Homebrew CTD

Building a CTD with simple to use and readily-available consumer parts

Sophie Scopazzi

Homebrew CTD

Building a CTD with simple to use and readily-available consumer parts

by

Sophie Scopazzi

Instructor:	A. Cifuentes-Lorenzen
Project Duration:	Sept, 2020 - Dec, 2022

Cover:	The CTD in the water, sampling
Style:	EPFL Report Style, with modifications by Batuhan Faik Derinbay

Preface

This manual began within an Introduction to Oceanography Lab with Dr. Cifuentas-Lorenzen at California State University Maritime Academy during the Spring 2020 semester. The original iteration was the same in principle, but used different sensors [1]. During the class, I got these sensors to work and the data to save to an SD card, but stopped short of a water-tight housing - everything worked as intended on the bench. However, the aforementioned sensor's compatibility for a watertight enclosure left me wanting something better.

Fast forward. I took the entire setup home during the COVID-19 stay-at-home orders and got busy with life, this project never truly escaping me. Seeing the project sit in its plastic tub on my shelf, I finally returned September 2022 to finalize the project. I decided that doing so required different sensors (detailed in this manual), so set about making it happen.

While it is of course possible to buy an off-the-shelf CTD, doing so does not afford the level of education equivalent with building own's own (nor is it as fun!). The learning and problem solving involved with coding, soldering, connectivity of sensors, building a housing, and more, are all invaluable educational experiences that are part and parcel to this project.

I make this manual to document what I have made in the hopes of creating easy to follow instructions anyone with basic tools can follow to create their own - for themselves, or in a classroom setting.

Sophie Scopazzi
Cal State Maritime, April 2024

[1] Waterproof 1-Wire DS18B20 Digital temperature sensor, SparkFun Pressure Sensor Breakout - MS5803-14BA

Contents

Preface i

1 What is a CTD? 1

2 Design Objectives 2

3 Equipment Used 3
- 3.1 Sensors . . . 3
- 3.2 Sensor Accuracy . . . 3
- 3.3 Datalogger . . . 4
- 3.4 Power . . . 4
- 3.5 Housing . . . 5
- 3.6 Misc. Equipment . . . 5
- 3.7 Overall Cost . . . 5

4 How to: Breadboards, Soldering, Coding 6

5 Atlas Scientific EZO-EC Embedded Conductivity Circuit 8
- 5.1 Wiring . . . 8
- 5.2 Calibration . . . 9

6 Blue Robotics Bar30 High-Resolution 300m Depth/Pressure 10
- 6.1 Accuracy . . . 10
- 6.2 Wiring . . . 10

7 Adafruit MicroSD Card Breakout Board 11
- 7.1 Wiring . . . 11

8 Power Supply 12
- 8.1 Battery Life Calculation . . . 12
- 8.2 Anker PowerCore 10000 Portable Charger . . . 12
- 8.3 Blue Robotics Pressure Switch . . . 12

9 3D Printing: Trays, Stands and Holders 13
- 9.1 3D Printed Equipment Tray (4x) . . . 13
- 9.2 3D Printed Equipment Tray Holder (2x) . . . 14
- 9.3 3D Printed Housing Holder (2x) . . . 14
- 9.4 3D Printed Flange Holder . . . 15
- 9.5 Misc. Small 3D Printed Parts . . . 15

10 Housing 16
- 10.1 Pressure Ratings of Parts . . . 16
- 10.2 Building the Housing . . . 17
- 10.3 Flange . . . 17
- 10.4 Flange Without Machining Tools . . . 18

11 Deploying the CTD 19

12 Data Accuracy 21

13 The Arduino Program 26

14 Takeaways and Final Thoughts 27

A Wiring Diagram 28

B Arduino Program 29

C MATLAB Program **32**

1

What is a CTD?

”CTD” is an acronym that stands for **C**Conductivity, **T**emperature, and **D**epth due to what the instrument measurements. The salinity value is calculated from the conductivity of the water, as the salinity of water changes its conductivity. Using a CTD, it is possible to plot the salinity and temperature of a water column by the depth.

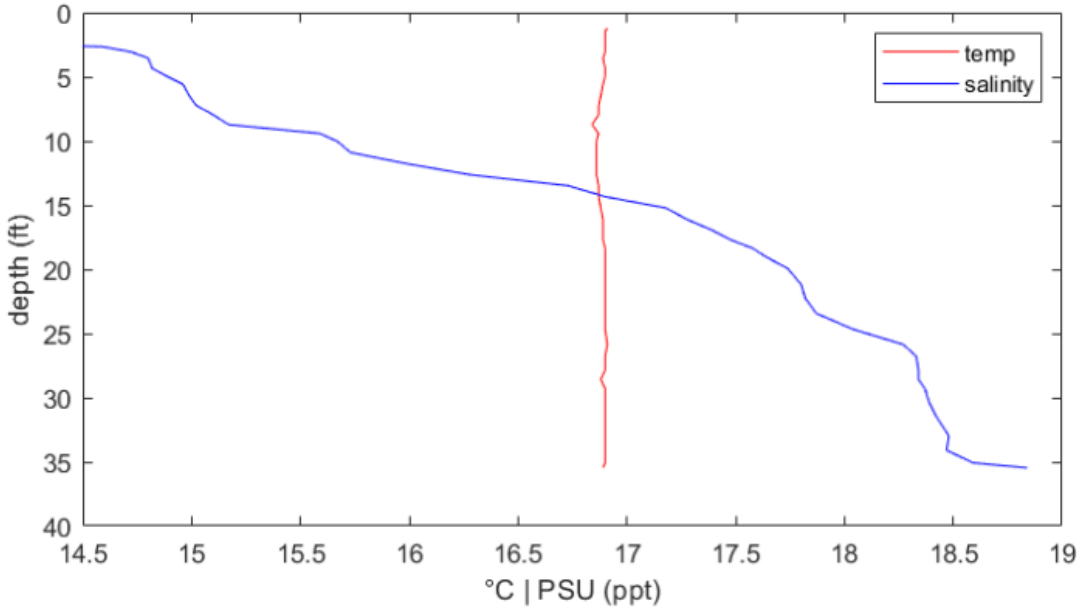


Figure 1.1: Example of what CTD data looks like using test data from the device built

This instrument, which supplies what may seem like only a few data points, can help answer a lot of questions about the water column. Using the information from a CTD it is possible to infer further and answer many questions such as, ”Is the water column stable and not mixing, or unstable and prone to mixing?” and, ”What is the depth of the thermocline or halocline?” From this information it’s possible to look further into the life living in the epipelagic zone (the top photic zone, where there is light in the water column).

Another important use for CTD data is calculating the speed of sound through the water column, as the temperature of water effects the speed of sound through the medium. This knowledge is vitally important for multi-beam sonar mapping of the seafloor.

2

Design Objectives

My goals were twofold: 1) A more cost-accessible sensor package still delivering on data fidelity requirements and 2) creating an accessible manual detailing how to build it, intended for use by individuals or teachers as part of a class, but also for anyone else who wants to make one.

1. Cheaper than pre-built existing CTDs currently on the market, ideally under $700 USD.
2. Built with basic and accessible tools – soldering, 3D printers, basic shop tools (like those found in a college makerspace).
3. Profile to at least 150ft.
4. Profile continually for twenty-four hours and/or able to observe at least one complete tidal cycle.
5. Acceptable sensor accuracy (given overall cost constraints):
   (a) **Pressure:** 2 inch resolution in water column (dZ)
   (b) **Temperature:** ±1°F
   (c) **Salinity:** ±2-3%

3

Equipment Used

3.1. Sensors

Part	Cost
1. Atlas Scientific Conductivity Probe K 10	$183
2. EZO™ Conductivity Circuit	$68
3. Blue Robotics WetLink Penetrator	$12
4. Blue Robotics Bar30 High-Resolution 300m Depth/Pressure Sensor	$85
Total	**$348**

1. This can be bought in a kit or separate like this. Pick the one that has the best use case for you (range of values it reads, etc.).
2. Needed to connect to the Arduino.
3. Allows the sensor from the Conductivity Probe to go through the watertight hull.
4. Pressure and depth sensor in one. If your application requires greater than ±1°C, get the Blue Robotics Celsius Fast-Response, ±0.1°C Temperature Sensor (I2C). I might incorporate this in the future.

3.2. Sensor Accuracy

The design objectives were selected with certain sensors in mind to keep the project cost-realistic.

- **Pressure:** 0.2mbar resolution, allowing for a depth resolution of 2mm, within our design objectives.
- **Temperature:** ±1°C, outside our design objectives. The temperature sensor is integrated with the depth/-pressure sensor, and as such is only accurate to ±1°C. This is outside the original design objectives. If a greater temperature accuracy is required an alternative to the integrated temperature sensor is Blue Robotics' Celsius Fast-Response, ±0.1°C Temperature Sensor (I2C). If your budget allows, I'd recommend using this sensor for temperature, but if your project doesn't need anything more accurate than ±1°C don't worry about it. As it is, I accepted the trade-off of accuracy and cost.
- **Salinity:** With an accuracy of ±2% this sensor is within our design objectives. As stated before, you will need to decide which salinity range you wish to purchase for your use-case. It is important to note that the sensors documentation gives a 90% accurate response time in one second. This means the slower the CTD is lowered in the water column the more accurate this sensor will be (more on this later in Section 11).

3.3. Datalogger

Part	Cost
5. Arduino Uno R3	$28
6. Screw Terminal Block Breakout Shield Module for Arduino UNO R3	$21
7. Adafruit MicroSD card breakout board+	$8
8. SD/MicroSD Memory Card (8 GB SDHC)	$10
9. Blue Robotics I2C Level Converter	$25
Total	**$92**

5. Arduino Uno R3 is cheap, easily found, and has a lot of supporting documentation online. The Arduino IDE is free and relatively user friendly.
6. Allows for ease of connecting wires. It is not strictly necessary but makes connecting power to everything especially easy.
7. Allows the Arduino to save sensor data onto an SD card.
8. Where the data is saved.
9. Converts and connects the depth / pressure sensor to the Arduino.

3.4. Power

Part	Cost
10. Anker PowerCore 10000 Portable Charger	$26
11. Blue Robotics Pressure Switch	$20
12. Adafruit 1 Watt Cool White LED - Heatsink Mounted	$4
Total	**$50**

10. Using a normal USB power bank to power the entire package makes it easy to charge, and supplies plenty of power.
11. The way to turn the CTD on and off that works under pressure. The only problem I've had is sometimes it is difficult to turn off (unscrew) after the CTD is at depth. It's also possible to make a magnetic switch.
12. This is needed because the power draw is so low the USB battery bank goes into sleep mode and stops supplying power. I had this LED lying around from another project and threw it on, and it worked. The light is a good visual, especially in the water, that the CTD is on. If you have some kind of LED lying around, try it before buying this.

