Digital Flight Data Recorder

Project Engineering

Year 4

Rokas Cesiunas

Bachelor of Engineering (Honours) in Software and Electronic Engineering

Galway-Mayo Institute of Technology

2020/2021

Project Hardware Components

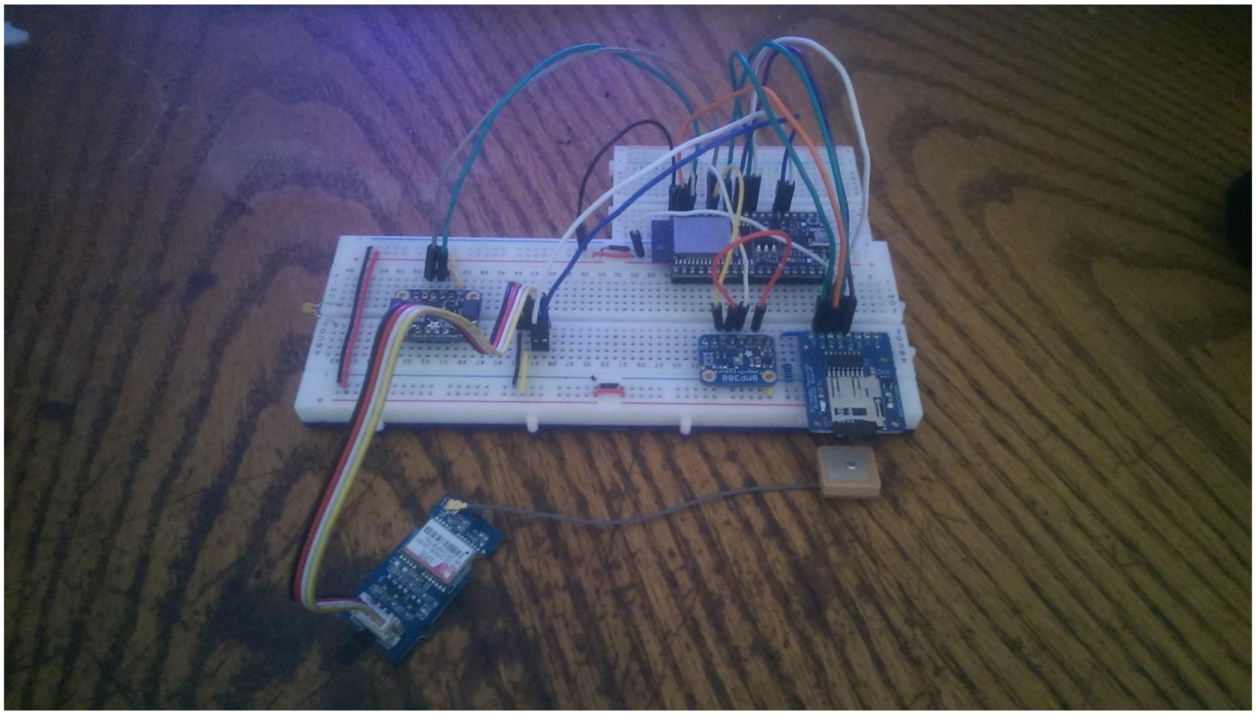


Figure 0-1 Diagram of Components

Declaration

This project is presented in partial fulfilment of the requirements for the degree of Bachelor of Engineering (Honours) in Software and Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

___________Rokas Cesiunas_____________

Table of Contents

1 Summary

This project takes inspiration from the black boxes that are legally required on all commercial flights and are also present on most private airplanes. The purpose of this black box is to record various sensors onboard the airplane and to store this data for investigation in the event of an accident or crash.

The goal of this project was to take this idea of a data recorder and reduce its size so that it may be used on remote control aircraft or drones. By having flight data and logs you will be able to look back on past flights to measure performance or to see what went wrong in the case of a malfunction or crash.

To do this, the sensors used were a Pressure sensor to find altitude, GPS for location and speed readings, Accelerometer, Magnetometer, Gyroscope for acceleration, orientation and rotation measurements. The main board used was an ESP-32, it handled the I2C communication and reading of these sensors. The code written in C/C++ and the web page in JavaScript.

At the end of this project, the built in ESP-32 Wi-Fi was used to able to send the data from the sensors to a local webpage. This webpage then redirected the data to a database. When you wanted to view the data, it would read the database and display the data using graphs.

