EEE3097S First Progress Report

Group 3
Shameera Cassim and Tristyn Ferreiro
CSSSHA020, FRRTRI001

18 January 2023

Admin

Contributions

Task	Completed by
Admin	Both
Subsystem Design: Compression	Tristyn Ferreiro
Subsystem Design: Encryption	Shameera Cassim
Data Analysis	Both
Experiment Setup: Overall	Both
Experiment Setup: Compression	Tristyn Ferreiro
Experiment Setup: Encryption	Shameera Cassim
Results: Overall	Both
Results: Compression	Tristyn Ferreiro
Results: Encryption	Shameera Cassim
Acceptance Test Procedures: Assessment table	Both
Acceptance Test Procedures: ATP Updates	Tristyn Ferreiro
Specifications Updates	Shameera Cassim

Project Timeline

Since we did not add a timeline into the previous submission, we have included a break down of the timeline up until this submission as well as the planned timeline for the remainder of the project.

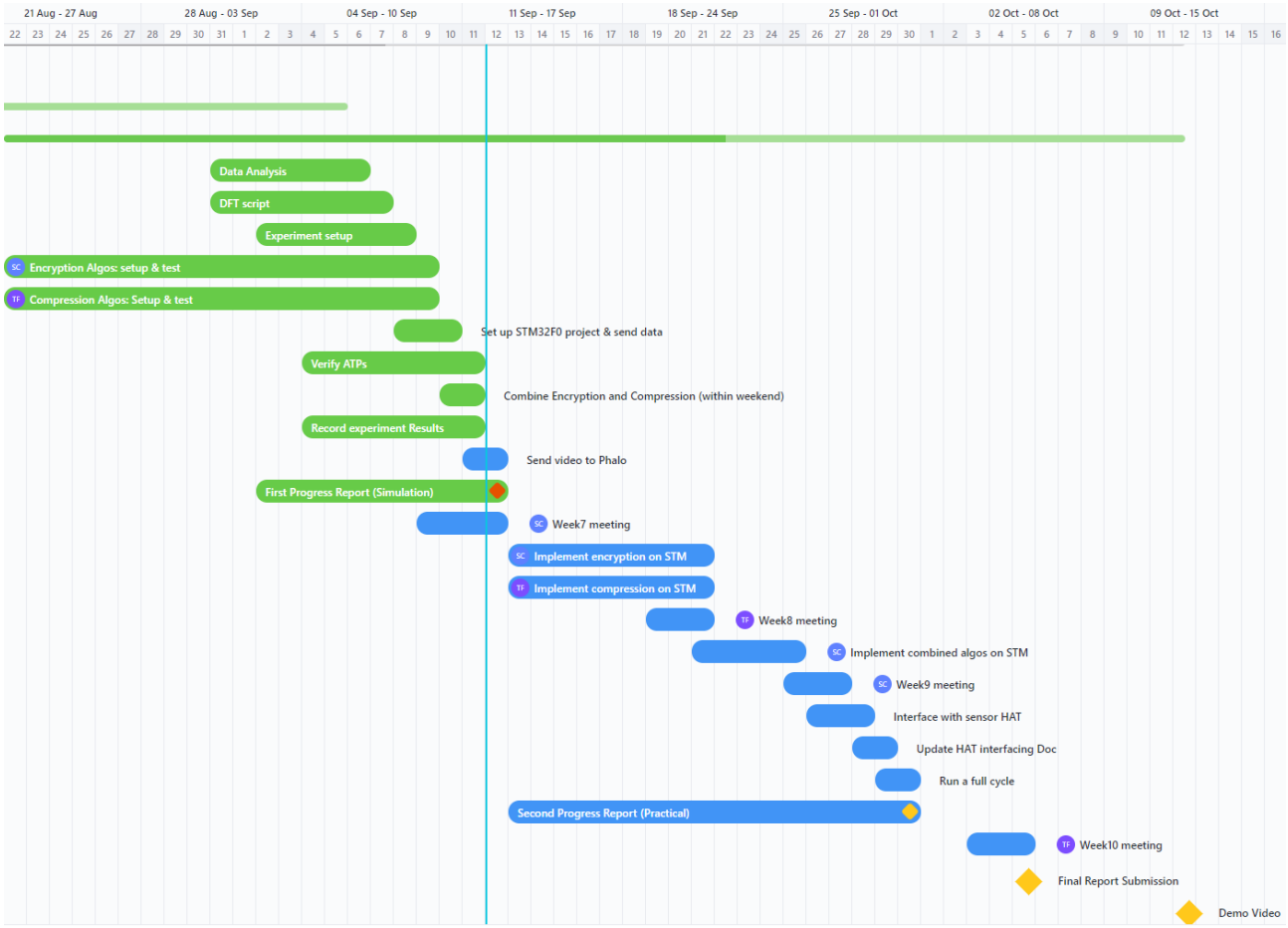

Figure 1: Timeline for the remainder of the project

Project Management tool

Due to limited functionality on Asana's free membership, the decision was made to switch the project management tool to ClickUp. The Asana project was imported into the new tool which is shown in the screenshots below:

