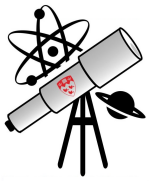
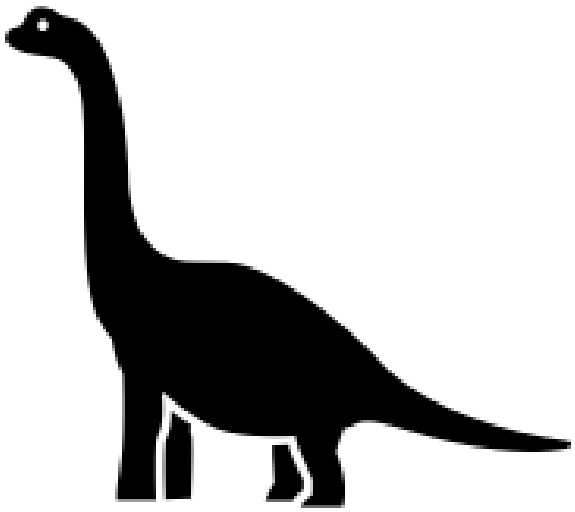
McGill Radio Lab



BRACHIOSAURUS
User Manual



Felix St-Amour

Prof. H. Cynthia Chiang

McGill University Department of Physics

January 26, 2025

Contents

Acknowledgments

I would like to thank the incredible people in the McGill radio lab who helped me throughout the summer. A special thank you to Jamie Cox without whom BRACHIOSAURUS would look quite different. Another special thank you to Ian Hendricksen for his patience and the weekly meetings to keep the project headed in the right direction. A massive thank you to Eamon Egan and Brandon Ruffolo for their help conceiving the electronics and the box of BRACHIOSAURUS.

Introduction

This document is meant as an introductory manual for Prof. Laura Thomson and her team at Queen's University along with any group or researchers who might be interested in this measurement method. It is also geared towards any future person who might pick up this project to improve upon it.

The BRACHIOSAURUS stands for Boxed Recorder Analyzing the Change in the Height of Ice with On-Site Accelerometer and Ultrasonic Readers Utilizing Support. It aims to harness the oscillatory motion of poles under wind conditions on glaciers in order to obtain the change in ice shelf thickness measurements over time. The wind causes the pole to oscillate at a frequency that is dependent on the exposed pole length. This means that at any time of the year, a pole oscillating ever so slightly on a glacier can tell us its exposed length and hence how much the glacier underneath is melting.

This document is separated into the following sections:

Get started with BRACHIOSAURUS is a summary section explaining the bare minimum to anyone who receives a fully assembled BRACHIOSAURUS and wants to know how to set it up and get the data stored on the micro SD card.

Assembly details the steps to assemble the BRACHIOSAURUS from its commercially available components.

PCB design explanation and comparison serves as written proof that I did not spend the entire summer fiddling my thumbs. It describes every electronic component present on the printable circuit board (PCB), explains why it is being used, and compares it to similar products. Future steps to reduce the cost and improve the BRACHIOSAURUS are also detailed.

Pole physics describes how the formula for frequency as a function of exposed pole length was derived and explains how the constants in the derived formula were found for the poles used at the McGill Arctic Research Station (MARS). Comments are given on the potential unaccounted differences between lab and field results.

Code explanation goes through the code present on the microcontroller to take data and the Python code used for data processing. A brief explanation of how to code on the ESP32 microcontroller is also given for anyone not used to the specificities of these microcontrollers.

1 Get started with BRACHIOSAURUS

Setup the box to get data

The fully assembled BRACHIOSAURUS (boxed recorder analyzing the change in the height of ice with on-site accelerometer and ultrasonic readers utilizing support) is designed for ease of use. The box is made of two main parts; the lid and the box itself. The lid attaches to the pole via a U-bolt that can be tightened with a wrench as well as a metal zip-tie. The box contains the electronics. The two are assembled with 4 watertight screws that can be opened with a standard #1 Phillips screwdriver.

Opening the box, two battery holders can be seen near the bottom (black rectangular boxes). They can be opened by removing the Phillips screw holding the battery holders shut and then sliding the lid of the battery holder in the direction of the arrow printed on the battery holders (it is possible that the screw has already been removed, it is not important). Insert 3 lithium AA batteries in each battery holder. If necessary, only one battery holder can be filled with batteries. If a battery holder does not have 3 batteries, it will not output current, so it will be rendered useless.

An ON/OFF switch can be seen at the top right of the printable circuit board (PCB). A small error causes the ON and OFF signs to be inverted. Flip the switch to OFF in order to power on the BRACHIOSAURUS. You will see that two green lights light up for approximately two minutes before going dark. From then on, the box will record data for two minutes every 4-6 hours.

Ensure that an industrial-grade micro SD card is inserted in the micro SD card mount at the top right. Close the box with the 4 waterproof screws, ensuring they fit tightly. Mount the box at the highest point of the pole and insert the pole in the ice leaving 1 m of exposed pole.

