CTDizzle

An Open Source CTD

Version: 6.0.0
2020-12-26

Table of Contents

License

Author's Note

You can use and copy any part of the CTDizzle as much as you like, but you are encouraged to make a donation to the American Civil Liberties Union (www.aclu.org) or any charity of your choice if you find it useful.

This guide operates under the assumption that you are using parts found within the Bill of Materials. This guide is not foolproof and the author acknowledges that there are likely better ways to build an open source CTD. The author has also made many expensive mistakes while iterating each build and recognizes that builders may make similar mistakes if not treading lightly. If you are unsure of how to perform a task, please take the time to research methods and practice skills. If you come up with a simpler or better way to complete a task, please share that knowledge!

If at any time you have questions, concerns, or need guidance, please to do not hesitate to contact:

Ian Black
iantimothyblack@gmail.com

CTDizzle Specifications

Maximum Sampling Rate: 1Hz
Tested Depth: 80m
Maximum Depth: 100m (Acrylic Tube), 200m (Aluminum Tube)
Conductivity Accuracy: +/- 2% (After Calibration)
Conductivity Response Time: 1s
Temperature Accuracy: +/- 0.1 degC
Temperature Response Time: 3s
Pressure Accuracy: +/- 100 mbar
Pressure Response Time: 20ms
Clock Accuracy: 29 PPM (+/- 3s per day)
Battery Life: Dependent (~60 hours @ 1Hz on a 3.7v 2200 mAh)

Bill of Materials

A complete list of parts used, cost, and quantities can be found in the CTDizzle GitHub repository under the documentation folder in a file called BoM.ods.

Other Open Source CTDs

Arduino-based Sonde: https://github.com/glockridge/MooredApplication
Oceanography For Everyone OpenCTD: https://github.com/OceanographyforEveryone/OpenCTD
PiCTD: https://github.com/HaanHouse/PiCTD

Required Skills and Experience

The construction of any open source CTD requires some experience using tools such as a soldering iron, wire strippers, drill, screwdriver, and wrench. If you are unfamiliar with soldering, it is highly recommended that you practice the lineman's splice on a piece of scrap wire and through-hole soldering on a piece of protoboard.

The CTDizzle in particular requires some experience 3D printing with Polylactic Acid (PLA) and Polyvinyl Alcohol (PVA) filaments. If you are following along with this guide, you will also be required to mix and place epoxies.

O-rings

The Blue Robotics components of this CTD build require you to be familiar with O-ring inspection and installation. Clean and greased O-rings are paramount to the success of a CTD deployment in the ocean. To inspect and install an O-ring…

1. Wash and dry your hands.
2. Use isopropyl alcohol and lint-free wipes to clean the O-ring.
3. Under a light source, lightly stretch the O-ring and use your fingers to feel and observe for any damage.
4. Apply a light amount of silicone grease to the O-ring before installation.
5. Use an O-ring pick to install the O-ring.

Software Setup

Arduino IDE

The Arduino Integrated Development Environment (IDE) is simple to use and allows you to quickly upload code to a board that understands Arduino. If you are new to Arduino, it is recommended that you set up the Arduino IDE using the default directories that the download wizard creates.
To install the IDE…

1. Go to https://www.arduino.cc/en/software to download the Arduino IDE installer that works best with your operating system.
2. Follow the installation wizard.
3. Hooray! You can now use the IDE.

Arduino Board

There are many different types of Arduino boards out there and not all are supported by default within the Arduino IDE. In order to get the *Adafruit Feather M0 Basic Proto* to work, we need to make sure the IDE knows what to look for by installing the right package.
To set up the M0…

1. Open the Arduino IDE.
2. Navigate to Tools > Board > Boards Manager.
3. In the search bar, type in Adafruit SAMD Boards
4. In the window, select Adafruit SAMD Boards. Confirm the Adafruit Feather M0 is in the list of boards included in the package.
5. Ensure the version is the most recent version.
6. Select Install.

Arduino Libraries

The CTDizzle requires the installation of Arduino libraries that help you communicate with each sensor. Libraries contain sets of functions that simplify the development process and allow you to focus more on higher level coding. In order to operate the CTDizzle, you will need to install several third-party libraries. Conveniently, they can be installed via the Arduino IDE.

Table 1: Third-party Arduino Libraries

Library Name	Developer	GitHub Repo
RTClib	Adafruit	https://github.com/adafruit/RTClib
BlueRobotics MS5837 Library	Blue Robotics	https://github.com/bluerobotics/BlueRobotics_MS5837_Library
BlueRobotics TSYS01 Library	Blue Robotics	https://github.com/bluerobotics/BlueRobotics_TSYS01_Library

To install these libraries…

1. Open the Arduino IDE.
2. Navigate to Tools > Manage Libraries.
3. Search for each library by name.
4. Install the most recent version.

Formatting the SD Card

It is also necessary to format the microSD card so that it can be used to store data. You can do this using the native formatter utility for your operating system, but it is recommended that you utilize the SD Memory Card Formatter tool developed by the SD Association.
To download the SD Memory Card Formatter…

1. Go to https://www.sdcard.org/downloads/formatter/
2. Download the correct tool for your operating system.
3. Follow installation instructions.

To format your card…

1. Insert your card into your computer. Make sure it appears in your file explorer.
2. Open the SD Card Formatter tool.
3. Select your card.
4. Select Quick Format.
5. Name your card if you want.
6. Select Format.

Uploading a Sketch

Uploading a code (aka sketch) is simple to do with the Arduino IDE.

To upload a sketch…

1. Open the Arduino IDE.
2. Make sure the sketch you want to upload is in the window.
3. Navigate to Tools > Board and select Adafruit Feather M0.
4. Navigate to Tools > Port and select the port that has the Adafruit Feather M0.
5. Navigate to Sketch > Upload. (or hit CTRL + U, or select the right-pointing arrow)
6. Wait for the sketch to compile and upload!

