The Irrigation Controller Manual

1st edition

Refreeno Harvey & Sean Eaton

2018

Contents

[User Interface 2](#_Toc518330707)

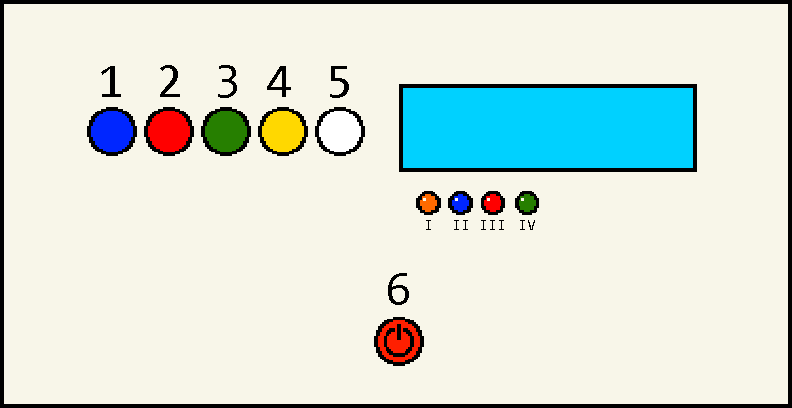
[Modes & Operation 3](#_Toc518330708)

[Maintenance 5](#_Toc518330709)

[Diagnostics & Troubleshooting 8](#_Toc518330710)

[Code Documentation & Explanation 23](#_Toc518330711)

# User Interface



LCD Screen

Controller Legend:

Buttons:

1. Yes / Addition
2. No / Subtraction
3. Next
4. Mode
5. Reset
6. Power

LEDs:

1. Water Level Indicator
2. Pump Circulator
3. Tank Valve
4. Drain Valve

Explanation of Buttons:

The controller will sometimes ask you a yes or no question. The blue button (1) is used to answer yes. The red button (2) is used to answer no. In Parameter Mode the blue button can be used to increase a number and the red to decrease it. The green next button (3) is used to move onto the next parameter in Parameter Mode and to move onto the next sensor in Sensor Read Mode. The yellow mode button (4) is used to switch between the four different modes. The white reset button (5) resets the controller. The power button (6) is black on the physical controller and it just turns the controller on or off.

Explanation of LEDs I – IV:

If the water level indicator light (I) is on then the tank is running low on water and must be refilled. The pump circulator light (II) will turn on while the pump circulator is in use. The tank valve (III) and drain valve (IV) lights will turn on when the respective valves are open.

# Modes & Operation

Overall Operation:

Connect all the plugs in the junction box to the correct place on the controller. Turn the controller on by pressing the power button. The controller will likely start in standby mode unless the time is exactly right for auto mode to begin. The yellow mode button (4) allows you to cycle through four various modes: standby mode, parameter mode, manual mode, and sensor read mode. While in auto mode you cannot switch to a different mode.

Standby Mode Operation:

This mode continuously checks the time and when the time is exactly on the hour it will switch to auto mode.

Auto Mode:

This mode automatically checks the water’s temperature, the temperature outside the tank, the water level in the tank, and the soil moisture sensors to decide if the tank valve should be opened for watering. While watering the soil moisture levels are monitored and watering is stopped once moisture levels are satisfactory.

Parameter Mode Operation:

Parameter mode allows you to adjust four parameters.

* **UPPER THRESH VWC**: the soil moisture level that signals the water to turn off (to provide more water, increase value)
* **LOWER THRESH VWC**: the soil moisture level that signals the water to turn on (to water more frequently, lower value)
* **TEMP UPPER LIMIT**: the maximum temperature the water can be for watering
* **TEMP LOWER LIMIT**: the minimum temperature the water can be for watering

These parameters are used in auto mode to decide if the garden should be watered or not.

How to adjust parameters:

1. Use the mode button to switch to Parameter Mode and push next to begin.
2. Adjust **UPPER THRESH VWC** using the add and subtract buttons and push next when you are done.
3. Repeat step 2 for **LOWER THRESH VWC**, **TEMP UPPER LIMIT**, and **TEMP LOWER LIMIT** parameters.
4. The controller will then display all these parameters on one screen. Check to make sure these values are correct and push next.  
   VU = UPPER THRESH VWC | VL = LOWER THRESH VWC  
   TU = TEMP UPPER LIMIT | TL = TEMP LOWER LIMIT
5. The last screen will ask if you want to save these parameters, answer using the blue or red button (yes or no).
6. Once saved these parameters will remain in the controller’s memory even if the controller is turned off or loses power.

Manual Mode Operation:

1. Use the mode button to switch to Manual Mode.
2. While in Manual Mode the blue, red, and green buttons are used to control the pump circulator, tank valve, and drain valve respectively.
3. To turn on the pump circulator push the blue button once. The blue pump circulator light should turn on and a whirring sound from inside the tank should be audible. To turn the pump circulator off simply push the blue button once again.
4. To open the tank valve, push the red button once. While the valve is open the red tank valve light should turn on and you should be able to hear the valve make a click sound when opening and closing. To close the tank valve, push the red button once again. After the tank valve is closed the drain valve is automatically opened to allow excess water to drain.
5. To open the drain valve, push the green button once. It is not normally necessary to manually open the drain valve because it is always automatically opened after the tank valve. However, manual control over the drain valve would be useful if you need to test the valve to make sure it is functional or if you need to manually drain the pipes for some reason. The drain valve is located in a nearby valve box with a green lid. A click should be heard when the drain valve is opened and closed. To close the drain valve push the green button once again.

Sensor Read Mode Operation:

1. Use the mode button to switch to Sensor Read Mode and push next.
2. “Water Level Sensor” should be displayed on the screen. If there is water touching the sensor then the water level indicator light will be dark. If there is no water touching the sensor the light will be lit. Push next to read the next sensor.
3. “Temperature Sensor” should be displayed on the screen. In testing the water temperature stayed in the 60 – 70 °F range. Push next to read the next sensor.
4. “Moisture Sensors” should be displayed on the screen. S1 is the VH400 sensor and S2 is the EC5 sensor. The average of these two values is the overall volumetric water content that is used in auto mode calculations.

# Maintenance

Note: Two people may be needed. To EXIT at any time during this operation or when done, press the mode button twice to switch back to standby mode.

**Checking Valves and Pump Circulator:**

1. Go to Manual Mode using the mode button. Press it two times if you are in standby mode

*Pump Circulator:*

1. To turn on the pump circulator press the blue button once. The blue light should come on and a whirring sound should be heard coming from the pump inside the tank.
   1. If pump circulator does not turn on, check pump circulator connections.
   2. If it still does not turn on, remove it:
      1. To check if clogged, unclip the black cap and inspect.
      2. Then reassemble and reconnect the pump circulator.
   3. If it still does not turn on then replace the pump circulator.
2. When replaced and connected, press the blue button once. The blue light should come on and a whirring sound should be heard.
   1. If it still does not turn on see Diagnostics & Troubleshooting section B.

*Tank Valve:*

NOTE: When the tank valve turns off, the drain valve automatically turns on after and closes after a set drain duration.

