Control of Salinity in the Fish Tank ME 121

Figure 1 is a schematic of the fish tank and the salinity control system. The pump circulates water continuously through a loop from the tank to the salinity sensor. When the salinity readings are outside of the desired range, the Arduino program briefly opens one of the two solenoid valves attached to either the fresh or salty water reservoirs.

This document describes a set of measurements and analyses that will help you to write an Arduino program to control the salinity of water in your fish tank. The goal is to develop a few simple models to characterize the behavior of the fish tank system. In other words, we want you to use engineering analysis, not guesswork, to design your control system for maintaining the salinity of the fish tank within desired limits.

The models you develop are not complete without measurements. Furthermore, the data you collect will be specific to your fish tank. Therefore, without doing the measurements on *your* fish tank, you will not be able to use the models effectively. The following measurements are necessary for developing your control algorithm

- Calibration data for the salinity sensor
- Calibration of flow rate through the solenoid valves
- *Response time* of the system to disturbances in salinity

In addition to the empirical models, two models are obtained from mass balances

- A batch mixing model to predict how much salty or fresh water needs to be added when the system is out of the dead band for control;
- An overflow bypass model to account for short-circuiting of added salty or fresh water to the overflow instead of being mixed.

Although the immediate goal is to develop a working Arduino program for controlling salinity of the fish tank, the models have additional value in helping you understand the behavior of your system. Learning to reason with these models will also develop your knowledge of sensors and control techniques, and to lay the foundation for working with more complex electromechanical systems in later classes.

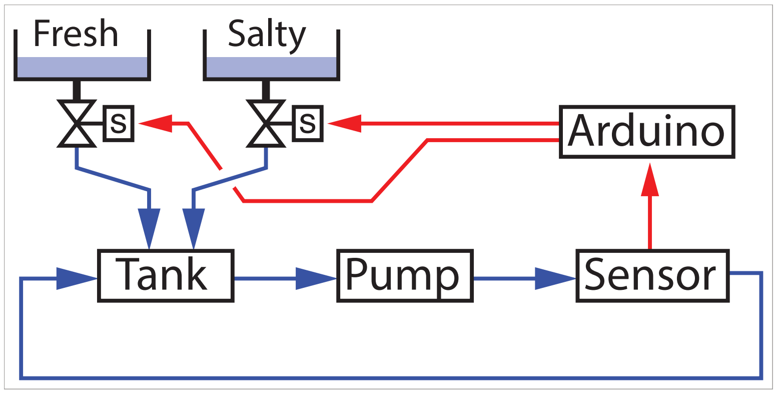

Figure 1 Schematic of flow loop and salinity control scheme for the fish tank.

After the models and empirical parameters of your fish tank have been established, two additional details are considered

- Display of the system status on an LCD panel
- Arduino code for implementing the response time without blocking execution.

1. Summarize the calibration data for the salinity sensor

We assume that you have calibrated your salinity sensor in an earlier exercise. Record the summary data from your calibration experiments in Table 1. Determine which calibration point has the most variation and use data from that measurement to estimate $\Delta\text{UCL} = \Delta\text{LCL} = 3\sigma$.

Table 1 Summary of calibration data for the salinity sensor.

Wt% NaCl	*n*	Mean	Standard deviation
0			
0.05			
0.10			
0.15			

ΔUCL = ΔLCL = ______________

2. Obtain piecewise linear regression for the calibration equations

Calibration equations provide the relationships between salinity of water flowing through the salinity sensor and the analog input readings from the voltage divider circuit of the sensor. Let r be the raw reading of the analog input channel, and let S be the salinity of water flowing through the sensor. The results of the calibration measurements can be described by

$$r = f(S) \tag{1}$$

where we call $f(S)$ the forward calibration equation. The left side of Figure 2 shows $f(S)$. Because the calibration data is relatively sparse, and because the forward calibration function changes more rapidly between 0% and 0.05% salinity than it does between 0.05% and 0.15%, it is easier to work with a piecewise linear calibration equation.