Communication Protocols

All the sensors in this CTD build use the inter-integrated circuit (I2C) serial communication protocol. This means that each sensor can be on the same send and receive data lines. Each sensor has its own address, which prevents confusion when talking to multiple sensors. For the sake of simplicity, the operating code for this CTD waits for one sensor to respond and send data before talking to the next sensor.

The SD card module of the Adalogger Featherwing operates under the Serial Peripheral Interface (SPI) protocol. Like I2C, multiple sensors can be connected on the same lines send and receive lines. However, SPI requires that each sensor be connected to its own chip select, thus taking up more pins compared to I2C. Conveniently, the SD card module in this build is the only component that operates under SPI.

The Atlas Scientific EC EZO can operate under I2C or Universal Asynchronous Receive Transmit (UART). By default, the EC EZO is shipped in UART mode. You can review the Atlas Scientific manual to change it to I2C mode either via software or manually. Each protocol has its advantages and disadvantages, but the CTD operating code assumes that the EC EZO is in I2C mode.

Pinouts

Each sensor wire/pin should be connected to the controller board pin as described by the table below. Note that each Blue Robotics sensor is 3.3V tolerant. If you connect one of those sensors to a 5V pin, you risk damaging it. The Atlas Scientific EC EZO can be connected to 3.3V or 5V. The Adalogger Featherwing SD module is automatically connected when you stack it on top of the Feather M0. The chip select just needs to be specified in the operating code.

Table 2: Sensor Pinouts

Blue Robotics TSYS01 Temperature Sensor	**Adafruit Feather M0**
Black Wire	GND
Red Wire	3V
Yellow Wire	SDA
Green Wire	SCL

Blue Robotics MS5837 Pressure Sensor	**Adafruit Feather M0**
Black Wire	GND
Red Wire	3V
Yellow Wire	SDA
Green Wire	SCL

Atlas Scientific EC EZO or Carrier Board	**Adafruit Feather M0**
GND	GND
VCC	3V
SDA/TX	SDA
SCL/RX	SCL

Adalogger Featherwing SD Module	**Adafruit Feather M0**
CS	10

Component Testing

Please inspect and test each component before soldering and install.

Vendor guides can be found in the documentation folder of the GitHub repository.

Building the CTDizzle

Preparing the End Cap

In order to pot the Atlas Scientific Conductivity probe to the end cap, you must drill out one of the holes to 12mm. In the following pictures, the side facing up is the “external” side and the side facing down is the “internal” side. “External” implies that it is going to be exposed to seawater.

In the left image orientation, the 12mm hole was drilled into the upper right hole using a 12mm drill bit. The hole was then deburred. You will want to test fit the conductivity probe in the newly drilled hole. If you are unable to insert it, use a round file to increase the diameter of the hole slightly. The conductivity probe should fit snugly and require some force to insert and remove.

