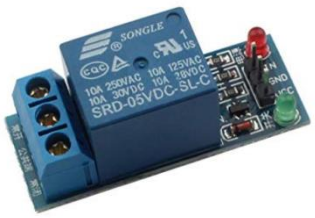
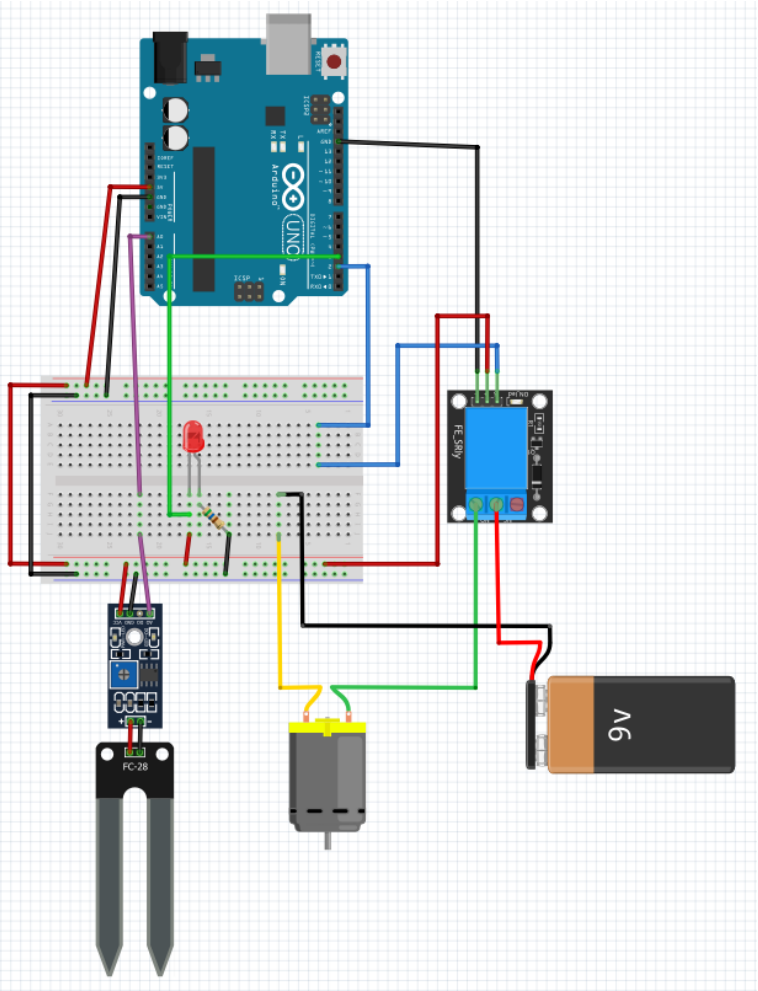


Figure 5: One channel relay from Amazon [8].

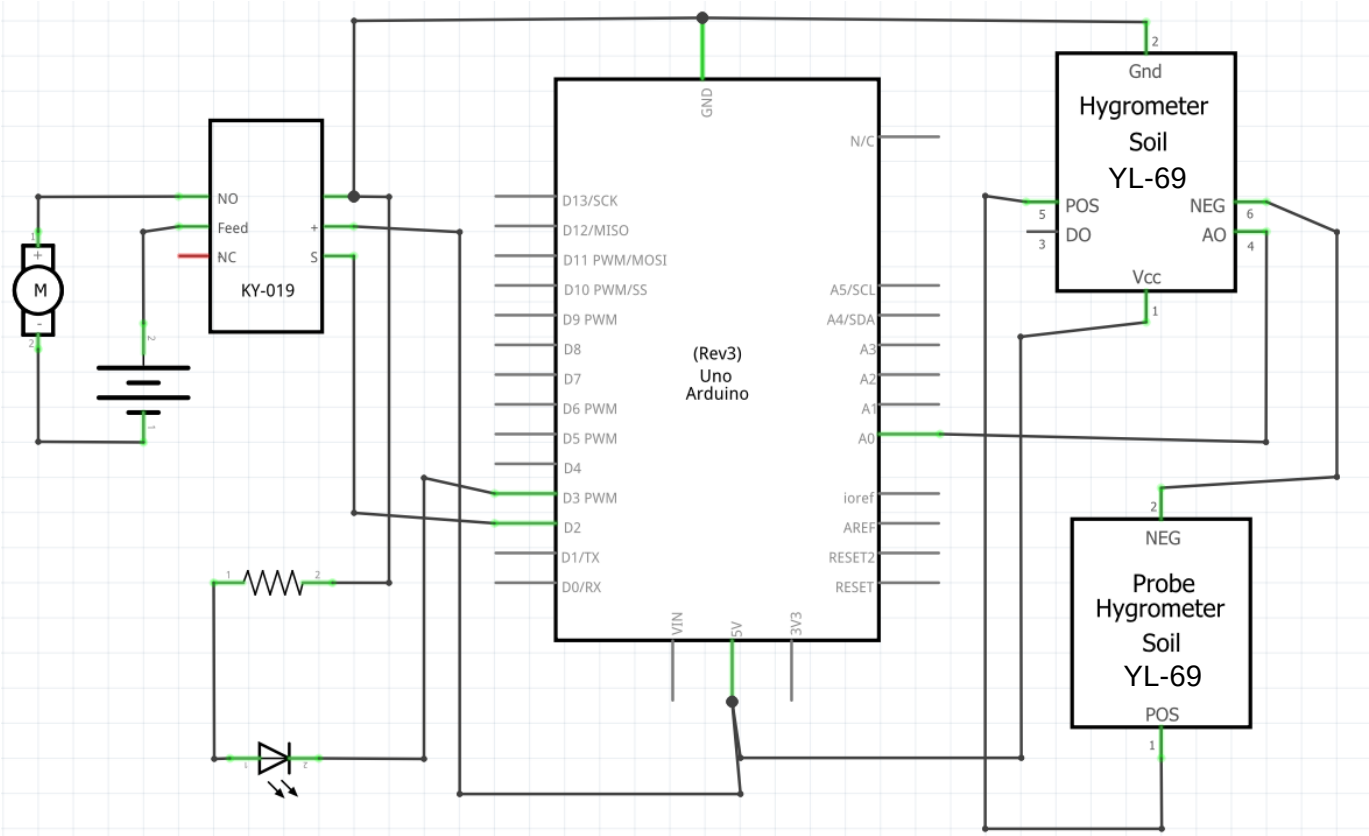
The lowest voltage one channel relay I could find was a 5V relay, as seen in figure 5. It was chosen as it had one channel, any more channels would've been surplus to requirement, and operated on only 5V. It was within budget at £3.99 for 5 and had next day delivery. Moreover, the relay was chosen as it had a relay module attached, making it easier to install in the subsystem and space saving.

PIN LEVEL SCHEMATIC

The pin level schematic and circuit diagram show how the circuit is wired up and which pins on the Arduino board the soil moisture sensor, water pump and relay module are connected to. Both the pin level schematic and circuit diagram were created using Fritzing and use a DC motor to model the water pump. This will be referenced when I come to physically build and code the subsystem.



CIRCUIT DIAGRAM



CODE

The code shown below operates an automatic plant watering subsystem, maintaining a constant soil moisture of 41% - 60%. See comments within the code which describe each command.

Automatic_Plant_Watering_System | Arduino 1.8.9 (Windows Store 1.8.21.0)

File Edit Sketch Tools Help

Automatic_Plant_Watering_System

```
const int RelayPin = 2; // Sets digital pin 2 to the relay module
const int SensorPin = A0; // Sets analog pin 0 to the soil moisture sensor
const int WarningLEDPin = 3; // Sets digitial pin 3 to the warning LED
const int TransmissionSpeed = 9600; // Sets the transmission speed to 9600 bits per second
const int IdealDrySoil = 787; // Sets the upper threshold for ideal soil moisture to 787
const int IdealWetSoil = 675; // Sets the lower thershold for ideal soil moisture to 675

/**
 * Code run once at beginning of execution for intitialisation of I/O
 */
 void setup () {
  Serial.begin(TransmissionSpeed); // Sets the transmission speed of communication channels

  // Inititalising pin I/O
  pinMode (SensorPin, INPUT); // Sets the soil moisture sensor as an input
  pinMode (RelayPin, OUTPUT); // Sets the relay module, which controls the water pump, as an output

  digitalWrite (RelayPin, HIGH); // Sets the relay module initial state to 5V, i.e. the pump is off
  delay(500);
 }

/**
 * Continuously running code which monitors the moisture readings
 */
 void loop() {
  float MoistureLevel = analogRead (SensorPin); // Reading value from soil moisture sensor

  //Prints the value read by sensor i.e. moisture level, in the serial monitor window
  Serial.print ("MOISTURE LEVEL: ");
  Serial.print (MoistureLevel);
  Serial.println(); // Starts a new line in the serial monitor window

  /**
   * Code which reacts to the moisture readings and controls the water pump and warning LED response
   */

  if (MoistureLevel < IdealWetSoil) { // MoistureLevel drops below 675, pump is off, moisture level is too wet
   digitalWrite (RelayPin, HIGH); // 5V power goes to the relay module, thus breaking the circuit
   } else if (MoistureLevel > IdealDrySoil) { // MoistureLevel goes above 787, pump on, moisture level is too dry
     digitalWrite (RelayPin, LOW); // 0V power goes to the relay module, thus connecting the circuit
   } else if (MoistureLevel < IdealDrySoil) { // MoistureLevel drops below 787, pump off, moisture level is within ideal range
     digitalWrite(RelayPin, HIGH); // 5V power goes to the relay module, thus breaking the circuit
   }

   /**
    * Code which turns the warning LED on when the MoistureLevel drops below 675
    */

   if (MoistureLevel < IdealWetSoil) {
     digitalWrite(WarningLEDPin, HIGH); // Warning LED on indicating another subsystem response i.e. an increase in temperature
   } else {
   digitalWrite(WarningLEDPin, LOW); // Warning LED off indicating no other subsystem response is required
   }

   delay(1000); // Wait 1 second before next loop
 }
```

Done Saving.