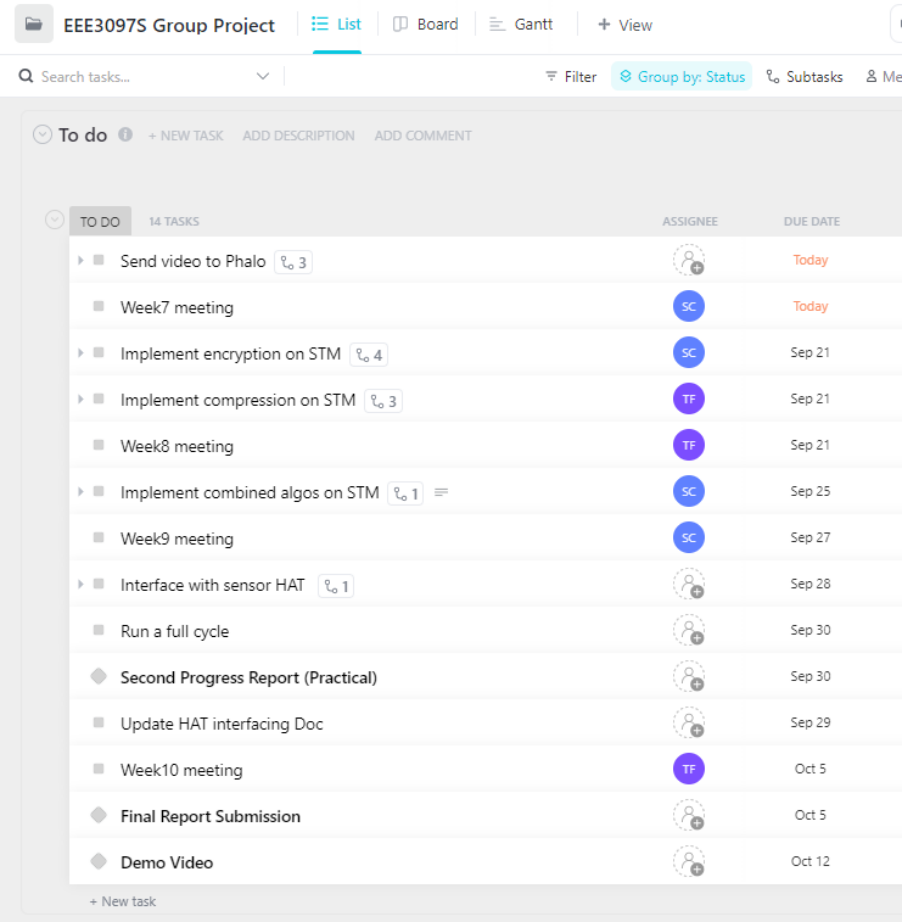

Figure 2: Screenshot of the project management tool showing the remainder of our tasks

The list above shows a break down of milestones and suggested tasks for the remainder of the project from the ClickUp task list. Some tasks have sub-tasks that need to be completed before the main task can be marked as completed. More tasks will be added as needed. Tasks not assigned to a specific person are either joint tasks or will be assigned in future.

Git

The GitHub repository for this project can be found via this link: git

Contents

Listings

1 Subsystem Design

This is a brief overview of the subsystem design and decisions made to date, some updates have been made since the Paper Design.

1.1 Compression

1.1.1 Algorithm Comparisons

Algorithm Types

Three algorithm types are considered namely Delta, Delta-of-Delta and Dictionary compression:
Delta compression only stores the difference between an object and a reference object, this reduces the amount of information needed to represent a data object. This compression works best with numerical data that represents something slowly changing over time. [2] The data recorded by the IMU is numerical and slowly changes over time. This is a suitable compression algorithm.
Delta-of-delta compression applies a second layer of Delta compression to the Delta-compressed data [2]. This further decreases the size of the data however it increases the compression time and it increases the source code. Since the STM32F051 used for testing has limited memory, this algorithm takes up valuable space needed for data.
Dictionary compression makes a list of the possible values that can appear in the data and then stores an index to the dictionary containing the unique values. This works well when there is repeating data. [2]. The LZSS compression is dictionary based.

Comparisons

The following values are used as benchmarks for comparison between compression algorithms:
Compression Ratio is the size of the compressed file compared to the size of the original file (i.e., the smaller the better):

$$\text{compression ratio} = (\frac{compressed file size}{original file size})$$

If compression ratio is less than 1 then compression has been successful.
The **compression speed** is important because it influences how much power is used to compress. i.e., the longer the algorithm runs the more power is used.
The **source code size** is considered because of the limited space available on the STM32F0 discovery board used for development and testing. Since there is only 64kB of flash memory available, the compression and encryption source code cannot take up more than 32kB combined.

The values in the table below were obtained from the lzbench mark [3] which was run on a number of different files. These values are just an estimate and do not give the real results of using the algorithms with the data expected in this design.

Table 1: Compression Algorithms

Algorithm	Ratio (%)	Compression speed	src size
LZSS	41.51	24MB/s	4kB
LZW	42.85	150 MB/s	20kB

Analysis

From table 1, it is clear that the LZW algorithm is both faster and has a better saving percentage, however, the size of the source code is much larger than the LZSS source code. Although the LZW savings percentage is more than for LZSS, it is not significantly greater. Additionally, compression algorithms work better on bigger sets of data and so it is better to use LZSS on more data than LZW on less. For these reasons, **LZSS is chosen as the algorithm**.

1.2 Encryption

1.2.1 Algorithm Comparisons

Algorithm Types

There are 2 main classes of encryption algorithms, symmetric and asymmetric. Symmetric algorithms have a single key for encryption and decryption while asymmetric algorithms have a public encryption key and a private decryption key. [4] This means that asymmetric algorithms are generally more secure. Given that the data for this specific application is not highly sensitive, using an asymmetric algorithm is not necessary. Further, symmetric algorithms have lower resource utilization and are typically much faster than asymmetric algorithms [5] which is necessary for this application. For those reasons, all the algorithms considered were symmetric.

Comparisons

The table below was put together using data from the article PRISEC: Comparison of Symmetric Key Algorithms for IoT Devices by Saraiva et al. [6] as well as from experimentation implementing the encryption algorithms chosen in the Paper Design.

Table 2: Compression Algorithms

Algorithm	Battery Drainage (mAh)
ChaCha20Poly1305	3.9
RC6	8.68

This table highlights that the algorithm used for encryption not only affects the level of encryption but also the battery drainage of encrypting data. This is an important consideration to meet UR03, the system must use minimal computation to be power efficient.It must also be noted that the speed of encryption is also aaffected by the algorithm choice. This is important because in order to reduce the amount of battery drainage overall (not just due to the algorithm), the device must be idle i.e. not running either compression or encryption for as much time as possible. This is to minimize power used to keep the device on.