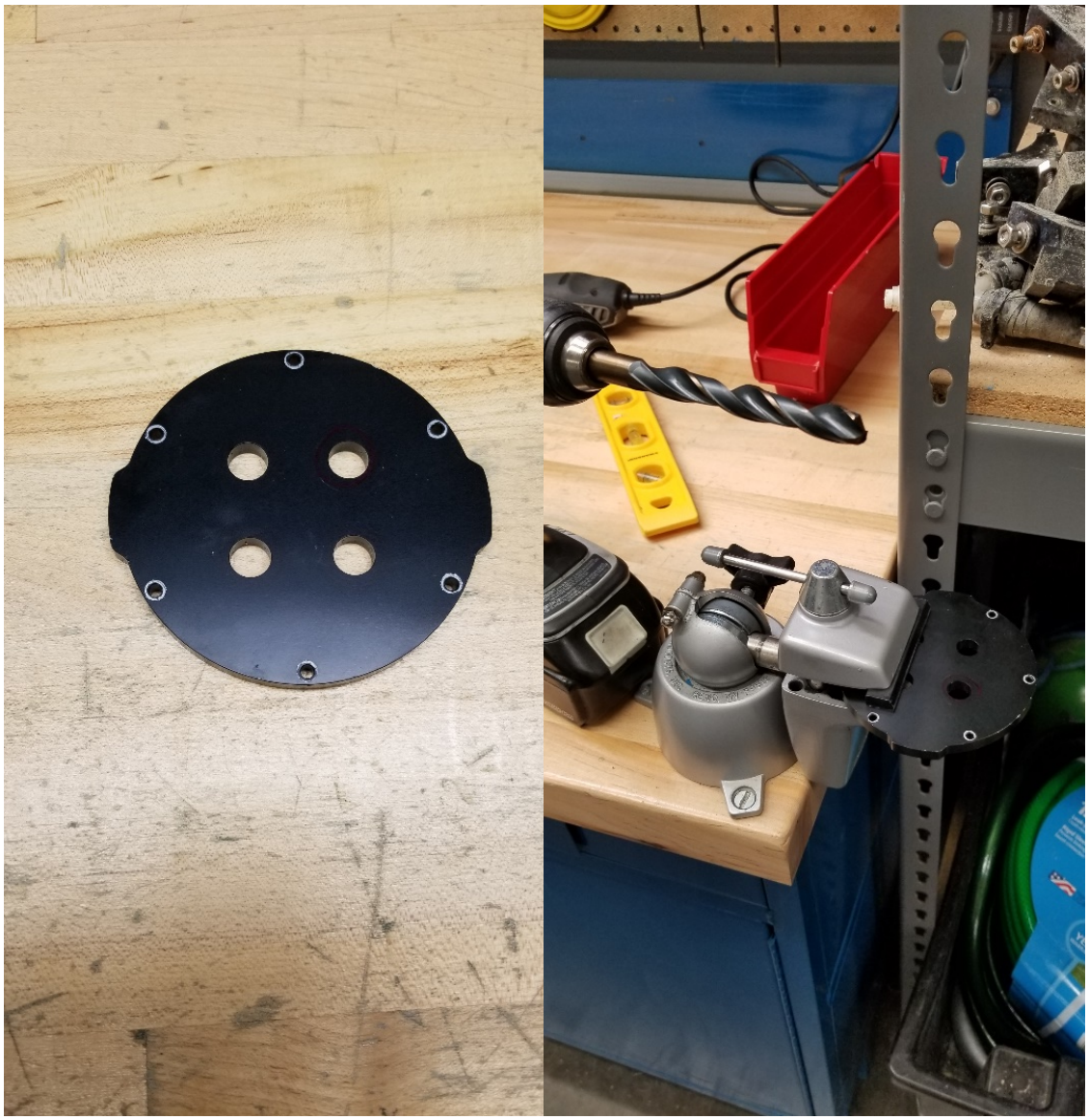


Figure 1: End cap 12mm hole placement.

Next attach the dremel_internal and dremel_external 3D prints to the endcap using some M3 screws and nuts. Notice that the internal and external 3d prints result in holes that are different shapes. The internal print will produce an irregularly shaped outline to accommodate the internal diameter of the end cap flange.

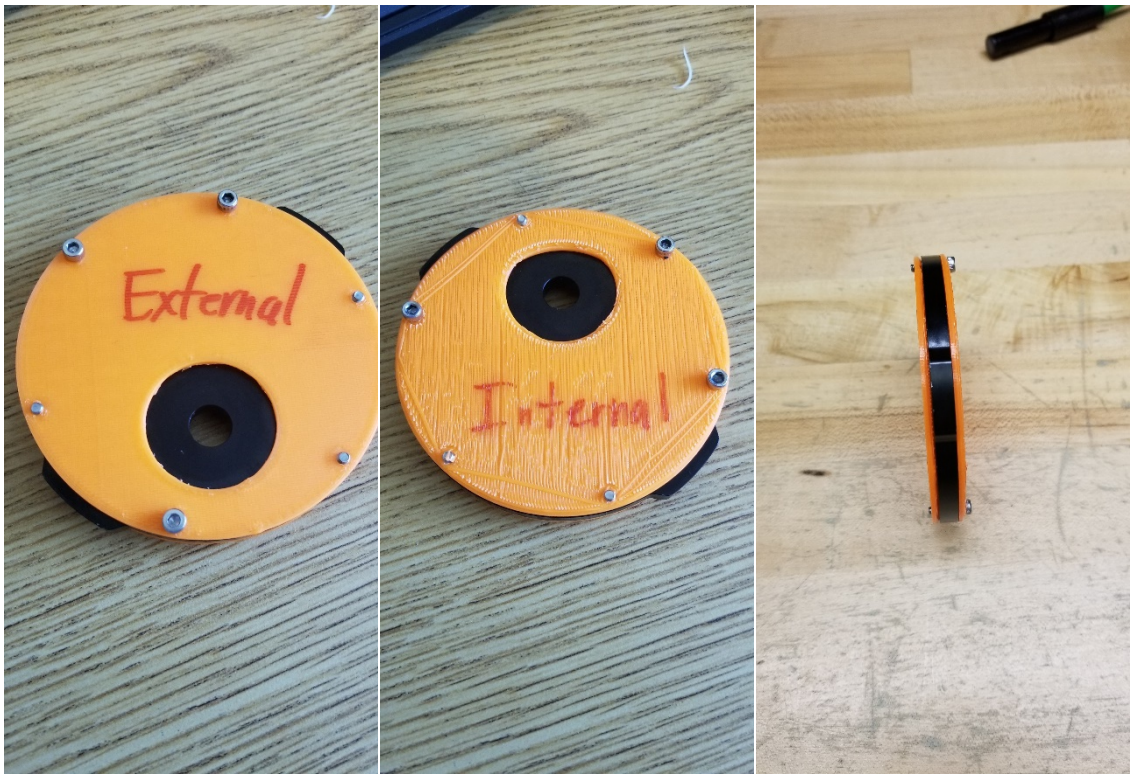


Figure 2: Rotary tool jig placement.

Using a Dremel or similar rotary tool, remove the exposed anodization. This can be done using a grindstone bit or sandpaper attachment. You should remove the anodization to the point where you see exposed aluminum and the surface finish is rough. This will facilitate better adhesion of the epoxy used to hold the conductivity probe in place.

Figure 3: End cap after removal of anodization.

Potting the Probe

Test fit the probe in the 12mm hole. Using the PVA (potting_*) prints as guides, mark on the probe the height of the prints. Remove the probe from the end cap.

Using sandpaper, rough up the portions of the probe that will be in contact with epoxy. This will ensure better adhesion.

Clean the end cap and probe with isopropyl alcohol. Apply masking or electrical tape around the edge of the raw aluminum on both sides. This will help keep the potting compound in the appropriate area in case the PVA potting jig warps during use. Install the probe and wrap the sensor area in electrical tape to prevent epoxy leakage.

