



Final Report: Milestone 4

Yashil Luckan and Pooja Jugnarayan*
Department of Electrical and Computer Engineering
University of Cape Town

October 15th, 2022

* Email addresses lckyas002@myuct.ac.za, and jgnpoo001@myuct.ac.za.

Table of Contents

1. Admin documents	5
1.1. Table of individual contributions	5
1.2. Project management tool	6
1.3. Github link	6
1.4. Timeline	7
2. Requirement analysis and paper design	8
2.1. Interpretation of the requirements	8
2.2. Requirements analysis & background to the project	8
2.3. Comparison of compression and encryption algorithms	9
2.3.1. Compression comparisons	9
2.3.2. Encryption comparisons	10
2.4. Feasibility analysis	11
2.4.1. Compression feasibility	11
2.4.2. Encryption feasibility	11
2.5. Possible bottlenecks	11
2.6. Subsystem design	12
2.6.1. Subsystem and sub-subsystem requirements	12
2.6.1.1. Compression subsystem requirements	12
2.6.1.1.1. STM transmission sub-subsystem requirements	12
2.6.1.2. Encryption subsystem requirements	13
2.6.1.2.1. STM transmission sub-subsystem requirements	13
2.6.2. Subsystem and sub-subsystem specifications	14
2.6.2.1. Compression subsystem specification	14
2.6.2.1.1. STM transmission sub-subsystem specifications	14
2.6.2.2. Encryption subsystem specification	14
2.6.2.2.1. STM transmission sub-subsystem specifications	14
2.6.3. UML diagram	15
3. Validation Using Simulated Data	16
3.1. Data	16
3.1.1. Data used	16
3.1.2. Data justification	16
3.2. Experiment setup using simulated data	17
3.2.1. Simulations and experiments to determine overall functionality	17
3.2.2. Subsection experiments	17
3.2.2.1. Compression experiments	17

3.2.2.2.	Encryption experiments	17
3.2.3.	Data types	18
3.3.	Results.....	19
3.3.1.	Overall functionality	19
3.3.2.	Subsection functionality	20
3.3.2.1.	Compression functionality	20
3.3.2.2.	Encryption functionality	22
3.3.3.	Additional Testing	22
4.	Validation Using the IMU	23
4.1.	IMU Module	23
4.1.1.	IMU analysis (ICM-20948 & ICM-20649)	23
4.1.2.	Steps to ensure accuracy between multiple IMU's.....	23
4.1.3.	The need for IMU validation	23
4.1.4.	IMU validation tests.....	24
4.2.	Experiment setup using IMU data	24
4.2.1.	Simulations and experiments to determine overall functionality.....	24
4.2.2.	Subsection experiments	25
4.2.2.1.	Compression experiments.....	25
4.2.2.2.	Encryption experiments	25
4.2.3.	Data types	25
4.3.	Results.....	26
4.3.1.	Overall functionality	26
4.3.2.	Subsection functionality	27
4.3.2.1.	Compression functionality	27
4.3.2.2.	Encryption functionality	28
4.3.3.	Changes in performance	29
5.	Acceptance test procedure	30
5.1.	Figures of merit.....	30
5.1.1.	Compression algorithm (GZIP) figures of merit	30
5.1.2.	Encryption algorithm (AES) figures of merit	30
5.2.	Experiment design	30
5.3.	Acceptance performance definition	31
5.4.	ATP Performance	31
5.5.	ATP Progress.....	32
5.6.	Traceability Matrix	32
6.	Consolidation of ATP's and Future Plan.....	33

7. Conclusion	33
8. References	34

1. Admin documents

1.1. Table of individual contributions

Item in table of contents	Contributors
Compression	Yashil
Encryption	Pooja
Interpretation of the requirements	Yashil and Pooja
Requirement analysis	Yashil and Pooja
Compression comparison	Yashil
Encryption comparison	Pooja
Compression feasibility	Yashil
Encryption feasibility	Pooja
Possible bottlenecks	Yashil and Pooja
Compression subsystem requirements	Yashil
Encryption subsystem requirements	Pooja
STM transmission sub-subsystem requirements	Yashil and Pooja
Compression subsystem specification	Yashil
Encryption subsystem specification	Pooja
STM transmission sub-subsystem specifications	Yashil and Pooja
UML diagram	Yashil and Pooja
Compression algorithm (GZIP) figures of merit	Yashil
Encryption algorithm (AES) figures of merit	Pooja
Experiment design	Yashil and Pooja
Acceptance performance definition	Yashil and Pooja
Traceability matrix	Yashil and Pooja
Data used and data justification	Yashil and Pooja
Initial analysis	Yashil and Pooja
Overall experiments	Yashil and Pooja
Compression experiments	Yashil
Encryption experiments	Pooja
Data types	Yashil and Pooja
Overall functionality experiments	Yashil and Pooja
Compression functionality	Yashil
Encryption functionality	Pooja
Additional Testing	Yashil and Pooja
Salient features	Yashil and Pooja
Steps to ensure accuracy between multiple IMU's	Yashil and Pooja
IMU validation tests	Yashil and Pooja
Simulations and experiments to determine overall functionality	Yashil and Pooja
Compression experiments	Yashil
Encryption experiments	Pooja
Data types	Yashil and Pooja
Overall functionality	Yashil and Pooja
Compression functionality	Yashil
Encryption functionality	Pooja
Changes in performance	Yashil and Pooja
Consolidation of ATP's and Future Plan	Yashil and Pooja
Conclusion	Yashil and Pooja

1.2. Project management tool

The project management tool of choice is Asana. The image above shows the breakdown of the tasks at hand.

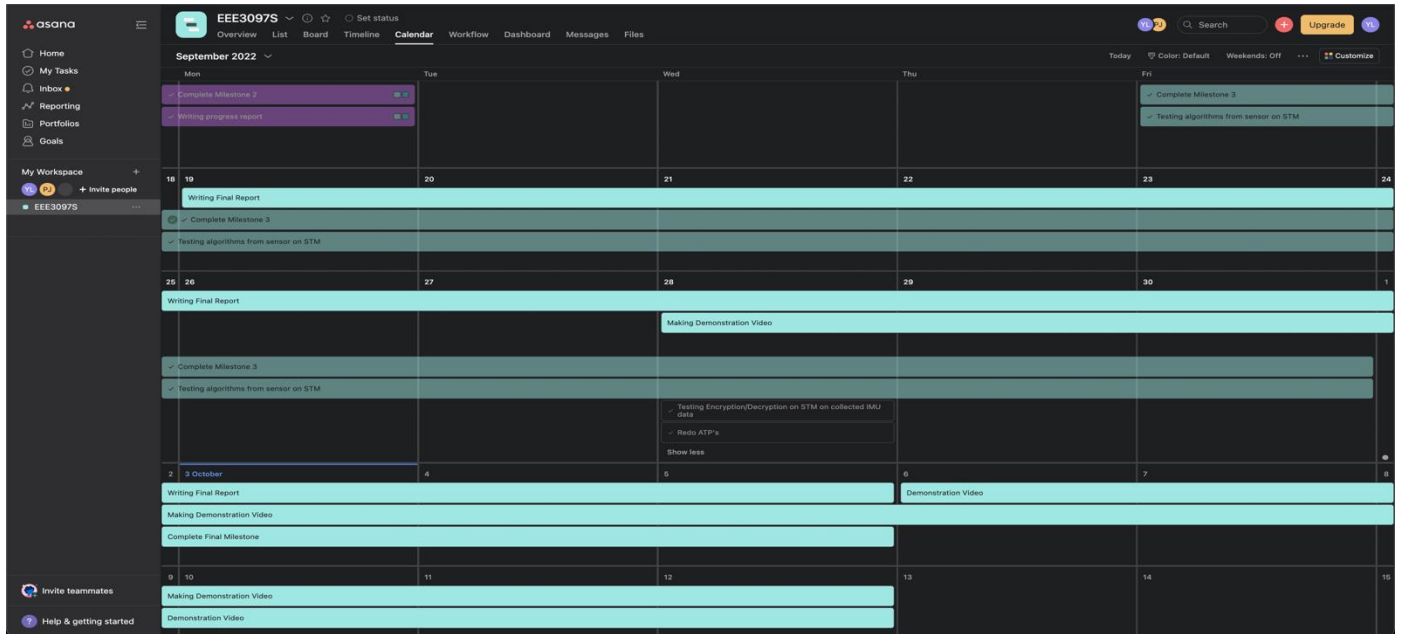
Category	Task	Due Date	Progress
To do	Redo ATP's	28 Sep	0%
	Testing Encryption/Decryption on STM on collected IMU data	28 Sep	0%
	Demonstration Video	12 Oct	0%
	Complete Final Milestone	Wednesday	0%
	Complete Milestone 3	30 Sep	0%
	Testing algorithms from sensor on STM	30 Sep	0%
	Making Demonstration Video	12 Oct	0%
Doing	Complete Milestone 2	12 Sep	0%
	Writing test algorithms	7 Sep	0%
	Testing algorithms on computer	9 Sep	0%
	Writing progress report	12 Sep	0%
Done	Complete Milestone 1	23 Aug	100%
	ATP	21 Aug	100%
	UML Diagrams	18 Aug	100%
	Requirement Analysis	16 Aug	100%
	Comparison of Compression Algorithms	17 Aug	100%
	Comparison of Encryption Algorithms	17 Aug	100%
	Feasibility Analysis & Possible Bottlenecks	18 Aug	100%
	Subsystem Requirements & Specifications	18 Aug	100%