Analysis

The two algorithms considered after the paper design were ChaCha20Poly1305 and RC6 as both have very low battery drainage when used on ARM architectures. Both also have STM32 libraries and other source code freely available for implementation. These algorithms both ensure better efficiency and minimal battery drainage, whilst being compatible with the STM. It must be noted that the specific library for the chosen algorithm will not be used so as to minimise the size of the encryption code.

After further research, it was found that the ChaCha20Poly1305 algorithm can be implemented using a much smaller file size than RC6 which is important as the STM Discovery board has limited memory available. As little of this space as possible must be used on code implementation to increase the amount of data that can be processed at a single time. Thus, the initial choice of encryption algorithm after the Paper Design was ChaCha20Poly1305.

However, after attempting to implement ChaCha20Poly1305, it was found that the source file was actually 23kB(which is still smaller than that of the RC6 implementations considered). This is too big for this implementation. This algorithm was also very tricky to implement correctly. For this reason, a new algorithm, **RSA, is chosen** for implementation. This algorithm is much simpler and hence has a much smaller source file size of 7kB including 2 extra functions for demonstration purposed. While this algorithm is relatively slow in comparison to the implementation of ChaCha20Poly1305, file size and correct implementation are much more important considerations.

1.3 Inter-Subsystem and Inter-Sub-subsystems Interactions

Below are some additions made to the interactions described in the Paper Design.

The system is designed to first compress the data and then to encrypt it, this is because encryption turns data into high-entropy data, making it random. However, compression relies on patterns in order to preform well and reduce the data size. For these reasons, the data will be compressed and then encrypted in order to observe the best results.
Then, in order to load all the code onto the STM for processing, the compression and encryption algorithms (including all their required methods) have been written into a single c file. This simplifies the implementation and allows for a more cohesive code structure.

2 Data Analysis

2.1 Data to be Used

The SparkFun 9DoF IMU Breakout incorporates the ICM-20948 chip with a logic shifter and GPIO pins for ease of use. This ICM-20948 chip outputs acceleration, gyroscopic and magnetic data in the x, y and z directions. For the purpose of the SHARC Buoy project as described in Jamie Jacobson's dissertation, only the readings from the accelerometer and gyroscope are needed. This is because the buoys will be placed in a region that "suffers greatly from magnetic distortion thereby rendering all magnetic readings to be unreliable." [7] Further, the ICM-20948 measures temperature and this data will also be used.

The test data files supplied include other values such as Yaw, Roll and Pitch (amongst others). All these values can be calculated by using the acceleration and gyroscopic measurements recorded. The calculations to obtain these values will not be done on the buoy to reduce processing time. This is inline with *UR01*. Additionally, the STM32F0 has minimal storage space available and fewer values per recording will allow for more data to be compressed and encrypted at once.
However, the calculation to obtain the actual temperature value (in degrees) will be done onboard the STM32F0 board. This calculation is not computationally expensive and will save space on the STM32F0 board by storing one temperature value instead of two register values.

Ultimately, only acceleration(accelorometer), angular velocity(gyroscope) and temperature data will be used.

2.2 Data-set(s) to use for Testing

The test data used for this report was provided by Prof Amit Mishra.

There are 3 sets of data - 1 in which a person walks around, sits and stands over the course of 5 minutes with an IMU in their pocket and 2 in which the IMU is placed on a turntable with a fixed rotation program about one of its axes over 10 minutes. The one turntable data set has a higher sample rate than the other.

In order to analyse the data, two MATLAB scripts were written. These were used to plot the data in the time domain and to both filter and plot the (filtered) data in the frequency domain. For each of the 3 data sets, a DFT was run on all of the data combined however, the plots were not found to be useful for analysis of the data. Additionally, for each of the 3 data sets, the accelerometer, gyroscope and temperature (as well as time) data was extracted and a DFT was run on each of the 3 measurements. These plots are shown in the figures below:

Time Domain Representations

Data gathered from the accelerometer, gyroscope and temperature sensor are included here. The plot of the combined data is excluded as it does not provide relevant insight into the data.