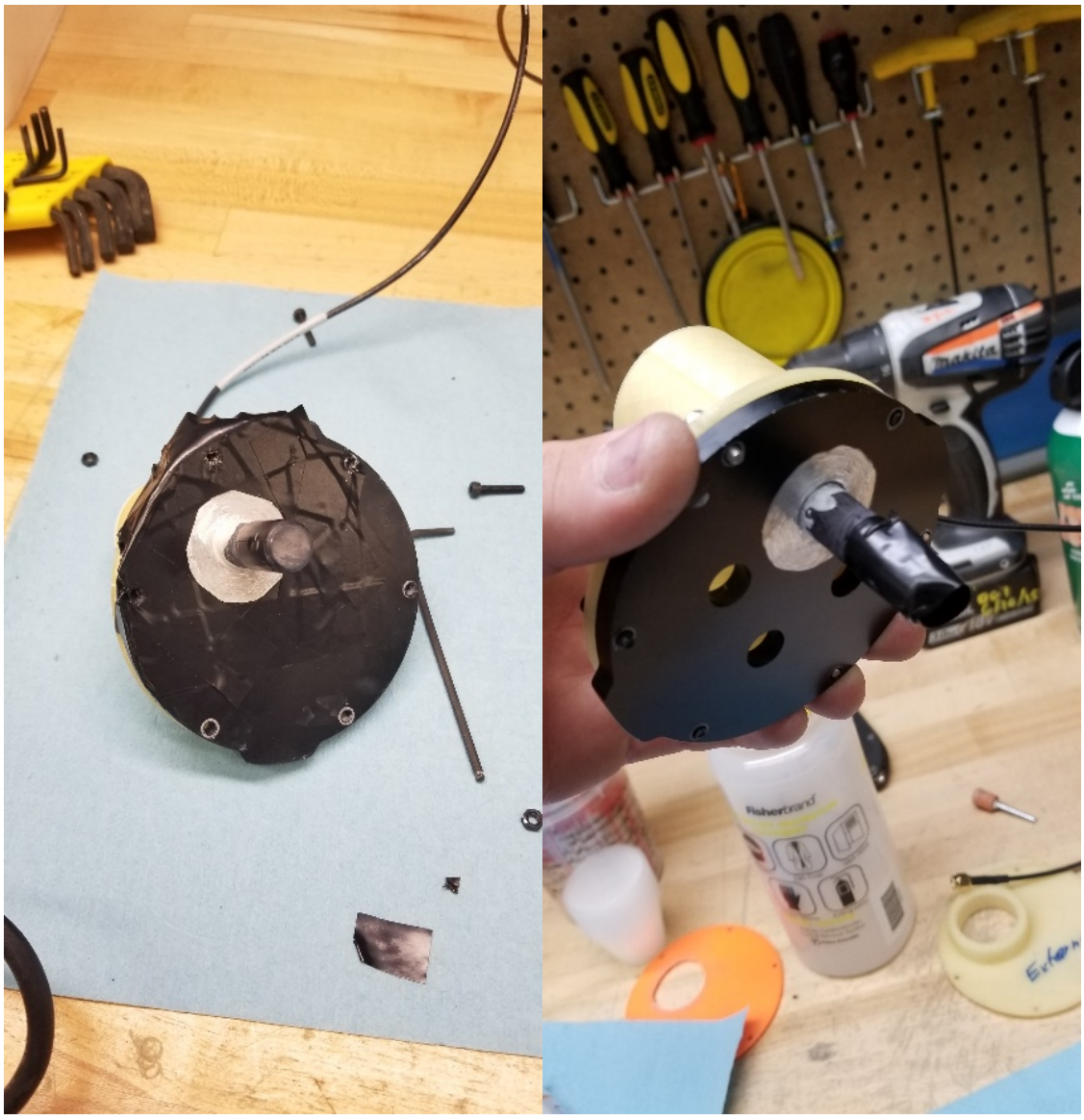


Figure 4: Taping of area around bare aluminum and probe end.

After installing the probe, install the potting_internal 3D print using some M3 screws to secure it to the end cap. Make sure it lines up with the correct side. You can support the end cap using a cup or mug. Once it is installed, mix and apply enough epoxy to fill the void to the rim. If bubbles appear, use a toothpick to remove them. Allow the epoxy to cure for 24-48 hours. Check the epoxy after a couple of hours to make sure that it has not leaked below the PVA jig. If it has, remove the jig and use isopropyl alcohol to clean up the epoxy.

After the internal side has cured, repeat the process with the external side.

