EEE3097S Final Report

Group 3
Shameera Cassim and Tristyn Ferreiro
CSSSHA020, FRRTRI001

17 October 2022

Admin Documentation

Contributions

Task	Section Numbers	Completed by
Admin: Collating all the admin documentation, organising gitHub and keeping PM tool up-to-date	-	Both
Introduction	1	Both
Requirement Analysis	2	Both
Subsystem Design: Encryption	3.1	Shameera Cassim
Subsystem Design: Compression	3.2	Tristyn Ferreiro
Subsystem Design: Inter-Subsystem and Inter-Sub-subsystems Interactions	3.3	Both
Subsystem Design: Bottlenecks	3.4	Both
Validation(Simulation): Necessity of Simulation-Based Validation	4.1	Tristyn Ferreiro
Validation(Simulation): Other	4.2, 4.3	Both
Validation(IMU): Necessity of Hardware-based validation Validation(IMU): Other	5.1 5.2, 5.3	Shameera Cassim Both
Experiment Setup(Simulation and IMU): Joint Algorithms	4.4.1, 4.4.2, 5.4.1, 5.4.2,5.4.3	Both
Experiment Setup(Simulation and IMU): Compression	4.4.3, 4.4.4, 5.4.10	Tristyn Ferreiro
Experiment Setup(Simulation and IMU): Encryption	4.4.5, 5.5.11	Shameera Cassim
Experiment Setup(Simulation and IMU): Overall	4.4.1, 4.4.2, 5.4.1, 5.4.2	Both
Experiment Setup: IMU	5.4.4-5.4.9	Both
Results(Simulation and IMU): Joint Algorithms	4.5.1, 4.5.2, 5.5.1, 5.5.2, 5.5.3	Both
Results: IMU	5.5.4-5.5.9	Both
Results(Simulation and IMU): Compression	4.5.3, 4.5.4, 5.5.10	Tristyn Ferreiro
Results(Simulation and IMU): Encryption	4.5.5, 5.5.11	Shameera Cassim
Acceptance Test Procedures	6	Both
Future Plans	7	Both
Conclusion	8	Both
Appendix A	9	Both

Project Timeline

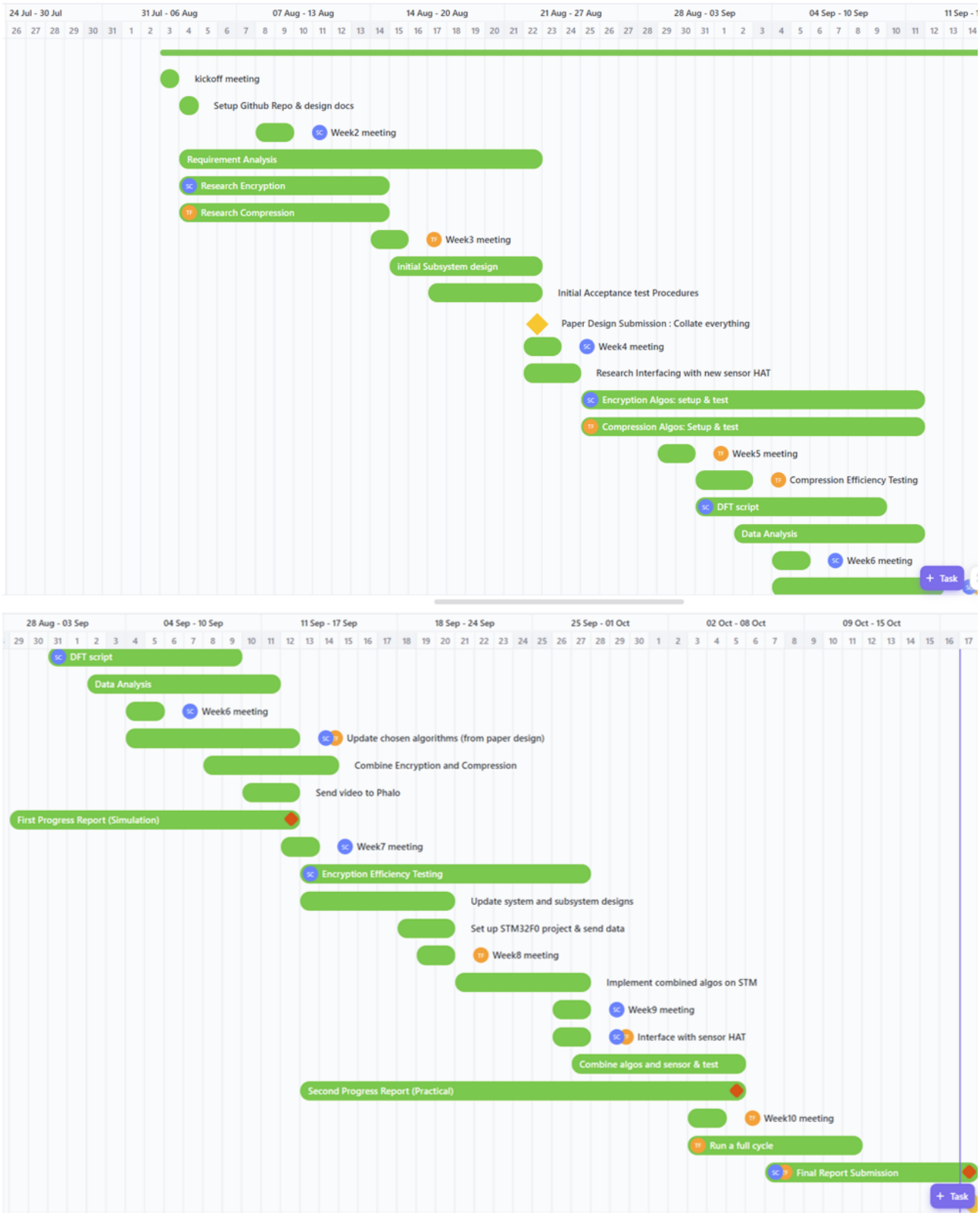

Figure 1: Overview of the timeline for the entire project

There were several delays throughout the project. Namely:

1. Interfacing with the sensor was done late into the project due to a delay in the sensors arriving. This resulted in a high stress period in the project as interfacing and complete system integration needed to be done in a short span of time.

2. Between weeks 4 and 6 there were many of tests and assignments for other courses which demanded our attention. This resulted in us carrying the work over from week 4/5 into week 5/6.

3. There was delay in getting the combined algorithms onto the STM32. This was a result of completely changing the system design which then required more testing before integrating with the STM. This delayed progress by a few days and added to the stress mentioned in 1.

4. Due to the delays mentioned in 1 and 3, there was a delay in interfacing with the sensor. This forced us to get the complete system integration done in a week which lead to some bugs that needed to be fixed before the final demo.

5. The progress report 2 and final report submissions were delayed due to high workloads in other courses taken.

Project Management tool

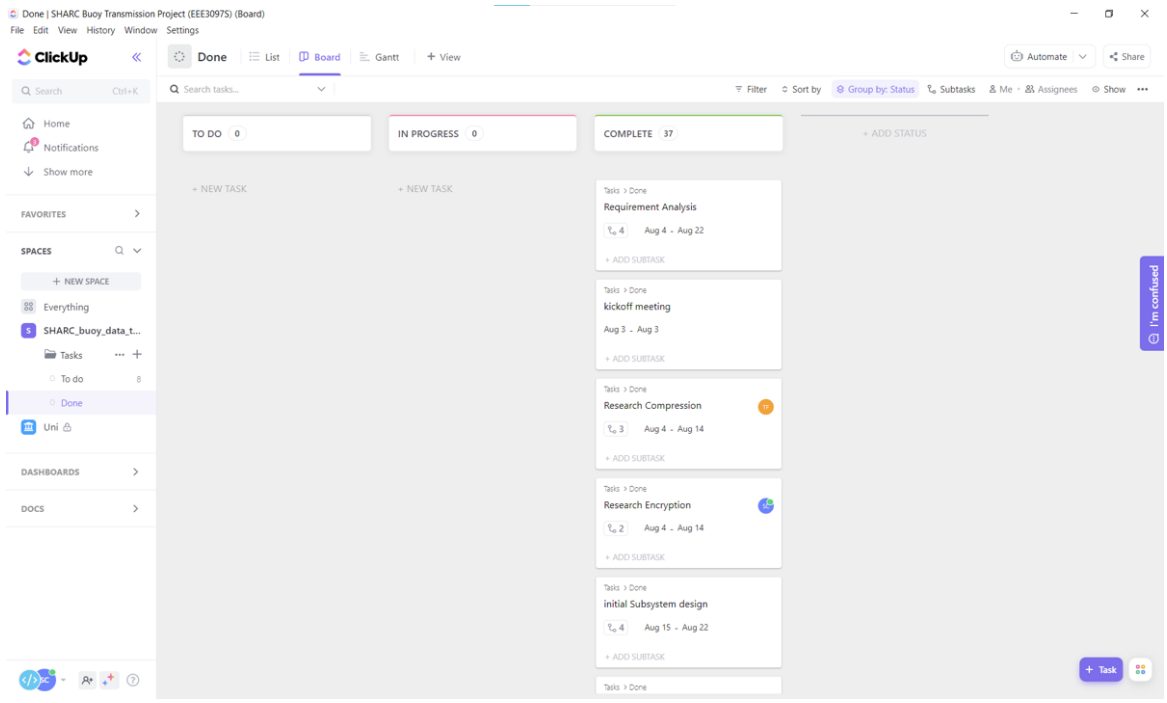

Figure 2: Screenshot of the project management tool showing the remainder of our tasks

The list above shows some of the milestones and suggested tasks for the project from the ClickUp task list. As we have reached the end of this project, all tasks are completed.

Git

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

Contents

List of Figures

List of Tables

Listings

1 Introduction

1.1 Problem Statement

One of the SCALE projects which UCT is involved in is the SHARC buoy Antarctic ice project. The focus of the ice project is to improve the understanding of Southern Ocean sea ice dynamics in the Antarctic Marginal Ice Zone (MIZ). Furthermore, the project aims to solve the reliability issues associated with remote sensing by installing the buoys on pancake ice. Access to more data from the MIZ would improve understanding of global climate drivers which is why the project was initiated [2].

This project aims to design a compression and encryption system for use onboard the SHARC buoy [2] SCALE project. A compression solution is required to resolve the data size problems highlighted in the SHARC Buoy design by Jamie Nicholas Jacobson [2]. The goal is to reduce the costs of data transmission by reducing the size of the data to be sent from the SHARC buoy over Iridium. Additionally, encryption is added to further secure the data being transmitted. For the completion of this project, the resulting compression and encryption system will run on the STM32F0Discovery board but be implementable on the SHARC buoy. The decompression and decryption will be run on a computer and the data will be transmitted over a serial connection, not Iridium. Conclusion of this project will result in a fully functional encryption and compression system that can be deployed on the SHARC buoy.

1.2 Scope

This project deals with implementation of encryption and compression of data from the IMU sensor on the SHARC buoy system [2]. This project considers different compression and encryption algorithms to apply to the data. Organised storage of the data and satellite transmission fall outside of the scope of this project. This project aims to provide a solution to the data size problems highlighted in the SHARC Buoy design by Jamie Nicholas Jacobson [2].

System testing is carried out on a STM32F0Discovery board not the SHARC buoy device. Testing is conducted at both a subsystem and full system level. Data testing makes use of live IMU sensor data and simulation data to validate the system at different phases.

1.3 Limitations

The system must be designed to use data from the ICM-20649 IMU however, at the time of this project, it is not available. Instead the ICM-20948 is used for the duration of this project and design decisions take ICM-20649 compatibility into account.

Additionally, at the time of this project, Raspberry Pis (floating point processors) are not available for use in development and testing. Instead an STM32F0Discovery with a fixed point processor is used which limits functional testing of the algorithms on a floating point processor architecture. Further, this limits the choice of algorithms.

Furthermore, the STM32F0Discovery has limited RAM and flash memory. This further limits the choice of algorithms as source code size must be minimal. The amount of data used to test the algorithms on onboard the discovery board is limited.

1.4 Assumptions

The deployed device will have sufficient storage to accommodate the system's source code. The device will have sufficient power to run both algorithms (particularly compression). The processor is 32-bit and ARM-based and can use fixed point architecture. The processor is compatible with a Serial Peripheral Interface (SPI) and able to use Hardware Abstract Layer(HAL) driver files.
Furthermore, the system is considered complete if it can complete a full data measurement cycle, send the data through encryption and compression and transmit without external assistance. Most of the data should be recovered using the decompression and decryption.

2 Requirement Analysis

This section identifies and outlines the project user requirements. From these user requirements, specifications and functional system requirements are developed. A feasibility analysis is performed on the derived specifications.