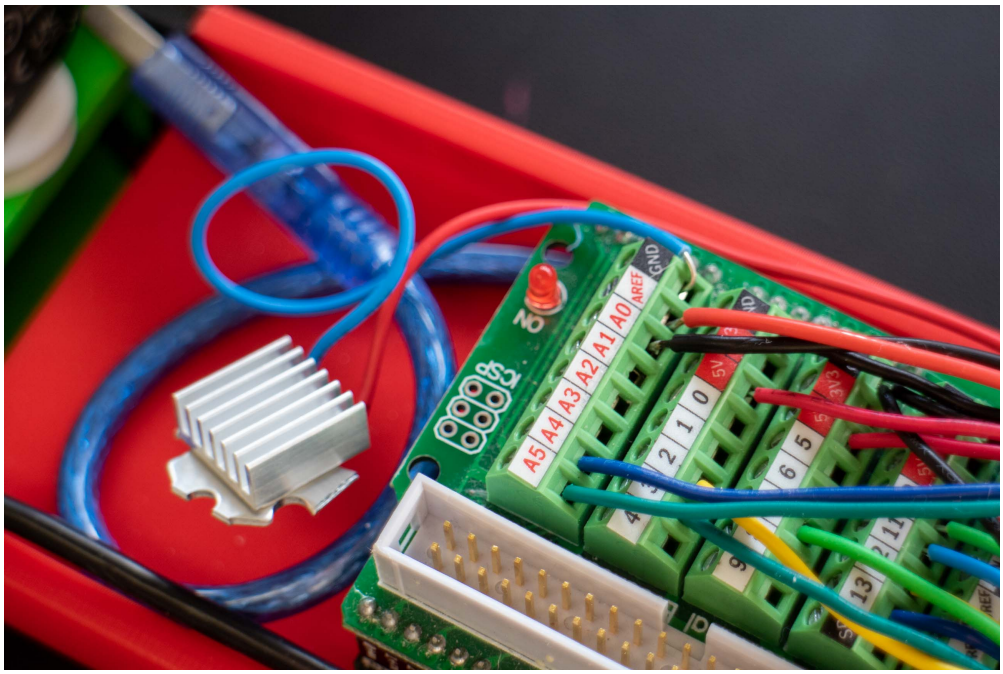


Figure 3.1: The LED is 5v and I put a small heat sink on it

3.5. Housing

Part (McMaster part #)	Cost
Thick-Wall PVC Plastic Pipe Fitting for Water (4881K218)	$30
Water- and Steam-Resistant EPDM Gasket (1082N15)	$5
Standard-Wall Clear Blue Rigid PVC Pipe for Water (49035K49)	$61
Pipe Cement for Plastic Pipe (74605A14)	$10
Steel Oval Eye Nut - for Lifting (3274T42)	$12
Low-Pressure Steel Unthreaded Pipe Flange (68095K382)	$42
Zinc-Plated Steel USS Washer (90108A032)	$18
Standard-Wall PVC Pipe Fitting for Water (4880K57)	$6
Medium-Strength Grade 5 Steel Hex Head Screw (91247A726)	$13
Zinc Yellow-Chromate Plated Hex Head Screw (91257A720)	$13
Medium-Strength Steel Hex Nut (95462A033)	$30
250-Feet Hollow Braid Polypro Rope, Yellow (Amazon)	$16
Total	**$256**

This is designed for my application of at least 150ft. It has a theoretical max depth of 300ft based on the minimum specification of the parts (the clear PVC pipe). You could take this list as is or look through designing another housing – especially if there is a greater or lower max depth desired. If you design another, please reach out and let me know what you make! Lastly, the PVC doesn't have to be clear, but I felt seeing the insides was more important and didn't sacrifice depth.

3.6. Misc. Equipment

Part	Cost
Soldering iron and solder (any kind)	$105
Heat Shrink Tubing Kit - 2:1 Shrink, 300W Mini Heat Gun	$14
Hook-up Wire Spool Set - 22AWG Solid Core - 10 x 25ft	$35
Wire strippers / cutters (any kind)	$14
Standard Jumper Wires Plus Set Solderless Breadboard	$11
Safety glasses or goggles	$10
Total	**$189**

This section is included for the benefit of anyone who does not have access to the above. Already having these parts from previous projects, or because you have access to a space that has these things, would lower the cost further.

3.7. Overall Cost

Entire overall cost: $925
Cost of everything minus misc. equipment: $736

This value may appear costly, but in comparison it is relatively cheap, especially if one already has access to soldering equipment and hardware for the housing (for example, in a Makerspace, machine shop, or lab space on a college campus).

**** I used a lathe to face (thin) the flange plate then a mill to drill the through-hulls and wells to hold the sensors. This is difficult to put a price on. If you do not have access to tools such as these, I recommend getting a bit more creative with the flange (see Section 10.3-10.4) ****

4

How to: Breadboards, Soldering, Coding

Breadboards

Breadboards are for temporary testing of wiring and are connected as shown in fig. 4.1.

The holes down the sides with red and blue colors go all the way down the board, to be used as power/hot (red) and ground (blue). The inside is connected as shown and is where components go. Common errors are not putting the wires into the row with a component and mixing up hot and ground connections. Staying consistency with wire colors is incredibly helpful.

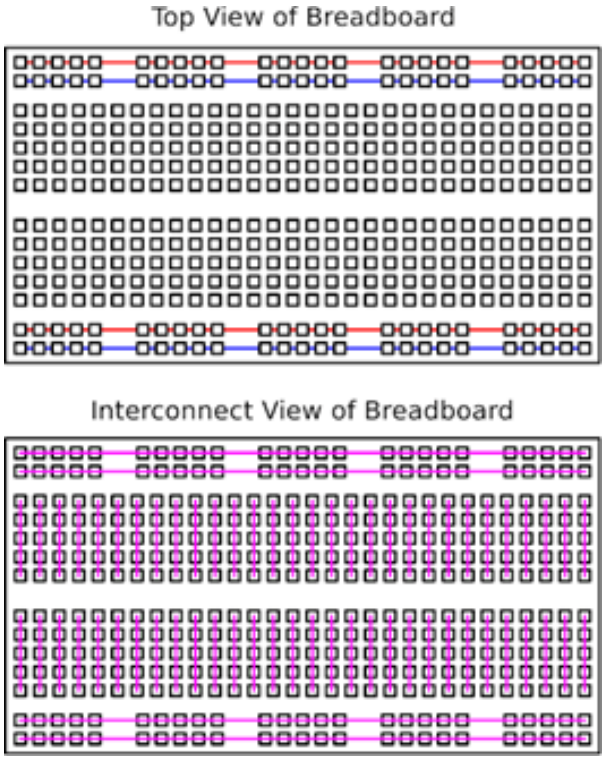


Figure 4.1: How a breadboard is connected

Soldering

Safety - Always wear safety glasses when soldering! Small pieces of flux or solder can fly off and get in your eyes. Just wear them. It is not worth the potential for eye problems. Do not breath in the solder smoke as it is the flux on the solder burning. It will not kill you quickly, but is not good for you either.

General - Given the choice, be consistent with wire colors. Future you will thank past you. Red = hot, power, voltage. Black (or blue if don't have black) = ground.

Stripping and Soldering - Use wire strippers of the correct size. Expose only about 5mm of wire. Before soldering the wires slip on the heat shrink. Put into the board and solder. It should look something like the photo shown (fig. 4.2). Clip the extra wire and inspect to be sure you did not inadvertently bridge any connections. If you did, unsolder using a vacuum gun or copper solder wick and redo.

Software

Programming is accomplished in the *Arduino IDE software* on Windows, Mac, or in a web browser. The latter is not recommended. Use the downloaded software if at all possible. Using the version on the web can create all kinds of headaches.

Unable to connect to board - COM PORT error

1. Ensure the USB is plugged into Arduino and computer.
2. In the Arduino IDE program navigate to 'Tools' –> 'Port' –> select 'Arduino UNO'.
3. Try a different cable, unplug/replug, or restart the program/computer

Coding

Common errors when coding are below. Working through understanding code can be difficult. Thankfully, Arduino has many wonderful examples and *language reference guides*.

1. Not having a ";" at the end of the line.
2. The baud rate stipulated in the code is different than the baud rate selected in the serial monitor.
3. Missing a capital letter, or having one where there must not be.

Helpful commands

```
// 'Comment' out the line with two "//" to add words without having them run as code
define rx 2; //define what pin receive is going to be, 'tx' would be transmit
Serial.begin(9600); //sets baud rate for the hardware serial (select this in the serial monitor)

//leave concise notes in your code ensuring you can go back and know what it is doing
```

The Main Arduino Program and Other Code

Start with the supplied code in the appendix or within the zip file for individual sensor code and the final deployment code *available on my website*. Each sensor has its own individual code that can be used for testing on a per-sensor basis supplied by the manufacturer, but I've included them in the zip file for ease.

5

Atlas Scientific EZO-EC Embedded Conductivity Circuit

This sensor gets TDS (Total Dissolved Solids), EC (Electrical Conductivity) and the Salinity (in Practical Salinity Units, ppt).

5.1. Wiring

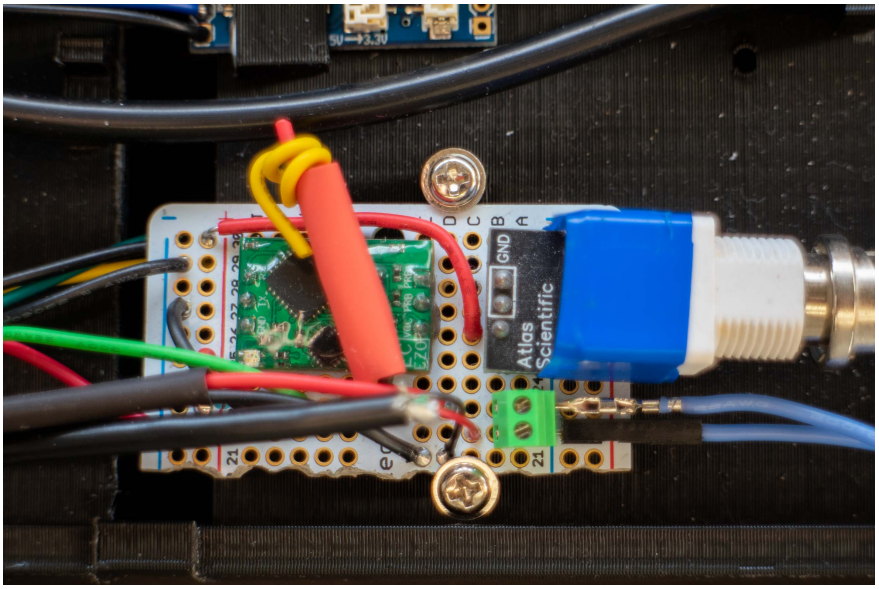


Figure 5.1: Salinity sensor power wiring. Red wire between screws.

There is an important part of the wiring missing from the diagram found in the manual, that is explained in the datasheet for the sensor. **Both parts of the equipment need 5V power**, not only the part in the green diagram. I recommend connecting it the same way as shown in figure 5.1, so both pieces get 5V power. The red bent wire close to the board is supplying power from the hot on the left to the row both of their power pins is in.

It might not be clear right away which pin is the center and shield, so I recommend laying them out in a breadboard as in the photo, and testing, *before soldering*. I also recommended, once you get to the soldering stage, to electrical tape around the sensor's BNC connector as shown in fig. 5.1 (the blue tape) to avoid any shorts or potential issues from the bottom of the board touching the perma-proto breadboard.

To get the salinity sensor's wire through the thru-hull, you will need to cut its wire. This may feel scary, but take your time, be careful and it will be okay. I recommend cutting the sensors cable about 3̃in from the sensor itself. Depending how confident you are with your cutting and soldering skills, you could give yourself less to work with, but I wasn't willing to take that chance with the most expensive sensor, which is why you see my cable so long in the photos. Once you have put the wire through the Blue Robotics thru-hull hardware, solder the wires with their matching color, then cover in heat shrink (don't forget to put the heat shrink out of the way on the wires before soldering).

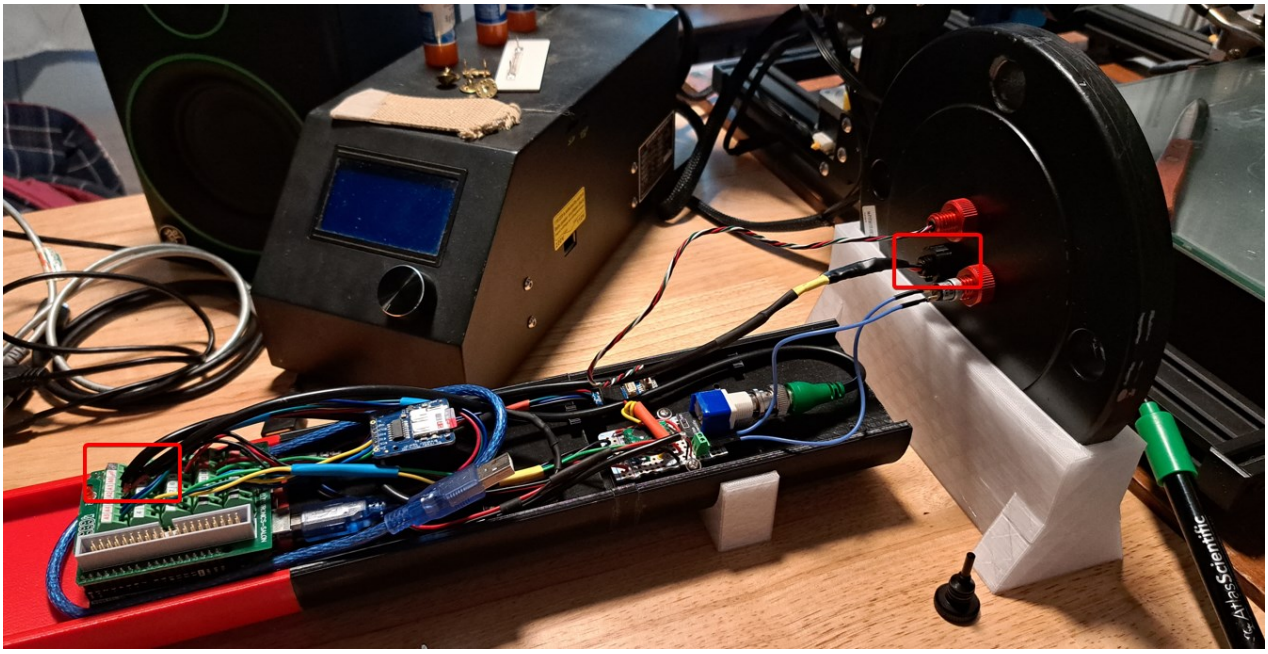


Figure 5.2: Salinity sensor wiring, parts in red boxes.