To control and display the salinity of the water in the fish tank, we need the *reverse* calibration equation

$$S = g(r) \tag{2}$$

which is displayed in the right side of Figure 2. We don't need to perform additional experiments to obtain the reverse calibration. We just use the same calibration data, but switch the roles of the independent and dependent variables. As with the forward calibration, we recommend using piecewise linear fits to the data.

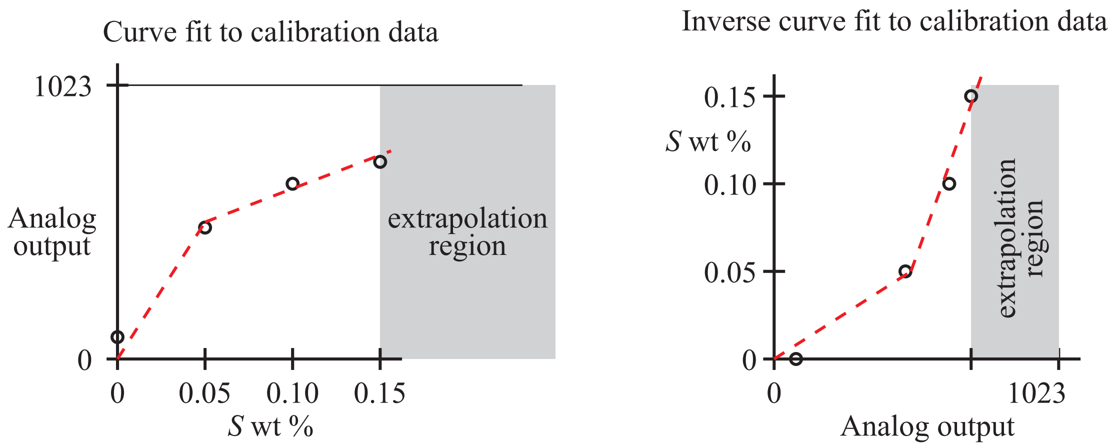

Figure 2 Forward and reverse calibration relationships. The reverse calibration is used to determine salinity from the reading of the voltage divider for the salinity sensor.

Use linear regression to obtain the forward and reverse calibration equations. You will need two straight line segments for the forward calibration and two straight line segments for the reverse calibration. Make sure you have at least 5 digits for each of your curve fit coefficients.

Write an Arduino function for the reverse calibration equation. The function return the salinity value given a raw analog input value from the voltage divider for the salinity sensor. Figure 3 is an incomplete listing of such a function. The `rbreak` value is the value of the reading where the fit formulas in a piecewise calibration equation. The code inside the final `else` block should prevent bad input from causing trouble in the control algorithm.

```
float salinity_value(int reading) {

  int rbreak = ...;      //  separation between linear segments
  int rlimit = ...;      //  upper limit of acceptable readings
  float salinity;        //  Compute salinity from calibration equations

  if ( reading<0 ) {
    // print error message, reading can't be negative

  } elseif ( reading<rbreak ) {
    // do something

  } elseif ( reading<rlimit ) {
    // do something else

  } else {
    // do something to be safe

  }
  return(salinity);
}
```

Figure 3 Unfinished Arduino code for computing salinity as a function of salinity sensor reading.

3. Measure flow rate through the solenoid valves.

The mass balance model described later requires knowledge of how much water is added to the tank from the solenoid valves. In this section, we describe a simple experiment to measure the flow rate from the valves, which provides the missing data in the mass balance model.

Figure 4 shows the arrangement of the fish tank system during the flow rate measurements. The pump is turned off and the flow loop does not need to have any water in it. The flow rate measurements only involve one of the supply tanks, the solenoid valve and the tubing that connects the solenoid to the PVC fish tank. Fill one of the tanks halfway with *tap water* – there is no need to use calibration samples for this measurement. Keep the hose from the solenoid attached and hold it so that water flowing through the solenoid is collected in a measuring cup. Any lightweight (plastic) cup will do as long as it is clean. After each run of the experiment, mass of the cup with collected water is placed on a laboratory scale. Before making the measurements, be sure to tare the scales.