4

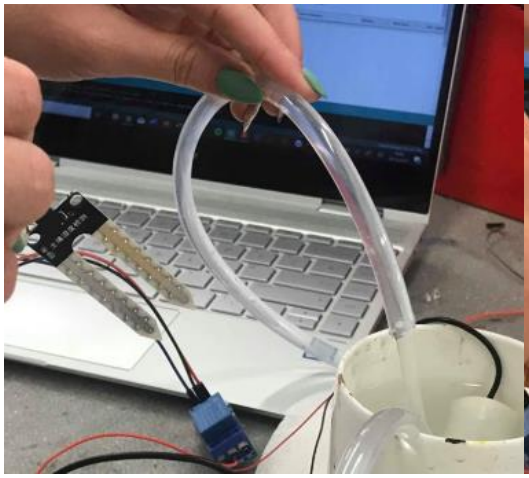
TESTING SUBSYSTEM BEHAVIOUR

In order to obtain the values required to program the subsystem to maintain an ideal soil moisture range of between 41% - 60% soil moisture the subsystem was tested to its extremes. Firstly, the subsystem was tested with the sensor completely dry, indicative of 0% soil moisture. The values produced by the soil moisture sensor in the serial monitor ranged between 1020 – 1024, therefore a midpoint value of 1022 was taken to show 0% soil moisture. Next, the subsystem was tested with the sensor completely submerged in water, indicative of 100% soil moisture. The values produced by the soil moisture sensor in the serial monitor ranged between 432 – 436, therefore a midpoint value of 434 was taken to show 100% soil moisture. These values were then used to calculate the soil moisture sensor values at 41% soil moisture and 60% soil moisture. The upper threshold of the ideal range, 41% soil moisture, was calculated to be 787, and the lower threshold of the ideal range, 60% soil moisture, was calculated to be 675. These values are then used to program the subsystem to maintain a 41% - 60% soil moisture level. The upper threshold is also used to program the warning LED which will turn on to indicate that the soil is too wet, and another subsystem should respond in order to dry the soil out until it is within the ideal range again.

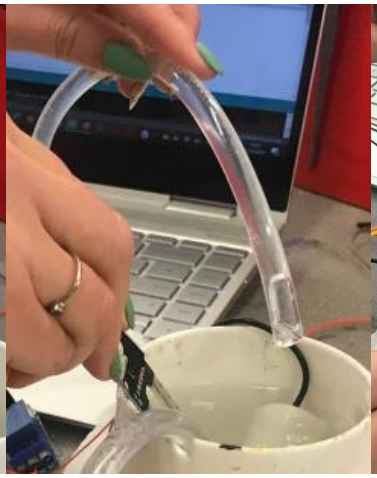
Soil moisture (%)	Test	Soil moisture reading
0	1	1021
	2	1024
	3	1020
41		787
60		675
100	1	433
	2	436
	3	432

Once the ideal range had been decided, the subsystem needed to be tested to ensure the water pump and warning LED were turning on and off correctly. To do this, the subsystem was set up so that the soil moisture sensor was inserted in a sponge, allowing a more gradual increase in moisture, and observed working over a range of moisture readings. The subsystem worked as required; for any soil moisture reading below the lower ideal soil moisture (when the soil was too dry) the pump turned on, when the soil moisture was within the ideal soil moisture range there was no subsystem response and, when the soil moisture readings were above the upper soil moisture limit (when the soil was too wet) the warning LED turned on.

0% Soil moisture – pump on



100% Soil moisture – pump off, warning LED on



41% - 60% Soil moisture – pump off

To test the reaction time of the subsystem I timed to see how long it would take the moisture level reading to return to the 0% soil moisture reading of 434, after the sensor was removed from a bowl of water i.e. 100% soil moisture reading of 1022, and thus how long it took the water pump to turn on to correct the change in soil moisture readings. The moisture level reading reduced almost instantly but took approximately 2 seconds to completely settle at the 0% soil moisture reading of 434. The water pump also took this long to turn on however, there was a further 1 second delay to the water being delivered to the plant, this was due to the length of the water pipe used meaning that time needed to be accounted for the water travelling down the 1m long tube to the plant. Another factor in the reaction time of the water pump may have been due to the fact that when the test was done the tube was tangled slightly, upon detangling the tube the water appeared to be delivered slightly quicker, thus improving the reaction time.

The speed of the water pump was said to be 80 – 120 L/H by the manufacturer, I tested this by measuring how long it took to pump 1 litre of water. 1 litre of water was pumped in 33.2 seconds, extrapolating that, it would take approximately 44 minutes 15 seconds to pump 80 litres of water and 66 minutes 25 seconds to pup 120 litres of water. Therefore, the speed range as stated by the manufacturers is correct. This experiment was completed using 6V, if the power supply was increased, the speed of the water pump would increase, however an increased speed is unnecessary for the automatic plant watering subsystem.

Test	Time taken to pump 1 litre of water (secs)
1	35.0
2	31.4
3	33.7

SUBSYSTEM LIMITATIONS

The subsystem has certain limitations to its functionality, such as; its speed of response, power requirements, the brightness of the warning LED and the probe length. The subsystem is limited by the speed of its response as it takes approximately 1 second for the subsystem to output the soil moisture readings and act upon them accordingly. In the context of the automatic plant watering subsystem this delay is not significant however, for other subsystems which require immediate responses this may be an issue.

Another system limitation is the power requirements of the various components. The Arduino board can only provide 3.3V or 5V of power, but the sensor requires 3.3V – 5V and the water pump also requires 3V – 6V, the warning LED will also have a small power requirement. Since the power required is more than the Arduino can provide, external batteries are required to power the water pump. When it comes to integrating the terrarium subsystems the power required will increase dramatically, thus increasing the number of external batteries required.

Additionally, despite only using a small resistor, the brightness of the warning LED is fairly low, reducing its effectiveness to alert the user to a soil moisture reading which is too low. This will be altered when it comes to integrating the terrarium subsystems as the warning LED will be replaced with a colour changing soil moisture reading. For example, the soil moisture reading will appear green in the app when it is within the ideal range, or too high, and will appear red in the app when the soil moisture reading is too low i.e. too wet.

The system is also limited by the length of the sensor probes. The sensor probes only extend approximately 5cm into the soil, thus only maintaining the ideal soil moisture in the top 5cm of soil and not the full depth of soil. This could be improved by increasing the length of the soil moisture sensor probe so that they extend throughout the full depth of the soil. The drainage of the water throughout the soil will also impact on the subsystem, however if the probes extend throughout the depth of the soil this would be accounted for as the soil moisture sensor would take an average reading.

CONCLUSION

In conclusion, the subsystem is effective in maintaining the soil moisture within the ideal range for a coffea arabica plant and producing a warning LED response when the soil moisture is too wet. The subsystem has a fast response time, therefore minimising the amount of time that the plants soil moisture might be outwith the ideal range. Despite this, the subsystem does not provide a method for drying out the soil if the soil is too wet aside from activating a warning LED. This could be improved upon when the whole terrarium system is integrated, instead of activating a warning LED, the subsystem could activate another subsystem response, such as increasing the temperature. Another improvement that could be made would be to install a drip irrigation system, rather than the water pipe, as this would provide more even watering of the soil and help prevent over watering. Further improvements could be made using a more accurate sensor and a water pump, with a quicker response time.

REFERENCES

[1] https://scanews.coffee/2014/03/08/science-basic-plant-biology-keeping-the-coffee-plant-happy/

Accessed: 10/10/2019

[2] https://en.wikipedia.org/wiki/Coffea

Accessed: 10/10/2019

[3] https://balconygardenweb.com/how-to-grow-a-coffee-plant/

Accessed: 10/10/2019

[4] https://www.rhs.org.uk/plants/popular/rhododendron/growing-guide

Accessed: 10/10/2019

[5] https://www.acurite.com/media//manuals/01410M-instructions.pdf

Accessed: 10/10/2019

[6] https://www.banggood.com/Soil-Hygrometer-Humidity-Detection-Module-Moisture-Sensor-For-Arduino-p-79227.html?cur_warehouse=UK

Accessed: 12/10/2019

[7] https://www.ebay.co.uk/itm/122508463023

Accessed: 12/10/2019

[8] https://www.amazon.co.uk/dp/B07Q215DXW/ref=pe_3187911_189395841_TE_dp_1

Accessed: 18/10/2019

JAMES STUART BURGESS
201615167

DM401

LIGHT SUBSYSTEM DESIGN

TO CONTROL CONDITIONS WITHIN A TERRARIUM FOR A DWARF COFFEE PLANT

INTRODUCTION

The proposed project requires a subsystem design for a smart terrarium, designed to house and care for a dwarf coffee plant, with a focus on light intensity. Light intensity is related with the amount of light energy provided to the plant, which can vary depending on the plant investigated, by colour and the actual strength of the light [1].

The challenge is to ensure that the coffee plants subsystem simulates an environment for optimal growth. Within this project, Coffea Arabic plant will be investigated, and the light intensity required to ensure optimal growth.

SUBSYSTEM REQUIREMENTS

A plant's most natural habitat provides the intensity of light needed for optimal growth. As a result, plants require varying levels of light intensity, and this can be classified accordingly to their light needs forming three categories; low, medium and high [2].

The Arabica species of coffee plant prefers to be in light to semi-shady locations with dappled sunlight, or in weaker latitudes full sunlight. They are marginal plants, so direct, harsh sunlight should be avoided [3]. Due to this, the Coffea Arabica plant is classified as a medium light level plant which is described as requiring 'filtered daylight' [4]. Medium light intensity plants prefer light intensity values between the ranges 2500 – 10,000 lux. Thus, the optimal intensity of light for the Coffea Arabica plant to ensure that the process of photosynthesis is maximised, sits between this range.