Table 1: Requirement Traceability Matrix

UR	FR	Specifications	ATP
UR01	FR01	SP01	AT02
UR02	FR06	SP06	AT09
UR03	FR05	SP05	
UR04	FR03, FR04	SP03, SP04	AT04, AT05
UR05	FR01, FR02	SP02	AT03
UR06	FR07	SP07	AT01

2.1 User Requirements

Table 2: 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 ICM-20948 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

2.1.1 Analysis of UR01

System must minimise data loss.

If excessive data is lost, it will not be useful for research. Thus, while some losses due to compression are acceptable, the data recovered should contain enough of the original data such that it is still usable. Important parameters including the mean wave direction, directional width, skewness and kurtosis parameters of the wave model can be calculated using the Fourier coefficients [2]. Thus, Fourier coefficients should be the focus of data preservation.

2.1.2 Analysis of UR02

System must be compatible with ICM-20649 and ICM-20948 IMU chips.

The final system should work with an ICM-20649 IMU chip. However, at the time of this project, the ICM-20649 is not available and so the ICM-20948 IMU chip will be used for testing and the duration of this project. The solution should still be relevant for the ICM-20649 and its data.

2.1.3 Analysis of UR03

System must use minimal computation to be power efficient.

The SHARC buoy system has limited access to power and thus, the implementation of encryption and compression needs to be computationally inexpensive to conserve energy. This will ensure that the charging requirements of the system are not greatly increased and that the system can still survive for a suffiecient time period once deployed.

2.1.4 Analysis of UR04

System must encrypt the data.

The data will be sent over the internet and requires a layer of encryption to protect the contents. After passing through the system, the data should not be human readable or decryptable without access to the key. As the data is not highly sensitive, complex encryption is not required. Additionally, the implementation must be lightweight as discussed in UR03.

2.1.5 Analysis of UR05

System must compress the data

Data gathered on the SHARC buoy will be transmitted using Iridium satellite connections. This is expensive and thus, the number and size of transmissions required must be minimised. Additionally, in order to preserve battery life [2], as discussed in UR03, the number of packets transmitted must be minimised.

2.1.6 Analysis of UR06

System must be able to transfer data to another device for decryption and decompression

When the SHARC buoy is deployed it will transmit the data via Iridium satellite networks [2] however, this transmission process is outside the scope of this project. For the purpose of this project, wired transmission must be used to transmit the encrypted-compressed data to another device.

2.2 Functional Requirements

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

Table 3: Functional requirements addressing the needs of the system

Requirement	Description
FR01	System to compress data such that 25% of the Fourier coefficients are recoverable after decompression
FR02	Encryption subsystem to pass processed data to the compressed subsystem
FR03	All 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, algorithms should be efficient
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

2.3 Specifications

Analysis of both the user and functional requirements resulted in the following system specifications.

Table 4: Specifications used to determine the design and functionality of the desired IP

Specification ID	Description
SP01	A minimum of 25% of the Fourier coefficients must be present after decompression
SP02	The compression ratio must be less than 1, i.e. the compressed data must be smaller than the original data.
SP03	100% of the 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	SPI must be used to interface with the ICM-20649 and ICM-20948
SP07	System must transfer the data over UART following encryption

2.4 Feasibility Analysis

The following feasibility analysis is performed to ascertain whether or not the specifications are achievable in the system design and development.

Table 5: Feasibility Analysis

Specification ID	Level of Feasibility	Reasoning
SP01	HIGH	There are many lossless and low loss compression algorithms that exist and could be used
SP02	HIGH	There are many compression algorithms that can compress files much smaller than 30% of the original size
SP03	HIGH	Provided the key is correct, 100% of the decrypted data should match the original sensor data
SP04	HIGH	Provided the encryption algorithm is implemented correctly, very little of the original data should match the cipher text in the encrypted file
SP05	HIGH	Unless there are unforseen circumstances that require otherwise, all code will be written in C
SP06	HIGH	Unless there are unforseen circumstances that require otherwise, SPI will be used when developing and testing
SP07	HIGH	While this is a key specification, as we each have access to an FTDI for transferring encrypted data over UART

2.5 Changes to Requirements, Specifications and Feasibility Analysis

Changes were made to the functional requirements, specifications and feasibility analysis. These are detailed in the tables below:

Table 6: The old and current traceability matrix side by side for ease of viewing

Old Requirement Traceability Matrix

UR	FR	Specs	ATP
UR01	FR01	SP01	AT02
UR02	FR06	SP07	AT07
UR03	FR05	SP04, SP06	
UR04	FR03, FR04	SP05	AT04, AT05
UR05	FR01, FR02	SP02	AT03
UR06	FR07	SP08	AT01

Requirement Traceability Matrix

UR	FR	Specs	ATP
UR01	FR01	SP01	AT02
UR02	FR05	SP06	AT09
UR03	FR04	SP05	
UR04	FR03	SP03, SP04	AT04, AT05
UR05	FR01, FR02	SP02	AT03
UR06	FR06	SP07	AT01

Table 7: Changes to Functional Requirements

Requirement ID	Changes	Reasoning
FR02	Encryption is required to pass processed data to compression (not the other way around)	System was redesigned to first encrypt and then compress data
FR03	All measured data (not compressed data) must be encrypted	System was redesigned such that encryption happens before compression
FR04	Requirement removed	Requirement covered in FR05
FR05	ID updated to FR04	FR04 removed
	Use minimal computation and processing, algorithms should be efficient	Updated to clarify requirement
FR06	ID updated to FR05	FR04 removed
FR07	ID updated to FR06	FR04 removed

Note: All old functional requirements and changes are specified in *Table 25* of *section 9*.

Table 8: Changes to Specifications

Specification ID	Changes	Reasoning
SP02	Updated to use the term data instead of files	STM32F0Discovery does not use files
SP03	100% of the data (not compressed data) must be recovered after decryption	System was redesigned to first encrypt and then compress data thus decompression happens before decryption
SP04	Specification removed	This cannot be measured accurately
SP05	ID updated to SP04	SP04 removed
SP06	ID updated to SP05	SP04 removed
SP07	ID updated to SP06	SP04 removed
SP08	ID updated to SP07	SP04 removed
	The data should be transmitted following compression and not encryption	System was redesigned to first encrypt and then compress data

Note: All old specifications and changes are specified in *Table 26* of *section 9*.

Table 9: Changes to Feasibility Analysis

Specification ID	Changes	Reasoning
SP06	SPI will be used instead of I2C when developing and testing	System was redesigned to use SPI rather than I2C
SP07	Level of feasibility changed to high	Gained access to serial to UART interface

3 Subsystem Design

3.1 Encryption

3.1.1 Algorithms

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. [3] 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 [4] which is necessary for this application. For those reasons, all the algorithms considered were symmetric.

An important consideration in choosing an encryption algorithm is amount of computation required and algorithm speed. This is to ensure that UR03, the system must use minimal computation to be power efficient, is met. It must also be noted that the speed of encryption is also affected by the algorithm choice. This is important because in order to reduce the amount of power usage overall, 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.

The table below was put together using data from the article PRISEC: Comparison of Symmetric Key Algorithms for IoT Devices by Saraiva et al. [5] as well as from experimentation implementing the encryption algorithms chosen in the Paper Design.

Table 10: Encryption Algorithms: Battery Drainage

Algorithm	Battery Drainage (mAh)
ChaCha20Poly1305	3.9
RC6	8.68

Table 10 shows the battery drainage for ChaCha20Poly1305 and RC6 when implemented on ARM architectures; both values are low. While this is not an explicit requirement, this does indicate that they have low power usage, Both also have STM32 libraries and other source code freely available for implementation. Thus, these algorithms are both efficient and compatible with the STM. It must however be noted that the specific library for the chosen algorithm is not used so as to minimise the size of the encryption code.

In the initial system design, the encryption algorithm to be used was ChaCha20Poly1305 as it is usually smaller to implement than RC6. **However**, while attempting to implement ChaCha20Poly1305, it was found that this algorithm is tricky to implement and had a file size of 23kB which was larger than required. For this reason, a new algorithm, **RSA** is implemented instead. This algorithm is much simpler and hence has a much smaller source file size of 7kB including 2 extra functions for demonstration purposes. This implementation was able to provide correct encryption and decryption quickly using very little space on the STM for source code.

3.1.2 Algorithm Design

The original system design used ChaCha20Poly1305 for encryption. However, after trying to implement this, the file size was found to be quite large and correct implementation was tricky. Thus, while ChaCha20Poly1305 was the first choice for the encryption algorithm, RSA was used in the final implementation to ensure correctness and to minimize source code size.

The implementation was based on Yiğit Usta's implementation of RSA. However, the following adaptations were made:

Hard-Coded Key
The public and private keys as well as the p and q values were hard-coded to 8 bit integers for the final implementation. Thus, all the initialization functions were removed. This was done to improve efficiency and so that data could be decrypted without transmitting a key (which would compromise the security). Further, this specific implementation used files to store keys and was thus incompatible with the STM.
8-bit integers were used for the keys to ensure that encrypted bits were less than 32-bits as the STM32 uses a 32-bit architecture. Additionally, this allows the system to maximise the amount of data that could be transmitted at a time without completely compromising on data security.

Unsigned Long Long Integers changed to Integers
The original implementation of both encryption and decryption output unsigned long long int values which are 64-bits in size. This was not compatible with the STM32's 32-bit architecture and both functions were thus changed to output integer values.

Removal of File usage
The STM32 cannot work with files. For this reason, encryption was redesigned to take in a 1D array and to write output to a 1D array. The output array in the combined implementation of the subsystems is a global variable which is accessed by compression so that data can be compressed and transmitted the data to a computer.

3.2 Compression

3.2.1 Algorithms

Two algorithm types are considered namely 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. [6] The data recorded by the IMU is numerical and slowly changes over time. This is a suitable compression algorithm. Alternatively, dictionary compression makes a list of recurring groups of values that appear in the data and then replaces them with an index value from the dictionary. This algorithm works well when there is repeating data. [6].

In this system, the compression algorithm is run on encrypted data. Although encryption randomises data, the chosen encryption algorithm encodes the data to numerical values. This means that it uses repetitive numbers and number sequences. The algorithm best suited for repetitive data is a **dictionary-based compression**.

Important considerations in choosing a compression algorithm are compression ratios and algorithm speed. These must be carefully considered to ensure that SP02, the compression ratio must be less than 1, and UR03, the system must use minimal computation to be power efficient, are met. Additionally, the device used for testing (STM32F0Discovery) has limited RAM and flash memory. This requires consideration of the algorithm's source code size and further limits the algorithms available.

Two dictionary-based algorithms are compared using compression ratio percentage, compression speed and source code size as benchmarks. The compression ratio is defined as:

$$\text{compression ratio} = \left(\frac{compressedfilesize}{originalfilesize}\right)$$

The values in the table below were obtained from the lzbench mark [7] which has run compression on a number of different types and sizes of 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.
From *Table 11*, 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. Considering the RAM limit of the STM32F0Discovery, LZW is not a suitable solution. Although, LZSS's compression speed is significantly

Table 11: Compression Algorithms

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

smaller, it is still fast enough to be efficient for this use since megabytes of data will be compressed at a time not gigabytes. For these reasons, **LZSS is chosen as the algorithm**.

3.2.2 Algorithm Design

The implemented version of LZSS compression is based off of Haruhiko Okumura's LZSS encoder-decoder. Adjustments have been made to suit the requirements of this design however, the core algorithm uses Haruhiko Okumura's algorithm.

Removal of File usage
In order to run the algorithm on the STM32F0, the source code has been adapted to both take in and store the compressed bits in an array aas the processor is not file-compatible.

Printing compressed bits to integer not char
Additionally, the compression algorithm originally compressed bits to characters. However, the STM32F0 is a 32-bit processor and is unable to use the full range of character values the algorithm was originally designed for. Additionally, the UART protocol writes any value it does not recognise to a random and unrecognisable character. Using the algorithm in this form would make decompression impossible. To solve these problems the compression algorithm has been altered to instead use only integers. The compression algorithm now outputs the value of the compressed bits as integers rather than characters. To accommodate this change, the decompression algorithm has been changed to read in integer values from a file.

Solving reverse 2's Complement
However, when implementing this algorithm on the STM32F0, any negative integer was converted to a positive 8-bit integer. A corrective step has been implemented to convert the number back to its negative form before storing in the array and transmitting. The code used to do this is shown below.

Listing 1: Compression negative number fix

```
int correctBitbuffer(int bitbuffer) {
        int val;
        int tempvar = (int)log2(bitbuffer)+1; // get number of bits
        if (tempvar >=8) {
                val = 256 - bitbuffer;
                val = -1 * val; // convert to negative value
                return val;
        }
    return bitbuffer;
}
```

This solution works as the encryption key is 8-bit. If a larger or smaller key is required, this method would need to be adapted accordingly.

