Preparation for Salinity Control ME 121

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 develop a handful of simple models to characterize the behavior of the fish tank system. The models are not complete without measurements, and the data you collect will be specific to your fish tank. The following models are derived from measurements

- Calibration equations for the salinity sensor
- The control deadband
- The deadtime between salinity adjustments
- The flow rate through the solenoid valves

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 deadband
- 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.

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 deadtime 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 the data from that measurements to compute $\Delta UCL = \Delta LCL = 3\sigma$.

Table 1 Summary of calibration data for the salinity sensor.

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

$\Delta UCL = \Delta LCL =$ _____

2. Obtain piecewise linear regression for the inverse calibration

Use linear regression to determine the expected output in terms of the salinity. Make sure you have at least 5 digits for each of your curve fit coefficients. Figure 1 shows the qualitative appearance of the forward and inverse calibration relationships.

Write an Arduino function to return the calibration for salinity value as a function of the output of the salinity sensor. Figure 2 is an incomplete listing of such a function. The `rbreak` value is the value of the reading where the fit formulas change if you are using a piecewise calibration equation. The code inside the final `else` block should prevent bad input from causing trouble in the control algorithm.

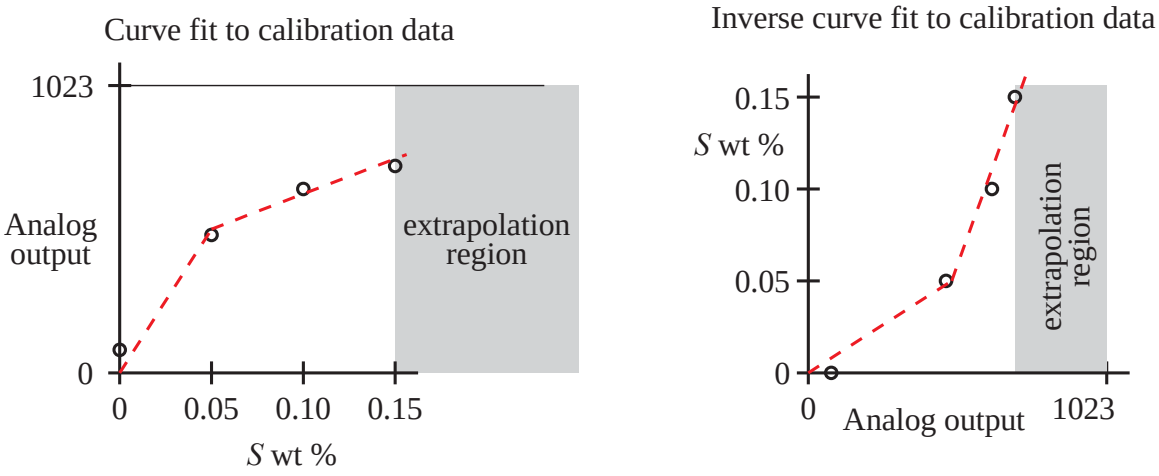

Figure 1 Forward and inverse calibration relationships. The inverse calibration is used to determine salinity from the reading of the voltage divider for the salinity sensor.

```
float salinity_value(int reading) {

    int rbreak = ...;     //  separation between linear segments
    int rlimit = 1023;    //  upper limit of acceptable readings

    if ( reading<rbreak ) {
        // do something

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

    } else {
        // do something to be safe

    }
    return(salinity);
}
```

Figure 2 Unfinished Arduino code for computing salinity as a function of salinity sensor reading.

3. Determine the deadtime compensation

The deadtime is the time interval between the introduction of a disturbance to the fish tank salinity, and the re-establishment of equilibrium. The deadtime of most fish tanks (for ME 121) is on the order of 10 seconds. 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. To measure deadtime, you need to set up an experiment to measure the salinity and print the raw analog reading to the Serial Monitor. The Arduino code for measuring the deadtime is not the same code that you use to control the salinity.