The flow rate measurement involves opening a solenoid valve for a known amount of time and collecting the water from the valve in a cup. A simple Arduino program controls when the solenoid is open. In order to account for variations during the measurements, the valve is opened and closed several times. The flow rate is then computed from the total mass of water collected divided by the total time the valve is open.

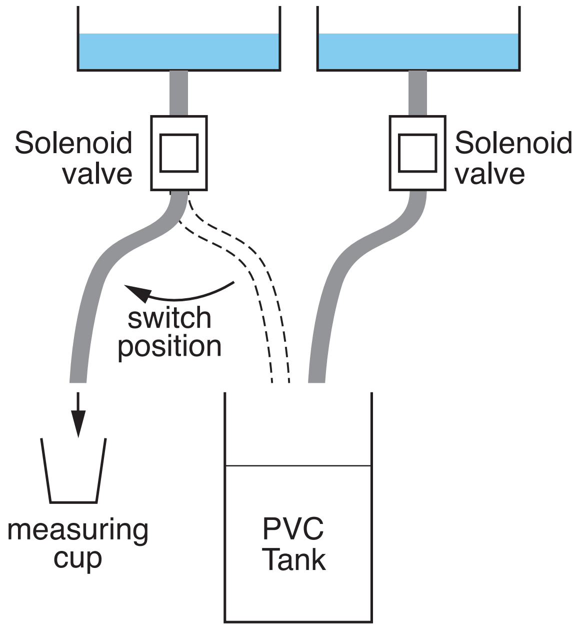

Figure 4 Using a measuring cup to capture water during the flow rate measurements.

Figure 5 is an outline of an Arduino program to collect the data for the flow rate measurements. You will need to translate this outline into a working program. We recommend that these statements are in the body of the `setup` function, and that the `loop` function for this program is empty. Putting the code in the `setup` function means that it will be executed only once when the Arduino is turned on, or once after the reset button is pushed.

```
wait enough time to set up your measuring cup (say 10 seconds)
repeat N times:          ( N >= 10 )
   open the valve
   wait one second
   close the valve
   wait one second
stop
```

Figure 5 Outline of Arduino code to make mass flow measurements by cycling the valve N times.

Use the scales to weigh the water collected during the experiment. Record your data in a table like that in Table 2. It would be a good idea to use a spreadsheet to store the data. Compute the mean and standard deviation of your measurements. Table 2 provides a sample of the data collection and reporting for the mass flow measurements. and perform the conversion and averaging. If you have time, repeat the measurements for the other solenoid valve.

Table 2 Layout of a table for collecting and analyzing measurements of mass flow rate through one of the solenoid valves.

Reading #	Mass (g) (valve 1)	Δt_1 (s)	$\dot{m}$ (g/s)
1			
2			
3			
4			
5			
		Mean:	
		Std. dev	

4. Measure the response time of the system

To create an effective and robust Arduino program for controlling the salinity of the fish tank, we have to account for the *response time* of the system, which is the time it takes the system to respond to a sudden disturbance. The rate at which water circulates, the amount of water stored in the tank and flow loop, and the efficiency of mixing in the fish tank all determine how quickly the system responds to an addition of fresh or salty water. Those features are intrinsic physical characteristics of the system that cannot be directly altered by any control system.

The response time is measured by observing the output of the salinity sensor after a sudden pulse of very salty water is introduced into the fish tank. Figure 5 shows a typical output of the salinity sensor during a response time measurement. The top of Figure 4 has a line plot that represents the state of solenoid valve. Between time t_1 and t_2, the valve is open and water flows from the reservoir into the fish tank. At t_1 there is no immediate change in the output of the salinity sensor. For our purposes, we define the response time as the delay between t_1, the start of the input of salty water, and the time at which the sensor reads a new steady value.