Retrieve data

When returning to the box, dismount it from the pole and open the box (special instructions are given in the Code section for boxes out of reach, look at the end of the ESP32 code). Retrieve the micro SD card and insert it in your computer. On the micro SD card, you should be able to find `data_[box_number].csv` and `motion_[box_number].bin`, the first being for the data taken and processed by BRACHIOSAURUS and the second for the motion read by the accelerometer (the data from the accelerometer is redundant as it was already processed by the BRACHIOSAURUS, but it is important for the first year of tests). Download these files to your computer ensuring that the box number is in the name of the files. Add the files to your desired folder along with the `python_DAQ.py` code that can be found here: `https://github.com/FelixStA52/BRACHIOSAURUS`. You can have as many data files as you want in this folder (i.e. data files from boxes 1 through 15, or more!). Now, in your terminal, run `cd path/to/python` where `path/to/python` is the path to the file containing the `python_DAQ.py` code as well as the data. You can then run `python python_DAQ.py` in the terminal. If everything runs smoothly, you should see that the code outputs the list of all box numbers as well as a message that the files `final_data_[box_number].csv` have been created. Table 1 presents what these files might look like.

Time	Temperature (^{o}C)	Length from motion.csv	Error on length from motion.csv	Length from ESP32	Error on length from ESP32	Length from depth sensor	Error on length from depth sensor
1745128716	5.02	1.32	0.04	1.33	0.04	1.30	0.01
1745150316	4.53	1.34	0.04	1.34	0.03	1.31	0.01
...	...	...	...	...	...	...	...

Table 1: Data as presented in the `final_data_[box number].csv` file. Note that the time is in the Unix format; the temperature in ^{o}C, and all lengths as well as the error on lengths are in meters.

If you are interested in the "raw data", the rows of the motion.bin file contain the time of the measurement followed by the magnitude of the measured acceleration. Each row represents one set of measurements. The data.csv file presents 7 columns. Column 1 represents how many times the box woke up. Column 2 represents the iteration of the data being taken within a wake-up cycle. Column 3 represents the Unix time of the wake-up cycle,

this can easily be converted into EST date and time. Column 4 represents the temperature as recorded by the accelerometer. Column 5 and 6 represent the two main frequencies of oscillation as computed by BRACHIOSAURUS. Finally, column 7 represents the distance as measured by the ultrasonic depth sensor.

The code was made to handle the majority of issues on its own, so it should not require any human input. Please contact Felix St-Amour (felix.st-amour@mail.mcgill.ca) if nonsensical values are returned. Some tuning of the code might be necessary after the first version of BRACHIOSAURUS.

2 Assembly

The instructions on how to assemble BRACHIOSAURUS could previously be found here. They were promoted to having their own manual with tons of fun and colorful pictures. Please refer to the assembly manual for instructions and protocols. This manual can be found in the following GitHub page: `https://github.com/FelixStA52/BRACHIOSAURUS`.

3 PCB design explanation and comparison

Microcontroller

The microcontroller used to control the BRACHIOSAURUS is an ESP32, more precisely a DFR0654 (roughly $15CA). It uses a USB-C communication cable to program it. Most ESP32, including DFR0654, can be programmed using the Arduino IDE, hence allowing for wide compatibility with Arduino-based libraries. The Arduino IDE uses Arduino code which is mostly based on C++ and C.

There are 3 main factors driving the selection of a microcontroller. The first one is operating temperature. The operating temperature of the DFR0654 ESP32 (like that of most ESP32 microcontrollers) is -40^{o}C. Other common microcontrollers like Arduino microcontrollers and the Raspberry Pi Pico are rated to -20^{o}C. In Arctic conditions, winters can easily reach -40^{o}C temperatures, so it is highly preferable to use a microcontroller that can withstand these temperatures. The microcontrollers that are limited to higher temperatures would likely be able to survive -40^{o}C temperatures, but it might cause some issues for the other peripherals (not being able to power them or not being able to communicate with them). As a first deployment, no unnecessary risks were taken, so all the electronics are rated to -40^{o}C.

The second factor is the ability to enter a very low power consumption mode. The ESP32 advertises itself as a low-power microcontroller. Unfortunately, the reality is that some ESP32 boards are not optimized to take advantage of the low-power capabilities of the ESP32. Even within the products advertised by a company, some can reach a low-power mode while others can't. Take as an example DF Robot, their DFR0478 ESP32 was tested to reach a low-power consumption mode in the hundreds of microamperes. While this is low, it is still too large for our applications (see the batteries section). Luckily, the DFR0654 can reach 10-20 microamperes in low-power mode. Another controller that can enter a similar low-power mode (and reach -40^{o}C) is the STM32, though it tends to be pricier than the DRF0654 ESP32.

The third factor is the sheer computational power of the microcontroller's chip. To contrast microcontrollers with your usual Raspberry Pi, a Raspberry Pi is its own computer while microcontrollers are made to run a code and interface peripherals with pins using a simple chip. Your typical Arduino will be able to perform a Fast Fourier Transform (FFT), a relatively labor-intensive operation, on an array of 64 numbers in tens of milliseconds (that's quite slow). The ESP32 is optimized to do the same FFT in a fraction of a millisecond. Matter of fact, the code running on the DFR0654 requires it to do FFTs on arrays of 2048 numbers which requires a lot more memory. As a comparison, an average Arduino crashes before reaching an FFT of 512 numbers because its memory is not good enough. In order to know if a certain microcontroller is sufficiently strong, many blogs discuss the power of specific microcontrollers specifically for their ability to perform FFTs.