2 Poster

Digital Flight Data Recorder

Rokas Cesiunas

BEng Software and Electronic Engineering

Example of data recorder

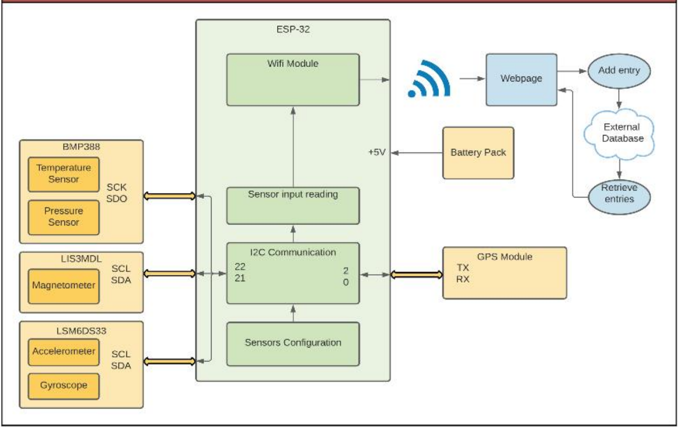
GMIT
INSTITIÚID TEICNEOLAÍOCHTA NA GAILLIMHE-MAIGH EO
GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Summary

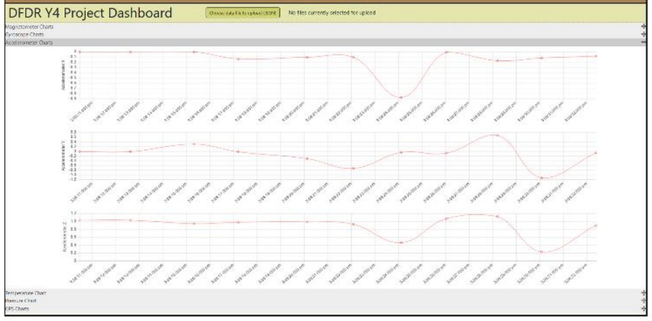
This project takes inspiration from the black boxes that are legally required on all commercial flights and are also present on most private airplanes. The purpose of this black box is to record various sensors onboard the airplane and to store this data for investigation in the event of an accident or crash.

The goal of this project was to take this idea of a data recorder and reduce its size so that it may be used on remote control aircraft or drones. By having flight data and logs you will be able to look back on past flights to measure performance or to see what went wrong in the case of a malfunction or crash.

Architecture Diagram



Results Webpage


DFDR Y4 Project Dashboard

Technologies

- Arduino IDE—C/C++ programming
- Sensor Configuration / communication—Wire Library I2C
- Serial Communication
- HTML / JavaScript

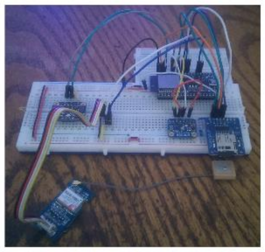


Figure 2-1 Project Poster

3 Introduction

This project was intended to be used on small remote-control aircraft or drones as a means of recording flight data which could be used to find the cause of a crash if it were to occur. The data recorded, such as location, speed, orientation, altitude, and rotation would give the operator information about what is happening onboard the flight. This data would allow you to go back, inspect all the parameters and performance of the aircraft. This would be useful for many reasons such as, building prototype drones to ensure that they meet specifications. Debugging flight systems or finding out caused a malfunction.

It is all built upon a main programmable board (ESP-32) with the sensors connecting to it. The sensors used were an accelerometer and gyroscope (LSM6DS33) which record the acceleration and rotation respectively, magnetometer (LIS3MDL) which can tell us magnetic direction, pressure sensor (BMP388) for altitude measurement, and GPS sensor (SIM28). The data is sent to a webpage via the built in Wi-Fi on the ESP-32 board.

4 Flight recorder background

A flight data recorder is a device that is used to record significant flight parameters, a cockpit voice recorder that records all sounds inside the cockpit would also be integrated to make up a flight recorder. These recorders are built into a secure box to ensure that the recordings can survive strong impacts 3400g and extreme temperatures over 1000°C as required by EUROCAE ED-112 [1].

These devices have been made mandatory since 1965 on all commercial airplanes, multiengine airplanes, airplanes having 10 or more passenger seats as well as helicopters. This device has to record 88 different parameters during flight, such as roll attitude, heading, and time of each radio transmission [2].

The reason for making the recorders mandatory is to help investigators in the case of an accident. The data recorded can provide invaluable information about what happened or what went wrong. This in turn can help to improve safety by learning from mistakes, improving existing aircraft and developing new safety features.