Experiment to measure response time

To measure response time, we are not concerned about the absolute levels of salinity in the system. We only need to measure time between the input of salty water and the time for the salinity reading to stabilize at the new level of salinity. Since we are only interested in the time of the response, and we don't need to worry about the actual salinity levels, the experiment can be easily repeated to get a good reading of the response time. The process for determining the response time follows these steps:

1. Put some very salty (0.15%) water solution in the salty reservoir tank.
2. Fill the fish tank with DI water and run the pump to circulate the water. NOTE: You do not need to flush your system with DI water before starting this experiment.
3. Have an Arduino program running to display the output of the salinity sensor on the Serial Monitor.
4. When the system is in equilibrium introduce a pulse of salty water from the supply reservoir and record the time that the pulse begins (see below)
5. When the system is in equilibrium at a higher level of salinity, copy the Serial Monitor output to an external file (say a spreadsheet).

Steps 4 and 5 can be repeated to obtain several measurements of response time.

The response time of most fish tanks (for ME 121) is on the order 10 seconds. The response time for your fish tank may be greater or less than 10 seconds depending on the efficiency of your pump, the length of hoses in your flow loop and how well the water in the tank mixes.

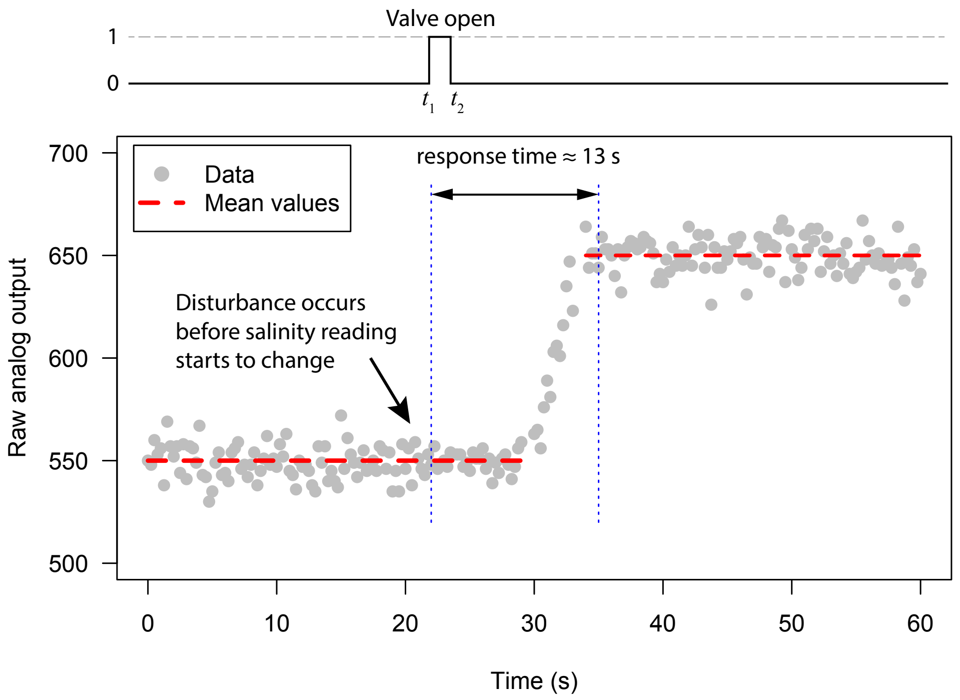

Figure 5 Typical response of a fish tank to a sudden, but short, input of salty water. Gray dots are measurements and dashed read lines are averages of measurements

Arduino code to measure response time

The goal of the response time measurement is to obtain data like that in Figure 5. There are (at least) two ways to measure the response time of your system. We'll call these method A and method B. Method A uses a simple Arduino program and human observation, and it requires careful coordination from at least two team embers. Method B uses a more complicated Arduino program that writes all the necessary data to the Serial Monitor. Using method B, and copying the Serial Monitor data to a spreadsheet, it is possible to construct a plot like that in Figure 5.

To use method A, write a simple Arduino code that prints the system clock (`millis()`) and the output of the salinity sensor reading to the Serial Monitor. While one team member watches the Serial Monitor, add some salty water to the tank from a bottle of 0.15% salt solution. The person watching the Serial Monitor needs to record the time when the salty water is added and the time when the system has attained equilibrium.

Figure 6 shows the Arduino code for measuring the response time with method B. Digital pin 9 of the Arduino board needs to be connected to the cascade control circuit for the solenoid valves controlling the flow from the salty water reservoir. In addition, a simple button circuit needs to be created and connected to digital pin 6. (Of course, you could change the code to use different digital pins.)