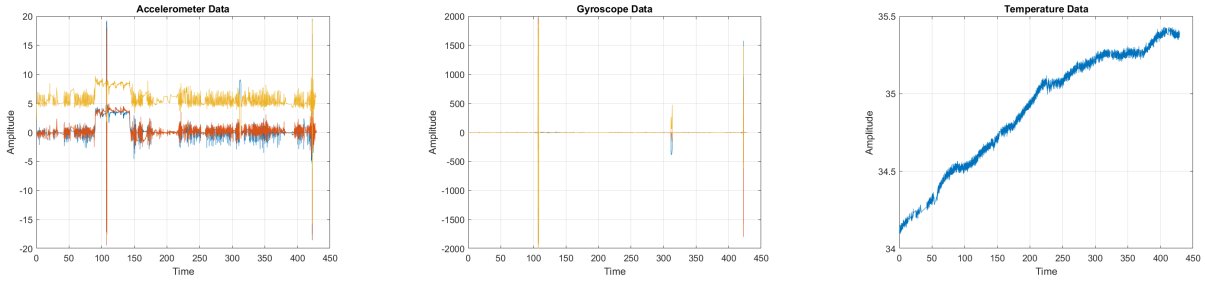

Figure 3: Time Domain representations of Walking Dataset

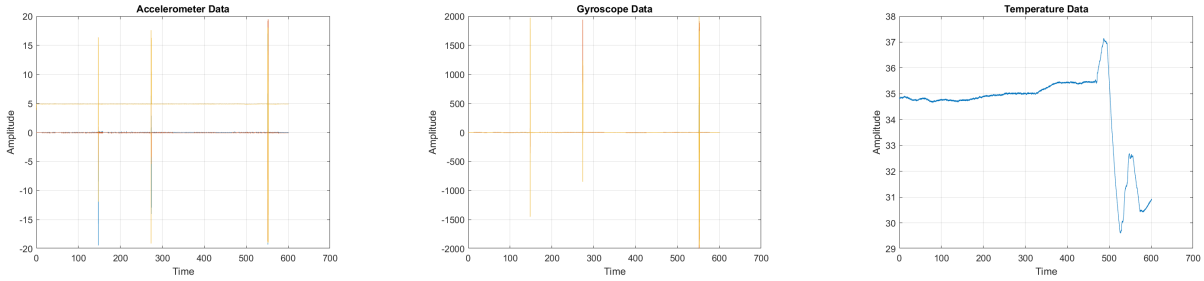

Figure 4: Time Domain representations of Higher Sampled Turntable Dataset

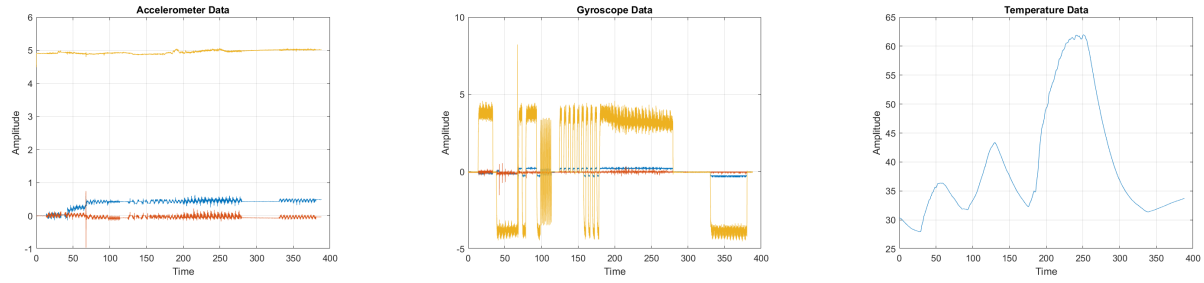

Figure 5: Time Domain representations of Lower Sampled Turntable Dataset

Whilst the time-domain plots show variation in the test data they are not useful for deciding which data-sets should be used for testing. SP01 requires that at least 25% of Fourier coefficients are preserved after compression, this requires analysis of the data in the frequency domain rather than in the time domain.

Frequency Domain Representations

Offset corrected DFTs of the data gathered from the accelerometer and the gyroscope are included here as they give valuable insight into the wave spectrum. The mean wave direction, directional width, skewness and kurtosis parameters of the wave model can be calculated using the Fourier coefficients [7]. This makes the Fourier coefficients valuable and is why they need to be preserved (SP01). Thus, when deciding on which test-data to use for testing, the frequency domain plots are important:

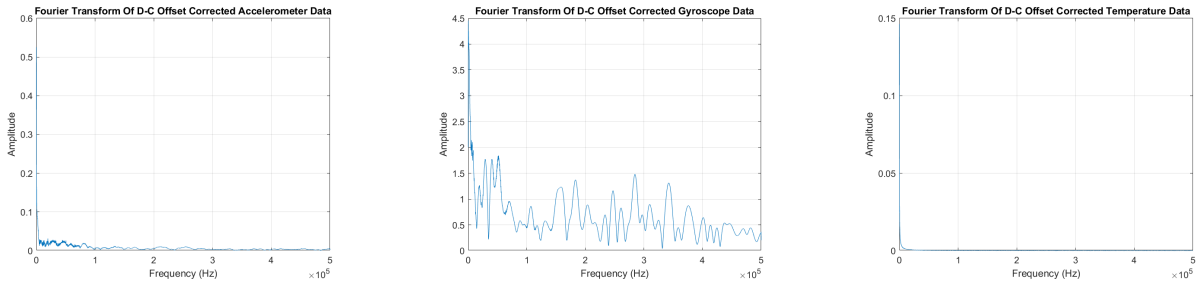

Figure 6: Corrected Offset FFT for Walking Dataset Acceleration, Angular Velocity and Temperature

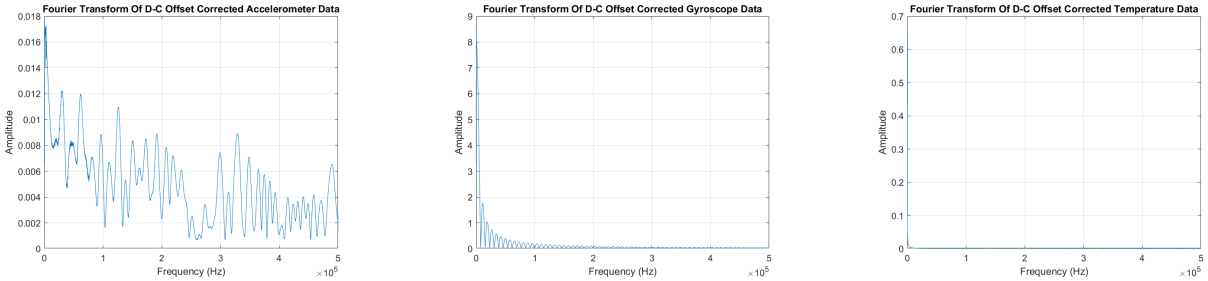

Figure 7: Corrected Offset FFT for Higher Sampled Turntable Dataset Acceleration, Angular Velocity and Temperature