1.3. Github link

<https://github.com/yashilluckan/EEE3097-Team6.git>

1.4. Timeline

The above picture shows the Asana timeline, and we are on track and the tasks at hand at this point in time have been completed i.e., progress is on time.



2. Requirement analysis and paper design

2.1. Interpretation of the requirements

Requirement Number	Requirement Description	Requirement Interpretation
U1	IMU to be used: ICM-20649 IMU provided: ICM-20948	Our system must be compatible with both the ICM-20649 as well as the ICM-20948. This includes all subsystems, encryption and compression algorithms.
U2	At least 25% of Fourier coefficients of the data must be able to be extracted.	Our compression algorithm must ensure that at least 25% of the Fourier coefficients of the data must be extracted. Both compression and encryption/decryption must not impact the fidelity of the data to such an extent.
U3	Reduce the amount of processing executed by the processor.	Our encryption and compression algorithms must not be CPU-intensive. Minimise the computation on these algorithms to save power as power is a limited resource in the application. The algorithms must also run as quickly as possible to reduce the amount of power consumed.
U4	All algorithms will run either on STM32F0DISCOVERY MCU or using IMU data on a computer.	All algorithms and the chosen programming language must be compatible with the constraints imposed by the STM32F0DISCOVERY. Such as 64KB Flash memory and 8KB RAM in an LQFP64 package.

2.2. Requirements analysis & background to the project

Before the requirements are analysed, it is important to briefly introduce the reason this project was tasked in the first place. A flexible buoy that is can be installed on the pancake ice in the Southern Ocean to gather environmental data. The buoy is called SHARC and implements an IMU (ICM-20649) to gather the environmental data via an accelerometer and gyroscope. We have been tasked with designing an ARM-based digital IP using an STM32F051 to encrypt and compress the data received by the IMU. The IMU provided is very similar to the IMU that will be used on the SHARC buoy as seen in 4.1.1. No data would be lost due to the use of lossless compression (GZIP) and encryption algorithms. The amount of processing will be kept to a minimum by implementing non-intensive algorithms due to the limited power source. The fact that the compression and encryption algorithms and IMU libraries should run on the STM implies that they must be less than 64kB combined and that the language used to program the STM must be ANSI C as most other languages are too large and the STM was designed to work with the C programming language.

2.3. Comparison of compression and encryption algorithms

2.3.1. Compression comparisons

The first question that arises when considering compression algorithms is: lossy or lossless compression? Lossy compression techniques involve some loss of data. Data that is compressed using lossy techniques cannot be reversed or reconstructed exactly.[1] On the other hand, when lossless compression techniques are used, the original file can be reconstructed exactly. This means that lossy data compression is not well suited for text-based situations but better suited for images and video. It is much harder to spot a missing pixel than it is to spot missing text for example. Types of lossy compression algorithms included for this comparison are Transform Encryption and Discrete Cosine Transform. Types of lossless compression algorithms include Lempel-Ziv-Welch (LZ77, LZMA) and Huffman Coding. To simplify matters further, it would be much simpler to use pre-existing lossless compression algorithms that make use of the abovementioned algorithms. BZIP2 uses Huffman coding and GZIP uses a combination of Huffman coding and LZ77 and LZIP uses LZMA. Therefore, in this comparison, BZIP2 and GZIP will be considered. Based on a study where a Ryzen 2600 with a Samsung SSD in RAID1 was used to compress a 939MB file using various lossless compression algorithms, the following results were obtained: [2]

Algorithm	Time [s]	Speed Ranking	Size [MB]	Compression Ratio Achieved	Compression Ratio Ranking	Command	CPU Usage	Power Usage	Memory Usage
GZIP	23.5	3 rd	177	5.3	4 th	Gzip	1 core (100%)	Low	Very low
Parallel GZIP	3.1	1 st	177	5.3	4 th	Pigz	All cores	Very Low	Low
BZIP2	60.7	4 th	134	7.0	3 rd	bzip2	1 core (100%)	Medium	Low
Parallel BZIP2	9	2 nd	135	7.0	3 rd	Ipbzip2	All cores	Very Low	Medium
LZIP	282	6 th	116	8.1	2 nd	Lzip	One core (100%)	High	Medium
Parallel LZIP	99.7	5 th	110	8.5	1 st	Plzip	All cores	High	High

Table 1: Comparison of compression algorithms

Upon comparison of the above data, LZIP compression algorithms achieves the best and 2nd best compression ratios. However, this comes at a cost as it uses the most memory resources, power and takes the longest time. BZIP2 compression algorithms achieve the 3rd best compression ratios with its parallel algorithm achieving the 2nd best speed and low power usage and its serial algorithm using more power and taking longer. GZIP is the winner in terms of power and memory usage when it comes down to serial application. However, it achieved the worst compression ratio across the board. The two compression algorithms that will be tested are GZIP and BZIP2.

2.3.2. Encryption comparisons

When considering encryption algorithms, the first decision to be made is: Asymmetric or Symmetric. Symmetric encryption involves the use of the same key for both encryption and decryption, thus the need for a secure transfer of the key is essential whilst Asymmetric encryption uses 2 keys, a public key for encryption and a private key for decryption, thus eliminating the need for secure transfer of the key. [3] The advantage of symmetric encryption is that it is significantly faster and requires less computational power which makes it seem ideal for the above scenario. [4] Both types of encryptions satisfy R2 as there is no data loss involved.

Types of symmetric encryption algorithms include DES, 3DES and AES. DES or data encryption standard which converts 64-bit blocks of plaintext data into ciphertext by dividing the block into smaller 32-bit blocks and applying encryption to each block. This involves 16 rounds of performing differing processes. This results in a 64-bit block being produced. The biggest disadvantage of DES is the length of the cipher which makes it easy to brute force and crack. The next symmetric encryption method is 3DES or triple data encryption which is simply an updated version of DES. It basically applies the DES algorithm 3 times which makes it 3 times harder to crack. This algorithm seems like an unlikely choice for the above application as it would use 3 times the computational power and whilst still not being the most secure which is not practical when low power is the priority. The last symmetric encryption algorithm to be compared is AES or advanced encryption system which uses methods of substitution and permutation. It involves the plaintext data being put into block and the encryption being applied using a key. The number of rounds performed depends on the size of the key which implies that a smaller key would allow for less computation and thus less power consumed and time elapsed. [4]

Factors	Key Length	Rounds	Block Size	Cipher Type	Speed	Security	Percentage Efficiency [%]	Memory Usage [kB]
DES	56 bits	16	64 bits	Symmetric Block Cipher	Slow	Not Secure Enough	30	18.2
3DES	168 bits or 112 bits	48	64 bits	Symmetric Block Cipher	Very Slow	Adequate Security	60	20.7
AES	128 bits, 192 bits or 256 bits	10, 12 or 14	128 bits	Symmetric Block Cipher	Fast	Excellent Security	90	14.7
RSA	Variable	1	Variable	Asymmetric Block Cipher	Slowest	Least Secure	65	31.5

Table 2: Comparison of encryption algorithms [5],[6]

From the comparison table above, it is again made clear that symmetric algorithms are a better fit as the lowest performing symmetric algorithm, DES performs better than RSA, a asymmetric algorithm. It is also clear that AES is the front runner from the symmetric algorithms due to its speed, efficiency, and memory usage. From the above algorithms, AES and DES will be tested as they have the lowest memory usage.