1. To open the tank valve, press the red button once. The red light should come on and a clicking sound should be heard coming from the valve.
   1. If valve does not open check the valve connections.
   2. If valve still does not open, remove the valve and check if it is clogged.
      1. Unscrew the four screws on the brass valve and inspect the inside (see the valve manufacturer’s manual and maintenance documents for more information).
      2. Reassemble the valve, reconnect, and test again.
   3. If valve still does not turn on, replace the valve.
2. When replaced and connected, press the red button. The red light should come on and a clicking sound should be heard coming from the valve.
   1. If valve still does not turn on, see Diagnostics & Troubleshooting section B.

*Drain Valve:*

1. To see if the drain valve opens, press the green button. The green light should turn on and a clicking sound should be heard coming from the valve. The drain valve is located in a nearby valve box with a green lid.
   1. Repeat steps 4a – 5a.

**Checking Sensors:**

1. Go to Sensor Read Mode using the mode button.

*Water Level Sensor:*

1. Press next once and the screen should display “Water Lvl Sensor”
2. The water level sensor can be tested by removing it from the tank and leaving it in the dry air (NOTE: no water should be on the tip of the sensor).
   1. The water level indicator light (orange) should turn on when the water level sensor is dry. The light signals that the tank needs to be refilled.
   2. When the sensor is placed in water the water level indicator light turns off meaning that the water level in the tank is satisfactory.
   3. If the water level indicator light does not turn off when placed in water, check sensor connections.
   4. If the light still does not turn off, replace the sensor.
3. When replaced and connected repeat step 3.
   1. If the water level indicator light does not turn off, see Diagnostics & Troubleshooting section D-2.
4. Navigate to the Temperature Sensor display by pressing next while in Sensor Read Mode. The temperature from the sensor is displayed in degrees Fahrenheit.
   1. If the temperature sensor reading does not display, check the sensor connections.
      1. If it still does not read, replace the sensor and reconnect.
      2. If it still does not read, see the Diagnostics & Troubleshooting section D-1.
   2. If the temperature sensor reading does display.
      1. Have the sensor on hand, an additional thermometer, and a cup of water for testing.
      2. Place the sensor in water and note the temperature before and after.
      3. If the temperature sensor reading does not change, check sensor connections.
         1. If it still does not change, replace the sensor.
         2. If it still does not change, see Diagnostics & Troubleshooting section D-1.
      4. The sensor should be compared with the additional thermometer.
         1. The temperature sensor has a margin of error of 2 degrees F. If the difference is greater than 2 degrees the sensor should be calibrated (have manufacturer perform this task).
5. Navigate to the Moisture Sensor display by pressing next while in Sensor Read Mode. The Volumetric Water Content (VWC) value for each sensor displays on the screen. S1 is the VH400 sensor and S2 is the EC5 sensor.
   1. If a moisture sensor reading does not read, check sensor connections.
      1. If it still does not read, replace sensor and reconnect.
      2. If it still does not read, see Diagnostics & Troubleshooting Section D-3.
   2. Have the sensors on hand and a cup of water.
   3. Move the sensor in and out of the water.
   4. If the reading does not change when moved, check sensor connections.
      1. If it still does not change, replace the sensor.
   5. When replaced and reconnected, repeat step 6c and 6d.
      1. If it still does not change, see Diagnostics & Troubleshooting section D-3.
   6. If the sensor’s reading is negative, then the sensor should be calibrated (have manufacturer perform this task).
   7. The same procedure can be followed again from steps 6a—6f for the other sensor.

NOTE: Soil moisture sensors should be calibrated every year (have manufacturer perform this task).

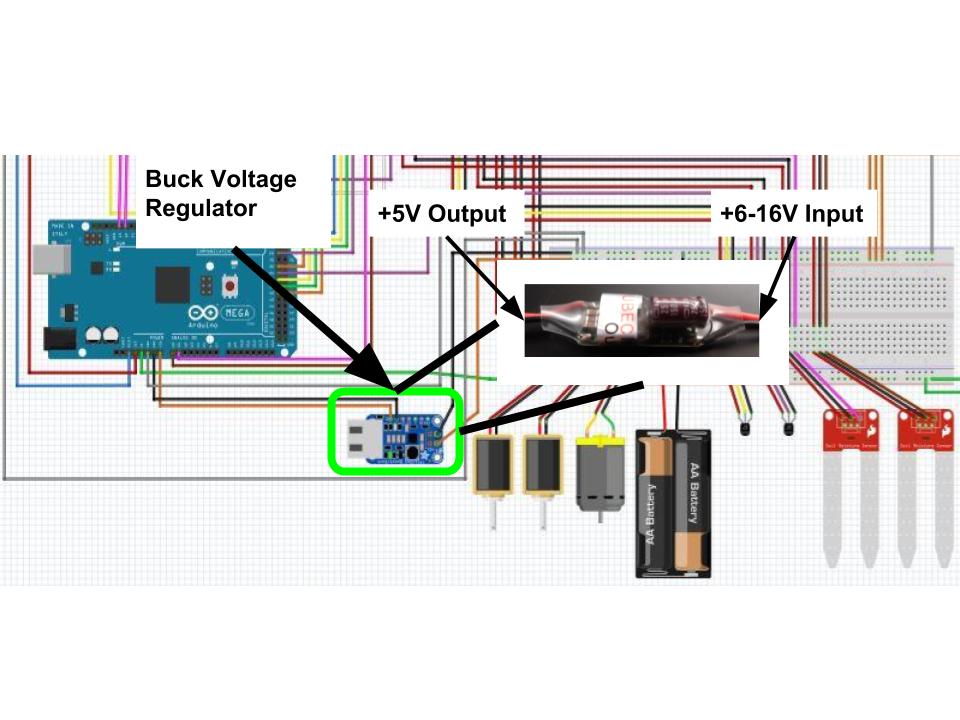
FOR FURTHER ISSUES SEE THE DIAGNOSTICS & TROUBLE SHOOTING SECTION

# Diagnostics & Troubleshooting

*WARNING: TURN OFF POWER SUPPLY FIRST BY UNPLUGING THE BATTERY’S POWER CORD*

Make sure you have any spare parts you may need on hand from the maintenance kit.

1. **Troubleshooting the voltage regulator, the toggle switch, and the microcontroller.**
2. Testing the microcontroller:
   1. Disconnect the battery from the microcontroller
   2. Connect a compatible USB cable to the microcontroller and a computer.
      1. A green light will illuminate on the microcontroller, indicating that the microcontroller is on.
      2. If the microcontroller has short-circuited or “died” the green light does not turn on.
      3. If the green light does not turn on, replace the microcontroller and re-upload the irrigation code.
3. Testing the buck voltage-regulator:
   1. Disconnect the buck voltage-regulator from the irrigation controller circuit as shown in Figure 1.