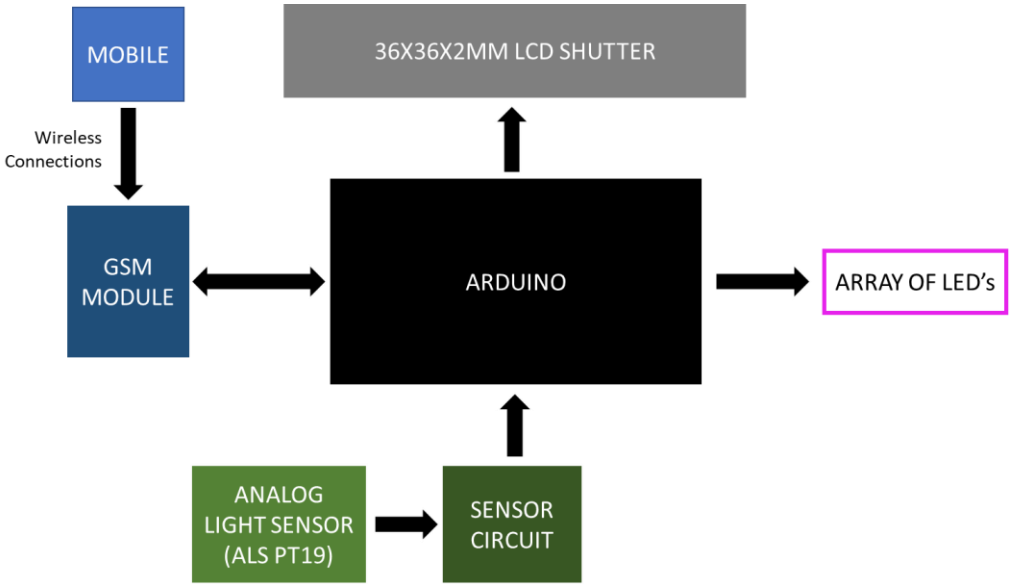
Another variable to consider with light intensity is colour, colours in light have differing wavelengths and those wavelengths can provide plants with varying energy levels [5]. Blue light wavelengths encourage vegetative growth through strong root growth and intense photosynthesis. Red Light wavelengths encourage stem growth, flowering and chlorophyll. Green light is known to be the least effective for plant growth as most of the light is reflected from the plant, which gives the plant itself its green colour to the human eye [6]. To optimise the growth of the Coffea Arabica plant a combination of red and blue wavelengths should be produced, it is known that the highest energy light exists at the purple end of the light spectrum at this combination of RGB (R255, G0, B255)[5].

FUNDAMENTAL REQUIREMENTS

- Subsystem should measure light intensity
- Subsystem should compare lighting conditions to optimal
- Subsystem should provide appropriate lighting and shade, until optimal conditions are met
- Optimal Lighting maintained (2500 < Light < 10000 Lux)
- Optimal colour light maintained (R 255, G 0, B 255)

SCHEMATIC BLOCK DIAGRAM

The schematic block diagram was produced to identify the components within the subsystem and how they interact within the system. It highlights what components are inputs and outputs in relation to the Arduino. The diagram states that data received from the analogue light sensor (ALS PT19) and sensor circuit are input into the Arduino. To then produce two output variables which are controlled by the Arduino the LCD shutter and the NeoPixel array of LED's. The diagram also shows how the GSM module and Arduino communicate between each other and commands and information can be shared.



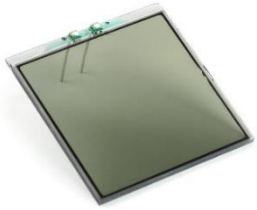
COMPONENTS

ANALOG LIGHT SENSOR – ALS PT19

The ALS-PT19 is an ambient light sensor that provides a good output linearity across a wide illumination range [7] which will allow an appropriate estimation of Illuminance (lux) values. Thus, allowing the subsystem to satisfy the optimal light intensity range accurately. This component has a dynamic response from 0 – 10,000 lux [8]. As the max lux value that can be read on this component is at the max range of the optimal light intensity, the optimal range will have to be modified to suit the prototype. More expensive light sensors could have been utilised which house the optimal light intensity range.

An LDR was also considered however the response to light intensity with this component does not produce a linear response to varying light intensities [9], this was deemed too complex to accurately represent the optimal ranges of light intensity. The analogue light sensor was chosen over this as it provided an accurate datasheet featuring graphs that could be referenced to find exact lux values.

LCD SHUTTER



The requirements of this component are to shade the plant when light intensity exceeds the optimal range, and to remain clear when the plant is below or within the optimal range. Therefore, this component was chosen as once energised when 3v is passed through it will hold their polarised/opaque state. It will then de-polarise and become transparent when no voltage passes through. More expensive LCD shutters could be purchased which were larger in size and would be better suited to the required 2L environment that this subsystem is set to work within.

ADAFRUIT NEOPIXEL RING

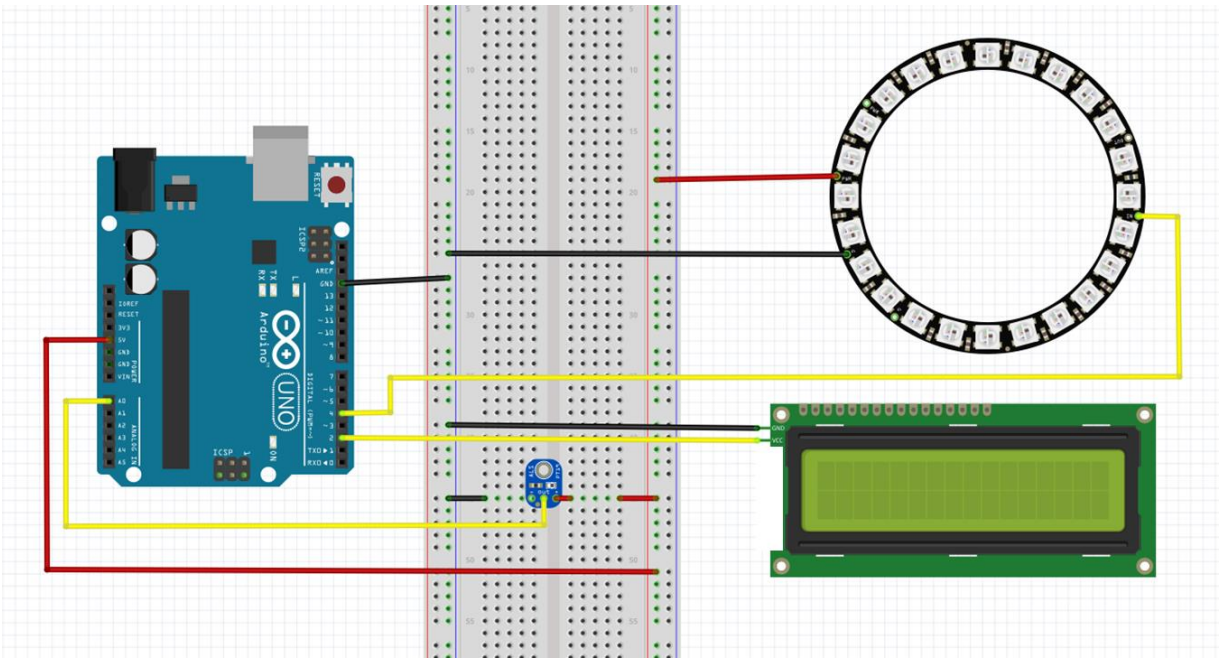
16 X 5050 RGB LED WITH INTEGRATED DRIVERS

This component was chosen as it allows RGB manipulation to ensure that the highest energy light could be produced. Initially an LED array was considered consisting of red and blue LEDs, however after testing, it was found that the LEDs didn't provide enough light to ensure that the plant received a light level within the optimal range.

This component consists of 16 ultra-bright smart LED NeoPixels [10] which will provide the subsystem with sufficient light intensity to ensure the subsystem is maintained within the optimal ranges without the need for an external light source.

PIN LEVEL SCHEMATIC

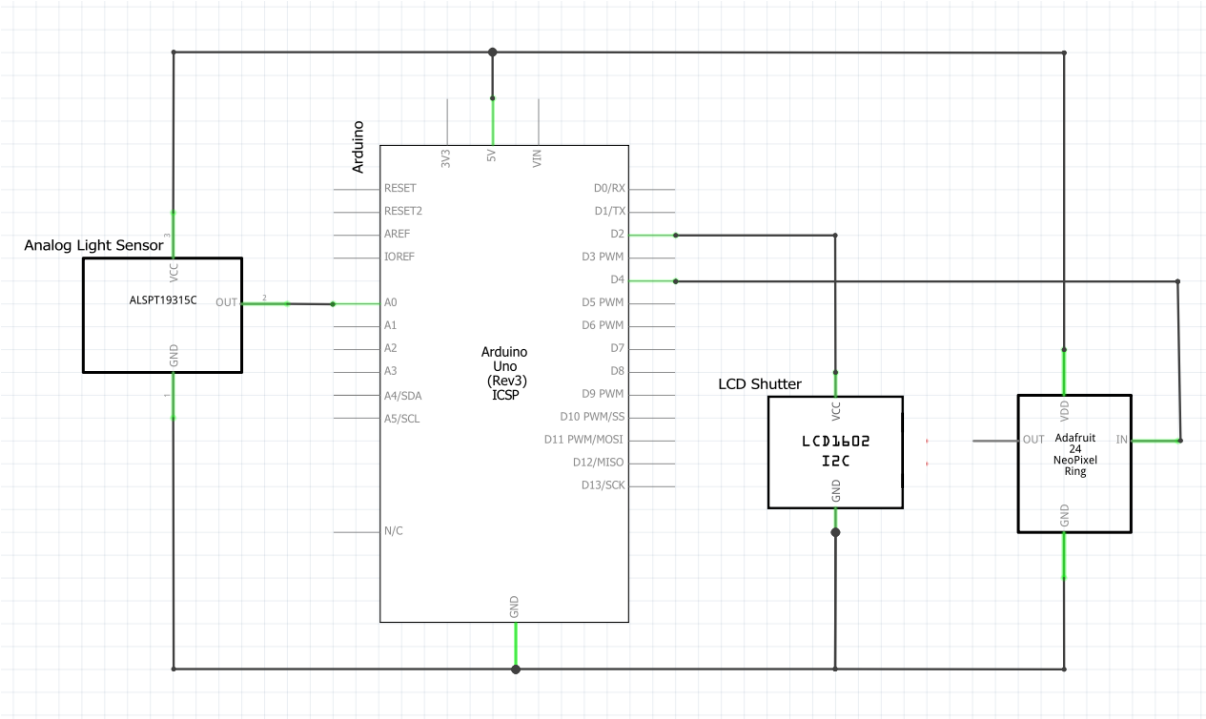
The Pin Level schematic identifies the specific components with the wires and pin numbers related to each. This is similar to the circuit diagram below. This illustrates that the analogue light sensor has three ports connected; GND – GND, VCC – VCC, OUT – A0. It also illustrates the LCD shutter utilises two ports connected; GND – GND, VCC – PIN 2. The NeoPixel ring is also displayed in the diagram, when using this component I will be using 3 of its ports, GND – GND, VCC – VCC, and IN – Pin4.



