Final Report

Takura Kawome and Dean Makoni

October 2021

1 Introduction

This report describes the steps, experiments and results that were taken to design an ARM based digital IP using Raspberry-Pi to encrypt and compress IMU data from SHARC buoy.

The first section of the report talks about system requirements, deriving the system specifications from the requirements and mapping these specifications to acceptance test criteria of the system. The acceptance test procedures are used to validate system functionality.

The second section is about paper design. The paper design section describes the flow of the overall system and gives details of the design of the different sub-systems and how they interact. Included in the paper design section is the feasibility analysis as well as the possible bottlenecks.

The following two sections, from the paper design section, describe the validation of the system using simulated data and using data from the ICM-20648. These two sections describe the experiments done to fulfill the acceptance test procedures and the results of these tests to validate functionality of the system.

The last section is about the future work that can be done to improve the system.

2 Requirement Analysis

This section discusses the requirements of the system.

2.1 User Requirements

The user requirements were obtained from the problem statement that was given as part of the project. The user requirements describe the objectives that the design is supposed to meet.

Table 1: User Requirements

Requirement ID	Description
UR001	System must make use of the ICM-20649
UR002	System must be able to to retain at least 25% of the Fourier Coefficient of the data
UR003	System must do the minimum processing to save the battery
UR004	System must use Raspberry pi to compress and to encrypt the data

2.2 Analysis of User Requirements

2.2.1 Analysis of UR001

System must make use of the ICM-20649

The ICM-20649 is a wide-range 6-axis motion tracking device for sports and other high impact applications. It has a gyroscope and accelerometer which are used to get information about the ice and wave dynamics.

Features

- 3-Axis gyroscope with programmable FSR of ±500 dps, ±100 dps, ±2000 dps, and ±4000 dps
- 3-Axis accelerometer with programmable FSR of ±4g, ±8g, ±16g, and ±30g

The Marinebio.org(2021), report that the temperatures recorded at the Southern Ocean ranges from -2^0 C to 10^0 C. The ICM-20649 has an operating temperature range of -40^0 C to 85^0 C making it suitable for this application.

2.2.2 Analysis of UR002

System must be able to to retain at least 25% of Fourier coefficients of the data

Compressing a file reduces data size and enables the file to be sent and received quickly. This allows us to get the best out of the processing power and storage space. However, there can be a trade-off of data loss when decompressing the compressed file. The chosen compression algorithm should be able to extract at least 25% of the lower fourier coefficients of the data.

2.2.3 Analysis of UR003

System must do the minimum processing

Power of the on-board processor is limited thus the need to minimise the amount of computation done by the system. The chosen compression and encryption algorithms should be efficient such that they do not require high processing power.

2.2.4 Analysis of UR004

System must use an on-board ARM based processor to compress and encrypt the data

The on-board ARM based processor that is going to be used is a raspberry pi. The IMU will be connected to the raspberry pi which will be used to read data from the IMU. A compression and encryption code will be loaded on the raspberry pi which will be used to compress then encrypt the read data from the IMU before transmitting it.

2.3 Compression Subsystem Requirements and Specifications

2.3.1 Functional Requirement(s) and Specification(s)

Table 2: Functional Requirement(s) and Specification(s) of compression subsystem

x **Requirement**	**Specification**
RMO1 S01-The subsystem shall be able to read raw data from the IMU.	**SP01 S01**-The Raspberry pi will communicate with the IMU using the I2C interface in order to receive the data
RMO2 S01-The algorithm shall be able to compress the raw data from IMU	**SP02 S01**-We shall use the huffman-algorithm to reduce the file size containing data from the IMU.

2.3.2 Performance Requirement(s) and Specification(s)

Table 3: Performance Requirement(s) and Specification(s) of compression subsystem

Requirement	Specification
RMO3 S01-The algorithm shall compress the data so that we can retrieve at least 25% of the fourier coefficients.	**SP03 S01**-The compression algorithm should account for at least 25% of the lower fourier coefficients if there is data loss at decompression.

2.3.3 General Constraint(s) and Specification(s)

Table 4: General constraints and Specification(s) of compression subsystem

Requirement	Specification
RMO4 S01-The subsystem shall do minimum computations to save power	**SP04 S01**-CPU usage when compression occurs should be low.

2.4 Encryption Subsystem Requirements and Specifications

2.4.1 Functional Requirement(s) and Specification(s)

Table 5: General constraints and Specification(s) of compression subsystem

Requirement	Specification
RMO1 S02-The subsystem must be able to encrypt the data from the IMU	**SP01 S02**-AES encryption algorithm will be used.
RMO2 S02-The subsystem shall be able to receive data from the compression subsystem	**SP01 S02**-Use Python scripts to handle data being sent between the subsystems

2.4.2 Performance Requirement(s) and Specification(s)

Table 6: Performance Requirement and Specification(s) of encryption subsystem

Requirement	**Specification**
RMO3 S02-The subsystem shall be able to encrypt the data with minimum possible logic steps.	**SP03 S02**-AES encryption library will be used which is more efficient.

2.4.3 General Constraint(s) and Specification(s)

Table 7: General constraints and Specification(s) of compression subsystem

Requirement	**Specification**
RMO4 S02-The subsystem shall do minimum computations to save power	**SP04 S02**-AES encryption library will be used which is more efficient.

3 Acceptance Test Procedure

Table 8: Figure of merits derived from specifications

Specification	**Figure of merit**
SP01 S01-The huffman-algorithm will be used to compress the data from the IMU.	the algorithm is supposed to reduce the file size containing the data from the IMU.
SPO2 S01-The Raspberry pi will communicate with the IMU using the I2C interface in order to receive the data	Print out data from the IMU to prove retrieval
SPO3 S01-The compression algorithm should account for at least 25% of the lower fourier coefficients if there is data loss at decompression.	File contents after decompression should be almost identical to original file contents.
SPO4 S01-The adaptive mode of Zstandard is supposed to be set to the minimum required.	Fastest time of execution
SPO1 S02-AES encryption library will be used.	Decrypt the encrypted data and compare it with the raw data.
SPO1 S02-Use Python scripts to handle data being sent between the subsystems	Data files need to be transmitted properly without fault.
SPO2 S02-AES encryption library will be used which is more efficient.	Fastest time of execution

4 Paper Design

4.1 Comparison of compression algorithms

Table 9: Comparison of compression algorithms

Compression Algorithm	Compression Ratio	Compression Speed(MBPs)	Decompression Speed(MBPs)
lz4	2.10	444.69	2165.93
ztsd	3.14	136.18	536.36
zlib	3.11	23.21	281.52
xz	4.31	2.37	62.97
gzip	19.66	125.55	2165.93

As indicated in the table above, there is a compromise between the compression speed and compression ratio. Iz4 with the fastest speed has a lower compression ratio and xz with the largest compression ratio has the slowest speed. Zstd however shows a good balance between speed and compression ratio, therefore, making it a more favourable algorithm. [1]

4.2 Comparison of Encryption Algorithms

Table 10: Comparison of encryption algorithms

Encryption Algorithm	Decryption	Pros	Cons
Data Encryption-Standard (DES)	It is a bit-oriented symmetric encryption algorithm with a key length of 56 bits.		It is easy to hack.
Triple DES	Uses 3 keys with 56 bits length each. The total length of the keys is 112 bits	More secure than the standard DES encryption	It is still easy to break the encryption
RSA encryption	3Used for data sent over the internet. Asymmetric Algorithm	More secure than DES	Asymmetric algorithms are slower than symmetric algorithms
Advanced Encryption (AES)	It is a byte-oriented symmetric encryption algorithm with various key lengths; 128-bit, 192-bit and 256-bit	It can not be hacked. The most secure of the four Symmetric encryption execute faster than asymmetric	

The longer bit lengths of AES and RSA make them more secure than DES. AES encryption algorithm offers more security and high performance hence making it a more suitable choice for the IP.

4.3 Compression Subsystem Requirements and Specifications

4.3.1 Functional Requirement(s) and Specification(s)

Table 11: Functional Requirement(s) and Specification(s) of compression subsystem

x **Requirement**	**Specification**
RMO1 S01-The subsystem shall be able to read raw data from the IMU.	**SP01 S01**-The Raspberry pi will communicate with the IMU using the I2C interface in order to receive the data
RMO2 S01-The algorithm shall be able to compress the raw data from IMU	**SP02 S01**-We shall use the huffman-algorithm to reduce the file size containing data from the IMU.

4.3.2 Performance Requirement(s) and Specification(s)

Table 12: Performance Requirement(s) and Specification(s) of compression subsystem

Requirement	**Specification**
RMO3 S01-The algorithm shall compress the data so that we can retrieve at least 25% of the fourier coefficients.	**SP03 S01**-The compression algorithm should account for at least 25% of the lower fourier coefficients if there is data loss at decompression.

4.3.3 General Constraint(s) and Specification(s)

Table 13: General constraints and Specification(s) of compression subsystem

Requirement	**Specification**
RMO4 S01-The subsystem shall do minimum computations to save power	**SP04 S01**-CPU usage when compression occurs should be low.

4.4 Encryption Subsystem Requirements and Specifications

4.4.1 Functional Requirement(s) and Specification(s)

Table 14: General constraints and Specification(s) of compression subsystem

Requirement	Specification
RMO1 S02-The subsystem must be able to encrypt the data from the IMU	**SP01 S02**-AES encryption algorithm will be used.
RMO2 S02-The subsystem shall be able to receive data from the compression subsystem	**SP01 S02**-Use Python scripts to handle data being sent between the subsystems

4.4.2 Performance Requirement(s) and Specification(s)

Table 15: Performance Requirement and Specification(s) of encryption subsystem

Requirement	Specification
RMO3 S02-The subsystem shall be able to encrypt the data with minimum possible logic steps.	**SP03 S02**-AES encryption library will be used which is more efficient.

4.4.3 General Constraint(s) and Specification(s)

Table 16: General constraints and Specification(s) of compression subsystem

Requirement	Specification
RMO4 S02-The subsystem shall do minimum computations to save power	**SP04 S02**-AES encryption library will be used which is more efficient.

4.5 Feasibility Analysis

4.5.1 Economic Feasibility

The project does not have a budget as most of the hardware requirements (Raspberry) Pi 0W, PC laptops) and software requirements (MatLab, Python, C) are available for free.

4.5.2 Operational Feasibility

The proposed system appears to solve the problems in the problem statement of the project. This system will allow for the easy, fast and cheap transmission of data from the IMU. Compressing the data will make it faster and easier to transmit data from the onboard IMU to the user. Encrypting the data will ensure the security of the transmitted data.

4.5.3 Schedule Feasibility

Given the manpower and complexity of the project, that is what needs to be done as indicated in the development timeline, the project can be done to completion within the given time of the semester.

4.6 Possible Bottlenecks

- Optimising the algorithms to reduce computation to the minimum might compromise the integrity of the data from the sensor.

4.7 Subsystem Design

4.7.1 Inter sub-System and Inter -Sub-subsystems Interaction

The compression subsytem gets raw data from the IMU and compresses it. The encryption subsystem gets the data from the compression subsystem and encrypts it before the data is transmitted.