Optional but highly recommended is the ability to use WiFi and Bluetooth Low Energy (BLE). This allows for other methods of communication if necessary. Most microcontrollers are not made specifically for WiFi and BLE which means that you need to use custom versions of these microcontrollers which are both pricier and might restrict the number of libraries you can use on the Arduino IDE (if the microcontroller uses an uncommon Arduino board library). The ESP32 is made specifically for IoT (Internet of Things) purposes, so it comes with WiFi and BLE with tons of articles, blogs, and videos explaining how to use it.

In the future, it might be possible to either optimize the choice of microcontroller to make it cheaper (and maintain the requirements previously stated) or to use solely the ESP32 chip without any board. Using only the chip could be an interesting option as the DFR0654 board has tons of functionality that are not being used for this specific project. The only part that is de facto mandatory is the voltage adapter (and capacitor to stabilize the incoming voltage) to take in the 4.5V from the batteries and transform it into a usable voltage for the chip, but everything else could potentially be dropped. This would reduce the price of the microcontroller to something in the range of $5CA.

Micro SD card mount and micro SD card

The micro SD card mount is quite simple to implement. All it requires is for the mount itself to be connected to a microcontroller with voltage in, ground, chip select, clock, MISO, and MOSI. The MISO and MOSI pins are used for communication with the micro SD card via the SPI protocol. Other communication protocols can be used, but SPI is by far the simplest and fulfills all the requirements for this project.

Multiple libraries exist on the Arduino IDE to interface micro SD cards via the SPI protocol. The one used here is aptly named "SD.h" library. It allows the user to interact with the file system of the micro SD card in a very similar fashion to what would be done in Python on a computer.

One last thing to note about the mount is that writing to a micro SD card tends to be an energy-intensive endeavor as it can require 100 mA. As a comparison, then entire PCB, without writing to the micro SD card, consumes about 50 mA while taking data.

When it comes to the micro SD card used, stronger requirements are at play. Most micro SD cards are not rated to temperatures of -40^{o}C, this means that the files are quite likely to become corrupted if the microcontroller tries to read and write to them at such temperatures. Therefore, the micro SD cards used need to be of industrial grade. This means that the micro SD cards are extremely resilient to harsh temperatures (rated to -40^{o}C), but they cost considerably more. A 256MB card is roughly $10CA while a 4GB card is roughly $20CA.

These micro SD cards will be able to survive a year without reaching their maximum capacity, but it is highly recommended that the data is retrieved and erased from the micro SD cards after one year. The Arduino program will aim at maxing out the memory of these cards with data for the first year. Hence, data taken after the first year (if the micro SD card is not retrieved and erased) will likely be lost.

It is also important to note a built-in method that can be used for storing data on the microcontroller. The ESP32 comes with a built-in SPIFFS file system which, for the purposes

of taking large sets of data, is largely insufficient due to the low storage capacity of the ESP32 (32 Mbit of flash memory) as well as the relatively low read-write cycle limit before the system starts deteriorating. Therefore, using a micro SD card is preferable.

Accelerometer

The accelerometer currently used is the LSM6DSOX (roughly $20CA). The accelerometer measures its acceleration and sends it to the microcontroller via the I2C protocol. The I2C protocol uses the SCL pin (clock imposed by the microcontroller) and the SDA pin (data sent on the rhythm of the clock). It acts both as the accelerometer and the temperature sensor of the BRACHIOSAURUS.

This accelerometer was selected despite its higher-than-average price for the simple fact that it is the only readily available accelerometer that has a temperature rating of -40^{o}C. Unfortunately, it was realized too late into the project that it would be considerably cheaper and not much more difficult to use an Inertial Measurement Unit (IMU) instead of the accelerometer. The difference between the two is that the IMU is a simple chip while the accelerometer is a board including an IMU chip, a voltage adapter, a capacitor, and other components. It would be quite easy to reverse-engineer an accelerometer board in order to keep only the IMU chip. This would bring the price down from $20CA to $5CA for a chip that includes a temperature sensor. Another positive factor is that the vast majority of IMU chips are rated to -40^{o}C while very few accelerometer boards are. Interfacing with the chip would remain similar as most use the I2C protocol.

Depth sensor

The main ultrasonic depth sensor used in the BRACHIOSAURUS is the HC-SR04 ($5CA). Ultrasonic depth sensors use one or two transceivers. When two are used, a transceiver sends a sound wave while the other transceiver listens for the echo. From the difference in time between the emission and the echo, the distance can be inferred since the speed of sound in the air is 343 meters per second.

The choice of a specific depth sensor is quite complex. The HC-SR04 is the simplest form that can be found. Its transceivers are open which means that only a metal grid protects the emitter and receiver. This makes the sensor quite prone to water and snow damage. Some depth sensors have closed transceivers which means that the emitter and transmitter are completely encapsulated in metal or plastic. While this makes them more resistant to outdoor weathering, it also means that they are weaker (have a shorter range) because the sound wave is damped by the enclosure.