JAMES STUART BURGESS
201615167

CIRCUIT DIAGRAM

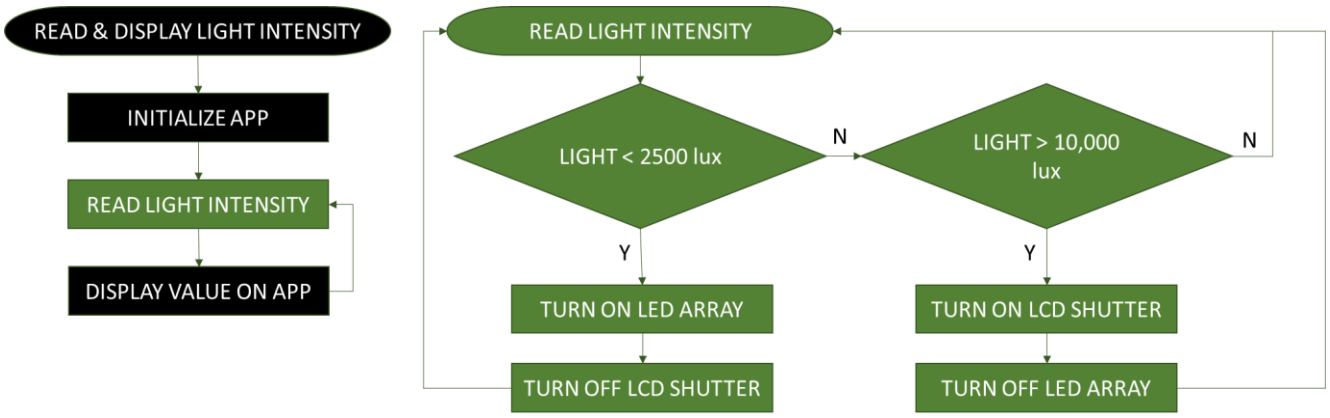
The circuit diagram is used to represent the electrical symbols for the components and their ports. Both the analogue light sensor and the NeoPixel ring have resistors built-in and thus, are not displayed on the circuit diagram.



Both the circuit diagram and pin level schematic can be used to aid the manufacture of the prototype model.

FLOWCHART

The flow chart was created to aid the coding process. It depicts that if light intensity value is greater than 2500 lux and is less than 10,000 the light intensity level is at its optimal level. The system therefore does not need to respond, and it will return to reading the light intensity value. If the value is less than the optimal range the LED array will need to be set to HIGH, and the LCD shutter to LOW to avoid reducing the light intensity. If the value is above the optimal light range, the LCD shutter will be set to HIGH, and the LED array should be set to LOW to avoid increasing the light intensity.



CODE

Within the code the optimal light ranges are defined as 256 < Light < 921. These values correspond to byte values produced by the analogue light sensor which were calculated using the dynamic range of the sensor (0 – 10,000 Lux). With the knowledge that the max number of bytes provided by a 5v source equates to 1024 bytes, it can be assumed that:

$$Max\ lux\ reading = Max\ byte\ reading$$

$$10000\ lux = 1024\ bytes$$

Thus, due to the sensors linear response it can be assumed that:

$$2500\ \ lux \sim 1024\ \times 0.25 = 256\ bytes$$

For the purpose of the prototype and due to the capabilities of the Light sensor utilised the max optimal light intensity must be modified to 9000 lux.

Thus, it can be assumed that:

$$9000\ lux \sim 1024\ \times 0.9\ = 921\ bytes$$

Light_Intensity_NEW_PrintWork

```
int shutterLedPin = 2;                          // shutterLedPin = LCD Shutter glass
int PhotoresistorPin = A0;                      //PhotoresistorPin = Analog Light Sensor ALSPT19
int Light = 0;


#include <Adafruit_NeoPixel.h>                  // include libary for Led light source
#define PIN 4                                   // define pin for Led light source
#define NUM_LIGHTS  16                          // define number of lights on NeoPixel ring

Adafruit_NeoPixel strip = Adafruit_NeoPixel(NUM_LIGHTS, PIN, NEO_GRB + NEO_KHZ800);



void setup() {                                  // setup function:
  Serial.begin(9600);                           // initialize serial communication at 9600 bits per second:
  pinMode(shutterLedPin, OUTPUT);               // shutter is an output
  pinMode(PhotoresistorPin, INPUT);             // Photoresistor (Analogue Light Sensor ALSPT19) defined as an input
  strip.begin();
  strip.show();                                 // Initialize all pixels to 'off'
}


void loop() {                                    // loop function:
  int sensorValue = analogRead(A0);              // read the input from analog pin 0:
Light = analogRead(PhotoresistorPin);            // defines light as input value

float voltage = sensorValue*(5.0/1024.0);        // convert sensor value into voltage, (voltage)/(no. of bytes)
float microamps =(voltage/1000.0)*1000000.0;     // convert value into microamps so Lux value can be read from graph (I=V/R)

Serial.print("Light Current = ");                // prints amp prior to sensor value
Serial.print(microamps);                         // print out the value read
Serial.print(" µA");                             // prints units of value read
Serial.println("");                              // new line between values
delay(1000);                                     // delay for stability
```

```
uint32_t low = strip.Color(0, 0, 0);            //set colour of neopixel ring for low
    uint32_t high = strip.Color(255, 0, 255);   //set colour of neopixel ring for high


  if(Light < 256){                              // condition if light value less than 307, (2500 lux)
    for( int i = 0; i<NUM_LIGHTS; i++){
        strip.setPixelColor(i, high);
        strip.show();}                          // Turn on Led light source
    digitalWrite(shutterLedPin, LOW);           // Turn off
  }


  else if(Light >= 256 && Light <= 921){        // condition if light value in between 307 and 900, lighting is at optimal level (2500<L<10000 lux)
    for( int i = 0; i<NUM_LIGHTS; i++){
        strip.setPixelColor(i, low);
        strip.show();}                          // Turn off Led light source
    digitalWrite(shutterLedPin, LOW);           // Turn off LCD shutter
  }


  else{                                         // condition if light value above 900, (10,000 lux)
    digitalWrite(shutterLedPin, HIGH);          // Turn on LCD Shutter
    for( int i = 0; i<NUM_LIGHTS; i++){
        strip.setPixelColor(i, low);
        strip.show();}                          // Turn off Led light source
  }
  delay(5);                                     // delay for stability

}
```

Comments describing the code and its actions are within the code above. Also, within the code, calculations have been made to convert the received sensor value into Light current so that accurate Lux values can be read from Figure 1. Which was provided by the datasheet for the ALS PT19 light sensor.

TESTING

Testing of the subsystem was conducted within a controlled environment similar to that of the intended terrarium. Using a lux light meter, it was found that the light intensity of the test environment was 603 Lux.

The variable to be manipulated was light intensity and this was varied by introducing a 50-lumen light source at varying distances from the light sensor shown in table 1. Three readings of the light current were then taken from the serial monitor in order to calculate an average. These values being produced by the sensor at each distance. Accurate Lux (Illuminance) readings were then read from figure 1, using the light current calculated within the code and provided within the serial monitor.

A Lux light meter app was also utilised to verify the lux values compiled, and measure the real lux values being produced.

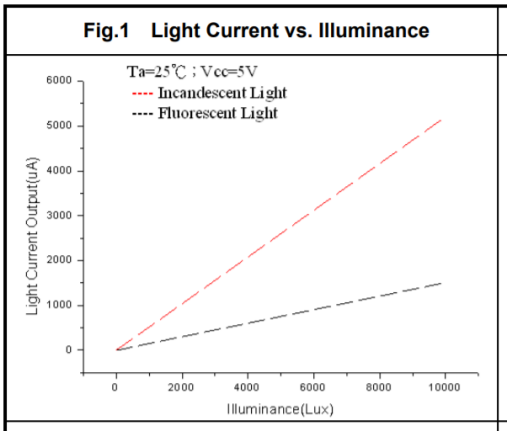


Figure 1: Light Current vs. Illuminance [7]

