NOTE: Some functions in setup() and loop() are not included in this document to conserve space

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

// Main Control Code

// Version 2.0

// 2/23/20

// The North American Council on Aquatic Housing and Development

// Extends sensor\_reading code base to actuate solenoids

// Usage: This program is intended to running live when our system control is being // tested.

// For debugging and testing use testing\_code\_v#.

// NOTE: To run this code you must have downloaded the LiquidCrystal\_I2C library (and

// added it to the sketch from the IDE),

// which is in Whitman's folder in the main directory

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

// Setup LCD screen

#include <Wire.h> // Include wire (i2c comm) and liquidcrystal libraries

#include <LiquidCrystal\_I2C.h>

LiquidCrystal\_I2C lcd = LiquidCrystal\_I2C(0x27, 20, 4); // 20x4 LCD screen, 0x27 is the address

// delcare global pins and heater state

const int SALINITY\_POWER\_PIN = 8;

const int saltyPin = 10;

const int freshPin = 11;

const int heaterPin = 12;

int heaterState = 0;

void setup()

{

// Setup pins and serial comms

pinMode(SALINITY\_POWER\_PIN, OUTPUT);

pinMode(saltyPin, OUTPUT);

pinMode(freshPin, OUTPUT);

// Setup LCD

lcdSimpleSetup();

// Setup serial comms

Serial.begin(9600);

// systemFlush();

}

void loop()

{

// declare local pins

const int SALINITY\_READING\_PIN = A0;

const int TEMPERATURE\_READING\_PIN = A1;

const int S\_SETPOINT\_PIN = A2;

const int T\_SETPOINT\_PIN = A3;

// declare constants and breakpoints for polynomial fit of salinity calibration

// Note on using constants:

// c --> prefix to show variable is a constant

// h,l --> used for high/low salinity line

// 1,2 --> first or second constant in eq. y = c1\*x + c2

// b1 --> low point determined from DI water average reading

// b2 --> mid point at 0.05 wt% to transition from one fit line to the next

// b3 --> high point determined from 0.15 wt% water reading

const float cl1 = 9.5754848e-05;

const float cl2 = -7.7561327e-03;

const float ch1 = 8.5607131e-04;

const float ch2 = -4.6129742e-01;

const int b1 = 81;

const int b2 = 603;

const int b3 = 708;

// setup variables

int numReadings = 30; // number of readings per salinity reading

int salinityReading; // current analog reading from salinity sensor

float salinityPercentage; // current salinity percentage

float tempReading; // current analog reading from thermistor

float systemTemp; // current temperature of system (deg C)

float solTime = 0; // time to hold solenoid open

int solPin; // pin of solenoid to open

// declare sigma(analog) and deadtime from calibration

// note: s & t prefixes refer to salinity and temperature

// Compute setpoint from pot reading

float sSetpoint = findSalinityPercentage(cl1, cl2, ch1, ch2, b1, b2, b3,

analogRead(S\_SETPOINT\_PIN));

const float sSigma\_analog = 2.265; // from calibration of salinity percentages

const unsigned long deadtime = 15000; // averaged

// convert sigma\_analog to wt %

// Compute setpoint from pot reading

float tSetpoint = findTempFromAnalog(analogRead(T\_SETPOINT\_PIN));

float tSigma\_analog = 0;

float tSigma = findTempFromAnalog(tSigma\_analog);

// compute UCL & LCL for salinity and temperature

float sUCL = findSalinityPercentage(cl1, cl2, ch1, ch2, b1, b2, b3,

analogRead(S\_SETPOINT\_PIN) +

5\*sSigma\_analog);

float sLCL = findSalinityPercentage(cl1, cl2, ch1, ch2, b1, b2, b3,

analogRead(S\_SETPOINT\_PIN) –

5\*sSigma\_analog);

float tUCL = tSetpoint + 3 \* tSigma;

float tLCL = tSetpoint - 3 \* tSigma;

// --------------- Things actually happen down here -----------------------

// take a salinity reading and convert to percentage salt

salinityReading = takeReading(SALINITY\_POWER\_PIN, SALINITY\_READING\_PIN,

numReadings);

salinityPercentage = findSalinityPercentage(cl1, cl2, ch1, ch2, b1, b2, b3,

salinityReading);

// take temperature reading and convert to system temperature

tempReading = analogRead(TEMPERATURE\_READING\_PIN);

systemTemp = findTempFromAnalog(tempReading);

// calculate which solenoid to open and for how long, and modify solPin and

// solTime.

setAdjustmentTimes(salinityPercentage, sSetpoint, sUCL, sLCL, deadtime, &solPin,

&solTime);

// turn solenoids on or off

toggleSolenoids(solPin, solTime, deadtime);

