Low-cost CTD Instrument

Arduino based CTD for autonomous measurement platform

Jonas Auråen

Master Thesis
Department of Physics, University of Oslo

UNIVERSITETET I OSLO

01.04.19

Abstract

This thesis describes the development of a low cost CTD instrument intended for use on an Autonomous Survey Vessel (ASV). The vessel will travel at water surface level, stop at regular intervals and release a sensor probe to do water profile measurements along a water column. The probe will primarily measure conductivity temperature and depth, and store these on an internal SD card. The probe is Arduino based and will also have wireless capability for communication while at surface level. The budget constraints of this project requires a low cost alternative of a normally very expensive instrument to be developed. The project is also an attempt at contributing to the efforts of those trying to make CTD instruments orders of magnitude cheaper then professional engineering equipment.

Contents

1 Introduction

Small automated vehicles doing automated measurements is an emerging trend especially in ocean research. Instead of using large and expensive research ships that can only be at one place at a time, one has started making smaller vessels that can be sent out in greater numbers to gather data. This thesis is a project within a greater project containing multiple master thesis with the end goal of making a surface vessel able to stop with regular intervals and do automatic water profile measurements. The task will be to make a simple lightweight and affordable sensor device, that can be suitable for use on an autonomous vessel.

Primarily this data is intended to assist sonar and echosounder equipment, to correct for refraction and absorption effects of sound waves. But there is also a large demand among researchers for cheaper oceanographic instruments in general. Another goal of this thesis is to contribute to the efforts of those trying to make cheap open source oceanographic instruments that are orders of magnitude cheaper then what is currently being offered.

The premise of this thesis is to prototype a CTD device at a fraction of the price of a commercial device, based on a number of existing projects. The idea is not to build all the individual parts from scratch but too combine multiple pre-existing efforts, like modular Arduino components, and open source sensors into a complete functioning CTD. The project will use the knowledge from similar attempts, like the OpenCTD project [20], Anwar Nazih Shabans Sound Speed Profiler [7] and attempt to improve upon them, while also experimenting with some novel ideas. The pre-existing projects serve as proof of what works and what doesn't, and lessons are taken from these to lessen the burden of trial and error in this project. Hopefully this can contribute to the long term goal of creating ultra-low-cost open source instruments capable of replacing professionally manufactured equipment.

.

.

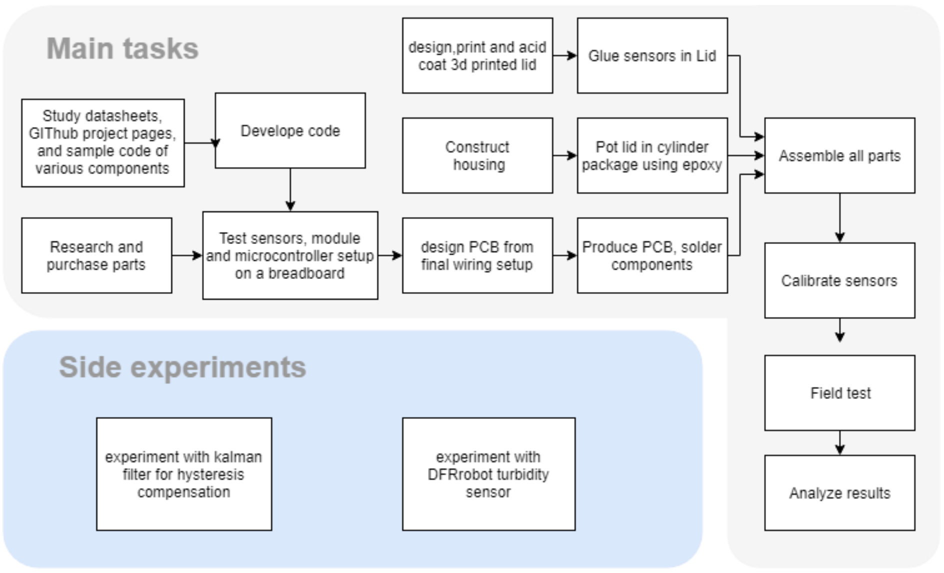

Main tasks
design,print and acid coat 3d printed lid
Glue sensors in Lid
Study datasheets, GIThub project pages, and sample code of various components
Develope code
Construct housing
Pot lid in cylinder package using epoxy
Assemble all parts
Research and purchase parts
Test sensors, module and microcontroller setup on a breadboard
design PCB from final wiring setup
Produce PCB, solder components
Calibrate sensors
Field test
Analyze results
Side experiments
experiment with kalman filter for hysteresis compensation
experiment with DFRrobot turbidity sensor

Figure 1 Project overview

2 What is a CTD

A CTD is one of the most common instruments in the oceanographer's arsenal. It is a collection of different sensors that take measurements collected by a control unit, combines them, and produce various other parameters. The letters CTD is an acronym for Conductivity, Temperature, and Depth, though in reality it does not measure depth, but rather pressure to calculate depth. The CTD control unit can also often do other things like communicate wirelessly with a surface vessel, receive GPS position, store data on a memory device, etc. Differences in salinity, temperature, and depth are the primary parameters that distinguish different water masses. There are large amounts of existing datasets containing CTD measurements from lakes and oceans around the world, that can be used to observe long term changes in climate and ocean conditions. A CTD can be used in a variety of ways. One way is to let the instrument ascend or descend in a water column, resulting in a vertical profile of the water. This gives a snapshot of the changes in the different layers of the water column. CTDs can also be deployed stationary and monitor changes at a fixed point over time, or they can be dragged horizontally to show anomalies in the horizontal plane.

Several parameters can be derived from the main three measurements made by a CTD. For example, measurements of conductivity, temperature, and pressure can be combined to calculate salinity. Pressure and temperature can be used to calculate depth, and temperature, salinity, and depth can be used to calculate sound speed.

2.1 Conductivity

Conductivity is a measure of the degree of which a material is capable of conducting electricity. In water this is closely related to the amount of ions dissolved in the fluid. An Ion is an atom or molecule that has either a positive or negative electrical charge, meaning either a surplus or deficit of electrons as opposed to protons. These ions come from electrolytes, which are salts and inorganic materials that dissolve into positively or negatively charged ions (cations or anions). Even though the water will become increasingly conductive with the addition of extra ions, it will remain electrically neutral, since the electrolytes will split into equal amounts of cations and anions. Conductivity is usually measured in milli- or micro Siemens per centimeter but can sometimes also be measured in Ohms per centimeter. One Siemens is the reciprocal of one ohm ($S = 1/\Omega$). The standard way of reporting conductivity is to report the conductivity measurement at 25° C or corrected to 25° C. This is called the specific conductance. The temperature of water will affect conductivity and will have to be corrected for this to be compared. Conductivity is not the same as conductance. Conductance depends on the length of the conductor and is measured in Siemens or Ohms. Conductivity on the other hand is the conductance across a specified length (usually 1 cm) and is measured in siemens per centimeter.

2.1.1 Salinity

Salinity is a term that refers to the concentration of dissolved salts in water. These salts dissolve into ions that are usually the largest contributor to the conductivity of water.

Cl
-
CHLORIDE ION
Na
+
SODIUM ION
Mg
2+
MAGNESIUM ION
S
2-
SULFATE ION
Ca
2+
CALCIUM ION
K
+
POTASSIUM ION
Br
-
BROMINE ION

Figure 2 The most common ions in seawater]15]

Measuring the salinity directly is problematic, as you cannot simply evaporate the water and measure the remaining salt. Chloride ions will disappear in the process. What is more common is to do a conductivity measurement and then assume the salinity based on the known relationship between conductivity and salinity in a known standard, such as seawater. The resulting assumption is given in PSU or practical salinity units. Seawater has a fairly uniform mineral profile. The most common ions in seawater are chloride , sodium, magnesium, calcium , potassium and bromine. A Salinity calculation based on a normal seawater mineral profile will give accurate results in most areas, though some exceptions exist. In freshwater this will vary a lot more and is largely dependent on the surrounding landmass. Freshwater usually has a higher bicarbonate ratio while seawater has greater sodium and chloride concentrations [15]

There are several different units in use to measure salinity. The old standard was to use either parts per thousand or grams/kilogram (1 ppt = 1 g/kg) or in some freshwater sources, mg/L. Now the most common unit is the unitless PSU, which is most likely to be used in database archives. In 2010 a new standard was introduced called absolute salinity, or TEOS-10. This is also the SI unit of salinity but is rarely used unless extra precision is needed.

2.1.2 Conductivity and temperature

The conductivity of water will increase as the temperature increases. A 1°C increase in temperature will typically result in a 2-4% increase in conductivity. This is due to increase in ionic mobility as well as the increased solubility of many salts and minerals.

Because of this dependency water conductivity is measured as specific conductivity, with measurements made at or corrected to a standardized temperature, usually 25°C.

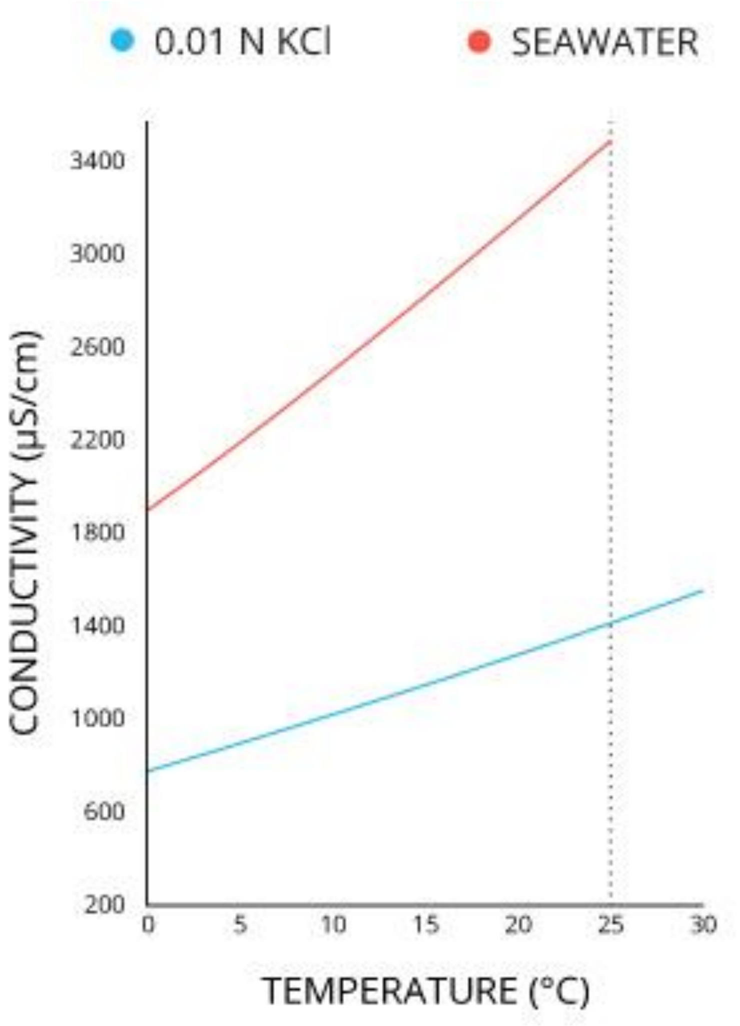


Figure 3 Temperature dependency of conductivity[15]

2.2 Temperature

Temperature is another very important feature of water. It can be measured by many different measurement principles but in a CTD this is usually done by a thermistor or a digital IC sensor. Temperature has a major impact on biological activity and growth of aquatic organisms in water. Up to a point, higher water temperatures correspond to more biological activity. Most water-based organisms are cold-blooded and unable to regulate their core body temperature. They have preferred temperature ranges and are sensitive to variations in temperature going too high or too low. Water chemistry is also influenced by temperature. The rate of chemical reactions, like the uptake of oxygen in water, will generally increase with higher temperature [14].

Temperature also changes water density. The denser water will sink and settle in layers of water with similar temperature. This effect is called thermal stratification. In lakes during summer the surface layer will be heated by sunlight causing a large temperature difference between the upper and lower layers of the water. This leaves the layers rigid since liquids with large difference in density don’t mix as easily. In autumn when the weather cools down again and the water layers become more uniform, a lot more upward and downward mixing of water can happen. This is important for all marine ecosystems as the dissolved oxygen and nutrients from the surface gets mixed into the deeper waters. When the water is thermally stratified, the bottom layers can become anoxic, or contain zero dissolved oxygen. Water reaches its maximum density at 4°C. [13]

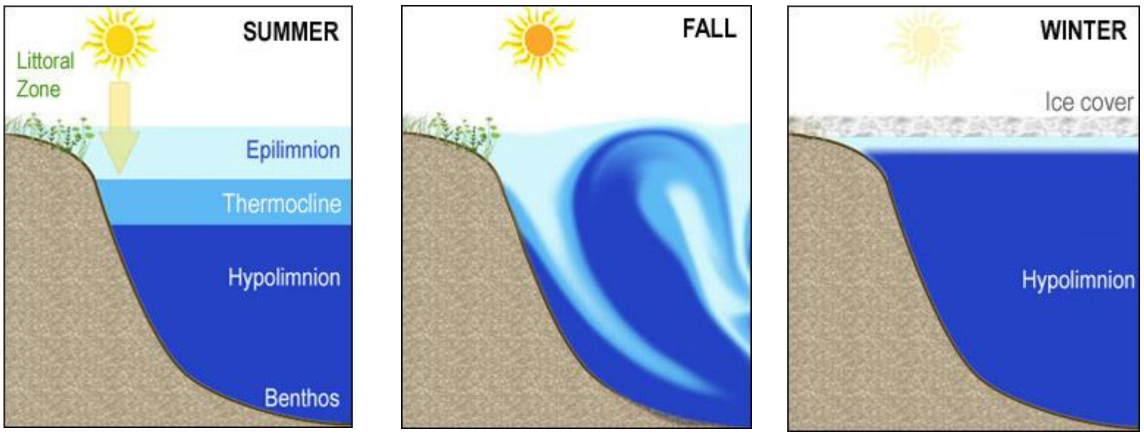


Figure 4Mixing of water layers in different seasons [12]

Water has a higher thermal capacity then air and will heat up or cool down much slower. This can act as a temperature capacitor for surrounding land masses. Coastal areas often have milder winters and cooler summers, whereas inland locations can have much larger variability between summer and winter

2.3 Pressure

Pressure is usually measured by a pressure gauge that detect changes in the shape of a coil of wire or tube of fluid due to outside pressure. It measures the amount of density, or total weight of water over the sensor. The relationship between pressure and depth are very closely related, so a pressure sensor is usually able to calculate depth pretty accurately. In stationary CTDs, an accurate pressure sensor can be used to detect tidal cycles and wave cycles

2.4 Turbidity

Turbidity is an often hard to measure optical property of water. It is the total amount of particles suspended in the water and determines the visibility in the water body. More suspended solids mean hazier/cloudier water. This is often measured by optical devices emitting light and monitoring the amount of scatter due to collisions with solid particles in the water. Typically, these particles are silt, clay, algae, plankton or any other type of finely divided matter. Turbidity can be used to determine the drinkability of water or can be used as an indicator of pollution. High turbidity is a sign of unhealthy water. It can provide shelter and sustenance for pathogens and lead to outbreaks of waterborne diseases if left untreated. The unit of measurement is Nephelometric Turbidity Units or NTUs.

2.4.1 CTD data and sonar

Perhaps the most important aspects of the CTD with respects to this project are those that relate to the sonar. Several projects in the hydroacoustic research group at the University of Oslo use echo sounders, sonars and hydrophones for detecting features of objects located under water. The main intention of the proposed automated surface vehicle carrying the CTD will be to collect data to aid the sonar equipment. There are several ways CTD data can help to interpret and to aid the deployment of sonar equipment. Primarily this has to do with absorption and sound speed in water.

2.4.2 Absorption

When sound moves through water, the acoustic energy is lost through the process of absorption. The intensity of a sound wave decreases with the distance along the propagation path. There is a large frequency dependent difference in absorption in different types of water. For example, the absorption in seawater at frequencies between 5 and 50 kHz is 30 times higher than in distilled water (Urik,1983, p.104). Several mechanisms contribute to this effect. First there are viscous losses that occur, both in fresh and salty water. Viscosity causes friction that converts acoustic energy into heat and is a direct loss of energy to the medium.

Rayleigh's expression:

$$\alpha = \frac{16\pi^2 \mu_s}{3\rho c^3}$$

Is an attempt to express the absorption effects of viscosity through an absorption coefficient. In this equation α is the absorption coefficient, μ_s the shear viscosity, ρ the density, c the sound velocity and f the frequency.

But viscosity by itself does not account for all the differences in absorption at different frequencies. Several other effects have been observed that affect this process. Other prominent contributors are the molecular relaxation of certain minerals and compounds. Molecular relaxation is a pressure-induced process which dissociates ions from molecules. For example, the magnesium molecule can under the varying pressure of a sound wave dissociate or re-associate its constituent ions, changing the absorption coefficient of the water. This effect only occurs below a frequency threshold, as the molecular relaxation takes a certain amount of time to manifest. At higher frequencies, the sound pressure cycles too fast for the relaxation to occur. In the 2-500 kHz range the magnesium sulphate is the main cause of absorption, and at the lower frequencies boric acid become a contributing factor[1].

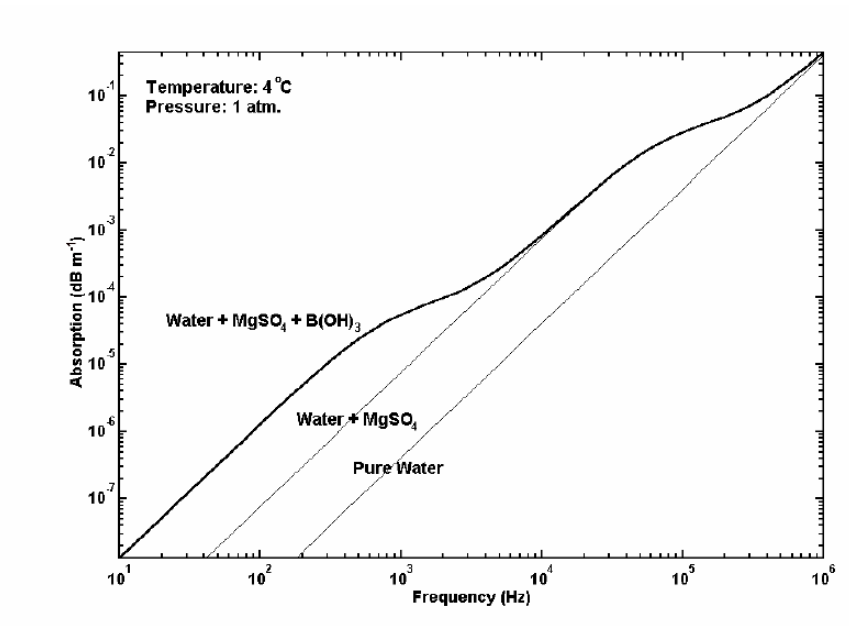


Figure 5 Absorption coefficients in sea water according to the Fisher-Simmons expression [4]

The exact relationship between these effects and the absorption rate is still not perfectly understood, and there are several different equations trying to describe it. One such is the Fisher-Simmons expression [2]:

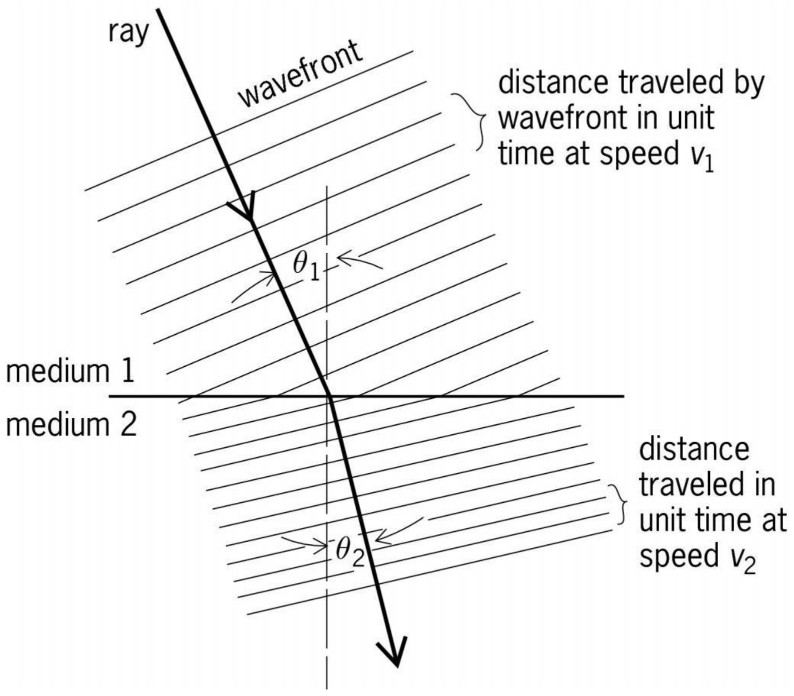
$$\alpha = A_1 P_1 \left(\frac{f^2}{f_1^2 + f^2}\right) f^1 + A_2 P_2 \left(\frac{f_2}{f_2^2 + f^2}\right) f_2 + A_3 P_3 f^2$$

Where the terms A_1, A_2, A_3, f^1, f^2 are functions of temperature and P_1, P_2, P_3 are functions of pressure. The three parts of the equations describe the contribution of boric acid, magnesium sulfate and viscosity respectively. (Urik,1983, p.105). Note that all the required measurements for the calculation of the absorption coefficient can be acquired by a CTD.

Knowing the absorption factor is important for sonar because the rate of absorption is the limiting factor that determines the highest useful frequency for object detection at a given range. CTD data can help determine what frequencies can be used at what ranges, when setting up sonar equipment. For example, the absorption rate in fresh water is relatively small up to about 200kHz, since little molecular relaxation occurs. This means that higher frequencies then what could be used in seawater, can be used at the same maximum range of detection in freshwater

2.4.3 Sound speed

Sound speed can also be determined from CTD data. The velocity of molecules increases with higher temperatures, and the density and salinity of the water makes the sound waves travel faster as the wave vibrates with more molecules. This is important for sonar applications because sound refracts when it changes speed. If a sonar transmits horizontally, the sound waves will bend because the sound speed changes with depth. A near horizontally aligned sound beam experiencing decreasing temperature



ray
wavefront
distance traveled by wavefront in unit time at speed v_1
θ_1
medium 1
medium 2
distance traveled in unit time at speed v_2
θ_2

Figure 5 Physical basis of Snell's law [11]

with depth will bend downwards since the lower part of the wave-front will travel in colder water with slower sound speed. In the pictures you can see the sound wave illustrated as a ray, changing direction when it changes speed. If the gradient is negative, like it often is in the upper layers the sound waves are bent downward. If the gradient is positive, as it is in deep ocean where the pressure causes the sound to travel faster, the waves are bent upwards.