Deadtime measurement code

Figure 3 shows a simple Arduino code to print the time and salinity for each reading to the Serial Monitor. As a backup, make an independent measurement of the time delay with a stopwatch or cell phone timer. IMPORTANT: You must keep track of the time when the disturbance was made to the system, e.g., when the fresh or salty water was added.

```
//  File:  analog_input_with_millis.ino
//
//  Read an analog input and write the system time in milliseconds
//  and the analog input value to the serial monitor

unsigned long start_time;   //  Allows computation of time since
                            //  the start of sketch
void setup() {
  Serial.begin(9600);
  start_time = millis();    //  Time at beginning of the sketch
}

void loop() {
  int reading = analogRead(1);        // Reading on channel 1
  Serial.print(millis()-start_time);  // Time from start of sketch
  Serial.print("\t");                 // Tab character
  Serial.println(reading);            // Print reading and newline
  delay(10);
}
```

Figure 3 Arduino code to demonstrate printing of time from start of a sketch along with a value read from an analog input.

The deadtime measurement should allow you to collect a large number (say, at least 50) measurements of salinity. If you are averaging individual analog input measurements, we recommend that each averaged reading is completed in no more than one second. The time limit of 1 second per reading is an arbitrary but practical goal that provides a balance between capturing the system variability and not waiting so long that the system may drift. Use the following calculation as a template for estimating whether your code can produce at least one reading per second.

10 ms for each reading: 5ms settling time and 5 ms between readings

20 readings for each average

2 ms allowance for `Serial.println()` (conservative)

⇒ 202 ms per reading ⇒ ~5 readings/second

Of course, you should also verify with an experiment that your code produces at least one reading per second.

Estimating the deadtime from salinity measurements

Estimate the deadtime with the following procedure. The starting and ending values of mass fraction of salt should not matter. You are measuring the time it takes your system to respond to the change in mass fraction.

1. Choose a value of salinity near the middle of the operating range, say $X_s = 0.0010$. Run your system at that X_s until the output of the salinity sensor stabilizes.
2. With the system stable at the initial salinity value, start (or re-start) your salinity reading code to begin the data collection.
3. Add a small amount of water with a different mass fraction, say $X_s = 0.01$. Start the stopwatch, and note the output of the `millis()` command in the Serial Monitor window.
4. Wait for the system to come into equilibrium again. When it does, stop the stopwatch and record the time interval. Copy all of the data – from the first equilibrium condition to the second equilibrium condition – and paste into a data file for plotting and analysis.
5. Plot the raw readings versus time, and use your timing measurement to determine the deadtime.

Refer to the plot in Figure 4 as an example of a deadtime measurement. Note that the deadtime is *not* equal to the time during which the output from the salinity sensor is changing from one nominal value to another, i.e., it is not just the steep ramp interval in Figure 2. *The start of the deadtime interval occurs before the salinity reading changes significantly*. Therefore, the start time *must be recorded by your observation* of when the disturbance began. It cannot be determined solely from the plot.

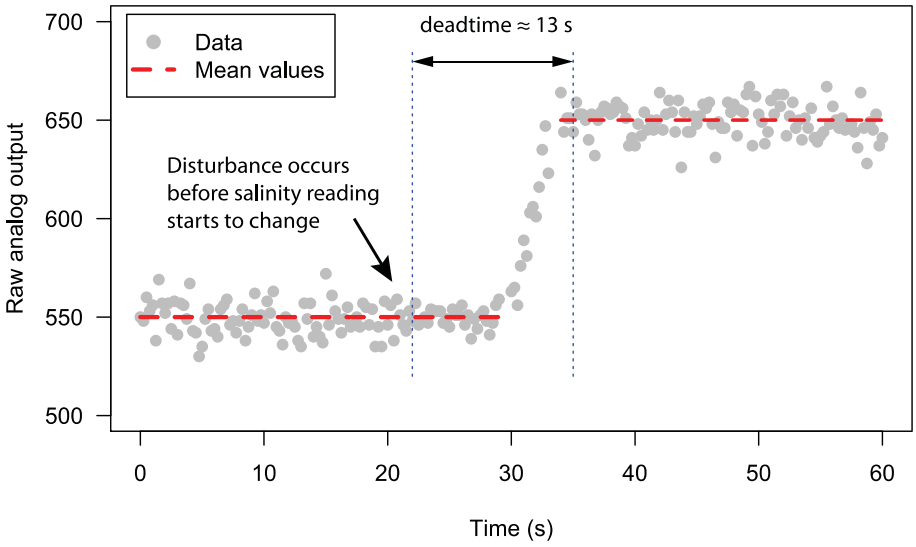

Figure 4 Typical result of deadtime measurement. Gray dots are measurements and dashed read lines are averages of measurements