For this project, the transceivers are made to protrude through the bottom of the box. O-rings then seal the box. If the transceivers used were closed, this would heavily damp the sound wave produced and the echo received (since the transceivers need to vibrate in order to emit and receive sound, the vibrations are damped by contact with the O-rings).

This was tested experimentally by removing the transceivers of a normal HC-SR04 depth sensor and replacing the transmitter and receiver by closed ones (the closed transceivers were chosen to have the same driving frequency as the open transceivers). This lead to much weaker signals when untouched (hanging from wires) and became completely null when the closed transceivers were touching any surface (since the sound became too damped).

Adding a truncated cone where the depth sensor is protruding from the would-be apex can likely help reduce the risk of weathering. Since the depth sensor is barely protruding from the box with the cutout piece on the O-rings, it was decided not to add such a cone.

In the selection of an ultrasonic depth sensor, 2 main factors were considered.

The first factor is the range of the depth sensor. Since the poles used are 5 m long, a range of 5-6 m was usually preferred in order to ensure the quality of measurements at and under 5 m. As a reference, cheap ultrasonic depth sensors like the HC-SR04 tend to go to 4 m.

The second factor is the temperature rating. Most depth sensors are rated to -20^{o}C due to the complexity of electronics on their board. As an example, the HC-SR04 uses custom chips to process the echo signal. This means that reverse-engineering a depth sensor is likely

a tedious task.

Unfortunately, no depth sensor was able to meet all of these requirements for a reasonable price. The cheapest depth sensor that complies with all of the above is the XL-MAxSonar from MaxBotix (see Digikey 1863-1008-ND) for the price of $150CA.

A step down from this is the LV-MaxSonar-EZ from MaxBotix (see Digikey 1863-1002-ND) for the price of C$45. This one is more prone to weathering.

Due to the complexity of creating our own ultrasonic depth sensor and the steep price for viable ones, it was decided to abandon the idea of a perfect sensor for the first iteration of the BRACHIOSAURUS. The results that will be taken on the first deployment will help set priorities for the development of a proper depth sensor. As an example, if the sensor dies right away regardless of the temperature, we will know that a closed transceiver is mandatory. If the results start being extremely inconsistent below a certain temperature then we will know that the temperature rating is extremely important.

As a side note, the beam angle of the sensor was also studied as it could have been an important factor to consider. It was found that, if the sensor is next to a 1" pole (like the one used in the Arctic), the echo reflected off of it was not strong enough to be perceived by the depth sensor. This means that even with a 180^o beam angle it should still be possible to ignore the pole being in the way.

Another type of direct distance measurement was considered along the way, that being a laser depth sensor. The issue with laser depth sensors is that they are mostly used in low-light environments (not outdoors). Outdoors laser depth sensors require a special filter and more power which increases the price to roughly $40CA. Another point to consider is that Arctic glaciers are not nice surfaces, they have rough snow that could scatter the laser's light while also reflecting a lot of light from the Sun to the laser's receiver. This would make it even harder for the laser sensor to take meaningful measurements. Considering these points, the idea of a laser depth sensor was abandoned.

As yet another side note, the inclination of the depth sensor does not influence the distance reading (this was tested in the lab). One can think of the depth sensor as emitting a sound wave that propagates radially in the air. When in comes into contact with the ground, this bounces back, again, radially. This means that the distance measured is always the radial distance to the closes object. Hence, the angle at which the depth sensor is inclined has no influence on the ground to sensor distance being read.

Switch

The switch ($2CA) is the simplest component on the board. It can take 12 V and 500 mA which is more than enough for the BRACHIOSAURUS as it links the batteries to the microcontroller which runs at 4.5 V and 250 mA at its maximum power when communicating via WiFi.

The switch has 3 pins; common, PIN1 and PIN2. Flipping the switch alternates contact between common-PIN1 and common-PIN2. Common is linked to the ESP32 while PIN1 is connected to the batteries and PIN2 is connected to nothing (therefore providing the ESP32 with no energy).

Battery holder, batteries, and energy

The battery holders ($3CA) used are designed to receive 3 AA batteries each. While experimenting with the setup, it was found that open-top battery holders were quite prone to spilling their batteries. Therefore, the battery holders used in the BRACHIOSAURUS have a lid to close them. The lid can be pushed into position and secured with a screw (though the screw is not mandatory).

The total time the BRACHIOSAURUS can survive on 6 AA batteries can be computed using the following method:

The best AA batteries to use in Arctic conditions are lithium batteries. Assuming very bad year-long conditions, the energy stored in one AA battery can be extrapolated to be

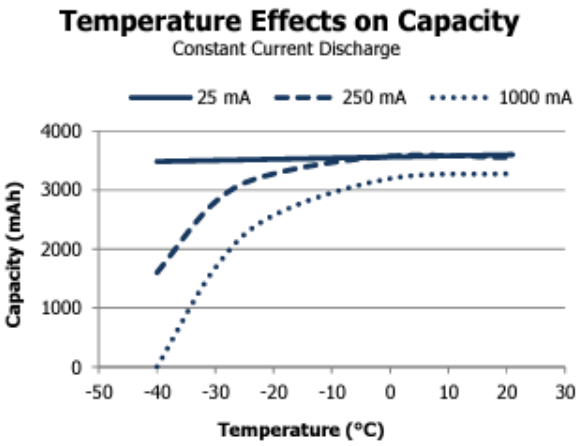


Figure 1: Energy stored in an ENERGIZER L91 lithium AA battery as a function of temperature.