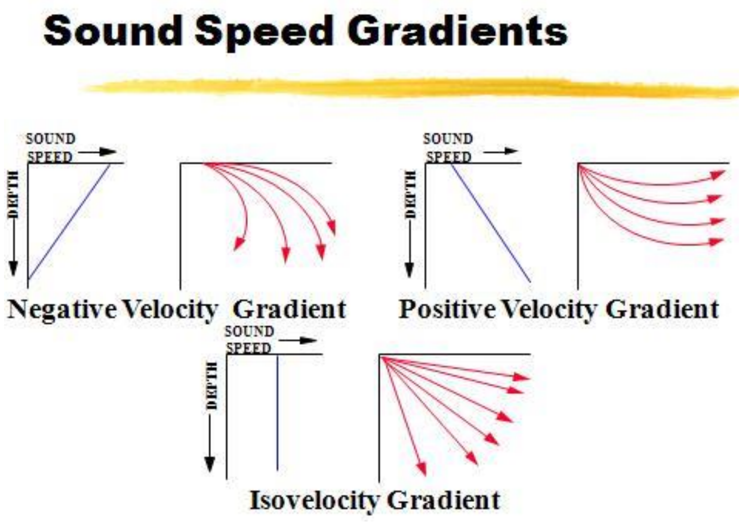


Figure 6 Sound Speed Gradients [10]

Temperature is the dominant variable affecting sound speed in the upper few hundred meters of water, though in some polar regions varying levels of salinity can also play an important part. Pressure starts being important deeper in the ocean, where the temperature stabilizes.

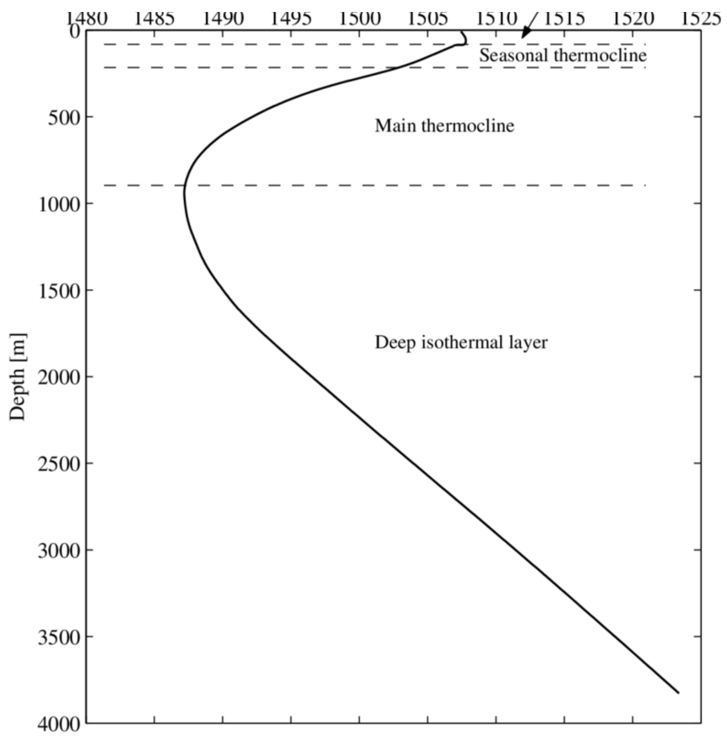


Figure 7 Typical ocean velocity profile divided into layers [1]

A typical sound speed profile in seawater will look like the figure on the right. The sea is divided into several layers. At the top there is the surface layer, which will be influenced by daily and local changes in wind and weather. Then there is the seasonal thermocline, where the temperature steadily declines with depth. This is characterized by a negative sound speed gradient. In the summer this will be more pronounced than in the winter.

Below this is the main thermocline, where the temperature continues to decline, but is only slightly affected by the seasons. Finally, there is the deep isothermal layer, where the temperature stays fairly constant and the increases in pressure will lead to a positive sound speed gradient

For sonar this has significant implications. The ocean will have an upper layer where it has the potential to trap acoustic energy. The surface layer will often contain an isothermic mixed layer. Instead of having a stable negative temperature gradient, like the water below it, the mixing of heated surface water and cooler water from below can lead to several meters where the temperature gradient is neutral. This in turn can lead to a positive sound speed gradient as the pressure increases with depth. The point where the sound speed gradient turns negative is called the Sonic Layer Depth and can cause an upper duct where the sound waves sent from a fixed point will bend upwards or downwards from the layer, causing a shadow zone where detection is difficult.

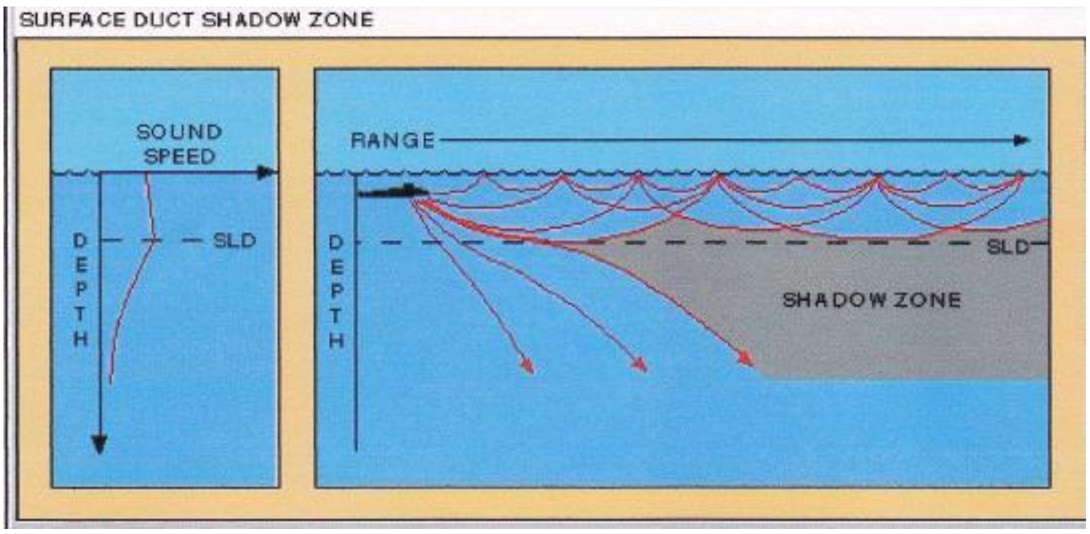


Figure 8 Surface Duct Shadow Zone [10]

2.4.4 Calculating sound speed from CTD data

Several attempts have been made to describe the behavior of sound speed in water. Even though it seems to depend entirely on salinity, temperature and depth, (the exception being contaminants such as air bubbles and biological organisms), the relationship between these are extremely complicated. Most equations are based on extensive measurements done in controlled environments with varying levels of temperature, salinity and pressure. Complex polynomials are then created to best fit the measurements. The international standard equation, or UNESCO equation, for sound speed is made by Chen and Milero [3]. It has later been recalculated by Wong and Zhu [5] following the adoption of the International Temperature Scale in 1990. The equation looks like this:

$$c(S,T,P) = Cw(T,P) + A(T,P)S + B(T,P)S^{3/2} + D(T,P)S^2$$

$$\begin{aligned} Cw(T,P) = {} & (C_{00} + C_{01}T + C_{02}T^2 + C_{03}T^3 + C_{04}T^4 + C_{05}T^5) + \\ & (C_{10} + C_{11}T + C_{12}T^2 + C_{13}T^3 + C_{14}T^4)P + \\ & (C_{20} + C_{21}T + C_{22}T^2 + C_{23}T^3 + C_{24}T^4)P^2 + \\ & (C_{30} + C_{31}T + C_{32}T^2)P^3 \end{aligned}$$

$$\begin{aligned} A(T,P) = {} & (A_{00} + A_{01}T + A_{02}T^2 + A_{03}T^3 + A_{04}T^4) + \\ & (A_{10} + A_{11}T + A_{12}T^2 + A_{13}T^3 + A_{14}T^4)P + \\ & (A_{20} + A_{21}T + A_{22}T^2 + A_{23}T^3)P^2 + \\ & (A_{30} + A_{31}T + A_{32}T^2)P^3 \end{aligned}$$

$$B(T,P) = B_{00} + B_{01}T + (B_{10} + B_{11}T)P$$

$$D(T,P) = D_{00} + D_{10}P$$

Where T is temperature in Celsius, S is salinity in Practical Salinity Units, and P is pressure in Bar. The A's and C's are numerical coefficients given by a table

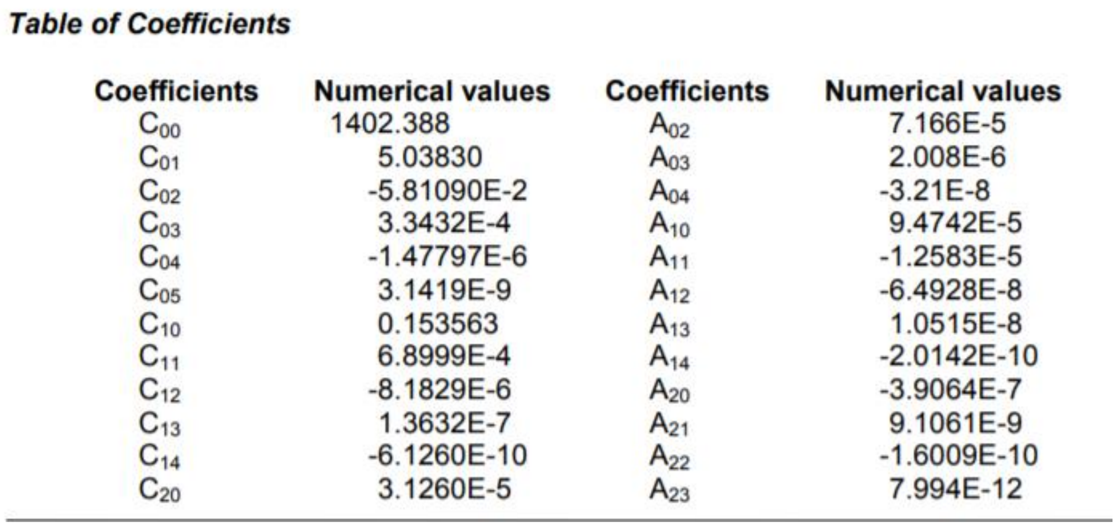


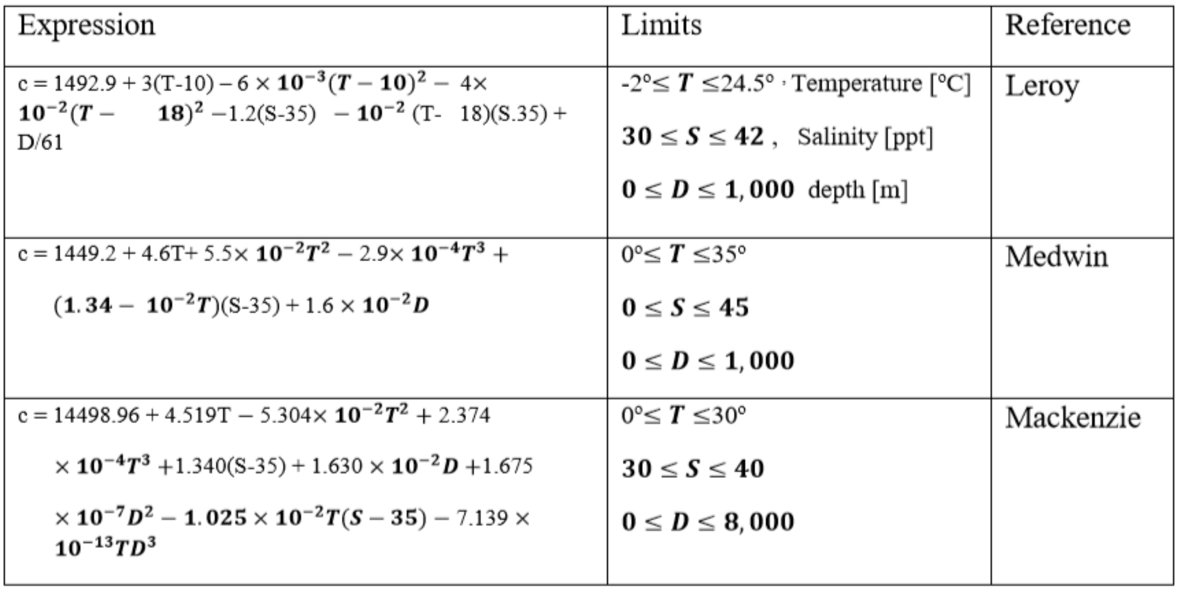
Table of Coefficients

Coefficients	Numerical values	Coefficients	Numerical values
C_{00}	1402.388	A_{02}	7.166E-5
C_{01}	5.03830	A_{03}	2.008E-6
C_{02}	-5.81090E-2	A_{04}	-3.21E-8
C_{03}	3.3432E-4	A_{10}	9.4742E-5
C_{04}	-1.47797E-6	A_{11}	-1.2583E-5
C_{05}	3.1419E-9	A_{12}	-6.4928E-8
C_{10}	0.153563	A_{13}	1.0515E-8
C_{11}	6.8999E-4	A_{14}	-2.0142E-10
C_{12}	-8.1829E-6	A_{20}	-3.9064E-7
C_{13}	1.3632E-7	A_{21}	9.1061E-9
C_{14}	-6.1260E-10	A_{22}	-1.6009E-10
C_{20}	3.1260E-5	A_{23}	7.994E-12

Table 1 UNESCO equation table of coefficients [6]

This equation is one of the more comprehensive equations, both in terms of range of validity and computational requirements. The equation is valid in the range of 0 to 40 ℃ in temperature, 0 to 40 parts per thousand for salinity, and 0 to 1000 bar for pressure. Other equations like the Del Grosso equation is similarly comprehensive but has a more restricted range of validity. It is preferred by many authors since the UNESCO equation has a number of known flaws. For example, there have been several experiments proving that the UNESCO equation will give incorrect values under high pressure conditions [9].

In addition to these there are several simpler equation made for practical use, but with restrictions in range of validity and reduction in accuracy:



Expression	Limits	Reference
$c = 1492.9 + 3(T-10) - 6 \times 10^{-3}(T-10)^2 - 4\times 10^{-2}(T-18)^2 - 1.2(S-35) - 10^{-2}(T-18)(S.35) + D/61$	$-2° \leq T \leq 24.5°$ · Temperature [°C] $30 \leq S \leq 42$, Salinity [ppt] $0 \leq D \leq 1{,}000$ depth [m]	Leroy
$c = 1449.2 + 4.6T + 5.5\times 10^{-2}T^2 - 2.9\times 10^{-4}T^3 + (1.34 - 10^{-2}T)(S-35) + 1.6 \times 10^{-2}D$	$0° \leq T \leq 35°$ $0 \leq S \leq 45$ $0 \leq D \leq 1{,}000$	Medwin
$c = 14498.96 + 4.519T - 5.304\times 10^{-2}T^2 + 2.374 \times 10^{-4}T^3 + 1.340(S-35) + 1.630 \times 10^{-2}D + 1.675 \times 10^{-7}D^2 - 1.025 \times 10^{-2}T(S-35) - 7.139 \times 10^{-13}TD^3$	$0° \leq T \leq 30°$ $30 \leq S \leq 40$ $0 \leq D \leq 8{,}000$	Mackenzie

Table 2 Other expressions for speed of sound in water [7]

3 Open Source Oceanographic Instruments

Monitoring the oceans and fresh waters is an important goal for a multitude of reasons. The ocean, more so than land masses, are deeply impacted by increasing carbon dioxide emissions from human activities. The changes in water temperature, acidification and deoxygenation of water leads to changes in oceanic circulation and chemistry that can threaten the biological ecosystems underwater, as well as cause unpredictable and destructive weather events on land. In a study of the Virginia Institute of Marine science it is estimated that the decline in ocean health can cost the global economy $428 billion per year by 2050, and 1.979 trillion by 2100 [27]. These are critical environmental processes happening underwater that needs to be monitored and understood, and to do this a vast amount of data and measurements are needed. The cost of such ventures is currently what is prohibiting this from happening at the scale that is needed. The measurement sites are often remote and the ocean is a harsh environment that requires specially tailored equipment and technology Usually robust equipment mounted on elaborate mooring systems or research ships are deployed, but these are tremendously expensive and the equipment can be bulky and dangerous to handle. The technological developments in areas such as GPS and autonomous vehicles has opened up the possibility of newer and smarter ways of doing this, and with other developments in consumer electronics greatly decreasing costs and increasing availability of products one could expect a wave of cheaper and smarter underwater instruments to be at hand. This however has yet to occur. The arena is still dominated by professional engineering companies, creating expensive solutions for small scale use. Daniel P. Langis (2015) of the California State Maritime Academy lists three reasons why the developments of ultra-low-cost instruments have failed to materialize.:

> *1. Many individuals have made attempts at portions of ultra-low-cost sensor development (such as conducting studies on individual low-cost sensors) but little work has been done to propose an end-to-end solution that would make them a reality. A complete solution must consider how to integrate complex issues such as product development, data quality, testing, calibration, maintenance, and user interaction – all while driving down total cost (Blanchard, 2008, p.10).*
>
> *2. The instrumentation industry is dominated by a limited number of commercial companies who have a strong hold on the market. Those companies create first-rate products and do provide reliable end-to-end solutions for the issues above. Although*

expensive, the costs for oceanographic instruments are incorporated into budgets as the price of conducting research; programs also have large amounts of capital already invested in instruments. These factors reduce pressure and create a justifiable reluctance to develop and adopt new technologies.

3. Organizational processes and traditions are very hard to change. The adoption of any revolutionary practice may redefine processes, alter job responsibilities, and create internal disruptions which require significant organizational change. The development of ultra-low-cost sensors must also consider how the technology will be utilized and provide time and recommendations for adopting new (Langis,2015, p.5)

To better understand how low-cost instrumentation could be applied in oceanography it is useful to understand the strengths and weaknesses of the current available approaches. This is addressed in a document published by The National Academy of Science Committee where they list problems associated with the current infrastructure of ocean research and detail trends and strategies for the coming two decades in this area.

3.1.1 Ships

Research ships can be equipped with numerous gadgets and instruments and can deploy mooring systems to remain on sites for remote observation. They have a distinct advantage over other platforms in their versatility and mobility, but they are also increasingly costly. In the US, according to the National Ocean Council's Federal Oceanographic Fleet Status Report (2013, p. 20) "*Fuel costs have increased some 400% since 2003, aging vessels require higher maintenance costs, personnel costs for salaries and training are increasing, and new safety and environmental standards are becoming more stringent*" (National Ocean Council, 2013, pp. 19-20). The consequence is that the ships available are used less, and that researchers are actively searching for alternate methods of doing measurements to reduce ship time

3.1.2 Moorings

Mooring systems are instruments mounted on a cable traversing from an anchor element at the ocean floor to a buoy at the surface level. The surface element often contains satellite communication and tracking devices. The equipment is transported by ships and can stay on the same spot for a long time, sometimes providing comparable data sets going decades back in time. These will still be useful in the coming decades for their high frequency fixed location data, but their mobility, their complexity, and the cost and safety concerns of their deployment make them unsuitable for the type of mass measurements that other platforms could provide.

3.1.3 Autonomous vehicles

Autonomous vehicles carrying instruments is a new approach made possible by GPS and satellite technology. These can be UAVs, gliders with expendable CTDs or self-sustaining floating devices. The ARGO is an example of such a device that has been implemented in a global program that has already revolutionized ocean data measurement. Argo is an international collaboration that collects temperature and salinity data from the upper 2000 meters of the global oceans. Currently about 3200 floats are in use, with another 800 being deployed per year[17]. The data is gathered from an autonomous float that spends most of the time submerged at a parking depth. At this depth it is neutrally buoyant which means it has a density equal to the surrounding water. At every 10 days or so the float will pump fluid into an external bladder and gradually rise to the surface while collecting measurement data. When it surfaces it transmits the data to a satellite before it submerges again. Each float is designed to do about 140 cycles and last about 4 years. The construction cost for one device is about 15 000 $ and about twice that with operating costs. This is still fairly expensive and is expected to rise, as the ARGO program has several new upgrades planned such as new biological sensors, chemical sensors and Iridium communication. The National Ocean Council [16] summarizes that:

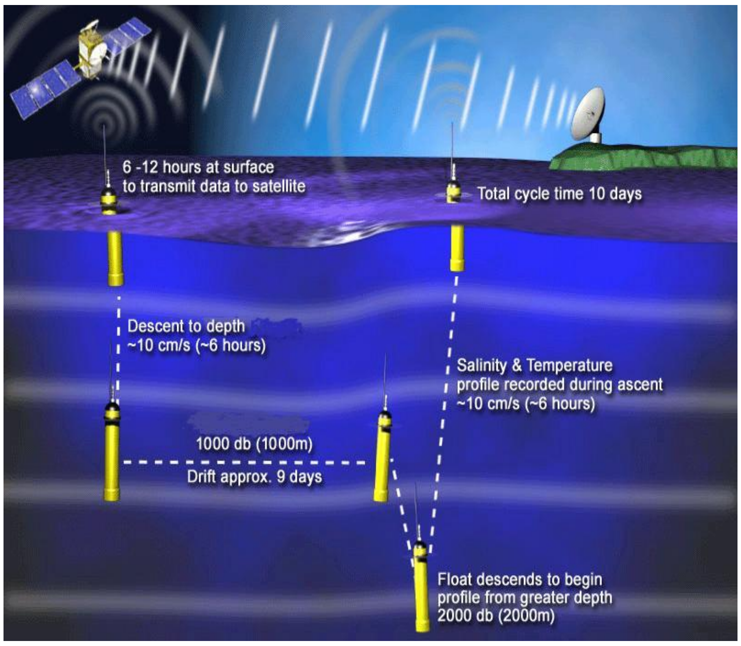


Figure 11 How Argo floats work [17]

> *In the past two decades, use of floats, gliders, ROVS, AUVS, and scientific seafloor cables has increased; use of ships, drifters, moorings, and towed arrays have remained stable; and use of HOVs has declined. Based on these trends, utilization and*

capabilities for floats, gliders, ROVs, AUVs, ships, and moorings will continue to increase for the next 20 years, and HOV use is likely to remain stable. Ships will continue to be an essential component of ocean research infrastructure; however, the increasing use of autonomous and unmanned assets may change how ships are used. (2011, p. 31)

3.1.4 Low cost instrumentation projects

While there are a number of exciting technological advancements in oceanographic equipment, there is still a widely recognized need for less expensive instrumentation. The spatial resolution of data can be greatly increased and models used for prediction can be significantly improved by having a large number of discrete measurements available. This will most likely be resolved by creating a large network of low-cost instruments that are orders of magnitude cheaper then what they are today. The National Ocean Council[16] thinks such a development is imminent. In their report they write:

Circa 1990, there were only a few 8-bit microprocessor systems with sufficiently low power consumption for autonomous deployments, and they had volatile solid-state memory and limited computational power and data storage. In 2010, processors with orders-of-magnitude-higher computational power can navigate systems, command sensors and actuators, adapt missions, and retain gigabytes of data in robust solid-state memory. There have been parallel improvements in power availability, including the transition from alkaline to lithium batteries. (National Ocean Council, 2011, p. 28-29)

3.1.5 The Arduino Platform

One way to take advantage of the consumer-driven advancements in microelectronics is to utilize the Arduino platform. Arduino is an open source platform that consists of a physical programmable circuit board and an Integrated Development Environment (IDE). The board has programmable digital and analog I/O ports and can communicate to other devices using standardized protocols such as UART, I2C and SPI. The main advantages of the Arduino platform is its price point and its large array of available add-on modules and components. Since the platform is open source there is also a very large online community and a wide variety of sample code and projects for support. Daniel Langis argues in his paper “An

Implementation Strategy for Low-Cost Sensors" that Arduino is a very suitable platform for the development and rapid evolvement of new low-cost sensors. He writes:

> *Despite its low cost, the Arduino is a highly versatile platform which can be used to interface with almost any type of modern integrated circuit, such as Analog-to-Digital Converters, memory cards, real-time clocks, or power switching devices. As for oceanographic research, it is equally feasible to connect a full suite of oceanographic sensors to a single Arduino controller. If individual low-cost sensors for oceanographic measurements can be developed, the Arduino is an ideal platform for integrating multiple sensors into successful low cost instruments (Langis,2015, p.15).*

There are also disadvantages that should be addressed. For example, The Arduino is not optimized for power consumption. This is important in long-term deployment in remote locations, where battery capacity is a limiting factor. There are also limitations in the built in features of the Arduino board. Like the lack of storage such as an SD card, lack of a real-time clock and a low-quality ADC with a limited sampling rate and a resolution of only 10 bit. These flaws can be corrected for however, by using external modular components, like an OpenLog SD card logger, or an Adafruit 16-bit ADC.