// Update LCD screen

lcdUpdate(sLCL, sSetpoint, sUCL, tLCL, tSetpoint, tUCL, salinityPercentage,

systemTemp, heaterState);

}

void toggleSolenoids(int solPin, int solTime, int deadtime){

// this function controls the solenoids, that is all it does

// allows system to make adjustments immediately after powering on

static unsigned long startTime = (-1)\*deadtime;

static int solStatus = 0; // 0 = closed, 1 = open

if (solStatus == 0 && solTime > 0 && (millis()-startTime) > deadtime) {

// checks that: solenoid is closed,

// time to open solenoid is > 0,

// deadtime has passed

digitalWrite(solPin, HIGH);

solStatus = 1; // set solenoid status to open

startTime = millis(); // set start time

}

else if ((millis()-startTime) > solTime){

// closes solenoid if solTime has passed

digitalWrite(solPin, LOW);

solStatus = 0;

}

if ((millis()-startTime) < deadtime) {

// prints a deadtime clock up in upper left corner

int lcddeadDisplay = (((deadtime-(millis()-startTime))/1000));

lcd.setCursor(0,0);

lcd.print(lcddeadDisplay);

lcd.print(" ");

if(lcddeadDisplay == 0){

lcd.setCursor(0,0);

lcd.print(" ");

}

}

}

float takeReading(int powerPin, int readingPin, int numReadings) {

// input: powerPin (digital out), readingPin (analog in), numReadings (>0)

// output: average of numReadings readings

float sum = 0.0;

digitalWrite(powerPin, 1); // Turn sensor on

delay(100); // Allow power to settle

for (int i = 0; i < numReadings; i++) { // Take readings and sum

sum += analogRead(readingPin);

delay(10);

}

digitalWrite(powerPin, 0); // Turn sensor off

sum /= numReadings; // Average sum over readings

return sum;

}

float findSalinityPercentage(float c1, float c2, float c3, float c4,

int b1, int b2, int b3, float reading) {

// input c1, c2, c3, c4 --> polynomial constants for two line fits

// b1, b2, b3 --> breakpoints to determine regin for evaluation

// reading --> analog reading value from sensor

// output: corresponding salinity in wt%

// check breakpoints to determine function region

if (reading < b1) { // reading < DI water breakpoint

return 0;

}

else if (reading < b2) { // reading < 0.05 wt% breakpoint

return evaluatePolynomial(reading, c1, c2);

}

else if (reading < b3) { // reading < 0.15 wt% breakpoint

return evaluatePolynomial(reading, c3, c4);

}

else{ // reading > 0.15 wt% breakpoint

return evaluatePolynomial(b3, c3, c4); // return value of polynomial at b3

}

}

float evaluatePolynomial(int x, float c1, float c2) {

// evaluates y = c1\*x + c2 and returns y

return c1\*x + c2;

}

float setAdjustmentTimes(float currentSalinity, float setpoint, float UCL, float LCL, int deadtime, int\* solPin, float\* solTime)

{

// input: current salinity, setpoint, UCL, LCL, deadtime, solenoid pin, solenoid

open time

// output: none

// modifies solTime and solPin for toggleSolenoids() function

if (currentSalinity > UCL || currentSalinity < LCL) {

// Set target salinity to 80% of the difference between current salinity and

setpoint

float targetSalinity = currentSalinity - (currentSalinity - setpoint) \* 0.8;

if (targetSalinity > currentSalinity) {

\*solPin = saltyPin; // sets salty pin to be adjusted

setTime(targetSalinity, currentSalinity, 1, solTime); // sets time for

salty solenoid

}

else {

\*solPin = freshPin; // sets fresh pin to be adjusted

setTime(targetSalinity, currentSalinity, 0, solTime); // sets time for

fresh solenoid

}

}

return 0;

}

float setTime(float targetSalinity, float currentSalinity, int addedSalinity, float\*

time)

{

// input: targetSalinity (of system),

// currentSalinity (of system),

// addedSalinity (% salinity of fluid to be added),

// solTime from loop()

// calculates time to open solenoid, and modifies solTime for use in

toggleSolenoids() function

const float overflowFraction = .2; // Fraction of added water that overflows

before mixing

const float totalMass = .143; // Total mass of water in a filled system

(kg)

float flowRate = 0; // Mass flow rate of solenoids (kg/s)

if (addedSalinity == 1) {

flowRate = .003633; // Experimental salty tank flow rate

}

else {

flowRate = .004167; // Experimental fresh tank flow rate

}

// calculate mass of water to add

float massToAdd = totalMass \*

(currentSalinity - targetSalinity) /

(currentSalinity - addedSalinity) \*

(1 / (1 - overflowFraction));

// calculate time needed to add appropriate quantity of mass

\*time = ( massToAdd / flowRate ) \* 1000; // x1000 to convert to ms. Sets solTime

in loop() to calculated time

}