5 Project Architecture

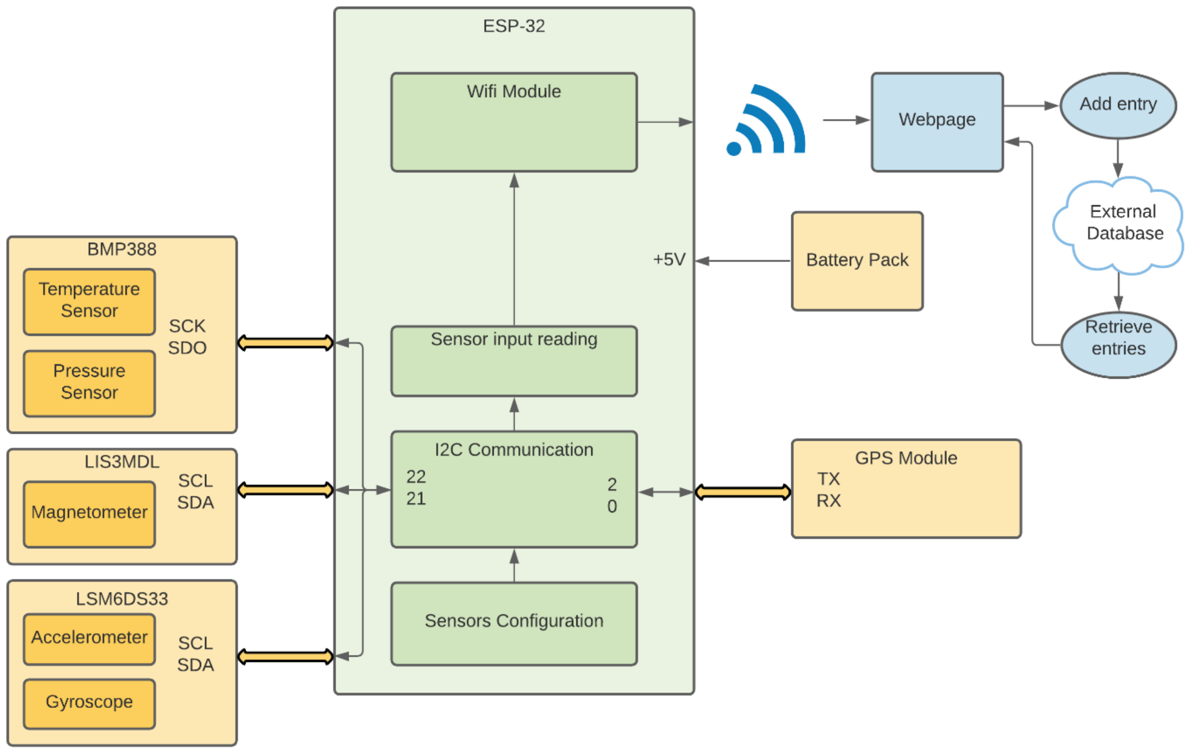
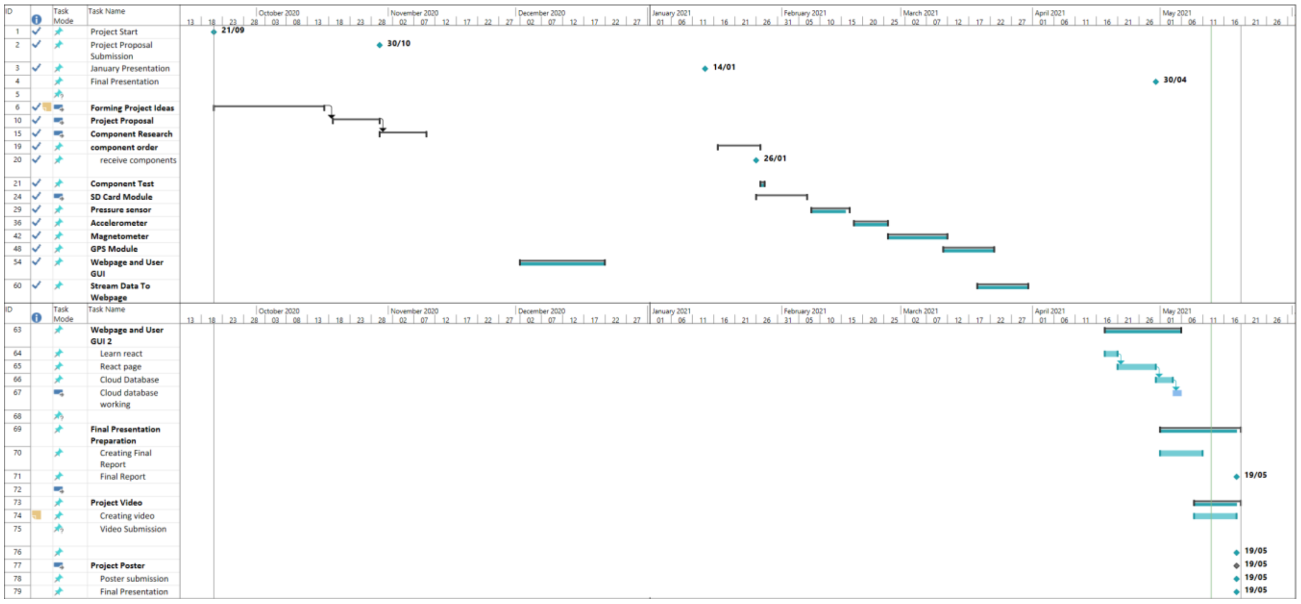