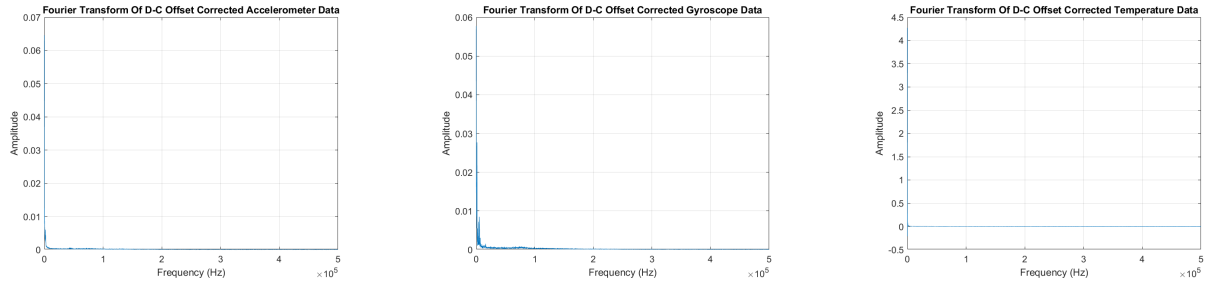

Figure 8: Corrected Offset FFT for Lower Sampled Turntable Dataset Acceleration, Angular Velocity and Temperature

Looking at figure 8, there are few Fourier coefficients in any of the data plots making it less suitable for testing than the data plotted in figure 6 or 7. Between the data plots in figure 6 and 7, figure 7's plots have the greatest number of Fourier coefficients (peaks). Using the data plotted in figure 7 will be a lot more effective in showing what percentage of coefficients are preserved. Thus, the higher sampled turntable data will be used for testing throughout this report.

2.3 Meeting ATPs and Specifications

SP01 and AT02 require at least 25% of coefficients to be preserved after compression. The test-data chosen to perform the experiments in this report has a large number of Fourier coefficients making it easy to see the effects of the compression algorithm on the Fourier coefficients of the data. This will ensure that AT02 is effectively tested for this report.

SP02 and AT04 require the data to be compressed. Dictionary compression algorithms (like lzss) work well with repetitive and similar data and in general, compression algorithms show higher saving percentages on larger data files. Overall, more data that is repetitive or similar will result in better compression.
The decision to only use acceleration, angular velocity and temperature values means that more recordings can be taken and compressed at one time. Since more recordings of similar values can be recorded at a time, the lzss compression will work more effectively. This will ensure that SP02 and AT04 are met.

All other ATPs and specifications do not require any specific form of data and can be tested on any data size.

3 Experiment Setup

The test data used for this report was provided by Prof Amit Mishra.

3.1 Overall System Functionality

3.1.1 Sending data from STM32F0

Aims to determine if data can be transferred correctly and completely from the STM32, inline with AT01.

Using STM32Cube, a project is setup and pins PA2 and PA3 are setup as UART lines using the STM32Cube interface. An array is filled with a sample of data taken from the test data (described in section 2.3). Using loops and string formatting, this data is transmitted over UART to a putty console. The output is saved and, using verify.py, compared to the data stored in the array.

The input data is a sample taken from the supplied test data. The sample contains numerical measurement values taken over different periods of time. The output is expected to be the complete and unchanged sample data.

3.2 Compression and Encryption work together

Aims to determine that the system works, inline with AT08.

The compression and encryption algorithms are combined into a single c file. The test data is passed into the program and the compressed data is passed to the encryption without human interaction.
The input is a csv file containing numerical measurement values taken over a period of 10 minutes. The expected output is an encrypted lzss compressed file.

Thereafter, the file is decrypted and decompressed. The contents of the file is verified against the original test data file using verify.py and by comparing the fourier coefficients (to ensure it meets SP01).
The input data is an encrypted, lzss-encoded file and the expected output is a csv file containing the same data as in the input csv file.
The original test file is compared to the decompressed file using verify.py and FFTCompareScript.m.

3.3 Compression

3.3.1 General

Aims to test that the compression subsystem works inline with AT02 and AT03.

Method
The lzss.c [1] code is setup using the most optimised setting chosen in experiment 3.2.2. A csv file containing test data is compressed using lzss.c and written to a new encoded lzss file. Hereafter, the encoded lzss file is decompressed using the lzss.c code and saved to a new csv file.

For the compression part: The input is a csv file containing numerical measurement values taken over a period of 10 minutes. The expected output is an encoded lzss compressed file which is smaller in size than the original data.
For the decompression part: The expected input is the lzss encoded file and the expected output is a csv file containing the same data as in the input csv file.

The fourier transform of the original and decompressed data is determined and the coefficients are compared to ensure that at least 25% of the coefficients have been preserved. This process is automated using the FFTCompareScript.m script.

3.3.2 Window Sizes

Aims to optimise the algorithm for the test data in line with UR03 and FR05

The LZSS is a dictionary based algorithm. This means it looks for patterns in the data and then assigns an index to the pattern and replaces a match with the index value. This requires the algorithm to use two different windows. A buffer window (to look at the current stream of data) and a look ahead window. These window sizes can be adjusted to find the best fit for certain data. Both values affect the compression ratio as well as the compression speed of the algorithm.

To find the best version of the compression algorithm for the data, a range of window sizes will be tested on the chosen data. The window sizes to be tested are:

Table 3: Compression Algorithms

Buffer size	Look-ahead size
10	4
10	5
11	4
11	5
12	4
12	5
13	4
10	5

Method
For each combination (line in **table 2**) the respective variables in the lzss.c source code [1] will be changed.

Listing 1: LZSS.c variables to be changed [1]

```
#define EI 11  /* typically 10..13 */
#define EJ  4  /* typically 4..5 */
```

The lzss.c code calculates and prints out the compression ratio as a percentage, for each combination this result will be recorded. Then for each run, the compression time will be recorded using a stop watch.

This experiment will need to be performed again when receiving data from the HAT as a way of ensuring that the algorithm is suitably optimised.