JAMES STUART BURGESS
201615167

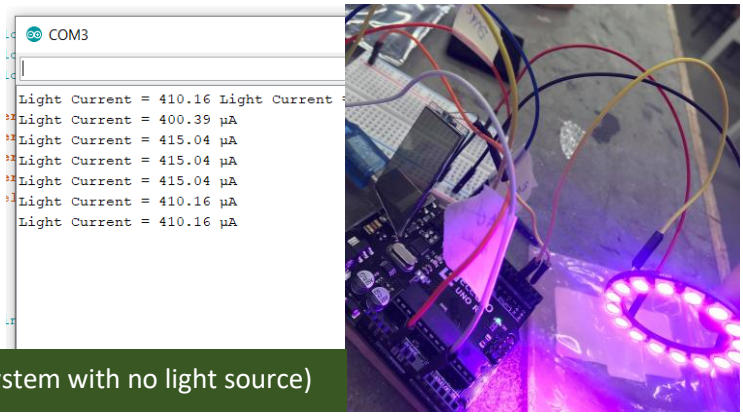
TEST RESULTS

					RESULTS (µA)					
Test No.	Action	DISTANCE FROM LIGHT SOURCE (mm)	MEASURED LIGHT INTENSITY (LUX)	SUBSYSTEM REACTION	1 LIGHT CURRENT	2 LIGHT CURRENT	3 LIGHT CURRENT	Average Current	LIGHT INTENSITY (LUX)	Pass/ Fail
1	No light source	N/A	603lx	LED = on LCD = off	419.92	415.04	424.08	419.68	600lux	pass
2	50 lumen Light source introduced	300	966.6	LED = on LCD = off	727.54	732.42	742.19	734.05	1150 lux	pass
3	" "	250	1209	LED = on LCD = off	859.38	859.38	869.14	862.58	1400 lux	pass
4	" "	200	1853	LED = on LCD = off	1025.39	1020.51	1030.27	1025.39	2000 lux	pass
5	" "	175	2736	LED = off LCD = off	1528.02	1522.41	1508.16	1519.53	3000 lux	pass
6	" "	150	3104	LED = off LCD = off	1684.08	1606.84	1674.65	1655.19	3200 lux	pass
7	" "	125	4284	LED = off LCD = off	2001.04	1962.98	1969.56	1977.86	3900 lux	pass
8	" "	100	6700	LED = off LCD = off	3256.33	3374.48	3369.73	3333.51	6200 lux	pass
9	" "	75	8449	LED = off LCD = on	4746.65	4769.58	4770.89	4762.37	9250 lux	pass
10	" "	50	10604	LED = off LCD = on	4811.02	4806.12	4809.30	4808.81	9700lux	pass

Table 1. Test Results

Light Intensity < 2500 Lux

It was expected that when the light intensity (illuminance) level was below 2500 lux, the LED light source should be set to high, and the light source would be switched on. The LCD shutter would be set to low and be transparent. This occurred during the tests

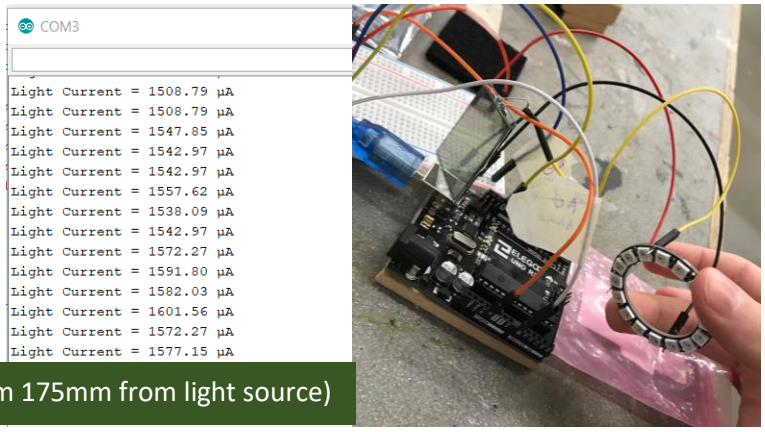


(Subsystem with no light source)

Optimal Light

2500 < Light Intensity < 9000

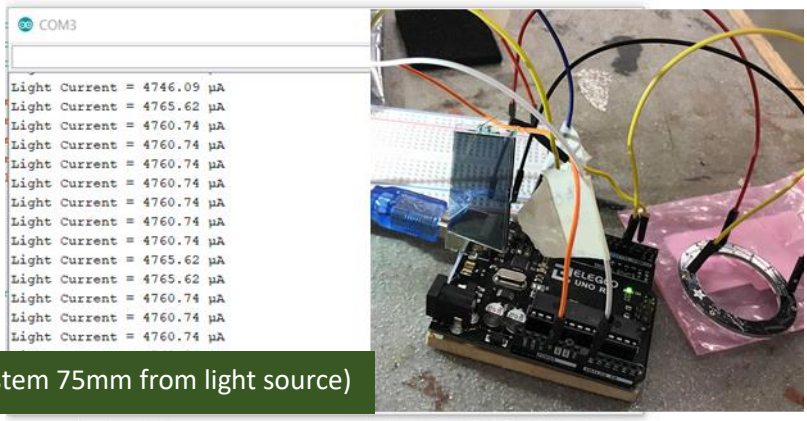
It was also expected that when the light intensity was within the optimal range both the LCD shutter and LED light source should be set to low. Light source off, and LCD shutter transparent. This occurred during the test.



(Subsystem 175mm from light source)

Light Intensity > 9000

It was expected that when the light intensity exceeded the optimal range, the LCD shutter should be set to high and should be opaque. The Light source should be set to low and should be switched off. This occurred during the test.



(Subsystem 75mm from light source)

For the purpose of testing the light current was calculated and used to measure the lux value on figure 1. and compare this to the value measured using the lux light meter app. However, it was noted that the measured light intensity differed from that calculated using the graph provided by the data sheet. A few anomalies occurred within the data captured, this can be assumed to be errors within the measuring apparatus, test conditions and inaccuracies when reading the graph.

CHANGES TO CODE

After testing the subsystem was completed, the code was modified to provide accurate LUX values. This was achieved by finding the gradient of the Light Current vs. Illuminance line in figure 1 (using y=mx+c), and dividing the found light current by this.

```
void loop() {                                   // loop function:
  int sensorValue = analogRead(A0);             // read the input from analog pin 0:
Light = analogRead(PhotoresistorPin);           // defines light as input value

float voltage = sensorValue*(5.0/1024.0);       // convert sensor value into voltage, (voltage)/(no. of bytes)
float microamps =(voltage/1000.0)*1000000.0;    // convert value into microamps so Lux value can be read from graph (I=V/R)
float LUX = (microamps/0.5)                     // converting light current value into illuminance (LUX)

Serial.print("Light Intensity = ");             // prints prior to sensor value
Serial.print(LUX);                              // print out the value calculated
Serial.print(" lux");                           // prints units of value read
Serial.println("");                             // new line between values
delay(1000);                                    // delay for stability
```

LIMITATIONS

One large limitation created by this subsystem was the use of the ALS-PT19 analogue light sensor. As the light sensor only created a response between the ranges of 0 – 10,000 lux, which coincides with the max ideal light intensity (2,500 – 10,000 lux). Due to this the max value of the optimal light intensity range had to be modified to 2,500 – 9,000 lux to ensure that the subsystem reacted to a light intensity greater that the optimal range. Limited access to an accurate and calibrated lux light meter, also limited the testing capabilities, to confirm that the subsystem responded to the correct light intensity.

The size of LCD shutter utilised, will also create a limitation, as the subsystem environment is much greater than the 36x36x2mm shutter used. This will impact the optimal growth of the Coffea Arabica, as it will not be able to prevent large light intensities within the required 2 Litre environment.

CONCLUSION

After testing the subsystem created, it has shown that it will perform as expected and will maintain the optimal lighting intensity required to ensure optimal growth of the Coffea Arabica plant, within the ranges of 2500 lux to 10000 lux. The testing also highlighted that the light source, created the required high energy light at the purple end of the light spectrum.

One aspect of the subsystem that may need to be considered further is the lighting aspect, and its position within the smart terrarium in comparison to the light sensor. This may cause a blink to occur within the subsystem as when the light source is activated.

In ideal circumstances an array of LCD shutters would be utilised to shade the entire terrarium environment and respond individually, however this would require an array of light sensors to control the input to these, which would substantially increase the cost of this subsystem.

Further modifications may need to be introduced to the subsystem and the component connections when the system needs to utilise the ESP32 rather than the current Arduino Uno R3. Further modifications may also need to be added into the coding of this subsystem in preparation for the combination with the temperature, humidity, and soil moisture subsystems.

REFERENCES

[1] JEANTY, J. (2019). The Effect of Light Intensity on Plant Growth. [online] https://www.hunker.com/. Available at: https://www.hunker.com/12340735/the-effect-of-light-intensity-on-plant-growth [Accessed 20 Oct. 2019].

[2] Aggie-horticulture.tamu.edu. (2019). Light, Temperature and Humidity | Ornamental Production. [online] Available at: https://aggie-horticulture.tamu.edu/ornamental/a-reference-guide-to-plant-care-handling-and-merchandising/light-temperature-and-humidity/ [Accessed 18 Oct. 2019].

[3] Plantopedia. (2019). Coffee Plant Care - How to Grow Coffe Arabica Indoors - Plantopedia. [online] Available at: https://www.plantopedia.com/coffea-arabica/ [Accessed 23 Oct. 2019].

[4] H. Trinklein, D. (2019). [online] Extension2.missouri.edu. Available at: https://extension2.missouri.edu/g6515#types [Accessed 26 Oct. 2019].

[5] Sensing, K. (2019). Can Colored Lights Affect How Plants Grow?. [online] Konica Minolta Sensing Americas. Available at: https://sensing.konicaminolta.us/blog/can-colored-lights-affect-how-plants-grow/ [Accessed 28 Oct. 2019].

[6] Watkins, D. (2016). The Best Plant Light Spectrum for Growing Flowering Plants. [online] Homeguides.sfgate.com. Available at: https://homeguides.sfgate.com/plant-light-spectrum-growing-flowering-plants-72801.html [Accessed 27 Oct. 2019].

[7] Cdn-shop.adafruit.com. (2013). Ambient Light Sensor Surface - Mount ALS-PT19-315C/L177/TR8. [online] Available at: https://cdn-shop.adafruit.com/product-files/2748/2748+datasheet.pdf [Accessed 12 Oct. 2019].

[8] Rapidonline.com. (2019). Adafruit 2748 ALS-PT19 Analog Light Sensor Breakout. [online] Available at: https://www.rapidonline.com/Adafruit-2748-ALS-PT19-Analog-Light-Sensor-Breakout-73-5282?IncVat=1&pdg=pla-339795874228:kwd-339795874228:cmp-757438067:adg-44804851896:crv-207912323492:pid-73-5282:dev-c&gclid=Cj0KCQjwrrXtBRCKARIsAMbU6bHK4SBRnNRtbt_kuh9szqOj1Ov7adMOqaE7jQAFqhlZlj-FIdZ7fI4aAhu-EALw_wcB [Accessed 15 Oct. 2019].

