University of Cape Town

EEE3097S

Engineering Principles: Electrical And Computer Engineering

First Progress Report

Authors:
David Young
Caide Spriestersbach

Student Numbers:
YNGDAV005
SPRCAI002

September 2022

Table of Contents

Table of Figures

List of Tables

Administrative Details

Individual Contributions

Both members worked equally on the project and contributed to each section. However, some areas were focused more heavily on by a specific member. These sections are listed below.

Name	Sections	Pages
David Young YNGDAV005	2.1, 2.2, 2.4, 3.1, 3.2, 3.4, 4.1, 4.3	8-10, 12-14
Caide Spriestersbach SPRCAI002	1.1, 1.2, 1.3, 2.3, 3.3, 4.2	1-7, 8-9, 11-12, 14

Table I – Table of individual contributions made by each member

Project Management Tool

Below is a screenshot of the front page of our project management tool as of Tuesday, 6th September 2022.

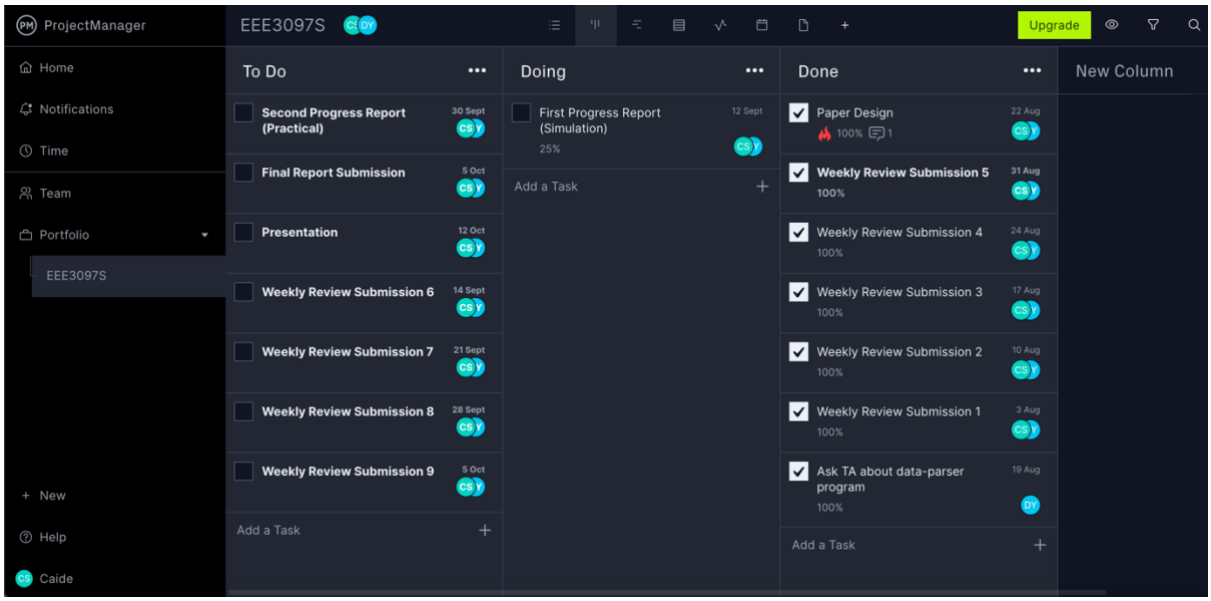

Figure 1 – Screenshot of project management tool front page

Development Timeline

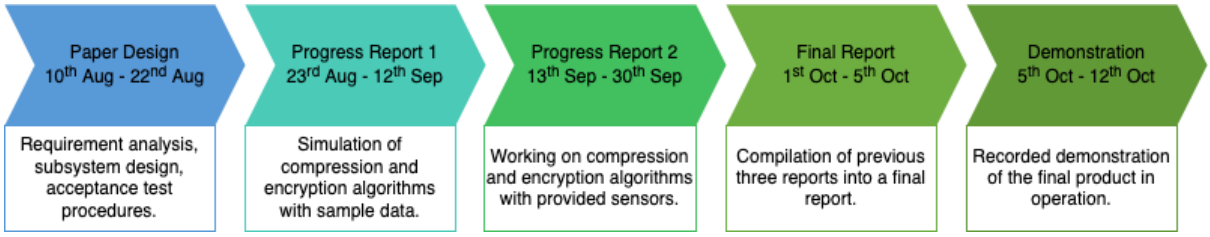

Figure 2 – Development timeline for the project

GitHub Link

The members made use of a GitHub git repository used throughout this project. Below is a link to the GitHub repository.

https://github.com/the-user-created/EEE3097S-Project

1. Data

Supplying the algorithms used for compression and encryption with simulated data was done to validate the workings of these algorithms. The following information concerns the data sets used, the file formats and how this relates to the compression and encryption for the SHARK Buoy project.