3000 mAh (referring to Figure 1) when drawing current to take data at 60 mA (this is more than the 45 mA measured in the lab). This makes the total available energy 18,000 mAh. The low-power mode the ESP32 enters when going to sleep is 30 μA (this is also more than the 20 μA measured in the lab). The ESP32 wakes up 2 minutes 6 times per day (more than the 1 minute and 50 seconds observed in the lab) and goes to sleep the rest of the time.

Now a bit of math: the wake-up energy per day is 60 mA for 2/60 hours 6 times per day for a great total of 12 mAh. The sleep energy per day is simply 0.03 mA for 24 hours for a total of 0.72 mAh. For simplicity, assume the total to be 13 mAh. This means that, on 13 mAh per day for a capacity of 18,000 mAh, the BRACHIOSAURUS can be expected to survive 1384 days or 3.8 years without human interaction.
This means that, under somewhat pessimistic conditions, the BRACHIOSAURUS should be able to survive more than 3 years without human interaction.

PCB layout

The layout of the PCB was made to reduce the size as much as possible while allowing for all the components to be easily swapped and tested. This section serves mostly as an explanation of the board and comments for future versions of the PCB.

Note that all the connections are made on the front copper layer while the entirety of the back copper layer serves as ground.

At the top left, U2 is the footprint for the DFR0654 ESP32 microcontroller. The USB-C connector to modify code on the microcontroller points towards the left as to allow easy access once the PCB is removed from the box.

U4 located at the top right is the footprint for the micro SD card mount imbedded on a board. This was the only version of the micro SD card mount that could be tested before ordering the PCBs, so it stands there as a relic of uncertain times.

Inside the U4 footprint is the U5 footprint for the vertical micro SD card mount. This mount has since been tested and works as expected which means that the U4 footprint is now rendered useless.

Under the microcontroller, the switch footprint can be seen with ON and OFF next to it. Unfortunately, the switch on this board is slightly more confusing than anticipated. When the orange slider on the switch points to ON, it is off. When it points to OFF, it is on. This will need to be corrected in future iterations of the PCB.

Right next to the switch can be found the BT1 and BT2 footprints for the connection with the wires from the batteries. The + and - signs indicate clearly the ordering of the wires that can be soldered directly in the through holes. Enough clearance was also given so that the wires could be put in small Molex connectors to make it easier to plug and unplug.

To the right of the BT footprints, the U3 footprint stands for the accelerometer footprint. Unfortunately, the through holes were made too small for standard male header pins which means that female header pins need to be connected first and then the accelerometer with male header pins can slide in.

To the right of the U3 footprint is the diagram with the BRACHIOSAURUS logo, the creator's name, date of PCB finalization, version of the BRACHIOSAURUS PCB, and lab of origin.

In the center of the PCB, two large footprints represent the space given to the battery holders. Preferably, the closed-top battery holders would slide open pointing down as this would give them the most space.

At the bottom left, the U1 footprint stands for the ultrasonic depth sensor HC-SR04 footprint. Even though the transceivers can be seen to go out of the PCB, they do not go

out enough to be able to go through the wall of the aluminum box. For this reason, male pins are soldered to the sensor's footprint and the connection is made via a 4-pin female to female connector. This way, the depth sensor is free to go as far as it can through the wall of the box.

Four holes marked H1, H2, H3, and H4 are present near the corners of the PCB, they allow mounts to be screwed to the PCB.

In the future, it would be possible to include a battery voltage reader. This is attained with a simple voltage divider circuit. Take two resistors and place them in series going from the system voltage to ground. Wire the point between the two resistors to an analog pin on the microcontroller. The voltage read is half the total voltage of the batteries.

The different peripherals could be more reliably powered on using MOSFETs. The source of the MOSFETs is wired to 3.3V and the gate is wired to a digital output pin. This would make it so that different periphrals can be powered ON on command, allowing for greater versatility. The voltage source from 3.3V is also much more stable than the one from digital pins which means that it is less likely that the voltage required to power the peripherals will be insufficient.

While the current method used for time keeping is fine, using an external real time clock could be more reliable over large periods of time.

Watchdogs were also considered at one point in the design. Watchdogs are used in most electronics that need to do a certain task for long periods of time without human interactions. Unfortunately, most readily available watchdogs have a maximal time limit that is smaller than an hour, meaning that the microcontroller would need to wake up at intervals smaller than an hour to send a trigger to the watchdog. The idea was abandoned because the test boxes were capable of surviving weeks without failing.

The space taken on the PCB could be further reduced. As it was mentioned earlier, re-

ducing the microcontroller and accelerometer to a chip would free up a lot of space. Now that the vertical micro SD card mount has been tested, there is no need for the U4 component to take up more space.

If the pole frequency reveals itself to be a good standalone way to measure the length of the pole, the depth sensor would no longer be required.