To ensure it is possible to fully remove the flange and easily access the electronics, the cable must be removable. This can be accomplished by the BNC connection, but I chose not to do so. The soldered connection holding the BNC connector to the perma-proto breadboard is not the strongest of connections and I am worried about breaking the soldering so I opted for a different solution. Using empty analog ports on the Arduino shield (A1 and A0) since they are unused terminals). I screw in the matching wires from the BNC connector and the sensor from the flange, making a temporary connection using the screw terminals. This affords the ability of easy removal without buying different parts (this is similar in idea to the power switch, as detailed later).

5.2. Calibration

Using the code salinSensor_test.ino, follow the instructions found in the EZO-EC Conductivity Circuit Datasheet to calibrate the sensor using the correct liquids. Once you have calibrated your sensor be sure to label the sensor and components, (if you have multiples) so you always use the same sensor and board.

6

Blue Robotics Bar30 High-Resolution 300m Depth/Pressure

This sensor gets depth (m, can convert to ft) and temperature (in °C, can convert to°F).

6.1. Accuracy

- Temperature sensor accurate to ±4°C
- 0.2 mbar (2mm) depth resolution

As per the Blue Robotics' website, "This sensor must be completely dried once per day for at least 2 hours, or the pressure and temperature readings will drift".

6.2. Wiring

Please follow the extremely well-made tutorial available on Blue Robotics' website. They've done a stellar job and there is no use repeating it here.

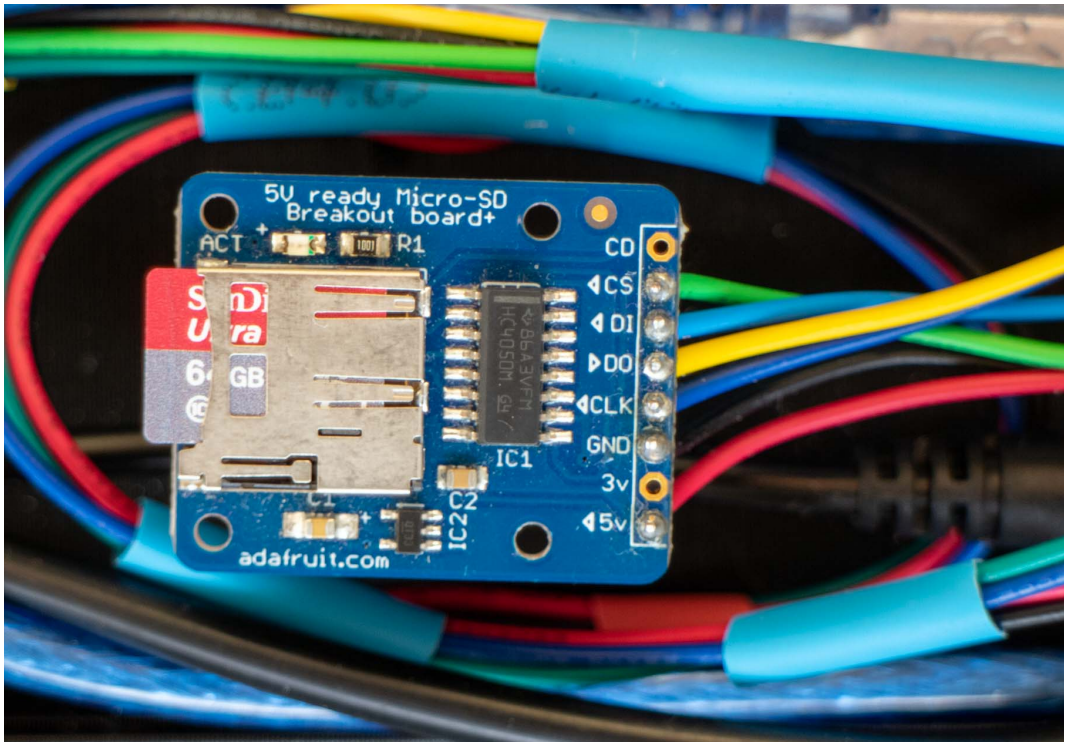
7

Adafruit MicroSD Card Breakout Board

Cheap and easy to physically connect, this allows the Arduino to save the sensor data to a micro SD card.

7.1. Wiring

Like the Blue Robotics' sensor, Adafruit has a well made tutorial found on their website. And to reiterate, make your life easier and, given the choice, do not use the same color wires for everything.



8

Power Supply

8.1. Battery Life Calculation

As designed the entire CTD package uses approximately 340mAh (0.34Ah) of power. This means, using the following formula (which assumes a perfect scenario, which is never the case), the CTD can be deployed for approximately [THIS MANY] hours (rounded down).

- **Formula:** $Time(H) = Capacity(Ah)/Current(Amps)$.
- **Plain English:** Time in hours = battery capacity in Ah divided by the current draw in amps.
- **Calculation:**
  $Time(H) = 10Ah)/0.31A$
  $Time(H) = 29.42$

The power draw without the 1W LED was actually so low the power supply would auto power off after thirty seconds, not sensing enough power draw to stay ”awake”. It uses more power, but there is still enough power to last at least twenty-four hours.

8.2. Anker PowerCore 10000 Portable Charger

Using an off-the-shelf USB power supply was selected because 1) they are cheap and readily available and 2) are easy to charge without specialized battery chargers, making them easier for folk not experienced with more specialized batteries (such as those found in hobby drones). I used a Velcro command strip to attach the battery to the tray, for easy removal if required.

This specific battery was selected for the following reasons:

1. With a capacity of 10,000mAh (10Ah), it has plenty of power to outlast the design objective.
2. This specific model fit within the PVC pipe, and even the tray, making it easy to fit.

8.3. Blue Robotics Pressure Switch

This switch allows turning the CTD on and off in the field. This not only saves battery, but also simplifies the data back-end by not having stretches of meaningless data between deployments.

The switch is a bit overkill for this situation, able to handle up to 5A of current and 120V AC or 26V DC, but it is pressure rated and quite up to the task, while remaining relatively inexpensive.

Much like the salinity sensor, to ensure the possibility of fully removing the flange, I had to get a bit creative as to how to integrate the switch into the power circuit. I settled on a small screw terminal block where I could attach both ends from the switch (see section [SOMETHING] for the wiring diagram).

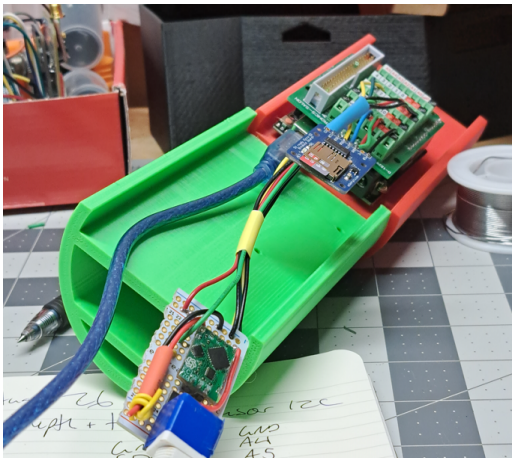
9

3D Printing: Trays, Stands and Holders

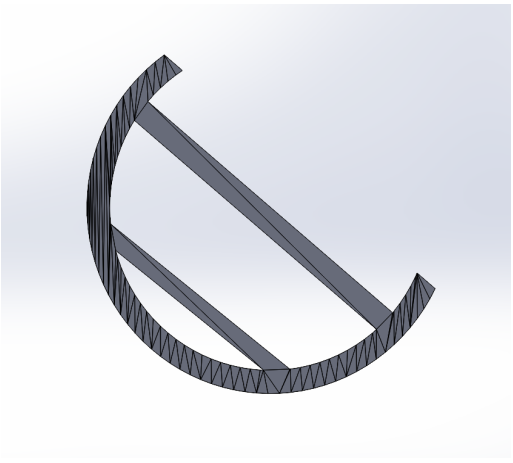
All files explained below can be found in the zip file downloadable from *my website* (https://www.sophiescopazzi.com/projects

9.1. 3D Printed Equipment Tray (4x)

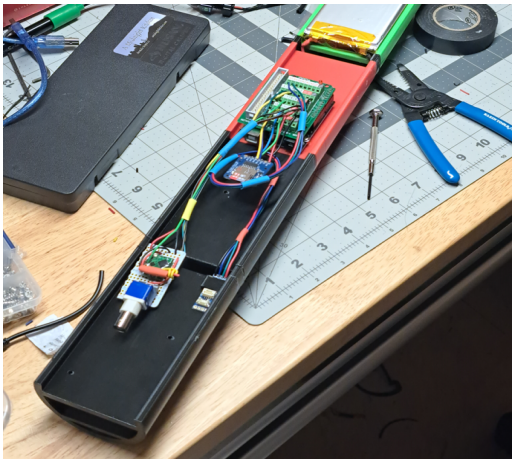
To make deployment and working on the equipment easier I recommended printing an equipment tray to secure the equipment to.



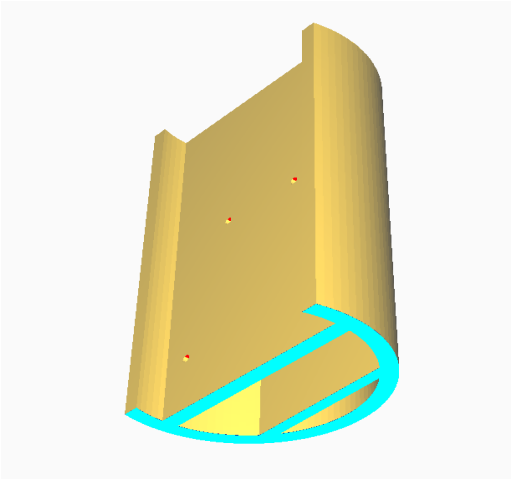
(a) First two I printed



(b) Bottom of print view



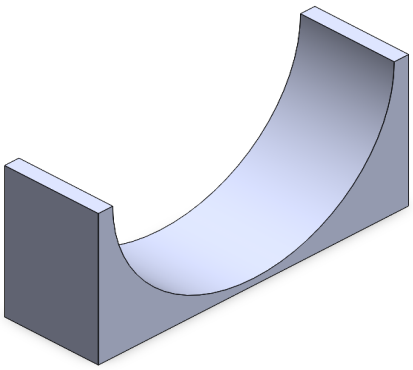
(c) Added two more (black ones)



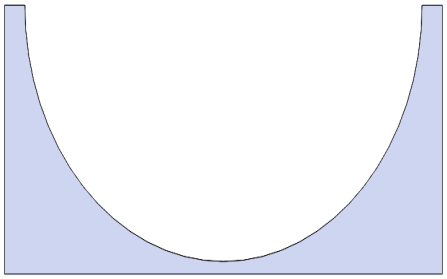
(d) From Ultimaker Cura

9.2. 3D Printed Equipment Tray Holder (2x)

For holding the equipment tray when it is not in the housing. It is much easier working on the tray when it's not rolling around (I did not use this for a bit and got fed up until I made this).