Figure 5-1 Architecture Diagram

6 Project Plan



ID		Task Mode	Task Name
1	✓	📌	Project Start
2	✓	📌	Project Proposal Submission
3	✓	📌	January Presentation
4		📌	Final Presentation
5		📌	
6	✓ 🟨	▬	**Forming Project Ideas**
10	✓	▬	**Project Proposal**
15	✓	▬	**Component Research**
19	✓	📌	**component order**
20	✓	📌	receive components
21	✓	📌	**Component Test**
24	✓	▬	**SD Card Module**
29	✓	📌	**Pressure sensor**
36	✓	📌	**Accelerometer**
42	✓	📌	**Magnetometer**
48	✓	📌	**GPS Module**
54	✓	📌	**Webpage and User GUI**
60	✓	📌	**Stream Data To Webpage**
63		📌	**Webpage and User GUI 2**
64		📌	Learn react
65		📌	React page
66		📌	Cloud Database
67		▬	Cloud database working
68		📌	
69		📌	**Final Presentation Preparation**
70		📌	Creating Final Report
71		📌	Final Report
72		▬	
73		📌	**Project Video**
74	🟨	📌	Creating video
75		📌	Video Submission
76		📌	
77		▬	**Project Poster**
78		📌	Poster submission
79		📌	Final Presentation

Figure 6-1 Project Plan

7 Sensors

The sensors I used on the project are pressure sensor, magnetometer, accelerometer and gyroscope, and a GPS receiver. Using all these combined, you are able to locate the device in 3d space as well as gathering other information about its movement or systems.

7.1 Pressure Sensor

Figure 7-1 A photograph of the Pressure sensor chip

The BMP388 is a digital pressure sensor, it senses the ambient atmospheric pressure and can give a reading back in Pascals. The BMP388 also has its own temperature sensor that is used to compensate the pressure readings and for later conversion to altitude in meters by finding the local mean sea level pressure.

To get readings correct readings from the BMP388, you need to read the calibration coefficients stored on the chip. These are programmed in the factory during calibration of the chip.

```
  //  Read Trimming Coefficients from Register
  uint8_t trim_data[BMP388_CALIB_DATA_LENGHT_BYTES];
  rslt = I2C_Read_Data_Bytes(I2C_Address, BMP388_CALIB_DATA_START, trim_data,
BMP388_CALIB_DATA_LENGHT_BYTES);
```

Some of these register values are 16 bits so it is necessary to add them together, and also save some as signed integers. Datasheet tells us which are 16 bit and or signed. After reading the coefficients into an array, the datasheet also gives us formula to convert these numbers into the calibration coefficients [3]. (appendix 10.1 and 10.2 for parsing and calculation of coefficients)

```
uint8_t sensorData[BMP388_DATA_LENGHT_BYTES];
uint32_t uncompDataTemp, uncompDataPress, data_msb, data_lsb, data_xlsb;
bool rslt = false;

rslt = I2C_Read_Data_Bytes(I2C_Address, BMP388_DATA_0, sensorData, BMP388_DATA_LENGHT_BYTES);
if (!rslt)
  return COMM_ERR;

data_xlsb = sensorData[0];
data_lsb = sensorData[1] << 8;
data_msb = sensorData[2] << 16;
uncompDataPress = data_msb | data_lsb | data_xlsb;

data_xlsb = sensorData[3];
data_lsb = sensorData[4] << 8;
data_msb = sensorData[5] << 16;
uncompDataTemp = data_msb | data_lsb | data_xlsb;

float compTemp = BMP388_Compensate_Temperature(uncompDataTemp);
float compPress = BMP388_Compensate_Pressure(uncompDataPress, compTemp);

*sensData = compTemp;
*(sensData + 1) = compPress;
```

To get the correct pressure out of the sensor, we read the register that stores this data, which is 3 bytes. Add those together and then send it to be compensated. The compensate pressure function takes the uncompensated pressure value and multiplies with the calibration coefficients and the compensated temperature value. The exact formulas are stated in the datasheet.

7.2 Magnetometer

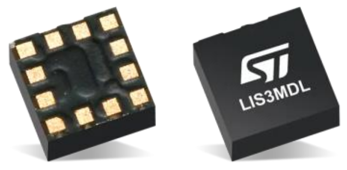


Figure 7-2 A photograph of the Magnetometer chip

The LIS3MDL is a three-axis magnetic sensor, it measure the magnitude of magnetism or geomagnetism. This sensor returns each axis as 2 bytes values in gauss. The use of this could be as a magnetometer or compass, it can tell us which direction the drone is facing. Using this sensor is very easy,

First, we need to send all the configuration parameters that we want to the chip, the last configuration also enables the chip to start taking continuous measurements.

```
rslt = I2C_Send_Data(I2C_Address, LIS3MDL_CTRL_REG2, LIS3MDL_4_GAUSS_FS);
rslt = I2C_Send_Data(I2C_Address, LIS3MDL_CTRL_REG1, LIS3MDL_TEMP_DISABLE | LIS3MDL_XY_UH_PERFORMANCE | LIS3MDL_800MS_ODR);
rslt = I2C_Send_Data(I2C_Address, LIS3MDL_CTRL_REG4, LIS3MDL_Z_UH_PERFORMANCE);
rslt = I2C_Send_Data(I2C_Address, LIS3MDL_CTRL_REG5, LIS3MDL_BLOCK_DATA_UPDATE);
rslt = I2C_Send_Data(I2C_Address, LIS3MDL_CTRL_REG3, LIS3MDL_CONTINUOUS_CONVERSION);
```