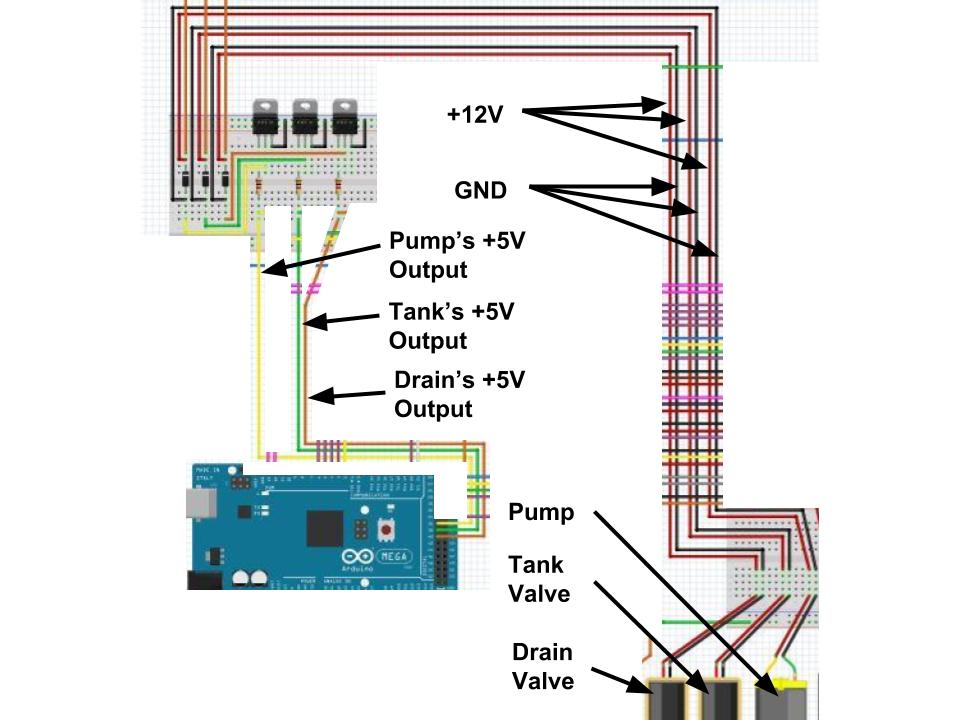
*Figure 1: Location of the Buck Voltage Regulator in the circuit.*

* 1. Apply 14 volts to the regulator by connecting it to a 14-volt power supply (or a charged 12V battery), and check if the regulator turns on. This can be done using a spare two pin plug to connect a voltage generator to the controller.
     1. Use a digital multimeter or similar device to measure the output voltage, which should be around 5 to 5.22V. This would be the part of the UBEC that connects to the Arduino.
     2. If voltage is not within that range, the regulator may not power or overpower the board therefore the regulator needs to be replaced.

1. **Troubleshooting the valves and the pump.**

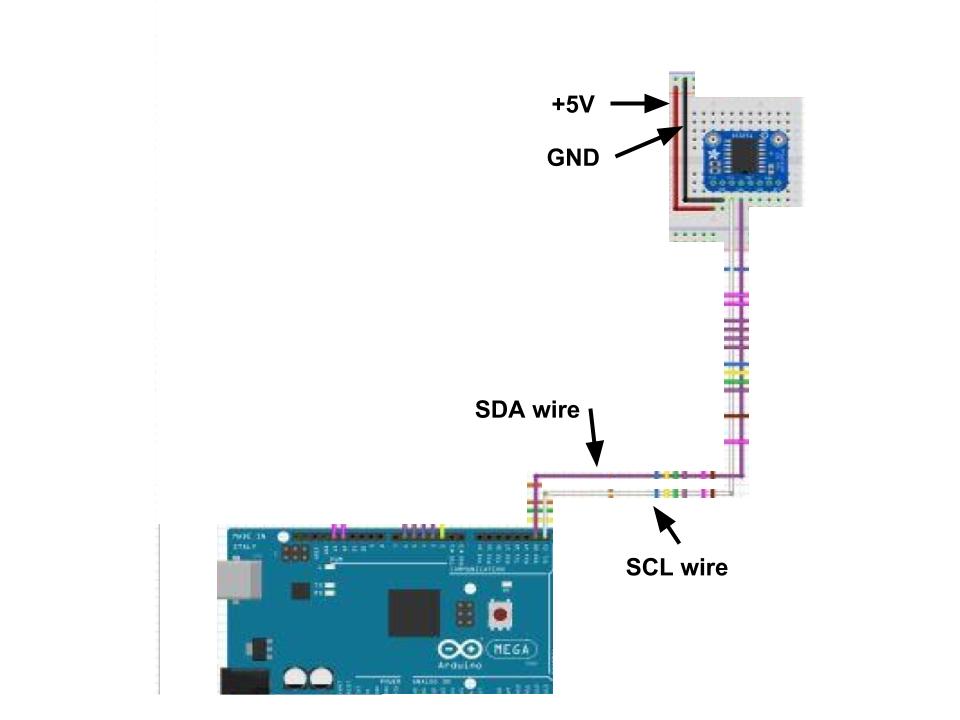
NOTE: This trouble shooting process can be used for the circulator pump as well.

1. Inspect the circuitry from the digital output pin from the microcontroller that activates that valve (or pump) as shown in Figure 2, to the leads of the valve (or pump) as shown in Figure 3.



*Figure 2: The circuit from the microcontroller to the valves and pump.*

1. Check if there are any breaks in the circuit that control the valve (or pump), including solder welds.
2. The wire connections and the resistor can be checked with a multimeter set to measure resistance.
   1. When measuring the wires, a beeping noise should be heard from the multimeter, indicating that there is a connection.
   2. If not, then replace the wires.
   3. When measuring the resistor, a resistance reading should display on the screen, indicating that there is a connection. NOTE: Hard contact is needed to read the resistor.
   4. If not, then replace the resistor.
3. Retest the circuit by turning on the valve (or pump) it controls.
4. If it still does not turn on, test the transistor.
5. The transistor can be checked with a multimeter, set to measure hFE readings.
   1. Reference an unused and functioning transistor to compare hFE readings to the existing transistor in the circuit, by touching the leads.
      1. If readings are the same or similar, the existing transistor works.
      2. If not, then replace the transistor.
6. Retest the circuit by turning on the valve (or pump) it controls.
7. If it still does not turn on, test the diode.
8. The diode can be checked with a multimeter, set to measure the diode’s voltage.
   1. Reference an unused and functioning diode to measure the voltage to compare to the existing diode in the circuit, by connecting the red test lead to the anode and the black test lead to the cathode of the diode.
      1. If the readings are the same or similar, the existing diode works.
      2. If not, then replace the diode.
9. Retest the circuit by turning on the valve (or pump) it controls.
10. If it still does not turn on, even though the circuit is said to be fine, replace and rebuild that valve (or pump) circuit, with a new resistor, transistor, and diode.
11. Retest the circuit by turning on the valve (or pump) it controls.
    1. If it the valve (or pump) still does not work test using one of the spare valves. Replace the valve (or pump) if necessary.
12. **Troubleshooting the DS3231 Real Time Clock (RTC).**
13. Open the ds3231 code sketch file in the Arduino software, found in File>Examples>RTClib. If you cannot find the RTClib folder download the file libraries Wire.h and RTClib.h from the following link: <https://learn.adafruit.com/adafruit-ds3231-precision-rtc-breakout/arduino-usage>
14. Upload the code sketch to the Arduino.
15. Open the Serial Monitor from the Tools tab.
16. If the Serial Monitor displays “Couldn’t find RTC”, unhook the USB Cable used to upload the code and:
    1. Inspect the circuitry from the SCL and SDA pins from the microcontroller to the Real Time Clock’s (RTC) SCL and SDA pins as shown in Figure 4.



*Figure 3: The circuit from the microcontroller to the Real Time Clock (RTC).*

* 1. Check if there are any breaks in the circuit, including solder welds.
  2. The wires connections can be checked with a multimeter, set to measure resistances.
  3. A beeping noise should be heard from the multimeter, indicating that there is a connection.
  4. If not, then replace wires.
  5. Retest the circuit by reconnecting the USB cable and opening the Serial Monitor.