3.2 Oceanography for everyone and OpenCTD

The OpenCTD project is a part of the Oceanography for Everyone community. They describe themselves as an opensource community of hardware developers, scientists and ocean stakeholders, that are making and sharing alternatives to expensive scientific equipment. The equipment is meant for citizen scientists interested in doing marine monitoring, fishermen exploring new tools to understand their catch or ocean enthusiast seeking new ways to interact with the sea" [18]

The equipment includes the Niskin3D, a 3D printable bottle that allows users to take discrete water samples at specific depths or in specific environmental conditions, designed to be integrated with the OpenROV which an open source project by a team of engineers in the San Francisco Bay area is. The BeagleBox, which is a single-board 3D printed robust field laptop designed to fit into a pelican case. And the OpenCTD which is a low-cost open-source instrument for measuring conductivity, temperature and depth creating a vertical water column profile.

The OpenCTD is a collection of sensors controlled by an Arduino microcontroller, contained in a watertight package made from a PVC pipe. It also contains a battery and a SD card for storing data. The battery is sufficient for 40 hour of use. It uses the DS18B20 Temperature Sensor, the MS5803-14BA pressure sensor and the Atlas scientific K=1.0 conductivity kit. The package is a regular PVC pipe cut into size and with a 3D printable lid to be fastened with glue. It has a GitHub page with construction instructions , and several field test logs, documenting its performance. OpenCTD was initially designed by Andrew Thaler, Kersey Sturdivant, and Russell Neches from Virginia Institute of Marine Science and Duke University's Nicholas School of the Environment.

3.3 Other similar projects

There are several other attempts at making low cost CTDs

3.3.1 Design of Sound Speed Profiler

Is a master thesis by Anwar Nazih Shaban , and is a previous attempt to tackle the same problems as this thesis. It is using a self-made conductivity sensor, the DS18B20 temperature sensor and the MS5803-14BA pressure sensor. The device is controlled by an Arduino UNO microcontroller, and can communicate using a Bluetooth chip called BlueSMiRF Silver. The data measured is logged on a memory card using Openlog[7].

3.3.2 Inexpensive Expendable Conductivity Temperature and Depth (CTD) Sensor

Is a paper from the Florida Institute of Technology attempting to make an expendable CTD at an affordable price. The project describes the construction of a CTD cartridge to be launched over the side of a vessel or from a drone that can do measurements over a 10 meter water column. The cost of each unit would be about 982 dollars and would consist of 5 primary electronic systems: a GPS chip, a memory card for data logging, wireless communication, and a Pic Chip microcontroller. The CTD is made with the atlas conductivity sensor, a wireless chip called the XBee that uses the ZigBee wireless protocol, and a data logging system called OpenLog. [19]

4 Sensors

A very important part of making this water profiler is choosing the right sensors. When choosing sensors there are several considerations to be made. This could be finding a suitable packaging or ensuring the sensor has adequate response time, and that the accuracy is sufficient. Usually the sensors will have a specified degree of uncertainty listed in its datasheet. But since the primary purpose of the water profiler will be to measure sound speed, which will be calculated using a complicated formula from three different sensors, the impact of the uncertainty of the different sensors is not always apparent. As a pre-purchase activity, several sensors were researched, and an excel sheet with the different sound speed formulas were made, experimenting with plugging in upper and lower values of a sensors listed measurement uncertainty, and seeing how they affected the formula output.

4.1 Sensors researched

	Atlas conductivity sensor	Gravity conductivity sensor	Adafruit Thermistor	DS18B20	MS5803-14BAPressure sensor
Listed uncertainty	+/- 2% psu in seawater	+/- 5% psu in seawater	+-1% of 10k ohm at 25℃ or ~+/- 0,25℃	+/- 0,5℃	+/-20 mbar
Reference value	35 psu	35 psu	15℃	15℃	4 meters (1.415 bar)
Uncertainty range in soundspeed using Mackenzie formula	1,97 m/s	4,96 m/s	0,97 m/s	1,93 m/s	~0 m/s
Uncertainty range in soundspeed using Medwins formula	1,98 m/s	4,96 m/s	0,997 m/s	1,99 m/s	~0 m/s

Some observations made:

-The pressure sensor barely affected the sound speed formula result in the 0-30 meter depth range.

-The difference in accuracy of the temperature sensors has a small impact on the sound speed calculation (+-0,05% of the final value). Although only two are listed here most others are in the same range.

-The choice of conductivity sensor seems to affect the final results slightly more (+/-0,166% of final value) and there is also a larger difference between the two options.

4.2 Temperature sensors

When choosing a temperature sensor there are several factors to take into consideration. These include temperature range, accuracy, response time, stability, linearity, and sensitivity. For this specific application one could also add having a package that supports high pressure underwater situations, and ease of use with Arduino. If this is to be used in an open source project an easy setup procedure and limited need for instructions would be a plus.

The most common types of temperature sensors are: RTDs, Thermocouples, Thermistors, Digital thermometer ICs, Analog thermometer ICs

4.2.1 RTDs

RTDs (resistive temperature detectors) consist of a coil of wire wrapped around a core of glass or ceramics. The wire is typically made of platinum or copper. As with the thermistor the RTD will change its resistance value with changes in temperature in a predictable manner. The RTDs can achieve high precision and have low drift over time. They can operate in a large range of temperatures, but have slower response time then for example the thermistor.

4.2.2 Thermocouple

A thermocouple sensor takes advantage of a phenomenon called the Seebeck effect. This is when a temperature difference between two different electrical conductors produce a voltage difference between the two elements. This is usually realized with two wires of different metals, joined in one end and separated at the other. The magnitude of the voltage difference between the wires then determine the temperature. Different combinations of metals are used for different applications. For example, Copper/Constantan is more accurate in lower temperatures then Iron/Constantan, which is a more common cheaper version. These are solid and cheap sensors that see a wide variety of use in the industry. They are however, often not the most accurate sensors, and they tend to drift over time.

4.2.3 Thermistor

A thermistor is named as such because it is a thermal resistor. It is made of a material typically a ceramic or polymer that changes its resistance dependent on the temperature. They typically have a smaller operating range then the other sensors, but are often very accurate and have a smaller response time. They can be either a NTC or a PTC type sensor, negative or positive temperature coefficient, which determines if the resistance increase or decrease as the temperature increases. A Thermistor response is not linear and need to be linearized in the intended temperature range before use.

4.3 Digital or analog thermometer ICs

There are a number of temperature sensor ICs with various working mechanisms and various features that could be used with Arduino.

4.3.1 DHT11 and DHT22

These sensors are very common in Arduino projects. They are small and cheap ICs that measure humidity and temperature. The temperature is measured with a thermistor. The chip contains some simple analog to digital conversion that yields a one-wire digital output. The interface is not dallas one-wire compatible, so each sensor needs its own pin on the microcontroller. There are some significant drawbacks with these chips. You can only sample the temperature once every second for the DHT11, or once every two seconds for the DHT22. The accuracy is also not particularly good, +-2 degrees for the DHT11 and ±0.5°C for the DHT22.

4.3.2 LM35DZ

The LM35DZ is a semiconductor temperature sensor. The basic working mechanism is to exploit the current and temperature characteristics of transistors. Two identical transistors are exposed to different currents and the difference in voltage outputs will be proportional to the temperature of the transistors. What is special about this IC is that it offers a pre-calibrated analog output voltage proportional to the temperature in degrees centigrade as opposed to absolute temperatures in kelvin. The accuracy of this sensor is typically ±0.5°C.

4.3.3 TMP36

The TMP36 is very similar and often interchangeable with the LM35DZ. It offers an analog output in degrees centigrade, but has slightly worse out of the box accuracy at ±1-2°C. The sensor is made to function on less than 50µA and therefore have very low self-heating.

4.3.4 DS18B20

The DS18B20 is a semiconductor temperature sensor with a digital output. It uses the Dallas 1-wire interface, which means that several sensors can operate on the same data line. This is useful if you want to have networks of sensors connected to the same microcontroller. The output data has a programmable resolution of 9-12 bits. The IC is powered either through a dedicated power line or through the data line (Parasitic Power Mode). In this mode the IC only requires two wires to function. This sensor also comes in a waterproof package, and is very often used in DIY projects that measure the temperature of liquids. In both the Sound Speed Profiler and the OpenCTD project a waterproofed DS18B20 has been used. The drawbacks are the stated ±0.5°C uncertainty in accuracy and the slow response time due to the large metal packaging.

4.4 Conductivity sensors

4.4.1 Atlas Scientific conductivity sensor

Atlas Scientific sells a set containing a conductivity probe, a circuit, and two calibration liquid solutions. The probe comes in three variants with different cell constants, K=0.1, K=1.0 and K=10. The cell constant, K, is the distance in cm between the probe’s electrodes divided by the surface area of the electrodes in cm^2. These have different accuracy ranges, so the K=0.1 is better at getting accurate reading in low conductivity conditions. This is because when the electrodes are placed closer together or made larger, the cell constant, K, will be less than one. This will raise the conductance and produce a value more easily interpreted by the instrument.

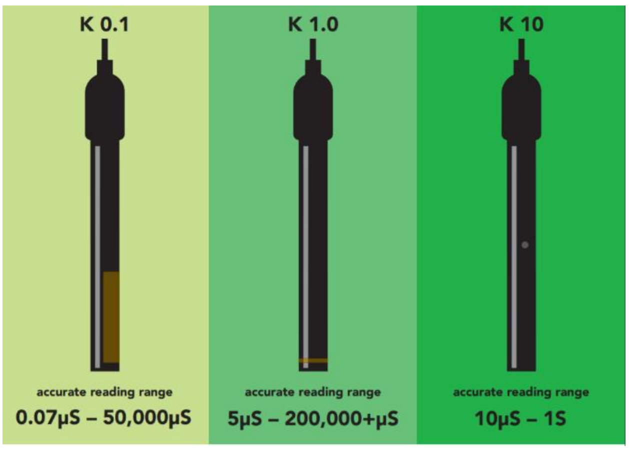


Figure 6 Different conductivity probe types [22]

Conductivity = Conductance x Probes cell constant (K)

OR Conductivity = Electrical Current/Voltage x Distance/Area

The circuit included with the probe can interpret the data and report the readings as total dissolved solids (ppm), practical salinity units(psu) or Siemens per centimeter (µS/cm. It also has temperature compensation feature, and can communicate with other devices through I2C or UART protocols [22]

4.4.2 Gravity conductivity sensor

The gravity conductivity sensor is from the DFRobot gravity series. The series is a collection of sensors and other electronics that are intended to be open-source and modular, so you can combine them for various DIY projects. The set contains a K=1 probe a signal conversion board, several calibration solutions and connectors. The listed accuracy of the set is +/- 5% psu in seawater, or about twice as much as the atlas sensor.

4.5 MS5803-14BA

The MS5803-14BA pressure sensor is a MEMS device, or a microelectromechanical system device, with moving parts between 1 and 100 micrometers in size. It claims an accuracy of 0,2 mbar or a depth resolution of 1 cm. The sensor can communicate with any microcontroller using the I2C or SPI interface. There is also an onboard temperature sensor, and different operating modes, optimizing for either current consumption or conversion speed. The sensor elements are surrounded by a antimagnetic steel cap and is covered in a gel membrane that protects it against 30 bar water pressure. The breakout version is an open source hardware product made by Sparkfun. They have broken out all the pins needed for

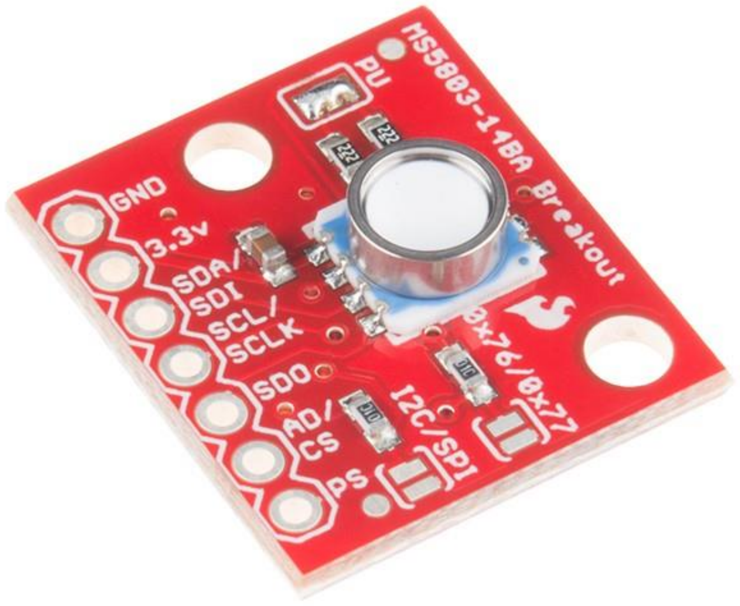


Figure 7 MS5803-14BA pressure sensor with protective gel coating [23]

Arduino, and included a GiThub and guide section with instructions, Arduino libraries and example code.[23]

4.6 DFRobot Turbidity Sensor

The DFRobot sensor is based on the Tyndall effect . which describes the scattering of light by suspended particles in a fluid. The higher number of particles the more light will be scattered. The sensor measures the amount of particles in the water by sending a light beam to an opposing light detector. The particle density is then a function of the light reflected into the detector.

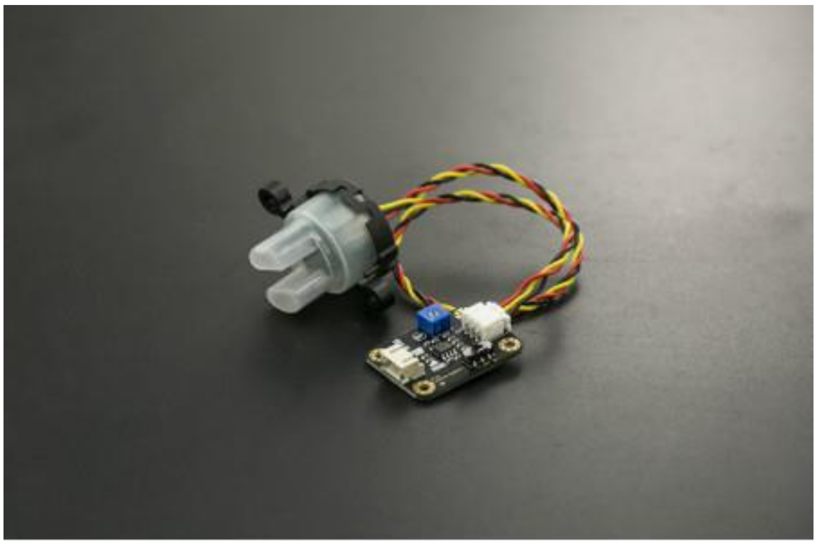


Figure 8 DFRrobot Turbidity sensor

The figure on the right is the circuit inside the probe. It has a photo transmitter diode (Photo TR) emitting light, and a receiver transistor registering the scatter. Ri and Ro are pull-down resistors. Pin 4 supplies 5v to the circuit and pin 1 is the signal readout.

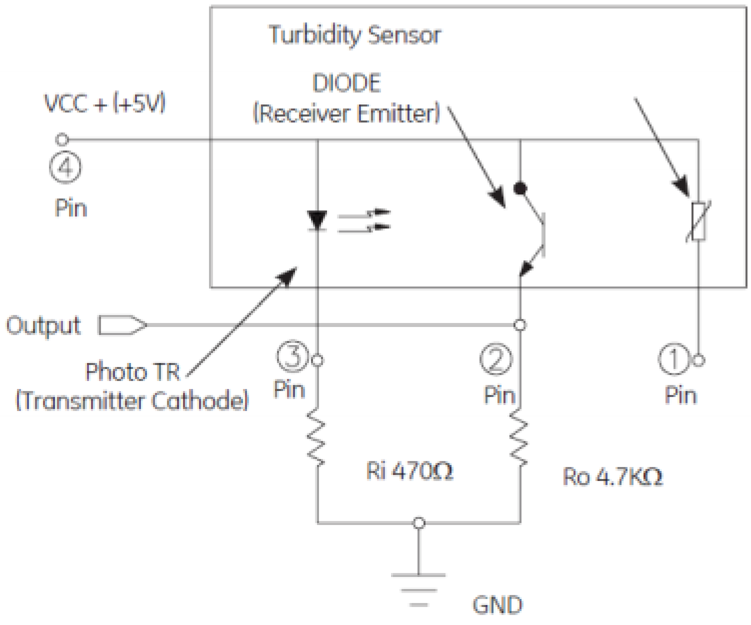


Figure 9 Turbidity probe schematic [21]

The figure below is the circuit schematic. It is a dual amplifier circuit which amplifies the signal from the probe since the Arduino data pins require 20mA to function. Toggling the SW1 switch puts the circuit in digital mode, which is a binary output that goes high when the signal reaches an adjustable threshold value. This value is set by an on-board potentiometer

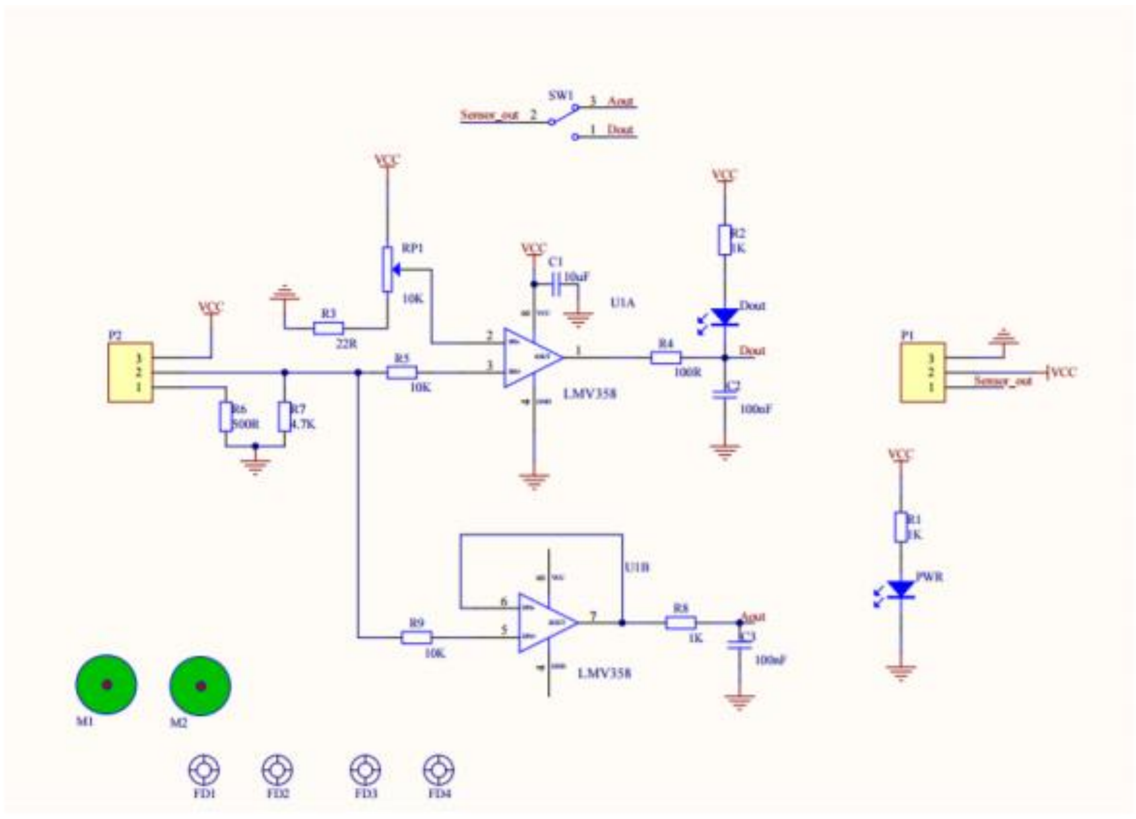
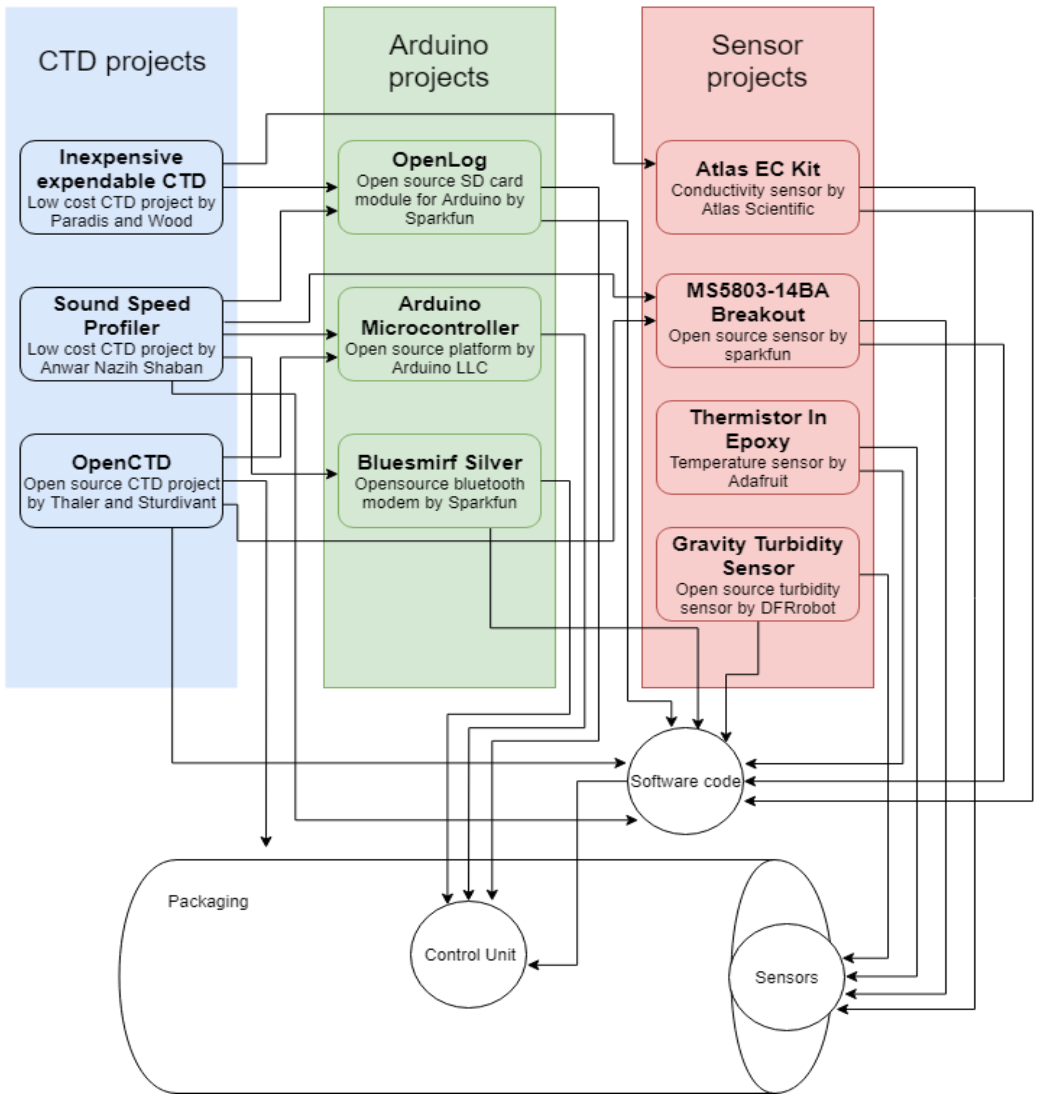


Figure 10 Turbidity circuit schematic

5 The Final Setup

Finally a setup was established based on the openCTD package solution. The figure below is an illustration of the components chosen and all the different projects that collaborated into making the finished CTD.