2.4. Feasibility analysis

2.4.1. Compression feasibility

In order to choose between lossless and lossy data compression, the application must be considered first. In this specific application data is being written to a CSV file which is text-based. As stated in Section 1.2.1, lossy compression is not widely used in text-based applications. One of the Requirement R2 states that at least 25% of the lower Fourier coefficients should be available to be extracted. With lossless data compression the file will be able to be reconstructed exactly, therefore there is a guarantee that regardless of the file size and data complexity that the original file will be exactly reconstructed and thus satisfy requirement R2 fully. Lossless data compression algorithms are more widely used in text-based applications and is much easier to implement than lossy data compression algorithms as those are mainly used for images and video application. Upon careful analysis of the comparisons made in Section 1.2.1, GZIP will be our algorithm of choice. GZIP achieves the best serial (non-parallel) time. The reason why the non-parallel algorithm is chosen is due to the fact that the STM32Discovery/Raspberry Pi boards will not have multiple cores and thus parallel algorithm implementation will not be feasible and benefits will not be noticed. Despite the fact that GZIP achieved the worst compression ratios, it uses the least memory power as it runs for less time than BZIP2 and LZIP. Less run time will mean less power is consumed. Power and memory is a limited resource in this specific application and thus pros of power and memory consumption outweigh the cons of compression ratio making it the most feasible for our application as it can also be run. GZIP and LZIP are not feasible for this application as uses too much power and memory and larger datasets will take long to compress.

2.4.2. Encryption feasibility

The feasibility of the encryption algorithms to be used comes down to how little power is consumed which is a combination of the computational power of the algorithm and the time taken for the algorithm to run as stated in R3. The other less important concern is the security of the algorithm. All algorithms above satisfy R2 as there is no data loss through the encryption process. From 1.2.2 it is clear that a symmetric approach is preferred over an asymmetric approach as it uses less computational power and is quicker whereas an asymmetric approach is more secure. AES is the fastest and safest of the symmetric algorithms and the computational power of the algorithm can be reduced by use of a smaller key and thus it seems most feasible for this project.

2.5. Possible bottlenecks

- Memory is limited on the STM, and if sampling rates exceed speeds of 8KBps then the data collection will not be successful.
- Storage is limited to 64KB on the STM so if the data collected, IMU libraries, encryption libraries and compression libraries exceed this figure then the system will end up in a bottleneck state.
- The applications could take too long to run resulting in a large delay and too much power being consumed.
- The compression and encryption algorithms could prove to be incompatible.
- The algorithm could be unstable on fixed point systems.
- Compression could fail to make the file small enough to transfer.
- Encryption key could be lost resulting in the data being unable to be decrypted.
- When transmitting data from the IMU to the STM and to the computer the input buffer of each stage i.e., the SPI interface between the IMU and the STM and the UART interface between the STM and the computer, could be overloaded if data is transmitted at a rate that is too high

2.6.Subsystem design

2.6.1. Subsystem and sub-subsystem requirements

2.6.1.1. Compression subsystem requirements

- FR1: Requirement U1 states that each algorithm must be compatible with the ICM-20649 as well as the ICM-20948 which will be tested using ATP A1.
- FR5: Requirement U2 states that at least 25% of the lower Fourier coefficients must be able to be retrieved which will be tested using ATP A4.
- FR9: Requirement U3 states that the amount of processing power must be reduced so as to minimise power consumption which will be tested using ATP A6.
- FR13: Requirement U4 states that our algorithms must be able to run on the STM or a computer and will be tested using ATP A7.

2.6.1.1.1. STM transmission sub-subsystem requirements

- FR2: Throughput must be synchronised to the output and input port of the IMU and STM and will be tested using ATP A2.
- FR6: After data loss as a result of transmission the file must retain 25% of the lower Fourier coefficients and will be tested using ATM A4.
- FR10: Latency should be kept to a minimum in order to reduce power consumption and will be tested using ATP A6.
- FR14: The data that is to be transmitted must be kept to below 64KB due to the memory on the STM and will be tested using ATP A7.

2.6.1.2. Encryption subsystem requirements

- FR3: From requirement U1 it can be inferred that the encryption algorithm must be compatible with the ICM-20649 as well as the ICM-20948 and will be tested using ATP A3.
- FR7: From requirement U2 it can be inferred that no data must be lost when the file is encrypted as at least 25% of the lower Fourier coefficients from the decrypted file must be able to be retrieved and will be tested using ATP A5
- FR11: From requirement U3 it can be inferred that the encryption algorithm must use as little computational power as possible and will be tested using ATP A6.
- FR15: From requirement U4 it can be inferred that the encryption algorithm must be less than half of 64KB and will be tested using ATP A7.

2.6.1.2.1. STM transmission sub-subsystem requirements

- FR4: Throughput must be synchronised to the output and input port of the IMU and STM and will be tested using ATP A2.
- FR8: After data loss as a result of transmission the file must retain 25% of the lower Fourier coefficients and will be tested using ATP A5.
- FR12: Latency should be kept to a minimum in order to reduce power consumption and will be tested using ATP A6.
- FR16: The data that is to be transmitted must be kept to below 64KB due to the memory on the STM and will be tested using ATP A7.

2.6.2. Subsystem and sub-subsystem specifications

2.6.2.1. Compression subsystem specification

- S1: The filetype produced by the IMU is compatible with GZIP compression.
- S5: GZIP is lossless. Therefore 100% of the Fourier coefficients will be present after extraction.
- S9: GZIP is the fastest algorithm for compression and will thus use the least power
- S13: Compression algorithm (GZIP) can be implemented on fixed point and floating-point systems. Thus, making it compatible with the STM and GZIP can be implemented using 32KB and thus be compatible with the STM.

2.6.2.1.1. STM transmission sub-subsystem specifications

- S2: Transmission from the IMU via jumper cable and USB will not result in the loss of data and throughput is 1 KBps.
- S6: Transmission via jumper cable and USB will not result in loss of data and thus 100% of the Fourier coefficients will be present after extraction.
- S10: Latency Mini USB is 5.2 nanoseconds/meter.
- S14: Data is in a format that can be understood by the STM.