4.7.2 Block Diagram

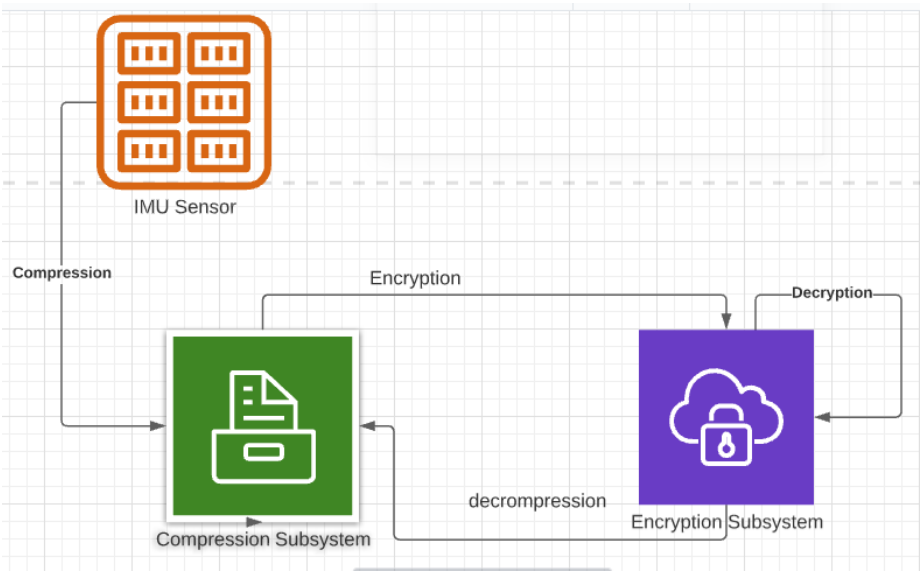

Figure 1: Block Diagram of the whole System

4.8 Acceptance Test Procedure

Table 17: Figure of merits derived from specifications

Specification	**Figure of merit**
SP01 S01-The huffman-algorithm will be used to compress the data from the IMU.	the algorithm is supposed to reduce the file size containing the data from the IMU.
SPO2 S01-The Raspberry pi will communicate with the IMU using the I2C interface in order to receive the data	Print out data from the IMU to prove retrieval
SPO3 S01-The compression algorithm should account for at least 25% of the lower fourier coefficients if there is data loss at decompression.	File contents after decompression should be almost identical to original file contents.
SPO4 S01-The adaptive mode of Zstandard is supposed to be set to the minimum required.	Fastest time of execution
SPO1 S02-AES encryption library will be used.	Decrypt the encrypted data and compare it with the raw data.
SPO1 S02-Use Python scripts to handle data being sent between the subsystems	Data files need to be transmitted properly without fault.
SPO2 S02-AES encryption library will be used which is more efficient.	Fastest time of execution

4.9 Experiment Design to test figures of merit

We are going to test SPO1 S01 by compressing the data file and then measure the size of the file after the compression. If the size of the compressed file is less than the raw data it means our compression subsystem is working as expected. For SP03 01, we are going to compress the raw data from the simulations and then encrypt it. After the encryption, we are going to decrypt the data file and compare the data to the raw data before compression was done. To test SP01 S02 we are going to compress our data then encrypt it. We are going to use decryption and decompression and analyse our output data. If we can get at least 25% of the raw data it means our subsystems are working as expected. To test the speed of our encryption algorithm SPO3 S02, we are going to use

the time library for the C program to measure the time taken to execute the encryption algorithm.

5 Validation using Simulated Data

To validate the design, data simulated using Simulink was used.Simulation-based validation makes it easy to optimise the data that is being collected, different scenarios can be tested that would be impossible to test with the actual hardware. Simulation makes it is easy to add noise to the data and then check how our subsystem performs with the data that has noise to it; this will be difficult to do with the actual IMU because our movements are only limited to hand movements. Simulation-based validation also makes it easy to run the experiment for a prolonged time and collect larger data samples that we can use to test our system; it is not easy to run the do this with the actual IMU because we can only do hand movements and it becomes tiring to collect data for a prolonged time.The behaviour of the gyroscope and the accelerometer can be modelled to match the actual sensors that are on ICM-20649.

5.1 Steps taken to collect simulated data

- To simulate the data, we used IMU Sensor Fusion with Simulink. The Simulink model is shown below.

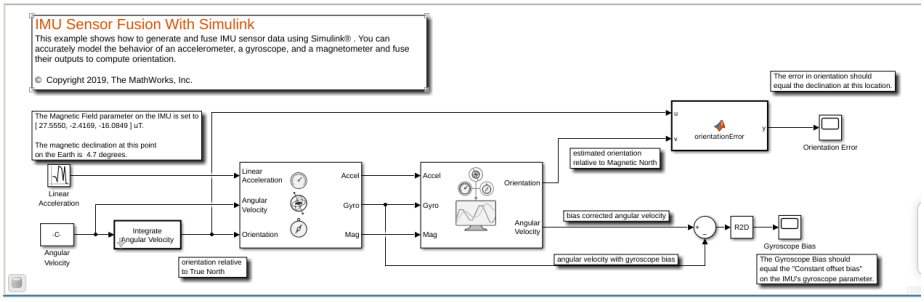

Figure 2: Simulink IMU Model

- parameters of the gyroscope and the accelerometer were adjusted to macth that of ICM-20649
- The simulations were run for different times and the input was adjusted to meet certain conditions eg a gaussian noise was added.
- The data collected from the IMU was saved in a csv file and send to raspberry pi.
- UDP connection was done between the PC and Raspberry Pi to simulate the data transfer from the IMU to the raspberry pi. This method of

transfer was chosen because UDP is simpler, data is transferred faster and there is more control of when data is sent out.

- The raspberry pi was simulated as the server and the PC the client.
- Different delays were used to observe how the raspberry pi reacts to receiving data at various speeds and observe if there was any data loss with the increase in speed of transmission by comparing the file contents of the received by the raspberry pi at different speeds.
- The raspberry pi receives the data in the form of 8-bit bytes.
- These are converted to string form and written into a text file which is sent for compression.
- The data was passed through the compression subsystem and then the encryption subsystem.
- After the compression and encryption the data was decrypted and decompressed.

5.2 Simulation Data Analysis

5.2.1 Input to the IMU sensor

The data from the accelerometer and the gyroscope was collected using Simulink. The two inputs to the IMU sensor were linear acceleration and angular velocity. A Gaussian noise was also added using Simulink AWGN channel.

5.2.2 Linear Acceleration Input to the IMU sensor

The image below shows the input of the model before a Gaussian noise was applied.

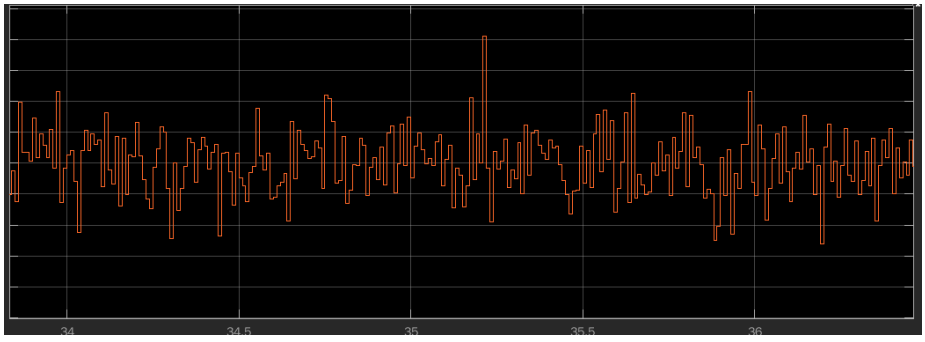

Figure 3: Linear acceleration before adding Gaussian noise

The image below shows the linear acceleration after the Gaussian noise was added.

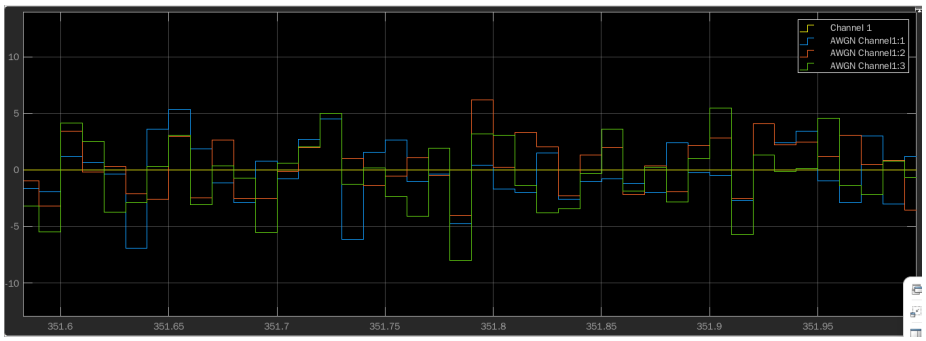

Figure 4: Linear acceleration after adding Gaussian noise

5.2.3 Angular Velocity Input to the IMU sensor

The image below shows the angular velocity input before applying a Gaussian noise.

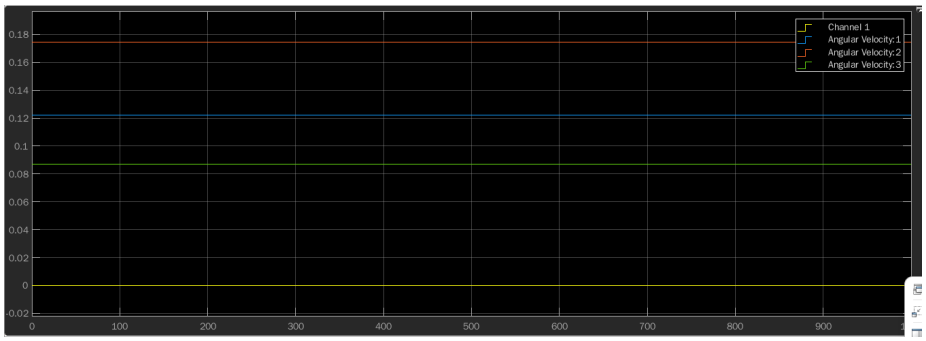

Figure 5: Angular velocity before adding Gaussian noise

The image below shows angular velocity input data after applying Gaussian noise.

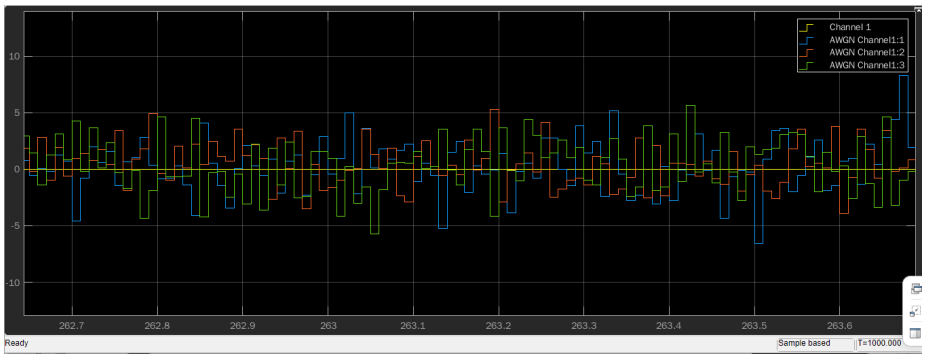

Figure 6: Angular velocity after adding Gaussian noise

Comparing Figure 4 and Figure 5 it can be seen that adding a Gaussian noise is going to randomise the input data set.

5.3 Data collected from the IMU Senor

The accelerometer and gyroscope data from the IMU sensor was collected separately.

5.3.1 Data from Accelerometer

The time domain of the data from the accelerometer is shown on the image below.It can be seen that the sinusoidal are not smooth because of the Gaussian noise that was added to the inputs of the IMU sensor.

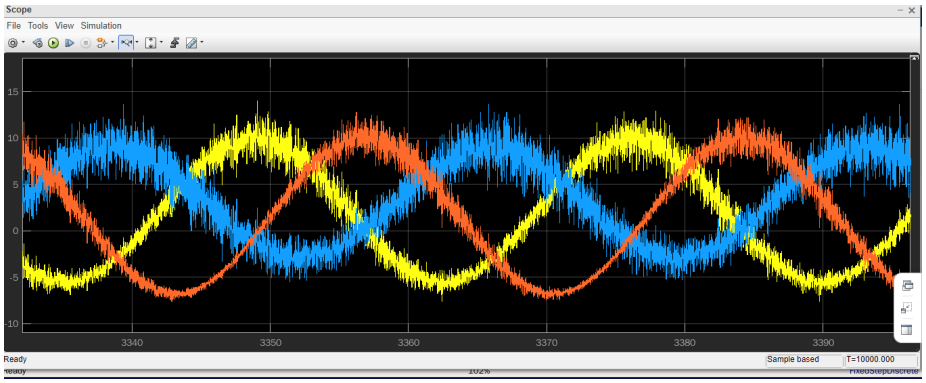

Figure 7: Time Domain of the data from the accelerometer

The frequency domain of the data collected from the accelerometer is shown on the image below.