To get readings from the sensor, we need to check our status register if it’s clear, and then read our mag data into the array. To get the correct reading, the data needs to be divided by the sensitivity chosen when configuring the sensor.

```
uint8_t mOutData[LIS3MDL_MAG_DATA_LENGHT] = { 0 };

uint8_t statusReg = 0;
uint8_t* statPtr = &statusReg;
bool rslt = false;

rslt = I2C_Read_Data_Bytes(I2C_Address, LIS3MDL_STATUS_REG, statPtr, ONE_BYTE);
if (!rslt)
  return COMM_ERR;

if (statusReg & NEW_MAG_DATA  == NEW_MAG_DATA) {

  rslt = I2C_Read_Data_Bytes(I2C_Address, LIS3MDL_MAG_DATA_START, mOutData, LIS3MD
L_MAG_DATA_LENGHT);
  if (rslt) {

    int16_t mag_X = (int16_t)mOutData[1] << 8 | mOutData[0];
    int16_t mag_Y = (int16_t)mOutData[3] << 8 | mOutData[2];
    int16_t mag_Z = (int16_t)mOutData[5] << 8 | mOutData[4];

    *(sensData + 0) = (float)mag_X / GAUSS_SENS_4;
    *(sensData + 1) = (float)mag_Y / GAUSS_SENS_4;
    *(sensData + 2) = (float)mag_Z / GAUSS_SENS_4;
  }
}
```

7.3 Accelerometer and Gyroscope

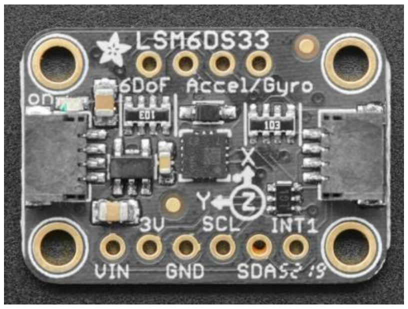


Figure 7-3 A photograph of the Accelerometer and Gyroscope test board

The LSM6DS33 is a dual sensor chip, it has an accelerometer and a gyroscope. The accelerometer is used for measure acceleration of the drone, but the measurement is converted so that it is in relation to gravity (g)

```
gyro_X = (int16_t)agOutData[1] << 8 | agOutData[0];
gOUT_X = (float)gyro_X * ANGULAR_RATE_SENSITIVITY_250MDPS * DEG_TO_RADS / 1000;
*(sensData + 0) = gOUT_X;

gyro_Y = (int16_t)agOutData[3] << 8 | agOutData[2];
gOUT_Y = (float)gyro_Y * ANGULAR_RATE_SENSITIVITY_250MDPS * DEG_TO_RADS / 1000;
*(sensData + 1) = gOUT_Y;

gyro_Z = (int16_t)agOutData[5] << 8 | agOutData[4];
gOUT_Z = (float)gyro_Z * ANGULAR_RATE_SENSITIVITY_250MDPS * DEG_TO_RADS / 1000;
*(sensData + 2) = gOUT_Z;

accel_X = (int16_t)agOutData[7] << 8 | agOutData[6];
aOUT_X = (float)accel_X * LINEAR_ACCEL_SENSITIVITY_2MG / 1000;
*(sensData + 3) = aOUT_X;

accel_Y = (int16_t)agOutData[9] << 8 | agOutData[8];
aOUT_Y = (float)accel_Y * LINEAR_ACCEL_SENSITIVITY_2MG / 1000;
*(sensData + 4) = aOUT_Y;

accel_Z = (int16_t)agOutData[11] << 8 | agOutData[10];
aOUT_Z = (float)accel_Z * LINEAR_ACCEL_SENSITIVITY_2MG / 1000;
*(sensData + 5) = aOUT_Z;
```

