**Documentation for Fish Tank Project**

**ME 121 - 001**

**MW Section, 10:15 AM – 12:05 PM**

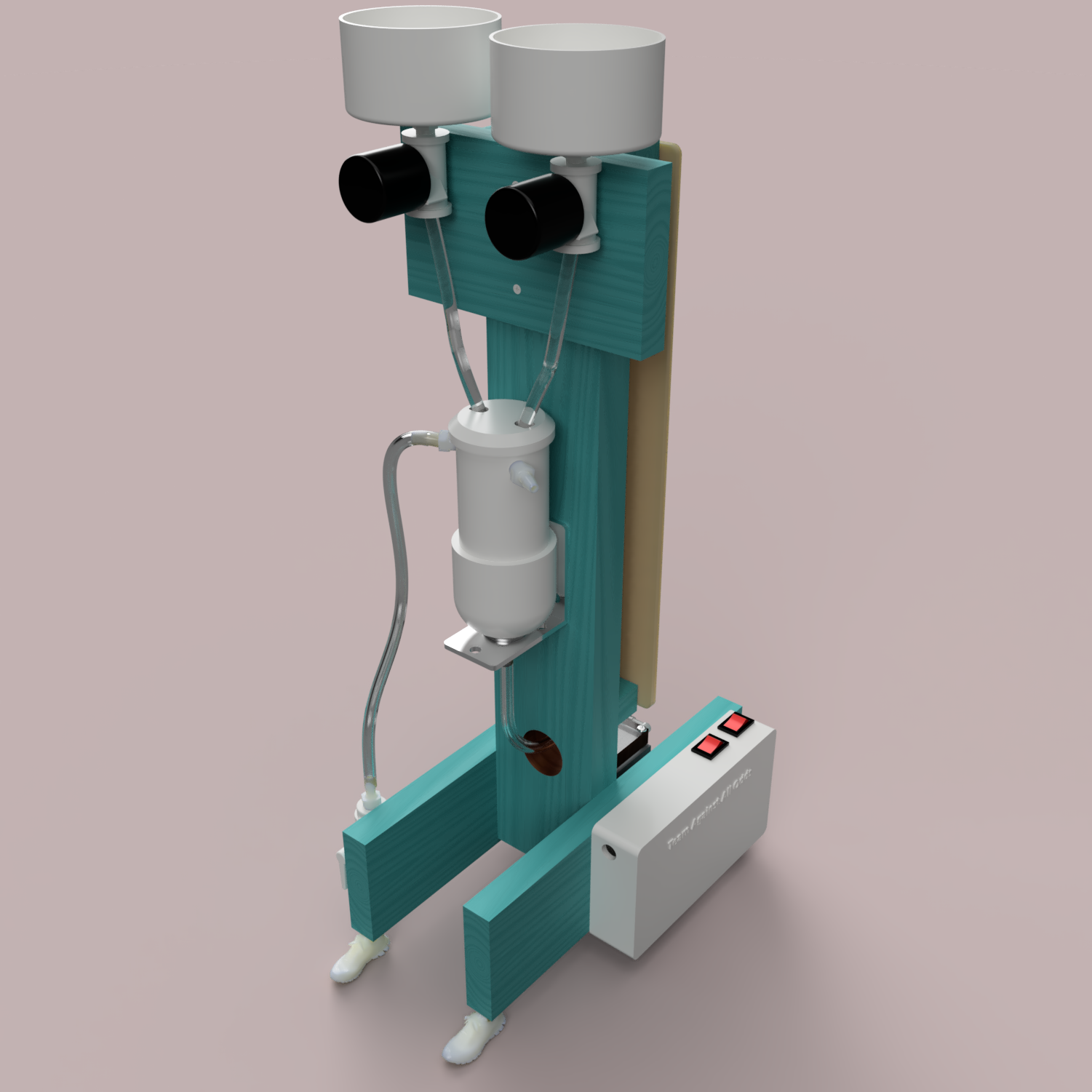
**March 20th, 2020**

**Group Members**

Emily Beeler, Dylan Duncan, Nahom Kiros, Juniper Lund

**Introduction:**

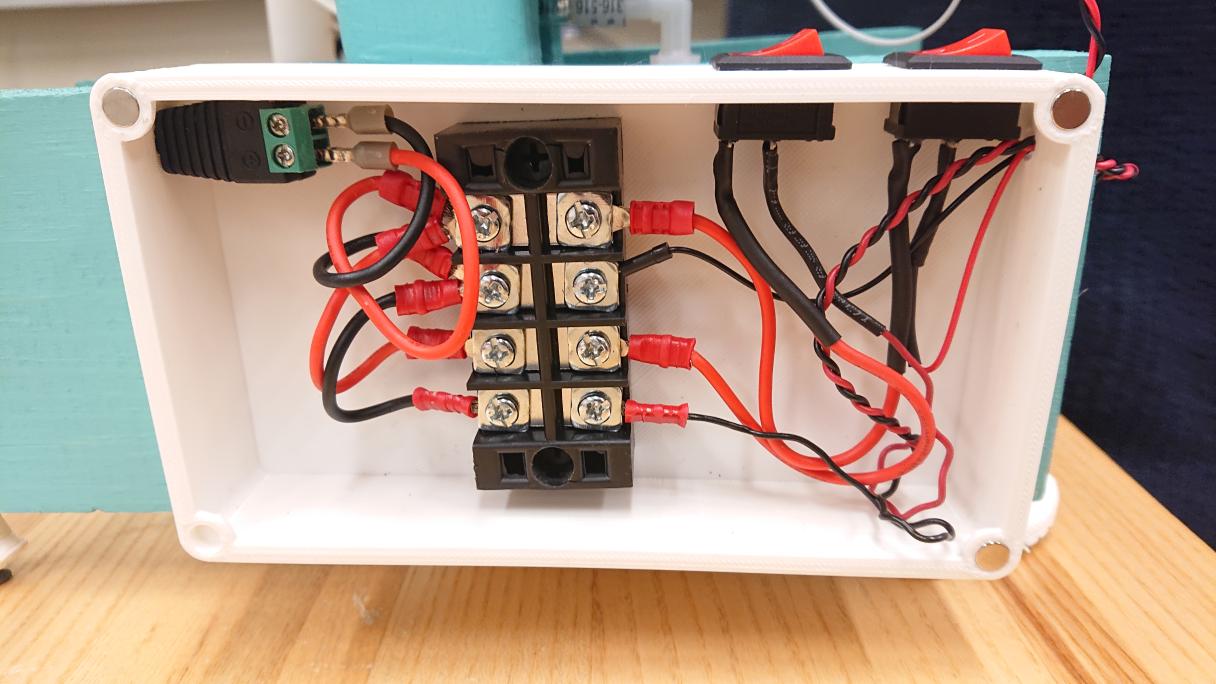
The goal of this team against all odds was to create a “fish tank” (*fig 1*) that would self regulate the temperature and salinity through an Arduino operated array of sensors, solenoid valves, and heater. The chosen volume for the “fish tank” was greatly reduced from that of a standard tank (*fig 3*) so as to speed up the response of the system which reduces the overall wait time during testing. An LCD (*fig 2*) displays information about the status of the tank including the current temperature and current salinity as well as their set points. The upper and lower bounds of both the temperature and salinity are displayed, as well as the status of if the heater is on or off. When the values read from the salinity sensor are below the readings that were calibrated with DI water, it can show as negative like pictured.



**Figure 1.** *The Fish Tank* **Figure 2.** *The Liquid Crystal Display*



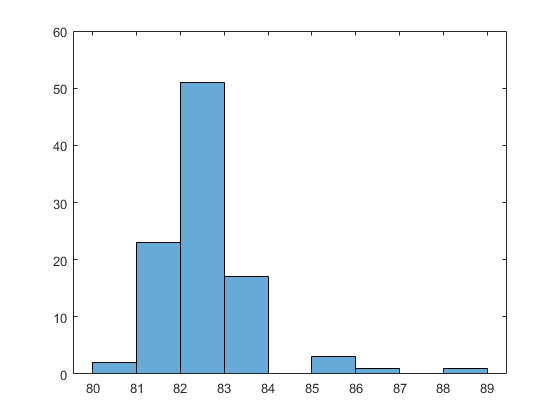
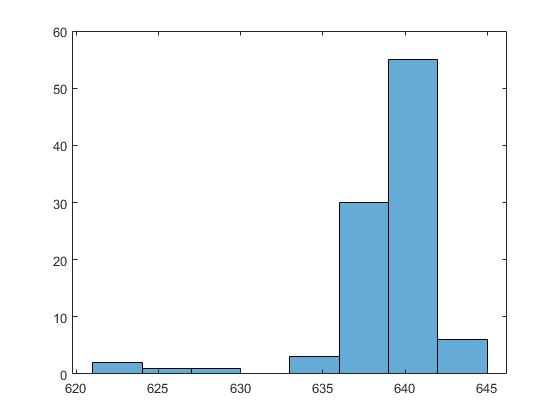
**Figure 3.** *The Fish Tank Flow Loop*

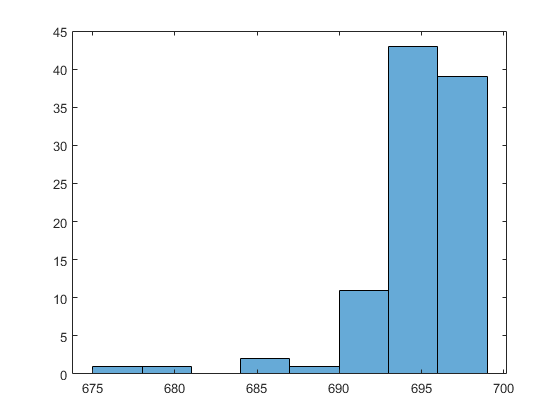
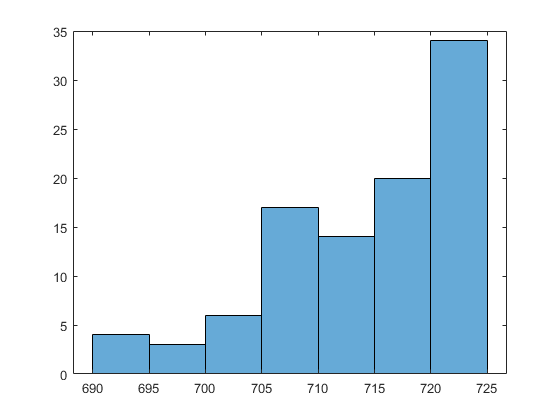