1.1. Data Used

The members decided to use simulated data as it would replicate the type of information we would receive from the Sense Hat B on the buoy in Antarctica. The file formats used in testing were binary, text and Comma-Separated Values (CSV), as it is common in Digital Signal Processing to use these file formats. Hence, it is essential to test the encryption and compression of these file types. Encryption and compression occurred on different data sets (in three file formats). The first data set was collected by the user carrying the Inertial Measurement Unit (IMU) in their pocket and the user walking around, sitting down and standing for over 5 minutes. The second data set recordings were from the IMU being fixed on a rotation program – the IMU was appointed to one of its axes, and the data were recorded for 10 minutes. The third data set was the same recording environment as the second, however, at a lower sample rate.

1.2. Justification of Data Used

The simulated data will accurately represent the data recorded on the buoy in Antarctica; therefore, the data sets will be suitable for testing the compression and encryption blocks. Different file sizes and types were tested to test the efficiency of the encryption and compression algorithm and ensure functionality. The file format in which the data will be stored and processed from the Sense Hat and IMU has not been finalised. The file format to be used will be completed once testing with the IMU and Sense Hat commences. It was then decided to test multiple file formats in the simulation phase to ensure that the encryption and compression would work with the file format used in the final product. The simulated data will be compared against the experimental data to test the acceptance test procedures and receive figures of merit. Following compression, encryption, decryption and decompression, the file will be in the same format as the IMU recorded; this can only be achieved in the simulation if the data set and format are the same as the one used in the practical application.

1.3. Initial Analysis of Data

Since the files contain similar data sets, it was decided to use the first thousand samples from the "EEE3097S 2022 Turntable Example Data 2.csv". The following three plots depict the relationship of the data collected from the IMU in the time domain; the following three plots show a sample of the Fourier Transform. Since the compression and encryption algorithms are both lossless, 25% of the Fourier Coefficients will be preserved – satisfying the user requirement. The project focuses on the encryption and compression of the files which contain the data; it was therefore not considered of high importance to value the IMU will determine the data set as this on the buoy – the following is, therefore, an example of how the data can be graphed for analysis.