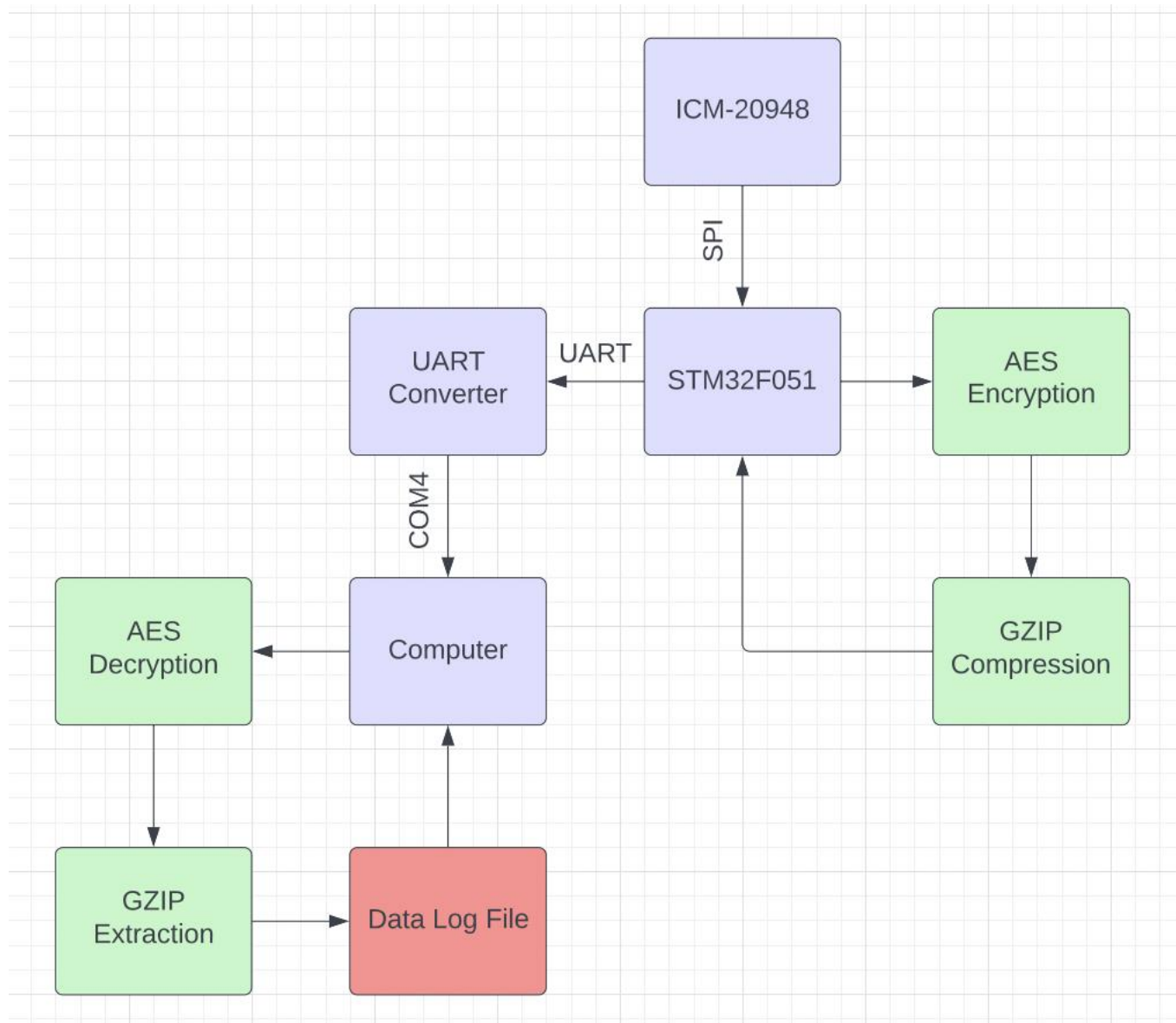
2.6.2.2. Encryption subsystem specification

- S3: The filetype produced by the IMU is compatible with AES encryption.
- S7: AES does not experience data loss thus the output will have 100% of the lower Fourier coefficients.
- S11: AES is a fast and efficient algorithm and thus will have a low power consumption.
- S15: The encryption algorithm, AES is compatible with systems from 8-bit to 64-bits thus AES is compatible with the STM and the AES algorithm is 14.7KB which is less than the available 64KB on the STM.

2.6.2.2.1. STM transmission sub-subsystem specifications

- S4: Transmission from the IMU via jumper cable and USB will not result in the loss of data and throughput is 1 KBps.
- S8: Transmission via jumper cable and USB will not result in loss of data and thus 100% of the Fourier coefficients will be present after decryption.
- S12: Latency Mini USB is 5.2 nanoseconds/meter.
- S16: Data is in a format that can be understood by the STM.

2.6.3. UML diagram



3. Validation Using Simulated Data

3.1.Data

3.1.1. Data used

The data to be used are CSV files that were communicated to us via email from Humphrey Chiramba. The following CSV files were chosen because it is data that was captured by the IMU, and the raw data is in a readable format.

The CSV files were modified to contain only 1000 lines of data, as this size still allowed the compression to work effectively whilst being small enough to mimic the size of the data files that will be created from the data received from the IMU.

The CSV files include:

1. EEE3097S 2022 Walking Around Example Data.csv
2. EEE3097S 2022 Turntable Example Data.csv
3. EEE3097S 2022 Turntable Example Data 2.csv.

The primary tests will be used on “Turntable Example Data.csv” and the other two will be used to verify our results.

3.1.2. Data justification

The following justifications are based off the updated ATP’s found under Section 4.3:

A1	The CSV files can be read in C without errors being shown, thus this ATP can be tested fully.
A2	<i>Unapplicable as tests using the aforementioned data are not conducted on STM yet. Only on a computer. However, code that causes the STM LED to flash is sent to the STM to ensure that data can be sent and interpreted by the STM. Timing of this data would serve no purpose as this data is irrelevant to the scope of this project.</i>
A3	The CSV files can be read in C without errors being shown, thus this ATP can be tested fully.
A4	Contents (characters) of the CSV files can be compared to each other natively in C and hence this ATP can be tested fully.
A5	The compression algorithm will take time to run and hence this ATP can be fully tested.
A6	The encryption algorithm will take time to run and hence this ATP can be fully tested.
A7	<i>Unapplicable as tests for compression and encryption are not conducted on STM yet. Only on a computer. Thus, it is not possible to check if the output file is the same as the original.</i>

3.2. Experiment setup using simulated data

3.2.1. Simulations and experiments to determine overall functionality

The following experiments were performed to determine overall functionality:

- Experiment 1: To test the reading and writing to the STM, we wrote code to cause an LED on the STM to flash
- Experiment 2: In order to determine the overall functionality of the program, a file was encrypted then compressed and thereafter decrypted and decompressed. A test script was written in the C programming language to check the integrity of the CSV file after this process. It was compared to the original CSV file.
- Experiment 3: In order to show that data can be sent from the computer to the STM and back to the computer code was written to send a message from the computer to the STM and back to the computer via PuTTY.

3.2.2. Subsection experiments

3.2.2.1. Compression experiments

The following experiments were performed on TurntableData.csv to test if the GZIP algorithm works:

- Experiment 1: To test if the file was successfully being read by the algorithm, GZIP was set to run on the CSV file, if the file was successfully read then an output was produced in the same directory e.g., TurntableData.csv.gz. If the file was unsuccessful the output would be a response in terminal describing the error e.g., not in gzip format.
- Experiment 2: The timing mechanism built into Unix based systems was used to time the algorithm (time) when the gzip algorithm was called from terminal. E.g., time gzip TurntableData.csv.
- Experiment 3: A test script was written in the C programming language to check the integrity of the CSV file after extraction. It was compared to the CSV file before compression.

3.2.2.2. Encryption experiments

The following experiments were performed on TurntableData.csv to test if the AES algorithm works:

- Experiment 1: To test if the file was successfully being read by the algorithm, a test was written to display an error if the CSV file was NULL in C and to display the plaintext that will be encrypted in terminal to further prove that the file is being read.
- Experiment 2: The timing mechanism built into Unix based systems was used to time the algorithm (time) when the AES algorithm was called from terminal. E.g., time gcc -o aes aes.c main.c.
- Experiment 3: A test script was written in the C programming language to check the integrity of the CSV file after decryption. It was compared to the CSV file before encryption.
- Experiment 4: The same test script as experiment 3 will be run on the encrypted and original file to show that there are differences in the file and it is sufficiently encrypted.

3.2.3. Data types

After compression the output file should be of type: .csv.gz.

After extraction the file will be the original file before compression and will be of type: csv.

The file type after encryption as well as the file type after decryption are .csv.