1. If the Serial Monitor displays “RTC lost power, let’s set the time!” unhook USB Cable and:
   1. Inspect the RTC’s 3V Lithium Coin Cell Battery slot, to see if the battery is missing or inserted incorrectly. Also, check if the battery is dead with a multimeter by reading the voltage.
   2. Check the leads connected to the RTC’s Vin and GND pins, and ensure that Vin is connected to the +5V rail and GND is connected to the ground rail.
   3. The wire connections can be checked with a multimeter, set to measure resistances.
   4. A beeping noise should be heard from the multimeter, indicating that there is a connection.
   5. If not, then replace wires.
   6. Retest the circuit by reconnecting the USB cable and open the Serial Monitor.
2. After inspecting the circuity, the date and time should display on the Serial Monitor.
3. If not, then replace the RTC.
4. For more information about the RTC go to the link previously provided above.
5. **Troubleshooting the sensors.**
6. *Temperature sensor.*
   1. Upload the following code to the Arduino microcontroller as shown in Figure 4:

// Include libraries

#include <OneWire.h>

#include <DallasTemperature.h>

// Data wire is plugged into pin 26 on the Arduino

#define ONE\_WIRE\_BUS 26

// Setup a oneWire instance to communicate with any OneWire devices (not just Maxim/Dallas temperature ICs)

OneWire oneWire(ONE\_WIRE\_BUS);

// Pass our oneWire reference to Dallas Temperature

DallasTemperature sensors(&oneWire);

void setup(void)

{

**Serial**.begin(9600); // Begin serial communication

**Serial**.println(“Arduino Digital Temperature // Serial Monitor Version”); // print a message

sensors.begin();

}

void loop(void)

{

// Send the command to get temperatures

sensors.requestTemperatures();

**Serial**.print(“Temperature is: “);

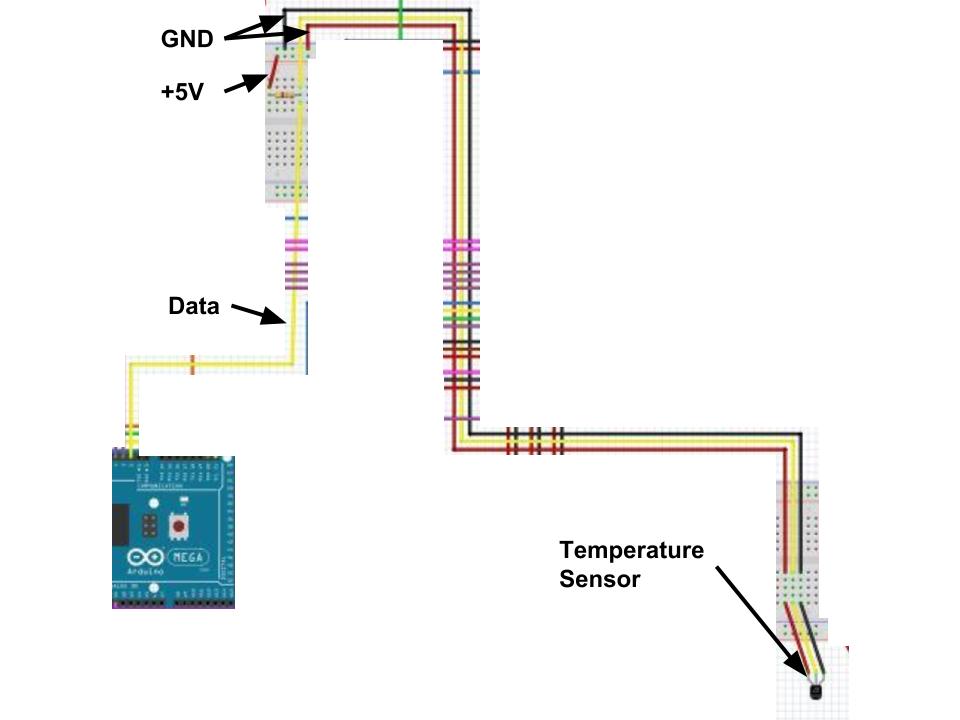
**Serial**.println(1.8\*sensors.getTempCByIndex(0)+32); //Converts C to F and displays temp on serial monitor

delay(1000); // updates every 1000 ms (1 s)

}

*Figure 4: Arduino code sketch for trouble shooting the temperature sensor.*

1. Open the Serial Monitor from the Tools tab.
2. If nothing displays or readings do not change, unhook USB cable, and inspect the circuitry as a shown Figure 5.



*Figure 5: The circuit from the microcontroller to the temperature sensor.*

1. The wire connections and the resistor can be checked with a multimeter set to measure resistance.
   1. When measuring the wires, a beeping noise should be heard from the multimeter, indicating that there is a connection.
   2. If not, then replace the wires.
   3. When measuring the resistor, a resistance reading should display on the screen, indicating that there is a connection. NOTE: Hard contact is needed to read the resistor.
   4. If not, then replace the resistor.
   5. Retest the circuit by reconnecting the USB cable and opening the Serial Monitor.
2. If it still does not display or readings do not change, even though the circuit is said to be fine, replace and rebuild that temperature sensor circuit, with a new resistor, and retest.
3. *Water Level Sensor.*
   1. Upload the following code to the Arduino microcontroller as shown in Figure 6:

const int waterLevelSensor = 34; // water level sensor on digital pin 34

const int waterLevelLED = 35; // water level led on digital pin 35

void setup()

{

pinMode(waterLevelSensor, INPUT);

pinMode(waterLevelLED, OUTPUT);

digitalWrite(waterLevelLED, LOW);

**Serial**.begin(9600);

}

void loop()

{

int isDry = digitalRead(waterLevelSensor); // read sensor

if (isDry == HIGH) // if liquid is not touching the sensor it will read HIGH

{

digitalWrite(waterLevelLED, HIGH); // LED turns on

**Serial**.println(“Refill Tank”); // Serial monitor displays the message, refill tank

}

else if (isDry == LOW) // if liquid is touching the sensor it will read 0V or LOW

{

digitalWrite(waterLevelLED, LOW); // LED stays off

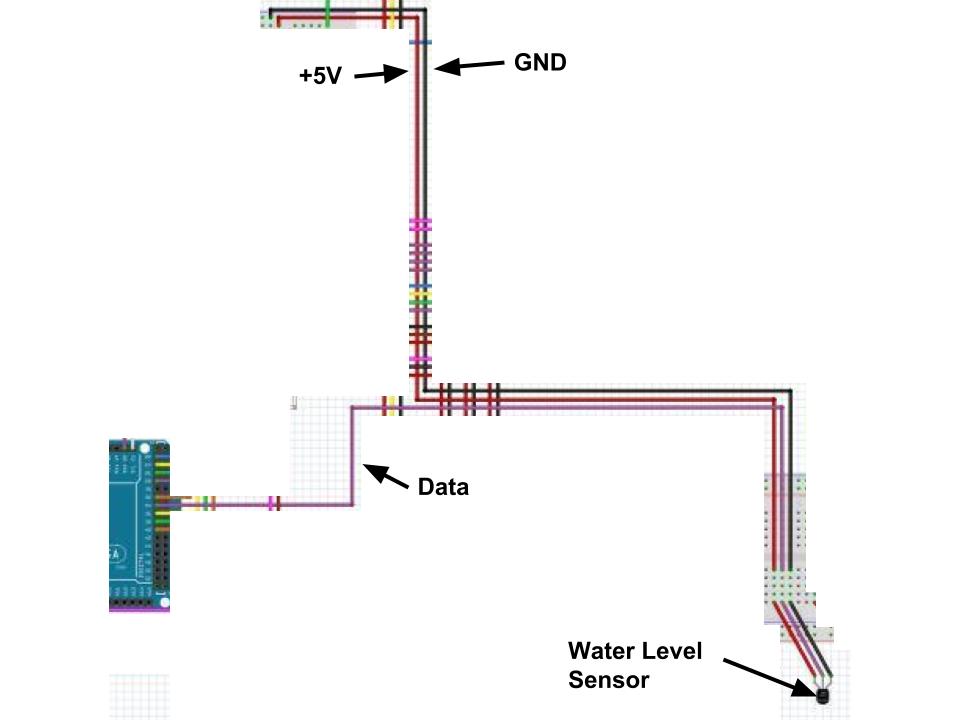
**Serial**.println(“Water Level is Fine”); // Serial monitor displays the message, water level is fine