CTD projects
Arduino projects
Sensor projects
Inexpensive expendable CTD
Low cost CTD project by Paradis and Wood
Sound Speed Profiler
Low cost CTD project by Anwar Nazih Shaban
OpenCTD
Open source CTD project by Thaler and Sturdivant
OpenLog
Open source SD card module for Arduino by Sparkfun
Arduino Microcontroller
Open source platform by Arduino LLC
Bluesmirf Silver
Opensource bluetooth modem by Sparkfun
Atlas EC Kit
Conductivity sensor by Atlas Scientific
MS5803-14BA Breakout
Open source sensor by sparkfun
Thermistor In Epoxy
Temperature sensor by Adafruit
Gravity Turbidity Sensor
Open source turbidity sensor by DFRrobot
Software code
Packaging
Control Unit
Sensors

Figure 11 Illustration of the different projects contributing to the construction of the CTD

5.1 Microcontroller

Originally a Qduino mini microcontroller was used, which is a lightweight Arduino microcontroller. This is the controller used in the OpenCTD project, and is a very low power unit that can be sustained for long times on battery power. This unit malfunctioned, and a switch was made to the more common Arduino UNO. Finally since the project expanded further, the need for additional hardware UART connections necessitated an upgrade to the Arduino Mega. This also solved a long running problem of getting the OpenLog SD card module and Bluesmirf Bluetooth to function at the same time.

5.2 Bluesmirf Silver

Bluesmirf Silver is a Bluetooth modem for Arduino by Sparkfun. It is a Breakout version of the RN-41 module. Bluetooth is a wireless technology standard that uses low power radio waves to transmit data on a frequency of around 2.45 GHz. This is in the ISM band of frequencies set aside for the use of industrial, scientific and medical devices. Bluetooth devices avoid interfering with other systems by transmitting very weak signals of about 1 milliwatt. Phones for comparison can transmit signals up to 3 watts in strength. This limits the range to about 10 meters, but at the same time limits interference from other systems. Another way Bluetooth avoids interference is by a method called spread-spectrum frequency hopping. This means a device will use 79 different random frequencies within a designated range, hopping from one to another. A Bluetooth device will change frequencies about 1600 times a second. This ensures that if interference occurs it only does so in a tiny fraction of a second. Bluetooth uses a “piconet” topology, where it forms a small network of 1 master and up to 7 slaves. The master can transmit data to one other device in the network at any given time, and will typically switch from slaves in round-robin fashion, while the slaves are in listen mode. In the CTD probe the Bluesmirf chip is placed in a slot through the 3d printed lid and covered in epoxy with only the antenna protruding to the outside of the CTD.

5.3 OpenLog

The Arduino Uno does not have a lot of onboard memory. It has three different pools of memory: the Flash memory where the Arduino sketch is stored. This is about 32k bytes, where 0.5k is used for the bootloader. The SRAM which is where the sketch creates and manipulates variables when its running. This is 2k bytes of volatile memory which means the stored data will be lost when power is turned off. lastly there is 1k bytes of EEPROM which is a memory space that programmers can use to store long term information. Naturally some external storage is needed. There are several options available, most of these involves either an extra SD card module, or a connection with a computer either through USB or wireless. Since this will be deployed underwater, connection with a computer will be difficult.

For this project a serial data logger called OpenLog was inherited from a previous project. OpenLog is an opensource data logger chip that connects to the microcontroller over a serial connection. It holds a 16GB SD card for storage.

5.4 Improvements over past projects

-This version of the CTD has a working quality EC sensor, which is a basic requirement of any CTD. The Sound Speed Profiler never achieved a sufficiently accurate EC measurement.

-Both the OpenCTD and the Sound Speed Profiler used the DS18B20 sensor which is a digital IC inside a large steel casing. This larger casing makes the sensor respond very slowly. Moving from one water bath to another, from about 40°C to 20°C, it required 25 seconds to stabilize at the new temperature. In the field, this would require really slow casts to acquire good readings. The thermistors in epoxy requires only about 5 seconds to completely stabilize in the same setup.

-The openCTD has no wireless communication

-A Bluetooth setup was present in the Sound Speed Profiler, but the implementation was flawed. The Bluetooth chip was placed inside the package and would only transmit the data real time. This would not work in the field since Bluetooth does not carry through water. a new implementation was made, where the antenna of the sensor was placed through a slit in the 3d printed lid, so it can transmit when at the surface. Code was then added to be able to send commands through the Bluetooth module to the Arduino. The sensor can now receive a start command through Bluetooth, start logging, store the measurements on the Openlog module, and resurface. When surfaced communication can be restored and a command can be sent to the Arduino to send the the data stored in the SD card over Bluetooth. This allows continuous use without opening the package and manually retrieving the SD card, which is operating procedure in both the OpenCTD and the Sound Speed Profiler. The ability to send a start command at an appropriate time, also prevents the storage of large amounts of trash data, since it won’t need to run continuously. This was an issue reported in the use of the OpenCTD where continuous logging led to large chaotic data logs.

-The Arduino UNO was upgraded to an Arduino Mega for the multiple hardware UART support. This was to correct an issue from the Sound Speed Profiler, where running several UART connections on the UNO, which has only one UART port and had to emulate others using the software serial library, led to crashes and unpredictable behavior. This upgrade increases stability in the simultaneous use of the Bluetooth module and the OpenLog module, and allows for the easy expansion of Atlas PH and dissolved oxygen sensors in future improvements. These are two UART sensors that was considered included, but ultimately cut due to costs.

- The inclusion of a DFRrobot Turbidity sensor.

- The inclusion of a Kalman filter, to attempt to improve thermistor readings through sensor fusion and a prediction model of the hysteresis effect

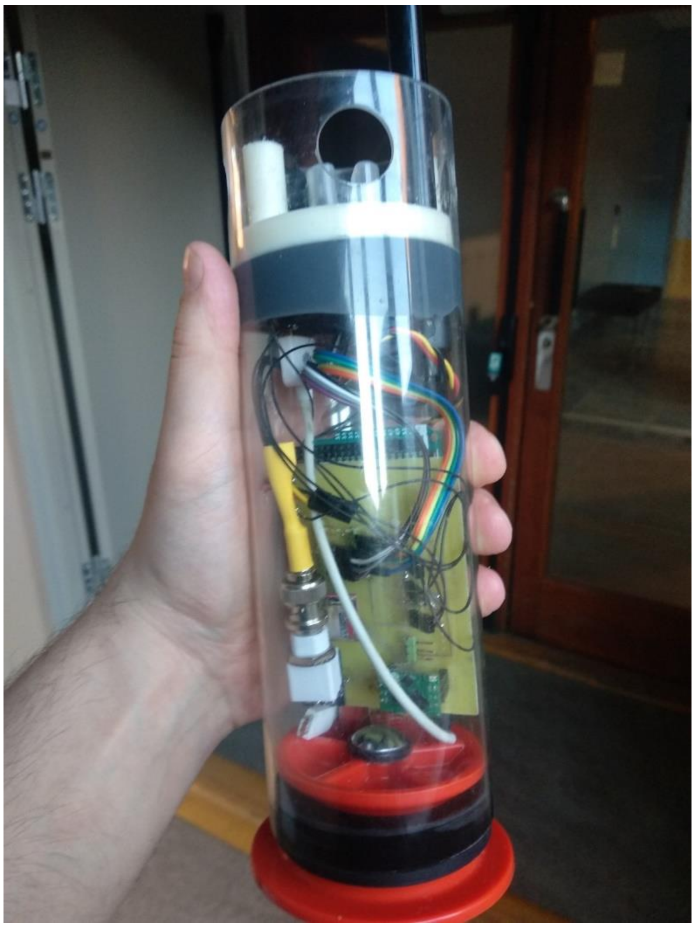


Figure 12 The finished probe

5.5 Cost of the project

The budget for this project was limited by the budget allotted to each master program at the University of Oslo, so cost was a significant issue. The listed price of the CTD ended up at about 384 $, though several components were inherited from a previous project. The low price of many of the components are from Chinese web retailers, where electronics can be bought very reasonably. The Conductivity sensor is by far the costliest sensor in the CTD, but it was made a priority. This was because of reading about other projects problems with creating a working conductivity sensor, the importance of an accurate reading for a reliable sound speed profile, and its proven application in similar projects. There are several other sensors of interest that were cut due to costs. Notably two Atlas Scientific dissolved oxygen and PH sensors. These might be included to the next iteration of the CTD which is in the works in another master thesis. The price of 3d printed parts is not included in this list, and neither is the PCB board and components. These costs are negligible, but does require some expensive equipment, like a 3d printer and access to an electronics lab.

Parts	Price $
Arduino Mega	30
Pressure sensor MS5803_14ba	59.95
Openlog (SD card reader)	14,95
Bluesmirf silver (Bluetooth)	24,95
DFRobot Turbidity sensor	9,90
Atlas EC kit	215
3 inch acrylic glass tube	5
Waterproof USB port	3
Micro USB to USB cable	2.99
Type A to type B USB cable	1.99
4 thermistors in epoxy	3.99
Mobile power bank	12

6 Calibrating and preparing the sensors

6.1 Atlas conductivity sensor

The default mode of the circuit is UART mode. I2C communication is possible, but the protocol is considerably more complex, and somewhat changes the behavior of the chip. For this project the UART port is already in use from both the Openlog chip and the Bluetooth chip, in addition to the USB port for uploading code. So it is necessary to switch to I2C. Instead of the UART "continuous mode" that gives measurements on a regular interval, you have to send the "R" command for a single reading every time you want to take a measurement. After the command the circuit needs 600ms processing time before it replies.

A conductivity measurement is also significantly influenced by the temperature. Conductivity will increase about 2% per °C. For the values taken at different temperatures to be comparable a compensation has to be made to show what the value would be at a reference temperature (usually 25°C). The physical causes for this dependency lies in the relationship between electrical conductivity, diffusion coefficients and the viscosity of water. There are several formulas describing this relationship. For example $EC_{25} = \frac{EC}{0.889*} 10^{A/B}$

Where the parameters A and B are:

$A = 1.37023\,(t - 20) + 8.36 \cdot 10^{-4}\,(t - 20)^2$

$B = 109 + t$

EC_{25} is the conductivity at 25°C, EC is the measured conductivity and t is the temperature in °C. A more common linear approximation is

$$\frac{EC}{EC_{25}} = 1 + a(t - 25)$$

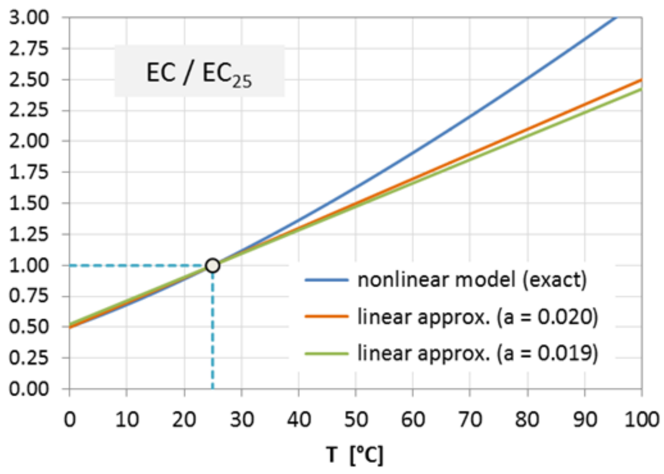


Figure 13 Difference between linear and non-linear temperature compensation models [32]

Where a is temperature compensation factor. Several values of a are recommended in different literature. Groundwater textbooks often assume a 2% increase of EC per 1◦C, which leads to a factor of 0.02, while geophysicists commonly use a factor of 0.025 [31]. The chart on the right shows the difference between the non-linear model, and the linear model with two different compensation factors. As is apparent, the linear model is fairly accurate in the range of 0 to 30°C which is the usual operating range of most CTDs.

These calculations can be done internally within the Atlas sensor circuit if you supply an external temperature measurement, or they can be done in post processing using the raw conductivity and temperature measurements.

6.1.1 Calibrating the sensor

Calibrating the sensor is done by uploading a calibration sketch to the Arduino and sending commands through the serial monitor interface in the Arduino IDE. The instructions and commands are detailed in the Atlas EC kit datasheet. Either a two point or three point calibration is possible, with one point being a dry measurement or zero point. Calibration fluids are included in the EC kit, one at 12880µS and another at 80000 µS. The operating range of the Arduino CTD will be around 0 to 40000 µS. Extra care must be taken to ensure the temperature of the fluid is exactly 25°C, or the values on the bottle will have to be recalculated.

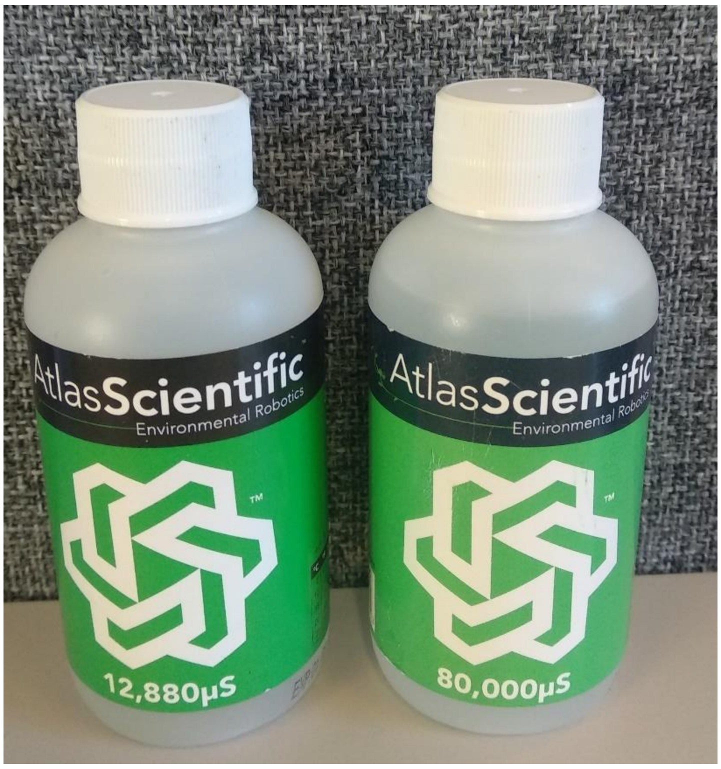


Figure 14 EC calibration fluids

6.2 DFRobot Turbidity Sensor

This sensor has an analog output voltage that can be interpreted by the Arduino ADC. But exactly how this relates to the unit of measurement, the NTU, is not entirely clear. The sensors wiki page gives some clues however. The graph on top shows an appropriation of the relationship between the voltage and the NTUs. The one on the bottom shows that there is also a temperature dependence. Fluctuations in water temperature can affect the electronic components, causing a measurement error. There also seems to be some variation between sensors. Using the formula in the graph on the sensor used in this project gave a large value of NTUs in clear water, where this should be zero. The proper way to calibrate a turbidity sensor would be to obtain calibrating fluids with known NTU values, note the probe voltage outputs at each calibrating point, and do a curve fit approximation to cover the whole voltage range. This has not been attempted in this project, since calibrating fluids are very expensive, and the sensitivity of the sensor are no where near what is required for use in a CTD.

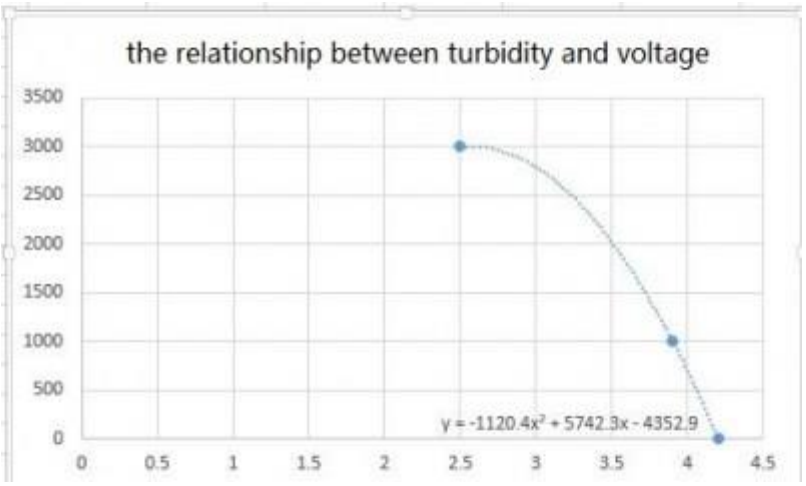


Figure 15 The relationship between turbidity and voltage [24]

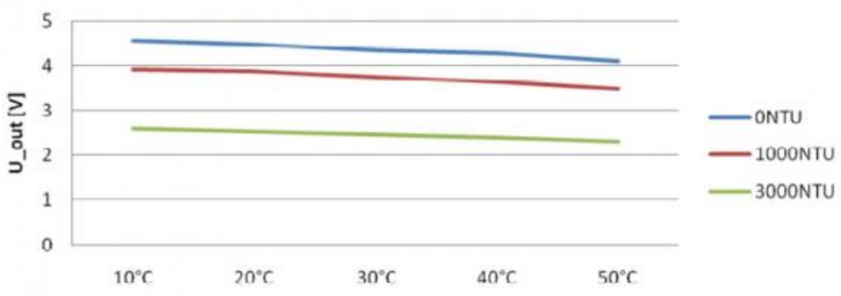


Figure 16 Output voltag at different temperatures [24]

6.3 Thermistors

The thermistor is a temperature sensitive resistor. The Adafruit Thermistor in Epoxy is a 10000 ohm resistor with +-1% ohm listed uncertainty. This means that the thermistor is calibrated to be 10 000 ohms +- 100 ohms at 25 ˚C. Since the microcontroller cannot read resistance directly the thermistor is connected in series with a 10k resistor, making a voltage divider circuit. When measuring the output voltage between the resistors, the voltage will change as the thermistor resistance changes. The voltage divider equation is:

$$V_{out} = V_{in} \times \left(\frac{R2}{R1+R2}\right)$$

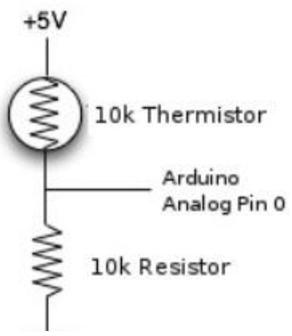
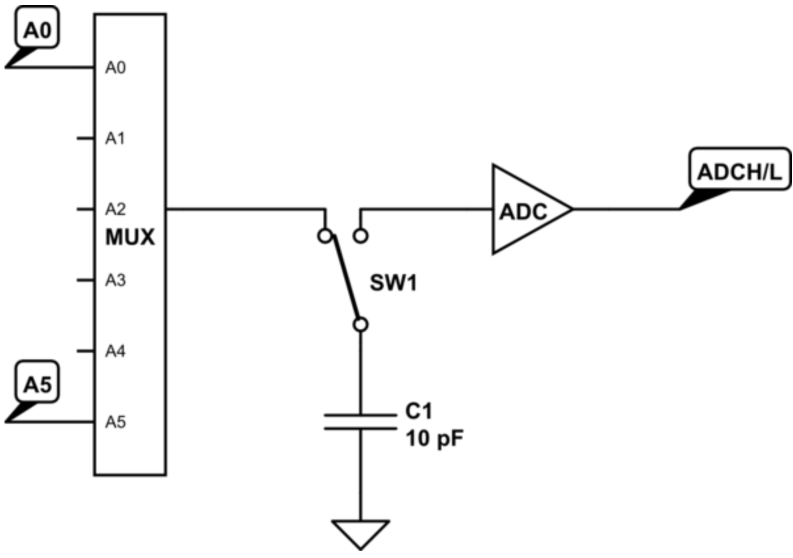


Figure 17 Thermistor in a voltage divider connected to an arduino

Where:

V_{out} : *Voltage between thermistor and known resistor*
V_{in} : V_{cc}, *i.e.* 5V
$R1$: *Known resistor value*
$R2$: *Resistance of thermistor*

On an Arduino Mega there are several analogue input pins. These pins have an Analog to Digital Converter (ADC), that converts an analog voltage to a digital value. The ADC on Arduino Mega has a 10-bit resolution which means it can detect 2^10 or 1024 discrete analog levels. They working mechanism of an ADC is to charge up an internal capacitor and measure the time it takes to discharge across an internal resistor. The microcontroller then measures the number of clock cycles before the capacitor is discharged. So typically, the Arduino ADC will receive a voltage between 0 and 5v and translate it to a similarly scaled number between 0 and 1023.