3.3 Inter-Subsystem and Inter-Sub-subsystems Interactions

The system is designed to first encrypt and then compress the data. In general, encryption transforms data into high-entropy data, making it random while compression relies on patterns in order to perform well and reduce the data size. However, the encryption algorithm chosen, RSA, converts the input data into purely numerical data. This means while there isn't necessarily a pattern to the data, there are many repetitions of characters which allows for more efficient compression. Thus, the system is designed to first encrypt and then compress the data.

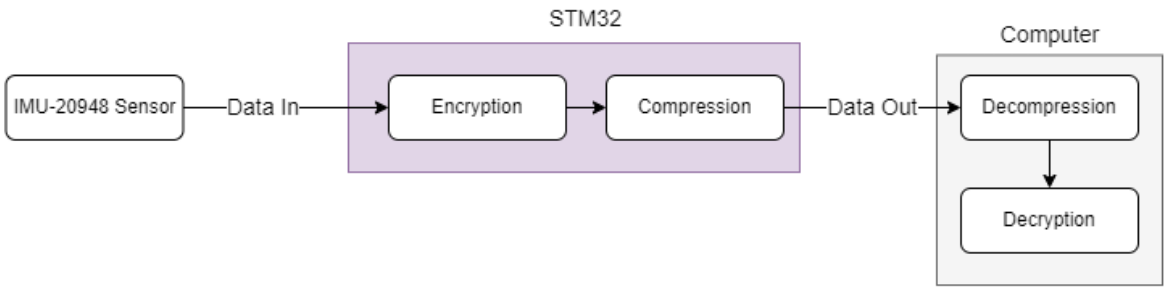

Figure 3: Diagram of Inter-Subsystem interactions

Figure 3 shows the final system flow as it is expected to interact. However, the system interaction for testing is slightly different, as show in *Figure 4*:

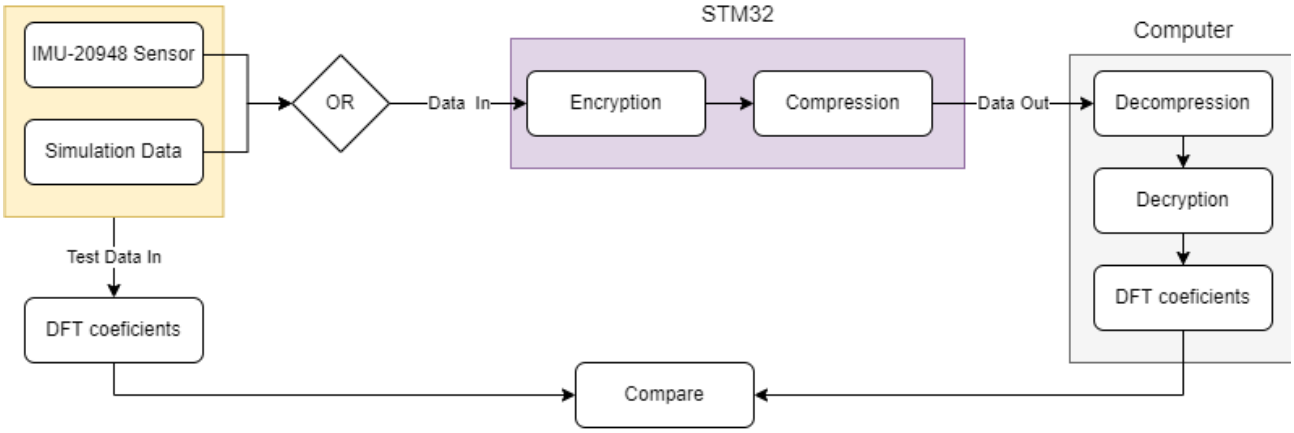

Figure 4: Diagram of Inter-Subsystem interactions for Testing and Development

Figure 3 illustrates the system configuration for extensive validation testing. Both simulation and sensor data are used to test the system at different phases of development. Additionally, the Fourier coefficients of the data before compression and after decompression must be compared to confirm that at least 25% are preserved, inline with SP01.

3.3.1 Physical Interfacing

Some parts of the system need to be connected physically and will be interfaced as described in table 1 below.

Table 12: Interfacing specifications

Interface	Description	Pins/Output
I001	SparkFun 9DoF IMU Breakout board to STM for data transfer (SPI)	• MISO: Breakout MISO* to STM PB14 • MOSI: Breakout MOSI* to STM PB15 • SCLK: Breakout SCLK* to STM PB13 • CS: Breakout CS* to STM PB12 (GPIO) • Power: Breakout 1V8-5V5* to STM 3V • GND: Breakout GND* to STM GND
I002	STM to FTDI for data transfer to computer (UART)	• STM PA2(RX) to FTDI RXD • STM PA3(TX) to FTDI TXD
I003	FTDI to computer for data transfer from STM	• Assuming the FTDI is connected to a Micro B USB port (which can be used to connect to a USB cable to connect to a computer), output via cable

*connection indicated in the image below: [8]

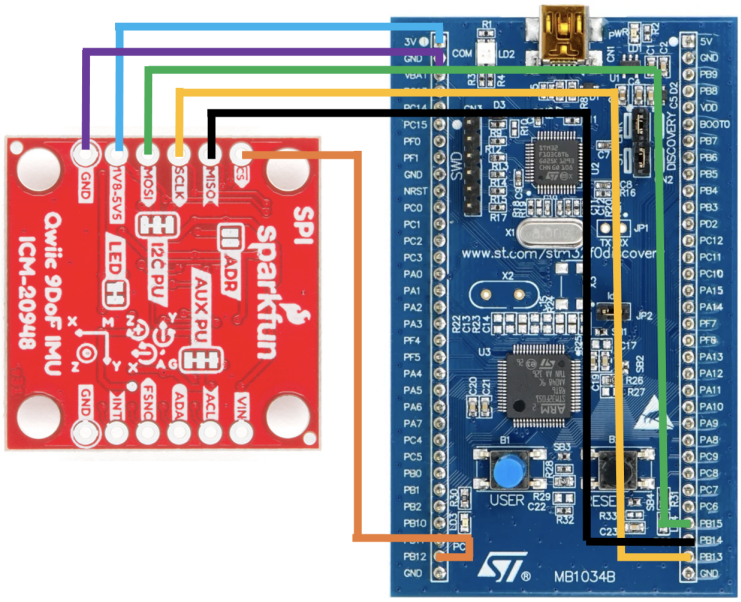

Figure 5: Showing interfacing between SparkFun 9DoF IMU Breakout board and STM32F0Discovery

3.3.2 Interfacing with the IMU

Salient Features of the ICM-20948 compared to ICM-20649

The IMU used in the testing and development of this system is the ICM-20948. However, the IMU used on the SHARC buoy, and hence the IMU intended to be used with the system, is the ICM-20649. While these IMU's are similar, they have some differences in what they can each do.

One of the most notable differences between the 2 for this application is that the ICM-20948 is a 9-axis

device while the ICM-20649 is a 6-axis device. Both chips have a 3-axis accelerometer and a 3-axis gyroscope, however, the 20948 has a 3-axis compass which the 20649 does not. Thus, it is important to ensure that the system does not extract or use the compass readings. This will maintain that the system can be implemented using the 20649.

The ICM-20649 has higher full scale resolution for both the accelerometer and the gyroscope. The 20948 has a gyroscope with a maximum programmable FSR of $\pm 2000 dps$ and an accelerometer with a maximum programmable FSR of $\pm 16g$. The corresponding values for the 20649 are $\pm 4000 dps$ and $\pm 30g$ respectively. This means that the 20649 can read almost double the acceleration and double the angular velocity with accuracy than the 20948. However, for the purpose of the SHARC buoy, low values of $\pm 500 dps$ and $\pm 2g$ [2] are needed; making system testing possible on the 20948.

The 20649 IMU has a wake-on-motion interrupt for low power operations. This feature is not available on the 20948. Thus, it will not be configured or used in this implementation but this feature should be used in the actual implementation on the SHARC buoy to maximise power conservation.

Software Interfacing: Code design

The code used to interface with the ICM20948 is based on the stm32_hal_icm20948 github and has been adapted for the design requirements.

The code was adapted to be compatible with the STM32F0 not the STM32F4. The gyroscope and accelerometer reading methods were changed to read from the high and low registers for each value and then combine them into a 16 bit value. The calibration was changed to also read from high and low offset registers for all values.

3.4 Possible Bottlenecks

Data is being transferred and processed throughout the system, key bottlenecks identified in the process are:

3.4.1 SPI & UART communication lines

Between the sensor and the STM32:
The connection is made using SPI which is a four-wire protocol. In order to transmit over SPI the voltage of the wires need to be driven high and low using a resistor. Additionally the wires have internal capacitance which slows down the process. This would cause a bottleneck between the flow of data into the STM32F0 and the encryption algorithm.

Between the FTDI and the STM32:
The connection is made via UART to transmit data to a PC. Because the FTDI is only used to recieve data, only one wire is used. In order to transmit via UART the voltage of the wire needs to be driven high and low using a resistor. Additionally the wire has internal capacitance which slows down the process. This would cause a bottleneck between the flow of data from the STM32F0 to the FTDI and ultimately the PC.

3.4.2 Compression and Encryption

The compression algorithm requires a block of data to compress at one time to be effective. This requires all encrypted bits to first be stored until encryption is done, before sending them to compression. This causes a bottleneck between encryption and compression. The less efficient the encryption algorithm is, and the more data there is to encrypt, the greater the compression idle time is.

3.4.3 STM32F051

The STM32F051 has limited RAM (8 kBytes) and the tasks of reading data from the sensor, processing the data and then transferring it to the computer are process heavy. Ultimately the STM32F051 will be a bottleneck in the entire system.

4 Validation Using Simulated or Old Data

4.1 Necessity of Simulation-Based Validation

In the initial stages of system design and testing, limiting the amount of unknown elements is important. Combining many unknown elements at anytime will make finding points of failure difficult and could result in system failure or complete system redesign. This can prove expensive and time consuming and should be avoided. Simulation data provides a way of isolating subsystems and testing that they work correctly individually and together before interfacing with additional hardware. This ensures that when interfacing occurs, it is easier to identify core points of failure and bugs in the design since it is already known that the subsystems work in isolation.

Additionally, simulating data allows testing of the system's performance and interaction under a variety of different situations. Simulations allow system tests under conditions and using data that might be rarely observed in the real-world [9]. This provides the ability to identify key performance criteria of the system and flaws or points of improvement in the system design.

Furthermore, simulations allow tests to be repeated as necessary [9] which is invaluable in system development. When using changing data, it is difficult to notice system faults or the sources of failure. Repetitive testing eliminates unknown factors and proves effective in finding minor or hidden bugs and points of failure of the system. This cannot be or is not easy to achieve when using live data or in real-world testing. Repetitive testing also provides insight into system integrity, if the same data is repeatedly used then the system should respond consistently and in a predictable matter. If it does not, then the system cannot be reliable. This does not take into account systems where randomisation is favoured. Using real-world data would make it difficult and unlikely to identify the lack of system integrity.
Ultimately, simulation-based design is necessary for early development of a system. It allows for repetitive testing, making bug fixing and identifying points of failure simpler. Without simulation-based validation, it would be difficult to complete system testing or ensure the system's integrity.

4.2 Steps of Validation

The following steps were taken to validate the system using simulated data:

1. Send data to and from the STM (ensure data can be transferred successfully)
2. Compress and decompress simulated data on a PC(ensure compression works as expected)
3. Encrypt and decrypt simulated data on a PC (ensure encryption works as expected)
4. Compress and encrypt then decompress and decrypt simulated data on a PC(ensure algorithms work together)

For each of the above steps, the final output is compared to the original data using a python script, verify.py, and a MATLAB FFT script, FFTScriptDualPlot.m.

4.3 Data Used

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

The simulated data used for validation was a sample taken from the supplied simulation data. The sample contains numerical measurement values taken over different periods of time.

The SparkFun 9DoF IMU Breakout incorporates the ICM-20948 chip which 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" [2]. Additionally, the temperature sensor onboard the IMU chip will

not be used as the SHARC buoy uses a dedicated temperature sensor chip [2]. Thus temperature data from the simulation data is not considered or designed for.

The test data files supplied include other values such as Yaw, Roll and Pitch. 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 *UR03*, the system must use minimal computation to be power efficient. Additionally, the STM32Discovery board has minimal storage space available thus fewer values per recording will allow for more data to be compressed and encrypted at once.

Ultimately, only **acceleration** (accelorometer) and **angular velocity** (gyroscope) **data** is used.

4.3.1 Dataset(s) Analysis

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 are 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 datasets, 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 datasets, the accelerometer and gyroscope (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 and gyroscope are included here. The plot of the combined data is excluded as it does not provide relevant insight into the data.