}

}

*Figure 6: Arduino code sketch for trouble shooting the water level sensor.*

* 1. Open the Serial Monitor from the Tools tab.
  2. The water level sensor can be checked by placing it in and out of the water. (**NOTE**: To have the sensor read as dry, no water should be on the tip of the sensor)
  3. If Serial Monitor displays “Refill Tank” and the orange LED turns on, then the sensor is dry.
  4. If Serial Monitor displays nothing and the orange LED does not turn on, then unhook the USB cable, and inspect the circuitry as shown in Figure 7.



*Figure 7: The circuit from the microcontroller to the water level sensor.*

* + 1. The wire connections can be checked with a multimeter, set to measure resistances.
    2. A resistance reading should display on the screen, indicating that there is a connection.
    3. If not, then replace wires.
    4. Retest the circuit by reconnecting the USB cable and open the Serial Monitor.
    5. If nothing displays, even though the circuit is said to be fine, replace and rebuild that water level sensor circuit, and retest.
    6. If nothing displays, replace the sensor.
  1. If the Serial Monitor is still displaying “Refill Water” and the orange LED remains on after placing the sensor into water then unhook the USB cable and inspect the circuitry.
     1. The wire connections can be checked with a multimeter, set to measure resistances.
     2. A beeping noise should be heard from the multimeter, indicating that there is a connection.
     3. If not, then replace wires.
     4. Retest the circuit by reconnecting the USB cable and open the Serial Monitor.
     5. If Serial Monitor still does not display “Water Level Fine” and the orange LED still does not turn off when placed in water even though the circuit is said to be fine, replace and rebuild that water level sensor circuit, and retest.
     6. If still unchanged, then replace the sensor.

1. *Soil Moisture Sensors.*
   1. Upload the following code to the Arduino microcontroller as shown in Figure 8:

// Soil moisture sensors

const int VH400 = 0; // Analog pin 0

const int EC5 = 1; // Analog pin 1

void setup()

{

pinMode(VH400, INPUT);

pinMode(EC5, INPUT);

**Serial**.begin(9600);

}

void loop()

{

// The following reads the voltage for the VH400 and converts it to Volumetric Water Content

float VH400\_integer\_reading = analogRead(VH400) \* 0.0049;

float VH400\_slope = 0.2007;

float VH400\_intercept = 0.0307;

float VH400\_readings = VH400\_slope \* VH400\_integer\_reading + VH400\_intercept;

// The following reads the voltage for the EC5 and converts it to VWC

float EC5\_integer\_reading = analogRead(EC5) \* 0.0049;

float EC5\_slope = 0.9932;

float EC5\_intercept = -0.3786;

float EC5\_readings = EC5\_slope \* EC5\_integer\_reading + EC5\_intercept;

// Displays the VWC percentage on the serial monitor

**Serial**.print(“VH400 :”);

**Serial**.println(VH400\_readings \* 100);

**Serial**.print(“EC5: “);

**Serial**.println(EC5\_readings \* 100);

**Serial**.println(“”);

delay(1000);

}

*Figure 8: Arduino code sketch for trouble shooting the soil moisture sensors.*

* 1. Open the Serial Monitor from the Tools tab.
  2. If a moisture sensor reading does not display or change, then unhook USB cable, and inspect the circuitry as shown in Figure 9.



*Figure 9: The circuit from the microcontroller to the moisture sensors.*

* + 1. The wire connections can be checked with a multimeter, set to measure resistances.
    2. A beeping noise should be heard from the multimeter, indicating that there is a connection.
    3. If not, then replace wires.
    4. Retest the circuit by reconnecting the USB cable and open the Serial Monitor.
    5. If a sensor reading, still does not display or change, even though the circuit is said to be fine, replace and rebuild that sensor circuit, and retest.
    6. If still unchanged, then replace the sensor.

1. **Troubleshooting the LEDs and buttons.**
   1. Upload the following code to the Arduino as shown in Figure 10:

// these are the leds, activating the pump and valves also activates the LED so only one pin is needed

const int waterLevelLED = 35;

const int pumpCirculator = 40;

const int tankValve = 41;

const int drainValve = 42;

// the button pins

const int buttonPin\_A\_Y = 5;

const int buttonPin\_S\_N = 4;

const int buttonPin\_next = 3;

const int buttonPin\_mode = 2;

void setup() {

// button set up

pinMode(buttonPin\_A\_Y, INPUT);

pinMode(buttonPin\_S\_N, INPUT);

pinMode(buttonPin\_next, INPUT);

pinMode(buttonPin\_mode, INPUT);

// leds

pinMode(waterLevelLED, OUTPUT);

pinMode(pumpCirculator, OUTPUT);

pinMode(tankValve, OUTPUT);

pinMode(drainValve, OUTPUT);

**Serial**.begin(9600);

}

void loop() {

if (buttonPin\_A\_Y == HIGH) { // Water Level LED

digitalWrite(waterLevelLED, HIGH);

**Serial**.println(“Water Level LED is ON”);

}

else {

digitalWrite(waterLevelLED, LOW);

**Serial**.println(“Water Level LED is OFF”);

}

if (buttonPin\_S\_N == HIGH) { // Pump circulator LED

digitalWrite(pumpCirculator, HIGH);

**Serial**.println(“Pump circulator LED is ON”);

}

else {

digitalWrite(pumpCirculator, LOW);

**Serial**.println(“Pump circulator LED is OFF”);

}

if (buttonPin\_next == HIGH) { // Tank Valve LED

digitalWrite(tankValve, HIGH);

**Serial**.println(“Tank valve LED is ON”);

}

else {

digitalWrite(tankValve, LOW);

**Serial**.println(“Tank valve LED is OFF”);

}

if (buttonPin\_mode == HIGH) { // Drain Valve LED

digitalWrite(drainValve, HIGH);

**Serial**.println(“Drain valve LED is ON”);

}

else {

digitalWrite(drainValve, LOW);