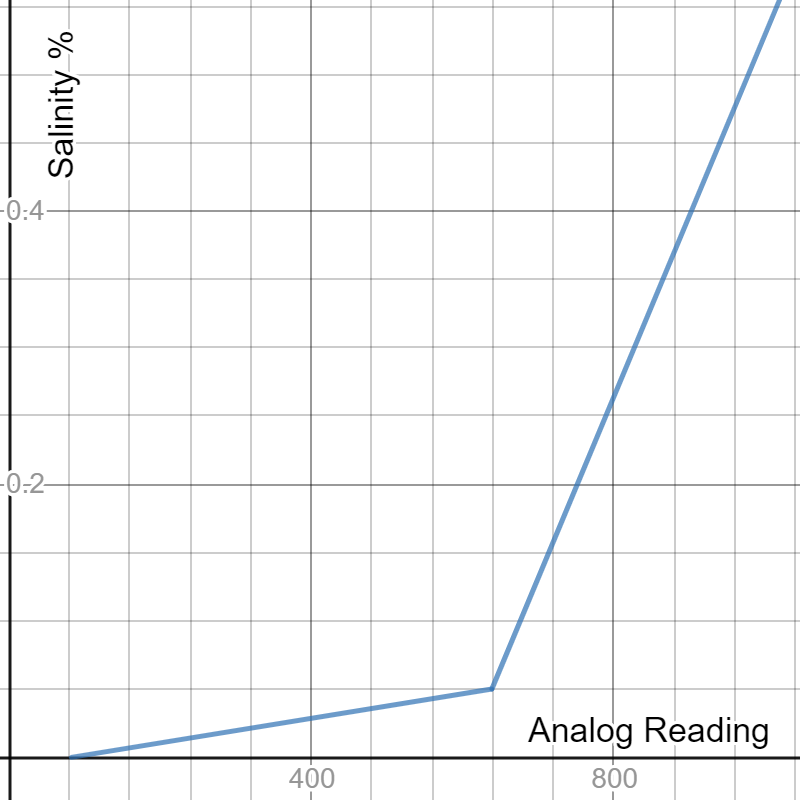
**Figure 4.** *Wiring for the system’s power*

**Fabrication and Control Implementation:**

The salinity sensor was calibrated by running four salinity reference samples through the system, collecting and averaging corresponding analog readings from the voltage divider, and using those average readings to generate a calibration curve. The first step in the process was gathering data from the four salinity samples. The salinity percentages of the solutions tested were: 0.00%, 0.05%, 0.10%, 0.15%. For each sample, the system was flushed with deionized water, then the sample solution was run through the system. An arduino program was created to analyze the data, in which 15 direct analog readings from the voltage divider are averaged and displayed in the serial monitor. The averaged readings displayed in the serial monitor were then recorded in larger batches ( ~ 100 entries) , and used as data for the salinity reference samples (figures 5-9). An average of the data entries was plotted for each salinity reference point, and this plot was used to create a piecewise function (figure 10), where salinity percent operates as a function of the analog readings output by the voltage divider. This function allows the system to report a salinity percentage value for any analog reading.

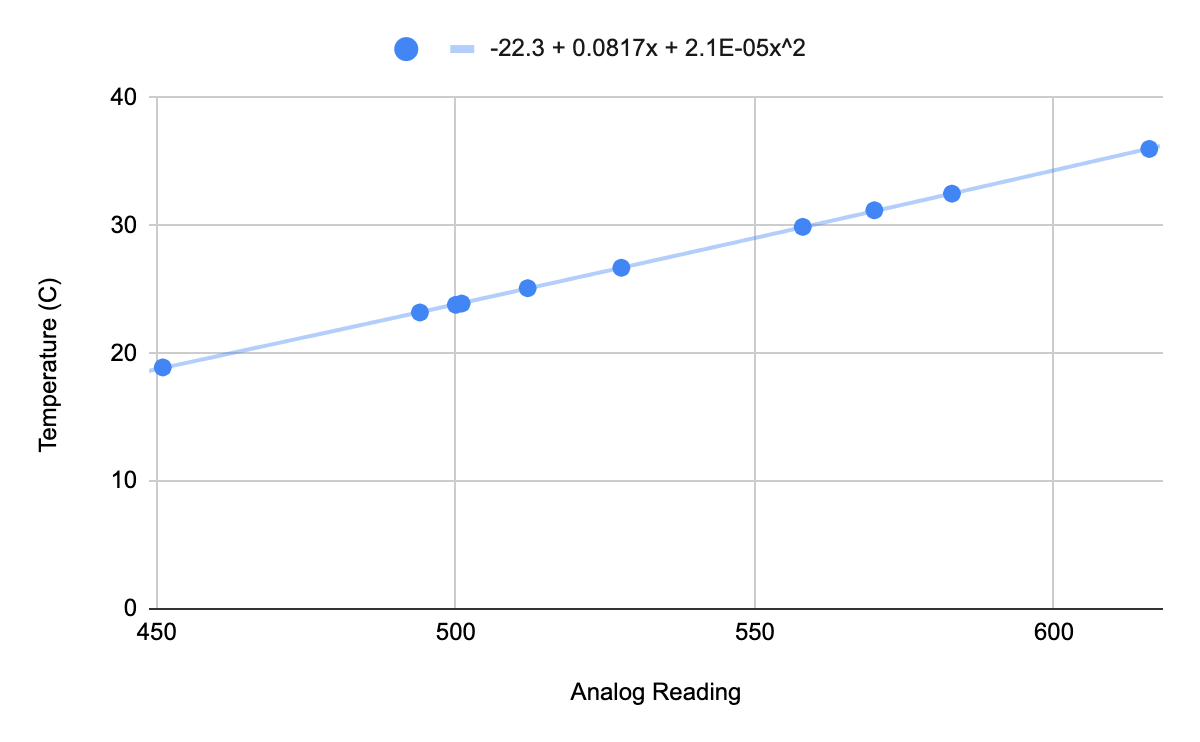
**Figure 5.** *Histogram of analog readings for 0.00% salinity* **Figure 6.** *Histogram of analog readings for 0.05% salinity*

**Figure 7.** *Histogram of analog readings for 0.10% salinity* **Figure 8.** *Histogram of analog readings for 0.15% salinity*



**Figure 9.** *Plot of calibration piecewise function*

Calibration of the thermistor was very similar to the calibration of the salinity sensor. In this process, the reference samples used were water samples of a known temperature. The thermistor was submerged in the various water samples at random, and the analog readings were reported to the serial monitor. The analog readings output by the thermistor for each reference sample were again averaged and plotted, this time with respect to the recorded temperature of the sample. The plot was used to create another calibration curve (figure 10), this curve representing a linear relationship where temperature (in degrees Celsius) is a function of the analog output of the thermistor.