[9] Kitronik. (2019). How an LDR (Light Dependent Resistor) Works. [online] Available at: https://www.kitronik.co.uk/blog/how-an-ldr-light-dependent-resistor-works/ [Accessed 18 Oct. 2019].

[10] Industries, A. (2019). NeoPixel Ring - 16 x 5050 RGB LED with Integrated Drivers. [online] Adafruit.com. Available at: https://www.adafruit.com/product/1463 [Accessed 27 Oct. 2019].

DEPARTMENT OF DESIGN, MANUFACTURE & ENGINEERING MANAGEMENT

University of Strathclyde Engineering

DM401: Advanced Product Design and Manufacture

INDIVIDUAL ASSESSMENT – TEMPERATURE SUBSYSTEM OF TERRARIUM

Student: Werner Landman 201839396

Course: Manufacturing Engineering and Management

Statement of academic honesty:

I declare that this submission is entirely my own original work.

- I declare that this assignment is entirely my own original work.
- I declare that, except where fully referenced direct quotations have been included, no aspect of this assignment has been copied from any other source.
- I declare that all other works cited in this assignment have been appropriately referenced.
- I understand that any act of Academic Dishonesty such as plagiarism or collusion may result in the non-award of my degree.

Signed: *Werner Landman*

Date: 1 November 2019

1. Sub-system specification:

The temperature parameters for this sub-system is based on the Coffea Arabica Pacas (botanical name) dwarf coffee plant. This plant will have similar requirements to its species coffea Arabica located within the “coffee belt” [Scott, 2015]. Thus, the temperature parameters for this plant type is applicable and easier to access.

18-21 degrees Celsius represents the optimal range for this type of coffee. Beyond 23 degrees Celsius, acceleration of the ripening process occurs and increases the chance of pest infestation [Poltronieri, 2016].

To summarise the operating range:

	$°C$
Lowest	18
Highest	21
Danger	23

2. Block diagram schematic:

To determine the hardware needed for this sub-system, a block diagram will enable a better understanding of the general requirements:

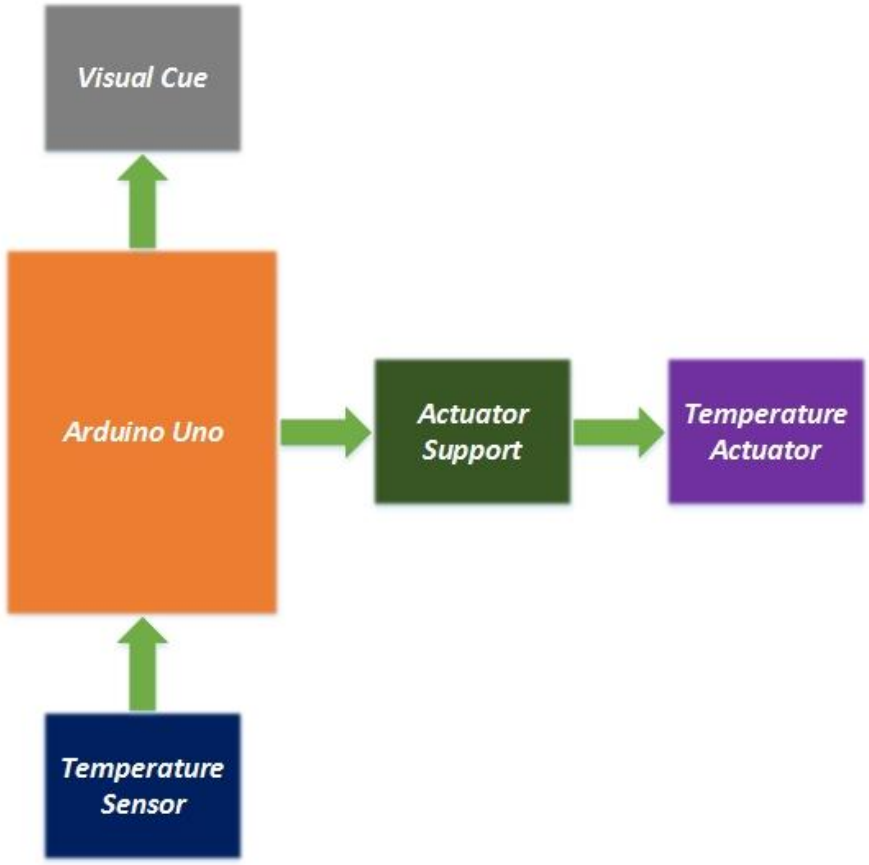


Figure 1: Block Diagram of temperature sub-system

3. Flowchart:

To chart the temperature process logic (which will later form the basis of programming logic) consider the following flowchart:

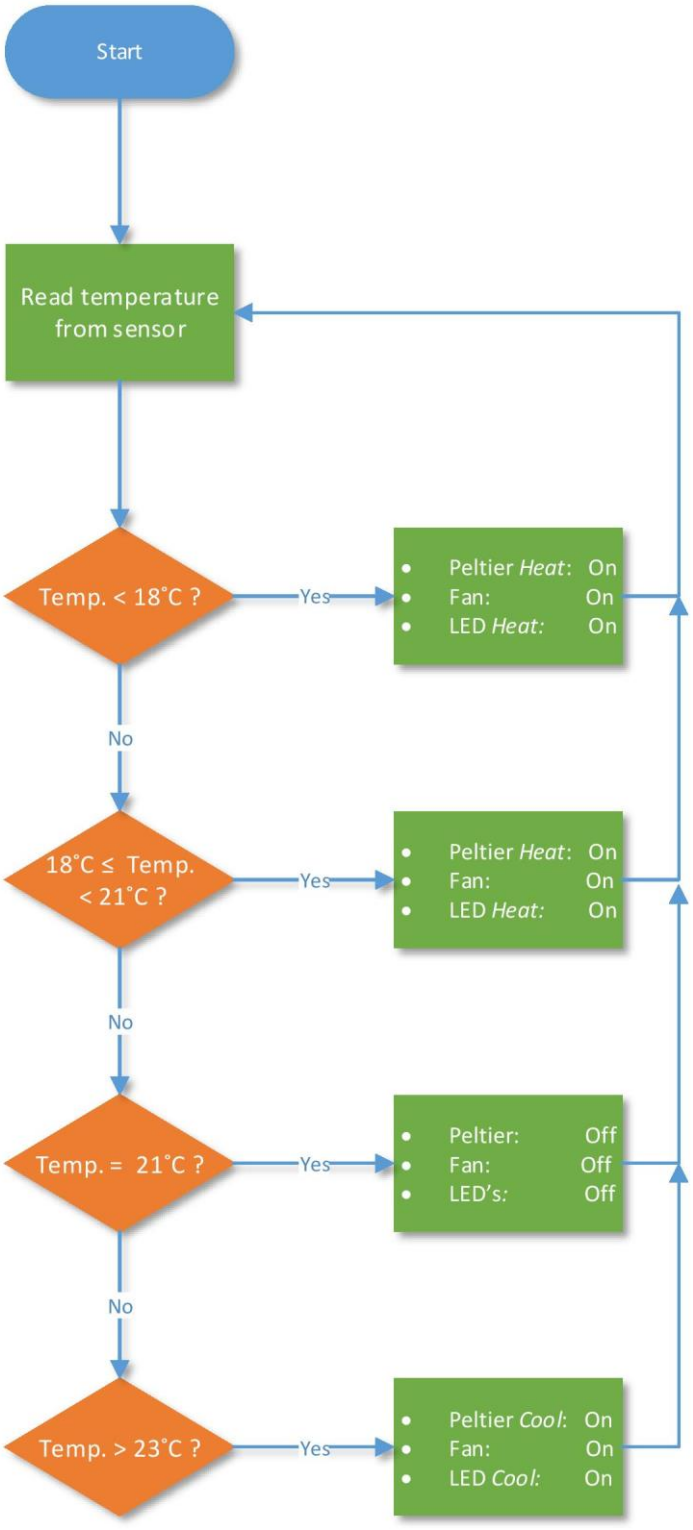
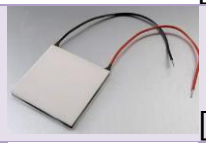
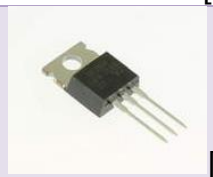
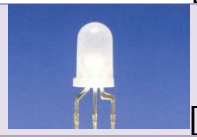


Figure 2: Flowchart of temperature logic

4. Components:

From block diagram	Technical embodiment		Function	Reason
Arduino Uno R3		 [1]	Controls function of hardware	-
Temperature sensor	DHT11	 [2]	Digital humidity and temperature	To reduce the number of components for both the heating and humidity sub-systems.
Temperature actuator	Peltier module	 [3]	Heat control	-
Actuator support	Battery 9V	 [4]	Additional power supply for the Peltier module and fan	-
-	N-channel MOSFET	 [5]	Power regulator	Prevents components being damaged by excess current/voltage through the drainage function.

-	H-Bridge (L293D)	 [6]	Reverses polarity of input components	Enables the Peltier Module to reverse the heating and cooling side respectively
-	Heat sink	 [7]	Increases heating efficiency of the Peltier Module	Reduces the time reach to the specified temperatures
-	Fan	 [8]	Increases heating efficiency of the Peltier Module	Reduces the time reach to the specified temperatures
Visual cue	Light Emitting Diode	 [9]	Shows heat/cool down of system	Visual diagnostic of sub-system phase
Arrows	Wires	 [10]	Enables electrical connection of components	-
Electrical stability	Resistors	 [11]	Reduces the current/voltage through components	Increases the functional life of components and prevents burnouts

Sub-system component cost:

Component	Quantity	Price*
Arduino Uno R3	1	Given
DHT11	1	£3.90 (Shared)
Peltier junction	1	Given
Battery 9V	1	£2.12
N-channel MOSFET	1	£2.21
L293D	1	£3.14
Heat sink	1	Given
Fan	1	£2.97
Light Emitting Diode	2	Given
Wires	-	Given
Resistors	-	Given
Total		£14.34

*Prices based on the websites Amazon and RS Components.