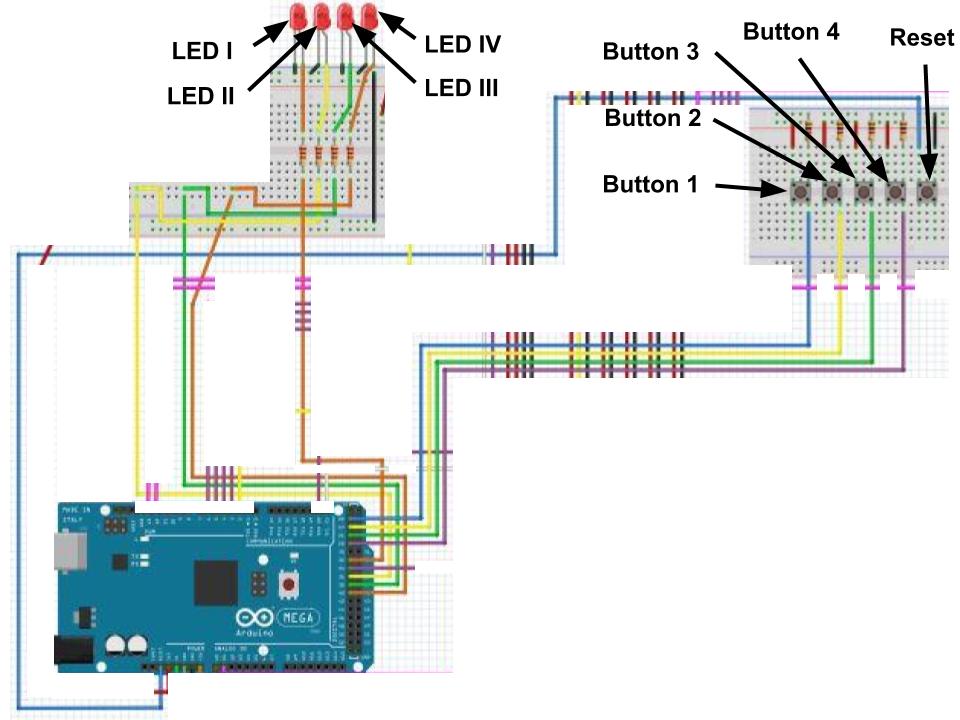
**Serial**.println(“Drain valve LED is OFF”);

}

}

*Figure 10: Arduino code sketch for trouble shooting the LEDs and buttons.*

* 1. Open the Serial Monitor from the Tools tab.
  2. Press the blue button (1), the orange LED (I) should turn on, and the monitor should display “Water Level LED ON”.
  3. If the orange LED does not turn on, then unhook the USB cable, and inspect the circuitry as a shown in Figure 11.



*Figure 11: The circuit from the buttons to the LEDs and Reset.*

* 1. The wire connections and the resistors can be checked with a multimeter set to measure resistance.
     1. When measuring the wires, a beeping noise should be heard from the multimeter, indicating that there is a connection.
     2. If not, then replace the wires.
     3. When measuring the resistor, a resistance reading should display on the screen, indicating that there is a connection. NOTE: Hard contact is needed to read the resistor.
     4. If not, then replace the resistors.
     5. Retest the circuit by reconnecting the USB cable, open the Serial Monitor, and then press the blue button.
  2. If it still does not turn on, test the blue button.
     1. Test the vertical left side of the upper and lower leads by setting the multimeter to read the resistances.
     2. A beeping noise should be heard from the multimeter, indicating that there is a connection.
     3. The same can be done to the left side of the button.
     4. Then measure the resistance horizontally across the the blue button, upper and lower; there should be no beeping noise heard from the multimeter.
     5. If there is a beeping noise then the button has short-circuited and needs to be replaced.
     6. Retest the circuit by reconnecting the USB cable, open the Serial Monitor, and then press the blue button.
  3. If it still does not turn on, test the LED.
     1. The LED can be checked with a multimeter, set to measure resistances.
     2. A resistance reading should display, indicating that there is a connection.
     3. If not, then replace the LED.
     4. Retest the circuit by reconnecting the USB cable, open the Serial Monitor, and then press the blue button.
     5. If it still does not turn on, even though the circuit is said to be fine, replace and rebuild that circuit from the blue button (1) to the orange LED (I), with a new resistor, button, LED, and retest.
  4. Procedure E. can be done for the other LEDs (blue (II), red (III), and green (IV)) and buttons (red (2), green (3), and yellow (4)) that corresponds to the pump circulator, tank, and drain valves, respectively.
  5. If the Reset button does not reset the Arduino microcontroller the wires and the button can be checked by using the same steps from E. e. i – ii and E. f. i – vi. NOTE: There is no code required for the Reset button.

1. **Troubleshooting the LCD screen.**
   1. Upload the following code to the Arduino microcontroller as shown in Figure 12:

#include <LiquidCrystal.h> // library code

const int rs = 13, en = 12, d4 = 11, d5 = 10, d6 = 9, d7 = 8; // defining what pins we are using

LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {

lcd.begin(16, 2); // 16 columns with 2 rows

lcd.print(“hello world”); // displays the message “hello world” on the LCD

}

void loop() {

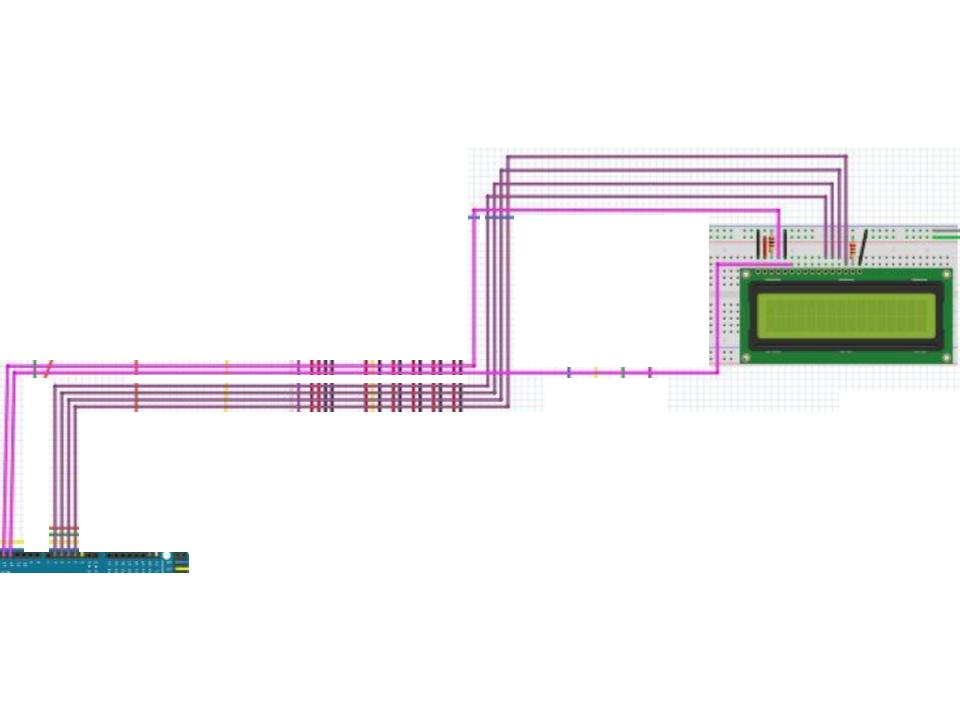
lcd.setCursor(0,1); // column 0, row 1 is the first column in the second row (below “hello world”)

lcd.print(millis()/1000); // print the number of seconds since reset

}

*Figure 12: Arduino code sketch for trouble shooting the LCD screen.*

* 1. If the LCD screen does not display “hello world”, then unhook USB cable, and inspect the circuitry as shown in Figure 13.