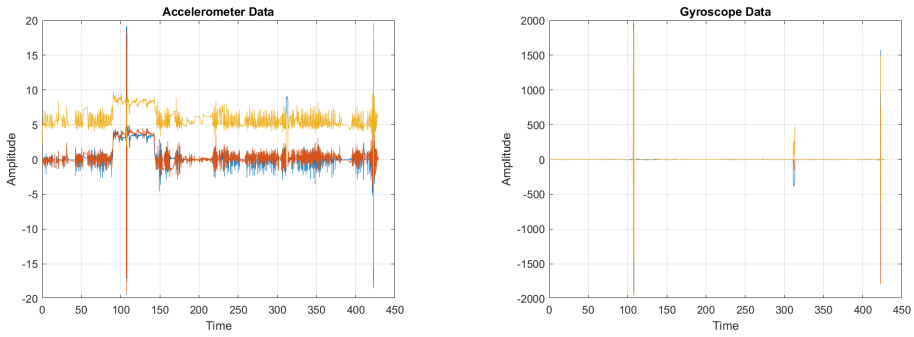

Figure 6: Time Domain representations of Walking Dataset

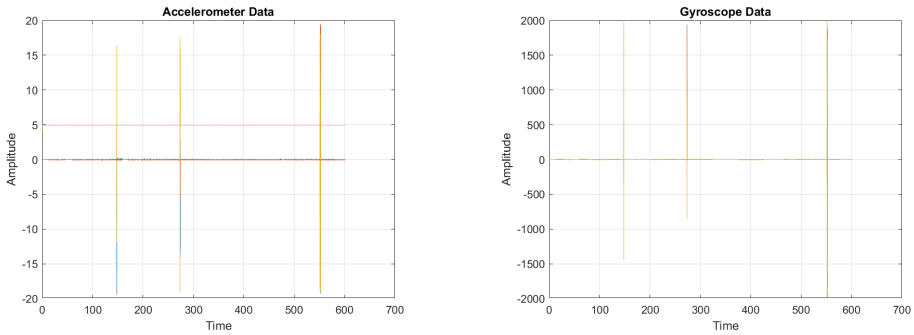

Figure 7: Time Domain representations of Higher Sampled Turntable Dataset

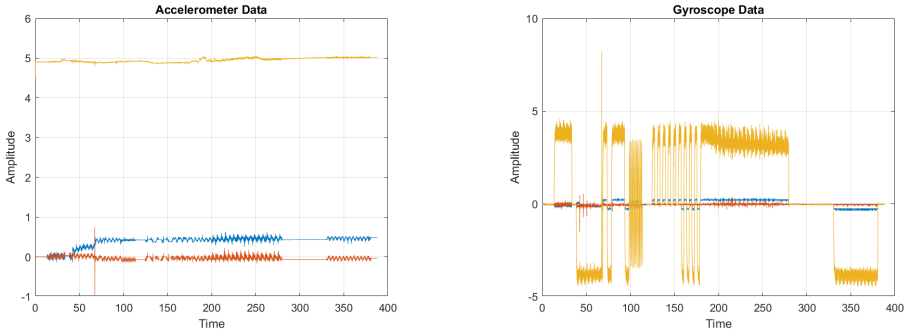

Figure 8: Time Domain representations of Lower Sampled Turntable Dataset

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 [2]. This makes the Fourier coefficients valuable and is why they need to be preserved as detailed in *SP01*. Thus, when deciding on which test-data to use for testing, the frequency domain plots are important:

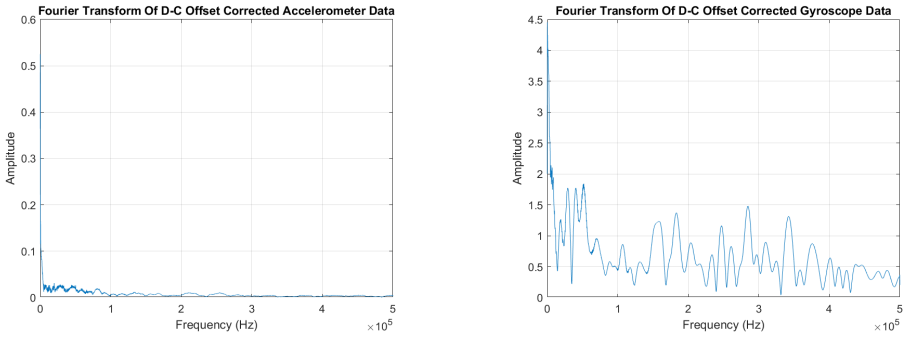

Figure 9: Corrected Offset FFT for Walking Dataset Acceleration and Angular Velocity

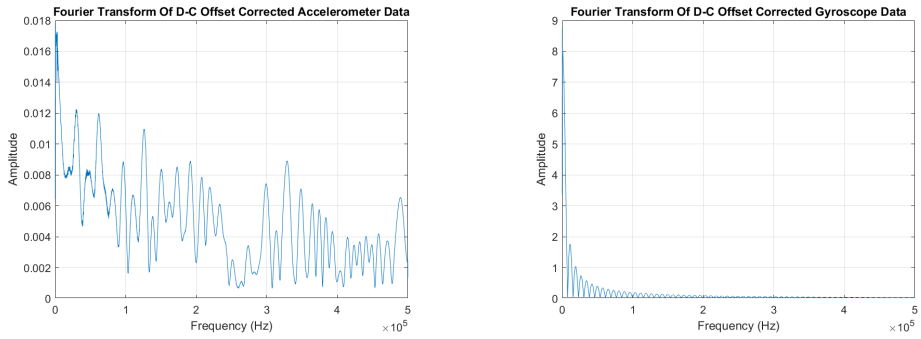

Figure 10: Corrected Offset FFT for Higher Sampled Turntable Dataset Acceleration and Angular Velocity

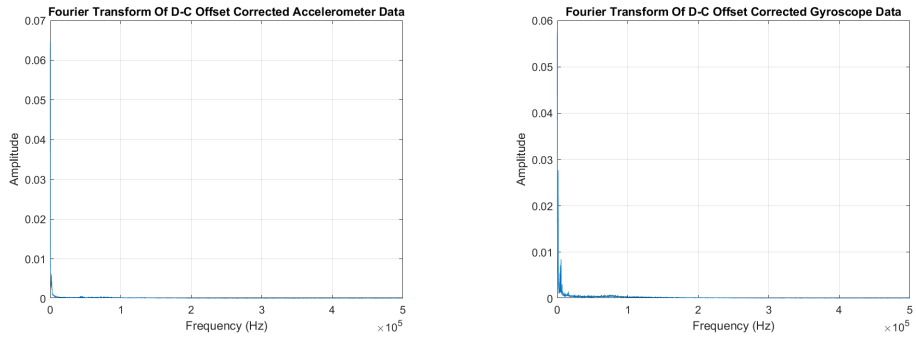

Figure 11: Corrected Offset FFT for Lower Sampled Turntable Dataset Acceleration and Angular Velocity

Looking at *Figure 11*, there are few Fourier coefficients in any of the data plots making it less suitable for testing than the data plotted in *Figure 9* or *Figure 10*. Between the data plots in *Figure 9* and *Figure 10*, *Figure 10*'s plots have the greatest number of Fourier coefficients (peaks). Using the data plotted in *Figure 10* will be more effective in showing what percentage of coefficients are preserved. Thus, the higher sampled turntable data will be used for testing throughout this report.

4.3.2 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 ATP 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. 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 and angular velocity data 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.

4.4 Experiment Setup

Note: *The simulation-based validation detailed below was carried out on the original system design where compression happened before encryption. This mainly affects the results recorded for the combined algorithms experiment. Following system redesign, further simulation-based testing was done. The combined_chars.c and combined_integers.c programs were used to test and validate the system redesign. ATP, specification and requirement references have been updated to match the new versions.*

4.4.1 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. Using loops and string formatting, this data is transmitted over UART to a PC. The output is saved and, using verify.py, compared to the data stored in the array.

The input data is a subset of the high sampled turntable data. This specific data was used for the reasons highlighted in *subsubsection 4.3.1*: Simulation Dataset Analysis.

4.4.2 Combined Algorithms

Aims to determine that the system works, inline with AT06: successful full system cycle on computer (without sensor).

The compression and encryption algorithms are combined into a single c file.

In the original system design, the test data is passed into the program and the encrypted data is passed to compression without human interaction. The input is the csv file described in section 4.2 above and the expected output is an encrypted lzss compressed file. The file is then decrypted and decompressed. The contents of the output file is verified against the original test data file using verify.py and by comparing the fourier coefficients (to ensure SP01 is met).The input data here is an encrypted, lzss-encoded file and the expected output is a csv file containing the same data as in the input csv file.

In the updated system design, the test data is encrypted and the encrypted data is passed to the compression algorithm without human interaction. The input to this is the sample data described in section 4.2 above hard-coded into an array and the output is an encrypted-compressed file of this data. The file is then decompressed and decrypted. The contents of the output file is verified against the original test data file using verify.py and by comparing the fourier coefficients (to ensure SP01 is met).The input data here is an encrypted-compressed file, the expected output is a file containing the same data as the input array. **The updated system was tested more extensively using the IMU and hence the experimental setup and results for it are not discussed in this section.**

4.4.3 Compression: General Functionality

Aims to test that the compression subsystem works in line with AT02: Fourier coefficients recovered and AT03: Compressed file size smaller than original

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 FFTScriptDualPlot.m script.

4.4.4 Compression: Window Sizes

Aims to optimise the algorithm for the test data in line with UR03 and FR04: the algorithm should use minimal computation and be efficient.

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 13: 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 2: 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.

4.4.5 Encryption: General Functionality

Aims to test that the encryption subsystem works inline with AT04 and AT05 and aims to compare input and output data in line with UR04, FR03 and SP04

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

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

In order to validate the algorithms, a sample set of data must be setup. After the data is setup, it must be encrypted and saved in a new file. A python script, verify.py, and a MATLAB FFT script, FFTScriptDualPlot.m, are used to check the input and output are not the same. The percentage of the file that remained the same after encryption is also calculated here. To confirm it is not visibly decipherable, the file will also be opened and viewed. 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 using verify.py and FFTScriptDualPlot.m. The files should be identical.

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

Listing 3: 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 4: Encryption Input 2

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

4.5 Results

4.5.1 Sending data from STM32F0

AT01 PASS: Successful interfacing with STM32F0

The experiment completed successfully. The data was transmitted to the PC over UART using the an STM32Cube project. The transmitted data was saved and compared to the expected sample data using verify.py and the values in the files match.

4.5.2 Combined Algorithms

AT06 PASS: Successful full system cycle on computer (without sensor)

The data-set was passed into the combined_chars.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 FFTScriptDualPlot.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:

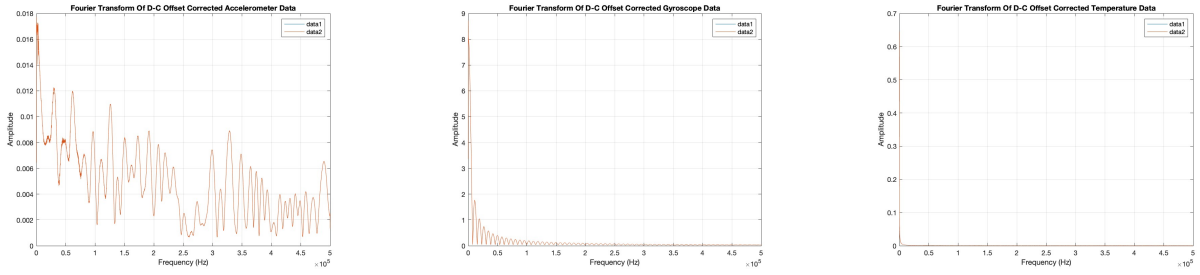

Figure 12: Corrected Offset FFT plots for comparison between data-set before and after compression and encryption

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

4.5.3 Compression: General Functionality

AT02 PASS: Fourier coefficients recovered and AT03 PASS: Compressed file size smaller than original

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:

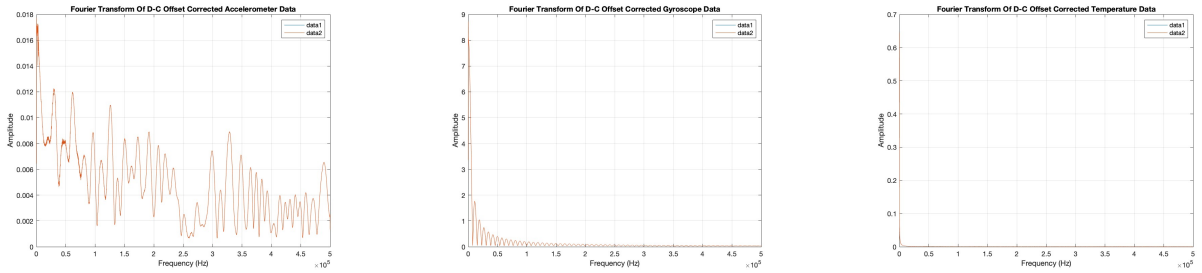

Figure 13: Corrected Offset FFT plots for comparison between data-set before and after compression

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 FFTScriptDualPlot.m, the coefficients of the data-set before and after compression were compared and found to be exactly the same. This satisfies AT02.