4. Determine the flow rate through the solenoid valves.

Measure the amount of water that flows from the top reservoir when the solenoid valve is open for 1 second. Use a delay command to repeat this measurement *at least* 10 times. Use the scales to weigh the water. Assume both valves are the same unless you have enough time to check the valves separately. 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. It would be a good idea to use a spreadsheet to store the data and perform the conversion and averaging.

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

Reading #	Mass (g) (valve 1)	Δt_1 (s)	Mass (g) (valve 2)	Δt_2 (s)	$\dot{m}_1$ (g/s)	$\dot{m}_2$ (g/s)
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
				Mean		
				Std. dev.		

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 $0.0005 \le X_{sp} \le 0.0010$. 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:

$$\text{LCL} = X_{sp} - \Delta\text{LCL} \tag{1a}$$

$$\text{UCL} = X_{sp} + \Delta\text{UCL} \tag{1b}$$

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

$$E = X_s - X_{sp} \tag{2}$$

where E is dimensionless, and is a measure of mass fraction difference. 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 5 shows a schematic of the batch model for correction of the salinity of water in the tank.

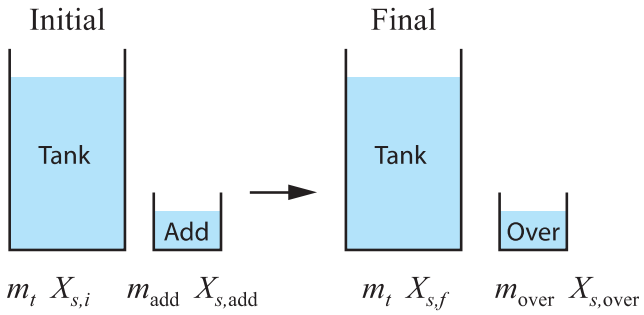

Figure 5 Schematic of the mass addition to correct an error in the tank salinity.

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
$X_{s,add}$	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, $m_{over} = m_{add}$
$X_{s,over}$	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
$\dot{m}_{sol}$	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

$$X_{s,over} = (1 - F)X_{s,i} + F\,X_{s,add} \tag{3}$$

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

$$X_{s,f} = X_{s,i} + G\,(X_{sp} - X_{s,i}) \tag{4}$$

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

$$m_{add} = m_t \frac{G(X_{sp} - X_{s,i})}{(1-F)(X_{s,add} - X_{s,i})} \tag{5}$$

The time that the solenoid is open is

$$\Delta t = \frac{m_{\text{add}}}{\dot{m}_{\text{sol}}} \tag{6}$$

where $\dot{m}_{\text{sol}}$ is the mass flow rate through the valve when it is open. Equations (1), (5) and (6) 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 6. 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.

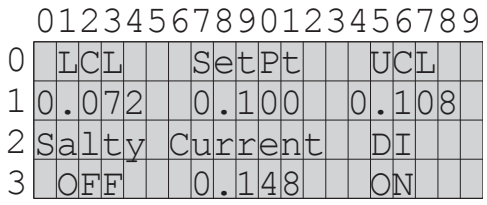

Figure 6 Layout of the LCD display for the salinity control verification.

Display of floating point values with the LiquidCrystal library is not straightforward. A detailed example of the problem and solution is provided on the `howto` web page for the class:

`http://web.cecs.pdx.edu/~eas199/B/howto/LCDwiring/arduino_to_LCD.html`

The trick is to use the built-in `dtostrf()` function to convert the floating point value to a a string, and then display the strain on the LCD panel with `lcd.print()`. Figure 7 lists an excerpt of code that can display the second line in Figure 6.

```
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 7 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, and using SP as a variable name produces a hard to find 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 delay to implement the deadtime blocks 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 8 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 (5). $X_{s,i}$ 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 (6)

```
//  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 8 Arduino code to demonstrate how to test whether the system should ignore salinity errors during the deadtime.