3.3.Results

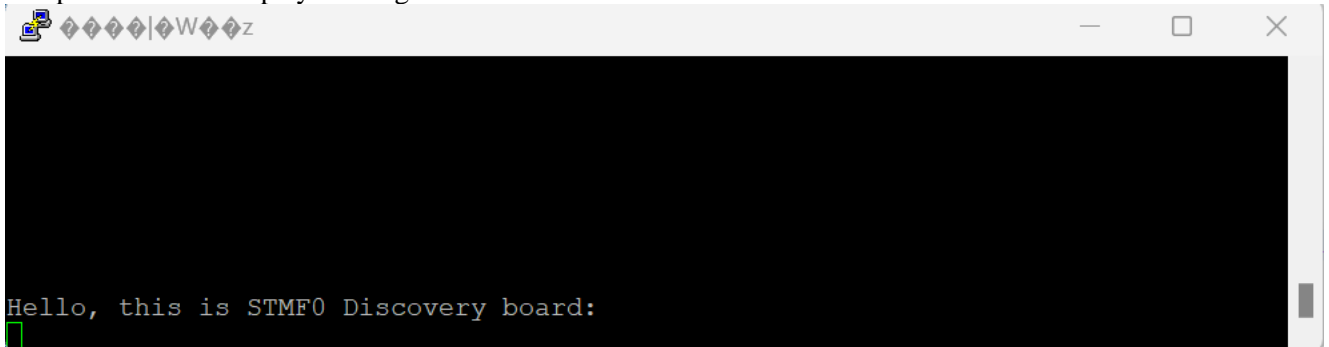
3.3.1. Overall functionality

- The first experiment showed the correct LED flashing on the STM board. This shows that data and instruction can be successfully written to the STM.
- The results from the second test yielded that there were no errors or mismatches between the files before encryption and compression and after extraction and decryption. The terminal output displayed: “Errors: 0”. This means that the compression algorithm is lossless and that no data was lost during the encryption and decryption process



```
PROBLEMS 4 DEBUG CONSOLE TERMINAL JUPYTER OUTPUT
pooja@DESKTOP-J7MFTR6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$ gcc integrityChecker.c -o test
pooja@DESKTOP-J7MFTR6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$ ./test
Total Errors : 0
pooja@DESKTOP-J7MFTR6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$
```

- This experiment showed that the data that was sent from the computer to the STM was sent back to the computer and was displayed using PuTTY



```

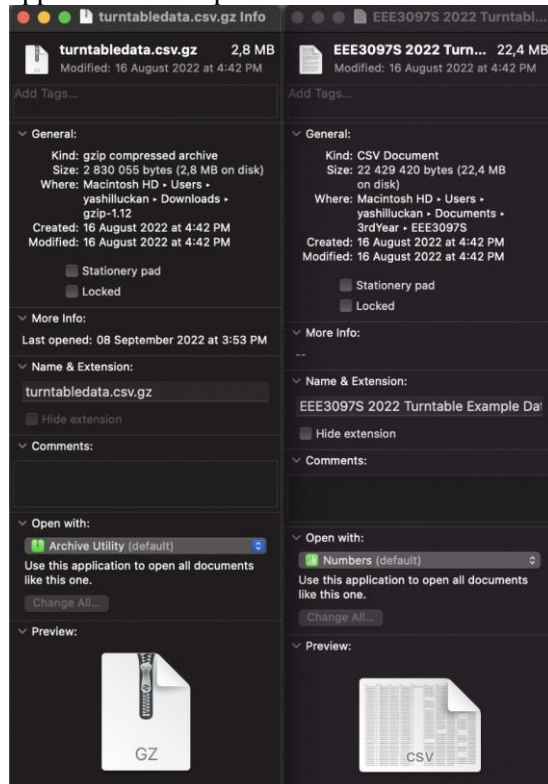
Hello, this is STMF0 Discovery board:

```

3.3.2. Subsection functionality

3.3.2.1. Compression functionality

- The results from test one shows the CSV file was able to be successfully compressed as no error messages such as “not in gzip format” were thrown to the terminal. Instead, the compressed file was delivered to the same directory as the original file and had the following name: TurntableData.csv.gz. The .gz extension signifies that the file was successfully compressed. The input file size was: 22 429 420 bytes. The output file size was: 2 830 055 bytes. This gives an approximate compression ratio of 0.126 and is 7.925 times smaller than the original file

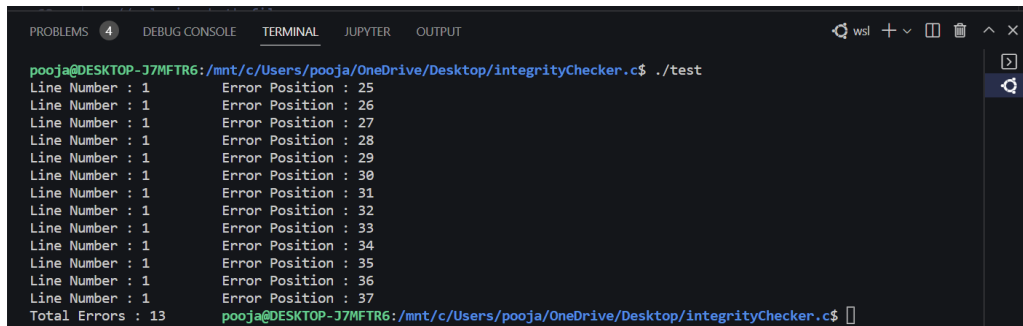


- The results from the second test yield that GZIP compression on TurntableData.csv took a real-world clock time of 0.455 seconds. The CPU clock cycles 0.011 seconds. Extraction took 0.051 seconds of real-world clock time and 0.011 seconds of CPU cycle time. The above values are an average after repeating the second experiment 2 times.
- The results from the third test yielded that there were no errors or mismatches between the files before compression and after extraction. The terminal output displayed: “Errors: 0”.



This means that the compression algorithm is lossless. Errors were intentionally introduced by deleting values in order to ensure the IntegrityChecker.c file works correctly and the following

output was produced after removing the headings of each value in the CSV file:



```
pooja@DESKTOP-J7MFT6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$ ./test
Line Number : 1      Error Position : 25
Line Number : 1      Error Position : 26
Line Number : 1      Error Position : 27
Line Number : 1      Error Position : 28
Line Number : 1      Error Position : 29
Line Number : 1      Error Position : 30
Line Number : 1      Error Position : 31
Line Number : 1      Error Position : 32
Line Number : 1      Error Position : 33
Line Number : 1      Error Position : 34
Line Number : 1      Error Position : 35
Line Number : 1      Error Position : 36
Line Number : 1      Error Position : 37
Total Errors : 13    pooja@DESKTOP-J7MFT6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$
```

3.3.2.2. Encryption functionality

- There was no error message displayed in terminal when the “TurntableData.csv” was run through the testing algorithm. The plain text displayed in terminal matched the text in the CSV file which showed that the file was being read accurately.
- The timing algorithm was run on for only encryption of the “TurntableData.csv” file. This experiment was repeated three times and the real time average was 0m0.082 which is sufficient. The timing algorithm was run another three time for both encryption and decryption of the file which took an average 0m0.103s in real time. This shows that the algorithm runs in a reasonable amount of time and that encryption alone is quicker than both encryption and decryption.
- The results from the third test yielded that there were no errors or mismatches between the files before encryption and after decryption. The terminal output displayed: “Errors: 0”.

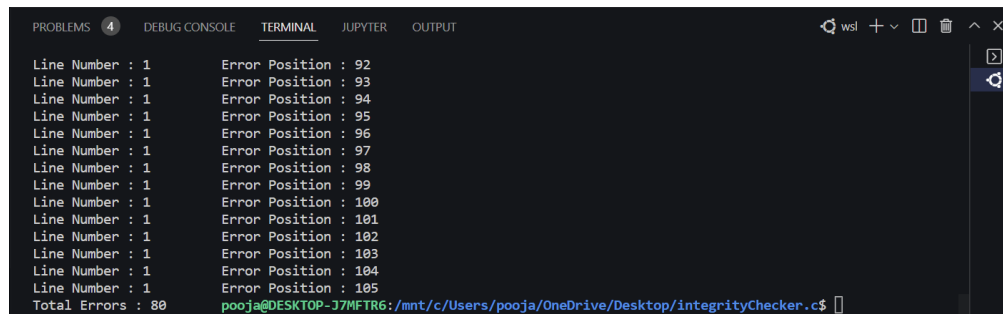


```

PROBLEMS 4 DEBUG CONSOLE TERMINAL JUPYTER OUTPUT
pooja@DESKTOP-37MFTR6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$ gcc integrityChecker.c -o test
pooja@DESKTOP-37MFTR6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$ ./test
Total Errors : 0
pooja@DESKTOP-37MFTR6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$

```

This shows that no data was lost during the encryption and decryption process. Errors were intentionally introduced by deleting a row in the CSV file in order to ensure the IntegrityChecker.c file works correctly and the following output was produced after removing the headings of each value in the CSV file:



```

PROBLEMS 4 DEBUG CONSOLE TERMINAL JUPYTER OUTPUT
Line Number : 1      Error Position : 92
Line Number : 1      Error Position : 93
Line Number : 1      Error Position : 94
Line Number : 1      Error Position : 95
Line Number : 1      Error Position : 96
Line Number : 1      Error Position : 97
Line Number : 1      Error Position : 98
Line Number : 1      Error Position : 99
Line Number : 1      Error Position : 100
Line Number : 1      Error Position : 101
Line Number : 1      Error Position : 102
Line Number : 1      Error Position : 103
Line Number : 1      Error Position : 104
Line Number : 1      Error Position : 105
Total Errors : 80
pooja@DESKTOP-37MFTR6:/mnt/c/Users/pooja/OneDrive/Desktop/integrityChecker.c$

```

- The results from the final test showed a 360504 Errors which indicates that little to no values in the encrypted and original file matched. This proves the validity of the encryption algorithm.