4.5.4 Compression: Window Sizes

Table 14: 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 14* 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 system must be compatible with ICM-20649 and ICM-20948 IMU chips. 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.

4.5.5 Encryption: General Functionality

AT04 PASS: Encrypted data not human readable and AT05 PASS: Decrypted data identical to original data

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.

AT04, UR04, FR03 and SP04 require 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 plain text 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, UR04, FR03 and SP04 are passed for this text case.

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

iAmAFile hello t

It must be noted here that ”\n” was used as a marker for the end of data in this function (this has since been updated to use ”}” as the marker for the end of the data) 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, UR04, FR03 and SP04 are 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 Validation Using the IMU

5.1 Necessity of Hardware-Based Validation

While simulation-based validation is valuable, it does not replicate all potential bugs caused by hardware and thus does not show how the system will perform once deployed. Therefore, while it is important, it is not a replacement for hardware-based validation. Data used in simulation-based validation often fails to reflect the level of complexity of real-world applications. [9] Hardware-based validation allows an engineer to test subsystem interactions and run timing tests in scenarios that resemble the real-world application of the system. This is important to ensure firstly that the system can actually handle complex data and secondly the efficiency of the system when handling complex data.

For this application specifically, while the encryption and compression subsystems can be tested with complex data very similar to expected real-world data, simulating this on a computer does not validate the actual implementation of the system because the architecture has different specifications- most importantly that the computers used have a 64-bit architecture while the STM32 runs a 32 bit architecture. Thus, seemingly working implementations may not work at all on the required hardware. Further, the STM32 has limited capabilities compared to current computers and thus, requires much more optimized and simple software solutions to be implemented.

Efficiency is incredibly important for this system to minimize power usage. Thus, timing tests are of vital importance. These tests cannot be simulated as the hardware of a computer and the STM32 are simply too different. Thus hardware-based validation is necessary for this application.

Despite hardware-based validation being important, it is a step that should be taken later in the development process just before deployment. Solutions should be largely ready for deployment with possible minor bug fixes when they are implemented on hardware. This ensures that hardware is not damaged and destroyed as it can be expensive to fix and replace. Thus, hardware-based validation should only be completed after simulation-based validation has been completed and sufficient steps have been taken to ensure that the simulation can represent the hardware as accurately as possible.

5.2 Steps of Validation

The following steps were taken to validate the system using simulated data:

1. Implement encryption on STM. Data must be passed in, encrypted and passed back to the PC (ensure the data decrypts correctly).
2. Implement compression on STM. Data must be passed in, compressed and passed back to the PC (ensure the data decompresses correctly).
3. Implement combined project to encrypt and then compress data on the STM. Data must be passed in, encrypted, compressed and passed back to the PC (ensure data decompresses and decrypts correctly).
4. Connect the IMU to the STM and ensure data can be read from the IMU and passed back to the PC correctly.
5. Implement second combined project to take in data from the IMU, encrypt it, compress it and pass it back to the PC. The IMU data must also be passed to the PC so that it can be compared to the decompressed-decrypted output file.

For each of the above steps, the final output is compared to the original data using a python script, verify.py, and a MATLAB FFT script, FFTScriptDualPlot.m.

5.3 Data Used

The test data was gathered using the SparkFun 9DoF IMU Breakout board's IMU (ICM-20946) sensor.

The data used in this section is live data read from the IMU. The SparkFun 9DoF Breakout board has an onboard compass, magnetometer, accelerometer, gyroscope and temperature sensor. To comply with *UR02*, the system must be compatible with the ICM-20649, the board's onboard compass and magnetometer will not be used for data collection. Additionally, the on-chip temperature sensor will not be used as the SHARC buoy has a dedicated on-board temperature sensor [2]. Only acceleration and angular velocity in the x, y and z directions will be recorded and used for testing the system.

Additionally, minimal computation must be used so that the system is power efficient in line with UR03. Thus, no calculations will be performed onboard. Instead, only the raw acceleration and angular velocity data will be transmitted which can then be used for complex processing and calculations. Since the discovery board has limited flash memory and RAM, reducing the calculations and data values to store per recording will allow for better system testing.

Ultimately, only **acceleration** (accelerometer) and **angular velocity** (gyroscope) **data** is used.

5.3.1 Dataset(s) used for Testing

The IMU module is also validated using the following tests: In order to validate the IMU module is working, the following tests were run:

Table 15: IMU testing

IMU Test	Description
IT01	Test that the startup sequence runs correctly by ensuring the initialization function is completed before any other functions are run
IT02	Test that the direction of measurements is recorded correctly by accelerating the sensor in the positive and negative x, y and z directions and confirm that the recorded value has the correct direction (accelerometer testing)
IT03	Test that the direction of measurements is recorded correctly by turning the sensor in the positive and negative x, y and z directions and confirm that the recorded value has the correct direction (gyroscope testing)
IT04	Test that magnitude of readings increase and decrease as expected relative motion of pendulum
IT05	Test that **all** relevant readings change if the sensor is moved in multiple axes as expected from wave motion (eg if there is acceleration in the x direction and the y direction, both readings should change)

Pendulum
To check that the sensors are working as expected a pendulum system is used. This will clearly show trends in any form of acceleration allowing for thorough testing of both the gyroscope and the accelerometer.

Wave
At the time of this experiment, equipment needed to simulate the real SHARC bouy environment is unavailable. An alternative simulation setup is made using a breadboard and a bed sheet. The system is placed on the breadboard and the breadboard is placed on the sheet. The sheet is gently moved up and down and in different directions to simulate waves.

Testing this gentle motion also helps ensure that the system can sense the effect that wave swells have on a sheet of ice.

5.3.2 Dataset(s) Analysis

The data plotted in this section was gathered using the tests outlined in *Table 19*. The test data is grouped into wave, pendulum and jolt data.

Time Domain Representations

Data gathered from the accelerometer and gyroscope for each of the 3 tests are included here. The plot of the combined data is excluded as it does not provide relevant insight into the data.

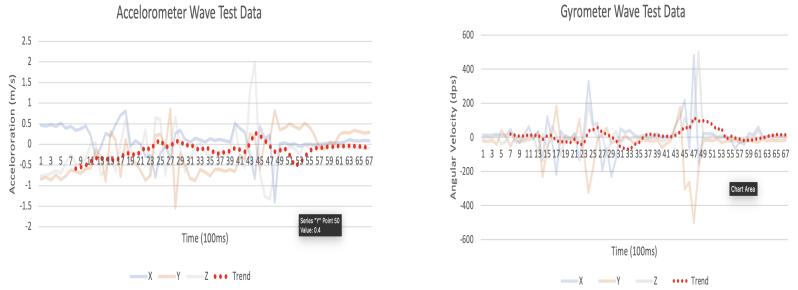

Figure 14: Time Domain representations of Wave Test Dataset

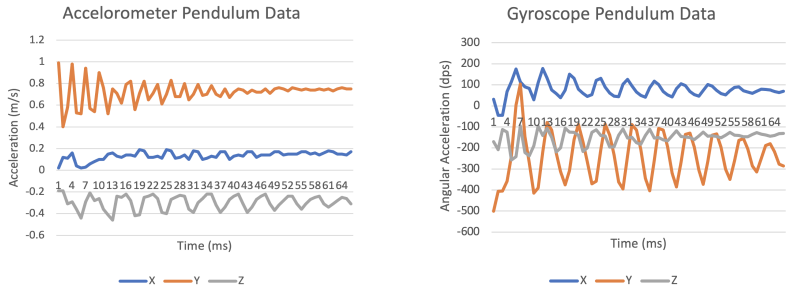

Figure 15: Time Domain representations of Pendulum Test Dataset

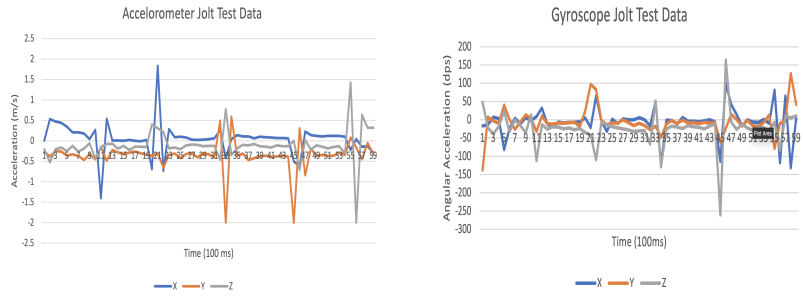

Figure 16: Time Domain representations of Jolt Test Dataset

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 [2]. This makes the Fourier coefficients valuable and is why they need to be preserved as detailed in *SP01*.

The plots below include DFTs of the unfiltered (orange) and lowpass filtered (purple) data.

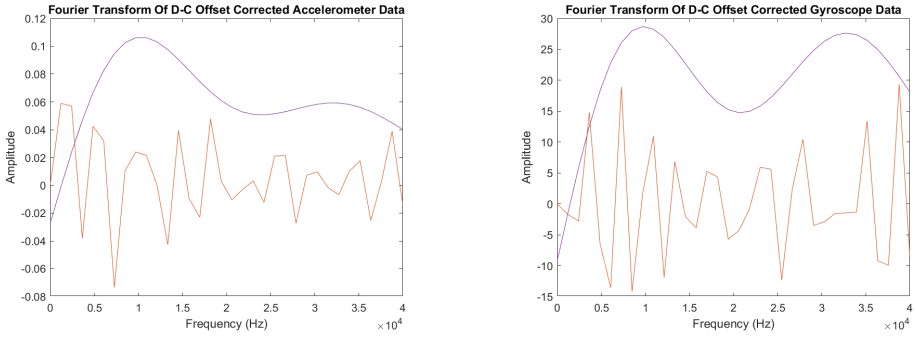

Figure 17: Corrected Offset FFT for Wave Test Dataset Acceleration and Angular Velocity

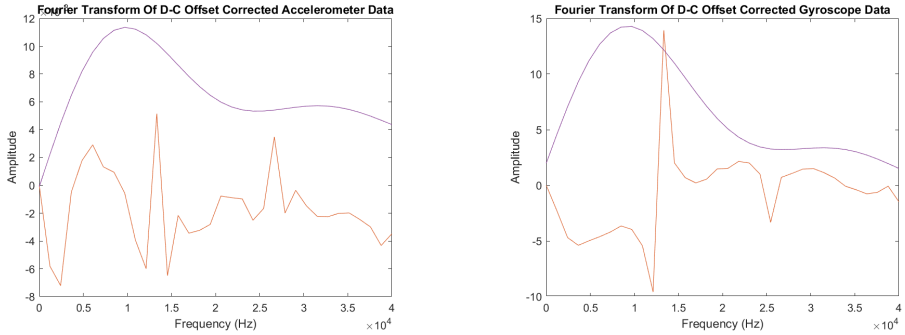

Figure 18: Corrected Offset FFT for Pendulum Test Dataset Acceleration and Angular Velocity

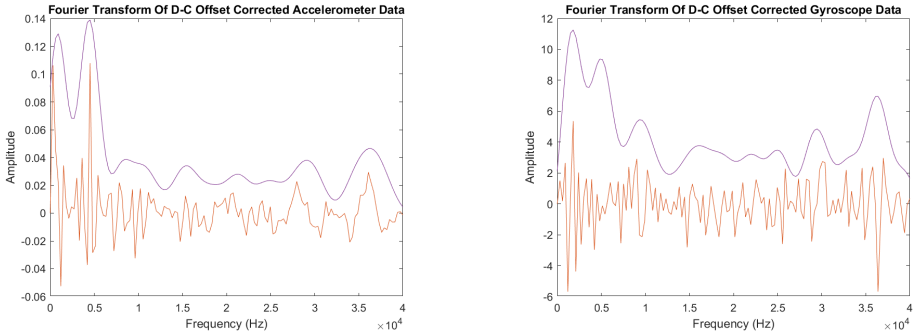

Figure 19: Corrected Offset FFT for Jolt Test Dataset Acceleration and Angular Velocity

Looking at all the figures above, each test has a different amount of Fourier coefficients. This range will allow thorough testing of the system's ability to preserve and recover Fourier coefficients after compression and decompression. Ultimately, these datasets are suitable for testing the system inline with *SP01*, a minimum of 25% of the Fourier coefficients must be present faster decompression.

5.4 Experiment Setup

5.4.1 Compression and Encryption work on STM32F0

Aims to determine that the algorithms work together on the STM32F0, in line with AT07: Successful full system cycle on STM32Discovery (without sensor).

The algorithms are flashed onto the STM32F0 using the SHARC_buoy_no_sensor project. This combined program encrypts and compresses a hard-coded input array and sends the compressed-encrypted data over UART to a computer. The expected output is compressed-encrypted data.