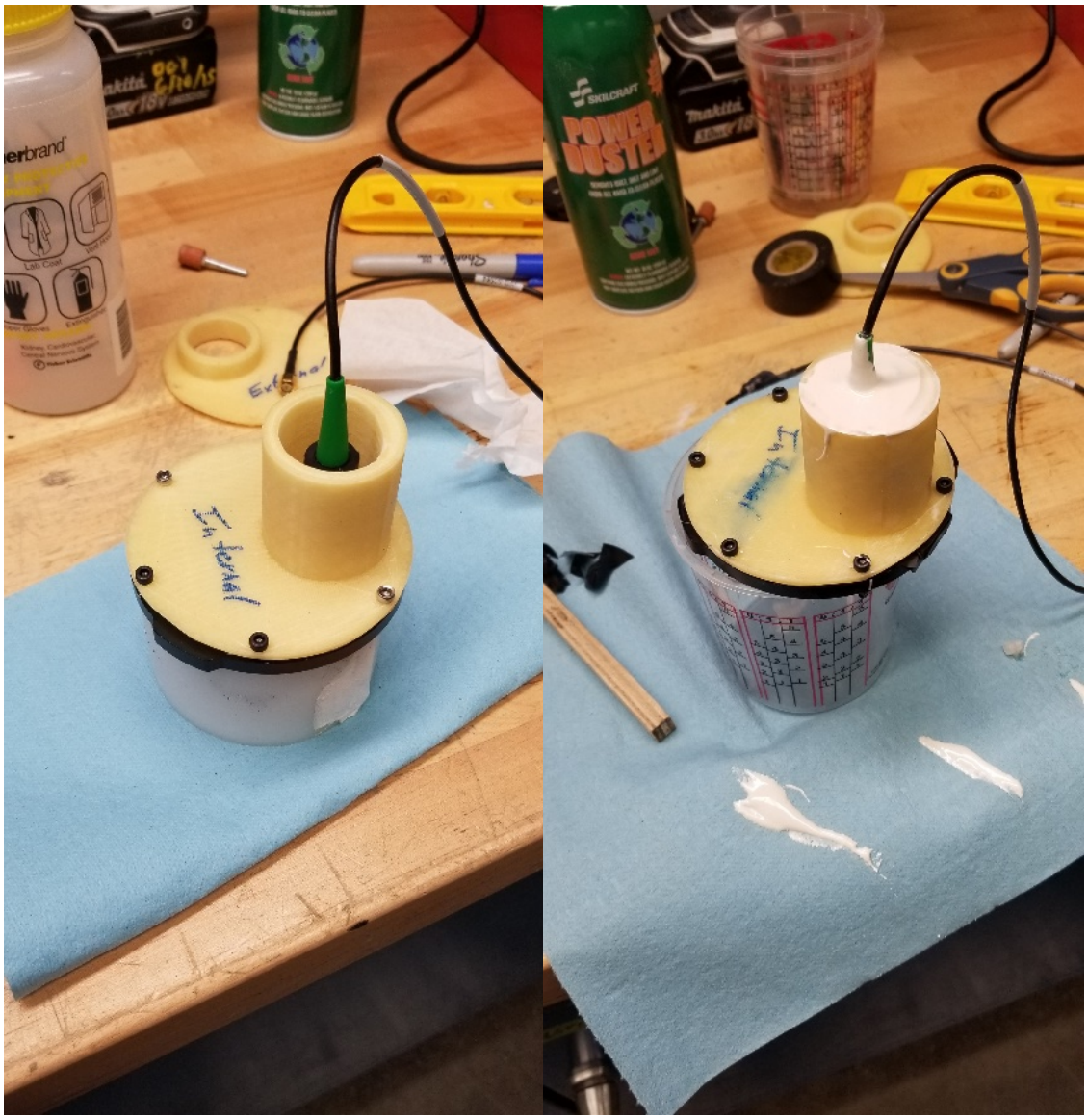


Figure 5: Installation of PVA jigs for epoxy application.

Once both sides have cured, place the setup up in a container of water. You may have to replace the water every few hours or so. Once the PVA 3D prints have dissolved way, clean up the epoxy with a scotch brite pad and a file to remove any burrs. Test fit the end cap flange to ensure they fit well together. If the epoxy make its way underneath the jig while curing, you can use a pair of hemostats or metal pick to remove the excess. Take care to not scratch the surface the end cap O-ring will seal to.

Figure 6: View of the external epoxy.

Installing the Temperature, Pressure, and Switch

Clean the end cap using isopropyl alcohol. Ensure the O-rings are clean and greased. The O-rings should be within the groove of each bulkhead and on the side of the end cap that is exposed to seawater. Insert each component into the end cap and add the nut. Tighten using a Blue Robotics tool or a crescent wrench. They should be tightened enough that you observe no gap between the bulkhead and end cap.

In these pictures, the DF13 connectors are removed from the sensors. If you do not want to remove them, Blue Robotics offers a DF13->pinout board. If you opt to remove the connectors, cut them off as close to the connector as possible.

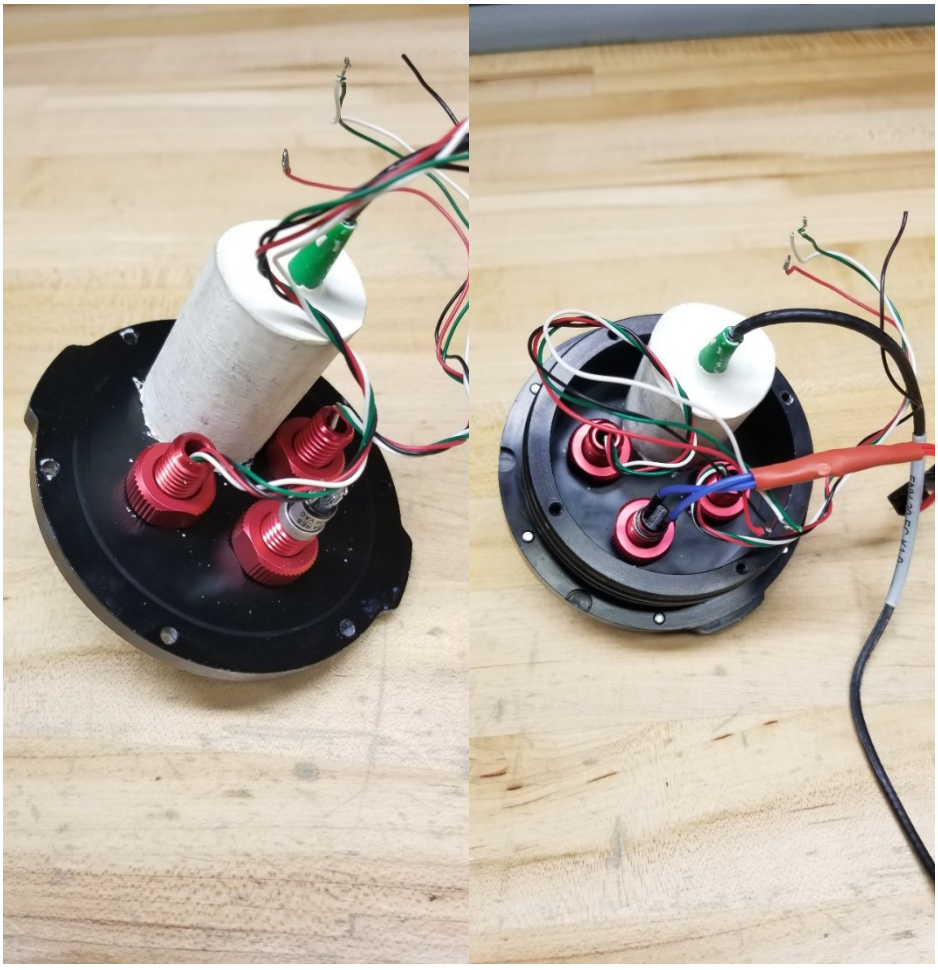


Figure 7: Installation of Blue Robotics sensors and switch.