Removing one of the battery boxes would also free up a lot of space. The lifespan of BRACHIOSAURUS would be reduced to 1.9 years which is perfect if the sites are visited yearly. The sampling rate could also be reduced in order to make the lifespan longer. A bit of time was also spend thinking about the addition of solar panels to allow for better longevity. It was realized that the addition of solar panels would not add a lot of value as the current setup already allows for good enough longevity. The addition of a solar panel would also come with its own set of issues like needing to waterproof one more hole, finding good rechargeable batteries, adding components on the PCB to handle the batteries recharging, etc. All of these issues would have taken unnecessary time out of the project.

Small solar panel cells could be added to the top of the box (their wires going inside) to help power and perhaps recharge the batteries during the summer, though this was not tested in the lab as the battery capacity was already sufficient.

PCB electrical connections

The entirety of the electrical connections can be seen in the KiCad desing foud here: `https://github.com/FelixStA52/BRACHIOSAURUS`. Some notable connections can still be noted.

The micro SD card mount requires the most connections to the microcontroller out of all the components. To connect it, we need to free the MISO pin, MOSI pin, SCK pin (clock), and one digital pin on the microcontroller.

The accelerometer only requires connections to SCL (clock) and SDA (data) pins on the microcontroller.

The depth sensor requires two digital pins. One of them is used to send a signal to the sensor to trigger the sound wave. The other digital pin is activated by the depth sensor when it receives an echo.

The most interesting part of the wiring is the following: all the peripherals are powered by a total of 5 digital pins. The reason for this is that the 3.3V output of the microcontroller is always powered on, even in low-power mode. This means that, in deep sleep, the peripherals would be powered on regardless. This would draw approximately 5-10 mA all the time which would drain the batteries in a matter of days.

Therefore, the peripherals are powered by digital pins that are pulled to HIGH (brought to 3.3V output) when the microcontroller wakes up and pulled to LOW (0V) when the microcontroller goes to sleep. A total of 5 pins are used since each pin has a limit on the current it can output. It was found that the micro SD card requires at least 2 pins to consistently write data (the depth sensor and accelerometer require 1 each). To be on the safe side, 5 pins are used to power all of the peripherals.

It might be interesting to note that, in a previous version of the BRACHIOSAURUS, jumper wires were used instead of a PCB and the components were velcroed to the box. Other than being ugly and messy, it made the connections much more prone to breaking as the wires could easily disconnect. Soldering wires directly could also be a cheaper option, but it would become harder to replace the different components.

4 Pole Physics

Theoretical background

Imagine the pole to be a simple damped mass-spring oscillator: the mass m_{tot} is a point mass located at the center of mass of the exposed pole while the spring constant k comes from the stiffness of the pole and the damping factor c from energy loss in the oscillation of the system. Per usual, the motion of the mass goes like

$$x(t) = Ae^{\zeta\omega_0 t}\sin(\sqrt{1-\zeta^2}\omega_0 t), \tag{1}$$

where

$$\zeta = \frac{c}{2m_{\text{tot}}\omega_0}, \tag{2}$$

and

$$\omega_0 = \sqrt{\frac{k}{m_{\text{tot}}}}. \tag{3}$$

Everything multiplying the time inside the sine term is the frequency f of oscillation of the pole. This can be simplified to

$$f = \sqrt{\frac{k}{m_{\text{tot}}} - (\frac{c}{2m_{\text{tot}}})^2}. \tag{4}$$

In this equation, it is important to note that the total mass for the center of mass consists of $m_{\text{elctr}} + m_{\text{pole}}$ where m_{pole} is the mass of the pole and m_{elctr} is the mass of the electronics.

One can imagine that, while the spring constant is constant in time, it is not so constant in relation to the length of the pole. In fact, the spring constant becomes a lot greater the shorter the pole is. This relates to the spring constant derived from the cantilever beam formula where

$$k = \frac{3EI}{L_{\text{cm}}^3}. \tag{5}$$

In this equation, E is Young's modulus, a property of the material of the pole, and I is the second moment of area (a property dependent on the shape of the pole used). L_{cm} is the length from the ground to the center of mass of the pole.

Talking about the center of mass, it can be found via simple mechanics derivations to be

$$L_{\text{cm}} = \frac{m_{\text{elctr}} \cdot L + 0.5 \cdot \sigma \cdot L^2}{m_{\text{elctr}} + \sigma \cdot L}. \tag{6}$$

In this equation, σ is the linear density of the pole (total mass of the pole / total length of the pole) and L is the length from the ground to the top of the pole (the exposed length of the pole).

Combining Equations (4), (5), and (6), we obtain the final equation of the frequency of the pole as a function of exposed pole length

$$f(L) = \sqrt{\frac{3EI(m_{\text{elctr}} + \sigma L)^2}{(m_{\text{elctr}} L + 0.5 \cdot \sigma L^2)^3} - (\frac{c}{2(m_{\text{elctr}} + \sigma L)})^2} \tag{7}$$

Equation (7) shows the frequency for a pole that is disturbed from equilibrium and then released without other forces acting on it, which is not the case for a pole in the wind. Luckily, the pole in the wind will be hit by a scattering of frequencies, this means that the frequencies that are not the resonant frequencies will be damped while the resonant frequency will be enhanced. Therefore, the motion of the pole under a Fourier transform will exhibit a maximum at the frequency presented in Equation (7).