The data received at the computer is then decompressed and decrypted. The contents of the file are verified against the original test data by comparing the fourier coefficients using FFTScriptDualPlot.m (to ensure it meets *SP01*: A minimum of 25% of the Fourier coefficients must be present after decompression).

5.4.2 Compressing and Encrypting IMU data

Aims to determine that data read from the IMU can be encrypted and compressed on the STM32F0, in line with AT09.

The SHARC_buoy project is updated to read live data from the IMU and send it to encryption. The data read from the IMU is first sent over UART to a computer before being encrypted and compressed. The compressed-encrypted data is then sent over UART to the computer.

The data received at the computer is then decompressed and decrypted. The contents of the file are verified against the recorded live data by comparing the fourier coefficients using FFTScriptDualPlot.m (to ensure it meets *SP01*: A minimum of 25% of the Fourier coefficients must be present after decompression).

5.4.3 Full System Efficiency

Tests the systems efficiency in line with FR04: System must use minimal computation and processing

The combined project is run on the STM32Discovery board with added HAL_GetTick() functions. These functions are used to calculate the time taken for encryption, compression and the full system to run. The program should be run multiple times before taking data readings to prevent results being skewed by cache warming.

5.4.4 IMU: Interfacing

Aims to determine that data can be read from the sensor and passed to a PC in line with AT08.

The SparkFun 9DoF IMU Breakout is connected to the STM32F0 via SPI as described in *Table 12* and inline with *SP06*: SPI should be used to interface the IMU. The ICM is configured to a full scale resolution of $\pm 500 dps$ and $\pm 2g$ as specified on page 132 of Jamie's thesis [2] to be compatible with the SHARC Buoy system. The ICM-20948 project is flashed onto the STM32F0 and reads the x,y and z values of the gyroscope and the accelerometer only. The measurement data is passed over UART to the computer.

5.4.5 IMU: Initialisation

Aims to test that the IMU startup sequence runs correctly in line with IT01

A startup message is added after all IMU initialisation methods have been called and transmitted over UART. The STM32F0 is plugged in, flashed and the serial output is checked for the "IMU initialised" message.

5.4.6 IMU: Accelerometer Verification

Aims to test that the direction of acceleration is correctly recorded in line with IT02

The sensor is jolted in the positive and negative x,y and z directions. The live measurements are transmitted to the computer over UART. The readings are expected to increase in magnitude. The readings are expected to increase in magnitude rapidly and then decrease back to the steady-state value.

5.4.7 IMU: Gyroscope Verification

Aims to test that the direction of angular velocity is correctly recorded in line with IT03

The sensor is rotated in the positive and negative x, y and z directions.The live measurements are transmitted to the computer over UART. For this measurement, the change in the reading should be monitored - the values should decrease in the negative direction and increase in the positive direction.

5.4.8 IMU: Pendulum Test

Aims to test that magnitude of readings increase and decrease in line with IT04

The IMU, STM32F0 and FTDI are all placed on a bread board; carefully positioned to ensure the board hangs evenly when stationary. The board is then pulled up to one side and let go, allowing it to swing back and forth until coming to rest. The live measurements are transmitted to the computer over UART. The readings are expected to oscillate with oscillation amplitudes decreasing with time.

5.4.9 IMU: Wave Test

Aims to test that all relevant readings can change simultaneously, in line with IT05

The IMU, STM32F0 and FTDI are all placed on a bread board and then placed on a sheet. The sheet is slowly moved up and down to simulate the swells of a wave. The corners of the sheet are lifted alternately to test different swell centres. The live measurements are transmitted to the computer over UART. The readings are expected to show the general trend of the swell of the wave with a clear rise and fall from the peak of the swell.

5.4.10 Compression: Efficiency Testing

Tests the systems efficiency in line with FR04: System must use minimal computation and processing

The input data would be an integer array of encrypted data and the expected output would be an integer array of encrypted-compressed data. The output is expected to be smaller than the input. The expectation is that as the file size increases the time take to compress the file will increase linearly but not exponentially.

5.4.11 Encryption: Efficiency Testing

Tests the systems efficiency in line with FR04: System must use minimal computation and processing

A further requirement of the system is to be efficient as there is minimal power available on the buoy. This means that the system should spend as little time as possible in an active state to conserve battery life. In order to test this, different sets of data with varying sizes will be encrypted and decrypted. The process of encryption will be timed (as only encryption will occur on the micro-controller) to find the optimal amount of data to be passed through the system at a time. The data sizes to be tested are:

The input data would be a character array of data read from the IMU and the output is expected to be an integer array of encrypted data to be passed to the compression algorithm.

5.5 Results

5.5.1 Compression and Encryption work on STM32F0

AT07 PASS: Successful full system cycle on STM32Discovery (without sensor) Data was passed to STM32F0Discovery to be encrypted and then compressed. The encrypted-compressed data was passed to a computer to decompressed and decrypted. The filtered(purple) and unfiltered(orange) Fourier Transforms for both datasets are plotted below:

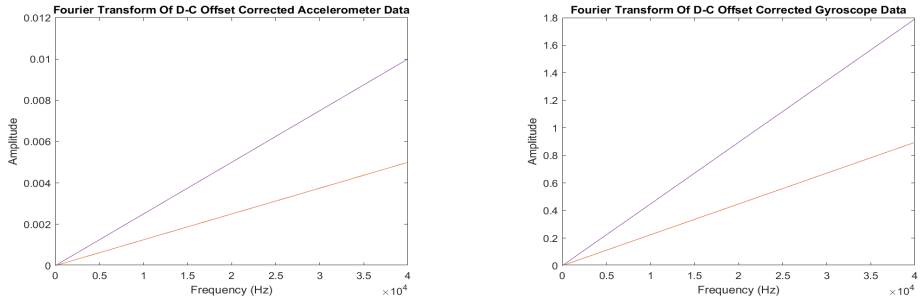

Figure 20: Frequency Domain representations of Demo Dataset

Looking at the Figure 20, there seems to only be two lines in each plot. However, this is because each line has two plots but the coefficients are the same and no change can be seen.

Since the coefficients match exactly, this confirms that the algorithms are lossless and 100% of coefficients have been preserved. Thus, AT07 is met.

5.5.2 Compressing and Encrypting IMU data

AT09 PASS: Full system cycle run successfully

A full system test was run taking readings from the IMU, encrypting then compressing this data and then transmitting both the IMU data and the encrypted-compressed bits to the PC. The encrypted-compressed bits are decompressed and decrypted and the Fourier Coefficients of this output is compared to those of the original IMU data. The filtered(purple) and unfiltered(orange) Fourier Coefficients for both datasets are plotted below:

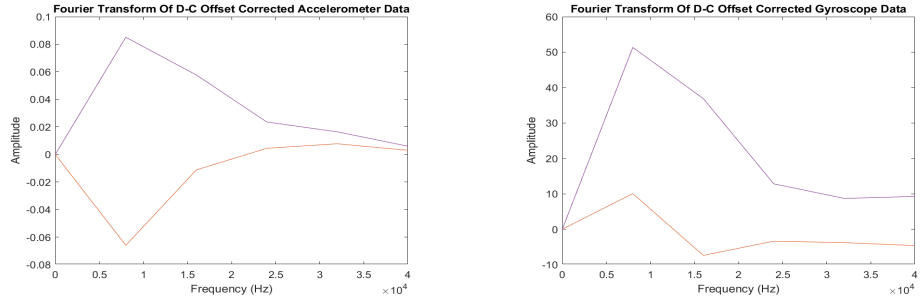

Figure 21: Frequency Domain representations of Demo Dataset

While the filtered data seems incorrect upon initial inspection, it must be noted that the filter takes in the absolute value of the function and thus, where there are negative coefficients, we expect the plot to be inverted.

From the above plots, we can see that the original IMU data and the decompressed-decrypted output data produce the same Fourier Coefficients and thus the same Fourier Plots.

The first 25% of the unfiltered Fourier Coefficients are tabulated below and the full set of coefficients for both the filtered and unfiltered data can be accessed here.

Table 16: Fourier Coefficients of Accelerometer Data

Uniltered IMU	**Unfiltered Output**
-5.551115e-18	-5.551115e-18
-6.596807e-02	-6.596807e-02
-1.125329e-02	-1.125329e-02
4.468071e-03	4.468071e-03
7.753289e-03	7.753289e-03
3.000000e-03	3.000000e-03
7.753289e-03	7.753289e-03
4.468071e-03	4.468071e-03
-1.125329e-02	-1.125329e-02
-6.596807e-02	-6.596807e-02

Table 17: Fourier Coefficients of Gyroscope Data

Unfiltered IMU	**Unfiltered Output**
-1.421085e-15	-1.421085e-15
1.004744e+01	1.004744e+01
-7.387889e+00	-7.387889e+00
-3.373440e+00	-3.373440e+00
-3.781111e+00	-3.781111e+00
-4.628000e+00	-4.628000e+00
-3.781111e+00	-3.781111e+00
-3.373440e+00	-3.373440e+00
-7.387889e+00	-7.387889e+00
1.004744e+01	1.004744e+01

From both the plots and tables, we can see that the Fourier Coefficients of the original data and the decompressed-decrypted output data are identical. This confirms that the algorithms are lossless and 100% of coefficients have been preserved. Thus, a full cycle of the system is run successfully and AT09 is passed.

5.5.3 Full System Efficiency

After running the system several times for cache warming, the following results were obtained from timing the implementation of the encryption and compression algorithms using the HAL_GetTick function:

Table 18: Full System Efficiency

Test	**No. of Readings**	**No. of Characters**	**Ticks Taken**
1	1	36	6
2	5	179	32
3	10	360	65
4	15	545	101
5	20	730	140

This data is plotted in *Figure 22* below. The individual efficiency trends are also included (and further discussed in the individual subsystem results). The individual trends are included for analysis.

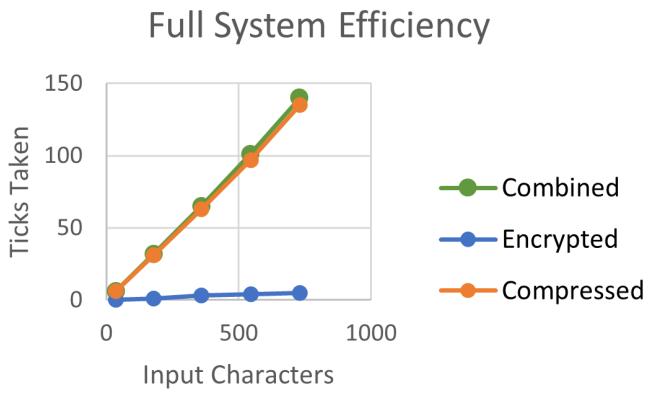

Figure 22: Plot of full system efficiency data

Figure 22 shows that overall, encryption is more efficient than compression. This is clearly indicated by the flatter gradient of the encryption trend line. As the size of data increases, the compression increases proportionately. Since compression has a stronger and more weighted trend than encryption, it has the greatest effect on the efficiency of the system. This shows that compression is a bottleneck in the system design.

5.5.4 IMU: Interfacing

AT08 PASS: data passed from the sensor to the STM32F0 and then to the computer via UART.

Accelerometer and gyroscope data was read from the sensor and passed to the STM32F0 via SPI interfacing. This meet SP06: SPI must be used to interface with the ICM-20649 and ICM-20948. The data was then transmitted to the PC over UART. The code used for this interfacing is available here. This code transmits data every 100ms from the sensor to the PC and the readings can be read using a serial terminal.

IMU Validation testing
In order to validate the IMU module is working, the following tests were run:

Table 19: IMU testing

IMU Test	Description	Success
IT01	Test that the startup sequence runs correctly	PASS
IT02	Test that the direction of acceleration is recorded correctly	PASS
IT03	Test that the direction of angular velocity is recorded correctly	PASS
IT04	Test that magnitude of readings change as expected	PASS
IT05	Test that **all** relevant readings change if the sensor is moved in multiple axes	PASS

5.5.5 IMU: Initialisation

IT01 PASS: Test that the IMU startup sequence runs correctly

The project is successfully updated to transmit the "IMU initialised" message. *Figure 23* shows the message received at the computer after startup followed by the first reading from the sensor.