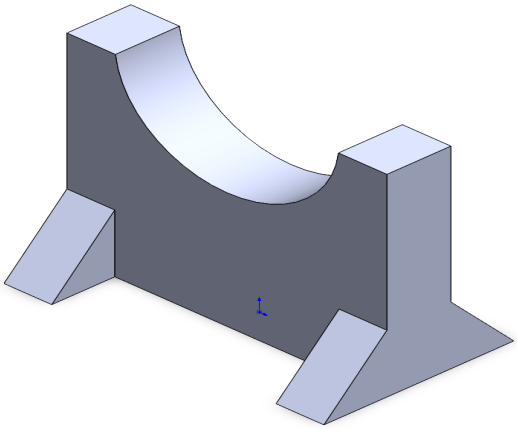
(a) Isometric



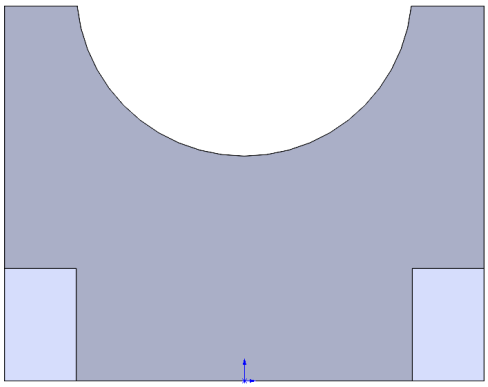
(b) Side

9.3. 3D Printed Housing Holder (2x)

Used to hold the housing while it is on the worktable. I highly recommend printing it! Same as before, I made it out of work-ability necessity (and it helps the CTD look good in the classroom / on display).



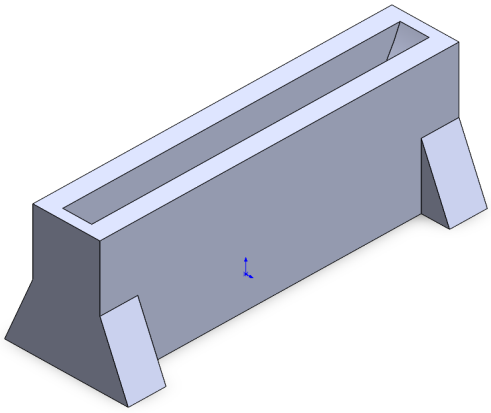
(a) Isometric



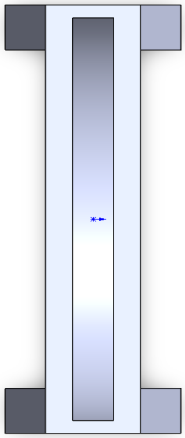
(b) Side

9.4. 3D Printed Flange Holder

I highly recommend printing this, or something like this to hold the flange and sensors safely on the table while you work on the CTD. Print this file about 104%.



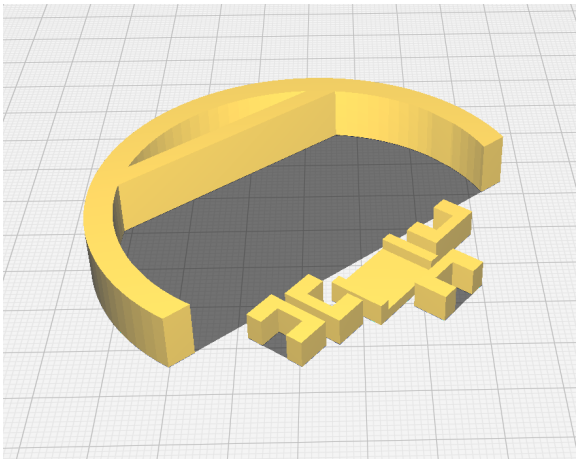
(a) Isometric



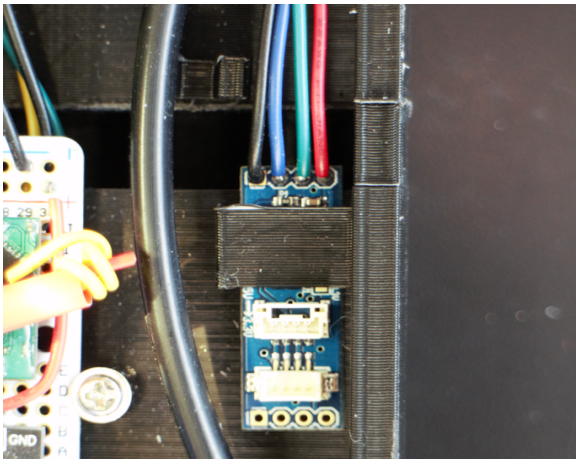
(b) Top

9.5. Misc. Small 3D Printed Parts

I 3D printed small parts to hold cables and parts in place. Depending how you choose to organize the internals, you may or may not need these, or will need to design your own.



(a) From Ultimaker Cura



(b) Holds the I2C converter down and cable (cable out of connector in photo)

10

Housing

10.1. Pressure Ratings of Parts

The minimum pressure rating for any one part is the clear PVC at 130psi, allowing for a theoretical max depth of 300ft (91.5m) (Blue Robotics' Water Pressure and Depth calculator). If greater depth is desired, or you simply do not feel the need to use a clear pipe and see the inside (clear is also more expensive), I'd recommend buying Standard-Wall Unthreaded Rigid PVC Pipe for Water (3" Pipe Size, 5 Feet Long). It is rated for 260psi, matching the PVC pipe fitting, leaving the limiting part the salinity sensor at 200psi, allowing for a theoretical deployment depth of 472ft.

I decided to use the Clear Blue Rigid PVC Pipe because 1) it met the 150ft profiling criteria and 2) it affords the ability to see the internals, creating peace of mind during deployments. While the clear PVC has a lower pressure rating compared to it is opaque cousin, I felt the trade-off was acceptable.

1. **Standard-Wall Clear Blue Rigid PVC Pipe for Water - 300ft, 91m**

   130 psi @ 72° F. Schedule 40.
2. Thick-Wall PVC Plastic Pipe Fitting for Water - 659ft, 200m

   285 psi at 72° F for steel. Schedule 80. Pressure class 150.
3. Water and Steam Resistant EPDM Gasket - 1,850ft, 564m

   800psi.
4. Low-Pressure Steel Unthreaded Pipe Flange - 659ft, 200m

   285 psi @ 72° F.
5. Standard-Wall PVC Pipe Fitting for Water - 601ft, 183m

   260 psi. Schedule 40.
6. Conductivity Probe K 10 - 462ft, 141m

   200psi.
7. Blue Robotics Pressure Switch - 3,116ft, 950m

   200psi.
8. Blue Robotics WetLink Penetrator - 3,116ft, 950m

   200psi.

10.2. Building the Housing

Below are the steps to make the housing exactly like I did. If you have better ideas for how to build the housing, or want to get different parts, absolutely experiment!

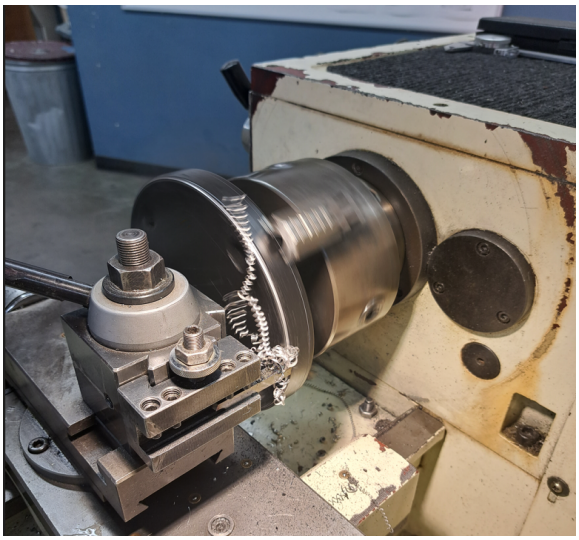
1. Ensure the connection areas of the cap, tube, and plastic flange are free from oil, particulate, or other matter. Once you have cleaned the connection areas, don't touch them with your hands (will leave oil from fingers).
2. Put on disposable gloves (do not get the PVC pipe cement on your skin). Fully cover the inside of the cap with the glue, then seat it fully onto the tube.
3. Have a paper towel or other disposable rag handy. Cover the inside of the plastic flange and seat it onto the plastic tube. If there is glue dripping into the flange area (there probably will be) wipe with rag before it dries. This is to ensure the circular area of the flange that protrudes from the main body presses into the rubber gasket used to seal the housing.

With that, almost all the housing is made. The last (and what I would call the most complex) part of building the housing is the part bolted to the housing to seal the watertight container - the metal flange.

10.3. Flange

I decided to go with a steel pipe flange, as 1) I had the resources available to machine it to desired specifications and 2) I figured it would be heavy enough to help the CTD sink without having to add additional ballast.

Using the Machine Shop at CSU Maritime, and the invaluable expertise and assistance of Mr. Long, we first used a lathe to face the flange, making it thinner - original 24mm, new 15mm. This step had two purposes: 1) make the part lighter, but more importantly 2) thinner, so the Blue Robotics through-hull bolts would properly fit. We quickly realized thinning the flange to 5mm was impractical on the lathe, due to the amount of time it would take to remove that amount of material.

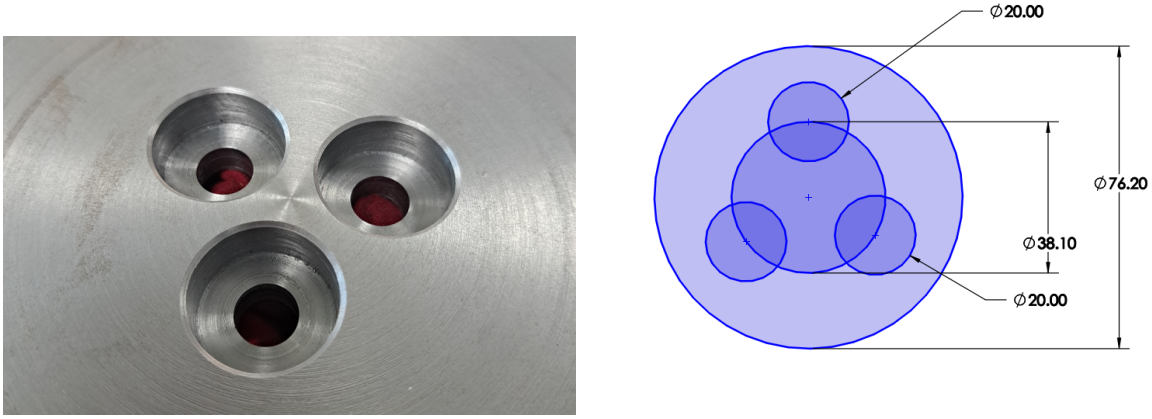


(a) Lathe

(b) Mill

Figure 10.1: Machining the flange

Instead, we moved to the mill with the idea of creating wells for the thru-hulls. In this way, we removed far less material for the same desired function. I began by using SolidWorks to model the layout for the holes and wells, ensuring everything would remain within the 3" of the PVC pipe (fig. 10.2(b). There are many ways to accomplish a diagram such as this (hand drawing, SketchUp or any other CAD program, and I'm sure more I'm not mentioning. I was/am in the process of getting better at using SolidWorks so I used it).



(a) Final flange

(b) Solidworks diagram

Figure 10.2: Closeup of flange holes, SolidWorks file

The holes in the flange, centered in the well, need to be made to a tight tolerance so the Blue Robotics' through-hulls will fit. They also cannot be too wide to afford a proper seat for the silicone gasket.

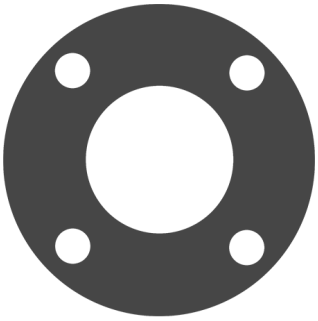


Figure 10.3: Gasket

After the first test of the CTD, to prove it did not leak, I cleaned the flange with fresh water, dried, scrubbed any rust-appearing parts, then coated it with three coats of spray paint to protect the flange from salt water and rust. I chose pink because it stands out in the water (and I like the color and had it from another project). Given the choice, I would recommend picking a bright color like yellow or pink.

