MAE 223 FINAL PROJECT REPORT

Team Name: Stokes and the Drifters
Andrew Lee, Daniel Nijenhuis, Eliana Arroyo-Fang, Isaac Need, Nandan Seshadri

Introduction

Our goal was to build a lagrangian drifter to survey ocean surface conditions. We were interested in characterizing wave heights, surface waves frequencies, and the general direction of surface flow. By measuring and quantifying these conditions, we aimed to build an understanding of the surface conditions using a high temporal resolution. This was the base functionality of the drifter, however we had more design requirements.

It was imperative that the drifter be operable by one person, so we aimed for a 5 kg weight limit for ease of deployment. In addition, it needed to continuously send its GPS-location so that the drifter may be retrieved in the event that it floated out of view. Lastly the drifter needed to be sufficiently robust in order to survive the nearshore wave conditions and remain watertight for a minimum of a couple of hours. The resulting drifter weighed 3 kg, uploaded GPS locations to thingsboard.io and dweet.io, and was completely watertight. It sampled acceleration in the X-, Y- and Z-directions and rotation around the same three axes through the use of an IMU. Unfortunately, due to the drifter's natural frequency being 1 Hz we ran into issues as it interfered significantly with the collected data.

Table 1: Functional Requirements

Requirement	**Specification**
Operable by one person	Total mass less than 5 kg
Watertight	Interior devoid of moisture after 2hrs drifting
Caps must be permeable to GPS signal	Use neoprene rubber
Sensitive to the frequencies expected in the ocean, 1 Hz to 0.05 Hz	Physically limited cutoff frequency and sampling Nyquist frequency >2Hz

System Overview

The electronics system comprised two modules, the IMU module and the GPS module, which were implemented through the use of an Arduino Mega 2560. The IMU module consisted of a 9 degree of freedom IMU (LCM20600 + AK09918) that read acceleration data and gyroscope data and wrote it to an SD card. The GPS module utilized a Botletics SIM7000 Shield, equipped with a dual LTE/GPS antenna, that sent its geographical coordinates to the cloud via a dweet (shown in Appendix A.1) as well as posted it to thingsboard.io (shown in Appendix A.2). The Arduino Mega 2560 was powered by a 9V battery and the Botletics Shield was powered by a 3.7 LiPo battery.

During initial testing, the modules were run independently using sample sketches and each performed as expected. However, when it was time to test the sample sketches on the integrated hardware system, issues arose. In order to resolve the pin conflict between the transmit pins of the GPS shield and the SD card chipSelect pin, we rerouted both transmit pins to pins 52 and 53 using cable connectors. This seemingly fixed the problem and enabled the GPS and the IMU sample sketches to run. Unfortunately, when it came time to run our final code that integrated everything into one sketch, the code got stuck. Upon further investigation of the sample sketches we took parts from and the arduino libraries they implemented, we realized there were more pins that each module implicitly used. After discovering that the problem was another pin conflict with pins 52 and 53, we simply moved the transmit pins to be pins 68 and 69 instead. This resolved everything and the code ran as expected.

The pseudocode for the Arduino script we used to operate our hardware is shown in Figure 1 (raw code can be found in the Appendix A.3). At the start of our script, we initialized all our variables. Then, within void setup(), we initiated all our modules. This included turning on and calibrating the IMU, initializing the data file on the SD card along with writing the header, checking the real-time clock, and setting up the Botletics Shield. Lastly in the void setup(), we set the value of the startMillis variable equal to the milliseconds since the start of the program. Although, after the first loop, the startMillis variable was used to keep track of the time when each sampling period began. At the beginning of our void loop(), an if-statement would only execute the rest of the loop when the difference between the current time and startMillis was greater than the sum of sampling lag and sampling period (30 seconds + 1 minute). The startMillis variable was then used in a while-loop to sample and write IMU data for our defined sampling period (1 minute). GPS data was collected and posted to thingsboard.io and dweet.io after each sampling period (~every 1.5 minutes) and afterwards a delay was set. Though the delay was not necessary because of the if-statement that began the void loop(), it was implemented for two reasons: to make sure that the program was looping as the delay time was printed to the serial monitor and to save battery by lowering computation between sampling periods.

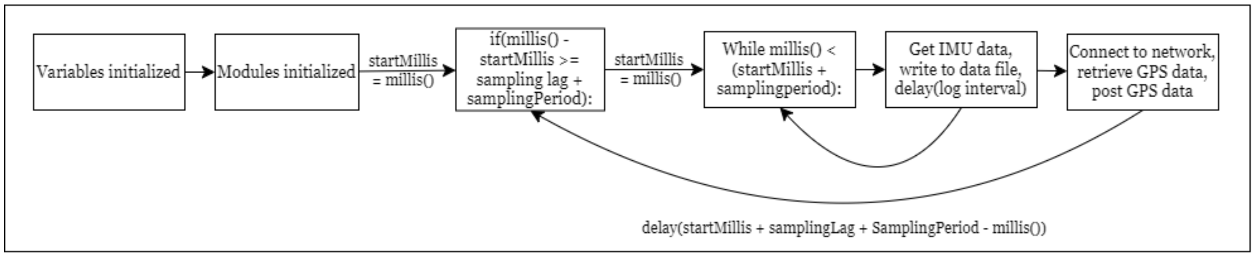

Variables initialized
Modules initialized
startMillis = millis()
if(millis() - startMillis >= sampling lag + samplingPeriod):
startMillis = millis()
While millis() < (startMillis + samplingperiod):
Get IMU data, write to data file, delay(log interval)
Connect to network, retrieve GPS data, post GPS data
delay(startMillis + samplingLag + SamplingPeriod - millis())

Figure 1: Arduino Code Flowchart

Mechanical design

The final design of the drifter was constructed using a sturdy PVC pipe, measuring 12 inches long with a 3.5 inch diameter. To ensure its water resistance, drain pipe caps were fitted at both ends of the pipe using hose clamps. Surlyn foam, 4 inches long and 8.5 inches in diameter, was used to provide reliable flotation. Within the PVC pipe, a 3D printed electronic enclosure made with PLA was used to prevent the electronics from being jostled. To maintain stability in varying sea conditions and be self-righting, a 800 gram ballast was integrated. A piece of glass was stuck at the bottom of the PVC pipe with epoxy to enhance water resistance as a fail proof for the bottom drain pipe cap that is submerged in water. This design was robust, enabling the wave drifter to gather oceanographic data while surviving harsh environments. Its durability was proven when, after failing to acquire data the day of the Scripps Pier testing, a team member swam the drifter out past the surf zone and returned the drifter an hour later with everything fully intact.

Before settling on the final design for the electronics enclosure, we phased through four iterations (Appendix B.1). During the first iteration (Appendix B.1 far left), we tested the strength of PLA for this application while trying to fit all the electronics within a 5 inch diameter container. After realizing the price of PVC pipe raised significantly with increasing diameter, we decided to shrink the enclosure down to a diameter 3 inches. The next constraint we took into consideration was ensuring a safe distance between the microprocessor and the batteries. We adopted a separate chamber to house the batteries that powered the Arduino board and the Botletics Shield. We initially used a screw-thread mechanism (Appendix B.2) to attach the chamber to the rest of the enclosure, however we found that it could be problematic due to the connectors twisting during assembly. We changed the design to use a combination of snap and twist mechanism (Appendix B.3) to resolve this problem. After a series of dimensional changes, we settled with the design seen in Figure 2 and in Appendix B.4.

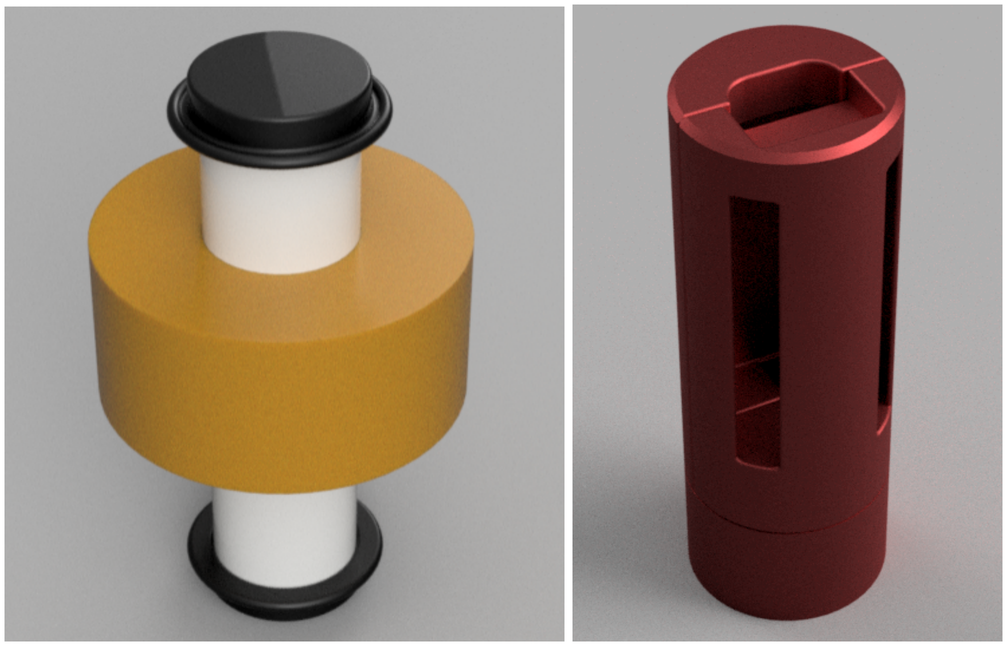


Figure 2: Left - CAD model of the wave drifter; Right - Final CAD of the electronic enclosure

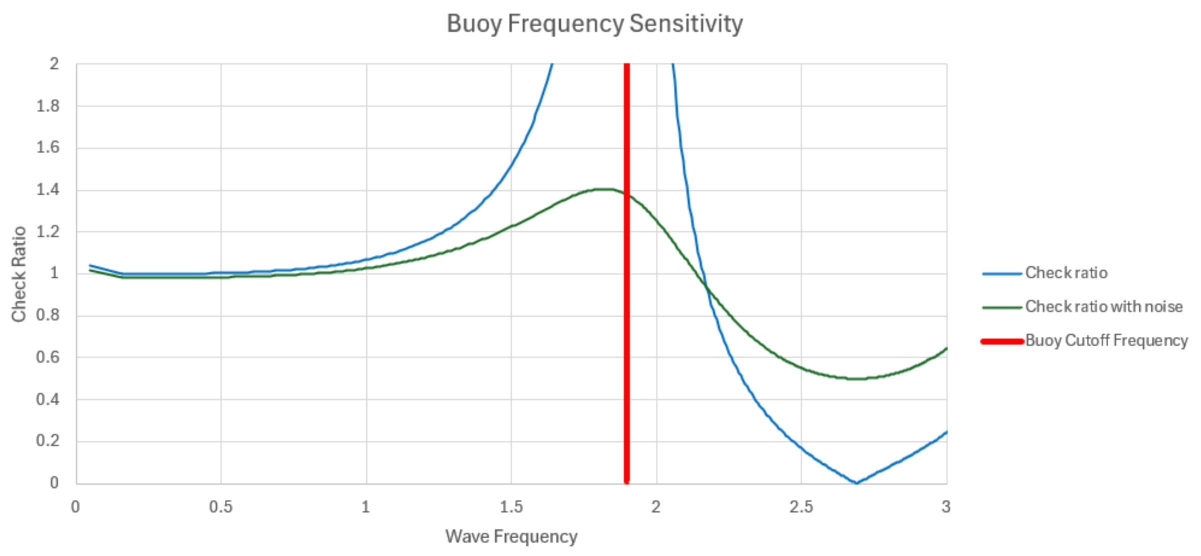


Figure 3: Frequency sensitivity based on slope aliasing check ratio

The final factor incorporated in the design of the drifter was the spatial sensitivity. The spatial sensitivity was calculated using the sensitivity check ratio developed by Yurovsky and Dulov [1]. As seen in Figure 3, the drifter had good response to frequencies below a frequency of about 1.9Hz. This was ideal because it was well above the expected maximum frequency of 1Hz for the types of wave we intended to measure.