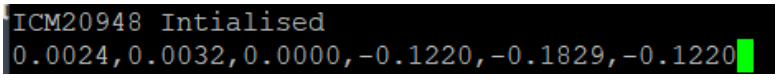

Figure 23: Diagram of Inter-Subsystem interactions

5.5.6 IMU: Accelerometer Verification

IT02 PASS: Test that the direction of acceleration is correctly recorded

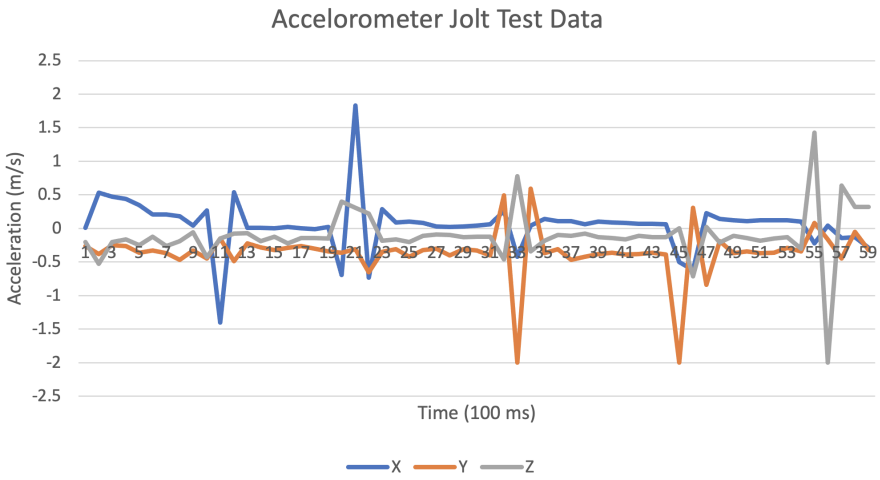

Figure 24: Plot of accelerometer jolt test data

The raw data used to plot *Figure 24* can be found here. Looking at *Figure 24* it is clear that the jolts are easily recorded by the sensor. Sharp movements in the positive and negative x, y and z directions are clearly shown by the spikes in the plot. Ultimately, the accelerometer is able to read acceleration.

5.5.7 IMU: Gyroscope Verification

IT03 PASS: Test that the direction of angular velocity is correctly recorded

The raw data used to plot *Figure 25* and *Figure 26* can be found at: jolt and roll.

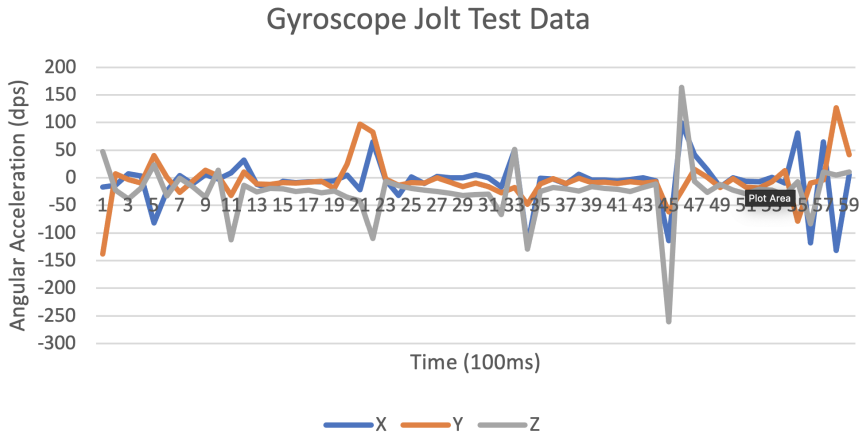

Figure 25: Plot of gyroscope jolt test data

Comparing the above data to *Figure 24* shows that the gyroscope does not show acceleration in one direction like what the accelerometer does. This is expected as the gyroscope measures angular acceleration. The spikes displayed show acceleration in more than one direction (angular acceleration). Looking at *Figure 24* again, the points where there are spikes in multiple axes correlates to the spikes seen in the gyroscope plot. The spikes at

the end of the plot are likely due to noise.

A more suitable test for the gyroscope is the roll test discussed in the experiment setup. The results are plotted below: *Figure 26* below:

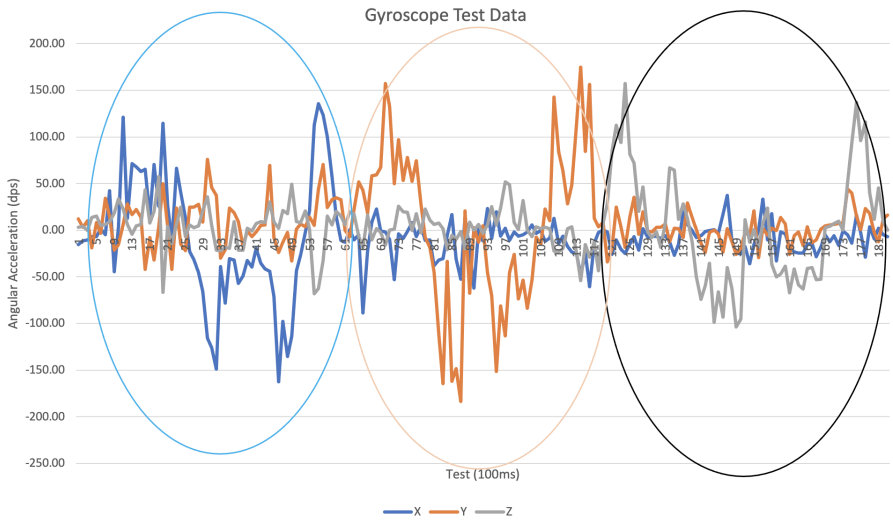

Figure 26: Plot of gyroscope roll test data

Looking at *Figure 26* it is clear where the sensor has been accelerated about a specific axes. The blue oval shows where the sensor has been rotated first anti-clockwise and then clockwise about the x axis. Focusing on the blue line, when the sensor is rotated anti-clockwise the general trend is decreasing and then on clockwise rotation the general trend is increasing. This is the expected outcome. Similar trends are seen in both the y and z axes as expected.

For each axes rotation there are also spikes in the other axes but these are not as significant. The simultaneous spikes in the other axes is due to noise caused by human motion. When rotating the sensor about a specific axes was difficult to keep it stable in all other axes which lead to the noise in the other axes.

Ultimately, the gyroscope is able to read angular acceleration.

5.5.8 IMU: Pendulum Test

IT04 PASS: Test that magnitude of readings increase and decrease

The following data was obtained from the pendulum test. More readings were obtained but only 30 are included here as they sufficiently represent the data. The full raw data can be found here. Using the cleaned wave test data, *Figure 27* has been plotted.

Looking at *Figure 27*, the recorded sensor data shows clear oscillations as expected. The acceleration plot shows oscillations with decreasing amplitudes. This is expected since a pendulum moves back and forth before coming to a stop. The pendulum was swung in the x-y plane which is shown by the strong oscillation trend in the y line. However, whilst running the experiment, the pendulum did not swing perfectly in the x-y plane which resulted in the oscillations in the z axis.

The gyroscope plot shows angular acceleration oscillations in all axes. The pendulum swings in multiple planes (x-y,y-z,x-z) throughout its movement. Over time, as the pendulum slows down, the amplitude of the oscillations decrease. This is as expected for a pendulum.

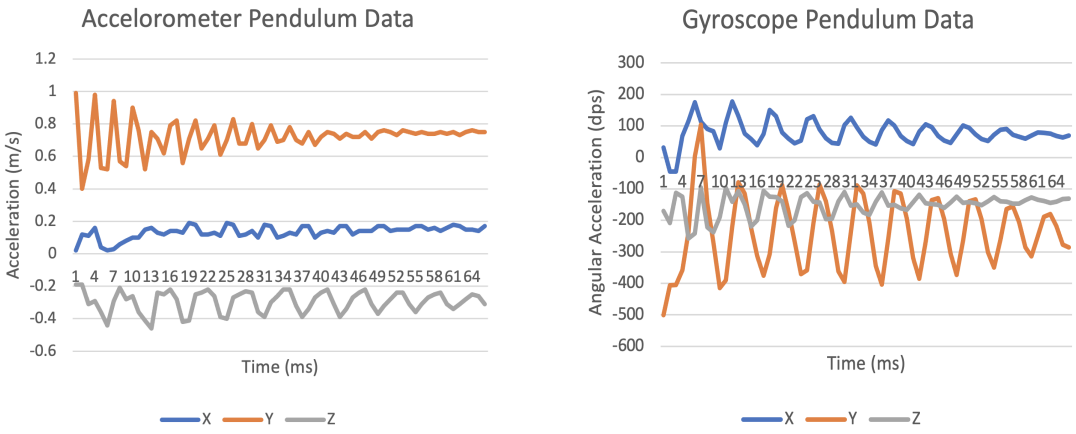

Figure 27: Time Domain representations of Wave Test Dataset

Ultimately, the accelerometer and gyroscope can take increasing and decreasing measurements relative to motion of the sensor.

5.5.9 IMU: Wave Test

IT05 PASS: Test that all relevant readings can change simultaneously

The following data was obtained from the wvae test. More readings were obtained but only 30 are included here as they sufficiently represent the data. The full raw cleaned data can be found here. Using the cleaned wave test data, *Figure 28* has been plotted.

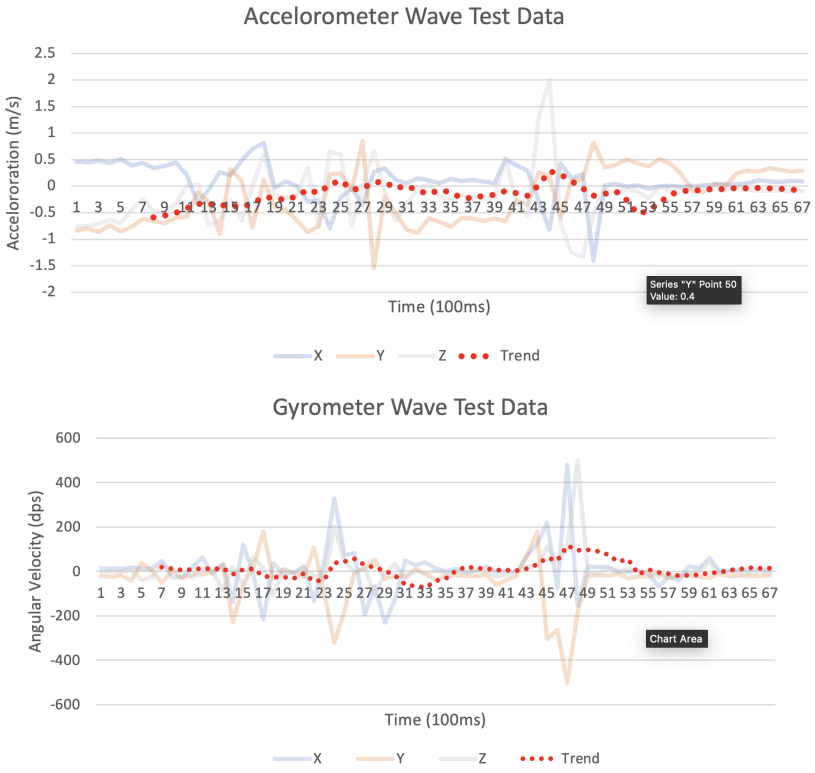

Figure 28: Plot of accelerometer and gyroscope pendulum test data

Looking at *Figure 28*, the recorded sensor data shows clear wave like trends as expected. Weighted average trend lines were added to both of the plots to show the general trend in the data. The motion of the wave causes acceleration in all three axes at the peak of the swell. From the peak, there is a gentle deceleration and then acceleration into the next swell. The trend lines are not as prominent as expected but this is likely due to the method used not being able to imitate wave swells well.

However, the general trends are there. The gyroscope shows the rise and fall to and from the peak of the swell. The peak of the swell is indicated by sharp spikes in all of the axes. Acceleration does not show the wave swells as clearly but rather indicates the general idea that there is acceleration in multiple directions at all times throughout the swell.

Ultimately, the sensor is able to sense slow changes in movement and still indicate the peaks of these slow changes.

5.5.10 Compression: Efficiency Testing

Meets FR04: System must use minimal computation and processing The following results were obtained from timing the compression algorithm using the HAL_GetTick function:

Table 20: Compression Efficiency

Test	No. of Readings	No. of Characters	Ticks Taken
1	1	36	6
2	5	179	31
3	10	360	63
4	15	545	97
5	20	730	135

This data is plotted: *Figure 29* below:

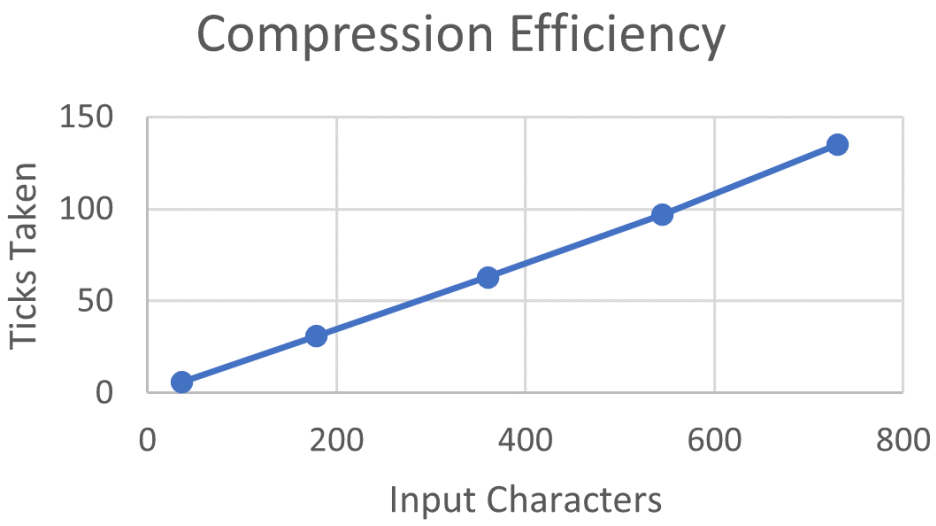

Figure 29: Plot of compression efficiency data

The time taken for compression increases linearly with the input data size. This trend is expected and shows clear efficiency in the data - an exponential trend would indicate inefficiency. Thus, when transmitting data, a trade-off must be made between data-size and efficiency to best minimize both cost and power usage.