10.4. Flange Without Machining Tools

If you do not have access to machining tools, I have a few suggestions, but also encourage creativity. If using plastic or thin metal, you will most likely need to ballast the CTD so it actually sinks. I was unsure exactly how it would float or sink, since I chose to not calculate prior to building, deciding to figure that out if it became an issue. As it is built, the CTD sinks flange first. I do not mind this, as it affords the sensors direct contact with the water column during its downward travel. Two ideas:

1. A steel plate of sufficient thickness for the depth rating desired, cut to size with an angle grinder, then drilled with a drill press or even by a hand drill, could work as a flat flange. This would change the bolts.
2. Laser cut, or otherwise cut in some manner, plastic of sufficient thickness for the desired depth rating. I'd be wary of the plastic strength to hold up to the stress around the bolts.
3. Pot the sensors in a PVC end cap using epoxy. This changes the entire design, but is an option.

11

Deploying the CTD

Putting the Housing Together

Using the McMaster bolts and associated parts listed earlier, close the flange securely as in figure 11.1. Using a criss-cross star pattern, as when securing a car tire, secure the nuts on the bolts as far as they will go. Once the flange is fully seated, tie a bowline (or other reputable knot, but I recommend a bowline) through both eye nuts.

(a) Overall layout

(b) Closeup

Figure 11.1: Example of knot and tying arrangement

Deploying in the Water Column

It is important to ensure the CTD falls sensor first in the water and is lowered *slowly*. The slower the better as all sensors will be more accurate as they adjust. This, so to speak, gives a higher 'resolution' (see fig. 11.2 and 11.3, these were taken *near* one another, but not in the *exact* location, which is why they appear similar).

Additionally, tie the bitter end of the rope to something you do not lose everything.

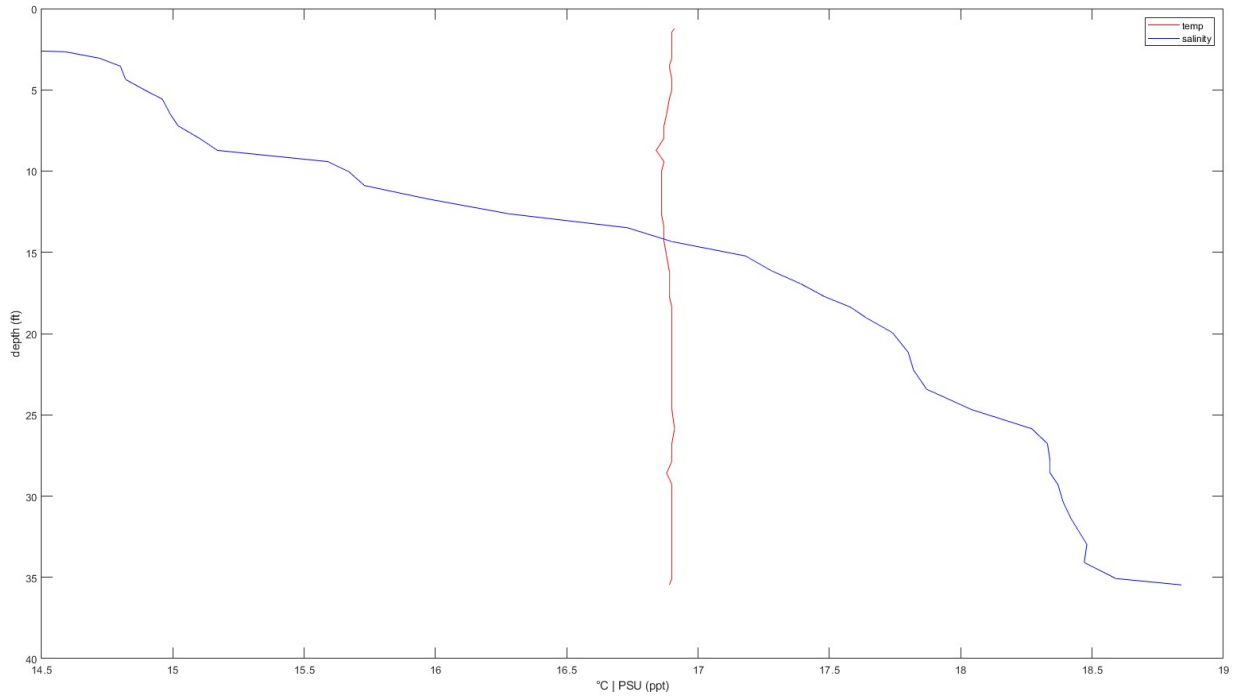


Figure 11.2: Fast drop, smoother

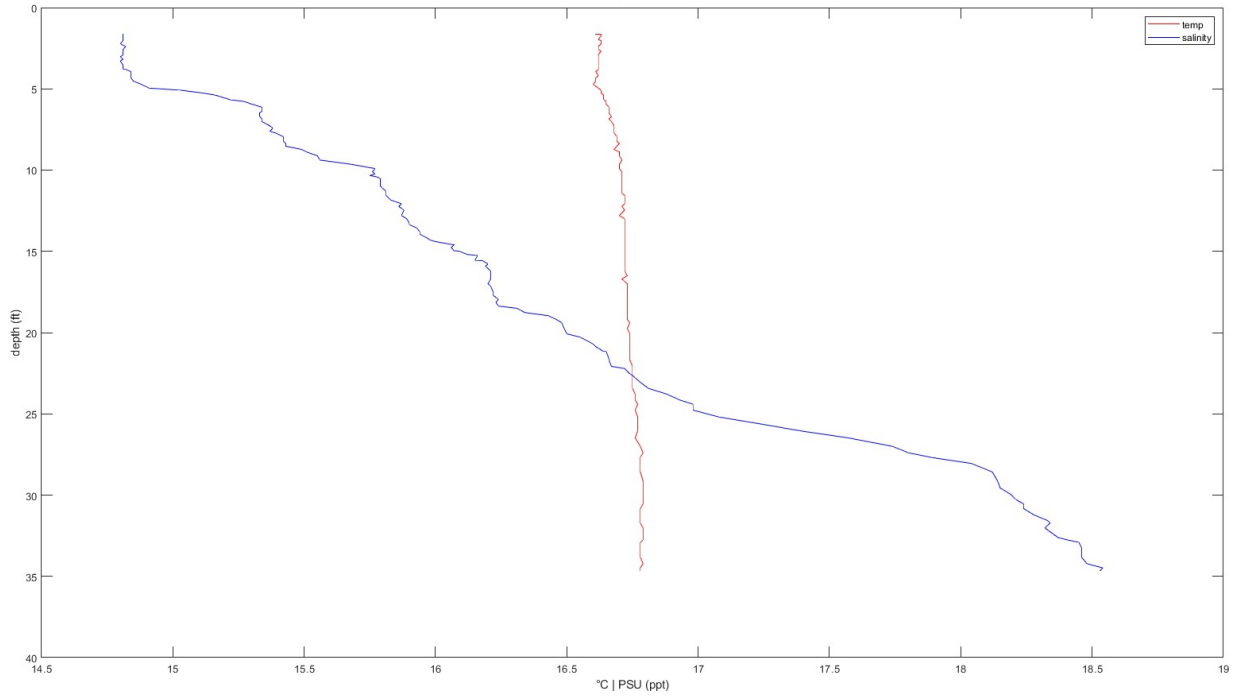


Figure 11.3: Slow drop, more granular

12

Data Accuracy

TLDR of the chapter: Compared to a properly calibrated Seabird CTD for ”true” values

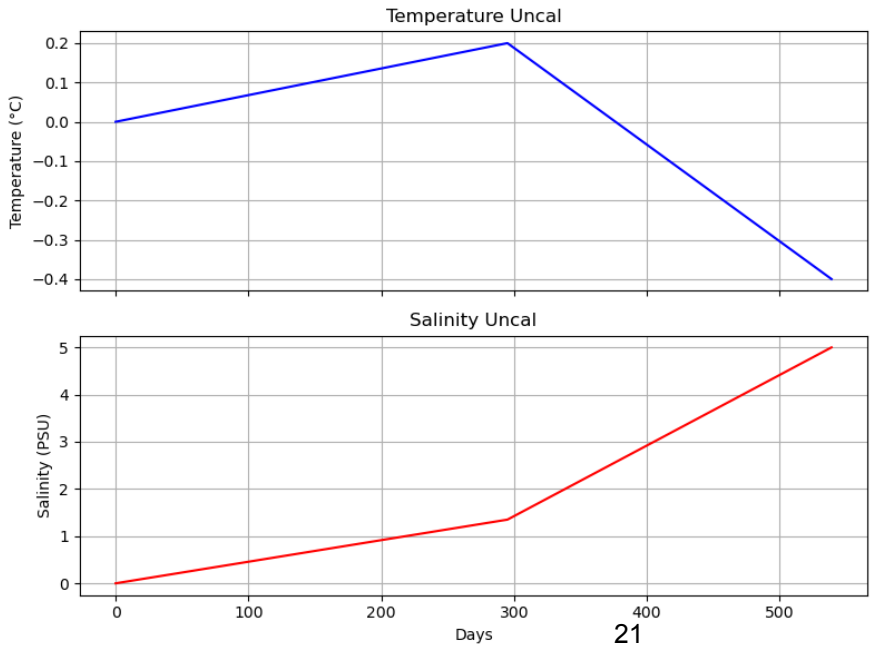
- The salinity sensors accuracy is greater than 0.5 PSU after approx. three to four months
- The temperature sensor is well within the stated ±4°C accuracy being within 0.2°C to -0.4°C

There have been two deployments following the initial test in Chapter 11:

1. **2023.8.18 RV Rutgers:** Comparing against a Seabird CTD with plots in MATLAB
2. **2024.4.17 RV Rutgers:** Comparing against a Seabird and RBR CTD with plots in Python

Both sensors were calibrated prior to Chapter 11’s test deployment on **2022.10.28** (the exact date of calibration is unknown), letting us see roughly how long between calibrations is reasonable. Assuming the first deployment is as accurate as the sensor platform can be, the following table shows the drift over time (assuming the Seabird CTD compared against are the ”true” ocean values). The temperature and salinity values in the table are added to the measured values to bring them into ”calibration” with the Seabird CTD.

Date	Days Between	Temperature (°C)	Salinity (PSU)
2022.10.28	n/a	0	0
2023.8.18	295	0.2	1.35
2024.4.17	244	-0.4	5.0



Considering the sensor is accurate to ±4°C, the temperature sensor is remarkably close to the Seabird values.

The salinity sensor is more than 0.5 PSU off a truer value approximately a few months after being calibrated.

Both sensors, especially the temperature sensor, have a slow response time and thus must be lowered very slowly. See the following two pages for figures.