3.4 Encryption

3.4.1 General

Aims to test that the encryption subsystem works inline with AT04 and AT05.

AT04 and AT05 require the evaluation of the encryption and decryption algorithms and validation that they do actually encrypt and decrypt the data.

Method
In order to validate the algorithms, a sample set of data must be setup. This is described fully in section 2.1 above. While the initial ATP test protocols for AT04 and AT05 state that the encryption should be done on the STM, this was not done for this submission because the HAT is not yet available and data cannot currently be sent across to the STM.

After the data is setup, it must be encrypted and saved in a new file. The new file will be compared to the original data using a MATLAB script (specifically, the visdiff function). The file will also be opened and viewed to confirm it is not visibly decipherable. These steps will validate the encryption algorithm.

In order to validate the decryption algorithm, the encrypted data will be passed through the decryption algorithm and the decrypted file will be compared to the original set of data. The files should be identical.

3.4.2 Comparison of Encrypted and Decrypted Data

Aims to compare input and output data in line with UR04, FR03, FR04 and SP04

The user requirements, functional requirements and specifications to be met in this section require the data:

- To be encrypted so that less than 10% of the original data remains in the encrypted file; and
- To be 100% recovered (i.e identical to the original data) after decryption

Method
This can be tested as described in section 3.4.1 above. However, the percentage of the file that remained the same after encryption must also be calculated. This would be added to the same script described above to ensure UR04, FR03, FR04 and SP04 are all fulfilled.

The 2 inputs used for the purpose of this report were: When using encrypt(char* msg[]), the following array was used as input:

Listing 2: Encryption Input 1

```
char* c[4] = {"iAmAFile", "hello", "t", "\textbackslash␣n"};
```

In order to test that the function also works for arrays of integers (as the data will be numerical), the following array was tested:

Listing 3: Encryption Input 2

```
char* c[4] = {"13", "14", "15", "\n"};
```

4 Results

4.1 Overall System Functionality

4.1.1 Sending data from STM32F0

AT01 PASS

The experiment completed successfully. The data was transmitted to the PC over UART using the code in this project. The transmitted data was saved and compared to the expected sample data using verify.py and the values in the files match.

4.2 Compression and Encryption work together

AT08 PASS.

The data-set was passed into the combined.c program and the output was an encrypted, compressed file. After passing this file through decryption and decompression, a csv file was obtained.

Using verify.py, the recovered csv and original csv were confirmed to be the same. Using FFTCompareScript.m, the two files are confirmed that all Fourier coefficients were present after compression. The following plots further show that the FFTs are the same for both versions of the data:

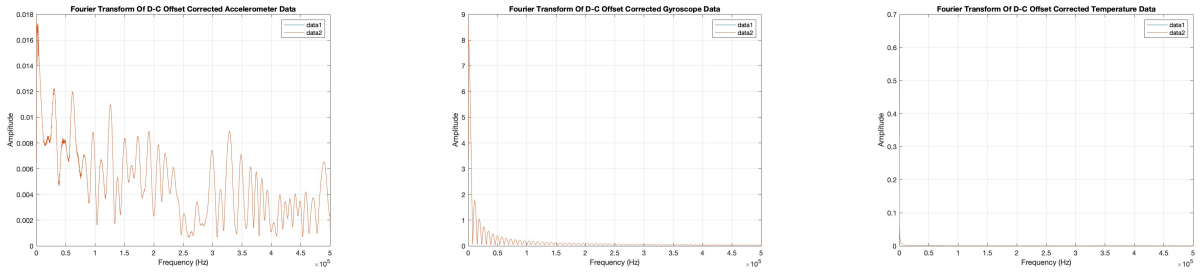

Figure 9: Corrected Offset FFT plots for comparison between data-set before and after compression and encryption

Since the compressed data was passed to the encryption automatically and the Fourier coefficients for both files are the same, AT08 has been passed.

4.3 Compression

4.3.1 General

AT02 and AT03 PASS.

After compressing the csv file (roughly 5MB), the encoded lzss compressed file was 25% of the original size at 1.4MB. This satisfies AT03.

The figures below compare the DFT plots of data before and after compression:

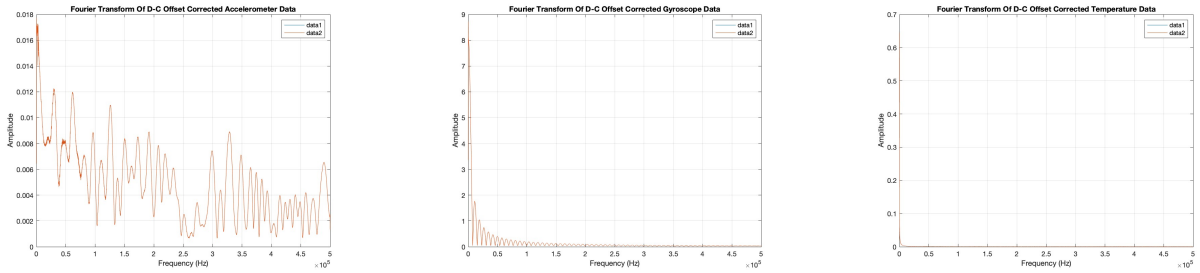

Figure 10: Corrected Offset FFT plots for comparison between data-set before and after compression

In the plots above there is no clear difference between the two plots. This shows that after compression none of the coefficients were lost.
Furthermore, using the FFTCompareScript.m, the coefficients of the data-set before and after compression were compared and found to be exactly the same. This satisfies AT02.

4.3.2 Window Sizes

Table 4: Compression Algorithms

Buffer size	Look ahead size	Ratio (%)	time (s)
10	4	29	3.82
10	5	28	2.83
11	4	27	4.73
11	5	25	5.13
12	4	25	8.55
12	5	23	7.45
13	4	24	13.51
13	5	22	11.27

The results in table 4 indicate that the best compression ratio is achieved when combining a buffer and look ahead size of 13 and 5. However, this is also the combination with the highest compression time. Choosing this option would not satisfy UR03.
The combination with the lowest compression time is a buffer and look ahead size of 10 and 5 respectively. However, this is also the combination with the second highest compression factor (biggest file size after compression).