All the analog ports share the same ADC. Every time a conversion is made, the MUX selects which pin to read (A0-A5) and charges up the sample and hold capacitor C1. The Switch (SW1) then switches C1 over to the ADC where it discharges over an internal resistor.

During the testing of the CTD, the voltages on the pins where interfering with each other, IE heating up one thermistor would influence the other. This is because the high impedance of the thermistor circuits leaves a low amount of current flowing into the ADC and switching between thermistors too quickly doesn’t leave the sample and hold capacitor enough time fully discharge. A fix for this issue is to add a short delay in code between readings.

6.4 Temperature Sensor Response Time

One of the major issues of both the Sound Speed Profiler and the OpenCTD is the response time of the temperature sensor. In one field test where the OpenCTD was tested alongside a SeaBird Model 911 plus CTD onboard the research ship Blue Heron, Andrew Thaler makes a note afterwards that:

"Casts need to be slow. On the order of 0.15 m/s slow. The temperature probes are not fast." [20]

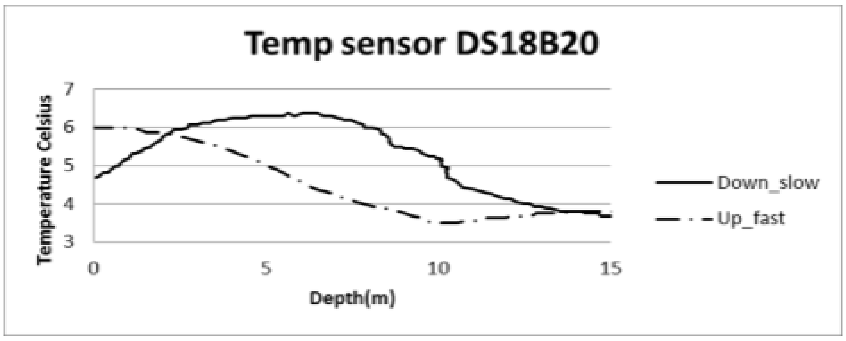


Figure 18 Sound Speed Profiler temperature hysteresis effect [7]

Likewise in Anwar Nazih Shabans [7] Sound Speed Profiler project a test was done raising and lowering the temperature sensor, where the sensor would record a very different temperature on the way down than at the very same depth on the way up. This due to hysteresis effects, or slow sensor response. The cast speed in Shabans test is also fairly slow, at about 0,12m/sec on the way down and 0,18m/sec on the way up.

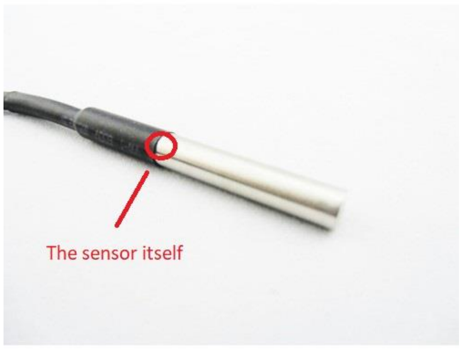


Figure 19 Sensor element inside casing of DS18B20

It is unclear why the DS18B20 waterproofed sensor has been chosen in these projects, but it is probably because it is an easily setup device in an already waterproofed steel casing package. This packaging is also why it is so slow. Peaksensors a manufacturer of temperature sensors notes on their website that:

Sensor speed is dominated by the thermal mass of a sensor. Robustness is the counter to speed of response. This balance requires a compromised decision that has to made in regards to which criteria is more important to the end user. This is because it is difficult to create a robust probe with a fast response time – the sensor sheathing and other protection elements that are required required to improve robustness has a significant effect on response time [28]

So for a project like this, where the fairly quick changes in temperature needs to be recorded, the compromise probably should lean towards a smaller less robust sensor. One such candidate is the cheap thermistors by Adafruit covered in a thin layer of epoxy. These are very small and should perform much faster, but are also fairly fragile. So extra thought would need to go into their placement and packaging.

To quantify the difference between the responses of the DS18B20 and the thermistor in epoxy, two water baths, one with a temperature of about 20 degrees and one with about 40 degrees were placed on a table. One thermistor and one waterproofed DS18B20 were placed in one of the baths and then rapidly moved over to the other, while the microcontroller logged a temperature sample every second.

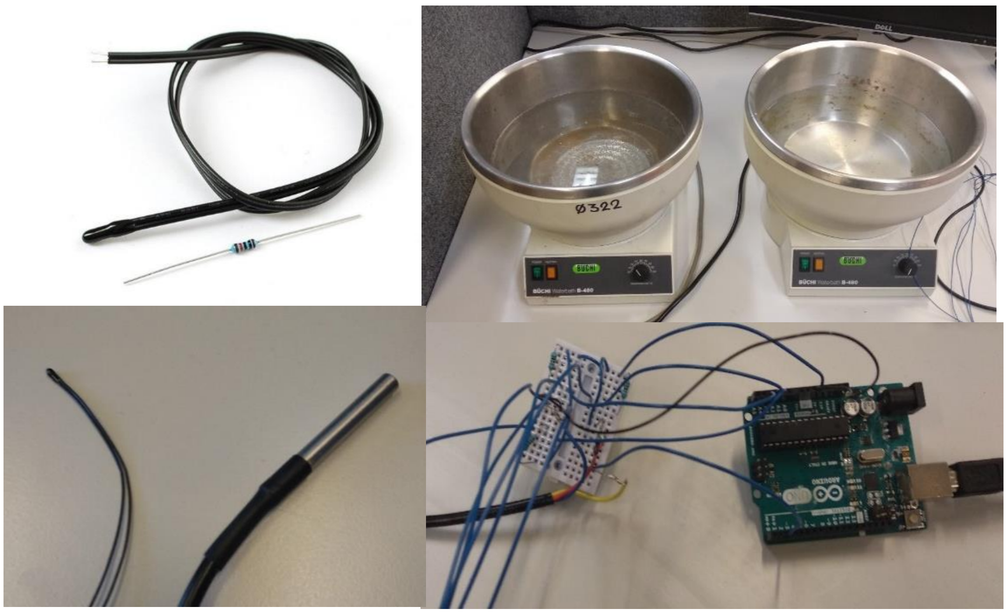


Figure 20 Temperature response test setup

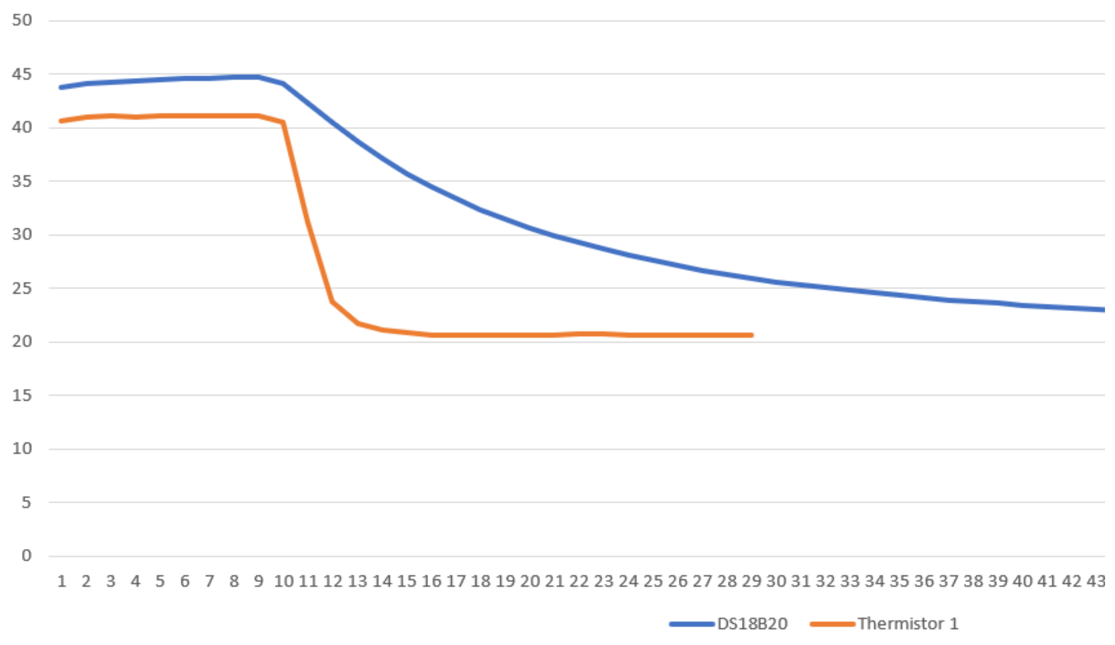


Figure 21 Temperature response time comparison

The graph shows the large difference in sensor response time. While the reading from the steel cased sensor used 27 second to come within two degrees of the final value and 45 seconds to reach a stable temperature, the thermistor used about 3 and 5 seconds to do the same. Note that they differ in temperature values, most likely because the factory calibration of the thermistor was off. The somewhat accurate out of the box calibration and easy digital readout of the DS18b20 is perhaps a reason why it is often chosen for projects of this nature.

6.5 Thermistor calibration

The different thermistor resistance values will correspond to temperature values in a very predictable but non-linear way. Usually the thermistor manufacturer will supply accurate tables that show the resistance value for each specific temperature. You can also use the Steinhart-Hart equation which is a mathematical approximation of the same relationship. The error in the Steinhart–Hart equation is generally less than 0.02 °C in the measurement of temperature over a 200 °C range The equation is:

$$\frac{1}{T} = A + B\ln(R) + C[\ln(R)]^3$$

Where T is the temperature, R is the resistance and A, B and C are the Steinhart-Hart coefficients. The coefficients are found by taking precise measurements at specific temperatures, say 20 ˚C, 25 ˚C and 30 ˚C, and solving three simultaneous equations. A further simplification is to use the B parameter equation:

$$\frac{1}{T} = \frac{1}{T_0} + \frac{1}{B}\ln\left(\frac{R}{R_0}\right)$$

This is the Steinhart-hart equation where

$$A = \frac{1}{T_0} - \frac{1}{B}\ln(R_0)\,, B = \frac{1}{B} \textit{ and } C = 0$$

T0 and R0 is the temperature and resistance at the reference temperature, usually 25 ˚C. This version of the formula only requires the input of one term, the B-parameter often supplied by the manufacturer, and can be considered accurate over smaller temperature ranges.

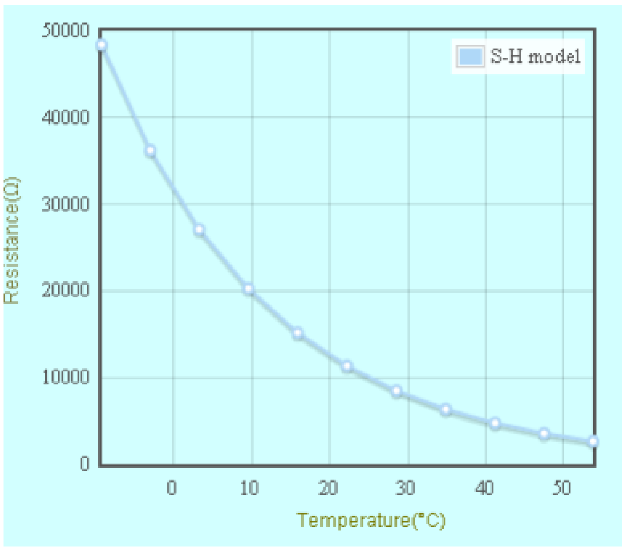


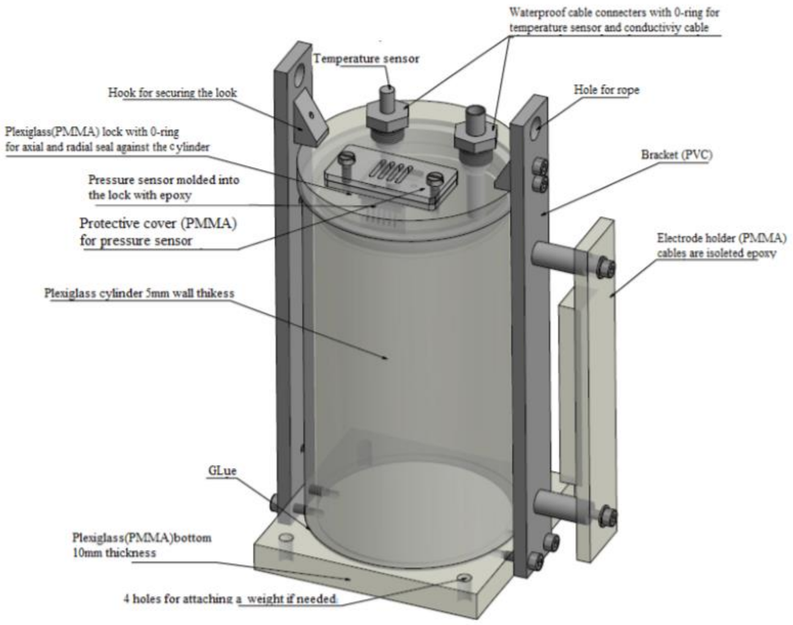
Figure 22 Typical Steinhart-Hart thermistor curve [25]

In the first stages of testing the CTD the B-parameter supplied from Adafruit were used in the Arduino code to convert the ADC readings to a temperature value. This calibration turned out to be accurate in temperatures near 25 ˚C, but unreliable in lower temperatures closer to 0 ˚C. The CTD needs to be accurate in this range as well, since the normal operating range will be around 0-25 ˚C. Instead a 3-point calibration was performed. This requires three resistance readouts at three known temperature values. The first attempt was using a container of melting ice for an accurate 0 ˚C point , assuming an accurate factory calibration at 10000 ohm for a 25 ˚C point, and using a DS18b20 for an approximate 12 ˚C point. Later a calibration was done by doing a cast alongside a commercial CTD, logging the resistance values and calculating new coefficients with the corresponding commercial CTD temperature values. The coefficients can be found by solving the the three simoultaneous equations:

$$\begin{cases} A + (\ln R_1)\,B + (\ln R_1)^3\,C = \frac{1}{T_1} \\ A + (\ln R_2)\,B + (\ln R_2)^3\,C = \frac{1}{T_2} \\ A + (\ln R_3)\,B + (\ln R_3)^3\,C = \frac{1}{T_3} \end{cases}$$

7 The Package

The packaging containing the sensors and electronics needs to be solid and watertight. In the Sound Speed Profiler project a custom made casing was made from a tube of plexiglass, with a lid and supporting beams on the side that functioned as a closing mechanism. There were some issues with waterproofing in this design, and also a vulnerability too pressure. When the inside of the tube is filled with air, the package will collapse when going deep enough.


Waterproof cable connecters with 0-ring for temperature sensor and conductiviy cable
Temperature sensor
Hook for securing the look
Hole for rope
Plexiglass(PMMA) lock with 0-ring for axial and radial seal against the cylinder
Bracket (PVC)
Pressure sensor molded into the lock with epoxy
Protective cover (PMMA) for pressure sensor
Electrode holder (PMMA) cables are isoleted epoxy
Plexiglass cylinder 5mm wall thikess
GLue
Plexiglass(PMMA)bottom 10mm thickness
4 holes for attaching a weight if needed
Figure 23 Sound Speed Profiler packaging [7]

The OpenCTD project uses a simpler package made out of a PVC pipe and a 3d printable lid and sensor outlets on one end, and a gripper plug on the other. The plug allows for opening and closing the package while still remaining waterproof. A gripper plug is a plug used by plumbers to temporarily close an open tube. It's a rubber plug that expands when you turn a wing nut bolt which can withstand a fair amount of pressure when tightened

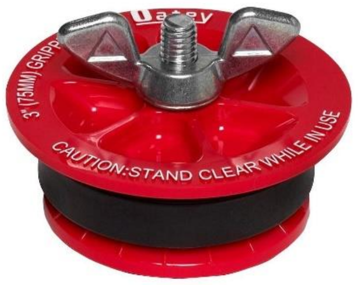


Figure 24 Gripper Plug

The package is also tested under various amount of pressure, so it is known to give in at about 25 meter depth. [20] If it is to be used even deeper the package needs to be filled with mineral oil, or other non-conductive fluid.

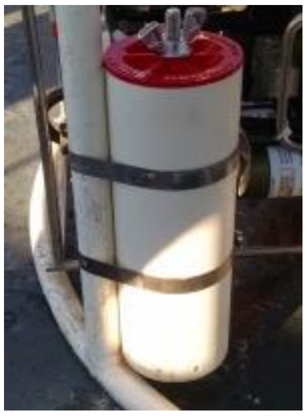
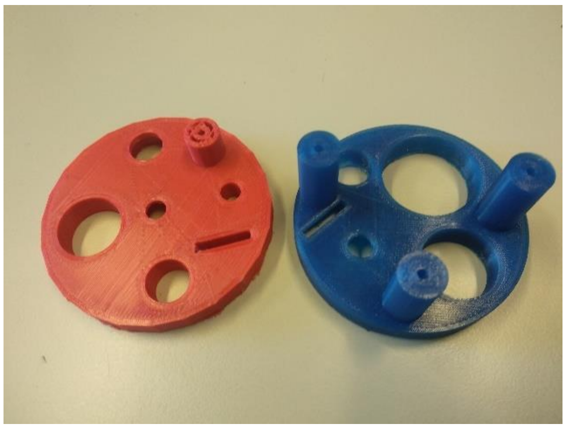


Figure 25 OpenCTD package [20]

A modified version of The OpenCTD package was used in this project. Instead of a PVC pipe a transparent plexiglass tubing was used instead. A lot of work also went into 3d modeling of the sensor outlet lid. The changes involve making protruding cylinders to hold the very small thermistors. These cylinders have a tiny hole in them to let the wires through and are filled with epoxy. The reason for placing them a distance from the lid is to avoid having them measure water trapped in the pocket created by the lid and the protective sleeve of the package, and also to avoid any temperature interference form the package and the other sensors. Bolt holes were also made to fasten the pressure sensor to the lid. In the OpenCTD design the sensor is only fastened by glue, and was vulnerable to be pushed in from outside pressure Several iterations were made to correct flaws and make the fit right. The first drafts were made using a Ultimaker 2+ 3d printer using PLA filament. PLA filament is one on the most common filaments for desktop printer use. These versions had "stringing" errors which is excess plastic strings in corners and hollow areas. They were also very porous and not suited for underwater use. Later a Dimension Elite printer was used using ABS+ filament. ABS+ is another common filament that is more resistant to high temperature and is less brittle then PLA. This yielded a better lid, but still not one that was fully watertight. The solution was to prepare the lid with acetone vapor. Acetone dissolves the ABS+ filament and can be used to smoothen and harden the outer layer of a 3D print. This will also deform the print slightly and one has to be careful not to overdo the application. It is possible to dip the print in liquid acetone, but using the vapors is a slower and more predictable approach.



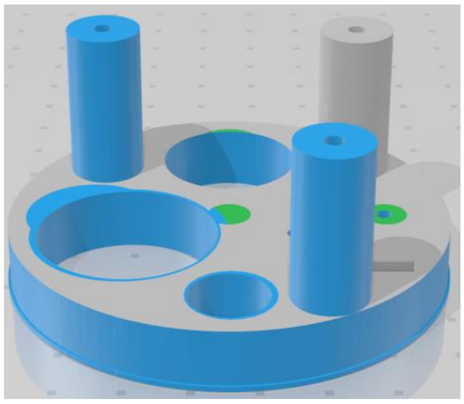


Figure 26 Several versions of the 3D printed lid

A setup was made where the plastic lid was placed at the bottom of a container with paper dipped in acetone fastened to the roof of the container with magnets. This because the vapor is heavier than air and will gather near the container floor. A protective layer of glass is placed between the paper and the print to avoid acid dripping.

The sensors were then carefully placed inside their sockets and fastened by a two-component epoxy. The epoxy used is a Scotch-Weld 2216 B/A Epoxy, which is a very viscous adhesive that is suitable for bulky application like filling gaps or potting containers. The sensor heads and components were covered in tape to avoid adhesive splatter. The lid was then potted into the plexiglass cylinder. This is when the lid is placed at the far end of the cylinder, and the cylinder is filled from the inside with a layer of adhesive. This application is done with a syringe.

Before potting the lid several holes were drilled in the sensor area of the cylinder. This is to ensure circulation of water inside the protective sleeve covering the sensors. When drilling a piece of wood was carved to support the cylinder from the inside and prevent the cylinder from cracking.

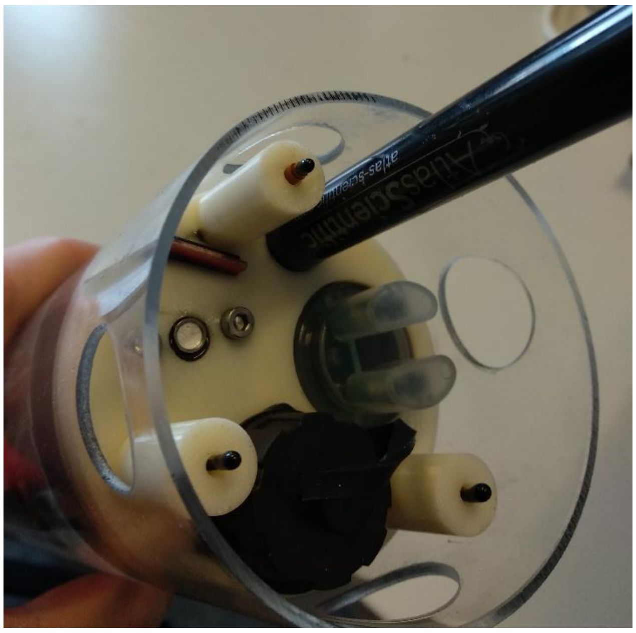


Figure 27 Finished 3D printed lid with sensors protruding

7.1 PCB design

After settling for a final wiring configuration, a PCB was designed to make the setup more compact. The PCB is a two-layer FR4 produced by Elab at UIO. It has pin rows to fit into the socket of the Arduino mega, as well as sockets for the Openlog module and atlas EC circuit modules, and pin rows to connect to the various wires coming from the sensor lid. All other wiring, resistors and capacitors are integrated into the PCB.   The design was made in several iterations in CADSTAR. Some issues encountered was for example dealing with the pin spacing of Arduino. On most Arduinos there is an irregular spacing between pins 7 and 8 that needs to be accounted for. This is due to an oversight in the original design of Arduino, that has been kept to retain compatibility with older modules and expansions. The tight spacing of the cylinder made the connector placement challenging and several reworks had to be done. Special USB B cables with shorter connector heads was bought and cut to be able to fit in the cylinder. The coax cable from the EC probe also had to be shortened and fitted with a new connector.