**Figure 1.** *Plot of thermistor data and calibration curve*

Separate control algorithms were derived from the measurements taken for both the thermistor and the salinity sensors. When writing our control algorithms, parameters (tables 1, 2) were used to keep the system stable and reduce potential errors. Parameters used in the salinity control system include gain, which determines the ratio of the output response to the error signal, and an overflow factor, which accounts for the water lost to the outflow port before circulating through the system. The temperature control algorithm used the parameter K, which represents the heating rate at maximum power to the heater. The value of K is used to scale the overall duration of the error response, relative to the rate at which the heater is able to release energy. Both control algorithms also implemented a deadband parameter, creating a margin surrounding the setpoint that will not produce an error signal — this serves to reduce oscillations in the system.

| Salinity Control Parameters |  |
| --- | --- |
| Gain (G) | 0.8 |
| Overflow Factor (F) | 0.15 |
| Deadband | 0.028 |

**Table 1.** *Parameters for the salinity control algorithm*

| Temperature Control Parameters |  |
| --- | --- |
| K | 0.506 ℃/s |
| Deadband | 2.76 |

**Table 2.** Parameters for the temperature control algorithm

**Discussion:**

In its final state, the fish tank was able to produce stable readings and respond appropriately to disturbances to the system. Although the final operations were relatively consistent, we did experience several issues throughout the course of the project. Initially, electrical noise would randomly scramble the display of our LCD; we were able to majorly improve this through braiding and shortening the wires on our breadboard. We also had some trouble with the overflow outlet in our tank, where water would not consistently flow out of it; this problem was resolved permanently by replacing the barb and tubing connected to the overflow outlet with slightly larger pieces.

Another issue we faced was with the calculation of our deadband for the salinity control algorithm. The original deadband was calculated using the standard guidelines ) for the largest deviation from our data sets. However, the calculated deadband was very large, which meant that our system required dramatic disturbances to produce a signal for error correction. Not only did this make it difficult to test the operation of the solenoids, but it also meant that the salinity control was less effective. When reevaluating our deadband, we found that issue was not with the deadband calculations, instead the issue was with the data used to compute the deadband. The standard deviation we had used ( 8.4 ) was noticeably larger than the standard deviation of our other data sets ( 1.1 - 3.4 ). This suggests an error in the original data collection, where, likely, we did not allow our system to settle before recording the data. We corrected the error by recomputing the deadband with the second largest deviation from our data sets, and that resolved the issues we were experiencing. If we were to revisit this project however, we would redo the data collection process for our salinity sensor in an attempt to avoid large deviations and, in theory, improve the accuracy of our sensor. Through this process we learned how important the early data collection is to the overall function of our system.

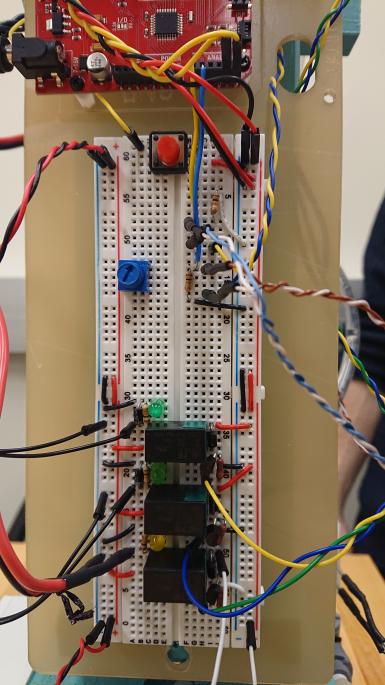
**Conclusion:**

This project enabled us to use both technical concepts and critical problem solving in a collaborative group environment. The calibration process directed a heavy focus on computational work and data collection, whereas the process of testing and implementation provided hands on experience with electrical and mechanical systems. Although we encountered several impediments, our goal was ultimately met, and the system was able to regulate both temperature and salinity within the tank. One suggestion for improving the project would be incorporating more hands on time during class periods, so as to provide more support for students when working through the debugging processes. As for students next term who may be attempting the project, we would suggest they take liberties to embellish and improve their system beyond the basic guidelines.