*Figure 13: The circuit from the microcontroller to the LDC screen.*

* 1. The wire connections and the resistors can be checked with a multimeter, set to measure resistance.
     1. When measuring the wires, a beeping noise should be heard from the multimeter, indicating that there is a connection.
     2. If not, then replace the wires.
     3. When measuring the resistor, a resistance reading should display on the screen, indicating that there is a connection. NOTE: Hard contact is needed to read the resistor.
     4. If not, then replace the resistors.
     5. Retest the circuit by reconnecting the USB cable.
     6. If it still does not display “hello world”, even though the circuit is said to be fine, replace and rebuild that circuit, with new resistors, and LCD, and retest.

# Code Documentation & Explanation

There are several libraries one should download before modifying this code or working on it.

* The Arduino-Temperature-Control-Library: <https://github.com/milesburton/Arduino-Temperature-Control-Library>
* The OneWire Library: <https://www.pjrc.com/teensy/td_libs_OneWire.html>
* The RTClib library: <https://github.com/adafruit/RTClib>

The Arduino Temperature Control Library and the OneWire library are necessary to use the DS18B20 temperature sensor. The temperature control library allows us to get the temperature and the OneWire allows us to wire the temperature sensor using only one wire (if you look at the schematic you'll see the temperature sensor is wired with its power and ground both connected to the ground rail. The data pin is both receiving power and communicating with the Arduino). RTClib is the library for the real-time clock used in this controller, the DS3231.

**Table of Global Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Name** | **Function** | **Value** |
| Constant integer | VH400 | Pin for the VH400 sensor | Analog 0 |
| Constant integer | EC5 | Pin for the EC5 sensor | Analog 1 |
| Constant integer | waterLevelSensor | Pin for the water level sensor | Digital 34 |
| Constant integer | waterLevelLED | Pin for the LED associated with the water level sensor | Digital 35 |
| Constant integer | pumpCirculator | Pin for controlling the pump circulator and the LED associated with it | Digital 40 |
| Constant integer | tankValve | Pin for controlling the tank valve and the LED associated with it | Digital 41 |
| Constant integer | drainValve | Pin for controlling the drain valve and the LED associated with it | Digital 42 |
| Constant integer | buttonPin\_A\_Y | Pin for the addition/yes button (blue on the controller) | Digital 5 |
| Constant integer | buttonPin\_S\_N | Pin for the subtraction/no button (red on the controller) | Digital 4 |
| Constant integer | buttonPin\_next | Pin for the next button (green on the controller) | Digital 3 |
| Constant integer | buttonPin\_mode | Pin for the mode button (yellow on the controller) | Digital 2 |
| Constant integer | rs | Pin connecting to RS on the LCD1602 module | Digital 13 |
| Constant integer | en | Pin connecting to E on the LCD1602 module | Digital 12 |
| Constant integer | d4 | Pin connecting to D4 on the LCD1602 module | Digital 11 |
| Constant integer | d5 | Pin connecting to D5 on the LCD1602 module | Digital 10 |
| Constant integer | d6 | Pin connecting to D6 on the LCD1602 module | Digital 9 |
| Constant integer | d7 | Pin connecting to D7 on the LCD1602 module | Digital 8 |
| Integer | VWC\_Upper | Stores the upper VWC limit parameter as defined by the user | Default is 44% |
| Integer | VWC\_Lower | Stores the lower VWC limit parameter as defined by the user | Default is 37% |
| Integer | tempUpperLimit | Stores the upper temperature limit (F) as defined by the user | Default is 90F |
| Integer | tempLowerLimit | Stores the lower temperature limit (F) as defined by the user | Default is 40F |
| Integer | backup\_VWC\_Upper | Stores whatever VWC\_Upper value was saved last time so that accidental changes aren’t permanent. | Default is 44% |
| Integer | backup\_VWC\_Lower | Stores whatever VWC\_Lower value was saved last time so that accidental changes aren’t permanent. | Default is 37% |
| Integer | backup\_tempUpperLimit | Stores whatever tempUpperLimit value was saved last time so that accidental changes aren’t permanent. | Default is 90F |
| Integer | backup\_tempLowerLimit | Stores whatever tempLowerLimit value was saved last time so that accidental changes aren’t permanent. | Default is 40F |
| Integer | buttonState\_A\_Y | Used for the addition/yes button code. | Default: 0 |
| Integer | buttonState\_S\_N | Used for the addition/yes button code. | Default: 0 |
| Integer | buttonState\_next | Used for the addition/yes button code. | Default: 0 |
| Integer | buttonState\_mode | Used for the addition/yes button code. | Default: 0 |
| Integer | lastButtonState\_A\_Y | Used for the addition/yes button code. | Default: 0 |
| Integer | lastButtonState\_S\_N | Used for the addition/yes button code. | Default: 0 |
| Integer | lastButtonState\_next | Used for the addition/yes button code. | Default: 0 |
| Integer | lastButtonState\_mode | Used for the addition/yes button code. | Default: 0 |
| Integer | modeState | Used for switching modes and keeping track of what mode you’re in. | 0 |
| Integer | parameterState | Used for moving onto the next parameter in Parameter Mode. | 0 |
| Integer | sensorState | Used for moving onto the next sensor in Sensor Read Mode. | 0 |
| Integer | autoState | Used for moving onto the next step in Auto Mode. | 0 |
| Integer | parameterUnit | Used for the DisplayParameters function which is used in Parameter Mode. | Default: % |
| Integer | manualControlPump | Used for controlling the pump in manual mode. Lets the button act like a light switch. | Default: 0 |
| Integer | manualControlTank | Used for controlling the tank valve in manual mode. Lets the button act like a light switch. | Default: 0 |
| Integer | manualControlDrain | Used for controlling the drain valve in manual mode. Lets the button act like a light switch. | Default: 0 |