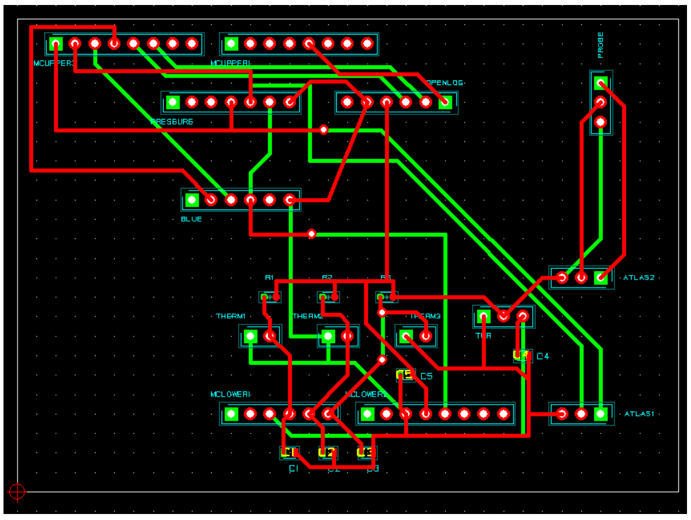
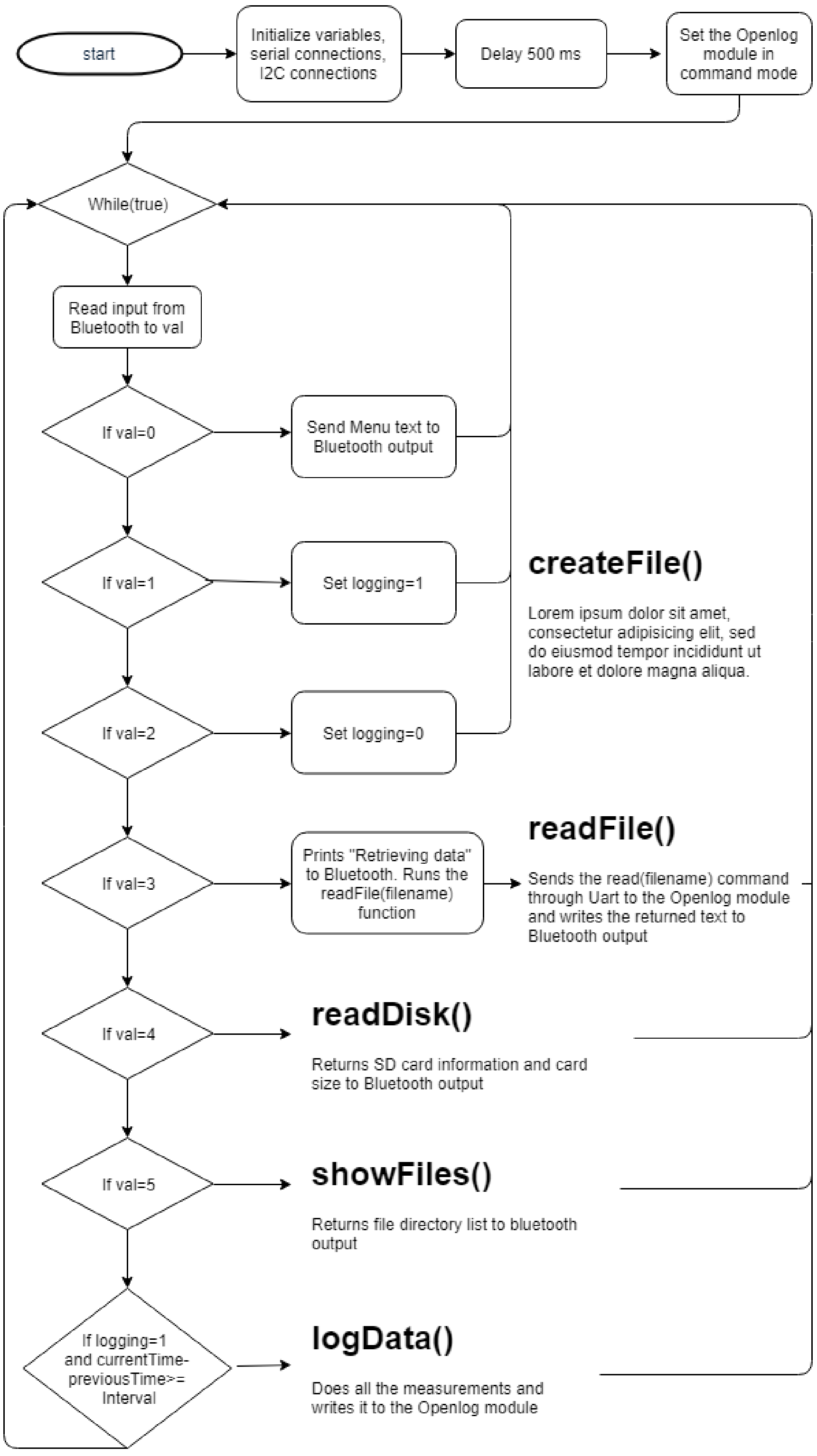


Figure 28 PCB routing Layout

8 Arduino program

The figure on the right is a flow chart of the main part of the Arduino program. It illustrates a branched set of loops that receives a command from a Bluetooth interface, like the Serial Bluetooth Terminal app for Android or Putty for Windows and executes appropriate actions. Sending a “0” over the Bluetooth terminal will return a text menu with available commands.



start
Initialize variables, serial connections, I2C connections
Delay 500 ms
Set the Openlog module in command mode
While(true)
Read input from Bluetooth to val
If val=0
Send Menu text to Bluetooth output
If val=1
Set logging=1
createFile()
Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
If val=2
Set logging=0
If val=3
Prints "Retrieving data" to Bluetooth. Runs the readFile(filename) function
readFile()
Sends the read(filename) command through Uart to the Openlog module and writes the returned text to Bluetooth output
If val=4
readDisk()
Returns SD card information and card size to Bluetooth output
If val=5
showFiles()
Returns file directory list to bluetooth output
If logging=1 and currentTime-previousTime>= Interval
logData()
Does all the measurements and writes it to the Openlog module

Figure 29 Arduino sketch flowchart

First all necessary variables and connections are initialized. Vectors and matrices used by the Kalman filter are initialized as arrays or two-dimensional arrays. There is a delay after the initialization of the Openlog module to ensure it is ready to receive commands. Several functions, like the readFile, readDisk, and showFiles functions assumes the Openlog module is in command mode. The interval used between measurements is 1000ms. This is mostly due to the limitations of some of the sensors. In I2c mode the Atlas EC circuit requires 6000ms to process a measurement command, and another 300ms to do a temperature compensation. The temperature compensation feature was present in earlier drafts of the code but removed later due to reports of faulty behavior in other projects. This may have been fixed in newer firmware releases from Atlas scientific. Currently only the pure conductivity measurement is reported, and the temperature compensation must be done in post processing. The delay between samples is handled with the native millis() function. As opposed to the delay() function this allows other tasks to be executed in the time between intervals.

The program uses several slightly modified functions from example code provided by the GitHub repositories and guide sections attached to the open source hardware projects. They provide simplified methods of communicating and handling data from their circuits. The two functions below are taken from Sparkfuns Openlog GIThub. They send a command to the OpenLog circuit over UART and handles the response appropriately. They are modified to return the response to Bluetooth output instead of the primary serial output.

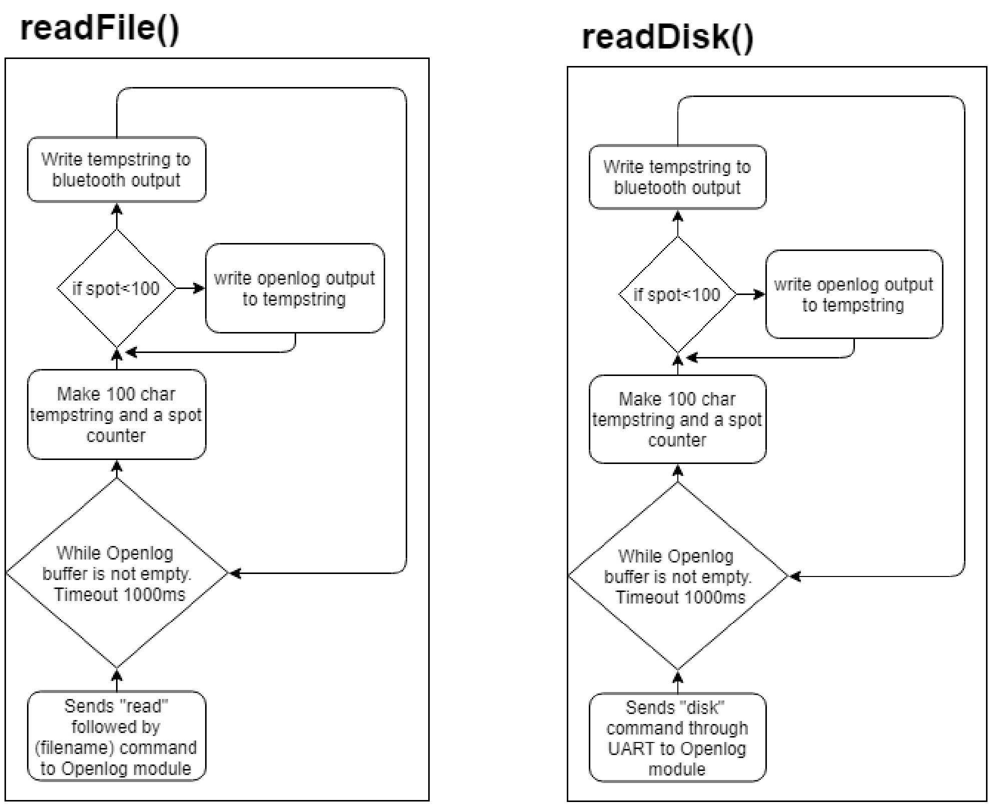
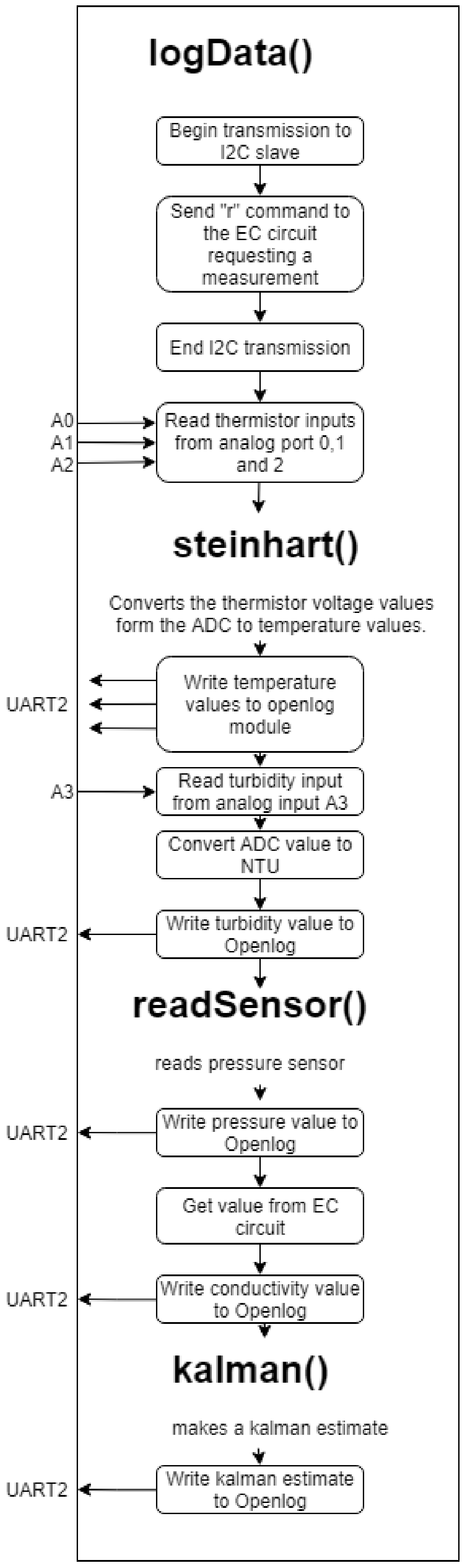

readFile()
Write tempstring to bluetooth output
if spot<100
write openlog output to tempstring
Make 100 char tempstring and a spot counter
While Openlog buffer is not empty. Timeout 1000ms
Sends "read" followed by (filename) command to Openlog module
readDisk()
Write tempstring to bluetooth output
if spot<100
write openlog output to tempstring
Make 100 char tempstring and a spot counter
While Openlog buffer is not empty. Timeout 1000ms
Sends "disk" command through UART to Openlog module

Figure 30 readFile and readDisk function Flowcharts

The logData function runs once for every measurement. It handles the I2c connection with the EC circuit and the pressure sensor. For the EC circuit all the commands are handled explicitly while the pressure sensor uses a simplified function given in an Arduino library available on the MS5803-14BA GIThub page. There are short delays added in between the analog readouts to ensure the ADC has enough time to do a proper sample. This is covered in the thermistor calibration section



logData()
Begin transmission to I2C slave
Send "r" command to the EC circuit requesting a measurement
End I2C transmission
A0
A1
A2
Read thermistor inputs from analog port 0,1 and 2
steinhart()
Converts the thermistor voltage values form the ADC to temperature values.
UART2
Write temperature values to openlog module
A3
Read turbidity input from analog input A3
Convert ADC value to NTU
UART2
Write turbidity value to Openlog
readSensor()
reads pressure sensor
UART2
Write pressure value to Openlog
Get value from EC circuit
UART2
Write conductivity value to Openlog
kalman()
makes a kalman estimate
UART2
Write kalman estimate to Openlog

Figure 31 logData function flowchart

9 Kalman filter

The Kalman filter was an experiment implemented to deal with the unknown performance of the thermistors. Several factors were of concern: their smaller size and apparent fragility, the addition of analogue circuitry and wiring, and the complications of packaging and insulating in epoxy. Because the thermistors were cheap, two extra sensors were included in the package. This adds redundancy in case of failures and allows for averaging if the sensors show individual inconsistencies. Also since hysteresis of the temperature sensor were one of the major issues with two similar projects, the OpencTD and the Sound Speed Profiler, a solution was sought to compensate for the effects of hysteresis. A Kalman filter is a versatile algorithm that can be utilized for these purposes. It is a way of combining multiple sources of input, like a mathematical prediction models of the system, different sensor measurements and prior knowledge of the system to a generate a statistically optimal estimate. There is also another temperature sensor embedded in the pressure sensor that could be used for these purposes. In the intended implementation it creates an estimate of the temperature that is a composite of several different sensor readings, the prior estimate of temperature, and a simple prediction model of the hysteresis effect.

Prediction step:

$$\bar{x}_k = F\hat{x}_{k-1}$$

$$\bar{P}_k = F\hat{P}_{k-1}F^T + Q$$

Update step:

$$K = \bar{P}_k H^T (H\bar{P}_k H^T + R)^{-1}$$

$$\hat{x}_k = \bar{x}_k + K(z_k - H\bar{x}_k)$$

$$\hat{P}_k = (I - KH)\bar{P}_k$$

The above formulas are a representation of the general implementation of the Kalman filter. The sensor vector z would contain readings from three different sensors and the matrix R would contain their variance found by letting the sensors run in a stable temperature environment and calculating their variance in excel. The hysteresis compensation effect was attempted modeled by storing the rate of change in temperature for the previous 5 samples of the CTD. A single variable would be added to the temperature estimate that was weighted to be larger with a recent dramatic change in temperature, and lesser affected by changes further back in time. Several variations of the Kalman filter was attempted but ultimately the project never progressed to the point where the implementation could be properly tuned.

9.1 Linear algebra on the Arduino

To implement the Kalman filter on Arduino special libraries were required to carry out the Kalman formulas. Doing vector and matrix operations are not natively supported by the Arduino IDE. Originally an Arduino port for the Eigen library was used. This is a large library for C++ with support for many kinds of linear algebra operations. It also requires a lot of the very limited storage of an Arduino microcontroller. Instead a smaller library called BasicLinearAlgebra.h by Tom Steward was chosen. This is a simpler library with much smaller overhead. It can perform simple matrix arithmetic and a few more advanced operations like transpose and inverse operations. The matrix elements are stored in c-style arrays, and can with a little extra consideration be combined naturally in normal algebraic expressions

10 Results

The Arduino CTD was field tested at Lysaker docks alongside a commercial CTD, a STD/CTD model SD204. The two CTDs were fastened together, attached to a rope and lowered by hand. The speed has been calculated from the SD204 data to be approximately 26 cm per second both descending and ascending. Three casts of 3,5, and 7 meters were done. Originally a deeper cast from the research ship "Trygve Braarud" was planned but fell through due to logistical reasons. Ideally more tests, with more varied temperature and salinity profiles would be preferred. The measurements were exported by text file and compared to measurements on the SD card of the Arduino CTD. Some differences in units of measurements had to be accounted for. The conductivity measurements from the Arduino CTD is reported in un-temperature compensated µS/cm while the SD204 output is reported in Practical Salinity Units. There are probably ways of reporting the true conductivity measurement through the SD204 computer interface, but this was not obtained while on site. Instead PSU values have been calculated from the Arduino CTD measurements in excel and compared in this way. The excel sheet formula is based on a salinity equation given in the paper Algorithms for computation of fundamental properties of seawater [30] . Some differences in calculation may have occurred. Likewise the pressure measurements of the SD204 were reported in meters below sea level instead of a pure pressure measurement in millibars. The formula used to calculate depth is $Depth = \frac{(p-p_0)}{g*100}$

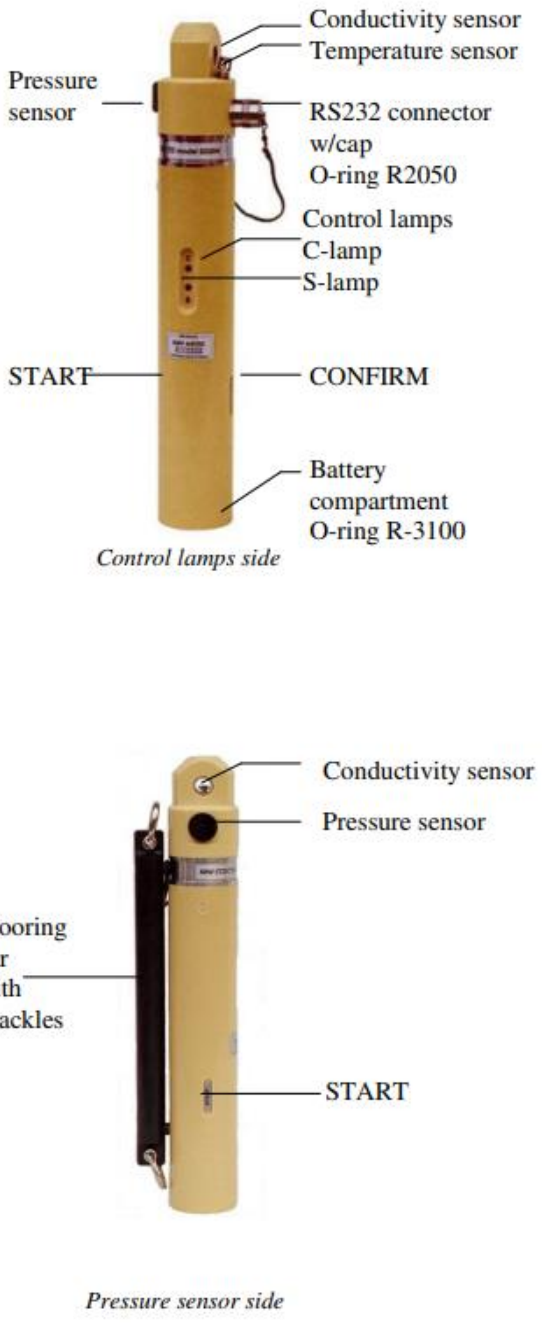


Figure 32 SD204 Commercial CTD [26]

Where p is the pressure reading in millibars, p_0 is the pressure at sea level and g is the acceleration due to gravity

10.1Salinity profile field test comparison

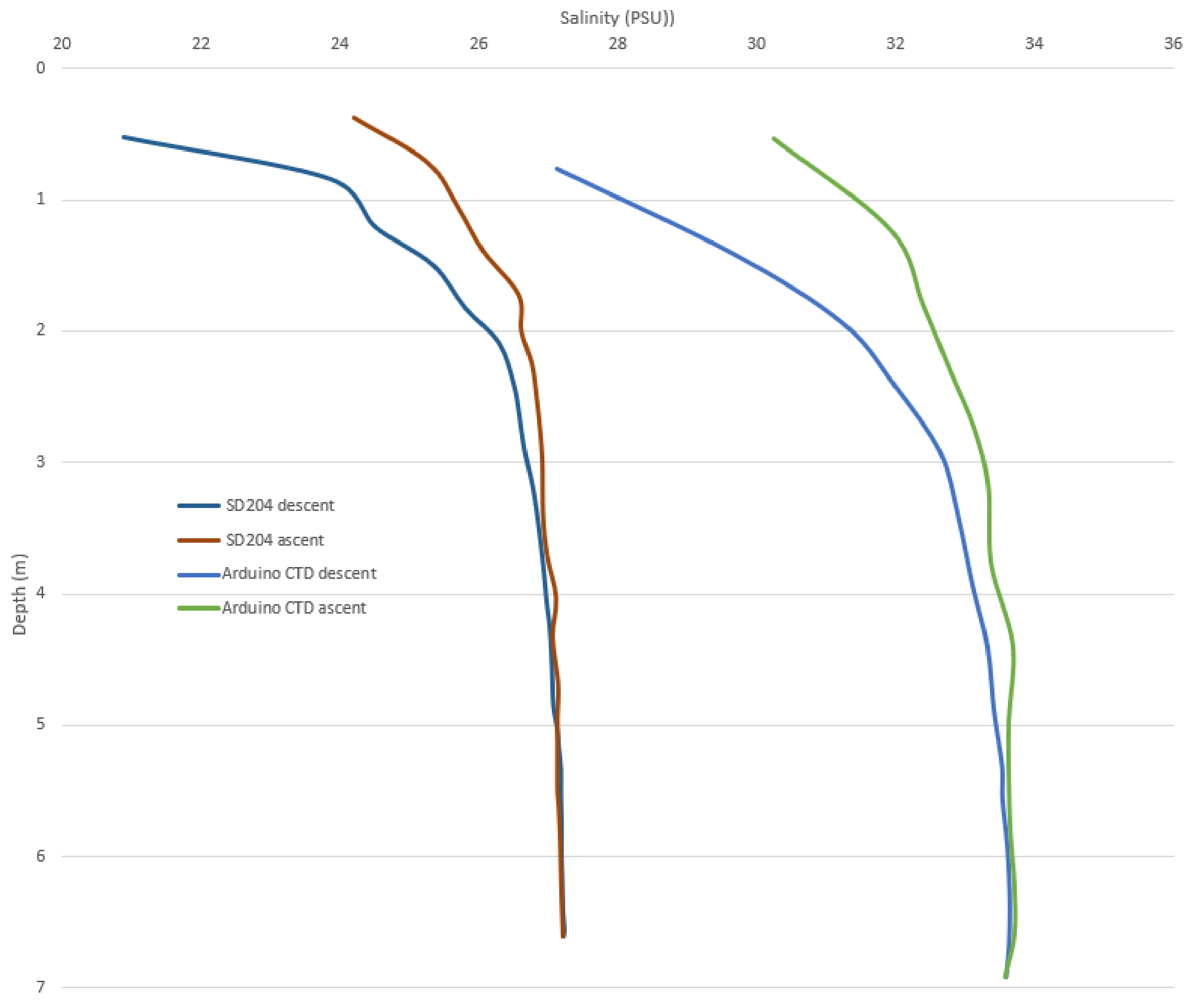


Figure 33 Salinity Profile SD204 comparison

Surprisingly there seems to be a hysteresis effect in both sensors reading different measurements on the way up then on the way down. This is surprising since the measurement pulse from the Atlas conductivity sensor is supposed to be instantaneous. This might be caused by the processing time of the Atlas EC circuit. It requires 600 ms from the time it receives a measurement command to return a reading. In this cast the CTD would have travelled 15cm in that time, and since this delay happens in both directions this could account for a 30cm offset between the ascending and descending reading at the same depth. This effect is present in both sensors, though more pronounced in the Arduino CTD.