Results

The core component of our data collection was the accelerometer in the IMU. In order to validate its readings, we executed a pendulum test which had one effective degree of freedom. We first collected data from our accelerometer and from a more accurate accelerometer provided by Drew. We then compared the Fourier transforms of the data sets to each other and the predicted natural frequency of the pendulum derived from first principles. As seen in Figure 4, the natural frequency aligned well between both accelerometers and the idealized physics prediction. The large peak at 2 Hz was not a concern because it was at a much higher frequency than what we expected to measure and it was above the highest frequency that the device would be sensitive to.

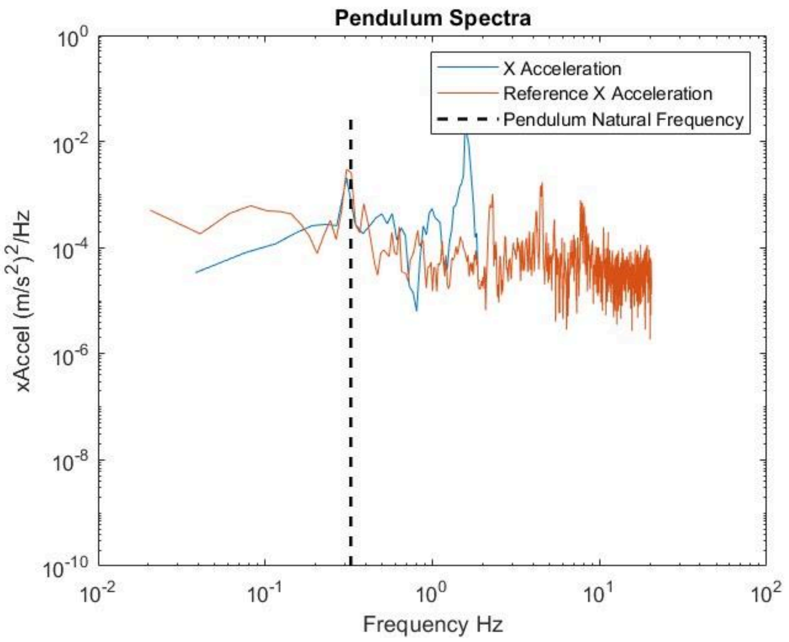


Figure 4: Fourier transform of reference accelerometer and device accelerometer

The next step in the accelerometer and data processing validation was a water channel test at the Scripps Hydraulics Lab. We used the clear wave channel and drew a reference scale on it to measure the wave height and provide a reference to verify frequency. The drifter was placed in the channel and subjected to tapered wave packets of the parameters shown in Table 2.

Table 2: Wave Parameters

Wave Frequency (Hz)	Peak-Valley Wave height (in)
1	4
0.5	4.5
0.25	4

In observing this experiment, it was immediately apparent that our drifter had a natural frequency of 1Hz. This mode experienced massive excitation when exposed to the 1Hz waves as well as significant excitation when exposed to the 0.5Hz and 0.25Hz waves.

Figure 5 shows the Fourier transform of data collected during a wave packet of 0.5 Hz waves which illustrates the problem very clearly. The wave energy excited an internal oscillation at 1 Hz that had an order of magnitude more energy than the waves generated by the wave tank.

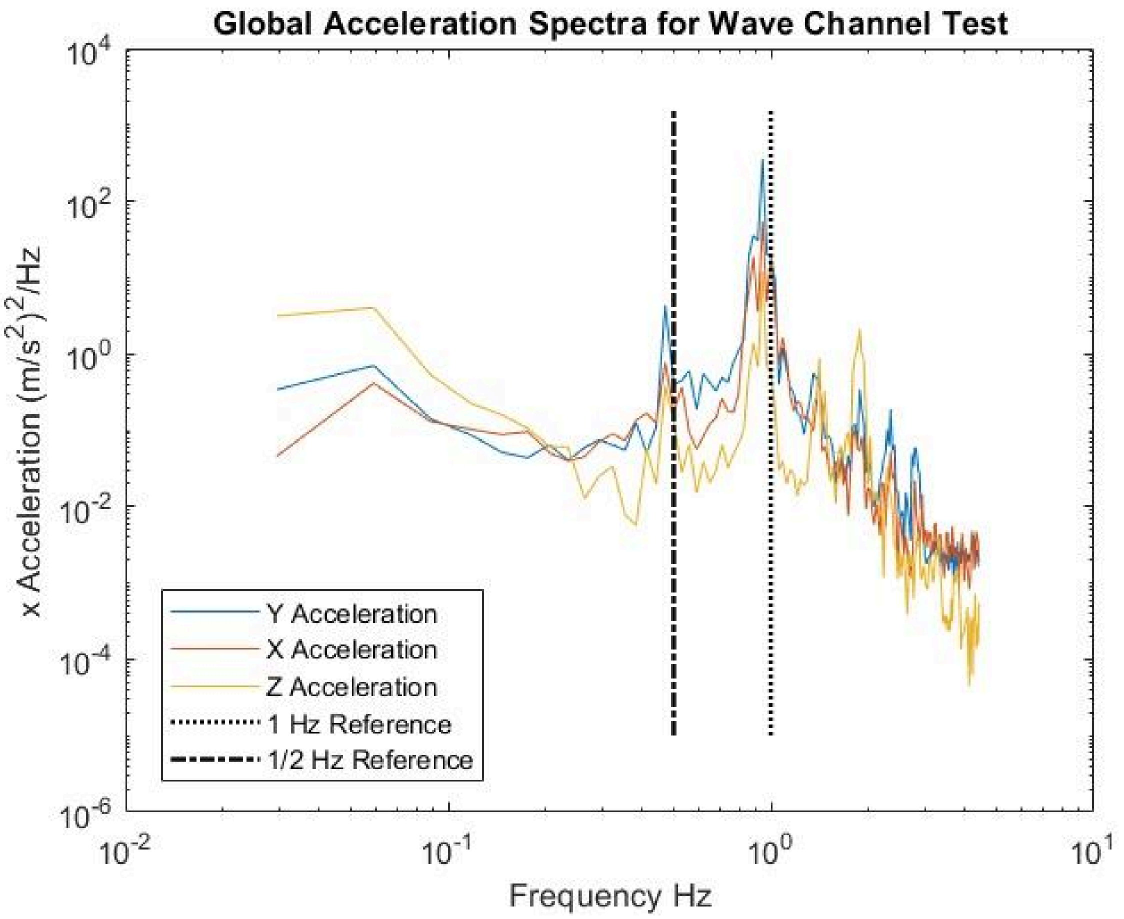


Figure 5: Fourier transform of 2 second period wave

After careful consideration we decided not to attempt to change the natural frequency of our drifter for two reasons: we would not have access to the wave channel again before our field test and the method to increase the natural frequency was not readily apparent without further analysis.

The Fourier transform seen in Figure 6 depicts the wave frequency measured at sea during our ocean trial. From left to right, there are three vertical reference lines: expected wave frequency, observed natural frequency, and expected natural frequency. The observed natural frequency was 0.65 Hz in both X and Y simultaneously. It was very interesting to note that the vector sum of these two frequencies was 0.92Hz which was nearly identical to the single degree of freedom natural frequency observed in the wave channel. The motion observed in the ocean was a 2 DOF oscillation about both X and Y instead of the single DOF pendulum action observed in the wave channel.

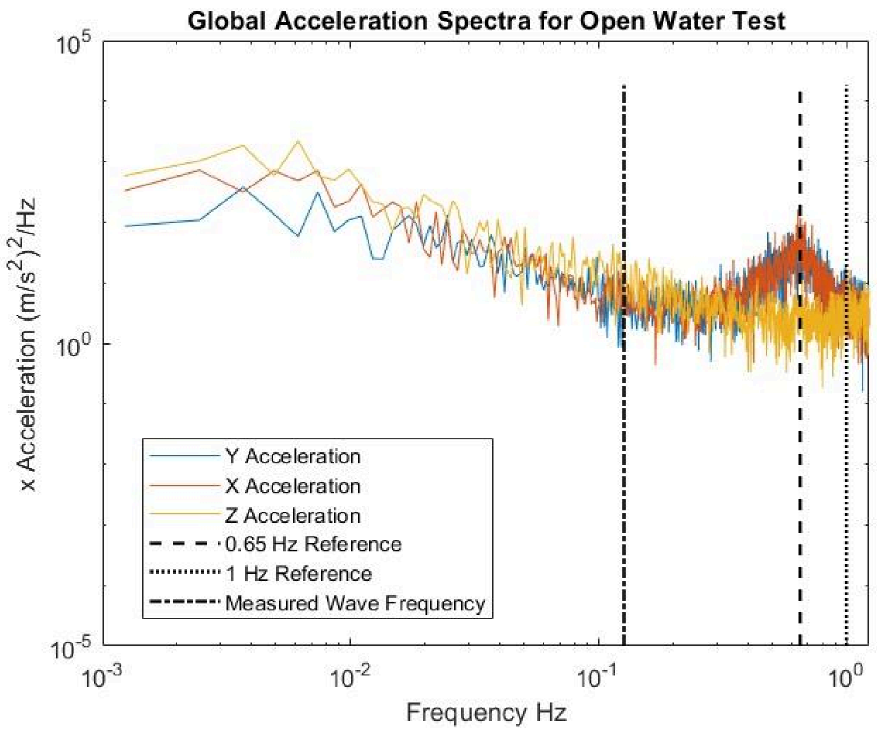


Figure 6: Fourier transform of orientation corrected accelerations of entire ocean test time series data

Conclusions

The biggest flaw in our project was related to the drifter design itself. The combination of the chosen PVC pipe length, the ballast weight, and the narrow foam collar used for floatation caused our complete assembly to behave like a pendulum in the wave tank and caused it to oscillate in a circle in the ocean. We observed the single DOF harmonic in the wave channel but failed to predict how that would compound in a multi directional wave field to create a lower natural frequency when inspecting each axis individually. This led to us not being able to extract meaningful wave frequency out of the data. In the future, we need to significantly increase the natural frequency of our design. This would eliminate the 1Hz peak which harmonized with the wave field.