5. Pin level schematic & circuit diagram:

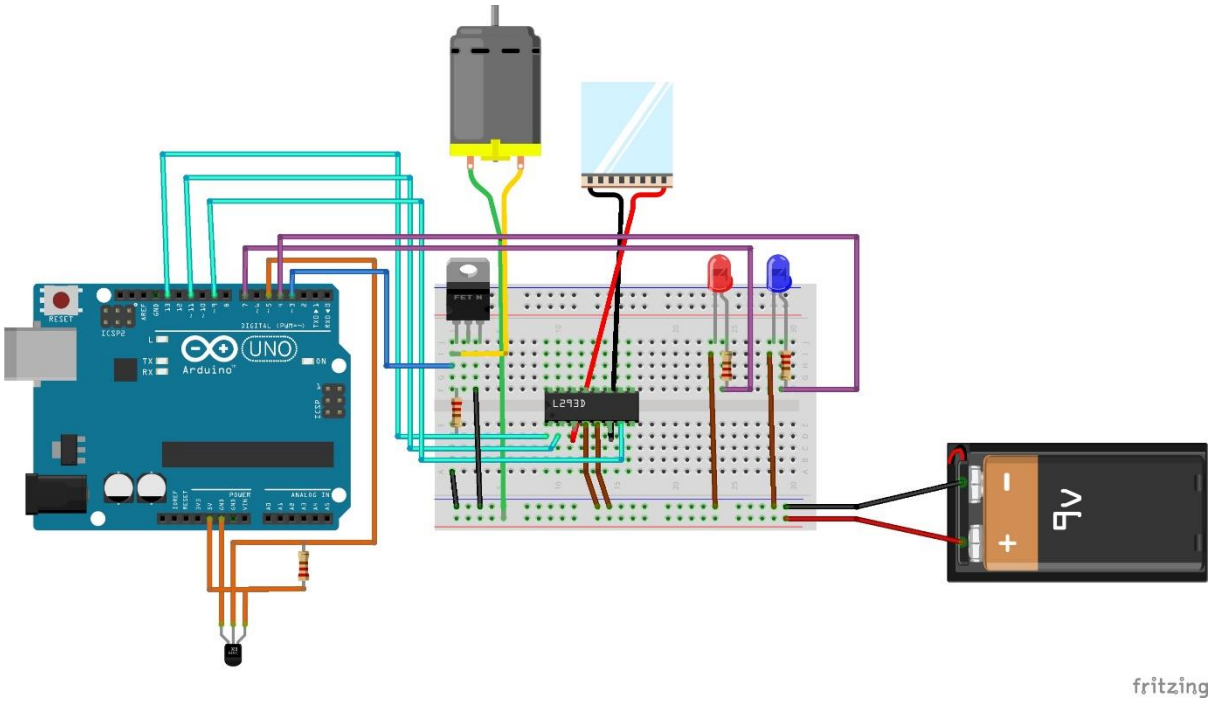


Figure 3: Fritzing diagram

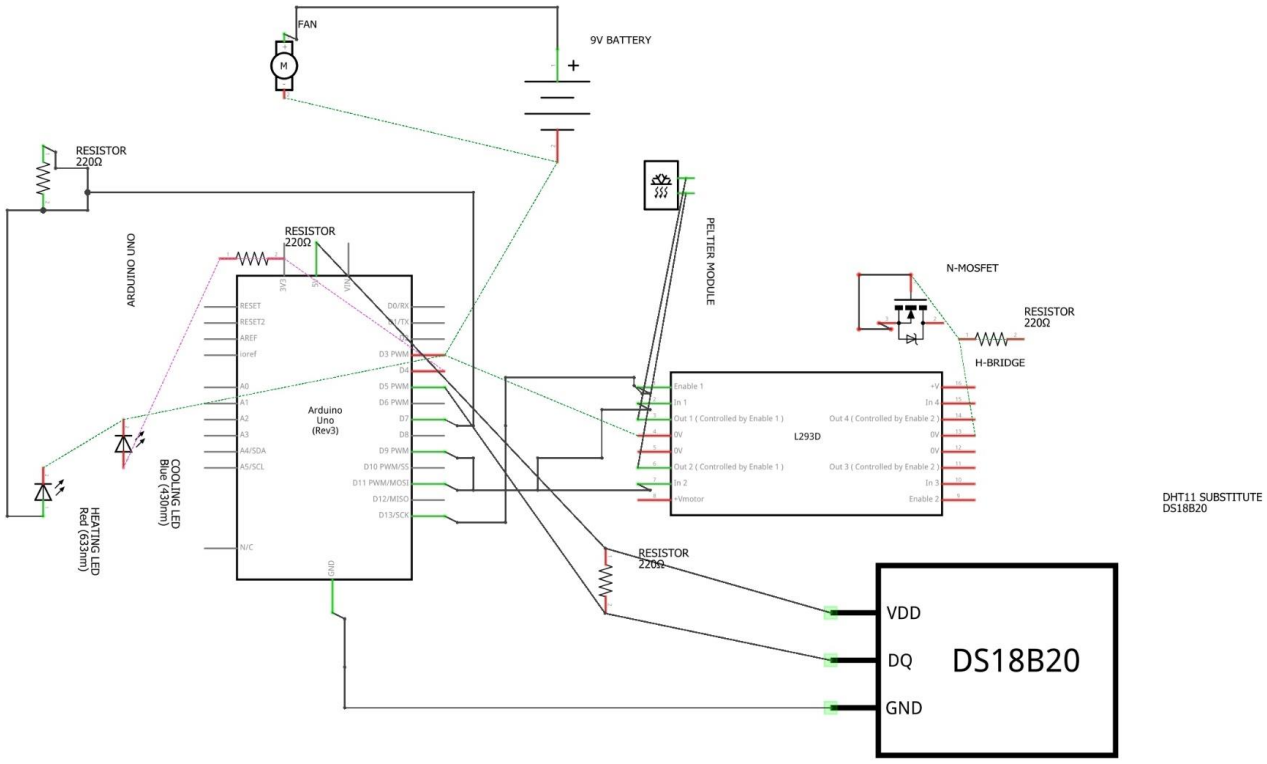


Figure 4: Pinout & circuit diagram:

6. Programming code:

Refer to appendix I for the full programme code and explanatory comments. The code for this system developed as the process continued.

7. Testing sub-system behaviour:

Simulation:

The website Tinker CAD enables the simulation of hardware reactions in according to programming routines. Unfortunately, the components needed to create this system is not available on the website. The temperature sensor (DHT11) and Peltier Module do not appear as components.

Thus, the DHT sensor was replaced by the TMP36 (an analogue sensor) and the Peltier Module was replaced by a resistor (same heating element principle within kettles).

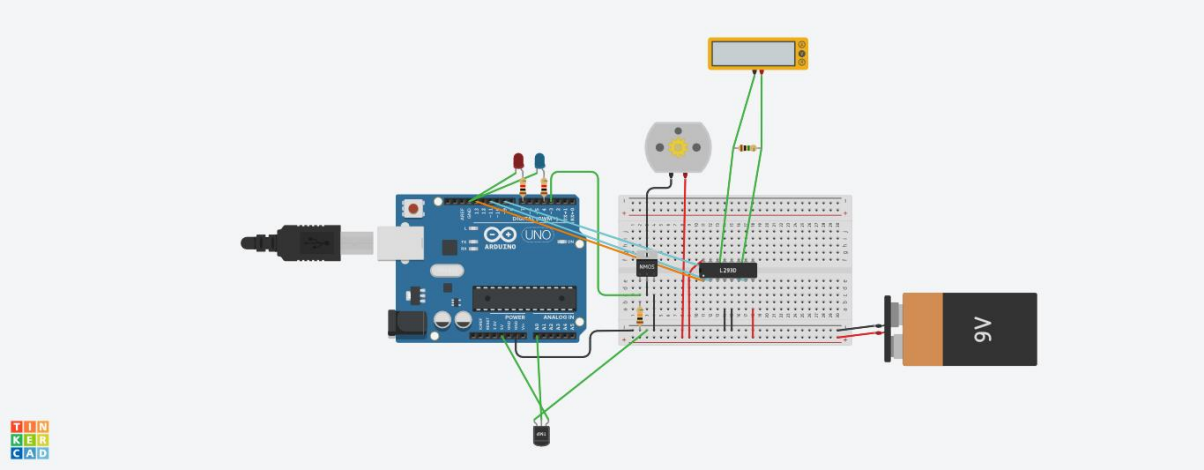


Figure 5: TinkerCad simulation setup

A TMP36 sensor requires different coding from the DHT11. The main purpose of this simulation was to achieve the coordinated function of actuators in response to sensor values.