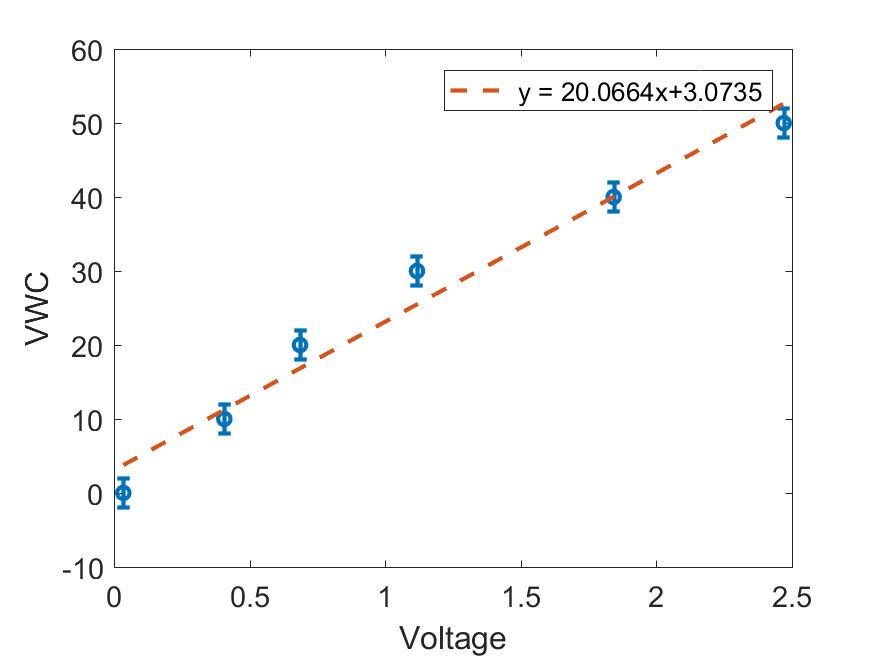
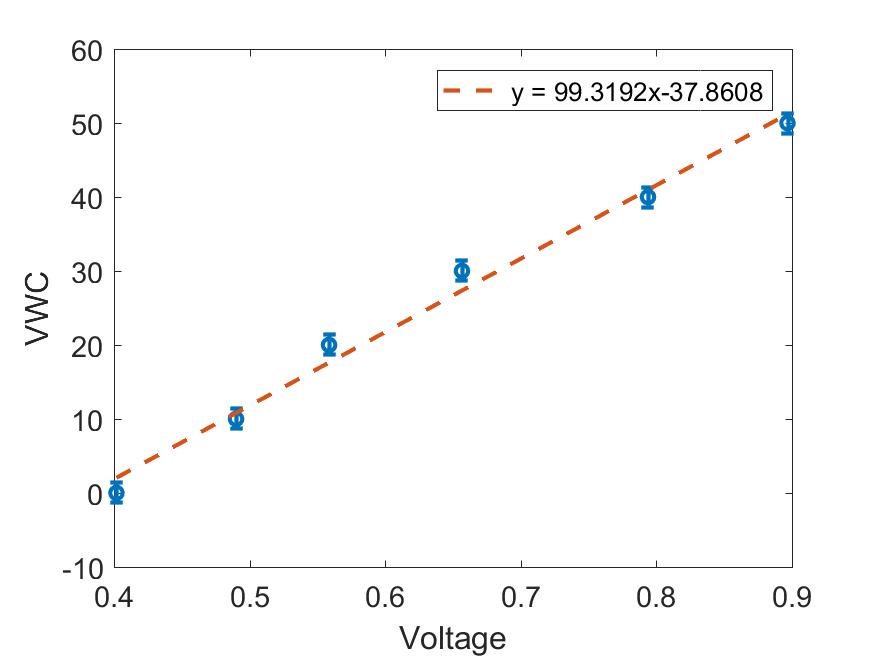
**Program explanation:**

The controller code heavily relies on modeState which manages what mode the user is in. Within Parameter Mode, Sensor Read Mode, and Auto Mode there are additional variables controlling what state those modes are in.

* The Arduino checks the state of the mode button every time it loops. If the button is pushed then the program adds one to the modeState variable (goes to the next mode).
* Standby Mode (modeState == 0): this mode's only job is to check the time and run auto mode every hour on the hour. We do this by checking the current time and seeing if the seconds and minutes are equal to zero. If they are then it means the time looks something like XX:00:00. When this does happen modeState is set equal to 5 which is Auto Mode.
* Parameter Mode (modeState == 1): this mode allows the user to change and set parameters. This is where parameterState is used.
  + Start Screen (parameterState == 0): this screen just tells the user to push the next button to continue through parameter mode. The next button code is identical to the mode button code.
  + Setting the Upper VWC Parameter (parameterState == 1): this screen displays the UpperVWC variable and allows the user to change it using the add and subtract buttons.
  + Setting the Lower VWC Parameter (parameterState == 2): same as previous screen but with the LowerVWC now
  + Setting the Temperature Upper Limit Parameter (parameterState == 3): same as previous screen but with tempUpperLimit
  + Setting the Temperature Lower Limit Parameter (parameterState == 4): same as previous screen but with tempLowerLimit
  + Displaying the changed parameters (parameterState == 5): just shows all four parameters so the user can double check things
  + Asking to save parameters screen (parameterState == 6): asks the user if they would like to save these new parameters. If yes it moves on, if no it uses the backup variables to revert any changes made.
  + Saving to EEPROM (parameterState == 7): this saves the new parameters to EEPROM so they can be used next time the controller gets turned on and changes the backup variables to match the new parameters.
  + Looping back to the start screen (parameterState == 8): this just goes back to the start screen.
* Manual Mode (modeState == 2): this mode allows the user to manual control the pump circulator, tank valve, and drain valve. This is where those button state variables and the manualControl variables come in handy. Before, users had to hold down the button if they wanted the pump to stay on or the valves to remain open. Now they can simply push the button once to turn on and push again to turn off. The tank valve is coded to automatically open after closing the tank valve.
* Sensor Read Mode (modeState == 3): this mode lets the user read all the sensors and it also uses the sensorState variable
  + Start Screen (sensorState == 0): tells the user to push next to continue through sensor read mode
  + Water Level Sensor Screen (sensorState == 1): this screen reads the water level sensor pin. If it reads HIGH that means there is no liquid touching the sensor so it turns on the water level LED to indicate that the tank needs to be refilled. If it reads LOW then liquid is touching the sensor and the light remains off showing that the tank does not need to be refilled.
  + Temperature Sensor Screen (sensorState == 1): this screen reads the temperature and displays it in degrees Fahrenheit.
  + Moisture Levels Screen: this screen shows the VWC readouts from our two sensors. S1 is the VH400 and S2 is the EC5.
  + Looping back to the start screen (sensorState == 4): this makes sure the water level LED has turned off and returns the user to the start screen of Sensor Read Mode
* Returning to Standby Mode (modeState == 4): The water level LED is once again turned off and all the program variables are reset. This is so that when the user switches modes again they end up on the mode's start screen instead of finding themselves on one of the many sub-screens.
* Auto Mode (modeState == 5): this is the focus of the program where many checks are ran before watering the garden. autoState is used here. There also aren't any screen changes.
  + Checking the water level (autoState == 0): just reads whether the water level sensor is HIGH (no water) or LOW (there is water) like in sensor read mode. If there is water it moves onto the next step. If there is no water it turns on the refill water LED and goes back to standby mode.
  + Checking the water temperature (autoState == 1): checks to see if the current temperature is within the established range of tempLowerLimit to tempUpperLimit. If it is then it moves onto the next step. If it is too cold it turns on the pump circulator for 5 seconds and then checks again in an hour (back to standby mode). If it is too hot it just checks again in an hour (back to standby mode).
  + Checking the soil moisture sensors (autoState == 2): this step reads the analog values from the VH400 and the EC5, converts them to values we can use, and averages both. If the averageSoilSensorVal is below VWC\_Lower the garden should be watered. If it is between the lower limit and upper limit the moisture is in the green zone and the garden should not be watered (back to standby mode). If it is above the upper limit like if it rains then the garden should not be watered (back to standby mode).
  + Closing the valve step (autoState == 3): this step checks VWC of the soil again by reading the soil moisture sensors. Once it passes the upper limit the tank valve is closed and it moves onto the last step.
  + Opening the drain valve step (autoState == 4): this step opens the drain valve for 1 minute (60000 milliseconds) and then closes it before going back to standby mode.

How Volumetric Water Content is Calculated:

The code used for reading the temperature sensors is based off these two graphs:

VH400 3.3V Calibration Curve EC5 3.3V Calibration Curve

Compared to 5V, powering these sensors at 3.3V resulted in more accurate readings. This is why the VH400\_slope is 0.2007 (20.0664 above) and the EC5\_slope is 0.9932 (99.3192 above). The same is true for the VH400\_intercept (0.0307) and the EC5\_intercept (-0.3786) variables.