Figure 6 does not show the code for `sensorAverage`, which computes the average of several readings on an analog input channel. You will need to write your own code for `sensorAverge`. We recommend average 5 to 10 readings and pausing no more that 10 milliseconds between readings while accumulating values to compute the average.

```
//  File:  fish_tank_response_time.ino
//
//  Record output from salinity sensor before, during and after a
//  pulse of salty water is added to the system.  The pulse is
//  controlled by a digital output that controls a solenoid circuit
//  Wait for button press to initiate measurement of response time

unsigned long start_time;   //  Store time at start of the experiment

int relay_pin  = 9;         //  Digital output to control relay
int button_pin = 6;         //  Digital input to monitor start
int salinity_power_pin = 7; //  Digital input to turn on salinity sensor
int salinity_input_pin = 1; //  Analog input for salinity sensor
int nave=5;                 //  Number of salinity readings to average

// ------------------------------------------------------------------
void setup() {
  float reading;

  pinMode(button_pin, INPUT_PULLUP);
  pinMode(relay_pin,OUTPUT);
  pinMode(salinity_power_pin, OUTPUT);
  Serial.begin(9600);

  // -- Wait for button press.  In the meantime, display salinity reading
  while ( digitalRead(button_pin) ) {
    reading = sensorAverage(salinity_power_pin, salinity_input_pin, nave);
    Serial.print(reading);
    Serial.println("\twaiting for button press");
  }

  start_time = millis();    //  Save millis() value at time = 0}
}

// ------------------------------------------------------------------
void loop() {

  float reading;
  int   relay_state, pulse_start = 5000, pulse_duration = 1500;
  unsigned long now;

  // -- Read current time and salinity. Save time relative to start_time
  now = millis() - start_time;
  reading = sensorAverage(salinity_power_pin, salinity_input_pin, nave);

  // -- If time is in the pulse window, send salty water to the tank.
  //    Use millis() to monitor duration of the pulse, NOT delay()
  if ( now>pulse_start & now<(pulse_start+pulse_duration) ) {
    digitalWrite(relay_pin, HIGH);
    relay_state = 1;
  } else {
    digitalWrite(relay_pin, LOW);
    relay_state = 0;
  }

  // -- Display status for this time step.
  Serial.print(now);
  Serial.print("\t");   Serial.print(reading);
  Serial.print("\t");   Serial.println(relay_state);
}
```

Figure 6 Arduino code for measurement of response time by method B. Note that the code for `sensorAverage` is not included in this listing.

After making an adjustment to the fish tank salinity – by either adding salty or fresh water – you should wait one deadtime interval before making another adjustment. Because the deadtime of your system depends on many factors that are unique to your system, you will need to measure the deadtime with an experiment. For this experiment, you write an Arduino code that measures and prints the raw analog reading of your salinity sensor to the Serial Monitor. The Arduino code for measuring the deadtime is not the same code that you use to control the salinity.

5. Write equations used in the control code.

Let X_{sp} be the mass fraction at the setpoint. The instructor will give you a value of X_{sp} when the fish tank is tested. You can assume that $0.0005 \le X_{\text{sp}} \le 0.0010$. The X_{sp} value will be given as a wt% of NaCl, but you will need to convert it to an appropriate value in your code. Given the setpoint, compute the UCL and LCL used in the control algorithm:

$$\text{LCL} = X_{\text{sp}} - \Delta\text{LCL} \tag{3a}$$

$$\text{UCL} = X_{\text{sp}} + \Delta\text{UCL} \tag{3b}$$

The fractional form for mass fraction (0.001 instead of 0.1%) is recommended.

Let X_s be the measured value of salinity, i.e., the measured mass fraction of salt in the tank. The error in the salinity reading is

$$E = X_s - X_{\text{sp}} \tag{4}$$

where E is positive when the current salinity reading is above the set point. Let G be the gain applied to the error. For example, a gain of 0.75 would mean that 0.75 E of the error is to be corrected in one change to the system.