7.4 GPS receiver

Figure 7-4 Photograph of the SIM28 GPS receiver chip

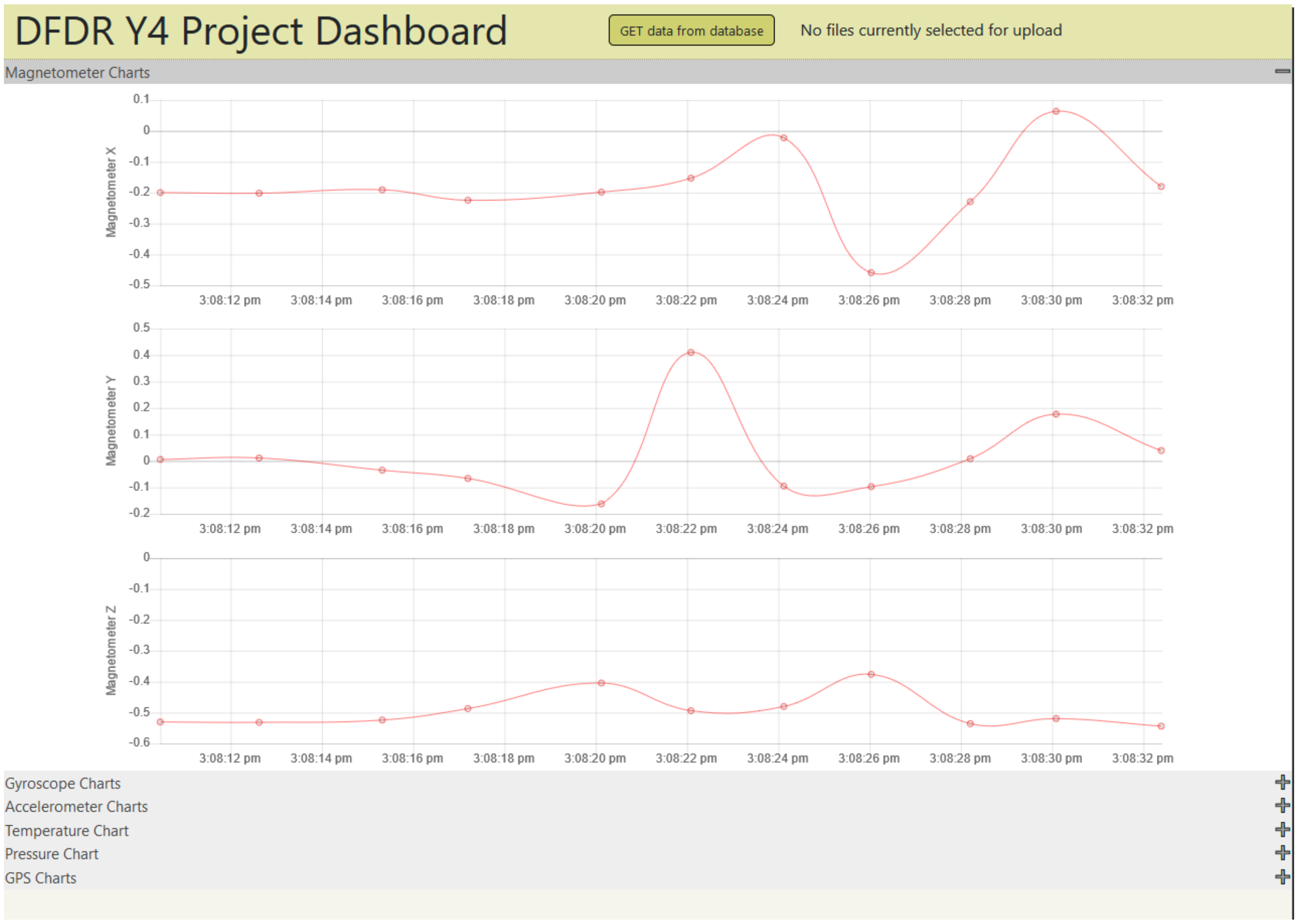
The SIM28 GPS receiver is used to get the location of the drone as well as the speed. Due to this GPS chip defaulting to 9600 baud rate, which is a little slow, during configuration we connect to the chip at 9600 and tell it to switch to 57600 (the highest I was able to get it to run)

```
gpsSerial.begin(9600);                  // the SoftSerial baud rate
delay(250);

gpsSerial.println(PMTK_SET_BAUD_57600);

delay(20);
gpsSerial.end();
delay(20);
gpsSerial.begin(57600);
delay(20);

gpsSerial.println(PMTK_TEST);
```

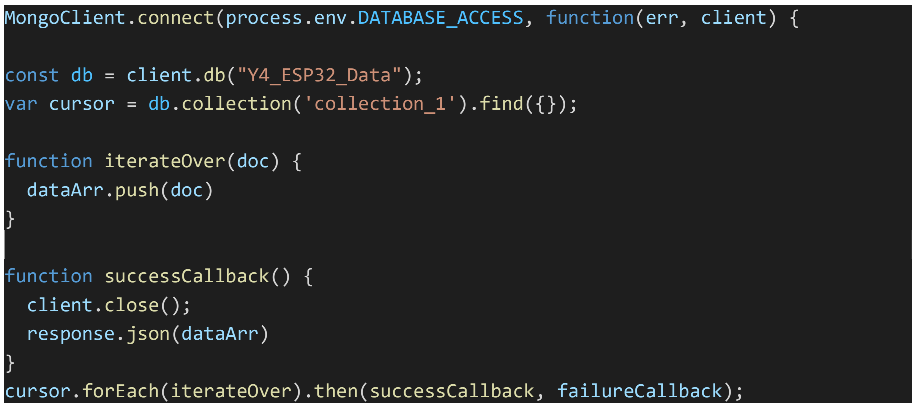
I also set the GPS to only send the NMEA - RMC string, the recommended minimum specific GPS data, and set the output rate at 1Hz

```
gpsSerial.println(PMTK_SET_NMEA_OUTPUT_RMCONLY);

gpsSerial.println(PMTK_SET_NMEA_UPDATE_1HZ);
```