Board Soldering

For access to the prototyping area, the M0 can be stacked on top of the Adalogger using stacking headers. The use of short feather headers allows for a more compact footprint. The first step is to solder a set of short male feather headers to the M0. A terminal block set can be installed in the prototyping area to make it easier to connect sensor wires to the board. You will need to solder wires from the GND, 3V, SCL, and SDA pins to the terminal block so that the conductivity, temperature, and pressure sensors can be connected to the board. The following pictures show one way to do that.

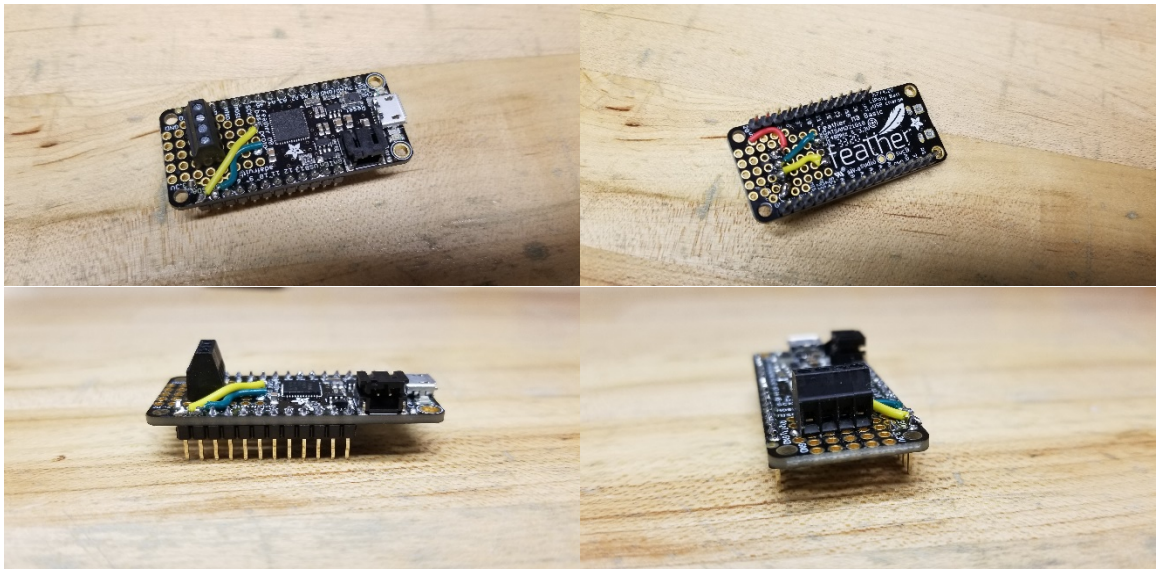


Figure 8: Soldering of male headers, wires, and terminal block to the Adafruit Feather M0 Proto

The Adalogger will stack below the M0. Next, solder a set of short female feather headers to the Adalogger. The following pictures show what the orientation should look like.

Figure 9: Soldering of female headers to the Adalogger Featherwing.

3D Printed Board Mount

The 3D printable mount allows you to mount the controller board stack and the Atlas Scientific EZO Isolation Board to the sensor flange. The mount is attached to the flange using 11mm M3 screws. The boards are attached to the mount using M2.5 thread nylon standoffs. This is one method of many to mount the electronics inside of the pressure case. Feel free to modify the drawings to suit your needs.

Figure 10: Attachment of the 3D printed mount.

Making the Switch Cable

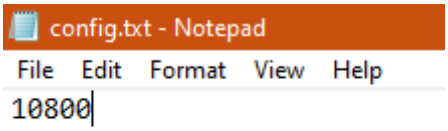
The addition of the Blue Robotics switch allows you to power cycle the sensor. This can help with conserving battery life and provides the ability to differentiate between profiles if using the CTD in a profiling fashion. If you are using a 5V rechargeable battery pack, you will need to splice the Blue Robotics switch wires into the V+ line on the USB cable.

Figure 11: Example of a 5V USB cable splice to the switch

config.txt

You have the option of placing a config.txt file on your microSD card. This file should contain only an integer on the first line that represents the number of seconds to wait in between samples. If unable to interpret the file, or if it doesn't exist, the CTD will default to one sample per second.

If you want to adjust the sampling rate to one sample every 3 hours, you will need to create a config.txt file like the screenshot below and then place it in the root directory of your SD card.



Bench Testing

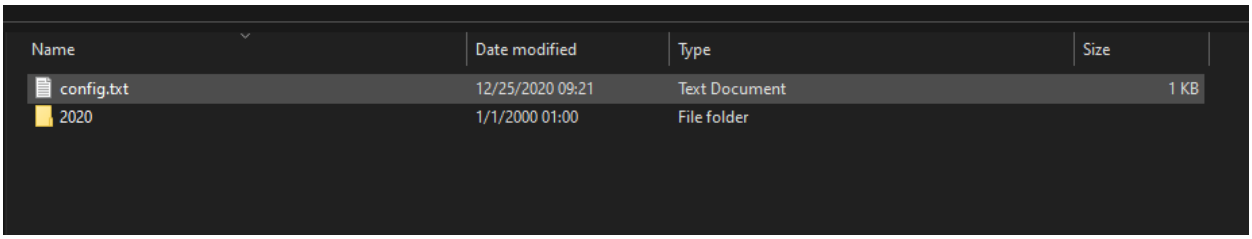
Once all the wiring is complete. Upload the ctdizzle sketch to your board.

Navigate to Tools > Serial Monitor. The setup sequence will begin once serial communication is established or after 30 seconds (whichever comes first). You will see something like the screenshot below. Confirm that the clock time and sensor values are appropriate.