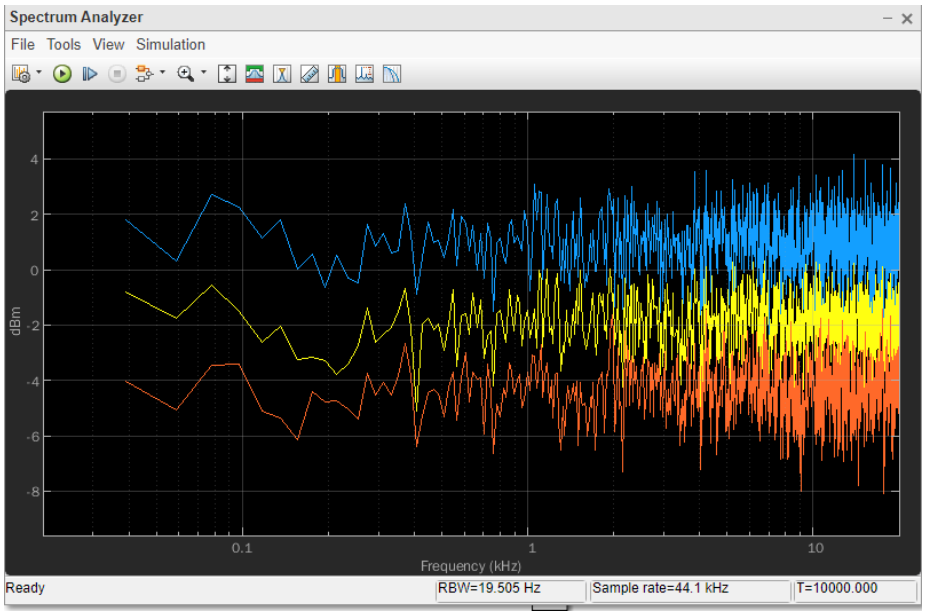

Figure 8: Frequency Domain of the data from the accelerometer

5.3.2 Data from the Gyroscope

The time and frequency domain plots of the data from the gyroscope are shown.

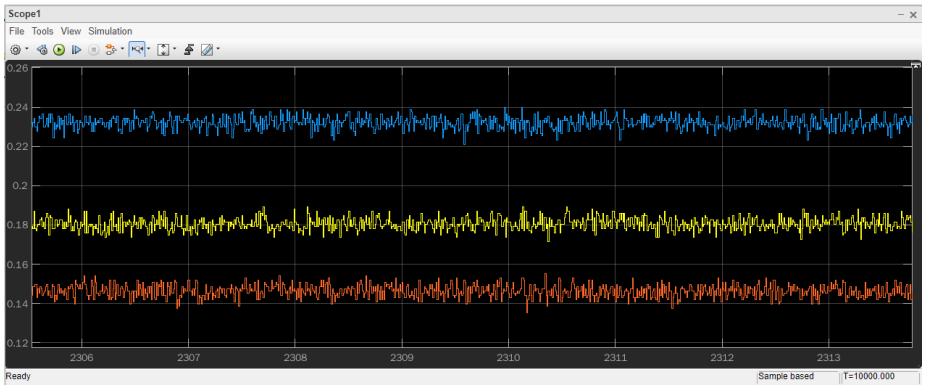

Figure 9: Time Domain of the data from the gyroscope

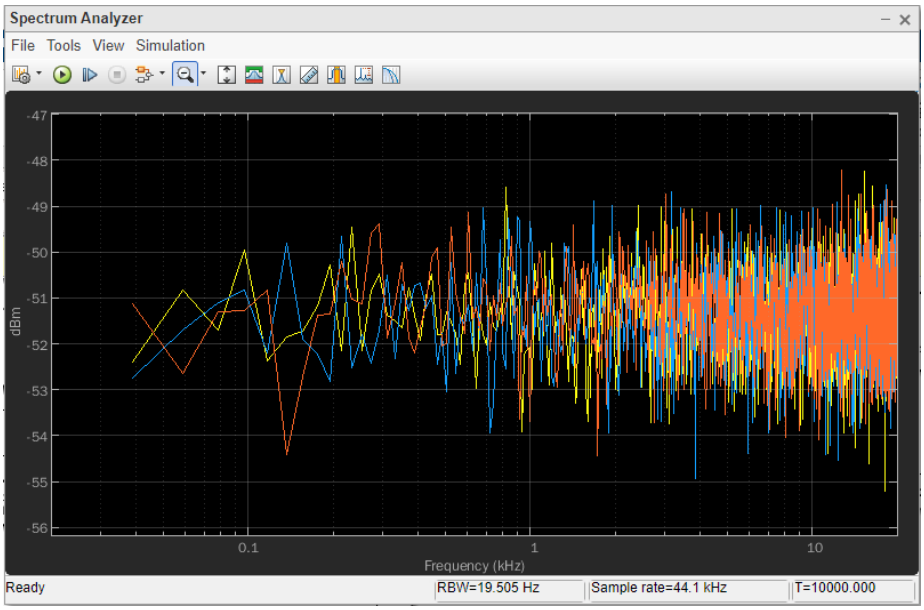

Figure 10: Frequency Domain of the data from the gyroscope

5.4 Experiment Setup

5.5 System Functionality

- A UDP connection was established between the PC and Raspberry Pi to simulation the data transfer from the IMU to the raspberry pi. This method of transfer was chosen because UDP is simpler, data is transferred faster and there is more control of when data is sent out. The raspberry pi was simulated as the server and the PC the client.
- Data transmitted through the UDP connection is a set of the X, Y and Z components of the accelerometer, gyroscope and magnetometer. As data is received by the raspberru pi, it is simultaneously written into a csv file.
- Different delays were implemented when transmitting the data from the PC to observe how the raspberry pi reacts to receiving data at various speeds and observe if there was any data loss with the increase in speed of transmission by comparing the file contents received by the raspberry pi at different speeds with the original file.
- The raspberry pi receives the data in the form of 8-bit bytes. The data is converted to string form before it is written into a csv file which is sent for compression upon completion of transmission. This makes it easier to compare the original file to the final file that goes through the compression, encryption, decryption and decompression processes.

- Execution time for each process, (i.e compression, encryption, decryption and decompression), was recorded to observe efficiency of the system.

5.6 Compression Subsystem

5.6.1 Expected input and output data

The compression block receives a csv file with the readings from the accelerometer, gyroscope and magnetometer sensors. After compressing the file, the compressed data is written to a text file and sent to the encryption block. Different compression algorithms were tested. These are the huffman algorithm, lz4, zstd, zlib and gzip.

5.6.2 Performance and Power Consumption

- The compression block receives a csv file with the fourier coefficients in string form.
- Different compression algorithms were tested. These are the huffman algorithm, lz4, zstd, zlib and gzip.
- The same file was compressed using the different compression algorithms. Being tested was efficiency, that is, execution time for compression and decompression separately, and finally the compression ratio of each algorithm, which is the ratio of the file size before and after compression.
- For gzip, lz4 and zstd the compression level was varied to see the effect of speed of execution on the compression ratio and integrity of the data. This was done to determine if limiting the amount of processing done by the raspberry pi was a good trade-off for compression ratio.
- After decompression using each of the algorithms mentioned above, the file contents of the decompressed file will be compared to the original file to observe if there are any changes. As stated in the requirements, the goal is to retrieve at least 25% of the lower Fourier Transfer coefficients.

5.6.3 Comparison of original data file to final decompressed data file

- After decompression using each of the algorithms mentioned above, the file contents of the decompressed file will be compared to the original file to observe if there are any changes. As stated in the requirements, the goal is to retrieve at least 25% of the lower Fourier Transfer coefficients.

5.7 Encryption Subsystem

5.7.1 Expected input and output data

The subsystem is going to work with .txt file from the compression subsystem and it is going to encrypt the data into a cipher text file. The description of the

test vectors of this subsystem is given below.

Test Vectors

y=E(file.txt)

y is the output from the encryption subsystem inform of a text file.
E is the encryption subsystem.
file.text is the input from the compression subsystem)

5.7.2 Experiments

For this subsystem, we are going to run two types of experiments that are functionality experiments and benchmarking experiments. Functionality experiments are going to test if the subsystem is working as expected and benchmarking experiments are going to test the performance of the subsystem eg time to encrypt different file sizes. A test harness (automated test framework) is going to be used to do all the experiments.

5.7.3 Functionanlity Experiments

- To check the validity of the encrypted data a checksum data is going to be implemented using the HMAC concept (keyed-hash message authentication code). The data from the compression subsystem is going to be passed through the HMAC and then the results of the HMAC are going to be sent to the AES encryption algorithm. The data is going to be decrypted and the results are going to be compared by the HMAC result (comparing the checksum data) using a python script. The above experiment is going to be carried out for different file sizes.

- To check the validity of the system's different modes of encryption **Monte Carlo** tests were used to verify if different modes of encryption were working properly (ECB mode,CBC mode and CFB mode). The Monte Carlo test is going to check for anomalous combinations of inputs that will cause the subsystem to malfunction.

- To check if the subsystem is secure a different key is going to be used to decrypt the encrypted data and the out put is going to be compared with the data before encryption using the following command - diff file1 file2

5.7.4 Benchmarking Experiments

- The subsystem is using AES encryption which is a symmetric type, the first benchmark experiment is going to test the time that it is taken by a symmetric algorithm (AES the one our encryption subsystem is using) vs an asymmetric algorithm(RSA). The experiment is going to be run on

our raspberry pi using python with a varying file size. The experiment is going to be repeated 10 times for each algorithm and the average time is going to be recorded. To measure the time, a timing module in python is going to be used.

- The second AES experiment is going to measure the amount of time taken by the AES algorithm to encrypt different file sizes. The experiment is going to be repeated 10 times for each file size and the average time is going to be recorded.
- The third experiment is going to test the average time taken by AES-128, AES-192 and AES-256 to encrypt a file size of 212 MB. The experiment is going to be repeated 10 times and the average time is going to be calculated. To record the running time, a time module in python is going to be used.

5.8 Results

5.9 System Functionality

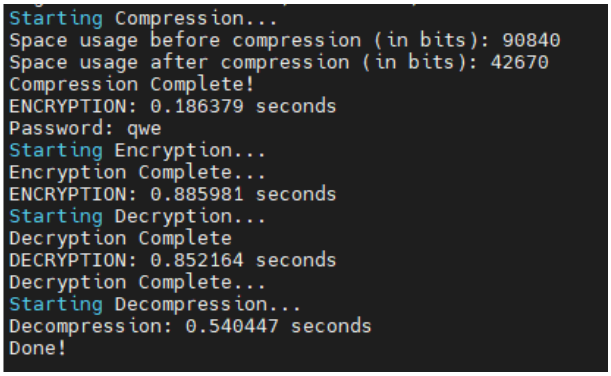

Figure 11: Overall system performance

Table 18: System performance shown in Figure 11

Process	**Execution time (s)**
Compression	0.186379
Encryption	0.885901
Decryption	0.852164
Decompression	0.548447

The whole system was automated, from reading the data from the sensors, to writing it to a csv file, to sending the file to the compression block to be

compressed, to sending the compressed file to the encryption block to encrypted, followed by decryption then finally decompression.
Table 2 above shows the execution time of each sub-sytem. The total execution time was 2.472891 seconds for a file size of 165KB.

5.10 Compression Subsystem

5.10.1 Comparing performance of the different compression algorithms

Table 19: Comparing best performance of each compression algorithm

Compression Algorithm	Compression Ratio	Compression Speed(MBps)	Decompression Speed(MBps)
huffman	2.03	0.094	0.034
gzip	2.62	0.497	0.588
zlib	3.22	1.99	14.21
zstd	2.22	0.331	0.446
lz4	1.69	0.479	0.539

Initially, zstd compression was the chosen compression algorithm for the subsystem. Based on the results of the experiment obtained, shown in *Table 1* above, the algorithm with the highest performance and low amounts of processing was zlib. This observation has changed the initial decision to use zstd and instead use zlib for the system.

Table 20: Comparing performance of each compression algorithm at maximum vs. minimum compression level

Compression Algorithm	Compression Ratio (min compression level)	Compression Ratio (max compression level)	Compression Speed(MBps) (min compression level)	Compression Speed(MBps) (max compression level)
gzip	2.62	3.27	0.588	0.315
zstd	2.22	4.42	0.331	0.122
lz4	1.69	2.21	0.479	0.35

Since the requirements did not state the importance of achieving the maximum possible compression, but rather stresses the need to minimise the amount of processing done by the algorithms, reducing the compression level for high performance of the algorithm is a good trade-off hence the selection of the minimum compression level.

5.11 Encryption Subsystem

The following tables shows the results from the functionality experiments.