The salinity profile of the Arduino CTD is clearly off from the profile of the SD204, though the curves are similar in shape. This might be caused by errors in calibration, or calculation. As a simple experiment, offsetting the Arduino CTD values by -6,5 PSU in the x direction and 31cm in the Y direction yielded this curve:

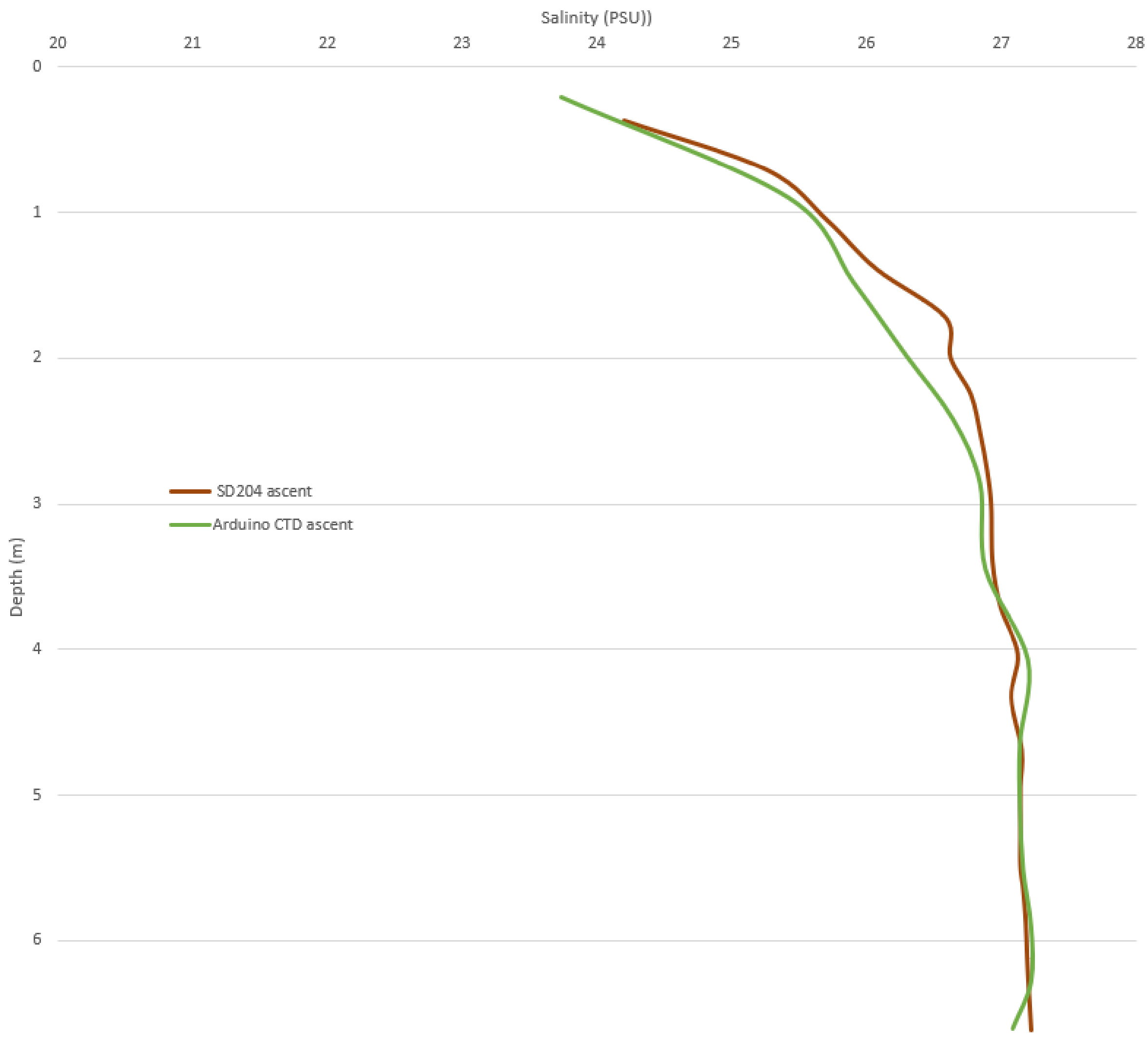


Figure 34 Salinity profile with offset

The 6,5 PSU offset to account for the calibration error and 31cm offset since the two pressure sensors measured different maximum depths at 6,92 and 6,61 meters. This results in curves that closely follows one another with the exceptions of a divergence at 2 meters followed by another smaller in the other direction at 4 meters. This is likely caused by the fact that the two CTDs joined together had an uneven weight distribution and would slightly rotate when pulled through the water. This same effect is visible on the temperature charts. The offset experiment is only a quick fix but shows promise that the conductivity sensor can perform well simply by getting the calibration right.

10.2Temperature profile field test comparison

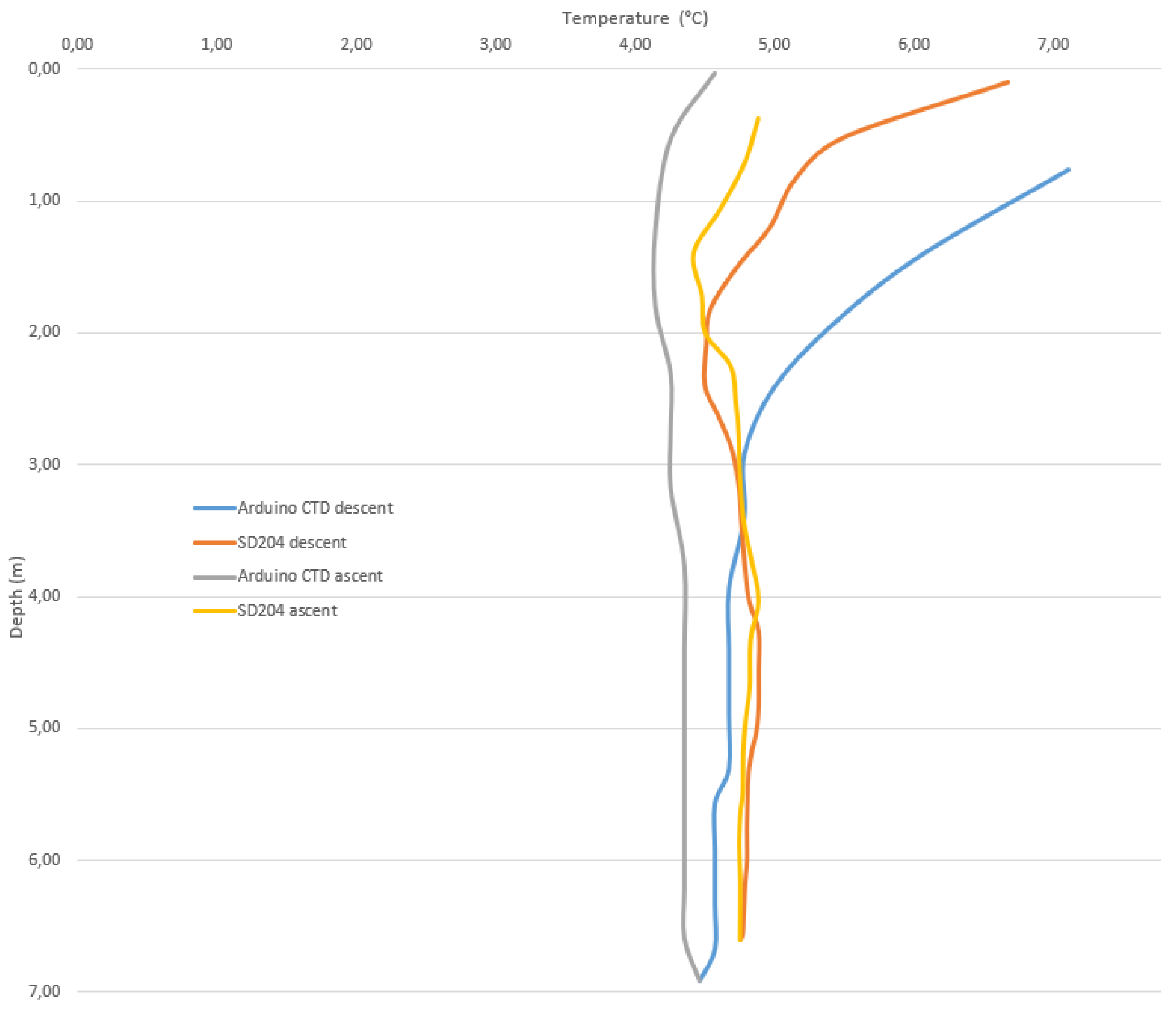


Figure 35 Temperature profile comparison

At the start of the descent both sensors show signs of a strong hysteresis effect caused by the dramatic temperature difference from air to water. This could be corrected by simply letting the instruments adjust for a few seconds below surface level before starting the descent. It is interesting however since it clearly shows the difference in sensor response time. The SD204 is decidedly quicker to adjust. Another strange effect is that the Arduino CTD shows a persistent difference in temperature at the same depth. If caused by thermistor response time the descending and ascending temperature values at the same depth should converge fairly quickly since the temperature values stay uniform throughout most of the cast.

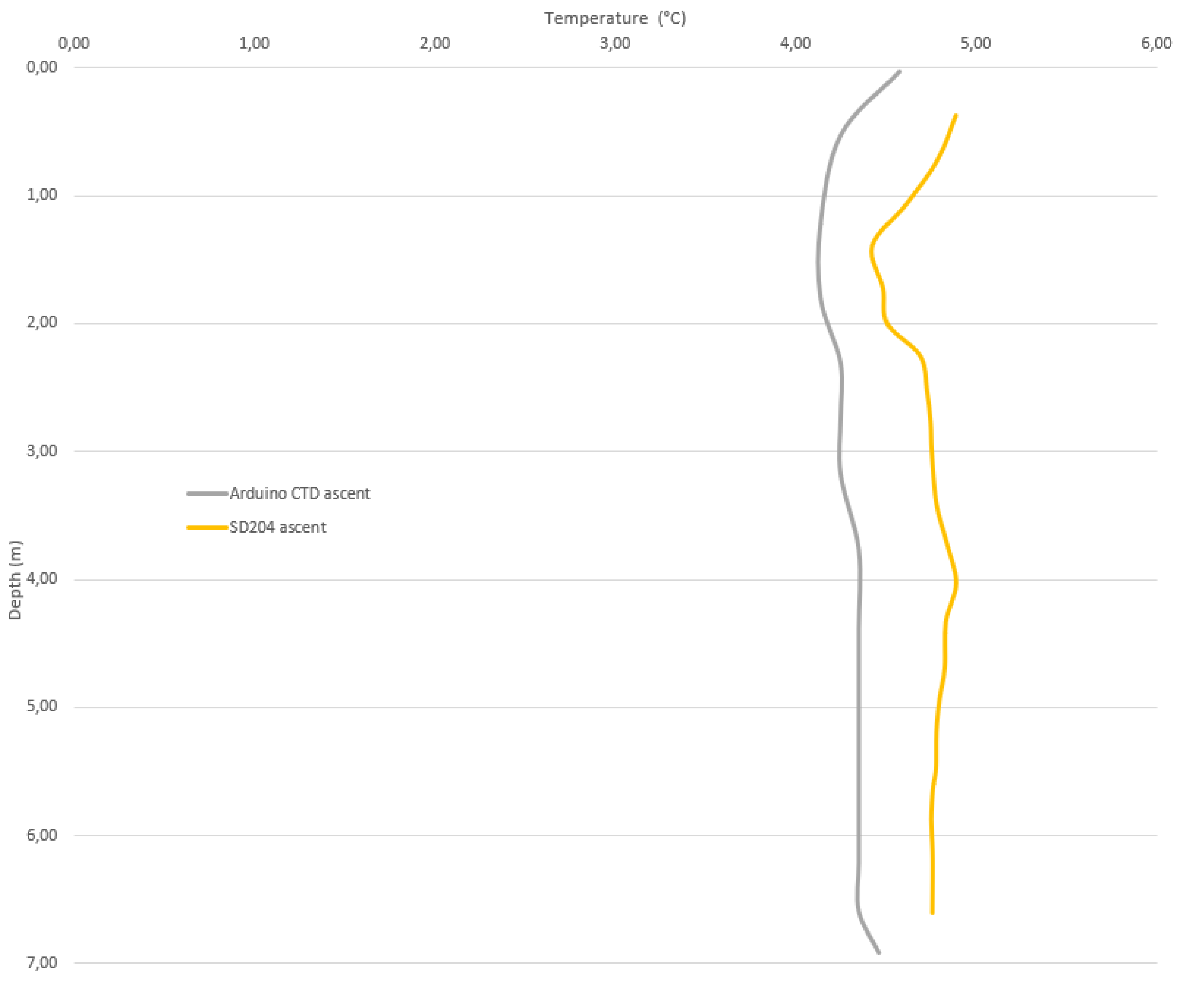


Figure 36 Temperature comparison ascent

Isolating the ascent you also see very similar curves, with some artifacts at 2 and 4 meters probably caused by rotation, but otherwise very little variation if corrected by an offset.

10.3Kalman Filter Profile

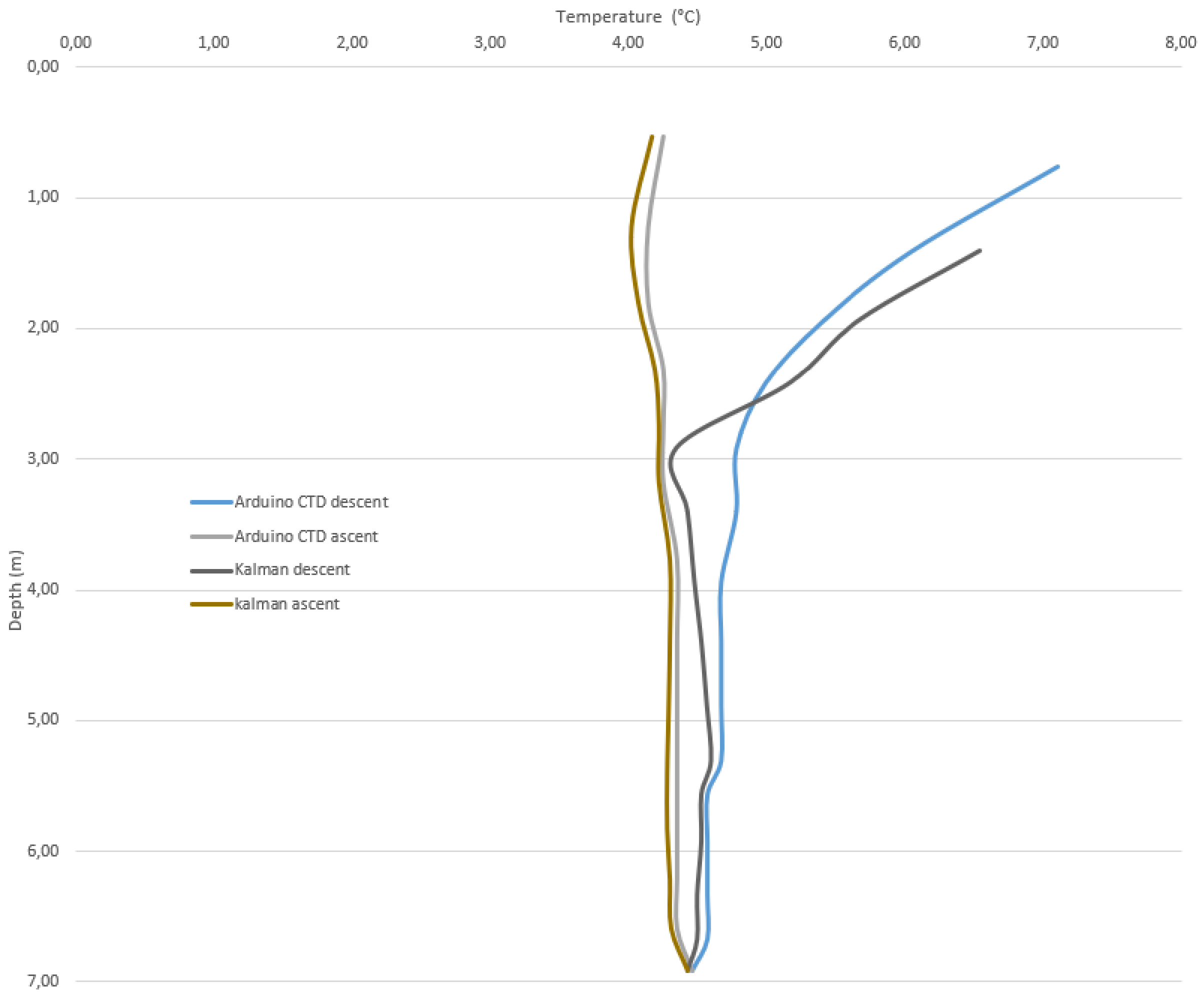


Figure 37 Kalman filter comparison

The Kalman filter in this configuration actually adds to the delay of the sensor instead of compensating for it. This is because the other two worse performing thermistors are weighing the estimate down. There is very little stochastic variation in individual thermistor readings, the error is dominated by bias effects, making the top performing thermistor always be the best estimate. The compensation is first visible once the temperature stabilizes and the Kalman filter overshoots, predicting the temperature to continue its momentum. This effect will be smaller with a more frequent sampling interval.

11 Conclusions and future work

Ultimately the project was successful in constructing a working low-cost CTD with wireless communication, with a clear improvement in temperature sensor response time over similar projects. A last-minute field test was also performed successfully alongside a commercial CTD.

An experiment was done to show the difference in response time between the Adafruit thermistor in epoxy and the DS18B20 digital temperature sensor. The stripped-down experiment in the lab showed a dramatic difference where the thermistor stabilized after an instant 20°C change in 5 seconds while the DS18B20 required 45 seconds to do the same. In the data from the field testing however it was noticed that the hysteresis effects were larger then should be expected from the lab tests, and larger than the hysteresis from the thermistors in the commercial CTD. This is most likely a coding problem, perhaps a delay in the reading of the pressure sensor, causing the temperature readings to be registered at the wrong depths. Father investigation is needed to correct this issue.

Some other extended features were tested but with limited results. The turbidity sensor was an experiment of what was possible with a very cheap open source sensor which turned out to not be very useful in a CTD. It is at best useful as an indicator of very high turbidity water. A useful sensor for clear water use would have to be sensitive in the 0-100 NTU range, while the DFRrobot sensor combined with the Arduino ADC seems to have a minimum resolution of about 50 NTU, and will fluctuate hundreds of NTU in clear water. There are few other such sensors available for purchase, however there is a paper called “Low-Cost Turbidity Sensor for Low-Power Wireless Monitoring of Fresh-Water Courses” [29] detailing the construction of the exact sort of sensor that would be appropriate in a low cost CTD. Future work could perhaps include an attempt at recreating this sensor.

The Kalman filter ultimately failed to be very useful since in practice, the best calibrated thermistor is always the best estimate. Little extra is to be gained from including information from the other two thermistors or the sensor embedded in the pressure sensor. The prediction model could if tuned correctly provide a slightly better hysteresis compensated estimate, but since the Arduino CTD has no real time communication ability this is probably better achieved in post processing. A much better estimate can be made with full knowledge of all the measurement samples in the data set, for example by averaging the ascending and descending temperature value at the same depth of a water column.

The robustness of the electronics is one of the main weaknesses of this project. The fragile nature of the PCB board and wiring setup combined with the tight spacing of the cylinder made for a lot of stability problems. The need to gently push the electronics together to make room for the gripper plug, as well as the need to pull it slightly outward to access the SD card or Arduino USB port causes wear on the solder joints and connectors that accumulates into fractures and breaks. This led to frequent failures and erratic behavior that was hard to troubleshoot and has made the CTD very unreliable in field testing. Future iterations should probably strive to make a more rigid setup, perhaps with a larger cylinder to allow for some flexibility while prototyping.