Experimental results

In order to test the formula derived in the previous section, a simple experiment was created. There are only two unknowns in this experiment, the product of EI and the damping factor c; the rest can be directly measured with a scale. The mass of the box (with batteries) is of 0.638 kg and the linear mass of the pole is 0.533 kg/m. These can be directly inserted in the formula when plotting the best fit curve shown in Figure 2.

The experiment is the following: a clamp is securely anchored to a table. The pole is held by the clamp. The box with the electronics is attached to the top of the pole in the

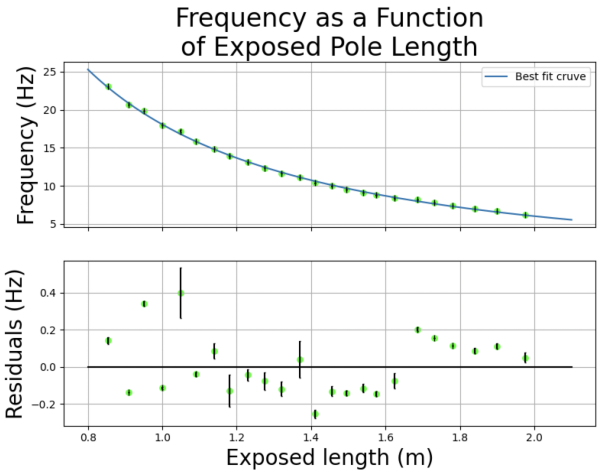


Figure 2: Example of a plot of the frequency as a function of length. For this given experiment, an aluminum pole of 1" diameter with 1/8" thick walls was used.

same way as it is attached to the pole on the glacier. The distance between the top of the pole and the clamp is then measured with a measuring tape. This distance is taken to be the exposed pole length. A small disturbance is then introduced via knocking on the pole which makes the pole vibrate at its resonant frequency. This frequency is then measured by BRACHIOSAURUS. This process is then repeated to obtain a plot of the frequency as a function of exposed pole length.

From these points, a best fit curve can be taken while imposing EI and c as unknown constants. This gives $EI = 60.5\ Nm^2$ and $c = 3.73\ \frac{Ns}{m}$, as seen in Figure 2.

Since the formula is not invertible (it is not possible to obtain a closed form of the length as a function of frequency), a root finding algorithm is used to find the length for each frequency. This gives measured lengths from the frequency with uncertainties of $\pm$ 3 cm, as seen in Figure 3.

As a side note on the factor EI, it is possible to derive it from the properties of the material, that is a pole of aluminum T6061 with 1" diameter and 0.125" thick wall. While it could have been derived, using it as an unknown simplifies the calibrations and incorporates material defects that would stray the result from the theoretical EI factor.

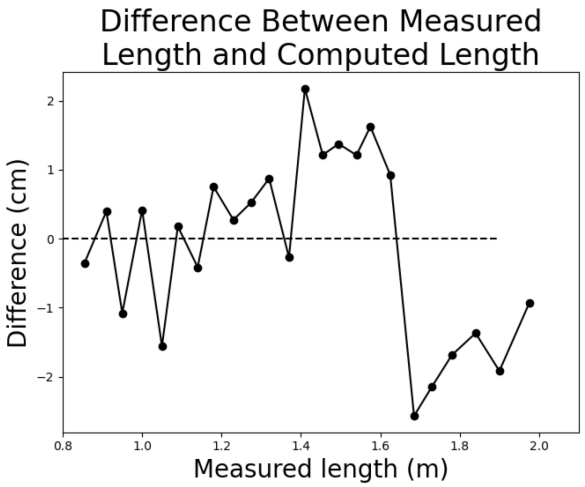


Figure 3: Plot presenting the difference between the length of the pole measured in the lab and the length of the pole determined by inverting the frequency v. length formula.

Another point that needs to be touched on is the accumulation of rime on the pole and BRACHIOSAURUS. This has the effect of changing the mass of the pole by a significant amount (the center of mass is probably altered at a much smaller scale than the mass) which leads to the formulas presented previously being wrong. Unfortunately, the code created for this experiment does not have a way to decipher between a sudden drop in ice levels and a sudden accumulation of rime. Deciphering between the two will be left as an exercise to glaciologists.

Recommendations for future poles

As seen in the previous section, it was found that aluminum poles have a fairly flat frequency v. length graph at the maximal length of the pole. Keeping in mind that the resolution of the frequency found from the acceleration is only of a tenth of a Hertz, differentiating between two lengths of the pole in this frequency region can prove to be quite difficult. If the pole was thicker, or made out of a sturdier material than aluminum, the decreasing profile of the frequency v. length graph would continue well into the longer ranges of the pole. In conclusion, a sturdier pole would allow for a better length resolution when the pole is almost entirely out of the ice.