3.3.3. Additional Testing

When all above tests were run with Walking Around Example Data.csv and Turntable Example Data 2.csv, identical functionality was observed and it was clear that the algorithms were both effective.

4. Validation Using the IMU

4.1. IMU Module

Two IMU modules were received, one from WaveShare and one from Sparkfun. The following information is based of the Sparkfun IMU.

4.1.1. IMU analysis (ICM-20948 & ICM-20649)

The ICM-20948 will be used for testing received from Sparkfun will be used for testing as it has similar properties to the ICM-20649 which will be used on the actual buoy. The ICM-20948 is the lowest power 9-Axis Device at 2.5mW which works well with the application due to the limited power source. The features of the ICM-20948 include a 3-Axis Gyroscope with Programmable FSR of ± 250 dps, ± 500 dps, ± 1000 dps, and ± 2000 dps which differs from the ICM-20649 which also has a 3-Axis Gyroscope but with programmable FSR of ± 500 dps, ± 100 dps, ± 2000 dps, and ± 4000 dps, however, there is still overlapping dpi which means that data can be accurately validated across both IMU's. Both IMU's have 3-Axis Accelerometers with Programmable FSR of $\pm 4g$, $\pm 8g$ and $\pm 16g$ but the ICM-20948 also supports $\pm 2g$ whilst the ICM-20649 also supports $\pm 30g$. The difference in programable FSR with regards to the accelerometer and gyroscope reading should not impact testing as results of similar lengths will still be obtained on both IMU's. Both IMU's have on-chip 16-bit ADCs and programmable filters, a 7 MHz SPI or a 400 kHz Fast Mode I²C, a digital-output temperature sensor, a VDD operating range of 1.71V to 3.6V which is within the range supplied by the STM. SPI will be used to communicate with the IMU so that the maximum transmission rate can be used if need be.

4.1.2. Steps to ensure accuracy between multiple IMU's

The first step when performing tests is to ensure that the IMU that we have is (if possible) configured to the same settings as the IMU used on the buoy. The tests in section 2.3 are used to validate the readings from the IMU. Where possible, the tests were conducted in the same environment with as little environmental noise as possible. We also tried to simulate the environment that the IMU would be in, in our application. All of the above was done so that the data collected by the ICM-20948 can be extrapolated to the ICM-20649.

4.1.3. The need for IMU validation

It is necessary to validate the real IMU data for many reasons. It is done to ensure that the data is accurate and that the IMU does indeed show results that represent reality. It is to ensure that that this unit can actually be implemented on the real buoy. If results are inaccurate on the buoy, they could be interpreted to the person viewing the data as accurate and give false results leading to an compromised study. It is also very difficult to validate the data once it is deployed on the buoy. Thus, it is vital to validate as much as we can now.

4.1.4. IMU validation tests

The following tests were performed to ensure the IMU is behaving appropriately.

- Test 1: To ensure the gyroscope is behaving as expected the IMU was rotated upon each axis, namely the x, y, and z-axis by a premeasured 10 degrees and the data displayed to the PuTTY terminal was confirmed to reflect this.
- Test 2: To ensure the accelerometer is behaving as expected the IMU placed in motor vehicle accelerating from 0-30km/h in 2 seconds. This acceleration numerically evaluates to 0.4249g. This was repeated for all three axes and was achieved by just rotating the IMU. The IMU readings were compared against the calculated value to confirm the accelerometer is in working order.
- Test 3: For completeness' sake, the magnetometer was also briefly tested by increasing and decreasing the distance of a magnet with respect to the IMU. This test was done to ensure all three sensors were in working order regardless of the fact that we don't require the magnetometer for our application.
- Test 4: In order to simulate the environment that the buoy will be deployed into, the IMU was moved around inside a freezer and the accelerometer and gyroscope measurements were compared to those taken at room temperature.

4.2. Experiment setup using IMU data

4.2.1. Simulations and experiments to determine overall functionality

The following experiments were performed to determine overall functionality:

- Experiment 1: To test the reading and writing to the STM, we wrote code to cause an LED on the STM to flash.
- Experiment 2: In order to determine the overall functionality of the program, a data log file was encrypted then compressed and thereafter decrypted and decompressed. A test script was written in the C programming language to check the integrity of the data log file after this process. It was compared to the original data log file.
- Experiment 3: To test if data can be captured from the IMU via the STM32F0, the data is from the IMU is sent to the STM via an SPI connection and will be displayed through PuTTY on a computer. This experiment consisted of the three tests mentioned in section two of this report.
- Experiment 4: To test the time taken to read from the IMU a single shot of data will be captured and timed via a C timing mechanism.

4.2.2. Subsection experiments

4.2.2.1. Compression experiments

The following experiments were performed on a log file produced by the PuTTY output when data is captured from the IMU to test if the GZIP algorithm works:

- Experiment 1: To test if the file was successfully being read by the algorithm, GZIP was set to run on the data log file, if the file was successfully read then an output was produced in the same directory e.g., PuTTY_log_2022.10.01.txt.gz. If the file was unsuccessful the output would be a response in terminal describing the error e.g., not in gzip format.
- Experiment 2: The timing mechanism built into Unix based systems was used to time the algorithm (time) when the gzip algorithm was called from terminal. E.g., time gzip PuTTY_log_2022.10.01.txt.
- Experiment 3: A test script was written in the C programming language to check the integrity of the data log file after extraction. It was compared to the data log file before compression.

4.2.2.2. Encryption experiments

The following experiments were performed if the AES algorithm works both on a computer and the STM using data captured from the IMU:

- Experiment 1: To test if the file was successfully being read by the algorithm, a test was written to display an error if the PuTTY log file containing just the IMU data was NULL in C and to display the plaintext that will be encrypted in terminal to further prove that the file is being read
- Experiment 2: The timing mechanism built into Unix based systems was used to time the algorithm (time) when the AES algorithm was called from terminal. E.g., time gcc -o aes aes.c main.c
- Experiment 3: A test script was written in the C programming language to check the integrity of the PuTTY log file after decryption. It was compared to the PuTTY log file before encryption.
- Experiment 4: The same test script as experiment 3 will be run on the encrypted and original file to show that there are differences in the file and it is sufficiently encrypted.
- Experiment 5: With both the encryption and decryption algorithms loaded onto the STM, output will be produced in PuTTY to show that the decrypted text is the same as the data captured by the IMU. This can be observed as the data is encrypted line by line.
- Experiment 6: Multiple lines of encrypted text produced on the STM will be displayed on PuTTY and the log file will be saved as a textfile.