5.5.11 Encryption: Efficiency Testing

Meets FR04: System must use minimal computation and processing The following results were obtained from timing the encryption algorithm using the HAL_GetTick function:

Table 21: Encryption Efficiecy

Test	No. of Readings	No. of Characters	Ticks Taken
1	1	36	0
2	5	179	1
3	10	360	3
4	15	545	4
5	20	730	5

This data is plotted: *Figure 30* below:

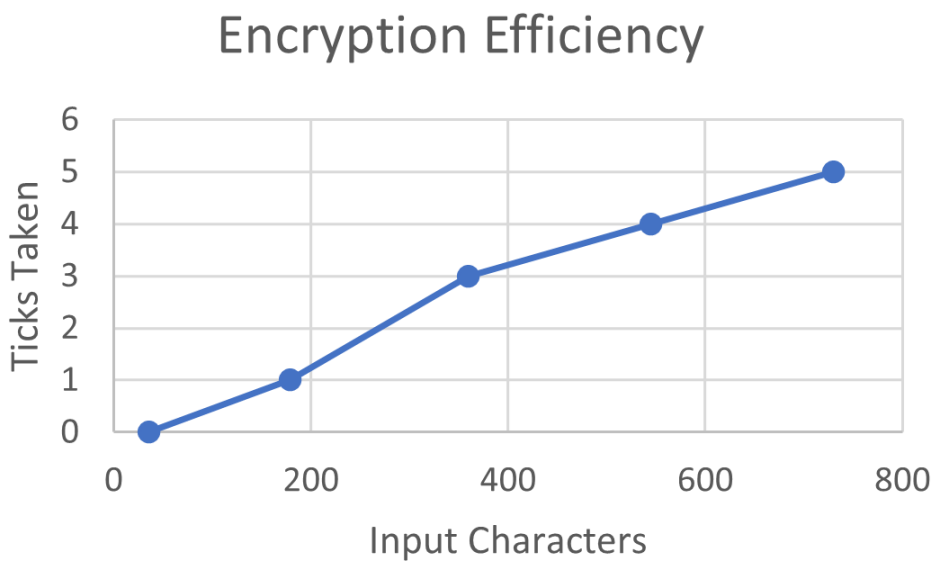

Figure 30: Plot of encryption efficiency data

The time taken for data to be encrypted increases linearly with input data size. This is expected and indicates that the algorithm is efficient - a linear trend is preferred to an exponential trend which would indicate inefficiency.

6 Acceptance Test Procedures

Success and Failure tests:

Table 22: 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	PASS
AT07	Full System	PASS
AT08	IMU	PASS
AT09	Full system	PASS

Table 23: 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	Ensure compression and encryption work when implemented together on computer
AT07	Full System (no HAT)	Ensure compression and encryption work when implemented together on STM32F0
AT08	IMU sensor	Ensure that data is read from sensor and sent to computer
AT09	Full system	Ensure that data read from the sensor completes a full cycle meeting all requirements and above ATPs

6.1 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

6.2 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.

6.3 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.

6.4 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.

6.5 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.

6.6 AT06

AT06	**System Test**
Evaluation type	System execution
Target	Full System (no HAT and tested on computer)
Test Protocol	1. Set up test case using sample data set and compute 25% of the lower Fourier coefficients. 2. Pass sample data through encryption which must then pass onto compression algorithm without user 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	• Encryption and compression and decryption and decompression complete without failure. • AT01 is met.
Fail condition	• Encrypted data is not passed to compression algorithm without human interaction. • Any of the algorithms fail. • Less than 25% of coefficients are retained. • AT01 is not met.

6.7 AT07

AT07	**System Test**
Evaluation type	System execution
Target	Full System (no HAT)
Test Protocol	1. Set up test case using sample data set and compute the Fourier coefficients. 2. Pass sample data through encryption and then compression algorithms on the STM32F051. 3. Perform decompression and then decryption on the computer. 4. Compute Fourier coefficients of the data. 5. Compare the coefficients to the test case and ensure that at least 25% of the coefficients are retained.
Pass Condition	• Encryption and compression and decryption and decompression complete without failure. • AT01 is met.
Fail condition	• Any of the algorithms fail. • Less than 25% of coefficients are retained. • AT01 is not met.

6.8 AT08

AT08	**Reading data from IMU sensor on STM32F0**
Evaluation type	System Execution
Target	Full System
Test Protocol	1. Data is read from the IMU sensor and sent to a computer by the STM32F051 2. Data from the IMU sensor is formatted. 3. Formatted data is sent from STM32F0 to computer.
Pass Condition	• Data is successfully read from the sensor and printed to the computer.
Fail condition	• Data is not read from the sensor (there is no change in the data or no data is printed to the computer) • Data is not transmitted correctly when formatted

6.9 AT09

AT09	**System Test with IMU sensor**
Evaluation type	System Execution
Target	Full System
Test Protocol	The IMU is connected to the STM32F051. The data from the sensor 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 Fourier coefficients are computed. 2. Data is sent through encryption and compression on the STM32F051 and then transferred to the computer. 3. Perform decompression and then decryption on the computer. 4. Compute 25% of lower Fourier coefficients of the data. 5. Compare the coefficients to the test case.
Pass Condition	• At least 25% of Fourier coefficients are retained after full system processing.
Fail condition	• AT06 is not met • AT07 is not met • AT08 is not met

7 Future Plans

The system must be tested using more data readings and with multiple separate transmissions. The discovery board used for system development and testing only has 8 kB of RAM. This limited testing to 8 readings from the IMU at a time. Before putting the combined algorithms on the discovery board, significant tests were done using various data sizes. However, the assumption that the system will work as specified with large amounts of data from the IMU cannot be made as simulation-based validation does not always replicate the real-world.

At the time of this project, a micro-controller with floating point architecture was not available. Due to this, the entire system has been developed and tested on a 32-bit fixed point processor. Whilst the system should be compatible and work as intended, that assumption should not be made. The system should be tested on a floating point processor to ensure compatibility.

As the ICM-20946 was not available, the ICM-20948 was used during the development and testing of this system. These two sensors have several differences as discussed under Salient Features of the ICM-20948 compared to ICM-20649 in the *subsubsection 3.3.2*. While care was taken to ensure that only features available on both were used to develop this solution, this cannot guarantee that the system will be be fully compatible with the ICM-20946. Thus, the full system should be extensively tested with the ICM-20946 before deployment.

8 Conclusion

8.1 System Design

The final system design implements RSA encryption and LZSS compression on the STM32F0Discovery. Data is read from the ICM-20849 using SPI. This data is then encrypted and compressed onboard the STM and sent to a computer via UART. Although compression relies on patterns, the encryption outputs numerical data with makes this system design effective. Once received at the computer, the bits can be decompressed and decrypted to retrieve the original sensor data. The algorithms are lossless and thus, all data is recovered.

8.2 Testing

8.2.1 Simulation-Based Validation

Simulation-Based validation was run on a computer using simulated data provided by Professor Amit Mishra. This data is available under Testing on our GitHub. This phase of testing was invaluable to the design of the system as it provided opportunity to find points of failure and hidden bugs in the system. This was a necessary step before interfacing with the sensor and saved time in later testing phases.

8.2.2 Hardware-Based Validation

Hardware-based Validation was completed at a later stage in the design process and allowed us to confirm that our system works to the project specifications. After successfully interfacing with the IMU, the IMU was linked to the combined algorithms. All 3 modules run without user interaction, additionally data sampled and delays in readings can be specified. Following several stages of testing, the original data can be recovered using the decompression and decryption scripts on a computer. This ultimately validates the designed system.

8.3 Final Remarks

In conclusion, the system designed in this report provides a suitable working solution to the problem statement. Innovative and extensive testing has been conducted to validate the success of the project in terms of the specifications and requirements outlined in this report. Following further testing using the bouy hardware, the system is a viable potential candidate for use onboard the SHARC ice buoys.

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] 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.

[3] 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].

[4] 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].

[5] 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

[6] 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].

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

[8] SparkFun, "SparkFun 9DoF IMU Breakout - ICM-20948 ," [Online]. Available: https://www.sparkfun.com/products/15335, [Accessed: 17 August 2022].

[9] A.-L. Boulesteix, R. H. Groenwold, M. Abrahamowicz, H. Binder, M. Briel, R. Hornung, T. P. Morris, J. Rahnenführer, and W. Sauerbrei, "Introduction to statistical simulations in health research," *BMJ Open*, vol. 10, no. 12, 2020. [Online]. Available: https://bmjopen.bmj.com/content/10/12/e039921

9 Appendix A: Documented Changes since Paper Design

9.1 Changes made to Requirements

Table 24: The old and current traceability matrix side by side for ease of viewing

Old Requirement Traceability Matrix

UR	FR	Specs	ATP
UR01	FR01	SP01	AT02
UR02	FR06	SP07	AT07
UR03	FR05	SP04, SP06	
UR04	FR03, FR04	SP05	AT04, AT05
UR05	FR01, FR02	SP02	AT03
UR06	FR07	SP08	AT01

Requirement Traceability Matrix

UR	FR	Specs	ATP
UR01	FR01	SP01	AT02
UR02	FR05	SP06	AT09
UR03	FR04	SP05	
UR04	FR03	SP03, SP04	AT04, AT05
UR05	FR01, FR02	SP02	AT03
UR06	FR06	SP07	AT01

Table 25: Old Functional Requirements addressing the needs of the system

Requirement	Description
FR01	System to compress data such that 25% of the 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

The following functional requirements have been updated in *Table 3*:

FR02 has been updated to reflect the system redesign (encryption happens before compression). The new requirement is: encryption subsystem to pass processed data to the compression.

FR03 has been updated to reflect system redesign, encryption happens before compression. The new requirement is that all the measured data must be encrypted.

FR04 has been removed as it is taken into account in FR05.

FR05 has been updated to add further explanation on the requirement. It is now: System must use minimal computation and processing, algorithms should be efficient.

FR05 through FR07 have been renamed in accordance with the changes named above: FR05 is now FR04; FR06 is now FR05 and FR07 is now FR06.

Specifications

Table 26: 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 20mAh of power must be used in encrypting the data.
SP05	Less than 10% of the original data can be the same as the original file after encryption.
SP06	All code should be written in C
SP07	I2C must be run on fast mode when developing and testing (ICM-20649 standard)
SP08	System must transfer the data over UART following encryption

The following specifications have been updated in *Table 4*:

SP02 has been updated to use the term data instead of files since the STM32F0discovery does not use files. SP03 has been updated to reflect the system redesign (encryption happens before compression). The new specification is: 100% of the data must be recovered after decryption.

SP04 has been removed as this cannot be measured accurately.

SP07 has been changed since the system design no longer uses I2C but rather uses SPI, The new specification is: SPI must be used to interface with the ICM-20649

SP08 has been updated to reflect the system redesign, the data should be transmitted following compression and not encryption.

SP05 through SP08 have been renamed in accordance with the changes named above: SP05 is now SP04; SP06 is now SP07; SP07 is now SP06 and SP08 is now SP07.

9.2 Feasibility Analysis

Table 27: Feasibility Analysis

Specification ID	Level of Feasibility	Reasoning
SP01	HIGH	There are many lossless and low loss compression algorithms that exist and could be used
SP02	HIGH	There are many compression algorithms that can compress files much smaller than 30% of the original size
SP03	HIGH	Provided the key is correct, 100% of the decrypted data should match the original data
SP04	HIGH	Provided the encryption algorithm is implemented correctly, very little of the original data should match the cipher text in the encrypted file
SP05	HIGH	Unless there are unforseen circumstances that require otherwise, all code will be written in C
SP06	HIGH	Unless there are unforseen circumstances that require otherwise, I2C will be run on fast mode when developing and testing
SP07	MED	While this is a key specification, it is unclear whether we will have access to an FTDI or another method of transferring encrypted data over UART