The field testing, fine tuning, and user interaction considerations of the equipment is ultimately where there this project is lacking. For an instrument to be truly low cost it needs to consider the practicality and lifetime costs of a user’s interaction with it. Improvements should be made to create a robust instrument that require little maintenance and has routines for easy re-calibration. This is where most where most non-commercial efforts at making low-cost instruments fall short since the scope of these projects seldom reach into the final stages of product development. This thesis has hopefully made a mid-tier contribution to the collaborative effort of making an ultra-low cost CTD, with improvements over previous projects most notably in temperature response time and wireless capability. The Sensors also show promise of being able to closely follow the readings of commercial instruments, but more time spent calibrating and testing is needed to confirm this. Some experiments, like the Kalman filter and turbidity sensor turned out be of limited usefulness, but the ideas and hard-learned lessons of this project may still be of aid to others. The next iteration of the sensor probe is currently underway by another master thesis at the University of Oslo

Appendix

11.1 Arduino code

```
unsigned long previousMillis = 0;        // for the millis function

const long interval = 1000;

#define SERIESRESISTOR 10000

#define THERMISTORPIN1 A0

#define THERMISTORPIN2 A1

#define THERMISTORPIN3 A2

#define THERMISTORPIN4 A3

#define TEMPERATURENOMINAL 25

#define THERMISTORNOMINAL 10000

#define BCOEFFICIENT 3950

float reading1;

float reading2;

float reading3;

//turbidity variable

float reading4;

float tvoltage;

float ntu;

// conductivity sensor libraries and variables

#include <Wire.h>                //enable I2C.

#define address 100

float EC_float = 0;

char EC_data[48];    // A 48 byte character array to hold incoming data from the conductivity
circuit.
```

```
char *EC;             // Character pointer for string parsing.

byte received_from_sensor = 0;  // How many characters have been received.

byte code = 0;


//pressure sensor libraries and variables

#include <MS5803_14.h>

MS_5803 sensor = MS_5803(512);

//matrix and vector libraries

#include <BasicLinearAlgebra.h>

using namespace BLA;


void setup() {

        Serial.begin(9600);   //start the serial monitor

        Serial1.begin(9600);   //start bluetooth

        Wire.begin();         //start i2c

        sensor.initializeMS_5803(false);

        Serial2.begin(9600);


}


void loop() {

        // Menu variables

        char val=6;         // variable to receive data from the serial port

        char logging;

        char character;

        char fileName[12];

        const byte numChars = 12;
```

```
boolean newData = false;

static byte ndx = 0;

char endMarker = '#';

char rc;

//kalman variables

float vartemp = 1.12184278324081E-05;

float voltage = 0.0;

BLA::Matrix<3,3> R;

R << 0.1, 0, 0,

0, 999, 0,

0, 0, 0.1;

BLA::Matrix<3> H;

H <<1,

1,

1;


BLA::Matrix<1,3> G;

G <<0,0,0;


BLA::Matrix<3> Z;

Z <<0,

0,

0;


BLA::Matrix<1> I;

I <<1;
```

```
BLA::Matrix<1> Xe;

Xe <<0;

BLA::Matrix<1> Xp;

Xp <<0;

BLA::Matrix<1> Zp;

Zp <<0;

BLA::Matrix<1> P;

P <<1;


BLA::Matrix<1> Pc;

P <<0;

BLA::Matrix<1> varProcess;

P <<0.25;


BLA::Matrix<3,3> C;

float reading1 =0;

float reading2 =0;

float reading3 =0;


float v1=0;

float v2=0;

float v3=0;

float v4=0;

float v5=0;

float lastreading1=0;

float lastreading2=0;

float lastreading3=0;
float Hys=0;
float steinhart1;
float steinhart2;
float steinhart3;
delay(500);
gotoCommandMode(); //Puts OpenLog in command mode
while(1){
        if( Serial1.available() )      // if data is available to read
        {;}
        val = Serial1.read();       // read it and store it in 'val'

        if( val == '0' )            // if val = 0 show menu
        {
                delay(500);                 // waits for a second
                Serial1.println("1 - Start logging");
                Serial1.println("2 - Stop logging");
                Serial1.println("3 - Retrieve data");
                Serial1.println("4 - SD Card info");
                Serial1.println("5 - View stored files");
        }
        if( val == '1' )            // if val = 1 start logging
        {
                Serial1.read(); // empty the buffer
                Serial1.println("Enter Filename and send # to confirm");
                while (newData == false){

                        while (Serial1.available() > 0 && newData == false) {
```

```
                    rc = Serial1.read();


                    if (rc != endMarker) {
                              fileName[ndx] = rc;
                              ndx++;
                              if (ndx >= numChars) {
                                        ndx = numChars - 1;
                              }
                    }
                    else {
                              fileName[ndx] = '\0'; // terminate the string
                              Serial1.println(fileName);
                              ndx = 0;
                              newData = true;
                    }
          }
     }

     newData = false;
     createFile(fileName);
     delay(1000);

     logging='1';
}
if( val == '2' )            // if '2' stop logging
{
        delay(1000);            // waits for a second
        Serial1.println("Stopping logging");
        logging='0';
        Serial2.write(26);
        Serial2.write(26);
        Serial2.write(26);
    }

    if( val == '3' )         // if '3' retrieve data
    {

        delay(1000);            // waits for a second
        Serial2.read(); // empty the buffer
        Serial1.println("Retrieving data");



        readFile(fileName); //This dumps the contents of a given file to the
serial terminal



    }if( val == '4' )          // if '4' show SD card info
    {

        delay(1000);            // waits for a second
        readDisk();
        }
        if( val == '5' )          // if val = 5 show files
        {

                delay(1000);             // waits for a second
                readLs();
        }

        unsigned long currentMillis = millis();

        if (currentMillis - previousMillis >= interval && logging=='1') {
                previousMillis = currentMillis;



                Wire.beginTransmission(address); // send conductivity measurement
command
                Wire.write('r');
                Wire.endTransmission();





                reading3 = analogRead(THERMISTORPIN3);
                reading2 = analogRead(THERMISTORPIN2);
                reading1 = analogRead(THERMISTORPIN1);

                steinhart1=steinhart(reading1);
```

```
                    steinhart2=steinhart(reading2);

                    steinhart3=steinhart(reading3);





                    Serial2.print(steinhart1);

                    Serial2.println(",");


                    Serial2.print(steinhart2);

                    Serial2.println(",");


                    Serial2.print(steinhart3);

                    Serial2.println(",");


                    reading4= analogRead(THERMISTORPIN4);

                    tvoltage = reading4 * (5.0 / 1024.0); // Convert the analog reading
(which goes from 0 - 1023) to a voltage (0 - 5V):

                    ntu =-1115.78*tvoltage*tvoltage+3446*tvoltage+353;


                    sensor.readSensor();


                    Serial2.print(sensor.pressure());

                    Serial2.println(",");

                    Serial2.print(sensor.temperature());

                    Serial2.println(",");
                    delay(600);


                    Wire.requestFrom(address, 48, 1); //retrieve conductivity measurement

                    code = Wire.read();

                    if (Wire.available() > 0) {

                          received_from_sensor = Wire.readBytesUntil(13, EC_data, 48);

                          // Null terminate the data by setting the value after the final
character to 0.

                          EC_data[received_from_sensor] = 0;

                    }

                    Wire.endTransmission();

                    // Parse data, if EC_data begins with a digit, not a letter (testing ASCII
values).

                    if ((EC_data[0] >= 48) && (EC_data[0] <=57)) {

                          parse_data();

                    }






                    Serial2.print(EC_data);

                    Serial2.println(",");
```

```
v5=v4;

v4=v3;

v3=v2;

v2=v1;

v1=steinhart1-lastreading1;

if(lastreading1 == 0){

        v1=0;

}

lastreading1=steinhart1;


Hys=v1*0.5+v2*0.3+v3*0.2+v4*0.1+v5*0.05;

if(Hys>1||Hys<-1){

        Hys=0.1;

}


Z(0) = steinhart1+Hys;

Z(1) = steinhart2+Hys;

Z(2) = steinhart3+Hys;


// kalman process

Pc = P + varProcess;

C=H*Pc*~H+R;

G = (Pc*~H)*C.Inverse();    // kalman gain

P = (I-G*H)*Pc;

Xp = Xe;

Zp = Xp;

Xe = G*(Z-H*Zp)+Xp;   // the kalman estimate of the temperature
```

```
                        delay(100);
                        Serial2.print(Xe(0));
                        Serial2.print(",");
                        Serial2.println();
                        Serial2.println();
                        if(isnan(Xe(0))){
                                Xe(0)=steinhart1;
                                v1=0;
                                v2=0;
                                v3=0;
                                v4=0;
                                v5=0;
                        }
                }
        }
}
float steinhart(float reading) {
        // convert the value to resistance
        reading = (1023 / reading)  - 1;     // (1023/ADC - 1)
        reading = SERIESRESISTOR / reading;  // 10K / (1023/ADC - 1)
        float result;
        result = reading / THERMISTORNOMINAL;     // (R/Ro)
        result = log(result);                  // ln(R/Ro)
        result /= BCOEFFICIENT;                  // 1/B * ln(R/Ro)
        result += 1.0 / (TEMPERATURENOMINAL + 273.15); // + (1/To)
        result = 1.0 / result;                 // Invert
        result -= 273.15;                        // convert to C
        return result;
}



void parse_data() {

        EC = strtok(EC_data, ",");

}



void createFile(char *fileName) {



        Serial2.print("new ");
        Serial2.print(fileName);
        Serial2.write(13);




        while(1) {
                if(Serial2.available())
                if(Serial2.read() == '>') break;
        }

        Serial2.print("append ");
        Serial2.print(fileName);
```

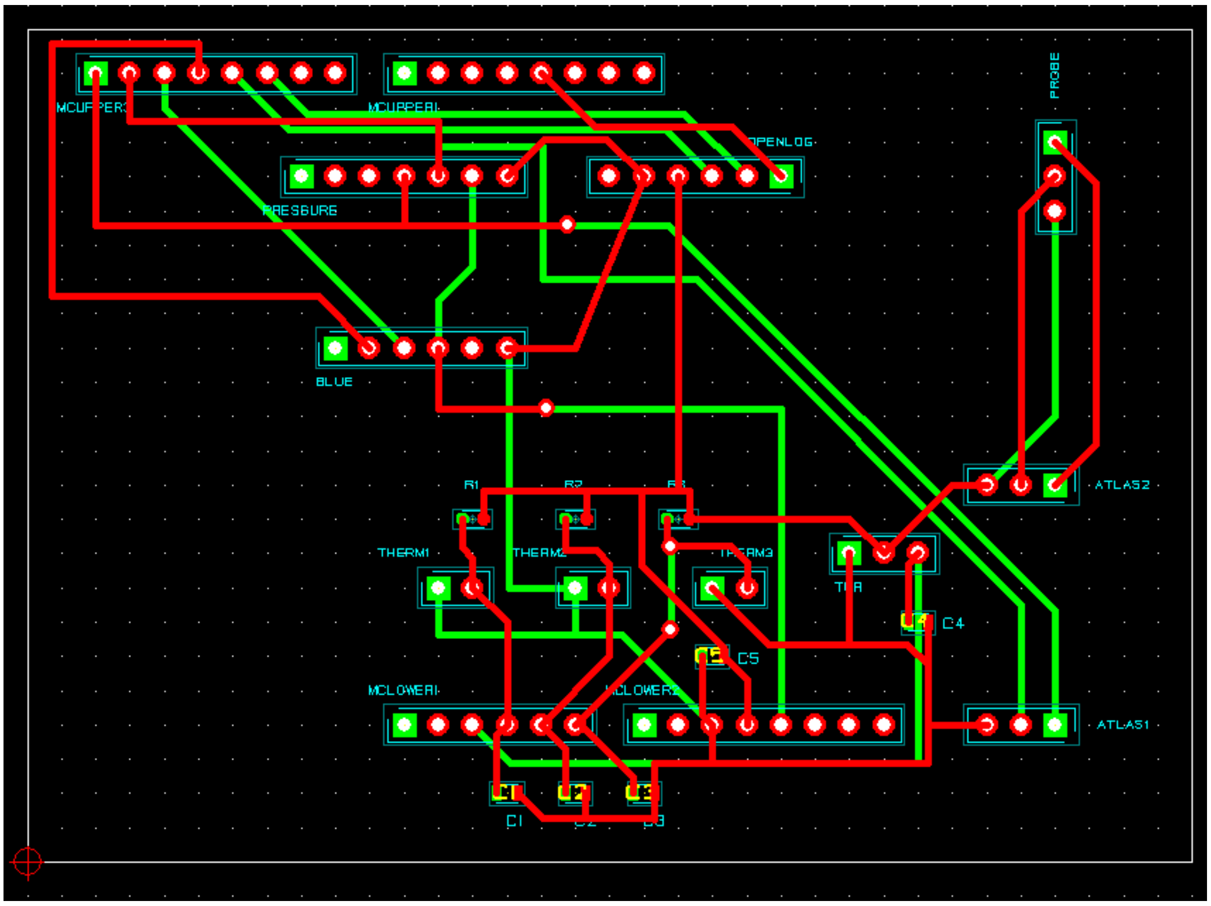
```
	Serial2.write(13); //This is \r



	while(1) {
		if(Serial2.available())
		if(Serial2.read() == '<') break;
	}



}

void readFile(char *fileName) {



	Serial2.print("read ");
	Serial2.print(fileName);
	Serial2.write(13); //This is \r

	while(1) {
		if(Serial2.available())
		if(Serial2.read() == '\r') break;
	}

	for(int timeOut = 0 ; timeOut < 1000 ; timeOut++) {
		while(Serial2.available()) {
			char tempString[100];
```

```
                        int spot = 0;
                        while(Serial2.available()) {
                                tempString[spot++] = Serial2.read();
                                if(spot > 98) break;
                        }
                        tempString[spot] = '\0';
                        Serial1.write(tempString);
                        timeOut = 0;
                }

                delay(1);
        }


}


void readDisk() {


        Serial2.print("disk");
        Serial2.write(13);



        while(1) {
                if(Serial2.available())
        if(Serial2.read() == '\r') break;
    }

    for(int timeOut = 0 ; timeOut < 1000 ; timeOut++) {
        while(Serial2.available()) {
            char tempString[100];

            int spot = 0;
            while(Serial2.available()) {
                tempString[spot++] = Serial2.read();
                if(spot > 98) break;
            }
            tempString[spot] = '\0';

            Serial1.write(tempString);
            timeOut = 0;

        }

        delay(1);
    }


}
void readLs() {

    Serial2.print("ls");
```

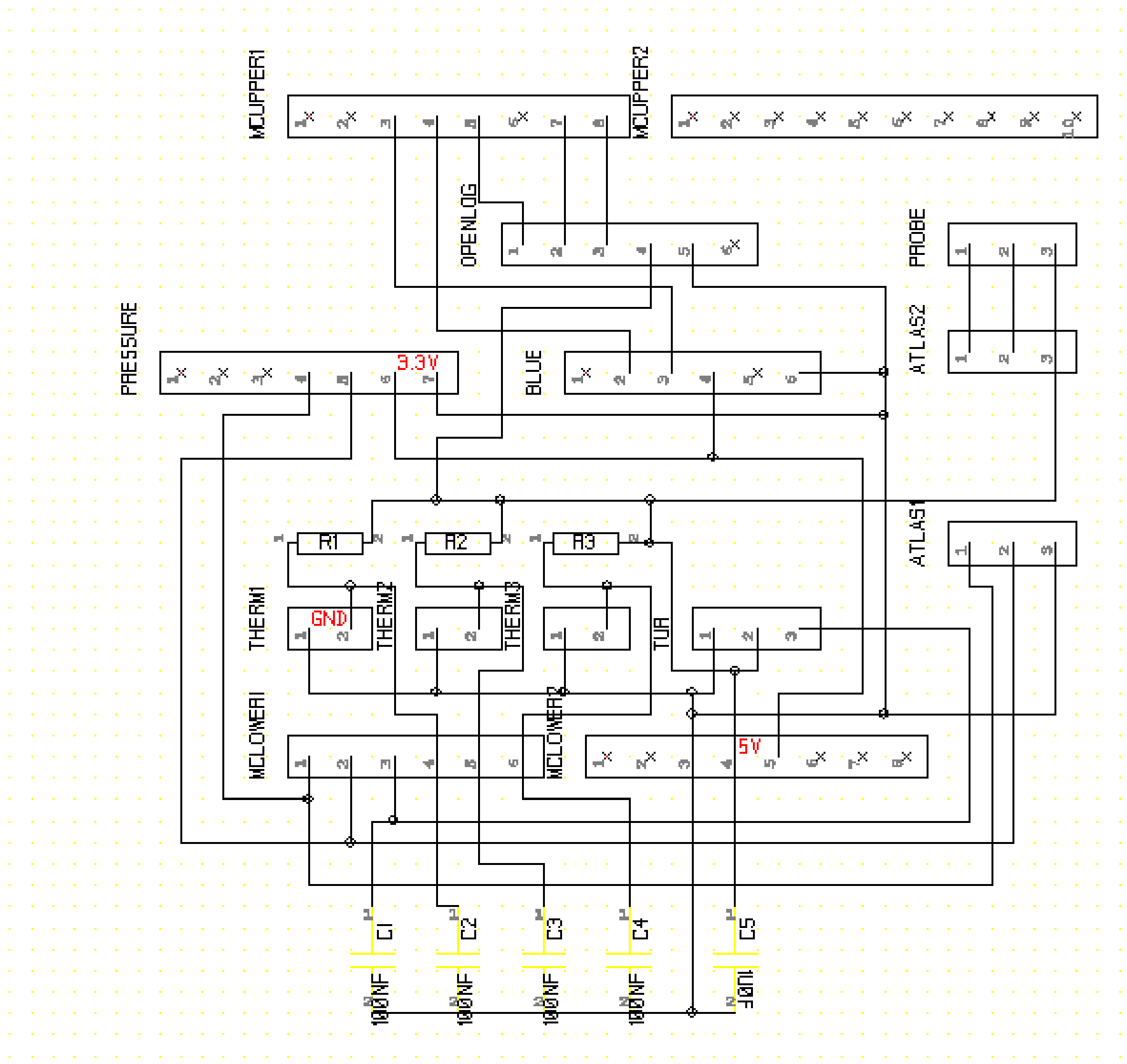
```
Serial2.write(13); //This is \r


while(1) {
        if(Serial2.available())
        if(Serial2.read() == '\r') break;
}



for(int timeOut = 0 ; timeOut < 1000 ; timeOut++) {
        while(Serial2.available()) {
                char tempString[100];


                int spot = 0;
                while(Serial2.available()) {
                        tempString[spot++] = Serial2.read();
                        if(spot > 98) break;
                }
                tempString[spot] = '\0';


                Serial1.write(tempString);
                timeOut = 0;


        }


        delay(1);
}
```

```
}


void gotoCommandMode(void) {

        Serial2.write(26);
        Serial2.write(26);
        Serial2.write(26);


        while(1) {
                if(Serial2.available())
                if(Serial2.read() == '>') break;
        }
}
```

11.2PCB routing Layout



11.3PCB schematic



References

[1] Urik,R. (1983) Principles of underwater sound. 3rd. edition. Westport: McGraw-Hill.

[2] Fisher, F., & Simmons,V. (1977) Sound absorption in Sea Water. Journal of Acoustic Society America

[3] C-T. Chen and F.J. Millero, (1977) Speed of sound in seawater at high pressures. Journal of Acoustic Society America

[4] Leighton, Timothy & C P Evans, R. (2007) Studies into the Detection of Buried Objects (Particularly Optical Fibres) in Saturated Sediment. Part 2: Design and Commissioning of Test Tank. ISVR Technical Report No 310

[5] Wong, G.S.K. & Zhu, S. (1995) Speed of sound in seawater as a function of salinity, temperature and pressure J. Acoust. Soc. Am.

[6] National Physical Laboratory (2000) Underwater Acoustics -Technical Guides - Speed of Sound in Sea-Water. Retrieved from http://www.comm-tec.com/library/technical_papers/speedsw.pdf

[7] Shaban, A (2017) Design of sound speed profiler -Water Parameter Sensor (Master thesis) University of Oslo

[8] Langis, D. P. (2015). Arduino Based Oceanographic Instruments: An Implementation Strategy for Low-Cost Sensors.

[9] C Leroy, Claude & Robinson, Stephen & Goldsmith, M. (2008). A new equation for the accurate calculation of sound speed in all oceans. The Journal of the Acoustical Society of America.

[10] Naval Postgraduate School Department of Oceaonography (2014) Ocean Accoustics. Retrieved from https://www.oc.nps.edu/~bird/oc2930/acoustics/

[11] McGraw-Hill Concise Encyclopedia of Physics. (2002) Refraction of waves. Retrieved from https://encyclopedia2.thefreedictionary.com/Refraction+of+waves

[12] RMB Environmental Laboratories, inc. (2018) Stratification and Mixing. Retrieved from https://www.rmbel.info/primer/stratification-and-mixing/

[13] Ocean Networks Canada (2014) Instrument overview CTD. Retrieved from http://www.oceannetworks.ca/sites/default/files/pdf/learning/community_observatories/instrument_overivew_CTD_19Aug2014.pdf

[14] LakeAccess. (2006) Lake Temperature. Retrieved from http://www.lakeaccess.org/russ/temperature.htm

[15] Fondriest Environmental, Inc. (2014, 3. March) Conductivity, Salinity and Total Dissolved Solids. Fundamentals of Environmental Measurements. Retrieved from https://www.fondriest.com/environmental-measurements/parameters/water-quality/conductivity-salinity-tds/

[16] National Ocean Council. (2013). Federal Oceanographic Fleet Status Report

[17] Argo. (2017). How Argo floats work. Retrieved from http://www.argo.ucsd.edu/How_Argo_floats.html

[18] Oceanography for Everyone. (2015) Oceanography for everyone. Retrieved from http://oceanographyforeveryone.com/

[19] Wood, S, & Pardis, R. (2013). Inexpensive Expendable Conductivity Temperature and Depth (CTD) Sensor.

[20] Thaler, A. & Sturdivant, K. (2013). Oceanography for Everyone - The OpenCTD. Retrieved from http://www.rockethub.com/projects/26388-oceanography-for-everyone-theopenctd#description-tab

[21] Sigdel, B. (2017) Water Quality Measuring Station. Helsinki Metropolia University of Applied Sciences

[22] Atlas scientific (2019) EZO-EC Datasheet. Retrieved from https://www.atlas-scientific.com/_files/_datasheets/_circuit/EC_EZO_Datasheet.pdf

[23] Sparkfun (2019) MS5803-14BA Pressure Sensor Hookup Guide. Retrieved from https://learn.sparkfun.com/tutorials/ms5803-14ba-pressure-sensor-hookup-guide?_ga=2.44228262.381375694.1555257905-1139013887.1523364757

[24]DFRrobot (2017) Turbidity Sensor Wiki. Retrieved from https://www.dfrobot.com/wiki/index.php/Turbidity_sensor_SKU:_SEN0189

[25] Thinksrs (2014) Calibrate Steinhart-Hart Coefficients for Thermistors Retrieved from https://www.thinksrs.com/downloads/pdfs/applicationnotes/LDC%20Note%204%20NTC%20Calculator.pdf

[26] SAIV A/S (2010) OPERATING MANUAL for STD/CTD model SD204 with Sound Velocity & Optional Sensors Retrieved from: http://station.saivas.net/manuals/SD204%20manual_total.pdf

[27] Virginia Institute of Marine Science. (2012). What is the monetary value of a healthy ocean? Retrieved from: www.sciencedaily.com/releases/2012/03/120322100417.html

[28] Peak Sensors (2011) Thermal Response and Temperature Sensors. Retrieved from https://www.peaksensors.co.uk/blog/thermal-response-temperature-sensors/

[29] Wang, Y. & Rajib, S. & Collins, C. & Grieve, B. (2018) ,Low-Cost Turbidity Sensor for Low-Power Wireless Monitoring of Fresh-Water Courses, in IEEE Sensors Journal

[30] Fofonoff, N.P. & Millard Jr, R.C. (1983) Algorithms for the computation of fundamental properties of seawater. Paris, France, UNESCO, UNESCO Technical Papers in Marine Sciences

[31] Hayashi, M. (2004). Temperature-Electrical Conductivity Relation of Water for Environmental Monitoring and Geophysical Data. Inversion Environ Monit Assess

[32] Aquion (2018). Temperature Compensation for Conductivity. Retrieved from: http://www.aqion.de/site/112