7.5 Webpage



The webpage is made using HTML and JavaScript. It uses Flexbox layout. The button on the top to send a GET request to the backend. When the backend receives the request, it connects to the MongoClient and gets all items inside the database and forwards to the frontend.



```
MongoClient.connect(process.env.DATABASE_ACCESS, function(err, client) {

const db = client.db("Y4_ESP32_Data");
var cursor = db.collection('collection_1').find({});

function iterateOver(doc) {
  dataArr.push(doc)
}

function successCallback() {
  client.close();
  response.json(dataArr)
}
cursor.forEach(iterateOver).then(successCallback, failureCallback);
```

The response data gets parsed into json data and stored into an array. It is stored as an array of two objects, the first object is the UTC time object and the second contains the data. Therefore data[0] is always the timestamp

```
let allData = [magX, magY, magZ, gyrX, gyrY, gyrZ, accX, accY, accZ, temp,

let chartElements = [
    'magnetometerChartX', 'magnetometerChartY','magnetometerChartZ',
    'gyroscopeChartX', 'gyroscopeChartY','gyroscopeChartZ',
    'accelerometerChartX', 'accelerometerChartY', 'accelerometerChartZ',
    'temperatureChart', 'pressureChart', 'gpsChart'
    ]

let chartLabels = [
    'Magnetometer X', 'Magnetometer Y', 'Magnetometer Z',
    'gyroscope X', 'gyroscope Y', 'gyroscope Z',
    'Accelerometer X', 'AccelerometeY', 'Accelerometer Z',
    'Temperature (C)', 'Pressure (Pa)', 'GPS '
    ]
```

To create the charts, we use chartJs. We can simplify the creation of the chart by defining the chart options and how chart data will be displayed elsewhere. We then only need to tell that we want a line chart, with the label on the left of the chart, the timestamps to be put on the-X axis and data on the Y-axis

```
    let chart1 = document.getElementById(chartElements[i]).getContext('2d');

    new Chart(chart1, {
      type: 'line',
      data: chartData(chartLabels[i], data[0], data[i]),
      options: chartOptions(chartLabels[i])
    });
```

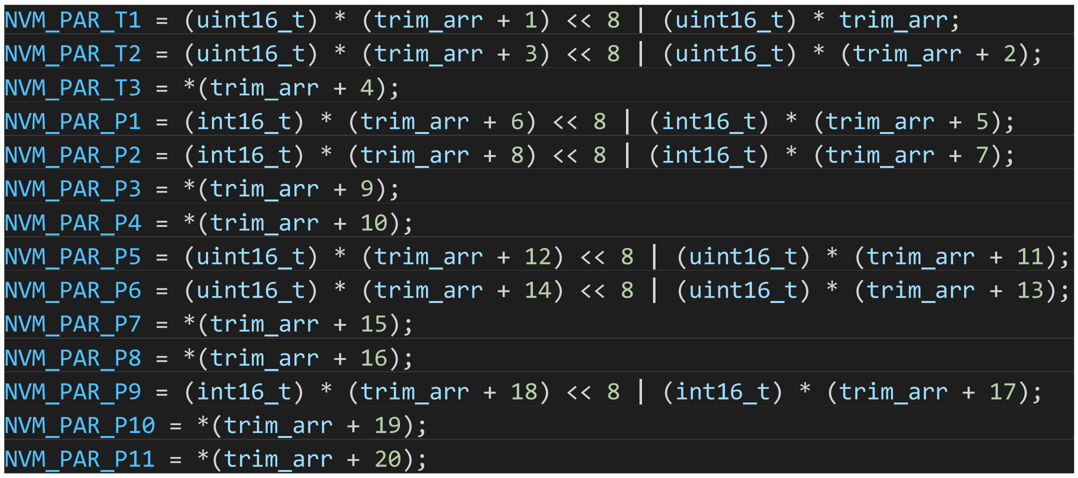
8 Conclusion

In the conclusion of the project, there is a breadboard prototype of the sensors connected to a main board that can send its results over Wi-Fi. There is a functioning webpage that can take in these results and store them in an external database. When clicked the webpage can also retrieve all entries in this database and display on the webpage as line charts.

In future I would like to add a way to animate the data over a 3d model, replicating the movement of the drone through simulation.