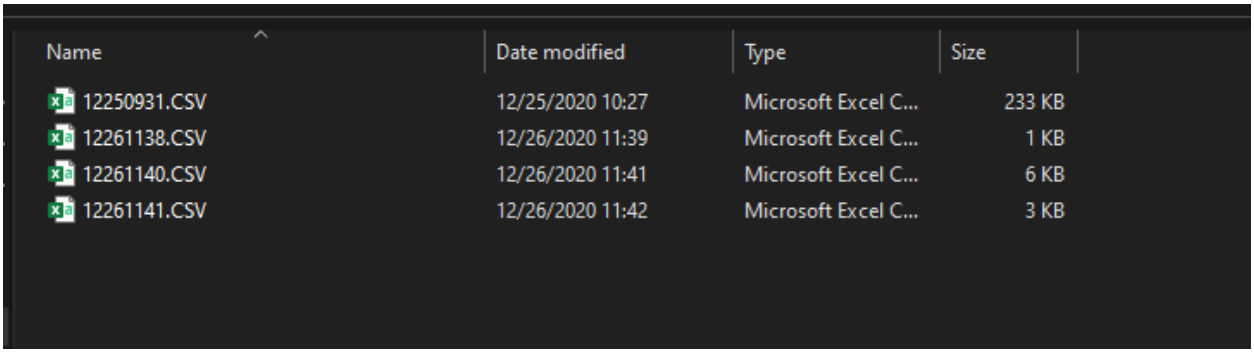
The first column is the datetime.
The second column is the conductivity, which should be zero if the sensor is in air.
The third column is the temperature. If this value is negative, check your SCL/SDA orientation.
The fourth column is the pressure, which should be zero or near zero in air.
The fifth column is the battery voltage. Since we are connected to the USB, this value is the voltage on the vbatt pin.

```
---------------------------------------------------------------
RTC initialized!
Moving to root/2020.
Data will be located at 2020/12261141.csv.
Log file initialized!
Reading configuration file.
Setting sampling rate to one sample every 1 second(s).
Configuration file read.
TSYS01 initialized!
MS5837 initialized!
2020-12-26T11:41:48,0.000000,23.429668,0.000000,4.318359
2020-12-26T11:41:49,0.000000,23.429668,0.010000,4.324805
2020-12-26T11:41:50,0.000000,23.421141,0.010000,4.324805
2020-12-26T11:41:51,0.000000,23.421141,0.010000,4.324805
2020-12-26T11:41:52,0.000000,23.421141,0.010000,4.324805
```

After you have let the unit run for a minute or so, disconnect the power and then insert the SD card into a reader device attached to a computer. Once you open the drive folder, the contents should look similar to the screenshot below. If you didn't create a config.txt file previously, it won't exist. If the year isn't 2020, then the folder name will also be different.



Opening the folder titled by year will show files for each time you turned the unit on.



If we open the 12261141.CSV file, we see data that matches the output from the serial monitor.

A	B	C	D	E
datetime	conductivity	temperature	pressure	voltage
2020-12-26T11:41:48	0	23.429668	0	4.318359
2020-12-26T11:41:49	0	23.429668	0.01	4.324805
2020-12-26T11:41:50	0	23.421141	0.01	4.324805
2020-12-26T11:41:51	0	23.421141	0.01	4.324805
2020-12-26T11:41:52	0	23.421141	0.01	4.324805
2020-12-26T11:41:53	0	23.421141	0.01	4.324805

Code

The main sketch for operation is the ctdizzle.ino file found in the ctdizzle folder. You'll notice that there are several .ino files in this folder, but they are all actually part of the same sketch! For more details on each function, please read the comments in the code.

board.ino

Upon upload, this sketch brings in the Wire and SPI libraries that allow the M0 to communicate to the sensors and clock.

error_cycle_led()

This function will turn the board LED on pin 13 on for 1 second and then off for 1 second.
It is usually put inside of a never-ending while loop and used as a visual indicator for an error.

get_voltage()

This function will calculate the battery voltage on the vbatt_pin. For the Feather M0, this is pin A7. The voltage is returned as a float.

initialize_comms(bps,i2c_clock)

This function initialize serial communications so that the board can talk to sensors.
bps refers to bits per second, or the baud rate for serial communication between the board and the host computer. i2c_clock refers to the clock frequency for I2C communication.

power_board_led(state)

This function will turn the board LED on pin 13 on or off, given a state. The state is either "ON" or "OFF".

power_leds(state)

This function will turn the board and EC EZO LEDs on or off, given a state. The state is either "ON" or "OFF".

wait_for_serial(seconds)

This function will wait for a serial connection to a terminal program on the host computer for X number of seconds before proceeding. While it is waiting, the board LED will cycle.

clock.ino

Upon upload, this sketch brings in the RTClib library that allows for communication with the clock on the Adalogger.

get_datetime()

This function returns the current date and time as a string in ISO-8601 format (YYYY-mm-ddTHH:MM:SS).

get_unixtime()

This function returns the current date and time as an integer in the number of seconds since 1970-01-01 00:00:00

initialize_rtc()

This function initializes the real time clock on the Adalogger. If the RTC has lost power and the board detects a serial connection to the computer, time is automatically updated. If no communication occurs with the clock, then the board error cycles the LED.

conductivity.ino

get_conductivity()

This function requests conductivity from the EC EZO and returns it as a float. There is a built-in wait time of 600ms to allow the EC EZO to perform its conversion.

power_ec_led(state)

This function will turn the EC EZO LED on or off, given a state. The state is either “ON” or “OFF”.

ctdizzle.ino

This is the main sketch for CTD operation. It contains the setup and loop functions. atmospheric_pressure is globally declared because it is used to “zero” the pressure sensor. This is of course operating under the assumption that you turned the sensor on above water and near sea level. interval is also globally declared. This value is pulled from a config.txt file located on the SD card. If no config file is found, then the interval defaults to 1. The CTD will then sample once per second.