Thus, choosing the combination of buffer and look ahead size of 11 and 5 is the best fit. It reduces the file to 25% of its original size and it has a very low compression time. This satisfies UR03.

4.4 Encryption

4.4.1 General

AT04 and AT05 PASS

Note: As the compression subsystem will pass an array to the encryption function, the encrypt() function does not take in a file like the other functions, but rather an array.

ATP04 requires the output to be unreadable without the key after encryption. For the purpose of this experiment, this can be done visually.
In order for AT05 to be passed, the decrypted file must be exactly the same as the original plaintext file. This

can be tested by simply looking at the files and comparing them for smaller inputs or by using a script(verify.py was used for this experiment) for larger sample sizes. Both methods were used here.

The encrypted output for Encrypted Input 1 (written to ciphertext.csv) was:

” 1157625
274625
1295029
274625
343000
1157625
1259712
1030301
0
1124864
1030301
1259712
1259712
1367631
0
1560896
0 ”

The text is converted entirely to integers which cannot be deciphered by simply looking at them and without knowing how those integers were acquired. Thus, AT04 is passed for this text case.

The decrypted output (written to ”decrypted.csv”) was:

iAmAFile hello t

It must be noted here that ”\n” is used as a marker for the end of data in this function and thus, this is the expected decrypted output. Thus, AT05 is passed for this test case.
The encrypted output for Encrypted Input 2 was:

117649
132651
0
117649
140608
0
117649
148877
0

Once again, the encrypted output cannot be used to deduce the actual data and thus, AT04 is passed for this sample set.
The decrypted output (to the decrypted.csv file) was:

13 14 15

This once again is identical to the input data and hence, AT05 is passed for this sample set.

5 Acceptance Test Procedures

Success and Failure tests:

Table 5: Assessment of ATP success or failure

Acceptance test	System	Success
AT01	Input	PASS
AT02	Compression	PASS
AT03	Compression	PASS
AT04	Encryption	PASS
AT05	Encryption	PASS
AT06	Full System	UNTESTED
AT07	Full system	UNTESTED
AT08	Full System	PASS

AT07: we do not have access to the HAT yet so could not be tested for the purpose of this report.
AT06: the code has not yet been implemented on the STM32F0 board so, AT06 could not be tested.
AT08: is a new ATP, added to test that the algorithms work together. While the functionality it tests is similar to that of AT06, it does not require the code to be implemented on the STM32F0.

Table 6: Acceptance Test Protocols used to determine that the system works to specification

Acceptance test	System	Description
AT01	Input	Ensure data is transferred correctly and completely from computer to STM32F051
AT02	Compression	Ensure that at least 25% of lower Fourier coefficients of data are preserved
AT03	Compression	Ensure file size is decreased after compression
AT04	Encryption	Ensure that the data is not readable without the key after encryption
AT05	Encryption	Data is readable and the same as the original once decrypted
AT06	Full System (no HAT)	Ensure compression and encryption works when implemented together on STM32F0
AT07	Full system	Ensure that data read from the sensor completes a full cycle meeting all requirements and above ATPs
AT08	Full system	Ensure compression and encryption works when implemented together

5.1 AT01

AT01	**Connection Test**
Evaluation type	Comparison
Target	STM32F051
Test Protocol	1. Set up test case using a sample set of data 2. Pass data from device to STM32F051 3. Pass data from STM32F051 to computer. 4. Compare data read and passed to computer to original sample set of data.
Pass Condition	• Data is in the same format as original data • Data values are the same as original data • All original data is present in the new data
Fail condition	• Data format differs from original • Any data value is different from the original data • Any data values are missing Extra data values are present in the new data

5.2 AT02

AT02	**Fourier Coefficients Test**
Evaluation type	Comparison
Target	Compression Subsystem
Test Protocol	1. Set up test case using a sample set of data and take 25% the lower Fourier coefficients. 2. Compress the original data using the compression algorithm on the STM32F051. 3. Decompress the data and compute the Fourier coefficients on a computer. 4. Compare the Fourier coefficients to the test case.
Pass Condition	• All of the Fourier coefficients present in test case are present in decompressed data (extra coefficients may be present but not less).
Fail condition	• Fourier coefficients present in the test case are not present in the decompressed data.

5.3 AT03

AT03	**Compression Ratio Test**
Evaluation type	Comparison
Target	Compression Subsystem
Test Protocol	1. Set up test cases using sample set of data and make note of file sizes. 2. Compress the original data using the compression algorithm on a computer. 3. Compare the size of the new file to that of the original file.
Pass Condition	• New file size is less than original file size.
Fail condition	• New file size is greater than or equal to old file size.

5.4 AT04

AT04	**Encryption key Test**
Evaluation type	Algorithm Validation
Target	Encryption Subsystem
Test Protocol	1. Set up test cases using sample set of data. 2. Encrypt data using the encryption algorithm and key on the STM32F051. 3. Check if data can easily be deduced or read after encryption on a computer.
Pass Condition	• Data is not deducible or human readable after encryption.
Fail condition	• Data can easily be deduced or read after encryption.

5.5 AT05

AT05	**Decryption validation Test**
Evaluation type	Algorithm Validation
Target	Encryption Subsystem
Test Protocol	1. Set up test cases using sample set of data. 2. Encrypt data using the encryption algorithm and key on the STM32F051. 3. Decrypt data using the decryption algorithm and key on a computer. 4. Compare new data to original data set.
Pass Condition	• Decrypted data is identical to original sample set data.
Fail condition	• There are differences between sample set data and decrypted data.