**Appendix:**

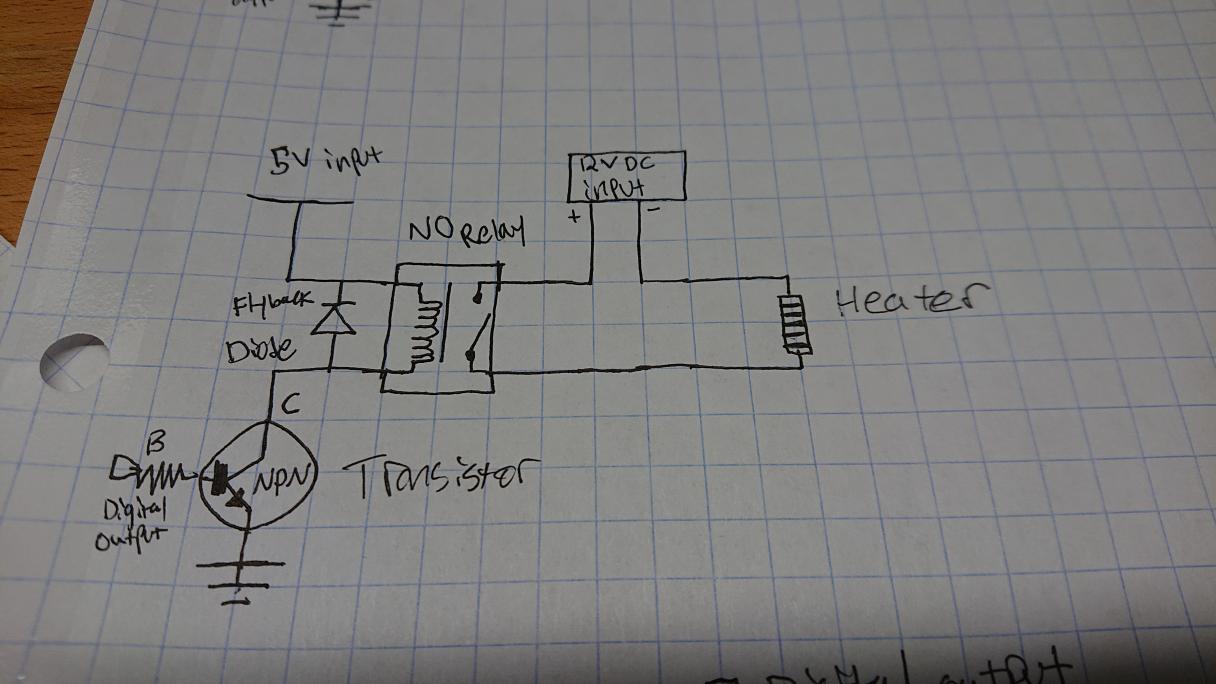
| **Arduino Pin Label** | **Wire Color** | **Purpose** |
| --- | --- | --- |
| 5V | Red | 5V supply for the breadboard |
| GND | Black | Ground for the breadboard |
| A0 | Yellow | Salinity Sensor |
| A1 | Blue | Thermistor Sensor |
| D3 | Blue | Power for thermistor |
| D4 | Yellow | Power for salinity sensor |
| D7 | Blue | Relay pin for fresh water solenoid |
| D8 | Yellow | Relay pin for salty water solenoid |
| D9 | Green | Relay pin for heater |
| D11 | Yellow | Detects button press to start system |