Figure 12.1: Enter Caption

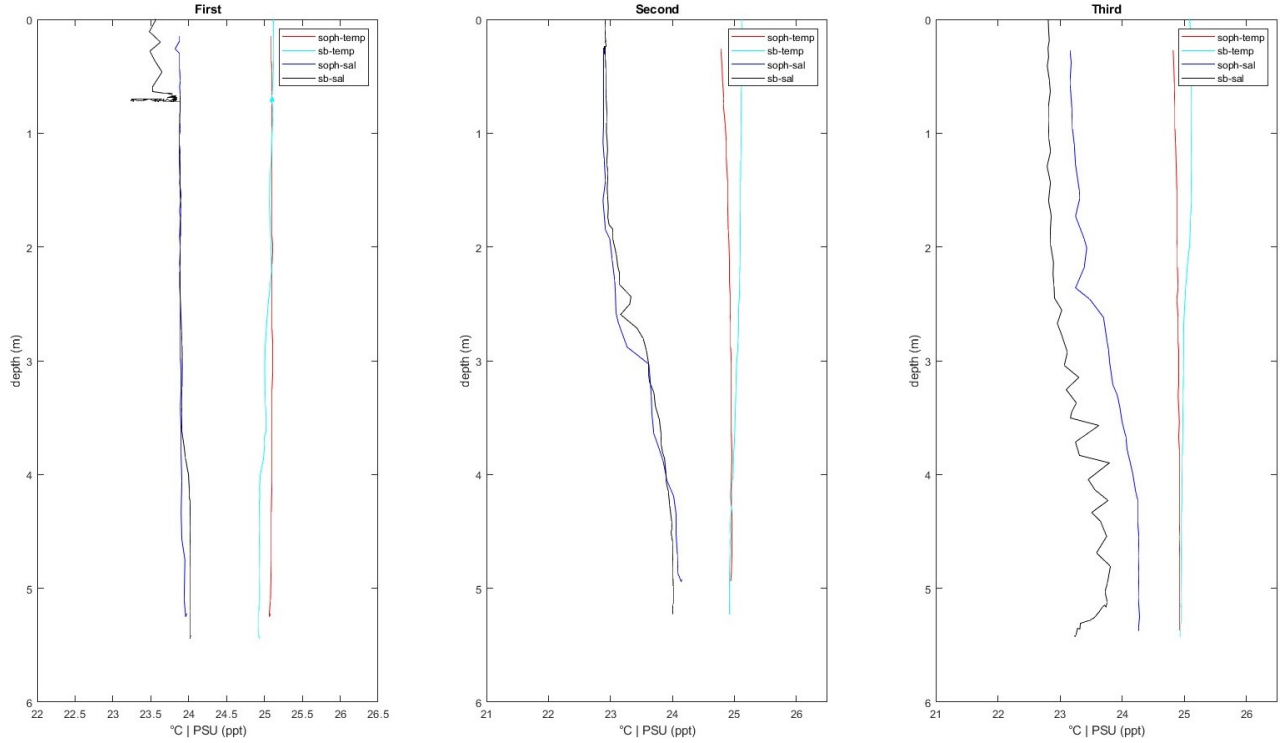


Figure 12.2: 2023.8.18: 1.35 added to the Homebrew_Arduin_CTD salinity values, and 0.2 added to the temperature values, and it's made pretty darn accurate (not the case during next deployment on following page)

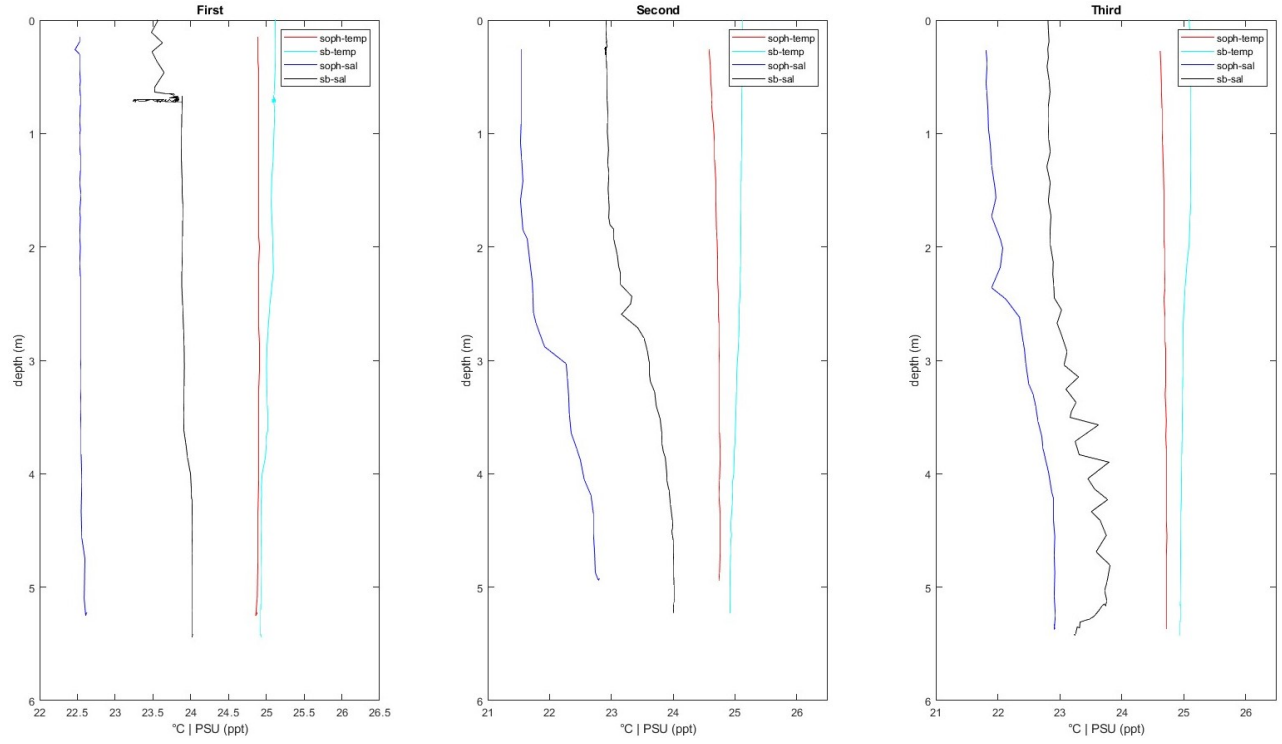


Figure 12.3: 2023.8.18: Raw recorded salinity values

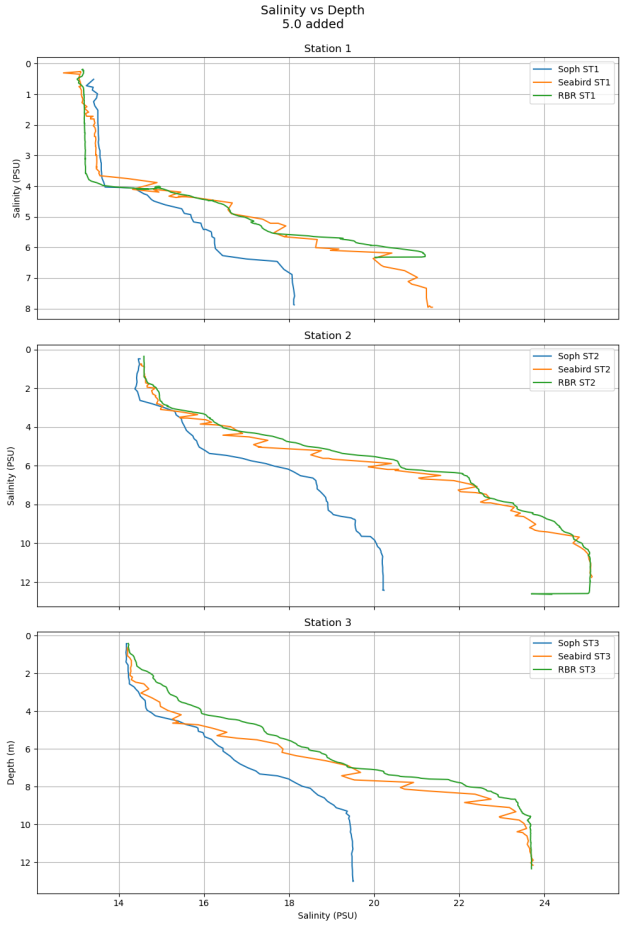


Figure 12.4: 2024.4.17: 5.0 added to the Homebrew_Arduin_CTD salinity values, and still not accurate

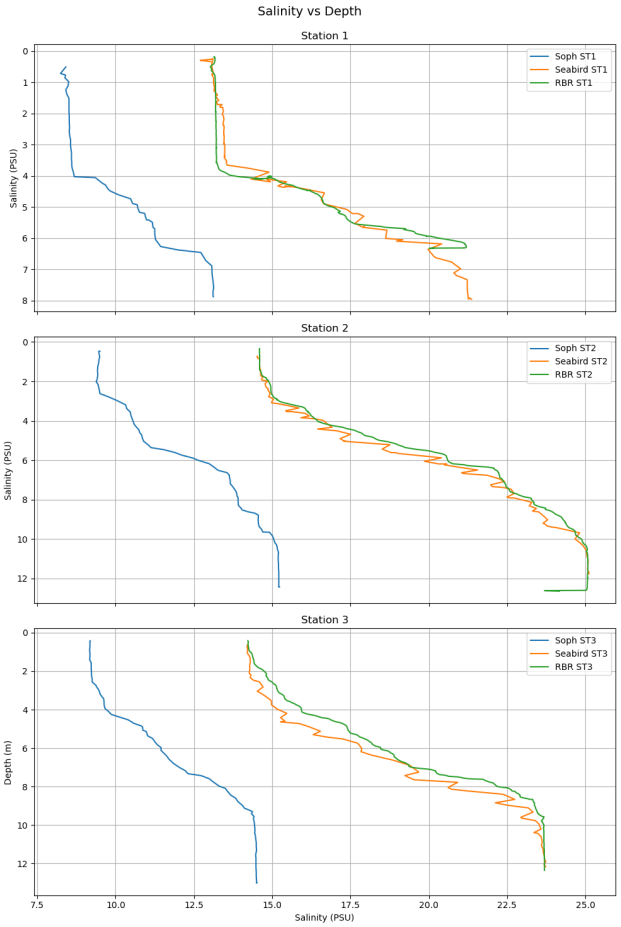


Figure 12.5: 2024.4.17: Raw recorded salinity values

The salinity at the surface will match, but at depth is way off the actual values. Again, I think this has to do with the response time of the sensor itself.

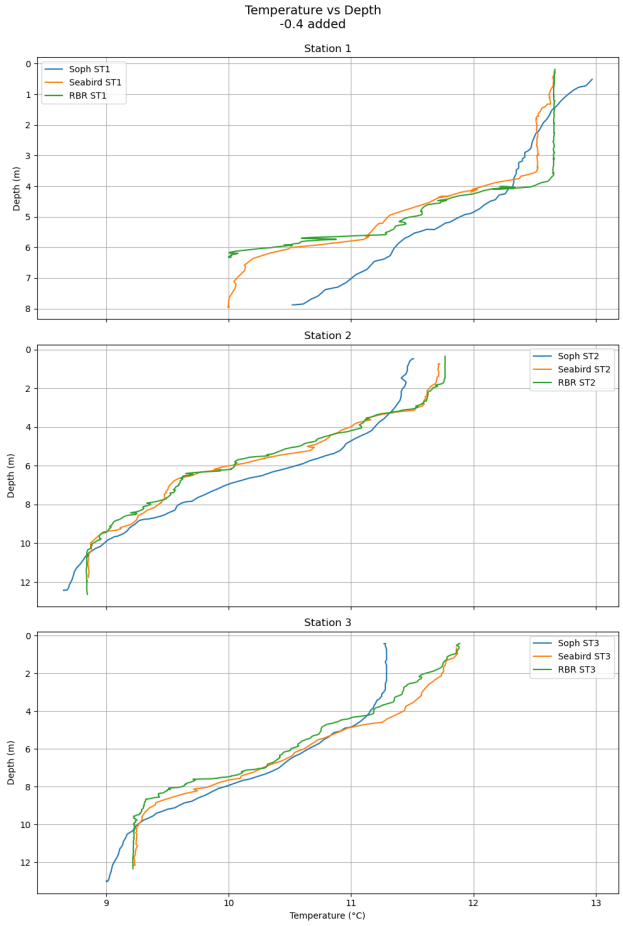


Figure 12.6: 2024.4.17: -0.4 added to the Homebrew_Arduin_CTD temperature values, and still not accurate

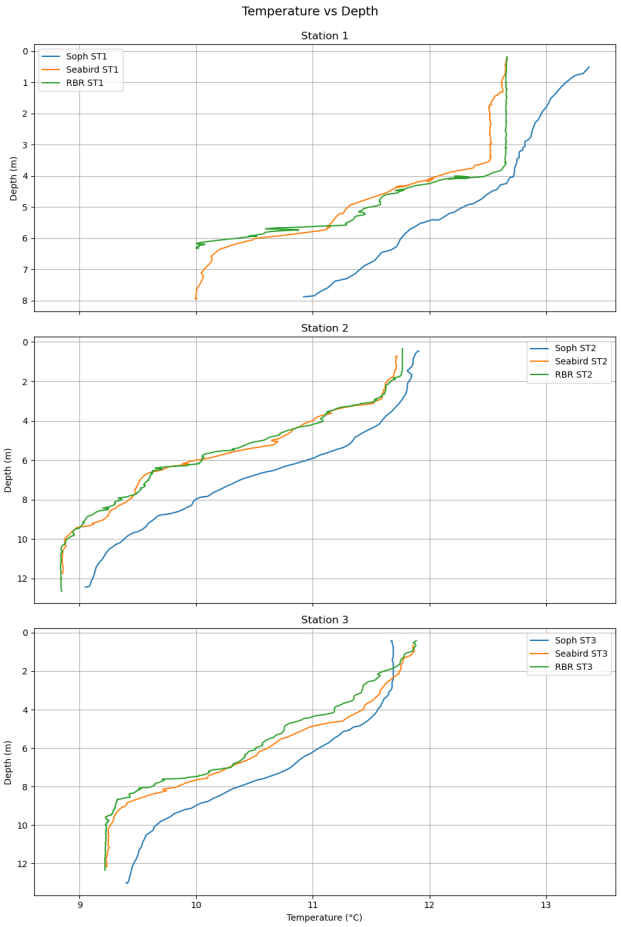


Figure 12.7: 2024.4.17: Raw recorded temperature values

Top right, this CTD can be seen to have a much slower response time. It was lowered at the same speed as the other sensors, meaning with this current sensor configuration it needs to be lowered *extremely* slowly.