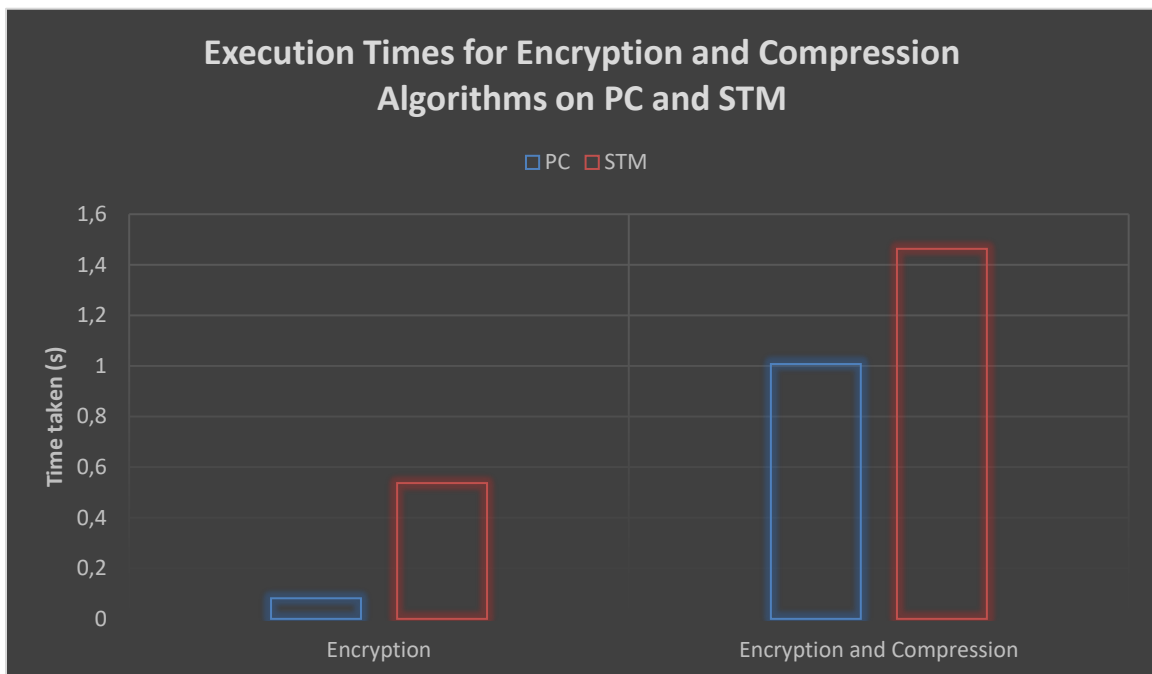
4.2.3. Data types

When data collection from the IMU is initiated in CubeIDE, data is sent as raw bits to the STM via a SPI connection. The STM then forwards these raw bits of data to the computer via a UART interface. This data is encrypted and displayed via PuTTY and saved as a .txt file. Once the data is in the .txt file, and then compressed into a .txt.gz file.

4.3.Results

4.3.1. Overall functionality

- The first experiment showed the correct LED flashing on the STM board. This shows that data and instructions can be successfully written to the STM.
- The results from the second test yielded that there were no errors or mismatches between the files before encryption and compression and after extraction and decryption. The terminal output displayed: “Errors: 0”. This means that the compression algorithm is confirmed to be lossless and that no data was lost during the encryption and decryption process.
- The third experiment showed that the data was outputted to PuTTY successfully. Not only was the data being displayed but the data was also accurate with the exception of experimental uncertainty in the three tests from section two and environmental effects such as Gaussian White Noise.
- The timing test yielded a time of 3.01 seconds.



4.3.2. Subsection functionality

4.3.2.1. Compression functionality

- The results from test one shows the data log file was able to be successfully compressed as no error messages such as “not in gzip format” were thrown to the terminal. Instead, the compressed file was delivered to the same directory as the original file and had the following name: PuTTY_log_2022.10.01.txt.gz. The .gz extension signifies that the file was successfully compressed. The input file size was: 479 bytes. The output file size was: 285 bytes which is 1.7 times smaller than the original file.
- The results from the second test yield that GZIP compression on the PuTTY data log took a real-world clock time of 0.008 seconds. Extraction process took 0.005 seconds. These values are an average after repeating the second experiment 2 times.
- The results from the third test yielded that there were no errors or mismatches between the files before compression and after extraction. The terminal output displayed: “Errors: 0”. This means that the compression algorithm is lossless. Errors were intentionally introduced by deleting values in order to ensure the IntegrityChecker.c file works correctly and the following output was produced after removing the headings of each value in the CSV file:

Total Errors: 160

Line Number: 1	Error Position: 1
Line Number: 1	Error Position: 2
Line Number: 1	Error Position: 3
.	.
.	.
.	.
Line Number: 1	Error Position: 160

4.3.2.2. Encryption functionality

- There was no error message displayed in terminal when the “PuTTY.log.txt” was run through the testing algorithm. The plain text displayed in terminal matched the text in the CSV file which showed that the file was being read accurately.
- The timing algorithm was run on for only encryption of the “PuTTY.log.txt” file. This experiment was repeated three times and the real time average was 0m0.082 which is sufficient. The timing algorithm was run another three time for both encryption and decryption of the file which took an average 0m0.103s in real time. This shows that the algorithm runs in a reasonable amount of time and that encryption alone is quicker than both encryption and decryption.
- The results from the third test yielded that there were no errors or mismatches between the files before encryption and after decryption. The terminal output displayed: “Errors: 0”. This shows that no data was lost during the encryption and decryption process.
- The results from the final test showed a 360504 Errors which indicates that little to no values in the encrypted and original file matched. This proves the validity of the encryption algorithm.
- When the encryption and decryption algorithms are run on the STM, the output captured from the IMU matches exactly the decrypted data and is completely different from the encrypted data which shows that the data was correctly encrypted and decrypted on the STM.

```
COM4 - PuTTY
Gyroscope: X:0.243902 Y:0.731707 Z:-0.609756
Plain Text: -0.0058590.0063480.9702150.1829270.000000-0.426829
Encrypted Text: eGq] =
Decrypted Text: -0.0058590.0063480.9702150.1829270.000000-0.426829

Acceleration: X:-0.005859 Y:0.006348 Z:0.970215
Gyroscope: X:0.182927 Y:0.000000 Z:-0.426829
Plain Text: 0.0024410.0146480.9643550.914634-0.304878-2.560976
Encrypted Text: 0-lliq
hN]g]hCa]l\&]3
Decrypted Text: 0.0024410.0146480.9643550.914634-0.304878-2.560976

Acceleration: X:0.002441 Y:0.014648 Z:0.964355
Gyroscope: X:0.914634 Y:-0.304878 Z:-2.560976
Plain Text: 0.0043950.0126950.9702150.487805-0.243902-1.951220
Encrypted Text: CQxS0I
p0C]HDl,]Pi]ny[+]
Decrypted Text: 0.0043950.0126950.9702150.487805-0.243902-1.951220

Acceleration: X:0.004395 Y:0.012695 Z:0.970215
Gyroscope: X:0.487805 Y:-0.243902 Z:-1.951220
```

- Multiple lines of encrypted text produced on the STM were displayed on PuTTY and the log file will be saved as a textfile which was then compressed.

4.3.3. Changes in performance

We noticed some small differences when changing the sampling rate of reading data from the IMU. The only differences include the log file gets larger in less time when sampling at a higher rate. At higher sampling rate, more data is being collected by the IMU per unit time, thus making the IMU more sensitive to changes in the environment. For example, if the IMU was set to sample at 1Hz, and the IMU was moved many times in one second, this movement would not be logged. While higher sampling rates led to more sensitivity, this isn't always a good thing as it also becomes more susceptible to undesirable factors such as Gaussian white noise. This is due to the fact that if more data is sampled from a noisy or undesirable environment, more noise is present in the results. These factors, however, are independent of the compression and encryption processes.

5. Acceptance test procedure

5.1. Figures of merit

5.1.1. Compression algorithm (GZIP) figures of merit

- GZIP compression algorithm is the fastest executing compression algorithm out of the algorithms in the comparison. This means that it will use the least power.
- GZIP uses the least memory out of the compared compression algorithms. This is useful as we only have 8KB of RAM on the STM.
- GZIP algorithm is lossless which is important for text-based applications such as this application where data is stored in CSV files.
- GZIP can be implemented via multiple programming languages making it versatile and useful if a programming language needs to be changed in order to comply with storage constraints.
- GZIP libraries are relatively which is useful as storage on STM is highly limited.

5.1.2. Encryption algorithm (AES) figures of merit

- AES encryption algorithm was chosen as it is symmetric encryption algorithm and symmetric encryption algorithms are faster and require less computational power.
- AES is the fastest of the compared symmetric encryption algorithms.
- AES is very secure encryption algorithm.
- AES is compatible with a range of different platforms from 8-bit microcontrollers to 64-bit processors to FPGAs which makes it ideal for this implantation.[6]
- AES uses the least memory of all compared algorithms, this it is ideal for this application.
- AES can be implemented using multiple different programming languages including C, C# and Python.