Another lesson we learned while working on the software was to make the code easily modifiable. Our code was not flexible to using variables outside of values planned for the final test, so when we reduced the sampling period and sampling lag to expedite testing in the wave tank, the code failed at multiple points due to various unforeseen reasons and it called for extra troubleshooting.

While we were able to solve the code issue within minutes on the day of the water tank testing, we encountered another roadblock with the electronics hardware. When conducting our ocean test, the glue on the pins to our accelerometer failed and the module disconnected during assembly. This resulted in a failure to gather any data from the unit. However, Andrew carried out a second ocean test the following day and he made sure to use extra glue on all our connections. In order to avoid such a catastrophic failure in the future, more secure pin adaptors could be soldered in place to ensure that there are no loose connections.

Reference:

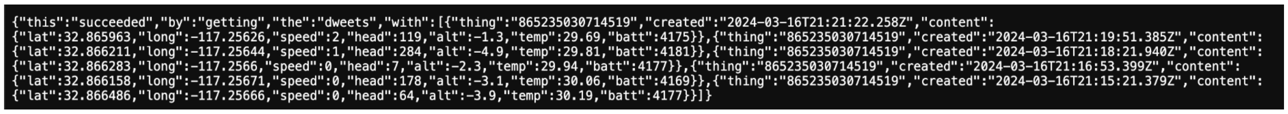
[1] Y. Y. Yurovsky and V. A. Dulov, "MEMS-based wave buoy: Towards short wind-wave sensing," *Ocean Engineering,* 2020.

Team Contributions

Team Member	**Project Contribution**	**Report Contribution**
Andrew Lee	Software	System overview
	Hardware	Conclusion
	Ocean Test 2	
Daniel Nijenhuis	Buoy construction	Introduction
	Enclosure brainstorm	Functional requirements
Eliana Arroyo-Fang	Software	System Overview
	Hardware	Revising & Editing
Isaac Need	Matlab data analysis	Results
	Enclosure brainstorming and fabrication	Conclusion
Nandan Seshadri	Enclosure design and fabrication	Mechanical design
	CAD modeling	
	Buoy construction	

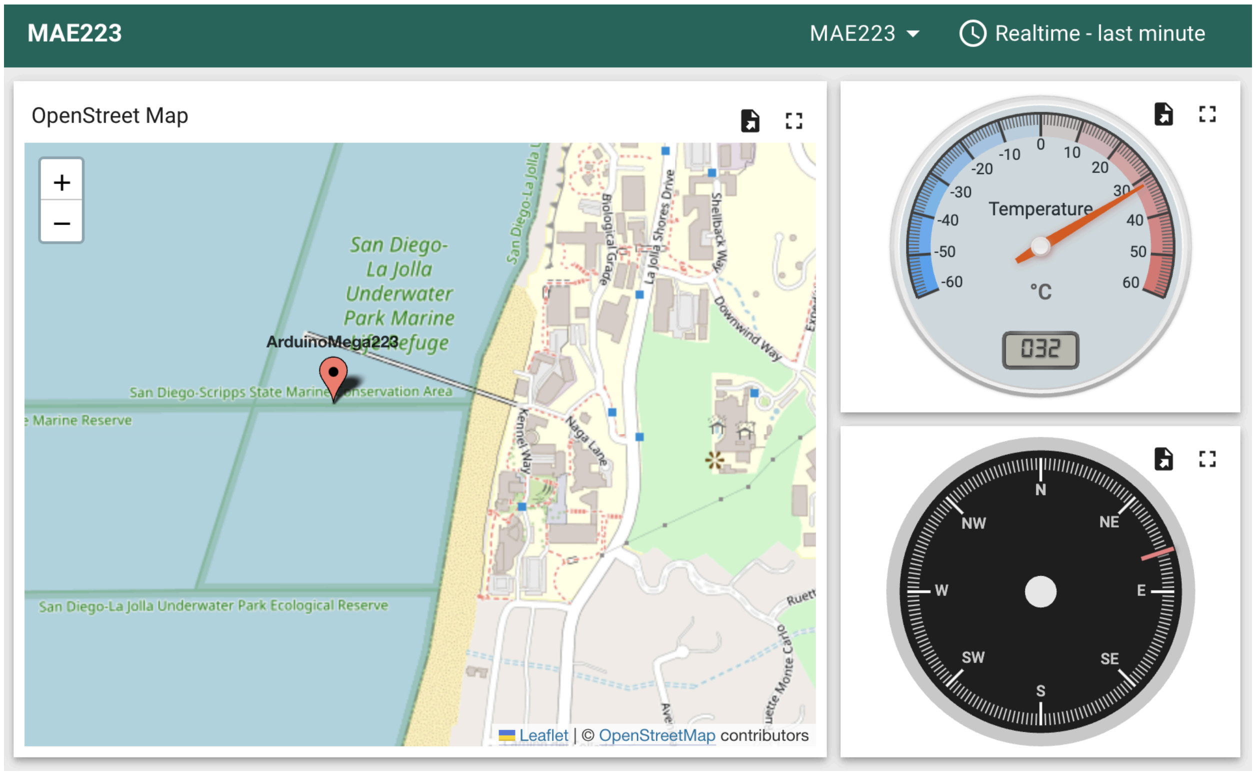
Appendix A - Code

A.1 Dweet screenshot



```
{"this":"succeeded","by":"getting","the":"dweets","with":[{"thing":"865235030714519","created":"2024-03-16T21:21:22.258Z","content":
{"lat":32.865963,"long":-117.25626,"speed":2,"head":119,"alt":-1.3,"temp":29.69,"batt":4175}},{"thing":"865235030714519","created":"2024-03-16T21:19:51.385Z","content":
{"lat":32.866211,"long":-117.25644,"speed":1,"head":284,"alt":-4.9,"temp":29.81,"batt":4181}},{"thing":"865235030714519","created":"2024-03-16T21:18:21.940Z","content":
{"lat":32.866283,"long":-117.2566,"speed":0,"head":7,"alt":-2.3,"temp":29.94,"batt":4177}},{"thing":"865235030714519","created":"2024-03-16T21:16:53.399Z","content":
{"lat":32.866158,"long":-117.25671,"speed":0,"head":178,"alt":-3.1,"temp":30.06,"batt":4169}},{"thing":"865235030714519","created":"2024-03-16T21:15:21.379Z","content":
{"lat":32.866486,"long":-117.25666,"speed":0,"head":64,"alt":-3.9,"temp":30.19,"batt":4177}}]}
```

A.2 Thingsboard.io Dashboard Display


MAE223
MAE223
Realtime - last minute
OpenStreet Map
+
−
San Diego-La Jolla Underwater Park Marine Refuge
ArduinoMega223
San Diego-Scripps State Marine Conservation Area
Marine Reserve
San Diego-La Jolla Underwater Park Ecological Reserve
Biological Grade
La Jolla Shores Drive
Shellback Way
Downwind Way
Kennel Way
Naga Lane
Leaflet | © OpenStreetMap contributors
Temperature
°C
032
N
NE
E
SE
S
SW
W
NW