Table 21: Results obtained from checking the checksum data after enycrypting files with different sizes using AES -128

File size of the encrypted data	Percentage match
10.4 kB	100%
28.6 kB	100%
212 kB	100%
8.65 MB	100%

The results above show that the subsystem was able to encrypt files with different sizes successfully.

Monte Carlo Results The expected results of the first iteration is given below:

KEY = 9dc2c84a37850c11699818605f47958c

IV = 256953b2feab2a04ae0180d8335bbed6

PLAINTEXT = 2e586692e647f5028ec6fa47a55a2aab

CIPHERTEXT = 1b1ebd1fc45ec43037fd4844241a437f

The results that were obtained after running the first iteration were saved in a file and are shown on the image below.

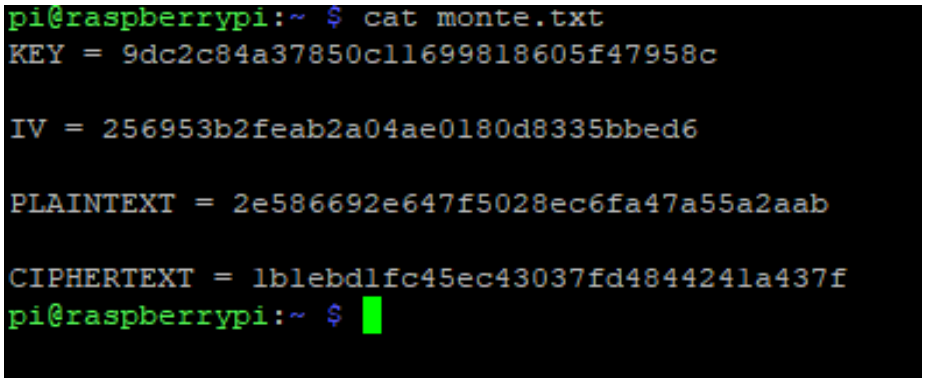

Figure 12: Monte Carlo First Iteration Results

It can be seen that the results of the first monte carlo iteration are the same as the expected vector results, this can be snowballed to other iterations. This shows that the subsystem is working as expected even in the eventuality of anomalous data.

The following table shows results from the benchmark experiments.

Power(CPU) Usage Results

The image below shows a sample of results that were obtained when power consumption was measured.

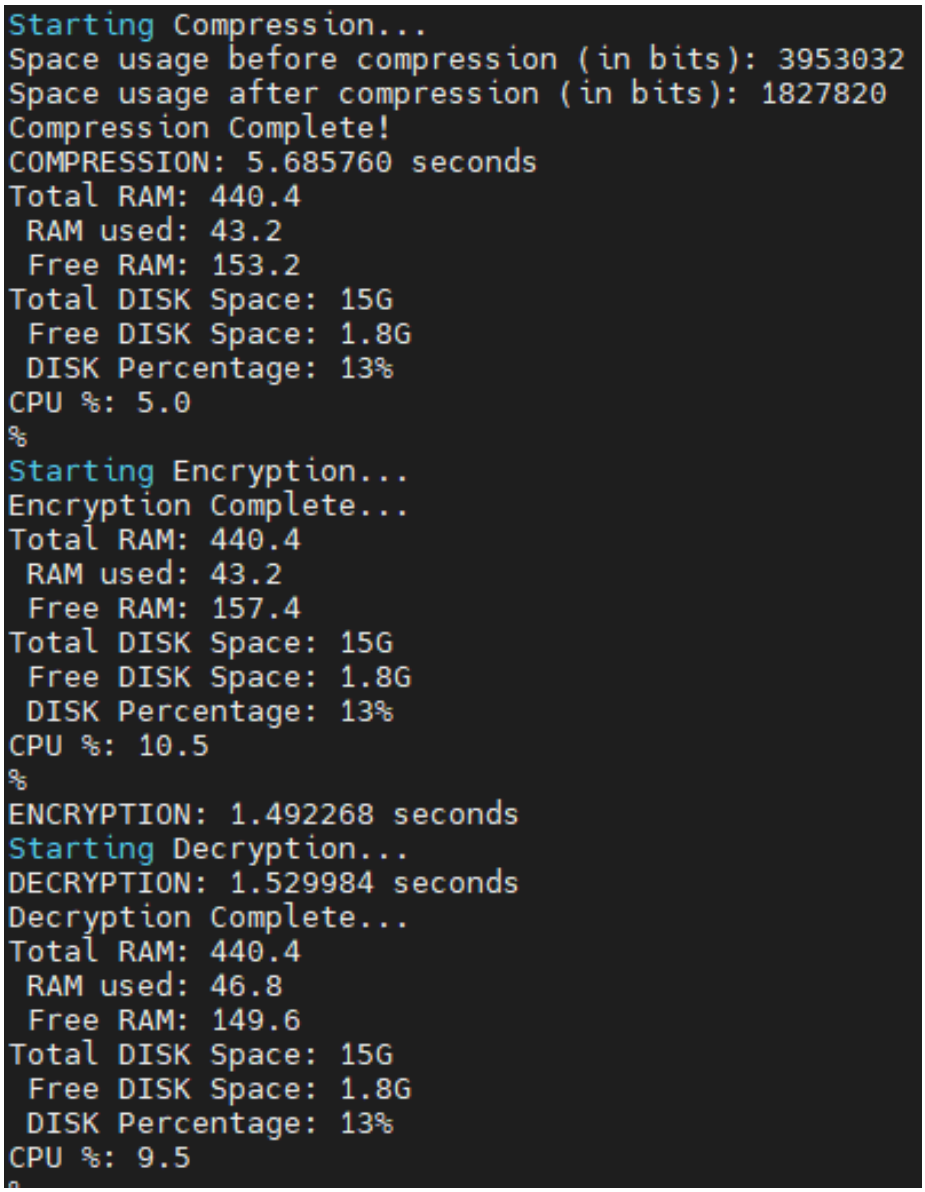

Figure 13: CPU sample Results

A graph was plotted from the results that were obtained from the power consumption experiment and they are shown on the image below

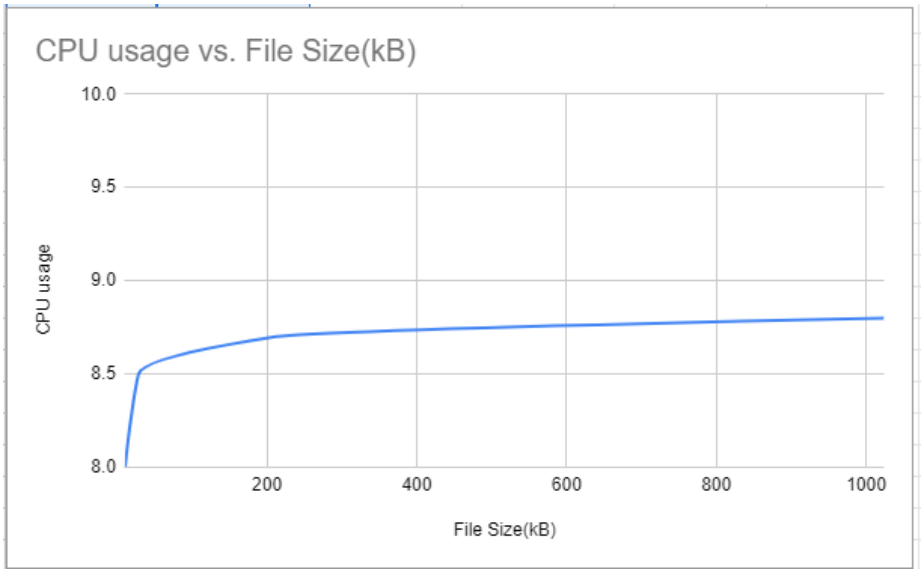

Figure 14: CPU sample Results

The image above shows that the subsystem is not using a lot of CPU even if the file size is increased, the curve is a logarithmic. THe CPU usage is in percentage.

Table 22: Comparison between AES-256 aand RSA

File size of the encrypted data	Time(ns) taken by AES-25	Time (ns) taken by RSA
10.4 kB	12	18
28.6 kB	30	43
212 kB	44	67
8.65 MB	102	150

The table above shows us that AES-256 encryption(the one we are using fro our subsystem)a symmetric method of encryption is better than RSA which is asymmetric.

Table 23: Time taken by AES-256 to encrypt different file sizes

File size of the encrypted data	Time(ns) taken by AES-256
10.4 kB	12
28.6 kB	30
212 kB	44
8.65 MB	102

Table 24: Comparing the time taken to encrypt a 500 MB file using different key sizes

Key Size	Average Time(ns)
AES-128	208
AES-192	483
AES-256	800

6 Validation using ICM-20648

The design was also tested using the ICM-20648. The buoy is going to make use of ICM-20649. ICM-20649 is being used for validation of our system. The table below shows the difference between ICM-20648 and ICM-20649.

Table 25: Comparing ICM-20649 and ICM-20648

ICM-20649	**ICM-20948**
6-axis motion tracking device	9-axis motion tracking device
gyroscope has a FSR of ±500 dps, ±1000 dps, ±2000 dps, and ±4000 dps	gyroscope has a FSR of ±250 dps, ±500 dps, ±1000 dps, and ±2000 dps
accelerometer has a FSR of ±4g, ±8g, ±16g, and ±30g	accelerometer has a FSR of ±2g, ±4g, ±8g, and ±16g
shock tolerance of 10000g	shock tolerance of 20000g
does not have a magnetometer	has a magnetometer
suitable for high impact applications	not suitable for high impact applications

Validating our subsystems using ICM-20648 is necessary because it is going to be a cheaper option to test how does our subsystems perform when they are integrated with the actual sensors.

6.1 Extrapolation with ICM-20649

- When running the tests on the ICM-20948, the full scale range of the gyroscope was set to ±500 dps which is available on the ICM-20649 and

the full scale range of the accelerometer was set to ±4g which is also available on the ICM-20649.

- We are also not going to collect the data from the 3 -axis magnetometer since our ICM -20469 is a 6 -axis motion tracking device that has an accelerometer and a gyroscope only.

6.2 ICM-20648 Data Analysis

The output data from the gyroscope, accelerometer and magnetometer was recorded and analysed separately.

6.2.1 Accelerometer

Accelerometers are electro-mechanical devices that sense either static or dynamic forces of acceleration. Static forces include gravity, while dynamic forces can include vibrations and movement. They measure in meters per second squared (m/s2) or in G-forces (g). Accelerometers are useful for sensing vibrations in systems or for orientation applications.
To check whether the IMU's Magnetometer is working, the sensor was laid stationary on a flat table, in the absence of a magnet, and the measurements were written to a csv file. These readings were then plotted to have a visual presentation.