5.2. Experiment design

In order to test the compression algorithm, the raw CSV file will be compressed using the GZIP algorithm. The file size will be noted prior to compression and post-compression. The compressed data file will then be extracted. Firstly, the compression and extraction will be tested on a computer and then compression will be tested on the STM and extraction will be tested on a computer. Once the compression algorithm is working, the compressed file will then be encrypted on the STM and then decrypted on a computer. The encryption and decryption will be tested by comparing the original file to the file after decryption. This will be tested on a computer to ensure the algorithm is working prior to loading it onto the STM.

5.3. Acceptance performance definition

ATPs:	
A1	The GZIP algorithms has built in file checking functionality. If the file contents are read successfully before the compression algorithm is run then an output file of format: .txt.gz is outputted into the same directory. If the file was unsuccessfully read or any other reason to that led to the file not being compressed, then an output to the system out will say "File not in gzip format".
A2	A timing mechanism in STMCubeIDE will be used to calculate the initialisation, read and write time of the IMU to the STM and for the necessary files to be downloaded to the STM. A pass will be granted if the time taken to write to the STM and receive the first reading is completed in under 20 seconds and a failure otherwise.
A3	An if-statement will be used to test if data was read in from the file by testing if the file was empty within a testing algorithm written in C. If the file contents are read before the encryption algorithm is run successfully then a "success" message will be displayed which will qualify as a pass. Else, a "Error" message will appear if the file was unsuccessfully read which will qualify as a fail.
A4	A test script will be written and will be run to check if the output file after extraction and decryption is the same as the raw data file. The test script will be written in C and will include an algorithm that compares every character of the output and input file and verifies that it is the same.
A5	A timing mechanism in Unix will be used to calculate the runtime of the compression algorithm. The mechanism will encapsulate the entire algorithm. A pass will be indicated by a time less than 1.1s and a fail will be indicated as a time greater than 1.1s.
A6	A timing mechanism in C will be used to calculate the runtime of the encryption algorithm. The mechanism will encapsulate the entire algorithm. A pass will be indicated by a time less than 1s and a fail will be indicated as a time greater than 1s.
A7	The encryption algorithm was set to encrypt every line of data received by the IMU to show that the encryption works.

5.4. ATP Performance

ATPs:	
A1	When the compression algorithm was run to compress the data file produced by PuTTY, no errors were thrown by the built-in file checker and thus this ATP is being met.
A2	The time taken to download the data to the STM and receive the first data reading from the IMU (includes initialisation and read/write times) was 3.01 seconds which is under the 20 seconds threshold and thus this ATP is a pass.
A3	When the encryption algorithm was run, no errors were observed, and the original text was displayed to be the same as the text entered and thus this ATP is being met.
A4	The file before encryption and compression was exactly the same as the file after decryption and extraction and thus this ATP is met as integrity is fully preserved.
A5	The time taken for the whole compression algorithm to run was less than 1s and thus this ATP is met.
A6	The time taken for the whole encryption algorithm to run was less than 1s and thus this ATP is met.

5.5.ATP Progress

ATP	Milestone 1	Milestone 2	Milestone 3	Milestone 4
A1	-	Pass	Pass	Pass
A2	-	-	Pass	Pass
A3	-	Pass	Pass	Pass
A4	-	Pass	Pass	Pass
A5	-	Pass	Pass	Pass
A6	-	Pass	Pass	Pass
A7	-	-	Pass	Pass

5.6.Traceability Matrix

UR	FR	Specifications	ATP	Pass/Fail
U1	U1.F1	U1.F1.S1	U1.F1.S1.A1	Pass
	U1.F2	U1.F2.S2	U1.F2.S2.A2	Pass
	U1.F3	U1.F3.S3	U1.F3.S3.A3	Pass
	U1.F4	U1.F4.S4	U1.F4.S4.A2	Pass
U2	U2.F5	U2.F5.S5	U2.F5.S5.A4	Pass
	U2.F6	U2.F6.S6	U2.F6.S6.A4	Pass
	U2.F7	U2.F7.S7	U2.F7.S7.A5	Pass
	U2.F8	U2.F8.S8	U2.F8.S8.A5	Pass
U3	U3.F9	U3.F9.S9	U3.F9.S9.A6	Pass
	U3.F10	U3.F10.S10	U3.F10.S10.A6	Pass
	U3.F11	U3.F11.S11	U3.F11.S11.A6	Pass
	U3.F12	U3.F12.S12	U3.F12.S12.A6	Pass
U4	U4.F13	U4.F13.S13	U4.F13.S13	Pass
	U4.F14	U4.F14.S14	U4.F14.S14	Pass
	U4.F15	U4.F15.S15	U4.F15.S15	Pass
	U4.F16	U4.F16.S16	U4.F16.S16	Pass

6. Consolidation of ATP's and Future Plan

In summary, all ATPs were satisfied. Due to all ATP's being satisfied, the overall system passes and is able to be used on the SHARC buoy however, it should be noted that a Raspberry Pi is necessary for complete implementation as the GZIP compression algorithm including its libraries are too large to fit on the 64KB storage on the STM.

7. Conclusion

In conclusion, the IMU data was successfully read and encrypted line-by-line on the STM and then sent to the computer to be compressed. The chosen compression algorithm successfully compressed and extracted the data file without any loss of data, the same can be said for the encryption algorithm. The encryption algorithm completed encryption successfully within reasonable time and the compression algorithm completed within reasonable time as well which proved these algorithms as viable for this application. To further prove viability and accuracy, the IntegrityChecker.c file was used for the data file on which encryption and compression was performed. The data from the IMU was proved to be accurate via multiple tests which allow one to extrapolate data from the ICM-20948 to the ICM-20649.

8. References

- [1] T. Brookes, "Lossy vs. Lossless Compression: What's the Difference?", *Howtogeek.com*, 2022. [Online]. Available: <https://www.howtogeek.com/744381/lossy-vs-lossless-compression-whats-the-difference/>. [Accessed: 12- Aug- 2022].
- [2] "Comparison of Compression Algorithms", *LinuxReviews*, 2022. [Online]. Available: https://linuxreviews.org/Comparison_of_Compression_Algorithms. [Accessed: 16- Aug- 2022].
- [3] "What types of encryption are there?", *Ico.org.uk*, 2022. [Online]. Available: <https://ico.org.uk/for-organisations/guide-to-data-protection/guide-to-the-general-data-protection-regulation-gdpr/encryption/what-types-of-encryption-are-there/>. [Accessed: 18- Aug- 2022].
- [4] J. Thakkar, "Types of Encryption: 5 Encryption Algorithms & How to Choose the Right One", *Hashed Out by The SSL Store™*, 2022. [Online]. Available: <https://www.thesslstore.com/blog/types-of-encryption-encryption-algorithms-how-to-choose-the-right-one/>. [Accessed: 18- Aug- 2022].
- [5] A. Bhattacharya, "Comparison of Various Encryption Algorithms and Techniques for Securing Data | Encryption Consulting", *Encryption Consulting*, 2022. [Online]. Available: <https://www.encryptionconsulting.com/comparison-of-various-encryption-algorithms-and-techniques-for-securing-data/>. [Accessed: 18- Aug- 2022].
- [6] M. Wahid, A. Ali, B. Esparham and M. Marwan, "A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention", *Symbiosisonlinepublishing.com*, 2022. [Online]. Available: <https://symbiosisonlinepublishing.com/computer-science-technology/computerscience-information-technology32.php>. [Accessed: 18- Aug- 2022].
- [7] *Eprint.iacr.org*, 2022. [Online]. Available: <https://eprint.iacr.org/2009/501.pdf>. [Accessed: 18- Aug- 2022].
- [8] *Research.cs.wisc.edu*, 2022. [Online]. Available: <https://research.cs.wisc.edu/wpis/examples/pcca/gzip/gzip.c>. [Accessed: 12- Sep- 2022].
- [9] "GitHub - openluopworld/aes_128: Implementation of AES-128 in pure C. No modes are given. Only one block of encryption and decryption is given here.", *GitHub*, 2022. [Online]. Available: https://github.com/openluopworld/aes_128. [Accessed: 11- Sep- 2022].
- [10] S. (Qwiic), W. Well, G. IMU and A. Worthless, "SparkFun 9DoF IMU Breakout - ICM-20948 (Qwiic) - SEN-15335 - SparkFun Electronics", *Sparkfun.com*, 2022. [Online]. Available: <https://www.sparkfun.com/products/15335>. [Accessed: 03- Oct- 2022].