A.3 Arduino code

```
#include "AK09918.h"
#include "ICM20600.h"
#include <Wire.h>
#include <SPI.h>
#include <SD.h>
#include "RTClib.h"

const int chipSelect = 10;

AK09918_err_type_t err;
int32_t x, y, z;
int32_t xoff, yoff, zoff;
AK09918 ak09918;
ICM20600 icm20600(true);
int16_t acc_x, acc_y, acc_z;
int16_t g_x, g_y, g_z;
int32_t offset_x, offset_y, offset_z;
double roll, pitch;
// Find the magnetic declination at your location
// http://www.magnetic-declination.com/
double declination_SD = 10.2;

String dataString;

String fileName = "";

unsigned long endLoop = 0;
unsigned long delayTime = 0;
unsigned long startMillis = 0;
unsigned long currentMillis = 0;
unsigned long nextStart = 0;
const unsigned long samplingPeriod = 1000UL * 60 * 1;
const unsigned long samplingLag = 1000UL * 30;
const unsigned long startLag = 1000UL * 2;
#define LOG_INTERVAL  250 // mills between entries (reduce to take more/faster data)

RTC_PCF8523 RTC; // define the Real Time Clock object

//-------Botletics variables-----------------------------
#include "BotleticsSIM7000.h" //
https://github.com/botletics/Botletics-SIM7000/tree/main/src

#if defined(ARDUINO_SAMD_ZERO) && defined(SERIAL_PORT_USBVIRTUAL)
  // Required for Serial on Zero based boards
  #define Serial SERIAL_PORT_USBVIRTUAL
#endif

#define SIMCOM_7000

#define PROTOCOL_HTTP_GET         // Generic
//#define PROTOCOL_HTTP_POST        // Generic

/************************* PIN DEFINITIONS *********************************/
// For botletics SIM7000 shield
#define PWRKEY 6
#define RST 7
#define RX 69 // Microcontroller RX // originally 10 // yellow wire in pin 11 to 50
#define TX 68 // Microcontroller TX // originally 11
```

```
#include <SoftwareSerial.h>
SoftwareSerial modemSS = SoftwareSerial(RX, TX); // originally (TX, RX)

SoftwareSerial *modemSerial = &modemSS;

// Use this for 2G modules
#ifdef SIMCOM_2G
  Botletics_modem modem = Botletics_modem(RST);

// Use this one for 3G modules
#elif defined(SIMCOM_3G)
  Botletics_modem_3G modem = Botletics_modem_3G(RST);

// Use this one for LTE CAT-M/NB-IoT modules (like SIM7000)
// Notice how we don't include the reset pin because it's reserved for emergencies on the
LTE module!
#elif defined(SIMCOM_7000) || defined(SIMCOM_7070) || defined(SIMCOM_7500) ||
defined(SIMCOM_7600)
Botletics_modem_LTE modem = Botletics_modem_LTE();
#endif

/****************************** OTHER STUFF ***************************************/
// For sleeping the AVR
#include <avr/sleep.h>
#include <avr/power.h>
// For temperature sensor
#include <Wire.h>
#include "Adafruit_MCP9808.h"

// Create the MCP9808 temperature sensor object
Adafruit_MCP9808 tempsensor = Adafruit_MCP9808();

#define samplingRate 1 // The time in between posts, in seconds

uint8_t readline(char *buff, uint8_t maxbuff, uint16_t timeout = 0);
char imei[16] = {0}; // Use this for device ID
uint8_t type;
uint16_t battLevel = 0; // Battery level (percentage)
float latitude, longitude, speed_kph, heading, altitude, second;
uint16_t year;
uint8_t month, day, hour, minute;
uint8_t counter = 0;
//char PIN[5] = "1234"; // SIM card PIN

char URL[200];  // Make sure this is long enough for your request URL
char body[100]; // Make sure this is long enough for POST body
char latBuff[12], longBuff[12], locBuff[50], speedBuff[12],
     headBuff[12], altBuff[12], tempBuff[12], battBuff[12];

void setup() {
  Serial.begin(9600);

  pinMode(chipSelect, OUTPUT);
  digitalWrite(chipSelect, HIGH);

  // join I2C bus (I2Cdev library doesn't do this automatically)
  Wire.begin();

  err = ak09918.initialize();
  icm20600.initialize();
  ak09918.switchMode(AK09918_POWER_DOWN);
  ak09918.switchMode(AK09918_CONTINUOUS_100HZ);

  err = ak09918.isDataReady();
  while (err != AK09918_ERR_OK) {
      Serial.println("Waiting Sensor");
      delay(100);
      err = ak09918.isDataReady();
  }

  Serial.println("Start figure-8 calibration after 2 seconds.");
  delay(2000);
  calibrate(10000, &offset_x, &offset_y, &offset_z);
  Serial.println("");

  Serial.print("Initializing SD card...");

  // See if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {
    Serial.println("Card failed, or not present");
  // don't do anything more:
    while (1);
  }
  Serial.println("card initialized.");


  if (!RTC.begin()) {
    Serial.println("RTC failed");
  }
  //RTC.adjust(DateTime(2024, 3, 7, 2, 15, 0));

  // Makes it so we dont overwrite files
  // Also FYI file names must be < 9 characters
  fileName = "waves00.CSV";
  for (uint8_t i = 0; i < 100; i++) {
    fileName[5] = i/10 + '0';
    fileName[6] = i%10 + '0';
    if (! SD.exists(fileName)) {
      // only open a new file if it doesn't exist
      File dataFile = SD.open(fileName, FILE_WRITE);
      String header =
"hour,minute,second,millis,X-acc,Y-acc,Z-acc,Gyro-x,Gyro-y,Gyro-z,x-offset,y-offset,z-offset
,x,y,z";
      dataFile.println(header);
      dataFile.close();
      break;  // leave the loop!
    }
  }

  Serial.println(F("*** SIMCom Module IoT Example ***"));

  #ifdef LED
    pinMode(LED, OUTPUT);
    digitalWrite(LED, LOW);
  #endif

  pinMode(RST, OUTPUT);
  digitalWrite(RST, HIGH); // Default state
  modem.powerOn(PWRKEY); // Power on the module
  moduleSetup(); // Establishes first-time serial comm and prints IMEI

  if (!tempsensor.begin()) {
    Serial.println("Couldn't find the MCP9808!");
    tempsensor.wake(); // Wake up the MCP9808 if it was sleeping and retry
    if (!tempsensor.begin()) while (1);
  }

  // Set modem to full functionality
  modem.setFunctionality(1); // AT+CFUN=1

  modem.setNetworkSettings(F("hologram")); // For Hologram SIM card

  // Perform first-time GPS/GPRS setup if the shield is going to remain on,
  // otherwise these won't be enabled in loop() and it won't work!
#ifndef turnOffShield
  // Enable GPS
  while (!modem.enableGPS(true)) {
    Serial.println(F("Failed to turn on GPS, retrying..."));
    delay(2000); // Retry every 2s
  }
  Serial.println(F("Turned on GPS!"));
  startMillis = millis() - (samplingLag + samplingPeriod);

  #if !defined(SIMCOM_3G) && !defined(SIMCOM_7500) && !defined(SIMCOM_7600)
    // Disable GPRS just to make sure it was actually off so that we can turn it on
    if (!modem.enableGPRS(false)) Serial.println(F("Failed to disable GPRS!"));

    // Turn on GPRS
    while (!modem.enableGPRS(true)) {
      Serial.println(F("Failed to enable GPRS, retrying..."));
      delay(2000); // Retry every 2s
    }
    Serial.println(F("Enabled GPRS!"));
  #endif
#endif

#ifdef PROTOCOL_MQTT_AIO
  mqtt.subscribe(&feed_command); // Only if you're using MQTT
#endif
}

void loop() {
  if (millis() - startMillis >= samplingLag + samplingPeriod) {
    DateTime now;
    File dataFile = SD.open(fileName, FILE_WRITE);
    startMillis = millis();
    while(millis() < startMillis + samplingPeriod) {
      // get acceleration
      acc_x = icm20600.getAccelerationX();
      acc_y = icm20600.getAccelerationY();
      acc_z = icm20600.getAccelerationZ();

      g_x = icm20600.getGyroscopeX();
      g_y = icm20600.getGyroscopeY();
      g_z = icm20600.getGyroscopeZ();

      ak09918.getData(&x, &y, &z);
      xoff = x - offset_x;
      yoff = y - offset_y;
      zoff = z - offset_z;

      // roll/pitch in radian
      roll = atan2((float)acc_y, (float)acc_z);
      pitch = atan2(-(float)acc_x, sqrt((float)acc_y * acc_y + (float)acc_z * acc_z));

      double Xheading = x * cos(pitch) + y * sin(roll) * sin(pitch) + z * cos(roll) *
sin(pitch);
      double Yheading = y * cos(roll) - z * sin(pitch);
      double heading = 180 + 57.3 * atan2(Yheading, Xheading) + declination_SD;

      uint32_t tick = millis();
      now = RTC.now();

      dataString += String(now.hour()) + "," + String(now.minute()) + "," +
String(now.second()) + "," + String(tick) + ","
        + String(acc_x) + "," + String(acc_y) + "," + String(acc_z) + ","
          + String(g_x) + "," + String(g_y) + "," + String(g_z) + ","
            + String(xoff) + "," + String(yoff) + "," + String(zoff) + ","
            + String(x) + "," + String(y) + "," + String(z);

      Serial.println(dataString);

      if (dataFile) {
        dataFile.println(dataString);
        dataString = "";
        Serial.println("wrote to SD");
      }
      delay(LOG_INTERVAL);
    }
    dataFile.close();
  // Connect to cell network and verify connection
  // If unsuccessful, keep retrying every 2s until a connection is made
  while (!netStatus()) {
    Serial.println(F("Failed to connect to cell network, retrying..."));
    delay(2000); // Retry every 2s
  }
  Serial.println(F("Connected to cell network!"));

  battLevel = readVcc(); // Get voltage in mV

  // Measure temperature
  tempsensor.wake(); // Wake up the MCP9808 if it was sleeping
  float tempC = tempsensor.readTempC();
  float tempF = tempC * 9.0 / 5.0 + 32;
  Serial.print("Temp: "); Serial.print(tempC); Serial.print("*C\t");
  Serial.print(tempF); Serial.println("*F");

  Serial.println("Shutting down the MCP9808...");
  tempsensor.shutdown(); // In this mode the MCP9808 draws only about 0.1uA

  float temperature = tempC; // Select what unit you want to use for this example

  delay(500); // I found that this helps

  // Turn on GPS if it wasn't on already (e.g., if the module wasn't turned off)
#ifdef turnOffShield
  while (!modem.enableGPS(true)) {
    Serial.println(F("Failed to turn on GPS, retrying..."));
    delay(2000); // Retry every 2s
  }
  Serial.println(F("Turned on GPS!"));
#endif

  while (!modem.getGPS(&latitude, &longitude, &speed_kph, &heading, &altitude)) {
    Serial.println(F("Failed to get GPS location, retrying..."));
    delay(2000); // Retry every 2s
  }
  Serial.println(F("Found 'eeeeem!"));
  Serial.println(F("---------------------"));
  Serial.print(F("Latitude: ")); Serial.println(latitude, 6);
  Serial.print(F("Longitude: ")); Serial.println(longitude, 6);
  Serial.print(F("Speed: ")); Serial.println(speed_kph);
  Serial.print(F("Heading: ")); Serial.println(heading);
  Serial.print(F("Altitude: ")); Serial.println(altitude);
  Serial.println(F("---------------------"));

  // If the shield was already on, no need to re-enable
#if defined(turnOffShield) && !defined(SIMCOM_3G) && !defined(SIMCOM_7500) &&
!defined(SIMCOM_7600)
  // Disable GPRS just to make sure it was actually off so that we can turn it on
  if (!modem.enableGPRS(false)) Serial.println(F("Failed to disable GPRS!"));

  // Turn on GPRS
  while (!modem.enableGPRS(true)) {
    Serial.println(F("Failed to enable GPRS, retrying..."));
    delay(2000); // Retry every 2s
  }
  Serial.println(F("Enabled GPRS!"));
#endif

  // Format the floating point numbers
  dtostrf(latitude, 1, 6, latBuff);
  dtostrf(longitude, 1, 6, longBuff);
  dtostrf(speed_kph, 1, 0, speedBuff);
  dtostrf(heading, 1, 0, headBuff);
  dtostrf(altitude, 1, 1, altBuff);
  dtostrf(temperature, 1, 2, tempBuff); // float_val, min_width, digits_after_decimal,
char_buffer
  dtostrf(battLevel, 1, 0, battBuff);

  sprintf(locBuff, "%s,%s,%s,%s", speedBuff, latBuff, longBuff, altBuff); // This could look
like "10,33.123456,-85.123456,120.5"

#ifdef PROTOCOL_HTTP_GET
  // GET request

  counter = 0; // This counts the number of failed attempts tries

  #if defined(SIMCOM_3G) || defined(SIMCOM_7500) || defined(SIMCOM_7600)
    // You can adjust the contents of the request if you don't need certain things like
speed, altitude, etc.
    sprintf(URL, "GET /dweet/for/%s?lat=%s&long=%s&speed=%s&head=%s&alt=%s&temp=%s&batt=%s
HTTP/1.1\r\nHost: dweet.io\r\n\r\n",
            imei, latBuff, longBuff, speedBuff, headBuff, altBuff, tempBuff, battBuff);

    // Try a total of three times if the post was unsuccessful (try additional 2 times)
    while (counter < 3 && !modem.postData("www.dweet.io", 443, "HTTPS", URL)) { // Server,
port, connection type, URL
```

```
      Serial.println(F("Failed to complete HTTP/HTTPS request..."));
      counter++; // Increment counter
      delay(1000);
    }
  #else
    sprintf(URL,
"http://dweet.io/dweet/for/%s?lat=%s&long=%s&speed=%s&head=%s&alt=%s&temp=%s&batt=%s", imei,
latBuff, longBuff,
            speedBuff, headBuff, altBuff, tempBuff, battBuff);

    while (counter < 3 && !modem.postData("GET", URL)) {
      Serial.println(F("Failed to post data, retrying..."));
      counter++; // Increment counter
      delay(1000);
    }
  #endif
    // You can also do a POST request instead

  counter = 0; // This counts the number of failed attempts tries

  #if defined(SIMCOM_3G) || defined(SIMCOM_7500) || defined(SIMCOM_7600)
    sprintf(body, "{\"lat\":%s,\"long\":%s}\r\n", latBuff, longBuff); // Terminate with
CR+NL
    sprintf(URL, "POST /dweet/for/%s HTTP/1.1\r\nHost: dweet.io\r\nContent-Length:
%i\r\n\r\n", imei, strlen(body));

    while (counter < 3 && !modem.postData("www.dweet.io", 443, "HTTPS", URL, body)) { //
Server, port, connection type, URL
      Serial.println(F("Failed to complete HTTP/HTTPS request..."));
      counter++; // Increment counter
      delay(1000);
    }
  #else
    //sprintf(URL, "http://dweet.io/dweet/for/%s", imei);
    //sprintf(body, "{\"lat\":%s,\"long\":%s}", latBuff, longBuff);

    const char * token = "865235030714519"; // From thingsboard.io device
    //sprintf(URL, "http://demo.thingsboard.io/api/v1/%s/telemetry", token);
    sprintf(URL, "http://thingsboard.cloud/api/v1/%s/telemetry", token);
    sprintf(body,
"{\"lat\":%s,\"long\":%s,\"speed\":%s,\"head\":%s,\"alt\":%s,\"temp\":%s,\"batt\":%s}",
latBuff, longBuff,
            speedBuff, headBuff, altBuff, tempBuff, battBuff);
  //  sprintf(body, "{\"lat\":%s,\"long\":%s}", latBuff, longBuff); // If all you want is
lat/long


    while (counter < 3 && !modem.postData("POST", URL, body)) {
      Serial.println(F("Failed to complete HTTP POST..."));
      counter++;
      delay(1000);
    }
  #endif

#elif defined(PROTOCOL_HTTP_POST)
  // GET request

  counter = 0; // This counts the number of failed attempts tries

  #if defined(SIMCOM_3G) || defined(SIMCOM_7500) || defined(SIMCOM_7600)
```