**Table 3.** *Arduino Pinouts*

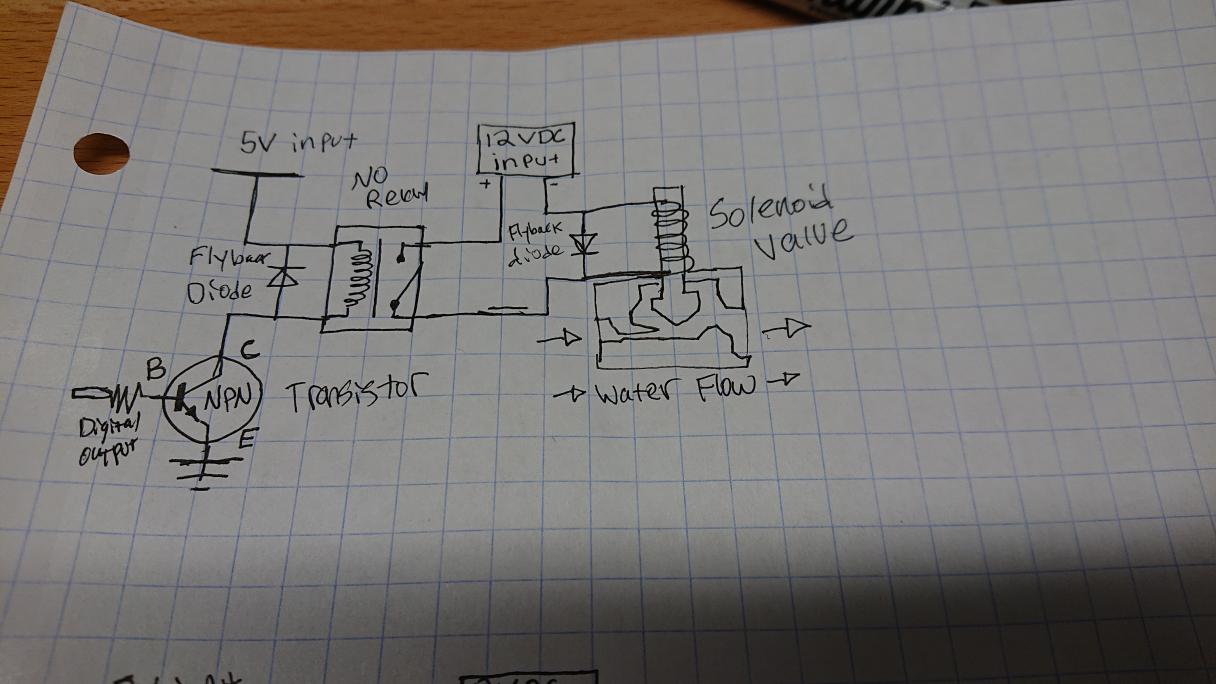


**Figure 11.** *The Breadboard*

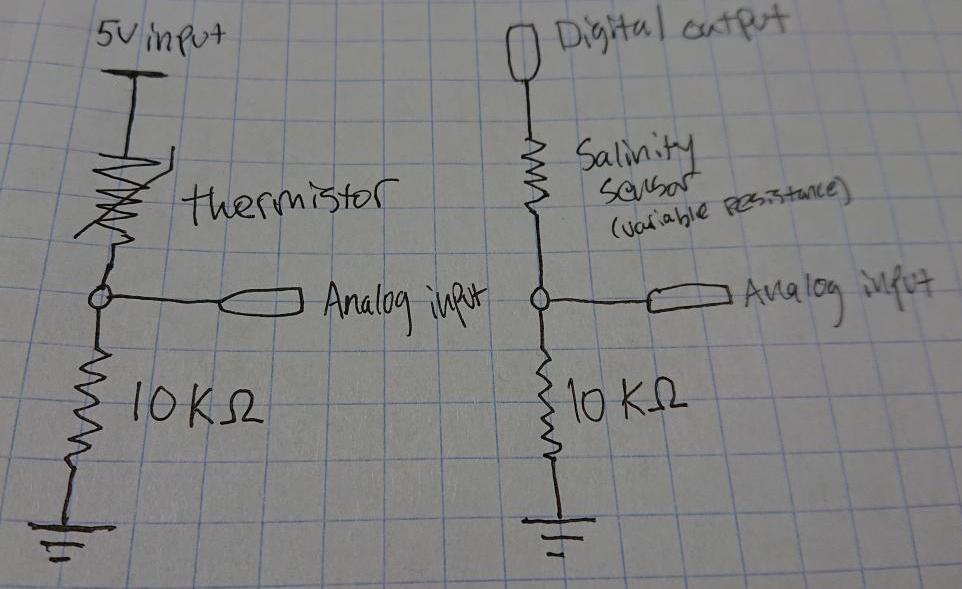
**Schematics:**



**Figure 12.** *Heater Schematic*



**Figure 13.** *Solenoid Schematic for Liquid Dispensing*

****

**Figure 14.** *(Left) Thermistor Schematic (Right) Salinity Sensor Schematic*

**Arduino Code for System Control:**

// Monitor and regulate salinity and temperature of a system

// Include LCD libraries

#include <Wire.h> // Library for I2C communication

#include <LiquidCrystal\_I2C.h> // Library for I2C communication

LiquidCrystal\_I2C lcd = LiquidCrystal\_I2C(0x27, 20, 4); // Change to (0x27,16,2) for 16x2 LCD.

//GLOBAL VARIABLES

int salinity\_power\_pin = 4; // Digital I/O pin salinity sensor

int temp\_power\_pin = 3; // Digital I/O pin thermistor

unsigned long salinity\_clock; //Deadtime measurement for salinity

unsigned long heat\_deadtime; //Deadtime measurement for heater

unsigned long heat\_clock; //Uptime measurement for heater

const int salinity\_input\_pin = A0; // Analog input pin

const int temp\_input\_pin = A1; // Analog input pin

const int h2o\_add\_relay = 7; //Digital output to activate fresh water system

const int salty\_add\_relay = 8; //Digital output to activate salt water system

const int heat\_on\_relay = 9; //Digital output to activate heater

int heat\_up\_time; //Variable for keeping track of heater up time, must be initialized outside loop

int heaterstatus = 0; //Variable for updating status of heater on LCD

// Constant setpoint and standard deviation values for salinity and temperature sensor systems

const float salsetpoint = 0.11, // in WT% SALINITY

tsetpoint = 21.5, // in DEGREES CELSIUS

salstddeviation = 3.4024, // in V-ANALOG

salty\_tank\_salinity = 1, // in WT% SALINITY

tempstddeviation = 0.69; // in V-ANALOG

int n = 15; // number of readings averaged

//---------------------------------------------------------------------------------------------------------------------------------------------

void setup() {

//Arduino pinMode Allocation

pinMode(salinity\_power\_pin, OUTPUT);

pinMode(temp\_power\_pin, OUTPUT);

pinMode(h2o\_add\_relay, OUTPUT);

pinMode(salty\_add\_relay, OUTPUT);

pinMode(heat\_on\_relay, OUTPUT);

heat\_up\_time = 0; //start the heater uptime timer at 0

//Display Control Limits

float saldeadband, tdeadband;

saldeadband = 3 \* salstddeviation; // deadband variable is 1/2 actual deadband for convenience V-ANALOG

tdeadband = 2 \* tempstddeviation; // deadband variable is 1/2 actual deadband for convenience V-ANALOG

//Serial monitor for troubleshooting

Serial.begin(9600);

// Initiate the LCD:

lcd.init();

lcd.backlight();

lcdsetup(salsetpoint, saldeadband, tsetpoint, tdeadband); //User-defined lcd setup function prints non updating data to lcd

//Wait for User readiness update sensor readings to monitors

int button\_pin = 11; // Digital input to monitor start

int startclick = LOW;

while (!startclick) {

startclick = digitalRead (button\_pin);

//Salinity readings

float salinityread, salinity;

salinityread = sensor\_reading\_ave(salinity\_power\_pin, salinity\_input\_pin, n); // Take an average of n readings and return average

salinity = salinityanalog\_to\_wtpercent(salinityread); // Convert average digital salinity reading of (0-1023) to the salinity % range of (0.0-0.56)

//Temperature readings and heater status

float temperature, temperatureread;

temperatureread = sensor\_reading\_ave(temp\_power\_pin, temp\_input\_pin, n); // Take an average of n readings and return average

temperature = temp\_to\_degrees(temperatureread); // Convert average digital salinity reading of (0-1023) to the temperature in C

//Monitor Updates

lcdupdate(salinity, temperature, 0);

/\*Serial.print("Salinity = ");

Serial.print(salinity);

Serial.print(" Analog Salinity Output = ");