Physical creation of sub-system:

To enable the physical creation, the pinout diagram was used as the hardware. To enable the code for the system, the TinkerCAD code was amended (for the DHT11) and evaluated through the system

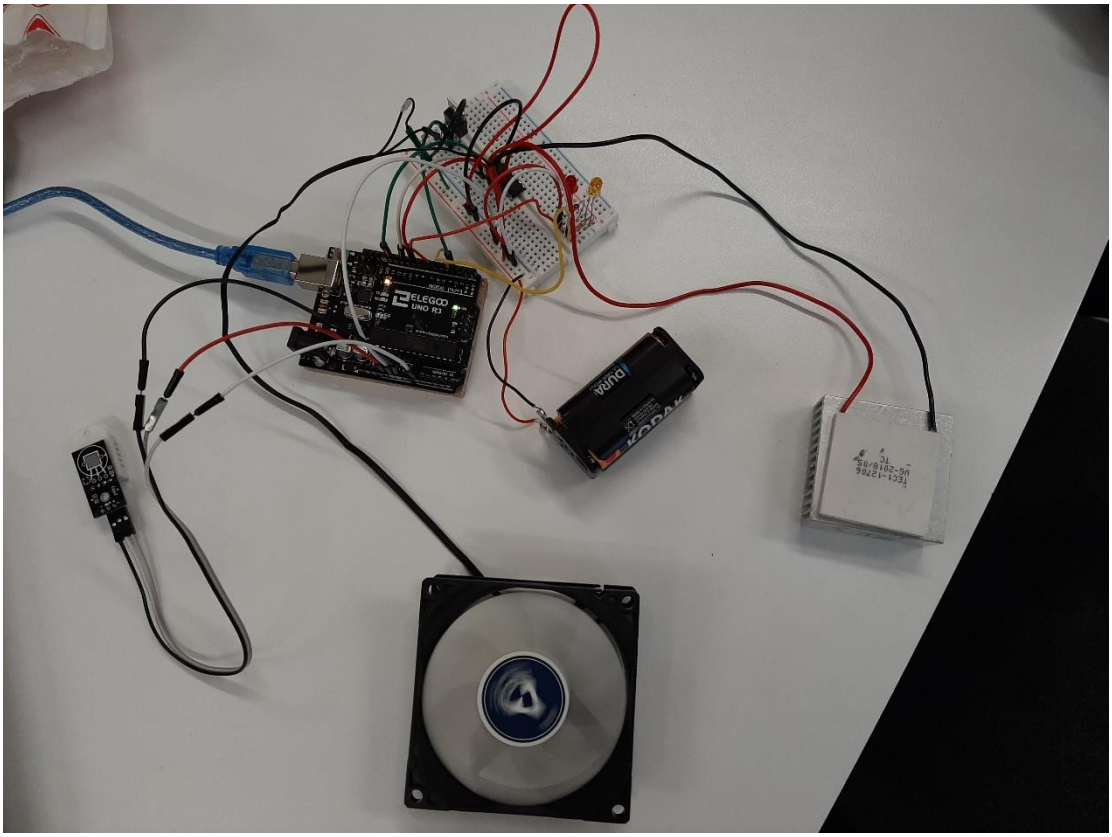


Figure 6: Testing of temperature system

.

The experiment to determine heating/cooling response time included holding the DHT11 sensor approximately 10cm from the Peltier Module, while the programme code was running.

Time it takes to reach the target temperature using a 9V battery:

Test	**Duration (s)**
1	12
2	8
3	13
4	9
Average	**11**

Time it takes to reach minimum temperature threshold using a 9V battery:

Test	**Duration (s)**
1	25
2	36
3	30
4	25
Average	**29**

8. Limitations:

Operating precision of components:

To quantify the limitations of the hardware, refer to temperature relevant details specified by the manufacturer:

DHT11 [Texas Instruments, 2015]:

- Measuring range 0 ~ 50 ℃
- Response 5-30 s
- Accuracy ±1 ~ 2 ℃

Peltier module [Peltier Cooling Module]

- Range −55 ~ 80 ℃
- Time to heat up 7s*
- Time to cool down 30s*

**Under specified conditions of 9V.*

From this, the code for this sub-system was changed to portray a more realistic system.

9. Conclusion:

Further experimentation with response time is required to fully quantify and then enable prediction of system behaviour. This will take place before the integration with other sub-systems.

Appendices:

Appendix I:

```
//Administrative Functions

#include <DHT.h>                          //Retrieve DHT related information from the
                                          downloaded library
#define DHTPIN 5                          //Specify Pin 5 as DHT
#define DHTTYPE DHT11                     //Specify DHT type = DHT11
DHT dht(DHTPIN,DHTTYPE);

const int HEAT_LED=7;                     //Names pin 7 as Heating LED
const int COOLING_LED=4;                  //Names pin 4 as Cooling LED
const int MOSFET=3;                       //Names pin 3 as MOSFET

int chk;
float temperature;                        //Expresses temperature as 10 bit number

double targettemp=21;                     //Specify number related to targettemp

//Setup Programme
void setup()
{
Serial.begin(9600);
dht.begin();                              //Enables function of DHT

  pinMode(MOSFET, OUTPUT);                //Declares MOSFET as an output pin
          pinMode(13, OUTPUT);            //Declares pin 13 as an output {part of H-
                                          Bridge}
  pinMode(11, OUTPUT);                    //Declares pin 11 as an output {part of H-
                                          bridge}
  pinMode(9, OUTPUT);                     //Declares pin 9 as an output  {part of H-
                                          bridge}
  pinMode(HEAT_LED,OUTPUT);               //Declares HEAT_LED as an output
  pinMode(COOLING_LED,OUTPUT);            //Declares COOLING_LED as an output
  pinMode(5,INPUT);                       //Declares DHT as an input
  digitalWrite(13, HIGH);                 //Enables function of the H-bridge
}

//Run programme continously
void loop()
{

 //Checks temperature readings
temperature = dht.readTemperature();      //Read temperature from DHT
Serial.print("Temperature:");             //Print specified characters on serial monitor
Serial.print(temperature);                //Print temperature reading from DHT on
                                          serial monitor
Serial.println("Degrees Celsius");        //Print specified characters on serial monitor
```

```
delay(2000);                          //Programme initiation delay - improves
                                      performance

//Temperature control through "if" statements
if(temperature<targettemp)

//DHT temperature reading below 21, use Peltier module & fan to heat terrarium
{
analogWrite(MOSFET, 50);              //Fan operates at a specified level
  digitalWrite(11, LOW);              //Peltier Module one side cold
  digitalWrite(9, HIGH);              //Peltier Module one side warm
  digitalWrite(HEAT_LED,HIGH);        //Red light on
  digitalWrite(COOLING_LED,0);        //Blue light off
}

if(temperature==targettemp)
//DHT temperature reading = 21, turn off system
{
 analogWrite(MOSFET, 0);              //Fan does not function
  digitalWrite(11, LOW);              //Peltier Module off
  digitalWrite(9, LOW);               //Peltier Module off
  digitalWrite(HEAT_LED,0);           //Red light off
  digitalWrite(COOLING_LED,0);        //Blue light off
}

if(temperature>targettemp+2)

//DHT temperature reading above 23, use Peltier module & fan to cool terrarium
{
analogWrite(MOSFET, 50);              //Fan operates at a specified level
  digitalWrite(11, HIGH);             //Peltier Module one side warm
  digitalWrite(9, LOW);               //Peltier Module one side cold
  digitalWrite(HEAT_LED,0);           //Red light off
  digitalWrite(COOLING_LED,HIGH);     //Blue light on
}
delay(1000);                          //Programme initiation delay - improves
                                      performance
}
```

References:

Scott, M., 2015. *Climate and coffee.* [Online] https://www.climate.gov/news-features/climate-and/climate-coffee [Accessed: 29 October 2019].

Poltronieri, P & Rossi, F., 2016. *Challenges in Speciality Coffee Processing and Quality Assurance.* [Online] Available from: https://www.mdpi.com/2078-1547/7/2/19/htm [Accessed: 26 October 2019].

DHT11 Humidity & Temperature Sensor. [Online] Available from: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf [Accessed: 26 October 2019].

Peltier Cooling Module – TEC1 -12704. [Online] Available from: https://www.circuitspecialists.com/content/81792/tec1-12704.pdf [Accessed: 26 October 2019].

Texas Instruments, 2015. *LM3488/-Q1 Automotive High-efficiency Controller for Boost, SEPIC and Fly-back DC-DC Converters.* [Online] http://www.ti.com/lit/ds/symlink/lm3488.pdf [Accessed: 26 October 2019].

Texas Instruments, 2016. *L293x Quadruple Half-h Drivers.* [Online] Available from: http://www.ti.com/lit/ds/symlink/l293.pdf [Accessed: 26 October 2019].

Pictures:

[1] [Online] Available from: https://media5.picsearch.com/is?wpMfakZ-828vtdUF9MtCH3VTsG-frH08ENhgUX0x0wc&height=341 [Accessed: 1 November 2019]

[2] [Online] Available from: https://media1.picsearch.com/is?rvIYtp42u0jCrQNVegrBPJs4oA7bIr8_4HloejPS2hA&height=199 [Accessed: 1 November 2019]

[3] [Online] Available from: https://media1.picsearch.com/is?rU5GA4eUNjC3Qj5aDaUJu-AV6BBrq-q3IKgNLdwS2a4&height=227 [Accessed: 1 November 2019].

[4] [Online] Available from: https://media5.picsearch.com/is?ecRXLamDVzVoBMwva9rP_fHcL4IsdCA3X--DoUe3LtU&height=240 [Accessed: 1 November 2019].

[5] [Online] Available from: https://media4.picsearch.com/is?xf0zVcL2ziSVnGqviyj0JZS-DE_UbXuJDpUX829Qsi4&height=274 [Accessed: 1 November 2019]

[6] [Online] Available from: https://media2.picsearch.com/is?8DALokNH6oPrQWXvXXNFjtYQ2sDl4tvUjqnR3I0VKBU&height=281 [Accessed: 1 November 2019].

[7] [Online] Available from: https://media5.picsearch.com/is?vGfjQG-nen4CaT2a-DyG8NbQfyvTrYxJ7XooBXh-Okk&height=265 [Accessed: 1 November 2019]

[8] [Online] Available from: https://media4.picsearch.com/is?LvgxtxqAASUsy8TQ7jKrk1cbuot0bLAyrQqBDm_IvJk&height=226 [Accessed: 1 November 2019]

[9] [Online] Available from: https://media1.picsearch.com/is?JX0zRwh8pbrjKQynAZAO4BQHFgvgjn8a-9zEMRTCT6E&height=341 [Accessed: 1 November 2019]

[10] [Online] Available from: https://media4.picsearch.com/is?jPr2SjmWj2vgY7RePxJJI-adG1mVd9T3p1ZpDcB-Ax8&height=210
[Accessed: 1 November 2019]

[11] [Online] Available from: https://media4.picsearch.com/is?2KMNrpRux23vNcH6-FI2sF1b6rnyLUigdkbIvOadr8M&height=194 [Accessed: 1 November 2019]