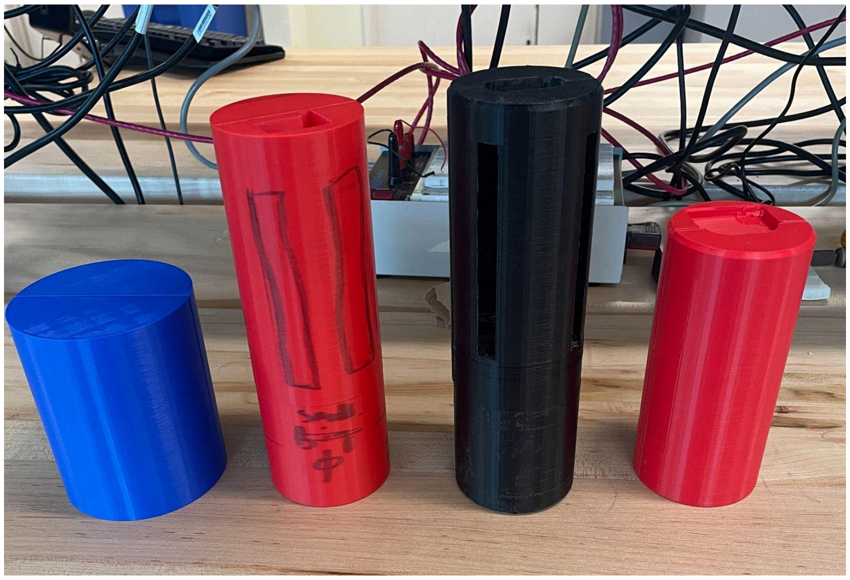
```
    // You can adjust the contents of the request if you don't need certain things like
speed, altitude, etc.
    sprintf(URL, "GET /dweet/for/%s?lat=%s&long=%s&speed=%s&head=%s&alt=%s&temp=%s&batt=%s
HTTP/1.1\r\nHost: dweet.io\r\n\r\n",
            imei, latBuff, longBuff, speedBuff, headBuff, altBuff, tempBuff, battBuff);

    // Try a total of three times if the post was unsuccessful (try additional 2 times)
    while (counter < 3 && !modem.postData("www.dweet.io", 443, "HTTPS", URL)) { // Server,
port, connection type, URL
      Serial.println(F("Failed to complete HTTP/HTTPS request..."));
      counter++; // Increment counter
      delay(1000);
    }
  #else
    sprintf(URL,
"http://dweet.io/dweet/for/%s?lat=%s&long=%s&speed=%s&head=%s&alt=%s&temp=%s&batt=%s", imei,
latBuff, longBuff,
            speedBuff, headBuff, altBuff, tempBuff, battBuff);

    while (counter < 3 && !modem.postData("GET", URL)) {
      Serial.println(F("Failed to post data, retrying..."));
      counter++; // Increment counter
      delay(1000);
    }
  #endif

  // You can also do a POST request instead

  counter = 0; // This counts the number of failed attempts tries

  #if defined(SIMCOM_3G) || defined(SIMCOM_7500) || defined(SIMCOM_7600)
    sprintf(body, "{\"lat\":%s,\"long\":%s}\r\n", latBuff, longBuff); // Terminate with
CR+NL
    sprintf(URL, "POST /dweet/for/%s HTTP/1.1\r\nHost: dweet.io\r\nContent-Length:
%i\r\n\r\n", imei, strlen(body));

    while (counter < 3 && !modem.postData("www.dweet.io", 443, "HTTPS", URL, body)) { //
Server, port, connection type, URL
      Serial.println(F("Failed to complete HTTP/HTTPS request..."));
      counter++; // Increment counter
      delay(1000);
    }
  #else
    //sprintf(URL, "http://dweet.io/dweet/for/%s", imei);
    //sprintf(body, "{\"lat\":%s,\"long\":%s}", latBuff, longBuff);

    const char * token = "865235030714519"; // From thingsboard.io device
    //sprintf(URL, "http://demo.thingsboard.io/api/v1/%s/telemetry", token);
    sprintf(URL, "http://thingsboard.cloud/api/v1/%s/telemetry", token);
    sprintf(body,
"{\"lat\":%s,\"long\":%s,\"speed\":%s,\"head\":%s,\"alt\":%s,\"temp\":%s,\"batt\":%s}",
latBuff, longBuff,
            speedBuff, headBuff, altBuff, tempBuff, battBuff);
  //  sprintf(body, "{\"lat\":%s,\"long\":%s}", latBuff, longBuff); // If all you want is
lat/long


    while (counter < 3 && !modem.postData("POST", URL, body)) {
      Serial.println(F("Failed to complete HTTP POST..."));
      counter++;
      delay(1000);
    }
  #endif

#elif defined(PROTOCOL_MQTT_AIO)
  // Let's use MQTT!

  MQTT_connect();

  // Now publish all the data to different feeds!
  // The MQTT_publish_checkSuccess handles repetitive stuff.
  // You can see the function near the end of this sketch.
  // For the Adafruit IO dashboard map we send the combined lat/long buffer
  MQTT_publish_checkSuccess(feed_location, locBuff);
//  MQTT_publish_checkSuccess(feed_speed, speedBuff); // Included in "location" feed
  MQTT_publish_checkSuccess(feed_head, headBuff);
//  MQTT_publish_checkSuccess(feed_alt, altBuff); // Included in "location" feed
  MQTT_publish_checkSuccess(feed_temp, tempBuff);
  MQTT_publish_checkSuccess(feed_voltage, battBuff);

  // This is our 'wait for incoming subscription packets' busy subloop
  Adafruit_MQTT_Subscribe *subscription;
  while ((subscription = mqtt.readSubscription(5000))) {
    if (subscription == &feed_command) {
      Serial.print(F("*** Got: "));
      Serial.println((char *)feed_command.lastread);
    }
  }

  // Control an LED based on what we receive from the command feed subscription!
  if (strcmp(feed_command.lastread, "ON") == 0) {
    Serial.println(F("*** Commanded to turn on LED!"));
    digitalWrite(LED, HIGH);
  }
  else if (strcmp(feed_command.lastread, "OFF") == 0) {
    Serial.println(F("*** Commanded to turn off LED!"));
    digitalWrite(LED, LOW);
  }
#elif defined(PROTOCOL_MQTT_CLOUDMQTT)
  // Let's use CloudMQTT! NOTE: connecting and publishing work, but everything else
  // still under development!!!
  char MQTT_CLIENT[16] = " ";  // We'll change this to the IMEI

  // Let's begin by changing the client name to the IMEI number to better identify
  strcpy(MQTT_CLIENT, imei); // Copy the contents of the imei into the char array
"MQTT_client"

  // Connect to MQTT broker
  if (!modem.TCPconnect(MQTT_SERVER, MQTT_SERVERPORT)) Serial.println(F("Failed to connect
to TCP/IP!"));
  // CloudMQTT requires "MQIsdp" instead of "MQTT"
  if (!modem.MQTTconnect("MQIsdp", MQTT_CLIENT, MQTT_USERNAME, MQTT_KEY))
Serial.println(F("Failed to connect to MQTT broker!"));

  // Publish each data point under a different topic!
  Serial.println(F("Publishing data to their respective topics!"));
  if (!modem.MQTTpublish("location", locBuff)) Serial.println(F("Failed to publish data!"));
// Combined data
  if (!modem.MQTTpublish("speed", speedBuff)) Serial.println(F("Failed to publish data!"));
  if (!modem.MQTTpublish("heading", headBuff)) Serial.println(F("Failed to publish data!"));
  if (!modem.MQTTpublish("altitude", altBuff)) Serial.println(F("Failed to publish data!"));
  if (!modem.MQTTpublish("temperature", tempBuff)) Serial.println(F("Failed to publish
data!"));
  if (!modem.MQTTpublish("voltage", battBuff)) Serial.println(F("Failed to publish data!"));

  if (!modem.TCPclose()) Serial.println(F("Failed to close connection!"));

#endif

  //Only run the code below if you want to turn off the shield after posting data
#ifdef turnOffShield
  // Disable GPRS
  // Note that you might not want to check if this was successful, but just run it
  // since the next command is to turn off the module anyway
  if (!modem.enableGPRS(false)) Serial.println(F("Failed to disable GPRS!"));

  // Turn off GPS
  if (!modem.enableGPS(false)) Serial.println(F("Failed to turn off GPS!"));

  // Power off the module. Note that you could instead put it in minimum functionality mode
  // instead of completely turning it off. Experiment different ways depending on your
application!
  // You should see the "PWR" LED turn off after this command
//  if (!modem.powerDown()) Serial.println(F("Failed to power down modem!")); // No retries
  counter = 0;
  while (counter < 3 && !modem.powerDown()) { // Try shutting down
    Serial.println(F("Failed to power down modem!"));
    counter++; // Increment counter
    delay(1000);
  }
#endif

  // Shut down the MCU to save power
#ifndef samplingRate
  Serial.println(F("Shutting down..."));
  delay(5); // This is just to read the response of the last AT command before shutting down
  MCU_powerDown(); // You could also write your own function to make it sleep for a certain
duration instead
#else

  // Only run the initialization again if the module was powered off
  // since it resets back to 115200 baud instead of 4800.
  #ifdef turnOffShield
    modem.powerOn(PWRKEY); // Powers on the module if it was off previously
    moduleSetup();
  #endif

#endif


  }
  nextStart = startMillis +  samplingLag + samplingPeriod;
  endLoop = millis();
  if (endLoop <= nextStart){
    Serial.println("---------------------------------------------------");
    delayTime = nextStart - endLoop;
    Serial.print(F("Waiting for ")); Serial.print(delayTime/1000); Serial.println(F("
seconds\r\n"));
    delay(delayTime);
  }
```

```
}

void moduleSetup() {
  // SIM7000 takes about 3s to turn on and SIM7500 takes about 15s
  // Press Arduino reset button if the module is still turning on and the board doesn't find
it.
  // When the module is on it should communicate right after pressing reset

  // Software serial:
  modemSS.begin(115200); // Default SIM7000 shield baud rate

  Serial.println(F("Configuring to 9600 baud"));
  modemSS.println("AT+IPR=9600"); // Set baud rate
  delay(100); // Short pause to let the command run
  modemSS.begin(9600);
  if (! modem.begin(modemSS)) {
    Serial.println(F("Couldn't find modem"));
    while (1); // Don't proceed if it couldn't find the device
  }

  type = modem.type();
  Serial.println(F("Modem is OK"));
  Serial.print(F("Found "));
  switch (type) {
    case SIM800L:
      Serial.println(F("SIM800L")); break;
    case SIM800H:
      Serial.println(F("SIM800H")); break;
    case SIM808_V1:
      Serial.println(F("SIM808 (v1)")); break;
    case SIM808_V2:
      Serial.println(F("SIM808 (v2)")); break;
    case SIM5320A:
      Serial.println(F("SIM5320A (American)")); break;
    case SIM5320E:
      Serial.println(F("SIM5320E (European)")); break;
    case SIM7000:
      Serial.println(F("SIM7000")); break;
    case SIM7070:
      Serial.println(F("SIM7070")); break;
    case SIM7500:
      Serial.println(F("SIM7500")); break;
    case SIM7600:
      Serial.println(F("SIM7600")); break;
    default:
      Serial.println(F("???")); break;
  }

  // Print module IMEI number.
  uint8_t imeiLen = modem.getIMEI(imei);
  if (imeiLen > 0) {
    Serial.print("Module IMEI: "); Serial.println(imei);
  }
}

// Read the module's power supply voltage
float readVcc() {
  // Read battery voltage
  if (!modem.getBattVoltage(&battLevel)) Serial.println(F("Failed to read batt"));
  else Serial.print(F("battery = ")); Serial.print(battLevel); Serial.println(F(" mV"));
  // Read LiPo battery percentage
//  if (!modem.getBattPercent(&battLevel)) Serial.println(F("Failed to read batt"));
//  else Serial.print(F("BAT % = ")); Serial.print(battLevel); Serial.println(F("%"));

  return battLevel;
}

bool netStatus() {
  int n = modem.getNetworkStatus();

  Serial.print(F("Network status ")); Serial.print(n); Serial.print(F(": "));
  if (n == 0) Serial.println(F("Not registered"));
  if (n == 1) Serial.println(F("Registered (home)"));
  if (n == 2) Serial.println(F("Not registered (searching)"));
  if (n == 3) Serial.println(F("Denied"));
  if (n == 4) Serial.println(F("Unknown"));
  if (n == 5) Serial.println(F("Registered roaming"));

  if (!(n == 1 || n == 5)) return false;
  else return true;
}

// Function to connect and reconnect as necessary to the MQTT server.
// Should be called in the loop function and it will take care if connecting.
#ifdef PROTOCOL_MQTT_AIO
  void MQTT_connect() {
    int8_t ret;

    // Stop if already connected.
    if (mqtt.connected()) {
      return;
    }

    Serial.println("Connecting to MQTT... ");

    while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
      Serial.println(mqtt.connectErrorString(ret));
      Serial.println("Retrying MQTT connection in 5 seconds...");
      mqtt.disconnect();
      delay(5000);  // wait 5 seconds
    }
    Serial.println("MQTT Connected!");
  }

  void MQTT_publish_checkSuccess(Adafruit_MQTT_Publish &feed, const char *feedContent) {
    Serial.println(F("Sending data..."));
    if (! feed.publish(feedContent)) {
      Serial.println(F("Failed"));
      txfailures++;
    }
    else {
      Serial.println(F("OK!"));
      txfailures = 0;
    }
  }
#endif

// Turn off the MCU completely. Can only wake up from RESET button
// However, this can be altered to wake up via a pin change interrupt
```