Serial.print(salinityread);

Serial.print(" Temperatureread = ");

Serial.print(temperatureread);

Serial.print(" Temp = ");

Serial.print(temperature);\*/

}

//Start Clocks

salinity\_clock = millis();

heat\_deadtime = millis();

}

void loop() {

//Arduino PIN Allocation

digitalWrite(h2o\_add\_relay, LOW);

digitalWrite(salty\_add\_relay, LOW);

//Declare Control Limits

float saldeadband, sal\_lcl, sal\_ucl, tdeadband, t\_lcl, t\_ucl;

int saldeadtime, tdeadtime;

saldeadband = 3 \* salstddeviation; // deadband variable is 1/2 actual deadband for convenience in V-ANALOG

sal\_lcl = salinity\_to\_analog(salsetpoint) - saldeadband; // salinity lower control limit in V-ANALOG

sal\_ucl = salinity\_to\_analog(salsetpoint) + saldeadband; // salinity upper control limit in V-ANALOG

tdeadband = 3 \* tempstddeviation; // deadband variable is 1/2 actual deadband for convenience

t\_lcl = temp\_to\_analog(tsetpoint) - tdeadband; // temperature lower control limit in V-ANALOG

t\_ucl = temp\_to\_analog(tsetpoint) + tdeadband;

saldeadtime = 10500; // deadtime for salinity adjustment in MILLIS

tdeadtime = 15000; // deadtime for system heating response in MILLIS

//Take Salinity Readings and average them to give stable output

float salinityread, salinity;

salinityread = sensor\_reading\_ave(salinity\_power\_pin, salinity\_input\_pin, n); // Take an average of n readings and return average V-ANALOG

salinity = salinityanalog\_to\_wtpercent(salinityread); // Convert average digital salinity reading of (0-1023) to the salinity % range of (0.0-0.56) WT% SALINITY

//Take Temperature Readings

float temperature, temperatureread;

temperatureread = sensor\_reading\_ave(temp\_power\_pin, temp\_input\_pin, n); // Take an average of n readings and return average V-ANALOG

temperature = temp\_to\_degrees(temperatureread); // Convert average digital salinity reading of (0-1023) to the temperature in C

//LCD Display Update

lcdupdate(salinity, temperature, heaterstatus);

//SALINITY CORRECTION CONTROLS

if ((millis() - salinity\_clock) > saldeadtime) { //Check for Deadtime

if (salinityread < sal\_lcl) { //Check for low salinity

salinity\_clock = millis(); //Reset the salinity clock for for solenoid open duration

int salty\_open; //Time for solenoid to be open

salty\_open = salty\_duration(salinity, salsetpoint, salty\_tank\_salinity); //Calculate time for solenoid to be open in user-defined function

//Serial.println(salty\_open);

while (millis() - salinity\_clock < salty\_open) { //Open salty solenoid for salty\_open duration

digitalWrite(salty\_add\_relay, HIGH);

//Serial.print(millis());

//Serial.println ("salty open");

}

digitalWrite(salty\_add\_relay, LOW); //Close salty solenoid

Serial.print(millis());

Serial.println ("salty closed");

salinity\_clock = millis(); //Reset the salinity clock for deadtime

}

if (salinityread > sal\_ucl) { //Check for high salinity

salinity\_clock = millis(); //Reset the salinity clock for for solenoid open duration

int h2o\_open; //Time for solenoid to be open

h2o\_open = h2o\_duration(salinity, salsetpoint); //Calculate time for solenoid to be open in user-defined function

while (millis() - salinity\_clock < h2o\_open) { //Open h2o solenoid for h2o\_open duration

digitalWrite(h2o\_add\_relay, HIGH);

//Serial.print(millis());

//Serial.println("h2o open");

}

digitalWrite(h2o\_add\_relay, LOW); //Close h2o solenoid

//Serial.print(millis());

//Serial.println("h2o closed");

salinity\_clock = millis(); //Reset the salinity clock for deadtime

}

}

//HEATER CONTROLS

if ((millis() - heat\_clock) > heat\_up\_time) { //Check for heater on timer before switching heater off

digitalWrite(heat\_on\_relay, LOW); //Switch heater off after timer is up

heaterstatus = 0;

if ((millis() - heat\_deadtime) > tdeadtime) { //Check for heater deadtime

if (temperatureread < t\_lcl) { //Check for low temperature

heat\_up\_time = heat\_duration(temperature, tsetpoint); //Reset heater timer

heat\_clock = millis(); //Reset heater timer

heat\_deadtime = millis() + heat\_up\_time; //Reset deadtime clock

digitalWrite(heat\_on\_relay, HIGH); //Turn on heater

heaterstatus = 1; //Update LCD

if (temperatureread > t\_ucl) {

digitalWrite(heat\_on\_relay, LOW);

heaterstatus = 0;

}

}

}

}

}

// -------------------------------------------------

// Reading Averaging Function

float sensor\_reading\_ave(int power\_pin, int input\_pin, int nave) {

float ave, sum;

digitalWrite(power\_pin, HIGH); // Turn sensor on

delay (100); // Wait to settle

sum = 0.0;

for (int i = 1; i <= nave; i++) {

sum += analogRead(input\_pin); // Accumulate the sum

delay(10); // Pause, briefly

}

digitalWrite(power\_pin, LOW); // Turn sensor off

ave = sum / float(nave); // Compute average

return ave;

}

//-----------------------------------------------------

//Digital salinity output conversion to Salinity percentage

float salinityanalog\_to\_wtpercent(float salread) {

float salpercent;

if (salread < 638.86) {

salpercent = salread \* 0.0000898761686149 - 0.00741828908132;