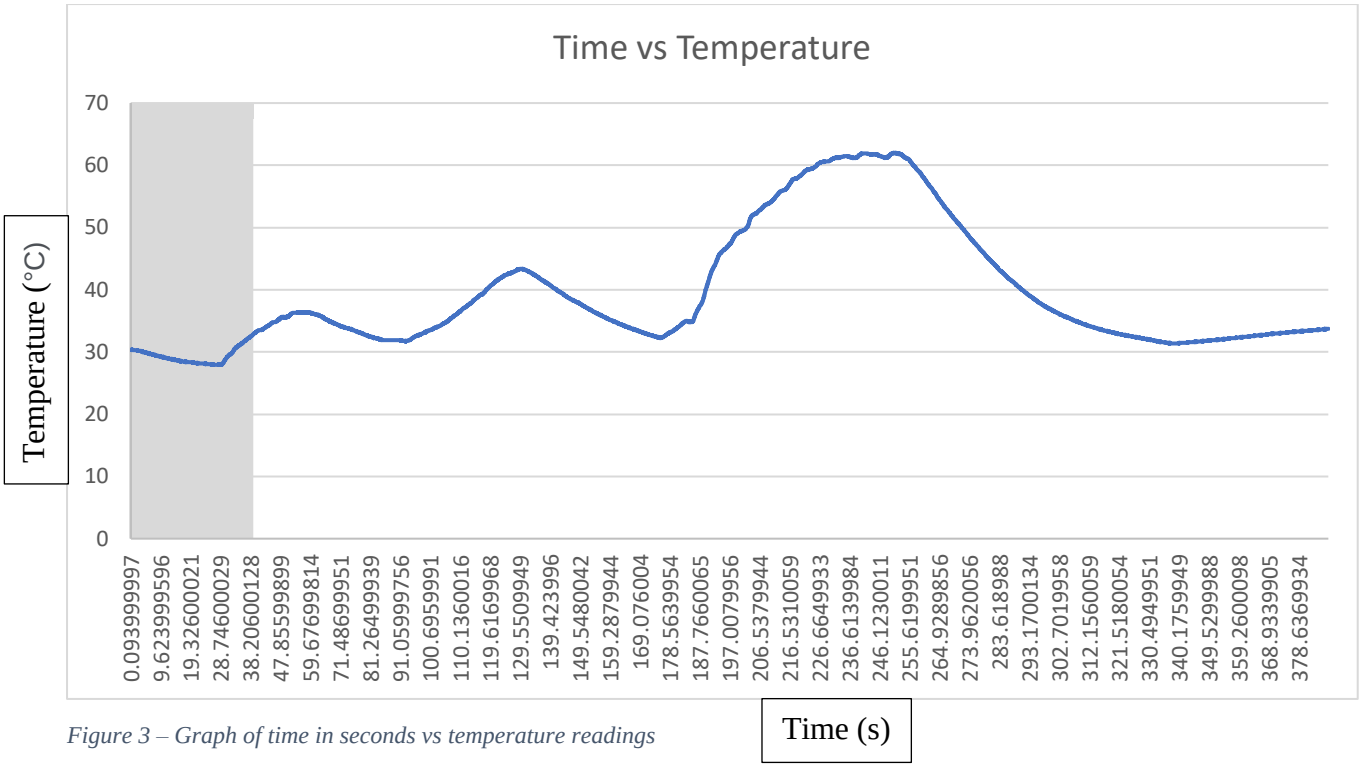

Figure 3 – Graph of time in seconds vs temperature readings

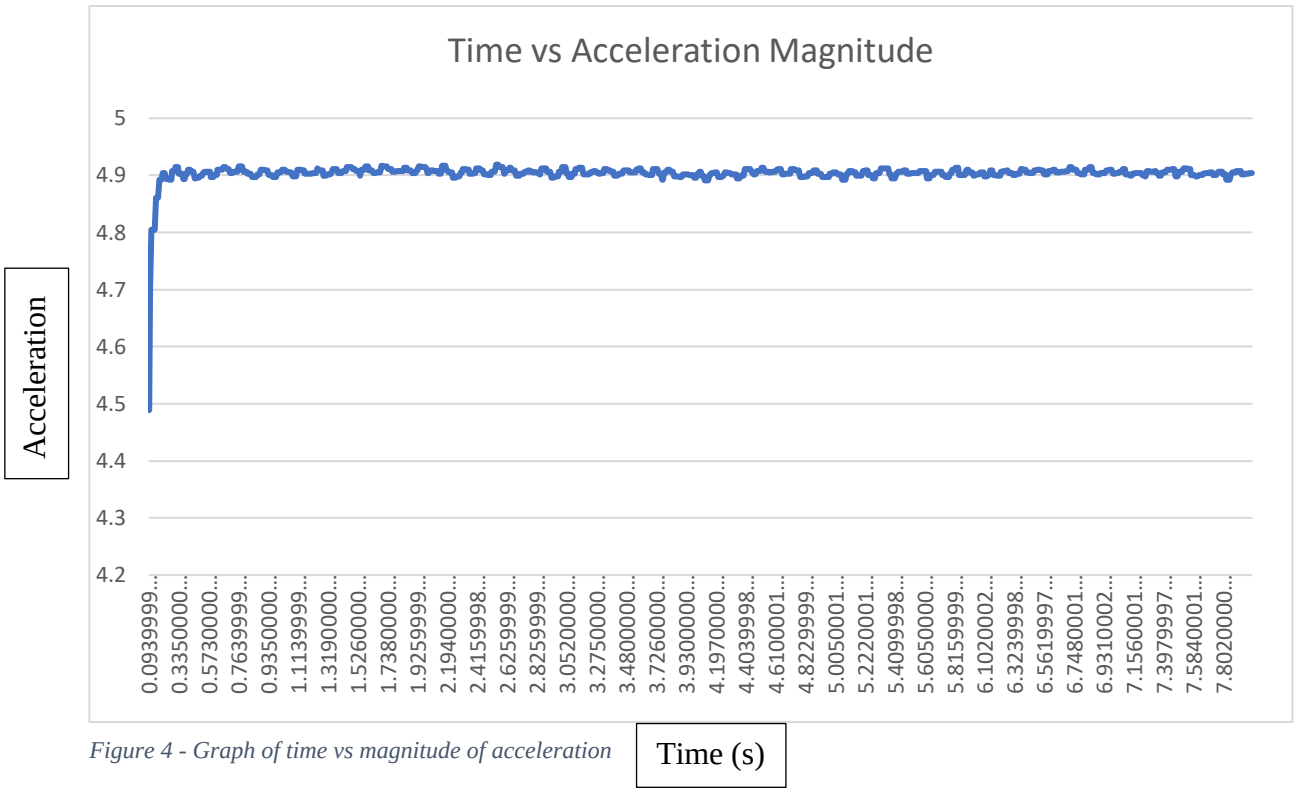

Figure 4 - Graph of time vs magnitude of acceleration

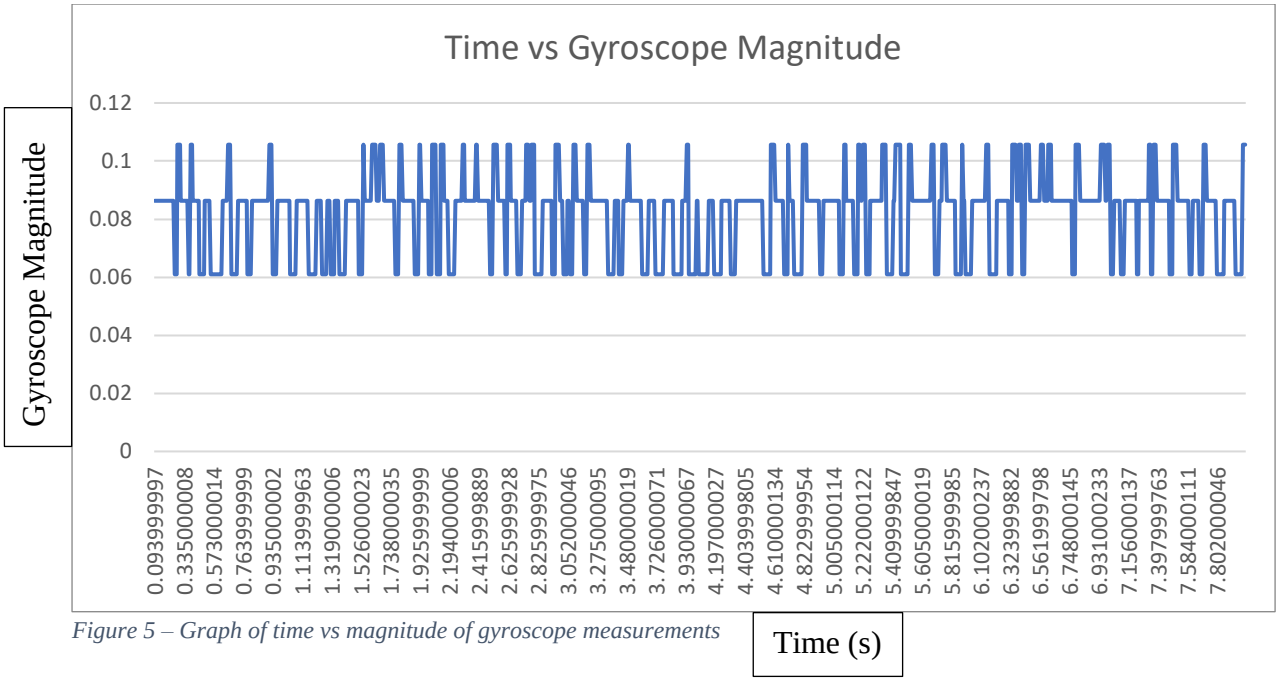

Figure 5 – Graph of time vs magnitude of gyroscope measurements

The following three graphs depict the Fourier transform of the simulated data. These graphs show what the data sets should produce once they have been decrypted, decompressed and analysed

what the data sets should make once they have been decrypted, decompressed and analysed using software such as Matlab. Matlab_R2022a was used to produce the following graphs.