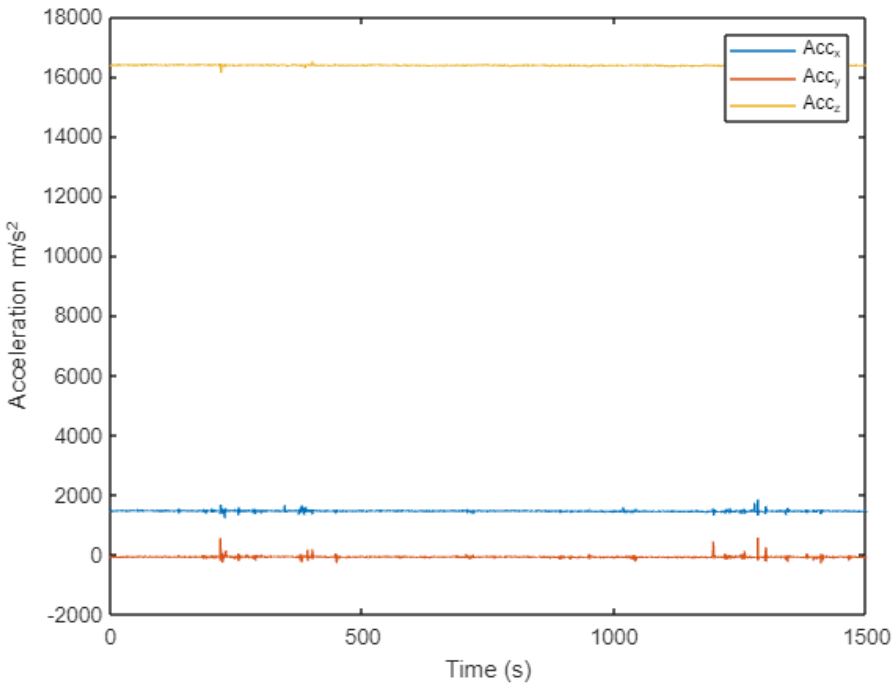

Figure 15: Accelerometor data when sense HAT is stationary

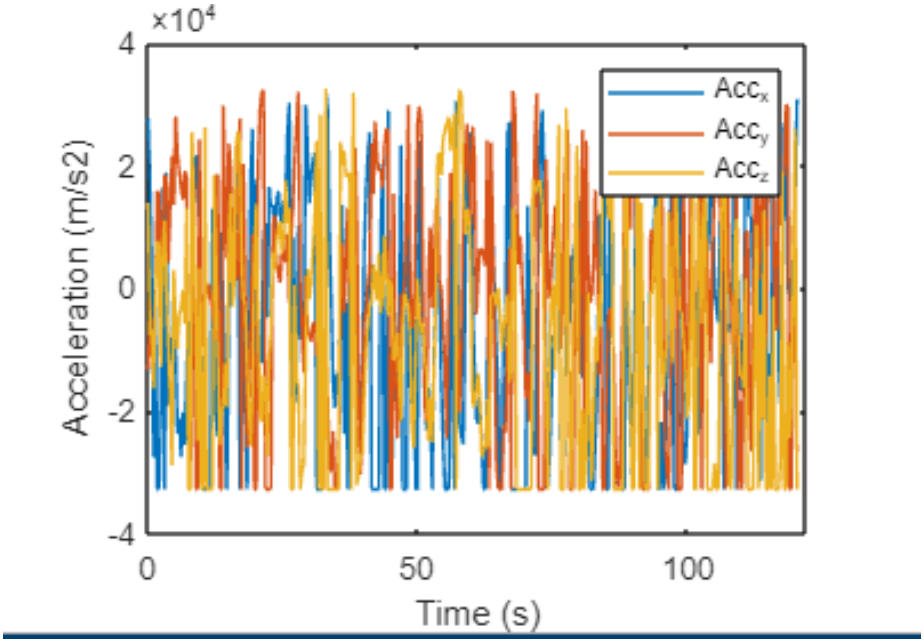

Figure 16: Accelerometor data when sense HAT is shaken

As shown in the differences of the readings between Figure 1 and Figure 2 when the sensor is stationary and shaken respectively, Figure 2 shows that the sensor is detecting motion, the Accelerometer functions as expected.

Histograms

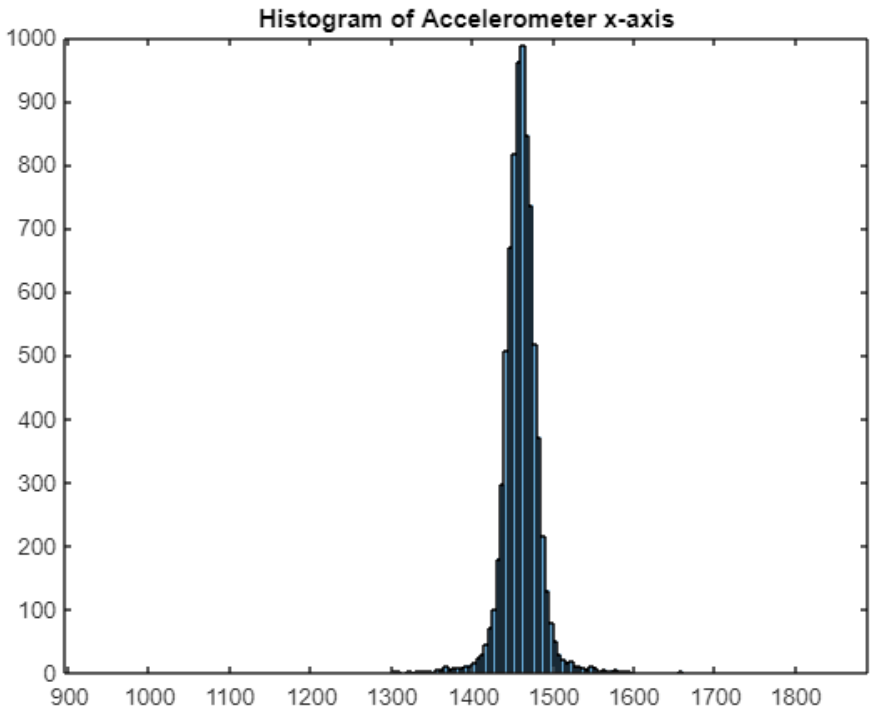

Figure 17: Histogram of the X coordinate data values of the accelerometer

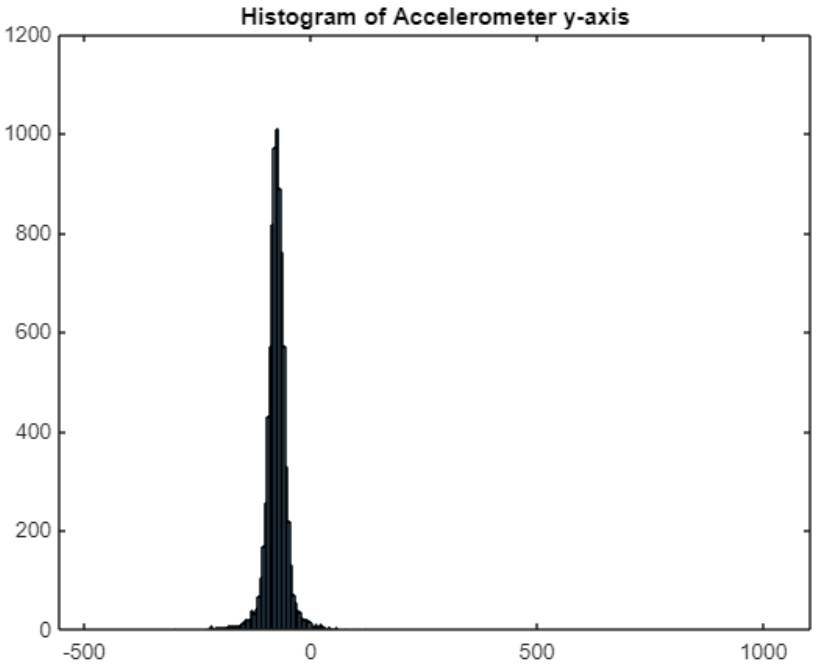

Figure 18: Histogram of the Y coordinate data values of the accelerometer

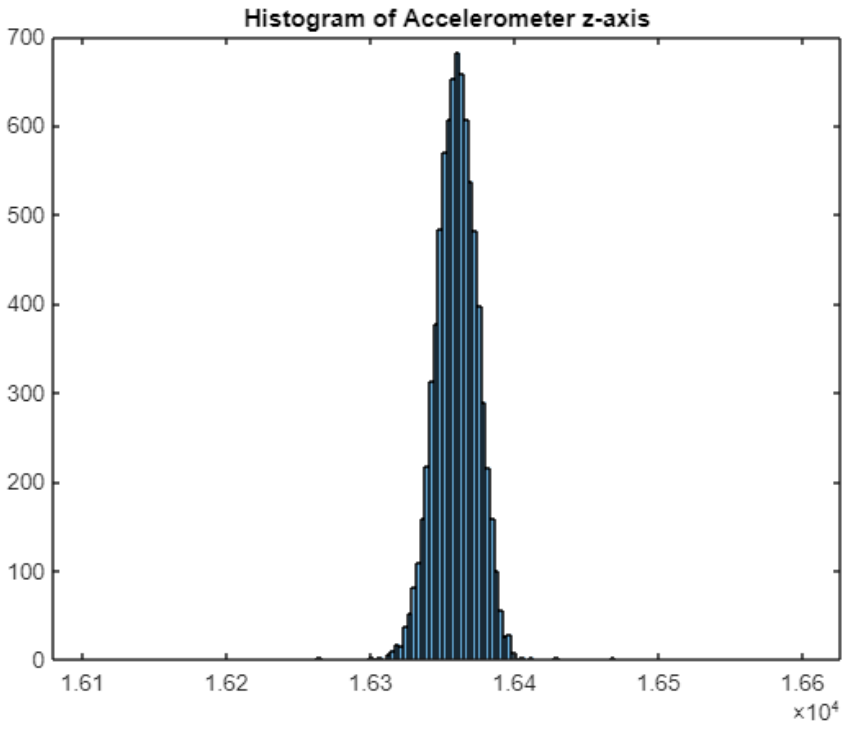

Figure 19: Histogram of the Z coordinate data values of the accelerometer

The histograms above are for the data recorded by the IMU sensors when the sense HAT is stationary. The shape of the data shows a Gaussian distribution. The readings of the accelerometer are normally distributed about a certain mean for each of the axes. The mean of the x-axis is approximately $1450 m/s^2$, the mean of the y-axis is close to zero and the mean of the z-axis is approximately $1.636 \times 10^4 m/s^2$. This shows that the forces of acceleration such as gravity are highly active on the z-axis of the accelerometer. There are little to none forces acting on the y-axis of the accelerometer.

6.2.2 Gyroscope

While accelerometers can measure linear acceleration, they can't measure twisting or rotational movement. Gyroscopes, however, measure angular velocity about three axes.
To check whether the IMU's gyroscope is working, the measurements when the sensor was laid stationary on a flat table were take and written to a csv file. These readings were then plotted to have a visual presentation. The senseHAT was then shaken continuously by hand, side to side, back and forth and in circular motion, and the measurements were written to a separate file. The readings were plotted again to get a visual presentation.

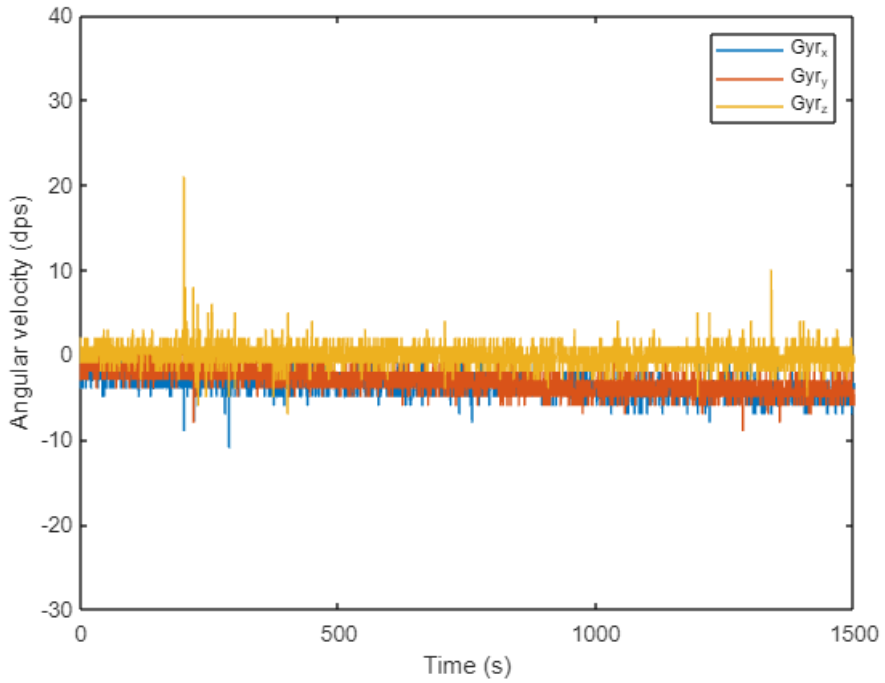

Figure 20: Gyroscope data when sense HAT is stationary

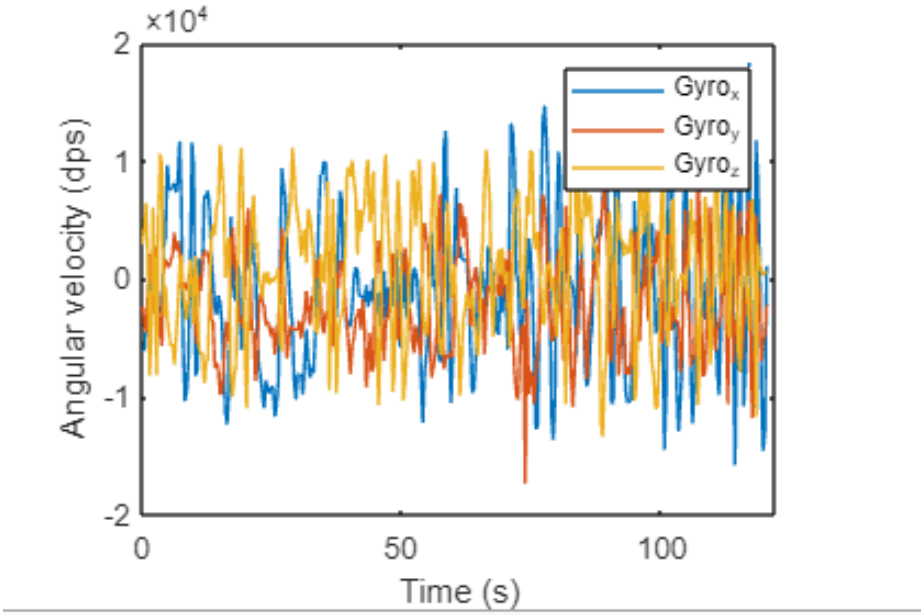

Figure 21: Gyroscope data when sense HAT is shaken

As shown in the differences of readings between Figure 3 and Figure 4 when the sensors are stationary and shaken respectively, the Gyroscope responds to the shaking motion within the FSR of +/- 4g as expected.