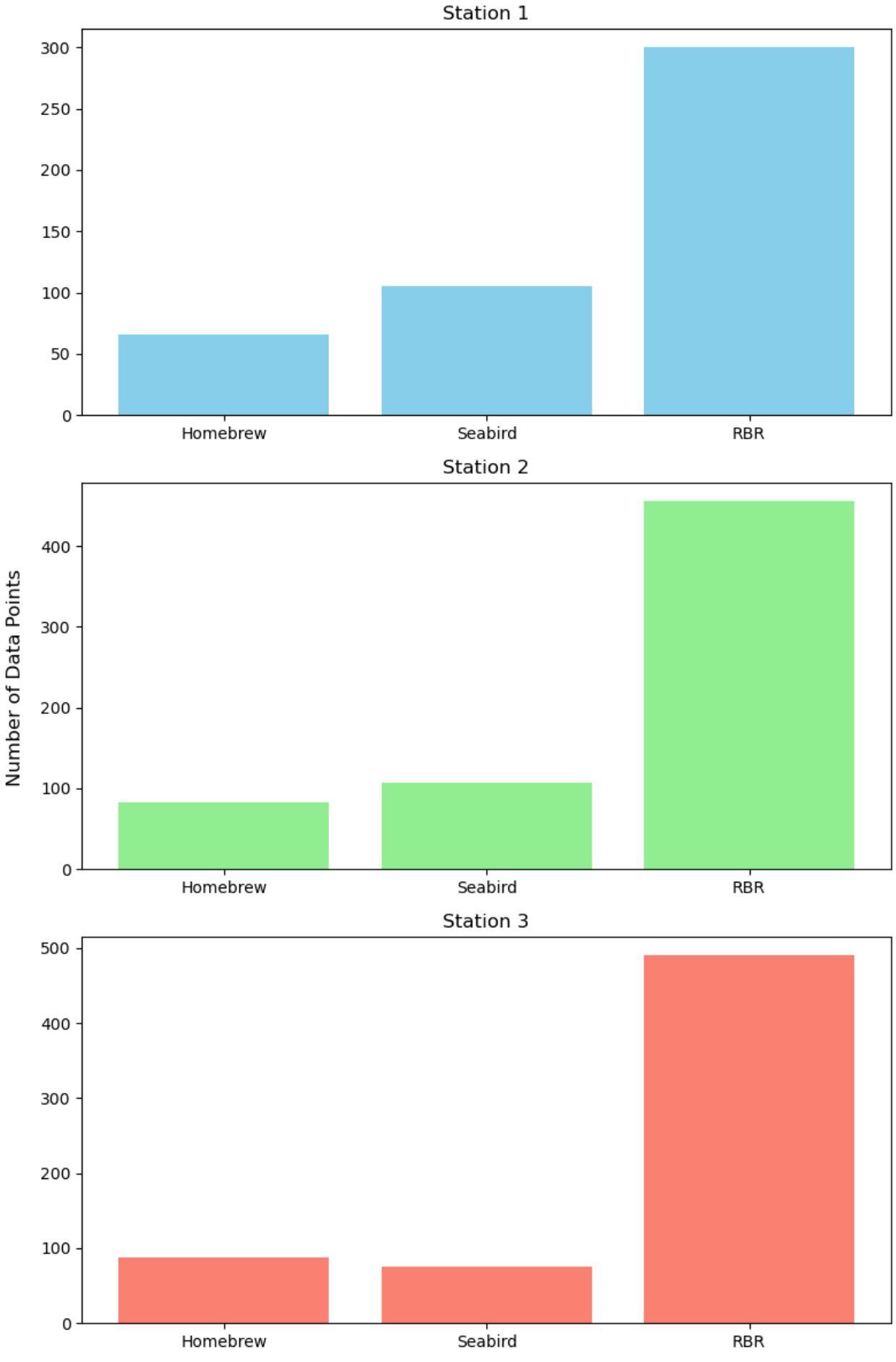


Figure 12.8: 2024.4.17: Amount of datapoints for each sensor for each station. The RBR is 10hz while the other two are 1hz

13

The Arduino Program

Where it (mostly) came from

The code is a combination of Blue Robotics' sample code and Atlas Scientific's sample code for their respective sensors. The SD part of the program is based off the Arduino SD Datalogger example program found in the Arduino IDE, but changed by me for my purposes.

If you find the coding aspect of this type of project intimidating, try and focus on one aspect at a time. Using a breadboard, wire and connect the SD card and make sure that works. Then add one sensor, then the other. Use the example programs provided by the manufacturer or elsewhere on the internet. Generally speaking, someone else has probably tried or done what you're doing so look around and see what help is already out there.

14

Takeaways and Final Thoughts

- Take detailed notes every step of the way. Comment your code. Keep a lab notebook. A week later (or sometimes a year...) when you return to the project you can know exactly where you left off. It also greatly helps in the coalition of documents such as these or when you share your work.
- Take the time to understand how the breadboard works and stay consistent with the colors of your wires. You do not want to wire everything together, have it not work, then start from scratch if you cannot find the one wire that is in the wrong spot. Knowing the breadboard and being consistent makes finding the one awry wire or connection much easier, especially if you take a week or two off (say over a holiday).
- Do not sit and wallow in a stuck state. If you are stuck on something, ask your teacher, fellow students, the internet, and do not give up! Maybe ask me too. The answer is out there – it is not impossible. More than once I got frustrated with something not working. I would ask for help, move on to another part of the project, or try again the next day. Sleeping on it and coming back tomorrow is helpful (maybe with a coffee or tea!), or simply take a walk around the building.

What I would do differently next time

Building the Housing

- Not face the flange on the lathe, instead only making the wells and thru-hulls. While this would increase the overall weight (ultimately not a big deal), it would decrease the steps and tools involved while not substantially changing the final product.
- I made the wells 9.5mm deep, which is 4.5mm or so too deep. As they are now, getting a wrench on the water-side of the flange to hold the Blue Robotics' thru-hulls in place to properly seat the nut on the dry side is a slight bit challenging. While you can get the wrench on it, it would simply be easier if I did not make the well as deep.

3D Printed Parts

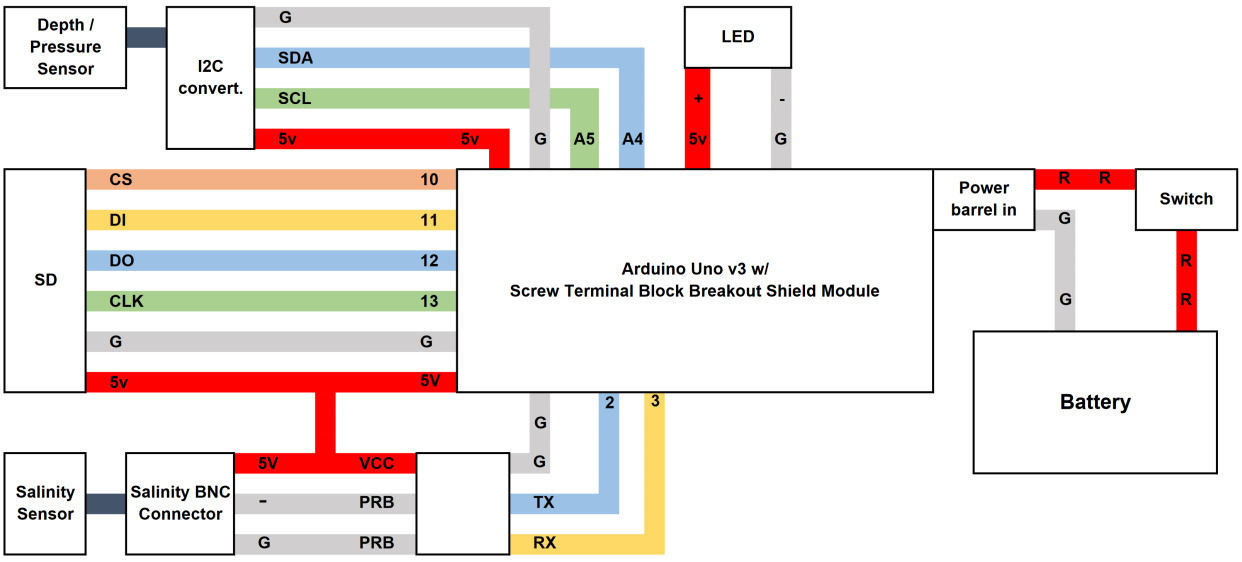
- Redesign the equipment tray to allow for greater clearance over the Arduino's shield. As designed, the wiring barely clears the PVC as it slides in.
- Redesign the flange holder's slot to properly fit the flange. When measuring I did not account for the slight protrusion on the housing side of the flange, meaning the holder is a tight fit. My print only works because of the low infill I printed it with, allowing it to flex to fit the flange. Thus, I do not recommend you make the flange exactly like I did anyhow (see section 10.3)

Wiring and Layout

- Solder pins to the I2C converter board instead of wires. This way I could put everything on a perma-proto breadboard.
- Totally redo the board with power and salinity to combined everything on one board. This would make the wiring far less messy.

A

Wiring Diagram



I do not recommend following my physical layout exactly 1:1 because there are far better ways to organize the internal sensor tray (i.e., use the wiring diagram to design your own physical wiring). The layout I ended up with formed from the organic state of starting with certain sensors, changing them partway through, and making it up as I went along while I worked through getting the CTD to operate as intended.