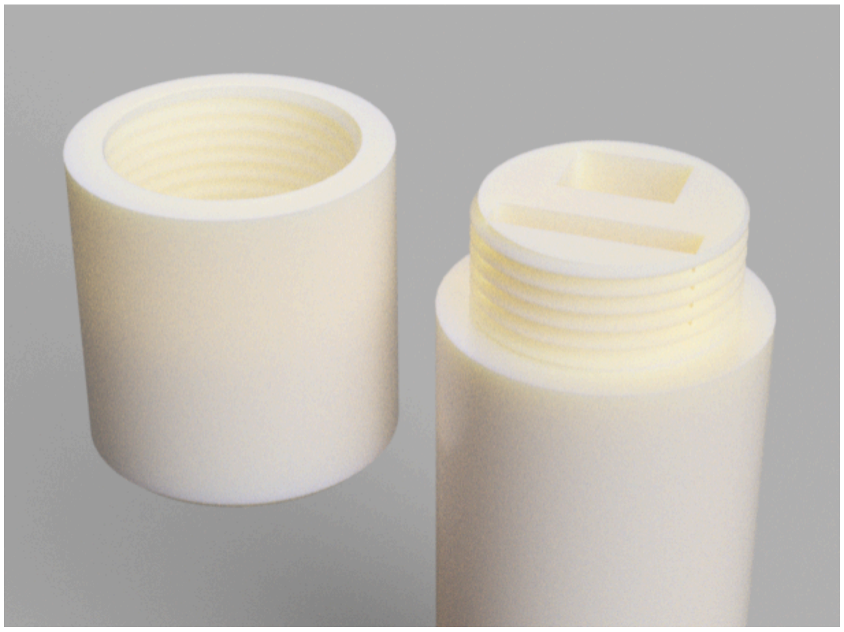
```
void MCU_powerDown() {
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  ADCSRA = 0; // Turn off ADC
  power_all_disable ();  // Power off ADC, Timer 0 and 1, serial interface
  sleep_enable();
  sleep_cpu();
}


void calibrate(uint32_t timeout, int32_t* offsetx, int32_t* offsety, int32_t* offsetz) {
    int32_t value_x_min = 0;
    int32_t value_x_max = 0;
    int32_t value_y_min = 0;
    int32_t value_y_max = 0;
    int32_t value_z_min = 0;
    int32_t value_z_max = 0;
    uint32_t timeStart = 0;

    ak09918.getData(&x, &y, &z);

    value_x_min = x;
    value_x_max = x;
    value_y_min = y;
    value_y_max = y;
    value_z_min = z;
    value_z_max = z;
    delay(100);

    timeStart = millis();

    while ((millis() - timeStart) < timeout) {
        ak09918.getData(&x, &y, &z);

        /* Update x-Axis max/min value */
        if (value_x_min > x) {
            value_x_min = x;
            // Serial.print("Update value_x_min: ");
            // Serial.println(value_x_min);

        } else if (value_x_max < x) {
            value_x_max = x;
            // Serial.print("update value_x_max: ");
            // Serial.println(value_x_max);
        }

        /* Update y-Axis max/min value */
        if (value_y_min > y) {
            value_y_min = y;
            // Serial.print("Update value_y_min: ");
            // Serial.println(value_y_min);

        } else if (value_y_max < y) {
            value_y_max = y;
            // Serial.print("update value_y_max: ");
            // Serial.println(value_y_max);
        }

        /* Update z-Axis max/min value */
        if (value_z_min > z) {
            value_z_min = z;
            // Serial.print("Update value_z_min: ");
            // Serial.println(value_z_min);

        } else if (value_z_max < z) {
            value_z_max = z;
            // Serial.print("update value_z_max: ");
            // Serial.println(value_z_max);
        }

        //Serial.print(".");
        delay(100);

    }

    *offsetx = value_x_min + (value_x_max - value_x_min) / 2;
    *offsety = value_y_min + (value_y_max - value_y_min) / 2;
    *offsetz = value_z_min + (value_z_max - value_z_min) / 2;
}
```