5.6 AT06

AT06	**System Test**
Evaluation type	System execution
Target	Full System (no HAT)
Test Protocol	1. Set up test case using sample data set and compute 25% o the lower Fourier coefficients. 2. Pass sample data through compression and then encryption algorithm on the STM32F051. 3. Perform decryption and then decompression on the computer. 4. Compute 25% of lower Fourier coefficients of the data. 5. Compare the coefficients to the test case.
Pass Condition	• Encryption and decryption complete without failure. • AT01 is met.
Fail condition	• Encryption or decryption fail. • AT01 is not met.

5.7 AT07

AT07	**System Test with Wave sensor HAT**
Evaluation type	System Execution
Target	Full System
Test Protocol	The Wave sensor HAT is connected to STM32F051. The data from the HAT is first printed to the computer and then sent to be compressed (onboard the STM32F051). 1. The data printed to the computer is used as the sample data set and 25% of the lower Fourier coefficients are computed. 2. Data is sent through compression and encryption on the STM32F051 and then transferred to the computer. 3. Perform decryption and then decompression on the computer. 4. Compute 25% of lower Fourier coefficients of the data. 5. Compare the coefficients to the test case.
Pass Condition	• Data is successfully read from the HAT and printed to the computer. • AT06 is met
Fail condition	• Data is not read from the HAT (there is not change in data or no data printed to the computer) AT06 is not met

5.8 AT08

AT08	**System Test**
Evaluation type	System execution
Target	Full System (no HAT)
Test Protocol	1. Set up test case using sample data set and compute 25% o the lower Fourier coefficients. 2. Pass sample data through compression which must then pass onto encryption algorithm without human interaction. 3. Perform decryption and then decompression on the data. 4. Compute 25% of lower Fourier coefficients of the data. 5. Compare the coefficients to the test case.
Pass Condition	• Compression and encryption complete without failure. • Decryption and decompression complete without failure. • AT01 is met.
Fail condition	• Compressed data is not passed to encryption algorithm without human interaction. • Encryption or compression fail. • Decryption o decompression fail. • AT01 is not met.

References

[1] Haruhiko Okumura, "LZSS encoder-decoder," [Online]. Available: https://oku.edu.mie-u.ac.jp/~okumura/compression/lzss.c, [Accessed: 6 September 2022].

[2] Joshua Lockerman, Ajay Kulkarni, "Time-Series Compression Algorithms, Explained," [Online]. Available: https://www.timescale.com/blog/time-series-compression-algorithms-explained/, [Accessed: 22 August 2022].

[3] , "IzBench," [Online]. Available: https://github.com/inikep/lzbench, [Accessed: 22 August 2022].

[4] Cloudflare, "What is encryption? — Types of encryption," [Online]. Available: https://www.cloudflare.com/learning/ssl/what-is-encryption/#:~:text=An%20encryption%20algorithm%20is%20the,by%20using%20the%20decryption%20key, [Accessed: 19 August 2022].

[5] Abhishek Tiwari, "Difference Between Symmetric and Asymmetric Key Encryption," [Online]. Available: https://www.geeksforgeeks.org/difference-between-symmetric-and-asymmetric-key-encryption/, [Accessed: 19 August 2022].

[6] D. A. F. Saraiva, V. R. Q. Leithardt, D. de Paula, A. Sales Mendes, G. V. González, and P. Crocker, "Prisec: Comparison of symmetric key algorithms for iot devices," *Sensors*, vol. 19, no. 19, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/19/4312

[7] J. N. Jacobson, "Sharc buoy: Robust firmware design for a novel, low-cost autonomous platform for the antarctic marginal ice zone in the southern ocean," Ph.D. dissertation, Rondebosch, Cape Town South Africa, 2021.

6 Appendix A: Requirements

Table 7: Requirement Traceability Matrix

UR	FR	Specifications	ATP
UR01	FR01	SP01	AT02
UR02	FR06	SP06	AT07
UR03	FR05	SP05	
UR04	FR03, FR04	SP04	AT04, AT05
UR05	FR01, FR02	SP02	AT03
UR06	FR07	SP07	AT01

User Requirements

Table 8: User requirements used to determine the design and functionality of the desired IP

User Requirement	Description
UR01	System must minimise data loss
UR02	System must be compatible with ICM-20649 and ICM20948 IMU chips
UR03	System must use minimal computation to be power efficient
UR04	System must encrypt the data
UR05	System must compress data
UR06	System must be able to transmit data to another device

Functional Requirements

Analysis of the user requirements resulted in the following functional requirements which describe how the system will function.

Table 9: Requirements addressing the needs of the system

Requirement	Description
FR01	System to compress data such that 25% of the lower Fourier coefficients are recoverable after decompression
FR02	Compression subsystem to pass processed data to the encryption subsystem
FR03	All compressed data must be encrypted
FR04	Encryption subsystem to encrypt data in a manner that allows efficient and complete decryption
FR05	System must use minimal computation and processing
FR06	All code must be implementable for a system using either ICM-20649 or ICM-20948
FR07	System to transmit data via USB to another device for decryption and decompression

Specifications

Table 10: Specifications of design to repeat user requirements

Specification ID	Description
SP01	A minimum of 25% of the lower Fourier coefficients must be present after decompression
SP02	The compression ratio must be less than 1, i.e. the compressed file must be smaller than the original file.
SP03	100% of the compressed data must be recovered after decryption.
SP04	Less than 10% of the original data can be the same as the original file after encryption.
SP05	All code should be written in C
SP06	I2C must be run on fast mode when developing and testing (ICM-20649 standard)
SP07	System must transfer the data over UART following encryption

SP04(Less than 20mAh of power must be used in encrypting the data): was removed as this cannot be measured accurately. It would be impossible to differentiate between the power used solely for encrypting the data and all other tasks being completed by the STM.