B

Arduino Program

```
// Order of CSV
// EC , TDS , SAL , mbar, C , meters

//This code was written in the Arduino 1.8.57.0

// TDS - Total Dissolved Solids
// 12,940 for 12880 23.0C      12,396 is actual thing to input
// 80,000 for 61601 at 22.4C   76,048 as actual thing to input

//Blue Robotics
#include <Wire.h>
#include "MS5837.h"
MS5837 sensor;

//EC
#include <SoftwareSerial.h>                           //we have to include the SoftwareSerial library, or else we can't use it
#define rx 2                                          //define what pin rx is going to be
#define tx 3                                          //define what pin tx is going to be

SoftwareSerial myserial(rx, tx);                      //define how the soft serial port is going to work

String inputstring = "";                              //a string to hold incoming data from the PC
String sensorstring = "";                             //a string to hold the data from the Atlas Scientific product
boolean input_string_complete = false;                //have we received all the data from the PC
boolean sensor_string_complete = false;               //have we received all the data from the Atlas Scientific product

//SD
#include <SD.h>
File dataFile;

void setup() {                                        //set up the hardware
  Serial.begin(9600);                                 //set baud rate for the hardware serial port_0 to 9600

  myserial.begin(9600);                               //set baud rate for the software serial port to 9600
  inputstring.reserve(10);                            //set aside some bytes for receiving data from the PC
  sensorstring.reserve(30);                           //set aside some bytes for receiving data from Atlas Scientific product

//BLUE ROBOT
  // Initialize pressure sensor
  // Returns true if initialization was successful
  // We can't continue with the rest of the program unless we can initialize the sensor
  Wire.begin();
```

```
  while (!sensor.init()) {
    Serial.println("Init failed!");
    Serial.println("Are SDA/SCL connected correctly?");
    Serial.println("Blue Robotics Bar30: White=SDA, Green=SCL");
    Serial.println("\n\n\n");
    delay(5000);
  }

  sensor.setModel(MS5837::MS5837_30BA);
  sensor.setFluidDensity(1029); //kg/m^3 (saltwater, 997 for freshwater)

//sets the output from the Arduino to the micro SD reader
   pinMode(10, OUTPUT);
 if (!SD.begin(10)) {
    Serial.println("initialization bad. check card"); //if doesn't init
    return;
  }
  Serial.println("initialization done."); //if does init

File dataFile = SD.open("dataFile.txt", FILE_WRITE);
  dataFile.print("#################################");//between data when turned on/off between deployments
  dataFile.println();
  dataFile.close();

}

void serialEvent() {                                //if the hardware serial port_0 receives a char
  inputstring = Serial.readStringUntil(13);         //read the string until we see a <CR>
  input_string_complete = true;                     //set the flag used to tell if we have received a completed string from the PC
}

void loop() {

  if (input_string_complete == true) {              //if a string from the PC has been received in its entirety
    myserial.print(inputstring);                    //send that string to the Atlas Scientific product
    myserial.print('\r');                           //add a <CR> to the end of the string
    inputstring = "";                               //clear the string
    input_string_complete = false;                  //reset the flag used to tell if we have received a completed string from the PC
  }

  if (myserial.available() > 0) {                   //if we see that the Atlas Scientific product has sent a character
    char inchar = (char)myserial.read();            //get the char we just received
    sensorstring += inchar;                         //add the char to the var called sensorstring
    if (inchar == '\r') {                           //if the incoming character is a <CR>
      sensor_string_complete = true;                //set the flag
    }
  }

  if (sensor_string_complete == true) {             //if a string from the Atlas Scientific product has been received in its entirety
    if (isdigit(sensorstring[0]) == false) {        //if the first character in the string is a digit
      Serial.println(sensorstring);                 //send that string to the PC's serial monitor
    }
    else                                            //if the first character in the string is NOT a digit
    {
      print_EC_data();                              //then call this function
    }
    sensorstring = "";                              //clear the string
    sensor_string_complete = false;                 //reset the flag used to tell if we have received a completed string from the Atlas Scientific product
  }
}
```

```
103
104 void print_EC_data(void) {                              //this function will pars the string
105   char sensorstring_array[30];                          //we make a char array
106   char *EC;                                             //char pointer used in string parsing
107   char *TDS;                                            //char pointer used in string parsing
108   char *SAL;                                            //char pointer used in string parsing
109   char *GRAV;                                           //char pointer used in string parsing
110   float f_ec;                                           //used to hold a floating point number that
          is the EC
111
112   sensorstring.toCharArray(sensorstring_array, 30);   //convert the string to a char array
113   EC = strtok(sensorstring_array, ",");               //let's pars the array at each comma
114   TDS = strtok(NULL, ",");                            //let's pars the array at each comma
115   SAL = strtok(NULL, ",");                            //let's pars the array at each comma
116   GRAV = strtok(NULL, ",");                           //let's pars the array at each comma
117
118 //PRINT TO SERIAL MONITOR
119   //Serial.print("EC:");      //we now print each value we parsed separately
120   Serial.print(EC);           //this is the EC value
121   Serial.print(",");
122
123   //Serial.print("TDS:");     //we now print each value we parsed separately
124   Serial.print(TDS);          //this is the TDS value
125   Serial.print(",");
126
127   //Serial.print("SAL:");     //we now print each value we parsed separately
128   Serial.print(SAL);          //this is the salinity value
129   Serial.print(",");
130
131   //Serial.print("GRAV:");    //we now print each value we parsed separately
132   Serial.print(GRAV);         //this is the specific gravity
133   Serial.print(",");
134   //Serial.println();         //this just makes the output easier to read
135
136 //f_ec= atof(EC);             //uncomment this line to convert the char to a float
137
138 ///////BLUE ROBOT, read sensor
139 sensor.read();
140   Serial.print(sensor.pressure()); //mbar
141   Serial.print(",");
142   Serial.print(sensor.temperature()); //C
143   Serial.print(",");
144   Serial.print(sensor.depth()); //meters
145   Serial.println();
146
147   delay(250); //found this delay works best
148
149 //Save EC data to SD card
150 File dataFile = SD.open("dataFile.txt", FILE_WRITE);
151   dataFile.print(EC);       //this is the EC value
152   dataFile.print(",");
153   dataFile.print(TDS);      //this is the TDS value
154   dataFile.print(",");
155   dataFile.print(SAL);      //this is the salinity value
156   dataFile.print(",");
157   //dataFile.print(GRAV);   //this is the specific gravity
158   //dataFile.print(",");
159
160 //Save Blue Robot data to SD card
161 sensor.read();
162   dataFile.print(sensor.pressure());    //mbar
163   dataFile.print(",");
164   dataFile.print(sensor.temperature()); //C
165   dataFile.print(",");
166   dataFile.print(sensor.depth());       //meters
167   dataFile.println();
168 // closes the file to ensure the file was written:
169     dataFile.close();
170
171 delay(750);
172 }
```

C

MATLAB Program

```
%  _    _                      _                          _____ _______ _____
% | |  | |                    | |                        / ____|__   __|  __ \
% | |__| | ___  _ __ ___    ___| |__  _ __ _____       __ | |       | |  | |  | |
% |  __  |/ _ \| '_ ` _ \ / _ \ '_ \| '__/ _ \ \ /\ / / | |       | |  | |  | |
% | |  | | (_) | | | | | |  __/ |_) | | |  __/\ V  V /  | |____   | |  | |__| |
% |_|  |_|\___/|_| |_| |_|\___|_.__/|_|  \___| \_/\_/    \_____|  |_|  |_____/
% (ASCII font Big from https://www.coolgenerator.com/ascii-text-generator
% Homebrew CTD Data Grapher / Visualizer
% Author: Sophie Scopazzi
% Author Website: sophiescopazzi.com
% Date: OCT 2022

% always
clear all
close all

%% INSTRUCTIONS - HOW TO USE THIS CODE
% There are comments throughout code as well
% If you want to contact me about this code, use my website

% 1. Put this script in same folder as datafile.txt from CTD
    % Check name of datafile is the same as in the script
% 2. Ensure sensors are in the correct order in the data from the Arduino
    % script, or none of it will make sense (EC, TDS, sal, etc) in order
% 3. Find correct time values in this plot, record for next step
    % Want data from downward travel, upward water column is disturbed
% 4. Re-comment step 3, put the values recorded into DEPLOY DATA PULLING
% 5. Plots or subplots as desired. Set xlims and ylims as necessary

%% 2. DATA INGEST AND CLEANING
% datafile from CTD, ensure name is correct, script in same folder as data
data = readtable('DATAFILE.TXT');

% ensure the data is labeled correctly, will depend on Arduino code
% get data from data table, BUT it has nans
    EC_nans       =   data(:,1);  %EC data
    TDS_nans      =   data(:,2);  %TDS
    sal_nans      =   data(:,3);  %salinity
    mbar_nans     =   data(:,4);  %mbar
    temp_nans     =   data(:,5);  %temp in C
    dmeter_nans   =   data(:,6);  %depth in meters

        % take nans out of above data
        EC_table      =   rmmissing(EC_nans);
        TDS_table     =   rmmissing(TDS_nans);
        sal_table     =   rmmissing(sal_nans);
        mbar_table    =   rmmissing(mbar_nans);
        temp_table    =   rmmissing(temp_nans);
        dmeter_table  =   rmmissing(dmeter_nans);

```

```
    % convert data from table to array for plotting / manipulation
    EC            =   table2array(EC_table);
    TDS           =   table2array(TDS_table);
    sal           =   table2array(sal_table);
    mbar          =   table2array(mbar_table);
    temp          =   table2array(temp_table);
    dmeter        =   table2array(dmeter_table);

% convert mbar to feet, incase you want to do that
% depth_feet = mbar.*0.033455256555148;

% convert meters to feet, in case you want to use the Mapping Toolbox
% deploy_feet = distdim(dmeter,'meters','feet'); %(needs Mapping Toolbox)
dfeet = dmeter.*3.28084; % I didn't want to use Mapping Toolbox

%% 3. TO FIND CORRECT START / STOP VALUES IN TIME
    % FIND TIME VALUES IN THIS PLOT than re-comment this out
    % look in this plot to find where depth values start and stop
    % this is what you will put in below, the (93:135), etc
    % uncomment below --- 72-75
    % n = length(mbar);
    % time = 1:n;
    % figure;
    % plot(time,mbar);

%% 4. DEPLOY DATA PULLING
    % take out only parts of the data for deployments
    % this is the values found 'FIND DEPTH VALUE PLOT'
    % if you have more or less than four, change it

    % ONE
    deploy1_EC       =   EC      (93:135,1);
    deploy1_TDS      =   TDS     (93:135,1);
    deploy1_feet     =   dfeet   (93:135,1);
    deploy1_temp     =   temp    (93:135,1);
    deploy1_sal      =   sal     (93:135,1);

    % TWO
    deploy2_EC       =   EC      (609:660,1);
    deploy2_TDS      =   TDS     (609:660,1);
    deploy2_feet     =   dfeet   (609:660,1);
    deploy2_temp     =   temp    (609:660,1);
    deploy2_sal      =   sal     (609:660,1);

    % THREE
    deploy3_EC       =   EC      (774:799,1);
    deploy3_TDS      =   TDS     (774:799,1);
    deploy3_feet     =   dfeet   (774:799,1);
    deploy3_temp     =   temp    (774:799,1);
    deploy3_sal      =   sal     (774:799,1);

    % FOUR
    deploy4_EC       =   EC      (832:979,1);
    deploy4_TDS      =   TDS     (832:979,1);
    deploy4_feet     =   dfeet   (832:979,1);
    deploy4_temp     =   temp    (832:979,1);
    deploy4_sal      =   sal     (832:979,1);

%% 5. GRAPHS
% however many you did, make that many subplot graphs
% or you can put them on their own plots (not subplot them)
% these are the same with the numbers changed, 1-4
    figure;
    subplot(2,2,1); % subplot since I want four graphs on one figure
    plot(deploy1_temp,deploy1_feet,'red');
    title('One');
    hold
    plot(deploy1_sal,deploy1_feet,'blue');
    set(gca,'Ydir','reverse'); % to make zero the top of Y axis
    ylim([0 40]);   % use the depth you deployed
    xlim([14.5 19]) % use the range of your data
```

```
    legend('temp','salinity')
    ylabel('depth (ft)');
    xlabel('°C | PSU (ppt)');

        subplot(2,2,2);
        plot(deploy2_temp,deploy2_feet,'red');
        title('Two');
        hold
        plot(deploy2_sal,deploy2_feet,'blue');
        set(gca,'Ydir','reverse');
        ylim([0 40]);   % use the depth you deployed
        xlim([14.5 19]) % use the range of your data
        legend('temp','salinity')
        ylabel('depth (ft)');
        xlabel('°C | PSU (ppt)');

    subplot(2,2,3);
    plot(deploy3_temp,deploy3_feet,'red');
    title('Three');
    hold
    plot(deploy3_sal,deploy3_feet,'blue');
    set(gca,'Ydir','reverse');
    ylim([0 40]);   % use the depth you deployed
    xlim([14.5 19]) % use the range of your data
    legend('temp','salinity')
    ylabel('depth (ft)');
    xlabel('°C | PSU (ppt)');

        subplot(2,2,4);
        plot(deploy4_temp,deploy4_feet,'red');
        title('Four');
        hold
        plot(deploy4_sal,deploy4_feet,'blue');
        set(gca,'Ydir','reverse');
        ylim([0 40]);   % use the depth you deployed
        xlim([14.5 19]) % use the range of your data
        legend('temp','salinity')
        ylabel('depth (ft)');
        xlabel('°C | PSU (ppt)');

%%%%%%%%%%%%%%%%%%%%%%%

    figure;
    plot(deploy1_temp,deploy1_feet,'red');
    title('Slow');
    hold
    plot(deploy1_sal,deploy1_feet,'blue');
    set(gca,'Ydir','reverse'); % to make zero the top of Y axis
    ylim([0 40]);   % use the depth you deployed
    xlim([14.5 19]) % use the range of your data
    legend('temp','salinity')
    ylabel('depth (ft)');
    xlabel('°C | PSU (ppt)');

        figure;
        plot(deploy4_temp,deploy4_feet,'red');
        title('Fast');
        hold
        plot(deploy4_sal,deploy4_feet,'blue');
        set(gca,'Ydir','reverse');
        ylim([0 40]);   % use the depth you deployed
        xlim([14.5 19]) % use the range of your data
        legend('temp','salinity')
        ylabel('depth (ft)');
        xlabel('°C | PSU (ppt)');
```