Appendix B - Mechanical Design

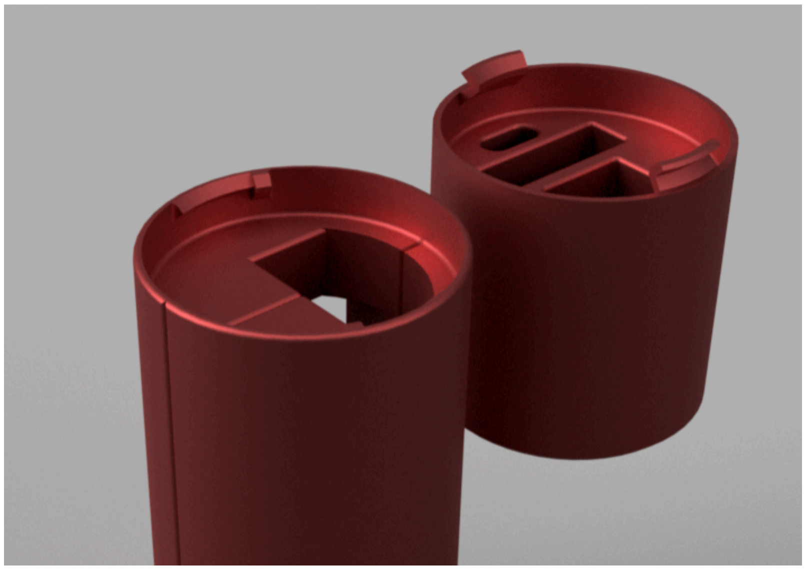
B.1 Electronics Enclosure Iterations



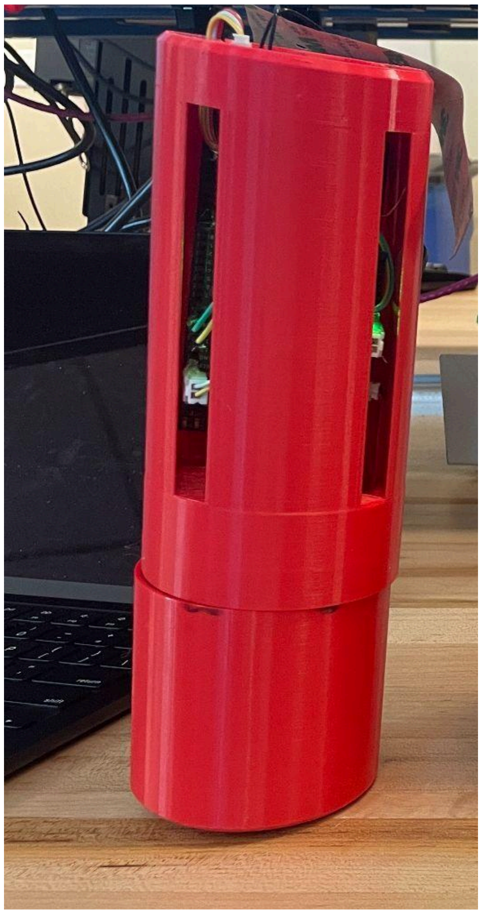
B.2 Screw mechanism to hold the battery chamber and the electronics chamber together



B.3 Snap+twist mechanism to hold the chambers together



B.4 Final electronics enclosure



Appendix C - Matlab Analysis Code

C.1 Primary analysis script

```
%Wrapper function for visualizing wave data

%capabilities: can truncate, use a low-pass filter, fourier transform, and

%kalman filter if gyro data is provided

%Isaac Need 3/7/2024

timeunits=1000; %1 second in the units the data logger clock is in (eg 1.0*10^6 is usec)

truncateIf=1;%toggle to truncate data, 1=yes, 0=no

shouldFilter=0;%toggle to use a low pass filter on the data, 1=yes, 0=no.

g=9.81; %acceleration due to gravity, m/s^2

%data file location

myData='C:\Users\xavin\OneDrive\Documents\UCSD\WI24\MAE223\Data\WaveChannel02_Packet2.csv';

%read data in and split into their own variables (ignoring RTC for now)

dataRaw=readmatrix(myData);

millis=dataRaw(:,4);

time=(millis-millis(1))/timeunits;

gyroOffset(:,[2:4])=dataRaw(:,[8:10])*pi/180; %convert degrees to radians

gyroOffset(:,1)=time;

accelRaw(:,[2:4])=dataRaw(:,[5:7]);

accelRaw(:,1)=time;

sampFreq=1/(mean(diff(time)));%sampling frequency

nyquistFreq=sampFreq/2; %Nyquist frequency

%%NOAA data from Scripp's Wave Bouy on 3/16/2024 at approx 1300

wvheight=0.6;%meters

wvPeriod=7.9;%sec

wvFreq=1/wvPeriod;

%available data sets

%%WaveChannel01_Packet1      WaveChannel01_Packet2        WaveChannel01_Packet2_v2
WaveChannel01_Packet3
```

```
%%WaveChannel02_Packet1      WaveChannel02_Packet2        WaveChannel02_Packet3

%%WAVES09

accelRaw(:,[2:4])=accelRaw(:,[2:4])*g/1000; %convert acceleration to m/s^2. IMU outputs mg's

%plot raw acceleration

figure(1)

plot(accelRaw(:,1),accelRaw(:,2))

hold on

plot(accelRaw(:,1),accelRaw(:,3))

plot(accelRaw(:,1),accelRaw(:,4))

xlabel('Time (sec)')

ylabel('Acceleration (m/s^2)')

title('Raw Acceleration')

legend('X Acceleration', 'Y Acceleration', 'Z Acceleration');

hold off

%define constants and expected frequency

fN_one=1;

fN_half=.5;

fN_quarter=.25;
```

```
%truncate the array. front/back windows were adjusted according to raw

%data

if truncateIf==1

    frontWindow=110; %front truncation window in seconds

    backWindow=150; %back truncation window in seconds

    for i=2:4
[truncatedAccel(:,[1,i]),~,~,~]=truncate(accelRaw(:,[1,i]),frontWindow,backWindow);
[truncatedGyro(:,[1,i]),~,~,~]=truncate(gyroOffset(:,[1,i]),frontWindow,backWindow);

    end

end

    if truncateIf==0
```

```
        truncatedAccel=accelRaw;

        truncatedGryo=gyroOffset;

    end

figure(2) %check truncated acceleration data

plot(truncatedAccel(:,1),truncatedAccel(:,2))

hold on

plot(truncatedAccel(:,1),truncatedAccel(:,3))

plot(truncatedAccel(:,1),truncatedAccel(:,4))

xlabel('Time (sec)')

ylabel('Acceleration (m/s^2)')

title('Truncated Acceleration')

legend('X Acceleration', 'Y Acceleration', 'Z Acceleration');

hold off
```

```
%filter the array. 4th order butterworth filter. Unused in final analysis

%because it was observed to mess up the kalman filter

if shouldFilter==1

    cutoffFreq=nyquistFreq*0.5; %cutoff frequency in Hz

    for i=2:4

    [filteredAccel(:,[1,i]),~,~]=lpFilterIN(truncatedAccel(:,[1,i]),cutoffFreq);

    end

   filteredAccel(:,1)=truncatedAccel(:,1);

   filteredGyro=truncatedGyro;

end

    if shouldFilter==0

        filteredAccel=truncatedAccel;

        filteredGyro=truncatedGyro;

    end

figure(3) %check filtered acceleration data
```

```
plot(filteredAccel(:,1),filteredAccel(:,2))

hold on

plot(filteredAccel(:,1),filteredAccel(:,3))

plot(filteredAccel(:,1),filteredAccel(:,4))

xlabel('Time (sec)')

ylabel('Acceleration (m/s^2)')

title('Filtered Acceleration')

legend('X Acceleration', 'Y Acceleration', 'Z Acceleration');

hold off
```

```
%fourier transform of the data

chunk=(300);%length of chunks to break input data into.

for i=2:4

[freqVec(:,i-1),spectra(:,i-1),Parseval(i-1)]=spectrumCB(filteredAccel(:,1),filteredAccel(:,i)
,chunk);

[freqVecRaw(:,i-1),spectraRaw(:,i-1),ParsevalRaw(i-1)]=spectrumCB(time,accelRaw(:,i),chunk);
%take fft of raw data for ref

end

for i=1:3

fft_var(i)=trapz(freqVec(:,i),spectra(:,i));

Parseval(i)

ParsevalRaw(i);

end
```

```
%check fourier transforms of each axis against the raw data

figure(4) %x axis

loglog(freqVec(:,1),spectra(:,1))

hold on

loglog(freqVecRaw(:,1),spectraRaw(:,1))%plot raw spectra

loglog([fN_one fN_one],[10^-5 max(max(spectra))*10]) %1 Hz ref
```

```matlab
loglog([fN_half fN_half],[10^-5 max(max(spectra))*10]) %0.5 Hz ref

loglog([fN_quarter fN_quarter],[10^-5 max(max(spectra))*10]) %0.25 Hz ref

xlabel('Frequency Hz')

ylabel('xAccel (m/s^2)^2/Hz')

title('X Spectrum')

legend('Filtered X Spectra', 'Raw X Spectra','One Hz Ref Frequency','Half Hz Ref
Frequency','Quarter Hz Ref Frequency');

hold off

figure(5) %y axis

loglog(freqVec(:,2),spectra(:,2))

hold on

loglog(freqVecRaw(:,2),spectraRaw(:,2))%plot raw spectra

loglog([fN_one fN_one],[10^-5 max(max(spectra))*10]) %1 Hz ref

loglog([fN_half fN_half],[10^-5 max(max(spectra))*10]) %0.5 Hz ref

loglog([fN_quarter fN_quarter],[10^-5 max(max(spectra))*10]) %0.25 Hz ref

xlabel('Frequency Hz')

ylabel('yAccel (m/s^2)^2/Hz')

title('Y Spectrum')

legend('Filtered Y Spectra', 'Raw Y Spectra','One Hz Ref Frequency','Half Hz Ref
Frequency','Quarter Hz Ref Frequency');

hold off

figure(6) %z axis

loglog(freqVec(:,3),spectra(:,3))

hold on

loglog(freqVecRaw(:,3),spectraRaw(:,3))%plot raw spectra

loglog([fN_one fN_one],[10^-5 max(max(spectra))*10]) %1 Hz ref

loglog([fN_half fN_half],[10^-5 max(max(spectra))*10]) %0.5 Hz ref

loglog([fN_quarter fN_quarter],[10^-5 max(max(spectra))*10]) %0.25 Hz ref

xlabel('Frequency Hz')

ylabel('zAccel (m/s^2)^2/Hz')
```

```
title('Z Spectrum')

legend('Filtered Z Spectra', 'Raw Z Spectra','One Hz Ref Frequency','Half Hz Ref
Frequency','Quarter Hz Ref Frequency');

hold off
```

```
%Compare fourier transforms of each axis

figure(7)

loglog(freqVec(:,1),spectra(:,1))

hold on

loglog(freqVec(:,2),spectra(:,2))

loglog(freqVec(:,3),spectra(:,3))

loglog([fN_one fN_one],[10^-5 max(max(spectra))*10]) %1 Hz ref

loglog([fN_half fN_half],[10^-5 max(max(spectra))*10]) %0.5 Hz ref

loglog([fN_quarter fN_quarter],[10^-5 max(max(spectra))*10]) %0.25 Hz ref

xlabel('Frequency Hz')

ylabel('xAccel (m/s^2)^2/Hz')

title('Axis Comparison Spectrum')

legend('X Spectra', 'Y Spectra', 'Z Spectra','One Hz Ref Frequency','Half Hz Ref
Frequency','Quarter Hz Ref Frequency');

hold off
```

```
%Kalman filter for fusing the gyro to the accelerometer data

kalmanFilt=imufilter('SampleRate',sampFreq,'AccelerometerNoise',0.00056753,'GyroscopeNoise',0.
162278389); %accelerometer noise variance from zero g test

[orient,angVel]=kalmanFilt(filteredAccel(:,[2:4]),filteredGyro(:,[2:4]));

angles=eulerd(orient,'ZYX','frame');

for i=1:3

    angles(:,i)=detrend(angles(:,i),1);

end

for i=1:3
```

```
[freqVec2(:,i),spectra2(:,i),Parseval2(i)]=spectrumCB(filteredAccel(:,1),angles(:,i),chunk);

[freqVec3(:,i),spectra3(:,i),Parseval3(i)]=spectrumCB(filteredAccel(:,1),angVel(:,i),chunk);

end

for i=1:size(angVel,1)

angVelMag(i)=sqrt(angVel(i,1)^2+angVel(i,2)^2+angVel(i,3)^2);

end

[freqVec4(:,i),spectra4(:,i),Parseval4(i)]=spectrumCB(filteredAccel(:,1),angVelMag,chunk);

figure(4) %sometimes seems to transpose x and y

loglog(freqVec2(:,1),spectra2(:,1))

hold on

loglog(freqVec2(:,1),spectra2(:,2))

loglog(freqVec2(:,1),spectra2(:,3))

loglog(freqVec(:,1),spectra(:,1))

loglog(freqVec(:,2),spectra(:,2))

loglog(freqVec(:,3),spectra(:,3))

loglog(freqVec3(:,1),spectra3(:,1))

loglog(freqVec3(:,2),spectra3(:,2))

loglog(freqVec3(:,3),spectra3(:,3))

loglog(freqVec4,spectra4)

loglog([fN_one fN_one],[10^-5 max(max(spectra2))*10]) %1 Hz ref

loglog([fN_half fN_half],[10^-5 max(max(spectra2))*10]) %0.5 Hz ref

loglog([fN_quarter fN_quarter],[10^-5 max(max(spectra2))*10]) %0.25 Hz ref

xlabel('Frequency Hz')

ylabel('xAccel (m/s^2)^2/Hz')

title('Orientation Spectrum')

legend('X Orientation Spectra', 'Y Orientation Spectra','Z Orientation Spectra','X ang Vel
Spectra','Y ang Vel Spectra','Z ang Vel Spectra','X Accel Spectra','Y Accel Spectra','Z Accel
Spectra','Accel Mag Spectra','One Hz Ref Frequency','Half Hz Ref Frequency','Quarter Hz Ref
Frequency');

hold off
```

```
%Figures

figure (8)

plot(filteredAccel(:,1),angles)

title('Orientation Estimate')

legend('Z-axis', 'Y-axis', 'X-axis')

xlabel('Time (s)')

ylabel('Rotation (degrees)')

figure (9)

plot(filteredAccel(:,1),angVel(:,1))

hold on

plot(filteredAccel(:,1),angVel(:,2))

plot(filteredAccel(:,1),angVel(:,3))

title('Angular Velocity')

legend('X-axis', 'Y-axis', 'Z-axis')

hold off
```

```
%Convert local to global accelerations

locAccel=filteredAccel(:,2:4);

for i=1:size(locAccel(:,1),1)

globAccel(i,:)=locAccel(i,:)*quat2rotm(orient(i,:));

end

globAccelQuant=globAccel;

for i=1:3

[freqVec4(:,i),spectra4(:,i),Parseval4(i)]=spectrumCB(filteredAccel(:,1),globAccel(:,i),chunk)
;

end

figure (11)

loglog(freqVec4(:,1),spectra4(:,1))
```

```
hold on

loglog(freqVec4(:,2),spectra4(:,2))

loglog(freqVec4(:,3),spectra4(:,3))

title('Global Acceleration Spectra for Wave Channel Test')

xlabel('Frequency Hz')

ylabel('x Acceleration (m/s^2)^2/Hz')

% loglog([.65 .65],[10^-5 max(max(spectra))*10],"--",'LineWidth',1.5,'Color','#000000') %0.65
Hz ref

loglog([fN_one fN_one],[10^-5 max(max(spectra))*10],":",'LineWidth',1.5,'Color','#000000')
%0.5 Hz ref

loglog([fN_half fN_half],[10^-5 max(max(spectra))*10],"-.",'LineWidth',1.5,'Color','#000000')
%0.25 Hz ref

legend('Y Acceleration','X Acceleration','Z Acceleration','1 Hz Reference','1/2 Hz
Reference');

hold off
```

```
%integrate to positions

figure (12)

plot(filteredAccel(:,1),globAccel(:,1))

hold on

plot(filteredAccel(:,1),globAccel(:,2))

plot(filteredAccel(:,1),globAccel(:,3))

title('Global Acceleration')

legend('Y Acceleration','X Acceleration','Z Acceleration')

hold off

%detrend and integrate global acceleration to velocity

globZAccel=detrend(globAccel(:,3));

globXAccel=detrend(globAccel(:,2));

zVel=cumtrapz(filteredAccel(:,1),globZAccel);

zVel=detrend(zVel);

zLoc2=cumtrapz(filteredAccel(:,1),zVel);
```

```
zLoc2=detrend(zLoc2);

%integrate angular velocity to position to position and detrend using lower order polynomials
as needed

zLoc=cumtrapz(filteredAccel(:,1),angVel(:,3)); % best results were achieved using short
windows ~5 seconds selected to start at peak accelerations

zLoc=detrend(zLoc,1);

figure (13)

plot(filteredAccel(1260:1310,1),zLoc*100/2.54) %plot Kalman location estimate in cm

hold on

xlabel('Time from Start (sec)')

ylabel('Z Position (in)')

title('Kalman Position Estimate for Open Water Test')

hold off
```