setup()

This function initializes all sensors and sets up logging files.

loop()

This function will request data from each sensor and then save it to a log.
At the beginning of the loop, the number of milliseconds since the microcontroller started is recorded, then operations are performed, and the number of milliseconds since the microcontroller started is sampled again. The difference between these values is then subtracted from the interval value. This provides more accurate timing between samples.

eos80.ino

This sketch provides functions for computing depth, salinity, and density anomaly.

compute_depth(relative_pressure,latitude)

Computing depth requires that relative pressure be in decibars and that the latitude be defined. The return for this function is depth in meters.

compute_salinity(conductivity,temperature,relative_pressure)

Computing salinity requires conductivity in uS/cm, temperature in degC, and relative pressure in decibars. The return for this function is salinity in PSU.

compute_density_anomaly(salinity, temperature, relative_pressure)

Computing density requires salinity, temperature, and relative pressure. The return for this function is the density anomaly relative to 1000 kg/m3.

pressure.ino

This sketch provides functions for controlling the MS5837 pressure sensor.

get_absolute_pressure()

This function will return the total pressure reading (including the pressure exerted by the atmosphere). The return is in milibars. There is a built in delay of 50ms to allow for the pressure sensor to perform conversions.

get_relative_pressure(atmospheric)

This function removes consideration of atmospheric pressure and returns the water column pressure in decibars. You will need to supply the atmospheric pressure. This can be done by taking a pressure sample on start-up, assuming the unit is above water, or by using a standard atmospheric pressure of 1013.25 mbar.

initialize_pressure()

This function initializes the pressure sensor.

sd.ino

This sketch provides functions for setting up the SD card and creating log files. Upon upload, the necessary libraries and files are defined.

initialize_logger()

This function initializes the SD card module, reads the config file (if it exists), makes a new directory(if needed) and creates a new file to log to.

log_data(dt,ec,t,p,v)

This function logs data to a csv file. To alter the order of logging you will need to modify the function defined in loop(), the formatting in log_data() and the header string in make_file().

make_directory()

This function creates a new directory on the SD card that is titled by the current year. If the folder already exists, then nothing exciting happens.

make_file()

This function creates a new log file on power cycle. Data is saved to this file in a format defined by the header string in this function and the log_data() function. The file is named by the date and time that it was created. It is in the format of mmddHHMM.csv

read_config()

This function reads from a simple configuration file (config.txt) located in the root directory of the SD card. Currently, this config.txt file can only consist of a single integer, which then determines the number of seconds in between samples. If no config.txt file is found, the number of seconds in between samples defaults to 1s.

set_file_creation_datetime()

This is a function that assigns a date and time attribute to the created log file so that it is easier to identify datafiles when using a file explorer. This function is called within the initialize_logger() function.

temperature.ino

This sketch contains functions for driving the TSYS01 temperature sensor.

get_temperature()

This function requests temperature from the TSYS01. There is a built-in 50ms delay to allow the sensor to perform its conversion.

initialize_temperature()

This function initializes the TSYS01.

Conductivity Calibration

You can calibrate the conductivity sensor using methods as outlined by the Atlas Scientific User Manual. Atlas Scientific provides code examples on GitHub. https://github.com/Atlas-Scientific/Ezo_I2c_lib

Temperature Compensation

It should be noted that temperature plays a significant role in calibrating a conductivity probe. The CTDizzle code does not implement temperature compensation for conductivity, so that functionality must be added. The Atlas Scientific EC EZO offers temperature compensation of conductivity, but anecdotally this has been problematic. Atlas Scientific has not provided information on how this compensation is performed.

Things to Think About

Corrosion

The Blue Robotics end caps are made of anodized aluminum and the provided screws are made of 316 stainless steel. Two contacting dissimilar metals in seawater effectively form a battery. The metal lower on the galvanic series scale (the anode) will corrode before the metal higher in the series (the cathode). Stainless steel is higher on the scale than aluminum so the aluminum end cap will corrode first. Anodization of the aluminum helps prevent this a little, but over time repeated removal and installation of the stainless screws can damage the anodization, exposing raw aluminum to seawater. It is recommended that you inspect the end cap before each deployment for signs of corrosion. You can also prevent corrosion by adding an anode that is lower on the galvanic series than aluminum. A chunk of zinc is a common anode used on oceanographic equipment.

Anoxic corrosion of the 316 stainless screws is also possible. Anoxic refers to a lack of oxygen in the seawater. Stainless will severely corrode in anoxic environments if left for a significant amount of time. If the CTD has been deployed for a long amount of time, it is recommended that you inspect the screws for corrosion. A corroded screw can lead to loosening of the end cap and reduction in the effectiveness of the O-ring, leading to flooding of the sensor.

Batteries and Charging

The Adafruit Feather M0 has a built-in 3.7V Lithium Polymer battery charger. In order to charge a battery, the board must be connected to a 5V source via the microUSB connector. The board can also be powered with an 5V source, such as a rechargeable battery pack intended for charging cell phones. These types of battery packs must be connected to the microUSB connector. These above directions describe how to create a switch cable for a 5V rechargeable battery pack, but you can do the same for a 3.7V LiPo. The JST Battery Extension Cable found in the Bill of Materials will allow you to do this without modifying the battery cable.

This iteration of the CTDizzle does not consider power savings. You can likely improve battery life by forcing the controller and peripherals to enter low power states when not in use.

Data Quality

If you are familiar with commercially available CTDs, you'll notice that the CTDizzle does not compare in terms of accuracy, response times, and resolution. This problem does not necessarily preclude any open source CTD from being used in scientific study, but it does pose a challenge, particularly where high accuracy and resolution is needed. Currently there is a lack of robust calibration and quality conformance procedures for the CTDizzle and other open source CTDs.