Use an analysis of the system as a batch process to predict the amount of DI or salty water to be added to the system to make a correction of $G{\times}E$. Figure 7 shows a schematic of the batch model for correction of the salinity of water in the tank.

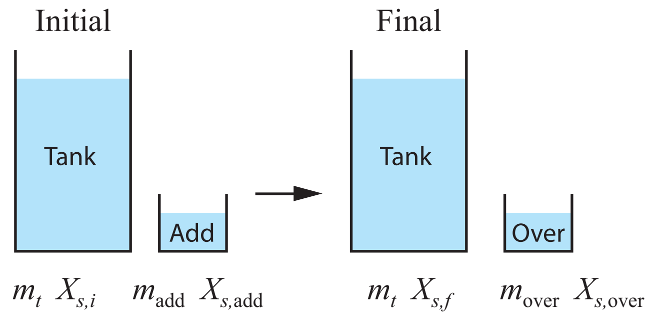
Initial
Final
Tank
Add
Tank
Over
m_t $X_{s,i}$
m_{add} $X_{s,\text{add}}$
m_t $X_{s,f}$
m_{over} $X_{s,\text{over}}$

Figure 7 Schematic of the mass addition to correct an error in the tank salinity.

Define the following variables.

m_t	mass of water in the tank, both before and after the correction
$X_{s,i}$	mass fraction of salt in the tank before correction (i = initial)
X_{sp}	set point for mass fraction of salt in the tank
m_{add}	mass of water added to correct the error in mass fraction
$X_{s,\text{add}}$	mass fraction of salt in the added water
$X_{s,f}$	mass fraction of salt in the tank after correction (f = final)

m_{over}	mass of water leaving the tank via the overflow, $m_{\text{over}} = m_{\text{add}}$
$X_{s,\text{over}}$	mass fraction of salt in the overflow
F	fraction of added water that short-circuits to the outflow without mixing
G	gain of the control system
$\dot{m}_{\text{sol}}$	mass flow rate through the solenoid valve

In the in-class example problem, we assumed that 15 percent of the added water was immediately short-circuited to the overflow. In general, we use F to designate the short-circuit fraction. Then

$$X_{s,\text{over}} = (1 - F)X_{s,i} + F\, X_{s,\text{add}} \tag{5}$$

where $0 \leq F \leq 1$. The in-class example problem showed that the final salinity after a correction applied with gain G is

$$X_{s,\text{f}} = X_{s,i} + G\,(X_{\text{sp}} - X_{s,i}) \tag{6}$$

where $0 \leq G \leq 1$. The mass necessary to make this correction is

$$m_{w,\text{add}} = m_{w,t}\,\frac{G(X_{\text{sp}} - X_{s,i})}{(1-F)(X_{s,add} - X_{s,i})} \tag{7}$$

The time that the solenoid is open is

$$\Delta t = \frac{m_{w,\text{add}}}{\dot{m}_{\text{sol}}} \tag{8}$$

where $\dot{m}_{\text{sol}}$ is the mass flow rate through the valve when it is open. Equations (3), (7) and (8) are used in the Arduino control code.

6. Display status of system on the LCD panel

Write an Arduino function to display the status of the fish tank on the LCD panel. The layout of the panel must use the template in Figure 8. The standardization of the display helps instructors correctly inspect many different fish tank designs.

The first and last rows of the display are labels that do not change during operation of the salinity control algorithm. The second row indicates the control settings for the LCL, SP and UCL. These values do not normally change during the operation of the tank. However, a potentiometer can be used to change the SP, which will also change the LCL and UCL. The third row of the display shows current status of the solenoid valves, and in the middle shows the current reading of the salinity sensor.

	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
0		L	C	L				S	e	t	P	t				U	C	L		
1	0	.	0	7	2			0	.	1	0	0			0	.	1	0	8	
2	S	a	l	t	y		C	u	r	r	e	n	t			D	I			
3		O	F	F				0	.	1	4	8				O	N			

Figure 8 Layout of the LCD display for the salinity control verification.

Display of floating point values on the serial-enabled LCD panel requires using the SoftwareSerial library. Refer to the example code from the class web site.