} else if (salread >= 638.86) {

salpercent = salread \* 0.00132661183338 - 0.797519235873;

} return salpercent;

}

// -------------------------------------------------------

//Salinity percentage to digital output converter

float salinity\_to\_analog(float percentread) {

float salinityread;

if (percentread < 0.05) {

salinityread = (percentread + 0.00741828908132) / 0.0000898761686149;

} else if (percentread >= 0.05) {

salinityread = (percentread + 0.797519235873) / 0.00132661183338;

} return salinityread;

}

// ------------------------------------------------------

// Temperature Degrees to analog converter

float temp\_to\_analog(float tempdegs) {

float tempread;

tempread = ((-0.019088132 \* sq(tempdegs)) + (10.651701 \* tempdegs) + 256.9406);

return tempread;

}

// -----------------------------------------------------

// Temperature Analog to Degrees converter

float temp\_to\_degrees(float tempread) {

float tempdegs;

tempdegs = ((0.000021486654 \* sq(tempread)) + (.0811732 \* tempread) - 22.1219);

return tempdegs;

}

// ----------------------------------------------------------------

//Calculate open time for salinity tank

long int salty\_duration(float salpercent, float salinitysetpoint, float salty) {

//Calculate target final salinity after gain

float gain, salfinal, massadd, masstank, percentretained, flowrate, t\_open;

gain = 0.8;

percentretained = 0.85;

masstank = 140;

flowrate = 5;

salfinal = salinitysetpoint + ((1 - gain) \* (salpercent - salinitysetpoint));

massadd = (1 / percentretained) \* ((salpercent - salfinal) / (salpercent - salty)) \* masstank;

t\_open = 1000 \* (massadd / flowrate);

long int timeopen;

timeopen = t\_open;

return timeopen; //in millis

}

// ----------------------------------------------------------------

//Calculate open time for DI h2o tank

long int h2o\_duration(float salpercent, float salinitysetpoint) {

float gain, salfinal, massadd, masstank, percentretained, flowrate, t\_open;

gain = 0.8;

percentretained = 0.85;

masstank = 140;

flowrate = 5;

salfinal = salinitysetpoint + ((1 - gain) \* (salpercent - salinitysetpoint));

massadd = (1 / percentretained) \* ((salpercent - salfinal) / (salpercent)) \* masstank;

t\_open = 1000 \* (massadd / flowrate);

long int timeopen;

timeopen = t\_open;

return timeopen; //in millis

}

// ----------------------------------------------------------------

//Calculate heater uptime

long int heat\_duration(float temperature, float tempsetpoint) {

long int heatuptime;

float K = 0.506;

heatuptime = 60000 \* abs(tempsetpoint - temperature) / K;

return heatuptime;

}

// ------------------------------------------------

// LCD SETUP Function

void lcdsetup(float salinitysetpoint, float salinitydeadband, float tempsetpoint, float tempdeadband) {

// Prints sensor values to LCD and labels them

lcd.setCursor(2, 0); // Set the cursor on the first column and first row.

lcd.print("LCL"); // Prints the text label for colum

lcd.setCursor(8, 0); // Set the cursor on the first column and first row.

lcd.print("SP"); // Prints the text label for column

lcd.setCursor(15, 0);// Set the cursor on the first column and first row.

lcd.print("UCL"); // Prints the text label for column

lcd.setCursor(0, 1);

lcd.print("S:"); // Print row salinity label

lcd.setCursor(2, 1);

lcd.print(salinityanalog\_to\_wtpercent(salinity\_to\_analog(salinitysetpoint) - salinitydeadband), 3); //Print salinity lcl in WT% SALINITY

lcd.setCursor(8, 1);

lcd.print(salinitysetpoint, 3); //Print salinity setpoint WT% SALINITY

lcd.setCursor(15, 1);

lcd.print(salinityanalog\_to\_wtpercent(salinity\_to\_analog(salinitysetpoint) + salinitydeadband), 3); //Print salinity ucl WT% SALINITY

lcd.setCursor(0, 2);

lcd.print("T:"); //Print row temperature label

lcd.setCursor(2, 2);

lcd.print(tempsetpoint - tempdeadband); //Print temperature lcl

lcd.setCursor(8, 2);

lcd.print(tempsetpoint); //Print temperature setpoint

lcd.setCursor(15, 2);

lcd.print(tempsetpoint + tempdeadband); //Print temperature ucl

lcd.setCursor(0, 3);

lcd.print("S="); //Print current salinity label

lcd.setCursor(8, 3);

lcd.print("T="); //Print current temperature label

lcd.setCursor(15, 3);

lcd.print("H="); //Print heater status label

}

//-------------------------------------------

// LCD update function

void lcdupdate(float salinitydisplay, float tempdisplay, int heatdisplay) {

lcd.setCursor(2, 3);

lcd.print(salinitydisplay, 3);

lcd.setCursor(10, 3);

lcd.print(tempdisplay, 2);

lcd.setCursor(17, 3);

if (heatdisplay == 0) {

lcd.print("OFF ");

} else {

lcd.print("ON ");

}

}

Hand Calculation Examples: Salinity

Data:

Calculate the upper control limit:

Calculate the lower control limit:

Calculate the deadband:

Calculate salinity % after corrections are made:

Calculate mass of water that needs to be added to reach the setpoint:

)(140g)

Calculate the duration of time the solenoid valve needs to be open:

Hand Calculation Examples: Temperature

Data:

Calculate upper control limit:

Calculate the lower control limit:

Calculate the deadband:

Calculate the duration of time the heater needs to be on:

Variable Definitions