C.2: Functions called by main script

```
function [f,a,Parseval] = spectrumCB(time, data, chunk)

% Written by Charlotte Bellerjeau for SIOC 221A

% Inputs:

% time: time vector of size [n x 1]

% data: data vector of size [n x 1]

% chunk: length of chunks to split the data into, each chunk will be

% one 'independent' spectral estimate

% Returns:

% f: frequency vector

% a: averaged, normalized spectrum with chunk x 2 degrees of freedom

% (no windowing in this function)

% Parseval: checking that the variance of the spectrum is the same

% as the variance of the original data

% split into chunks

ind1 = 1; %first index

%for however many overlapping chunks will for into the data

for i=1:floor(chunk\length(data))*2

%unless we've exceeding the length of the dataset

if ind1+chunk<length(data)

%add a column vector to data1 with the next chunk

data1(:,i) = data(ind1:ind1+chunk);

end

ind1 = ind1+chunk/2; %step index by forward by a half-chunk

end

%frequencies

dt = mean(diff(time)); %time between samples [days]

fn = 1/2/dt; % Nyquist frequency

N= length(data1(:,1));
```

```
T = dt*N; %length of record [days]

df = 1/T; % fundamental frequency [cpd]

f = 0:df:fn; % frequency vector [cpd]

%compute spectrum of each chunk and average

for i=1:length(data1(1,:))

data2(:,i) = detrend(data1(:,i));

a = fft(data2(:,i));

amp=abs(a(1:(N+1)/2)).^2; %take half of spectrum and square

amp = amp / N.^2; % MATLAB normalization

amp = amp .* 2; % lost variance

amp = amp / df; % definition of the spectrum

A(:,i) = amp;

end

a = mean(A,2);


%check Parseval's theorem

variance=std(data)^2;

int_spec = trapz(f,a);

%int_spec = sum(a)*df;

Parseval=int_spec/variance;

end
```

```
function
[truncatedArray,newVar,oldVar,nPointsSkippedFront]=truncate(dataArray,frontWindow,backWindow)

%Written by Isaac Need 2/20/2024

% truncates the provided data array along the 1 direction. Assumes col 1 is

%time stamps and col 2 is magnitude information.

%Truncation is in support of fft so the new and old data variance are returnd as well

%calculate old variance
```

```
oldVar=var(dataArray(:,2));

%calculate measurement period based on average gap between time stamp from input

%data in column 1

measuredIncrements=zeros(size(dataArray(:,1),1)-1,1);

for i=1:size(dataArray,1)-1

    measuredIncrements(i,1)=dataArray(i+1,1)-dataArray(i,1);

end

%find sampling frequency

realRate=mean(measuredIncrements); %real sampling period

realF=1/(realRate); %real sampling frequency

nPoints=size(dataArray,1);

%calculate number of points to skip based on front and back windows

nPointsSkippedFront=round(frontWindow*realF,0);%nDatapoints to skip at the beginning

nPointsSkippedBack=round(backWindow*realF,0);%nDatapoints to skip at the end

%truncate the data array

truncatedArray(:,1)=dataArray(nPointsSkippedFront:size(dataArray,1)-nPointsSkippedBack,1);

truncatedArray(:,2)=dataArray(nPointsSkippedFront:size(dataArray,1)-nPointsSkippedBack,2);

newVar=var(truncatedArray(:,2));

end
```

```
function [filteredData,newVar,oldVar]=lpFilterIN(rawData,cutoffFreq)

%Written by Isaac Need 2/20/2024

% Define filter parameters

measuredIncrements=zeros(size(rawData(:,1),1)-1,1);

for i=1:size(rawData,1)-1

    measuredIncrements(i,1)=rawData(i+1,1)-rawData(i,1);

end

%input data variance
```

```
oldVar=var(rawData(:,2));

 %find sampling frequency

realRate=mean(measuredIncrements); %real sampling period

realF=1/(realRate); %real sampling frequency

fc = cutoffFreq; % Cutoff frequency in Hz

fs = realF/2; % Nyquist frequency in Hz

order = 4; % Filter order

% Design a low-pass Butterworth filter

[b, a] = butter(order, fc/(fs), 'low');

filteredData(:,2)=filter(b,a,rawData(:,2)); %filter magnitude data

filteredData(:,1)=rawData(:,1); %pass time data through

newVar=var(filteredData(:,2));

end
```