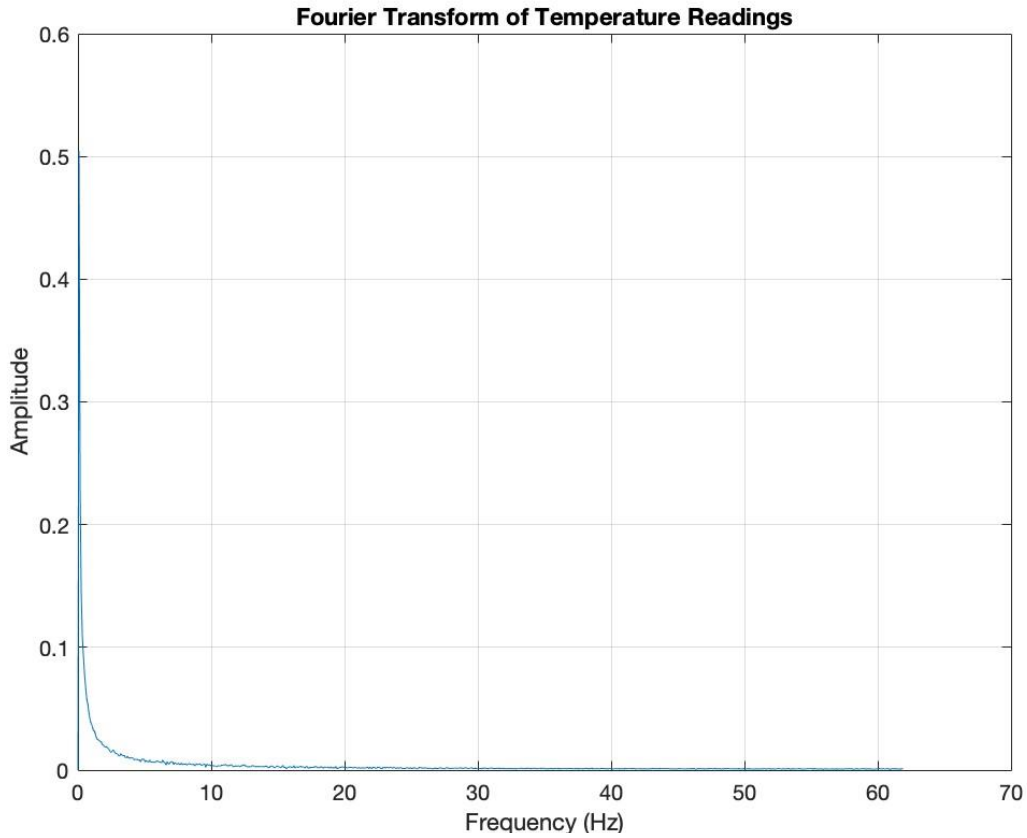

Figure 6 – Graph displaying the Fourier Transform of the Temperature data

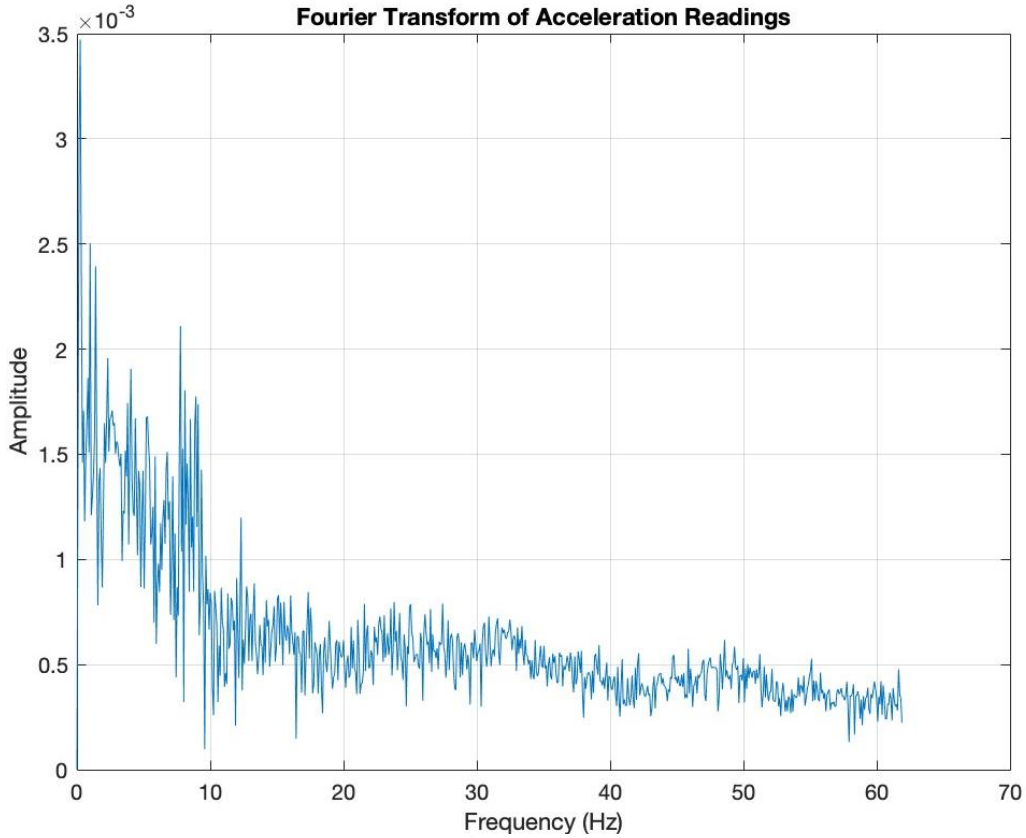

Figure 8 – Graph showing the Fourier Transform of the acceleration data

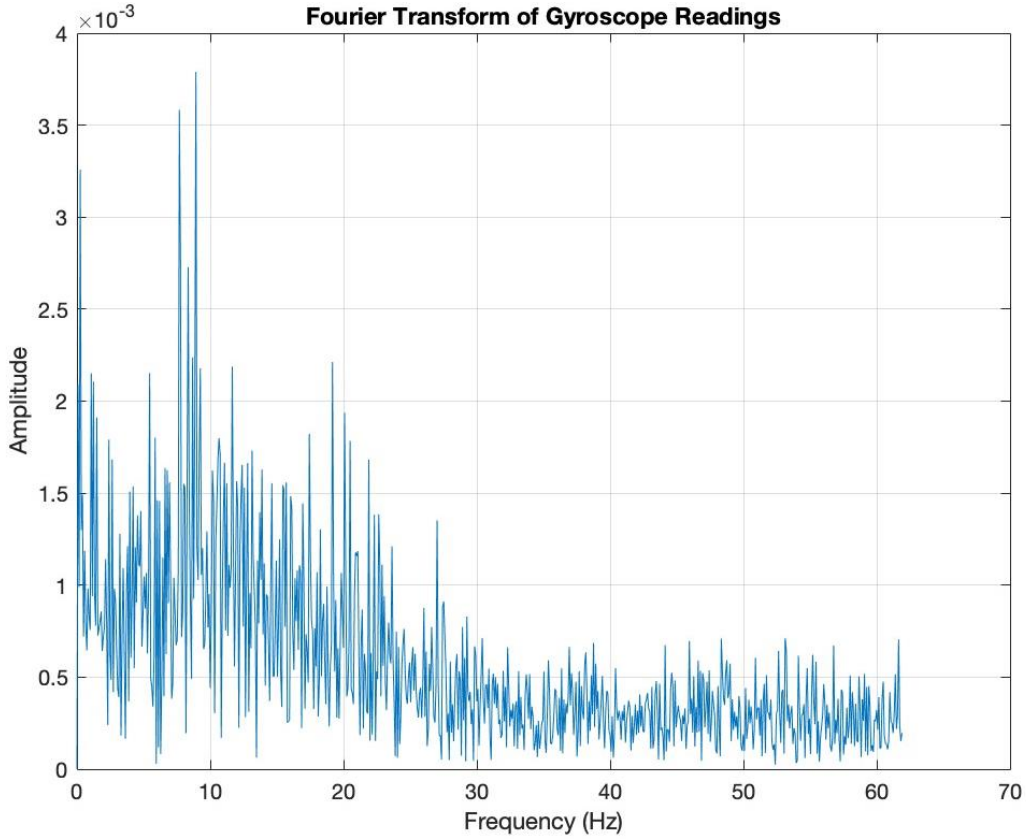

Figure 7 – Graph depicting the Fourier Transform of the gyroscope data

As seen above, significant noise is present in the graphs except for temperature – which has a spike at the origin, whereas the others follow an unexpected relationship.

2. Experiment Setup

2.1. Experiments to Check Overall Functionality of the System

The overall System uses a combination of encryption and compression functions in one Python file. The compression block will compress the input data file, passing the output file to the encryption block to encrypt the compressed file. The encrypted file can then be decrypted and decompressed to retrieve the original data. After decryption and decompression, a Python built-in comparison function is then used to make a deep comparison between the original data file and the resultant data file. Since both the compression and encryption algorithms are lossless, the input and output files should be identical.

2.2. Experiments for Compression Block

The compression block was tested across various use cases to determine values that could be used as comparisons for respective ATPs determined in the paper design. All the tests were executed across the same three sets of sample data:

- "EEE3097S_2022_Turntable_Example_Data_2.csv"
- "EEE3097S_2022_Turntable_Example_Data.csv"
- "EEE3097S_2022_Walking_Around_Example_Data.csv"

While the tests have peripheral objectives related to the ATPs, the main aim of the compression block is to reduce the physical space required to store the data whilst retaining all data inside the file. The Lempel–Ziv–Markov chain algorithm (LZMA) was used for compression. The original file size is compared to the compressed file size for each sample data set to determine the compression ratio. The execution time of the compression and decompression for each sample data set can also be determined using the Python time library. Lastly, the input and output files are compared for each sample data set to determine if the compression and decompression are lossless.

2.3. Experiments for Encryption Block

Testing the encryption block consisted of various stages. Testing was done in phases to obtain values which could then be used for comparisons and verify the respective ATPs. All the tests were done on the sample data provided by the compression algorithm:

- "EEE3097S_2022_Turntable_Example_Data_2.csv.lz"
- "EEE3097S_2022_Turntable_Example_Data.csv.lz"
- "EEE3097S_2022_Walking_Around_Example_Data.csv.lz".