To display floating point values you will need to use the built-in `dtostrf()` function to convert the floating point value to a string, and then display the string on the LCD panel with `lcd.write()`. Figure 9 lists an excerpt of code that can display the second line in Figure 8.

```
float LCL=0.072, SetPt=0.100, UCL=0.108;
char float_buffer[16];   // string buffer to hold formatted value

lcd.setCursor(0,1);
dtostrf(float_buffer,5,3,LCL);    lcd.print(float_buffer);
lcd.print("  ");
dtostrf(float_buffer,5,3,SetPt);  lcd.print(float_buffer);
lcd.print("  ");
dtostrf(float_buffer,5,3,UCL);    lcd.print(float_buffer);
```

Figure 9 Section of Arduino code that demonstrates how the `dtostrf()` function converts a floating point value to a string that can be displayed on the LCD panel with `lcd.print()`. Note that the values of LCL, SetPt, and UCL should be computed with formulas, and not assigned as constants as is done in the simple example presented here. Also note that SP appears to be a reserved word. Therefore, do not us SP as a variable name unless you want to spend lots of time trying to find a compilation bug.

7. Implement the control algorithm in Arduino code

The main steps in the control algorithm are

1. Measure salinity and update the LCD panel
2. If the system is not in the dead time:
   a. If salinity is greater than upper limit:
      Add fresh water: open valve, wait, close valve.
   b. If salinity is less than lower limit:
      Add salty water: open valve, wait, close valve.
3. Return to step 1.

Each of these steps needs to be refined. Note that these steps are different from the steps described in the overview document presented at the beginning of the class. The algorithm in the overview document were presented before we explained the need for deadtime.

Measure salinity and update the LCD panel

Each time the salinity is measured, the value displayed on the LCD panel should be updated. To make the display responsive to system changes, the salinity should be measured frequently. Although the salinity can be measured at any time, the decision on whether to act on the measurement needs to account for the deadtime.

If the system is not in the deadtime…

The simplest way to implement the deadtime delay is to use a command link

```
   delay(deadtime);
```

in your code. ***Don't do that!*** Using a simple to delay create the deadtime will block the program from doing any other useful tasks while it waits. By blocking execution, the code cannot update the LCD display. Later, when temperature control is added, blocking execution will prevent the temperature from being measured or updated.

Figure 10 shows the structure of an Arduino sketch that prevents the system from adding DI or salty water during the deadtime without blocking execution. The code uses a global variable `last_salinity_update` to store the time when the most recent change to the salinity was completed. The time is measured by the internal system clock, which is read with the `millis()` command. The decision on whether or not to allow salinity changes is made with the statement

```
   if ( ( millis() – last_salinity_update ) > deadtime ) {
```

The value of `(millis() – last_salinity_update)` is the difference between the current time and the last salinity change. The logic for changing the salinity is contained inside that "if" block.

Add fresh or salty water

Follow these steps

1. Compute the amount of water to be added from Equation (7). $X_{s,i}$ is the measured mass fraction, and X_{sp} is the set point.
2. Open the solenoid valve for a time period Δt given by Equation (8)

```
//  File:  wait_for_deadtime.ino
//
//  Structure of salinity control code to implement a deadtime
//  during which no salinity correction is made.  This code is
//  incomplete and will not compile.

unsigned long last_salinity_update;  //  Time of last correction

void setup() {
  Serial.begin(9600);
  last_salinity_update = millis();   //  Initial value; change later
}

void loop() {
  float LCL, UCL, salinity;
  int deadtime = ... ;

  salinity = salinity_reading( ... );
  update_LCD( ... );

  // -- Check for deadtime
  if ( ( millis() – last_salinity_update ) > deadtime ) {

    if ( salinity>UCL ) {
      //  add DI water:  steps are missing in this code
      last_salinity_update = millis();
    }

    if ( salinity<LCL ) {
      //  add salty water:  steps are missing in this code
      last_salinity_update = millis();
    }
  }
}
```

Figure 10 Arduino code to demonstrate how to test whether the system should ignore salinity errors during the deadtime.