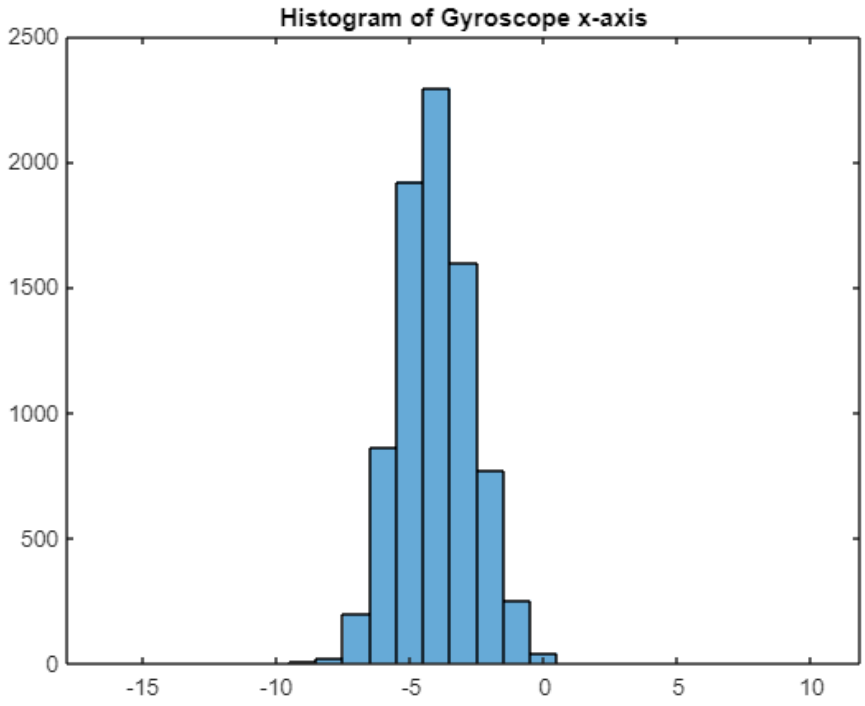

Figure 22: Histogram of the X coordinate data values of the gyroscope

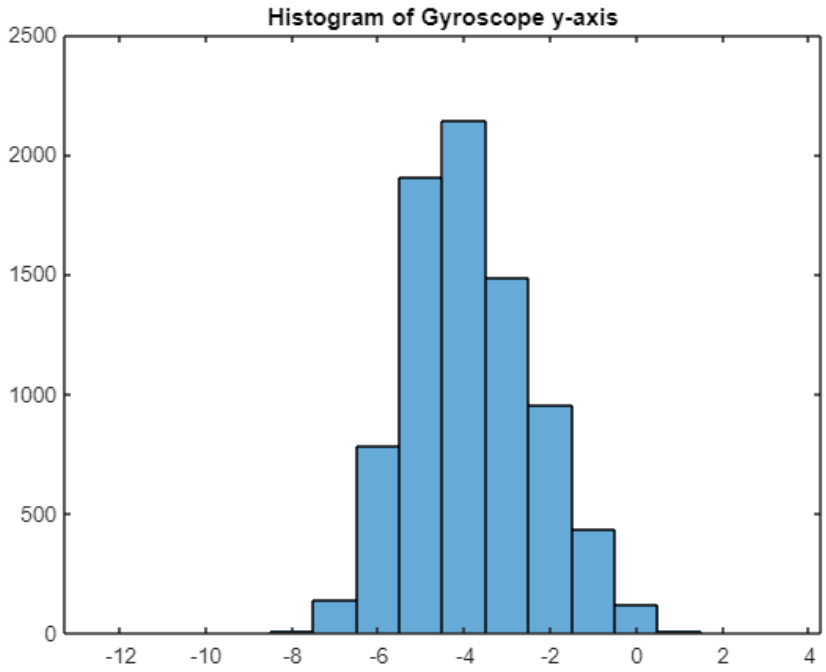

Figure 23: Histogram of the Y coordinate data values of the gyroscope

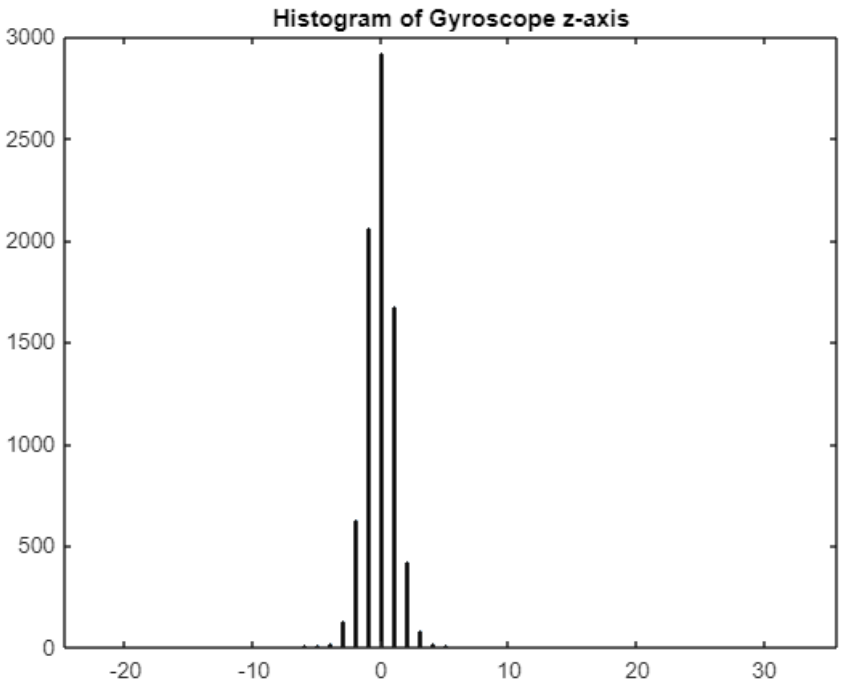

Figure 24: Histogram of the Z coordinate data values of the gyroscope

The histograms above are for the data recorded by the IMU sensors when the sense HAT is stationary. The shape of the data shows a Gaussian distribution. The readings of the accelerometer are normally distributed about a certain mean for each of the axes. The mean value of the x and y axes of the gyroscope are -4 dps (degrees per second) and the mean value of the z-axis is zero. This shows that there is no movement at all along the z-axis of the gyroscope as expected and noise recorded by the x and y axes.

6.2.3 Magnetometer

A magnetometer, as the name suggests, measures magnetic fields. It can detect fluctuations in Earth's magnetic field, by measuring the air's magnetic flux density at the sensor's point in space.
To check whether the IMU's Magnetometer is working, the sensor was laid stationary on a flat table, in the absence of a magnet, and the measurements were written to a csv file. These readings were then plotted to have a visual presentation. A magnet was then used to vary the measurements of the magnetometer. The magnet was continuously moved around the sensor with varying distances from the sensor. The measurements were written to a different csv file and plotted for visual presentation.

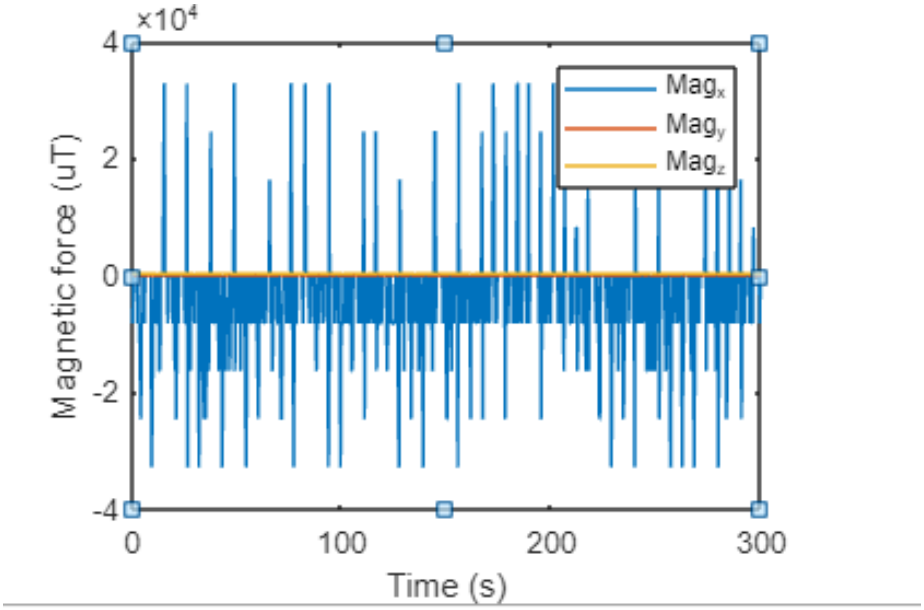

Figure 25: Magnetometer data when sense HAT is stationary

The graph above was plotted from the results of the raspberry pi when it was stationary.

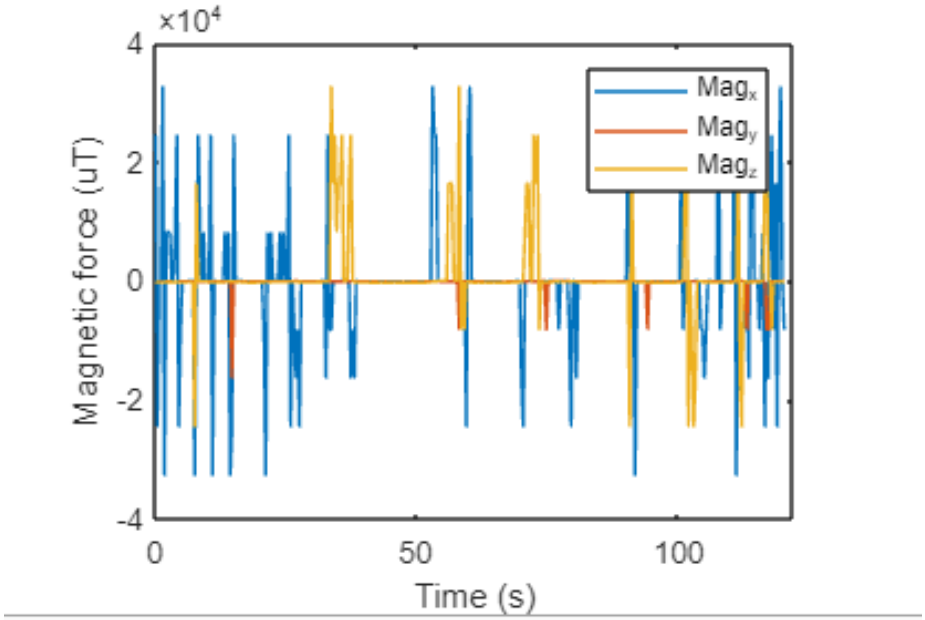

Figure 26: Magnetometer data when sense HAT is shaken

The graph above was obtained from the results that were obtained from the raspbian pi when it was shaken.