9 Appendix

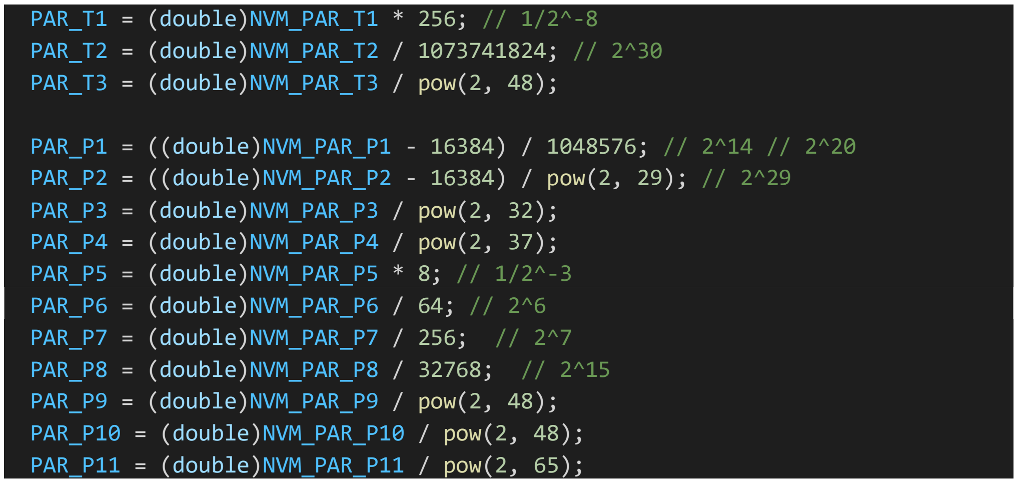
9.1 Calibration Coefficient Parsing



```
NVM_PAR_T1 = (uint16_t) * (trim_arr + 1) << 8 | (uint16_t) * trim_arr;
NVM_PAR_T2 = (uint16_t) * (trim_arr + 3) << 8 | (uint16_t) * (trim_arr + 2);
NVM_PAR_T3 = *(trim_arr + 4);
NVM_PAR_P1 = (int16_t) * (trim_arr + 6) << 8 | (int16_t) * (trim_arr + 5);
NVM_PAR_P2 = (int16_t) * (trim_arr + 8) << 8 | (int16_t) * (trim_arr + 7);
NVM_PAR_P3 = *(trim_arr + 9);
NVM_PAR_P4 = *(trim_arr + 10);
NVM_PAR_P5 = (uint16_t) * (trim_arr + 12) << 8 | (uint16_t) * (trim_arr + 11);
NVM_PAR_P6 = (uint16_t) * (trim_arr + 14) << 8 | (uint16_t) * (trim_arr + 13);
NVM_PAR_P7 = *(trim_arr + 15);
NVM_PAR_P8 = *(trim_arr + 16);
NVM_PAR_P9 = (int16_t) * (trim_arr + 18) << 8 | (int16_t) * (trim_arr + 17);
NVM_PAR_P10 = *(trim_arr + 19);
NVM_PAR_P11 = *(trim_arr + 20);
```

We need to store some of the register values in the array as 16 bits, so we shift the upper 8 bits and or the lower 8 to them.

9.2 Convert register values to calibration coefficients



```
  PAR_T1 = (double)NVM_PAR_T1 * 256; // 1/2^-8
  PAR_T2 = (double)NVM_PAR_T2 / 1073741824; // 2^30
  PAR_T3 = (double)NVM_PAR_T3 / pow(2, 48);

  PAR_P1 = ((double)NVM_PAR_P1 - 16384) / 1048576; // 2^14 // 2^20
  PAR_P2 = ((double)NVM_PAR_P2 - 16384) / pow(2, 29); // 2^29
  PAR_P3 = (double)NVM_PAR_P3 / pow(2, 32);
  PAR_P4 = (double)NVM_PAR_P4 / pow(2, 37);
  PAR_P5 = (double)NVM_PAR_P5 * 8; // 1/2^-3
  PAR_P6 = (double)NVM_PAR_P6 / 64; // 2^6
  PAR_P7 = (double)NVM_PAR_P7 / 256;  // 2^7
  PAR_P8 = (double)NVM_PAR_P8 / 32768;  // 2^15
  PAR_P9 = (double)NVM_PAR_P9 / pow(2, 48);
  PAR_P10 = (double)NVM_PAR_P10 / pow(2, 48);
  PAR_P11 = (double)NVM_PAR_P11 / pow(2, 65);
```

Now we need to convert the calibration coefficients into floating point numbers

10 References

[1] “Flight Recorder,” [Online]. Available: https://en.wikipedia.org/wiki/Flight_recorder.

[2] F. AviationAdministration, "AIRWORTHINESS AND OPERATIONAL APPROVAL OF DIGITAL FLIGHT DATA RECORDER SYSTEMS," [Online]. Available: https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC20-141.pdf.

[3] BOSCH, "BMP388 DATASHEET," [Online]. Available: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp388-ds001.pdf.