5 Code explanation

How to code on ESP32

The ESP32, like many other microcontrollers, can be programmed via the Arduino IDE. While the Arduino IDE 2 was used to program the microcontroller used in the BRACHIOSAURUS, the Arduino IDE 1 could also have been used. Since the DFR0654 ESP32 is not an Arduino microcontroller, some boards need to be imported via the "Board Manager" present on the IDE. All the steps to get started writing code on the DFR0654 ESP32 can be found on the product Wiki of the ESP32: `https://wiki.dfrobot.com/FireBeetle_Board_ESP32_E_SKU_DFR0654`.

Data taking code (Arduino code)

The following section is built as a general algorithm for how the code works. The code in its entirety can be found in the following GitHub page: `https://github.com/FelixStA52/BRACHIOSAURUS`.

```
void setup() {
    Start timing the code
    Power the digital pins for the peripherals
    Wait 1000 miliseconds to allow the peripherals to wake up

    Begin SCL and SDA communication

    for (set amount of loops per wake-up){
    if (there is a working SCL SDA communication){
        Get the instantaneous acceleration and temperature from the accelerometer
        Use the accRoutine function to obtain the main frequencies of oscillation
    }
    Use the ultraRoutine (routine for the ultrasonic depth sensor)
    function to obtain the distance measured
```

```
    if (there is an SD card){
        if (there is a working connection to the SD card){
        Open .csv file on the SD card
        Write the data taken before
        }
    }
    }
    Adjust the counter for the number of iterations
    Stop timing the code
    Compute the time elapsed for the code execution
    Adjust the time saved on the microcontroller
    Go to sleep for a set amount of time
}
```

Note that the void loop() we usually see with arduinos is not present. This is because putting the ESP32 to sleep makes it wake up at void setup(). Therefore, the loop is handled within setup and void loop() is never reached.

Here is the explanation of some of the custom functions used:

```
double ultraRoutine(){
    Activate the trigger pin on the depth sensor
    Wait for the pulse to come back
    Compute distance from the time between trigger and pulse
    Return distance
}
```

```
double accRoutine(){
    for (30 iterations){
    Take bogus data on the accelerometer, these points are not saved to anything
```

```
    (it was found that the first 10-20 datapoints measured go
    like a dacaying exponential,giving weird results when taking the FFT)
    }

    Start a timer
    for (the amount of samples needed, needs to be a power of 2 i.e. 2^n){
        Get the instantaneous acceleration
        Add the squares of each component of the acceleration
        to get a sort of amplitude
        Wait 4 miliseconds (this leads to sampling frequencies of ~200Hz)
    }
    Stop the timer

    //this part of the code might not be present on all the microcontrollers
    if (there is an SD card){
        if (there is a working connection to the SD card){
        Open a .bin file on the SD card
        Write the current time
        Write the array containing all the accelerations taken previously
        }
    }

    From the timer, compute the average sampling rate
    Perform the FFT on the data given the sampling rate
    Use the custom findMax function to find the main maxima
    Return the main frequencies of oscillation
}

float findMax(data, size of the data array){
    for (size of the data array){
```

```
        if (this data is the biggest one so far and it is a local max){
        Remember this data as being the biggest
        }
    }

    for (size of the data array){
        if (the datapoint is far enough from the maximal value){
            Add this value to a total sum
        }
    }

    Take the average of the total sum, this is the noise
    if (the biggest amplitude is not big enough compared to the noise){
        Return the maximal frequency as being 0 Hz
    }
    Return the maximal frequency found
}
```

Data unpacking code (Python code)

This one is quite a bit longer, so I might brush past some less important things.

```
    Look at all the files present in the same folder as the python code

    Go over all these files, find the ones labeled data_[box number].csv
    as well as the ones labeled motion_[box number].bin

    Create a list with all the unique box numbers

    def f(l, k, c, m=0.638, s=0.533):
        This is the function for the equation of motion of the pole
```

```
    It returns the frequency as a function of length
    Refer to the Pole Physics section to see the formula

def zero_fffunc(l, freq_solve, k=182.64026108, c=3.91926779):
    This is the function used to find the length as a
    function of frequency
    This of it as an inverse of the frequency as a function
    of length function (though it cannot be inverted)
    The values given for k and c are ones found while
    experimenting in the lab. Refer to the Pole Physics section
    to see the formula

for the number of boxes:
    Get the data for this box

    From the data as well as the motion, get a list of all
    the unique times during which there were measurements

    for all the unique times found:
        Get the data at a specific time

        for all the data sets:
            Take the ratio of the 2 frequencies; they
            should be harmonics of one another, hence the
            ratio should be 2 or 0.5

            Divide the highest frequency by 2 to bring it to
            the same level as the lowest frequency
            Average the two frequencies
```

```
if we have enough viable data sets:
    Find the frequency slightly before the median
    frequency, this should be the lowest harmonic

    Again, look at the ratios to what is though to
    be the lowest harmonic

    if the ratio is not 2 or 0.5:
        Delete the data sample

    if the frequency is twice as high:
        Bring it down to the lowest harmonic

Now that we have a set of fundamental frequencies
compute the length for each individual frequency
using the root finding fucntion

Save the length as well as the error to the
final data file

Take the FFT of the data in the motion file

Do the same peak selection as on the ESP32

Do the same frequency selection as for the data.csv file

Save the data to the final data file
```