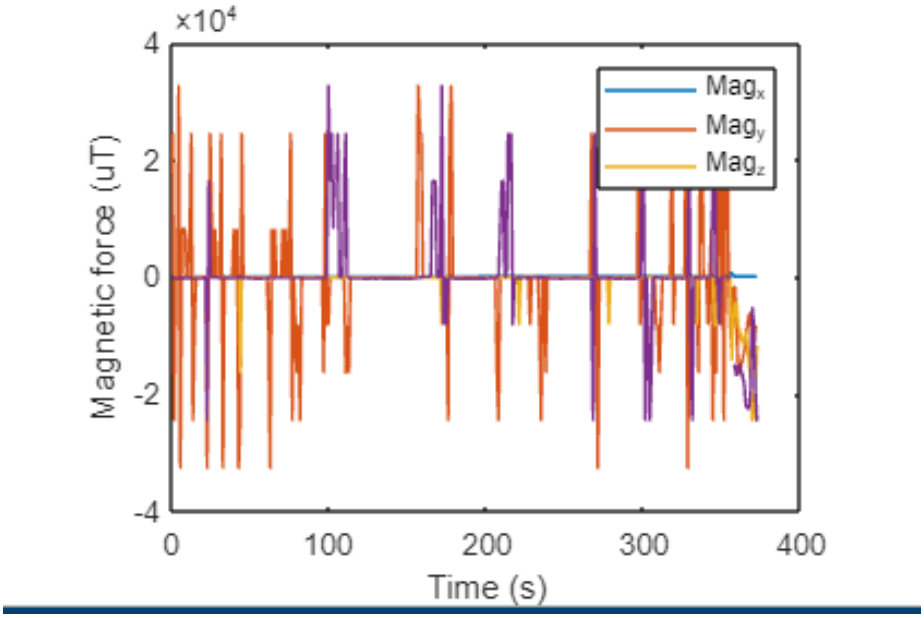

Figure 27: Validation Results when a magnet was used

The graph above were plotted from the results that were obtained when the disatnce between a raspberry pi and a magnet was varied.

6.3 Experimental Set Up

6.4 System Functionality

The system functionality was automated, from reading and recording the data from the sensors in a csv file, sending the csv file to the compression block to be compressed, sending the compressed file to the the encryption block for to be encrypted first the decrypted then finally sendi it back to the compression block for decompression. The tests done are listed below:

- The sense HAT (B) is connected to the raspberry pi using the I2C connection interface. The system is run by the main code in **icm20948.py**. This file reads the data form the accelerometer, gyroscope and magnetometer sensors in real-time.
- As the data is being read from the sensors, it is written to a csv file. A schedule was set for different time intervals for which data is read and recorded from the sensors into a csv file. The csv file with the data is sent to the compression block for compression.

- After the data is compressed, the compressed data is written to a file for observation. The compressed file is then sent to the encryption block to be encrypted, and the encrypted file is written to another file for observation.
- After encryption, the file with the encrypted data is then decrypted and the decrypted data is sent back to the compression block for decompression and the data is written to another file so that the data can be compared with the original file before compression.
- For the different scheduled times, execution time of each process, (ie compression, encryption, decryption and decompression), with the corresponding CPU and RAM usage are recorded to evaluate the performance of the system.

6.5 Compression Subsystem

6.5.1 Expected input and output data

The compression block receives a csv file with the readings from the accelerometer, gyroscope and magnetometer sensors. After compressing the file, the compressed data is written to a text file and sent to the encryption block. Different compression algorithms were tested. These are the huffman algorithm, lz4, zstd, zlib and gzip.

6.5.2 Performance and Power Consumption

- The same file was compressed using the different compression algorithms. Being tested was efficiency, that is, execution time for compression and decompression separately, and finally the compression ratio of each algorithm, which is the ratio of the file size before and after compression.
- For gzip, lz4 and zstd the compression level was varied to see the effect of speed of execution on the compression ratio and integrity of the data. This was done to determine if limiting the amount of processing done by the raspberry pi was a good trade-off for compression ratio.
- The next experiment was to test the performance of the different compression algorithms using different file sizes. Three files of different sizes were compressed by each algorithm. The compression ratio was recorded and the compression speed was derived from the execution time and compression ratio.
- To test for further performance and power usage, a python script was written to retrieve CPU and RAM usage of the raspberry pi after completion of each compression algorithm. To get accurate results, the CPU was warmed up by doing many iterations for it to reach close to its optimum operation level.

6.5.3 Comparison of original data file to final decompressed data file

- After decompression using each of the algorithms mentioned above, the file contents of the decompressed file will be compared to the original file to observe if there are any changes. As stated in the requirements, the goal is to retrieve at least 25% of the lower Fourier Transfer coefficients.

6.6 Encryption Subsystem

6.6.1 Expected input and output data

The subsystem is going to work with .txt file from the compression subsystem and it is going to encrypt the data into a cipher text file. The description of the test vectors of this subsystem is given below.

Test Vectors

y=E(file.txt)

y is the output from the encryption subsystem inform of a text file.
E is the encryption subsystem.
file.text is the input from the compression subsystem)

6.6.2 Experiments

For this subsystem, we are going to run two types of experiments that are functionality experiments and benchmarking experiments. Functionality experiments are going to test if the subsystem is working as expected and benchmarking experiments are going to test the performance of the subsystem eg time to encrypt different file sizes. A test harness (automated test framework) is going to be used to do all the experiments.

6.6.3 Functionanlity Experiments

- To check the validity of the encrypted data a checksum data is going to be implemented using the HMAC concept (keyed-hash message authentication code). The data from the compression subsystem is going to be passed through the HMAC and then the results of the HMAC are going to be sent to the AES encryption algorithm. The data is going to be decrypted and the results are going to be compared by the HMAC result (comparing the checksum data) using a python script. The above experiment is going to be carried out for different file sizes.

- To check the validity of the system's different modes of encryption **Monte Carlo** tests were used to verify if different modes of encryption were working properly (ECB mode,CBC mode and CFB mode). The Monte Carlo

test is going to check for anomalous combinations of inputs that will cause the subsystem to malfunction.

- To check if the subsystem is secure a different key is going to be used to decrypt the encrypted data and the out put is going to be compared with the data before encryption using the following command - diff file1 file2

6.6.4 Benchmarking Experiments

- The subsystem is using AES encryption which is a symmetric type, the first benchmark experiment is going to test the time that it is taken by a symmetric algorithm (AES the one our encryption subsystem is using) vs an asymmetric algorithm(RSA). The experiment is going to be run on our raspberry pi using python with a varying file size. The experiment is going to be repeated 10 times for each algorithm and the average time is going to be recorded. To measure the time, a timing module in python is going to be used.

- The second AES experiment is going to measure the amount of time taken by the AES algorithm to encrypt different file sizes. The experiment is going to be repeated 10 times for each file size and the average time is going to be recorded.

- The third experiment is going to test the average time taken by AES-128, AES-192 and AES-256 to encrypt a file size of 212 MB. The experiment is going to be repeated 10 times and the average time is going to be calculated. To record the running time, a time module in python is going to be used.

6.7 Results

6.8 System Functionality

Table 26: System performance

Process	File size (KB)	Execution time	CPU usage (%)
Compression	483	2.82	19
	325	1.981972	9.5
	158	0.18637	7.4
Encryption	483	3.77	14.3
	325	1.065759	4.8
	158	0.885901	9.5
Decryption	483	4.52	9.5
	325	1.046852	4.8
	158	0.852164	4.8
Decompression	483	1.81	10
	325	7.311268	5
	158	0.548447	4.8

The whole system was automated, from reading the data from the sensors, to writing it to a csv file, to sending the file to the compression block to be compressed, to sending the compressed file to the encryption block to encrypted, followed by decryption then finally decompression.
Table 26 above shows the execution time of each sub-system. The total execution time of the system increases as file size increases.
Since compression and encryption are going to be done by the raspberry pi on the actual implementation, it was important that these processes were optimised. As shown in **Table 26** above, the CPU usage by the compression and encryption processes for different file sizes was low hence the system does not consume low power thus meeting the low power consumption requirement.

6.9 Compression Subsystem

6.9.1 Comparing performance of the different compression algorithms

Table 27: Comparing best performance of each compression algorithm

Compression Algorithm	Compression Ratio	Compression Speed(MBps)	Decompression Speed(MBps)
huffman	2.14	0.094	0.034
gzip	2.77	1.500	2.144
zlib	3.76	0.523	20.18
zstd	4.34	0.0903	1.477
lz4	1.86	1.88	1.320

Initially, zstd compression was the chosen compression algorithm for the subsystem. Based on the results of the experiment obtained, shown in **Table 10** above, the algorithm with the highest performance and low amounts of processing was zlib. This observation has changed the initial decision to use zstd and instead use zlib for the system.

Table 28: Comparing performance of each compression algorithm at maximum vs. minimum compression level

Compression Algorithm	Compression Ratio (min compression level)	Compression Ratio (max compression level)	Compression Speed(MBps) (min compression level)	Compression Speed(MBps) (max compression level)
gzip	2.62	3.27	0.588	0.315
zstd	2.22	4.42	0.331	0.122
lz4	1.69	2.21	0.479	0.35

Since the requirements did not state the importance of achieving the maximum possible compression, but rather stresses the need to minimise the amount of processing done by the algorithms, reducing the compression level for high performance of the algorithm is a good trade-off hence the selection of the minimum compression level.

Table 29: Comparing performance of each compression algorithm using different file sizes

Compression Algorithm	File size (KB)	Compression Ratio	Compression Speed(MBps)
gzip	483	2.82	1.967
	325	2.85	1.883
	158	2.77	1.500
zlib	483	3.77	0.516
	325	3.87	0.544
	158	3.76	0.490
zstd	483	4.52	0.0863
	325	4.64	0.0894
	158	4.34	0.0903
lz4	483	1.81	1.94
	325	1.89	3.18
	158	1.86	1.88

As shown in **Table 12** above, the performance of the gzip compression algorithm increases as the size of the file increases. This is indicated by the increase of the compression speed.
For zlib, the general performace increases as the file size increases. However, the

compression speed for compressing the 325KB file is higher than the compression speed for compressing the 483KB file. This suggests that there may be an optimum sample size for optimum perfomance.
The perfomance of the zstd compression algorithm increases as the file size decreases. This is indicated by the highest compression speed being achieved by the smallest file size, 158KB, and the lowest compression speed achieved by the largest file size, 483KB.
The general performace of the lz4 algorithm increases as the file size increases. However, the compression speed for compressing the 325KB file is higher than the compression speed for compressing the 483KB file. This suggests that there may be an optimum sample size for optimum perfomance.

6.9.2 Power consumption

Table 30: Comparing CPU and RAM usage of each compression algorithm

Compression Algorithm	CPU %	RAM usage(MB)	Free RAM (MB)
huffman	5.0	42.4	298.8
gzip	9.5	40.3	300.5
zlib	5.0	41.2	299.6
zstd	9.5	40.3	300.3
lz4	9.5	40.4	300.6

Power consumption is not affected by RAM usage but is affected by the CPU usage. The higher the CPU usage, the higher the amount of power consumed. As shown in **Table 13**, gzip, zstd and lz4 have the highest CPU usage of 9.5% thus they consume more power. Huffman and zlib recorded the lowest CPU usage of 5.0% hence they consume less power. RAM usage affects the performance of the different compression algorithms. The higher the RAM usage, the higher the performance. The results in the table show that Huffman has the highest RAM usage, followed by zlib, then lz4 and lastly gzip and zstd which have the same RAM usage.

6.10 Encryption Subsystem

The following tables shows the results from the functionality experiments.

Table 31: Results obtained from checking the checksum data after enycrypting files with different sizes using AES -128

File size of the encrypted data	Percentage match
10.4 kB	100%
28.6 kB	100%
212 kB	100%
8.65 MB	100%

The results above show that the subsystem was able to encrypt files with different sizes successfully.

Monte Carlo Results The expected results of the first iteration is given below:

KEY = 9dc2c84a37850c11699818605f47958c

IV = 256953b2feab2a04ae0180d8335bbed6

PLAINTEXT = 2e586692e647f5028ec6fa47a55a2aab

CIPHERTEXT = 1b1ebd1fc45ec43037fd4844241a437f

The results that were obtained after running the first iteration were saved in a file and are shown on the image below.