While the tests have peripheral objectives related to the ATPs, the primary function of the encryption block is to convert plaintext data into ciphertext, which cannot trivially be decrypted without the password. The encryption algorithm used was the Twofish algorithm developed by Bruce Schneier. The original file was compared to the encrypted file for each file to determine whether the encryption was successful. The execution time for encryption and decryption of each data file was determined using the built-in Python time library. Determining if the

encryption was lossless was done by comparing the files provided to the algorithm (the compressed files) and the files following decryption.

2.4. Expected Data to be Retrieved and Returned From Each Block

The compression block is expected to retrieve IMU data as a CSV file. The compression block will read the CSV file and output the compressed .lz file. The encryption block will then retrieve this .lz file and encrypts it into a .lz.tf file. The decryption algorithm will then retrieve the .lz.tf file and decrypt it into a .lz file. This .lz file is retrieved by the decompression algorithm, which will output a CSV file.

3. Results

3.1. Results of Experiments to Check Overall Functionality of the System

The output for the overall functionality experiment is shown below:

```
Uncompressed file size = 22429420 Bytes
Compressed file size = 1488504 Bytes
Compression took 42.5765600204 seconds

Enter encryption password: 1234567890
Unencrypted file size = 1488504 Bytes
Encrypted file size = 1488512 Bytes
Encryption took 5.5818529129 seconds

Enter encryption password: 1234567890
Encrypted file size = 1488512 Bytes
Unencrypted file size = 1488504 Bytes
Decryption took 5.675347805 seconds

Uncompressed file size = 1488504 Bytes
Compressed file size = 22429420 Bytes
Decompression took 2.6749780178 seconds

The original file matches the output file!
```

Figure 9 – Command-line output from the overall functionality experiment

The sample data set used for this experiment was "EEE3097S_2022_Turntable_Example_Data.csv". The uncompressed file has a size of 22429420 bytes and a compressed size of 1488504 bytes – this which means that a compression ratio of 15.1 was achieved. The compression took 42.58 seconds, and the decompression took 2.67 seconds. The encryption took 5.58 seconds, and the decryption took 5.68 seconds. The encryption added 8 bytes to the compressed file because the compressed file was not a multiple of 16 bytes (block size). These 8 bytes acted as padding for the encryption algorithm so that

the file's content was correctly encrypted. The last line output in the figure above shows that the original file is identical to the decompressed file; therefore, the compression and encryption are both lossless.

3.2. Results of Experiments for Compressions Block

The output for the compression experiment is shown below in table form:

File	Original File Size [bytes]	Compressed File Size [bytes]	Compression Time [seconds]	Decompression Time [seconds]
Turntable_1	22 429 420	1 488 504	41.6754570007	2.6192178726
Turntable_2	19 827 842	939 264	30.4814031124	1.7084400654
Walking	14 332 601	1 353 856	29.0786149502	2.7873430252

Table II – Raw results for the compression block experiments

The values in the table below were calculated from those in the table above.

File	Compression Ratio	Compression Speed [bytes/second]	Decompression Speed [bytes/second]
Turntable_1	15.06843112	538 192.5385	8 563 403.692
Turntable_2	21.1099776	650 489.8061	11 605 816.56
Walking	10.58650329	492 891.4608	5 142 029.836
Average	15.588304	560 624.6018	8 437 083.363

Table III – Calculated results for the compression block experiments

As can be seen from the results shown in the table above, LZMA appears to be a high-speed and efficient compression algorithm.

3.3. Results of Experiments for Encryption Block

The output for the encryption experiment is shown below in table form:

Compressed File	Original File Size [bytes]	Encrypted File Size [bytes]	Encryption Time [seconds]	Decryption Time [seconds]
Turntable_1.lz	1 488 504	1 488 512	5.5718431473	5.5043358803
Turntable_2.lz	939 264	939 264	2.2277162075	2.3133990765
Walking.lz	1 353 856	1 353 856	4.4565820694	4.6864788532

Table IV – Raw results for the encryption block experiments

The values in the table below were calculated from those in the table above.

File	Used Padding	Encryption Speed [bytes/second]	Decryption Speed [bytes/second]
Turntable_1	Yes	267 147.5059	270 425.3578
Turntable_2	No	421 626.4158	406 010.3635
Walking	No	303 787.9655	288 885.5455
Average	-	330 853.9624	321 773.7556

Table V – Calculated results for the encryption block experiments

As seen by the data, we can conclude that Twofish is slightly faster at encrypting files than decrypting files. It is still, however, a high-speed and efficient algorithm.

```
Encrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz:
Enter encryption password: 1234567890
Unencrypted file size = 1353856 Bytes
Encrypted file size = 1353856 Bytes
Encryption took 4.532520771 seconds

Decrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz.tf:
Enter encryption password: 1234567890
Encrypted file size = 1353856 Bytes
Unencrypted file size = 1353856 Bytes
Decryption took 4.4951100349 seconds

The original file matches the output file!
```