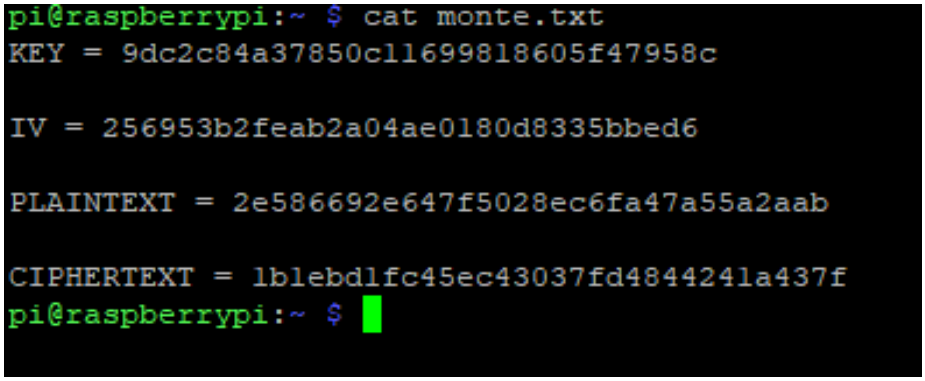

Figure 28: Monte Carlo First Iteration Results

It can be seen that the results of the first monte carlo iteration are the same as the expected vector results, this can be snowballed to other iterations. This shows that the subsystem is working as expected even in the eventuality of anomalous data.

The following table shows results from the benchmark experiments.

Power(CPU) Usage Results

The image below shows a sample of results that were obtained when power consumption was measured.

```
Starting Compression...
Space usage before compression (in bits): 3953032
Space usage after compression (in bits): 1827820
Compression Complete!
COMPRESSION: 5.685760 seconds
Total RAM: 440.4
 RAM used: 43.2
 Free RAM: 153.2
Total DISK Space: 15G
 Free DISK Space: 1.8G
 DISK Percentage: 13%
CPU %: 5.0
%
Starting Encryption...
Encryption Complete...
Total RAM: 440.4
 RAM used: 43.2
 Free RAM: 157.4
Total DISK Space: 15G
 Free DISK Space: 1.8G
 DISK Percentage: 13%
CPU %: 10.5
%
ENCRYPTION: 1.492268 seconds
Starting Decryption...
DECRYPTION: 1.529984 seconds
Decryption Complete...
Total RAM: 440.4
 RAM used: 46.8
 Free RAM: 149.6
Total DISK Space: 15G
 Free DISK Space: 1.8G
 DISK Percentage: 13%
CPU %: 9.5
```

Figure 29: CPU sample Results

A graph was plotted from the results that were obtained from the power

consumption experiment and they are shown on the image below

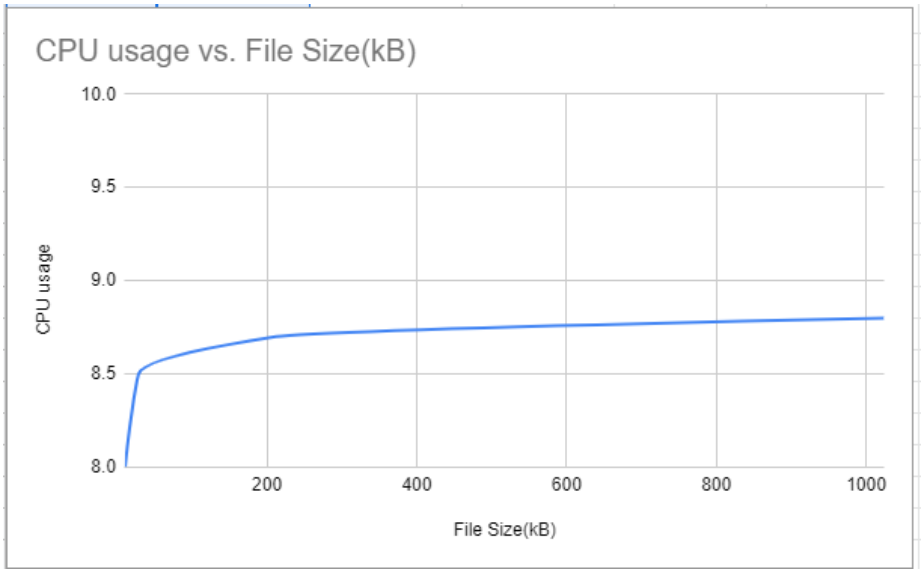

Figure 30: CPU sample Results

The image above shows that the subsystem is not using a lot of CPU even if the file size is increased, the curve is a logarithmic. THe CPU usage is in percentage.

Table 32: Comparison between AES-256 aand RSA

File size of the encrypted data	Time(ns) taken by AES-25	Time (ns) taken by RSA
10.4 kB	12	18
28.6 kB	30	43
212 kB	44	67
8.65 MB	102	150

The table above shows us that AES-256 encryption(the one we are using fro our subsystem)a symmetric method of encryption is better than RSA which is asymmetric.

Table 33: Time taken by AES-256 to encrypt different file sizes

File size of the encrypted data	Time(ns) taken by AES-256
10.4 kB	12
28.6 kB	30
212 kB	44
8.65 MB	102

Table 34: Comparing the time taken to encrypt a 500 MB file using different key sizes

Key Size	Average Time(ns)
AES-128	208
AES-192	483
AES-256	800

AES 128 takes the minimum time to encrypt the data and AES-256 takes the most time to encrypt the data.

7 Consolidation of ATP and Future plan

7.1 Compression Subsystem

Table 35: ATPs from Design Paper Review

Specification(s)	Figure of Merits
SP01 S01 We shall use Zstd algorithm to compress our data from the IMU.	The algorithm is supposed to compress the data from the simulations so that we will be able to retrieve at least 25% of the data.
SP02 S01 The Raspberry pi will communicate with the IMU using the I2C interface in order to receive the data	Print out data from the IMU to prove retrieval
SP04 S01 The adaptive mode of Zstandard is supposed to be set to the minimum required.	Fastest time of execution

7.1.1 Discussion of Compression Subsystem ATPs

Table 36: ATPs achieved by the Compression subsystem

Specification(s)	Figure of Merits
SP01 S01 We shall use Zstd algorithm to compress our data from the IMU.	Compression was achieved with a compression ratio of 2.22 as shown in Table 1
SP02 S01 The Raspberry pi will communicate with the IMU using the I2C interface in order to receive the data	Simulation of the communication of IMU and raspberry pi was done using udp connection with the raspberry pi acting as the server and the PC being the client sending the data
SP04 S01 The adaptive mode of Zstandard is supposed to be set to the minimum required.	The minimum compression level was used to achieve the fastest execution with minimum processing

7.2 Encyrption Subsystem

Table 37: ATPs from Paper Design

Specification(s)	Results
SP 01 S02 AES encryption library will be used.	Decrypt the encrypted data and compare it with the raw data.
SP 02 S02 Use Python scripts to handle data being sent between the subsystems	Data files need to be transmitted properly without fault.
SP 03 S02 AES encryption library will be used which is more efficient	Fastest time of execution

7.2.1 Old table of Discussion of Encryption Subsystem ATPs

Table 38: Discussion of encryption ATPs and results

Specification(s)	Results
SP 01 S02 AES encryption library will be used.	Encryption was achieved using AES-128
SP 02 S02 Use Python scripts to handle data being sent between the subsystems	Results not met. Python was used instead of C so there was no need to use the scripts
SP 03 S02 AES encryption library will be used which is more efficient	A python library Crypotome was used to provide AES that was used to encrypt data.

7.2.2 Updated table of Discussion of Encryption Subsystem ATPs

Table 39: Discussion of encryption ATPs and results

Specification(s)	Results
SP 01 S02 AES encryption library will be used.	Encryption was achieved using AES-128
SP 02 S02 100% encryption is supposed to be achieved	AES successfully encrypted all the data. There was no match when the enycrypted data was compared to the raw data
SP 03 S02 AES encryption library will be used which is more efficient	A python library Crypotome was used to provide AES. The AES was compared to other methods such as RSA and DES and it proved to be more efficient

7.3 Additional ATPs

Table 40: Comparing best performance of each compression algorithm

Specification(s)	Results
SP 04 S01 Raspberry pi is receiving data in real-time	As data was being sent from the PC it was printed out on the terminal to confirm reception

7.3.1 Future Work

The system is supposed to me modified so that it can read the data during specific times that might prove to have interesting results eg during the midnight. This will help by saving the battery of the buoy and it can record the conditions for a longer period of time.

8 Conclusion

The system we designed managed to meet all of the user requirements and specifications that were discussed in section 2. The compression subsystem successfully compressed files of different sizes and it passed all of its figure of merits. A trade-off to achieve a higher compression speed at the expense of compression ratio was made. This was because compression ratio was not mentioned in the requirements hence performance took precedence. The encryption subsystem also managed to encrypt files of different sizes and it proved to be secure after doing several tests and experiments, it also passed all of it's figure of merits. Overall system performance is good as indicated by the CPU usage of the different subsystems in the results section of the IMU validation tests. Changes to initial plans had to be made such as using python instead of C to implement the system. The challenge we faced during the design of the system is that we couldn't do the design with the actual IMU that is going to be in the SHARC buoy.

9 Admin Documents

Table 41: Task Distribution

Task	Group Member
1 IMU module	Dean and Takura
2 Experiment Set up	Takura and Takura
3 Compression Subsystem	Takura
4 Encryption Subsystem	Dean
5 System Functionanlty	Takura
6 Results-Compression Subsystem	Takura
7 Results Encryption Subsystem	Dean and Takura
8 ATP's	Dean and Takura
9 Validation using Simulated data	Dean and Takura
10 Validation using data for the IMU	Takura and Dean

9.1 Github Link

Visit our GitHub repository here.

9.2 Project Management tool

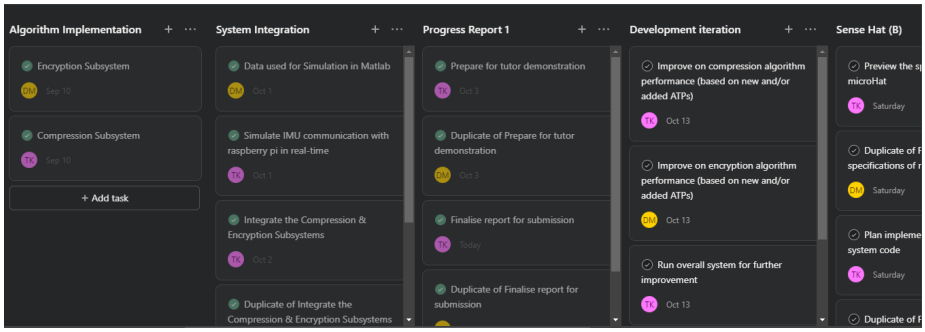

Figure 31: Group tasks

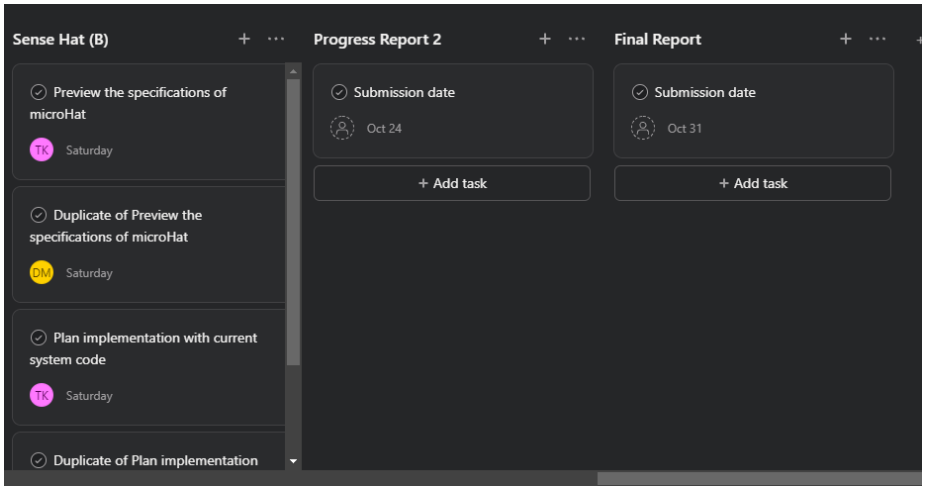

Figure 32: Continuation of group tasks