Figure 10 – Decryption using the correct password

```
Encrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz:
Enter encryption password: 1234567890
Unencrypted file size = 1353856 Bytes
Encrypted file size = 1353856 Bytes
Encryption took 4.514988184 seconds

Decrypting EEE3097S_2022_Walking_Around_Example_Data.csv.lz.tf:
Enter encryption password: t
Encrypted file size = 1353856 Bytes
Unencrypted file size = 1353856 Bytes
Decryption took 4.576939106 seconds

The original file does not match the output file!
```

Figure 11 – Decryption using an incorrect password

As can be seen by comparing Figure 10 and Figure 11, the decryption fails when an incorrect password is provided to the decryption algorithm.

3.4. Effects of Changing the Data Provided to the System

Noise was added to the data points in the "EEE3097S_2022_Turntable_Example_Data.csv" to determine whether such a change in the data will affect the compression performance or encryption performance. As seen in the figure below, the performance of both the compression and encryption has been severely affected compared to the result shown in Figure 9, which used the original data set.

```
Uncompressed file size = 22422023 Bytes
Compressed file size = 9091860 Bytes
Compression took 62.0316669941 seconds

Enter encryption password: 1234567890
Unencrypted file size = 9091860 Bytes
Encrypted file size = 9091872 Bytes
Encryption took 462.5292720795 seconds

Enter encryption password: 1234567890
Encrypted file size = 9091872 Bytes
Unencrypted file size = 9091860 Bytes
Decryption took 459.6317350864 seconds

Uncompressed file size = 9091860 Bytes
Compressed file size = 22422023 Bytes
Decompression took 14.3226840496 seconds

The original file matches the output file!
```

Figure 12 - Command-line output from the overall functionality experiment with noise

A table presenting the difference in the results between those in section 3.1 and this experiment is shown below:

	Change in Input File Size [bytes]	Change in Output File Size [bytes]	Change in Speed [bytes/second]	Change in Speed (with original data set as baseline) [Percentage]
Compression	4397	7 603 356	165 342.3952	-31.4%
Decompression	7 603 356	4397	6 819 408.507	-81.3%
Encryption	7 603 356	7 603 360	247 011.6069	-92.6%
Decryption	7 603 360	7 603 356	242 496.0141	-92.5%

Table VI – Changes due to adding noise to the data set

As can be seen, by the severe changes in performance, adding noise to the data set will harm the performance of both compression and encryption.

4. Simulated Data Acceptance Test Procedures

4.1. Compression ATPs

ATP	Description	ATP achieved	Comment
Compression time	The compression algorithm must not take more than 5 seconds per 10 kilobytes of data.	Yes	Table III shows that the compression algorithm achieved an average speed of ~560 000 bytes per second, which is far more than 2048 bytes per second.
Data loss	No data must be lost during the compression of the data file.	Yes	LZMA uses lossless compression, validated by the deep file comparison in section 3.1.
Compression effectiveness	The compression ratio of the original file size and the compressed file size must be at least 1.5.	Yes	Table III shows that the compression algorithm achieved an average compression ratio of ~15.6, which is far more significant than 1.5.

Table VII – Simulation Data ATPs for the compression block

4.2. Encryption ATPs

ATP	Description	ATP achieved	Comment
Encryption time	The encryption and decryption execution time should not exceed 10 seconds per 10 kilobytes of data.	Yes	Table V shows that the encryption algorithm achieved an average speed of ~330 000 bytes per second, surpassing the ATP.
Data loss	No data must be lost during the encryption and decryption of the data file.	Yes	As shown in section 3.1, the compression and encryption are both lossless.
Encryption security	The encryption must be strong enough to prevent decryption.	Yes	This ATP is difficult to test, but the encryption prevents trivial decryption attempts, as shown in section 3.3, figures 10 and 11.
Encryption integrity	The original data in the file must be identical to the decrypted data in the output file.	Yes	As shown in section 3.1, the compression and encryption are both lossless.

Table VIII – Simulated Data ATPs for the encryption block

4.3. Checksum ATPs

ATP	Description	ATP achieved	Comment
STM checksum	The STM must create a checksum of the original data file successfully.	No	No data collection from the STM could be created; therefore, no checksum.
Client checksum	The client's device must successfully create a checksum of the decrypted and decompressed file.	Yes	The checksum for the client was created using Python successfully.
Checksum comparison	The client's and STM's checksums must be identical.	No	Unable to create checksum on STM.

Table IX – Simulated Data ATPs for the checksum block

References

There are no sources in the current document.