PROJECT REPORT

Table of Contents

INTRODUCTION

An ARM based digital device, STM32 microcontroller, is used to compress, encrypt, and transmit all relevant data from an inertial measurement unit (IMU) for further analysis of the environments surrounding the Antarctic region. This STM32 microcontroller design system is one of many subsystems that make up the SHARC BUOY designed by Jamie Jacobson.

Over the course of the project we used two types of data , one set of data provided by the teaching assistant which we used for data simulations and the second was by using the IMU we were provided (ICM-20948) which was attached to the STM32f051T6R8 discovery board and used to fetch data which was outputted to a buffer for later compression and encryption .This was to test the effect of sampling rate on the noise seen and to identify whether the Sense HAT B was running as expected (with regards to accelerometer and gyroscope data).

Various methods of compression and encryption are compared to determine the most suitable algorithm for this application. First a theoretical comparison of the algorithms was made and then some were chosen for testing in implementation. These were then compared on multiple levels (such as speed and compression ratio) to determine the most suitable algorithm for the type of data outputted by the IMU.

The compression algorithms tested were the lz4, xz, zlib , zstd and huffman. For this set of experiments that were conducted the zlib compression technique was used as the golden measure. These experiments consisted of the compression of files with different sizes (measured in bytes) using the different compression techniques, the measuring of the execution time of the various compression algorithms and comparing the decompressed file to the original file for every technique. The first experiment was used to determine the various compression ratio and percentage compression of each algorithm. While the second experiment tested the speed-up ratio of each algorithm, the last experiment determined whether each technique was lossless or lossy. All these results were fundamental to determining the technique used for the designed system and whether they met the various subsystems' Acceptance Test Procedures (ATPs).

The encryption algorithms tested were BlowFish, Data Encryption Standard (DES) and Advanced Encryption Standard (AES). The BlowFish was used as a golden measure due to it being seen as one of the most primitive types of encryption. These algorithms were tested on the following factors: execution time (to be used to display battery usage), the security level of encryption, whether the encryption is lossless and the RAM usage of the system on the STM32 microcontroller. These factors were chosen to encompass the needs of the relative Acceptance Test Procedure (ATP) and to test the various algorithms for their suitability in this project.

Additionally, the user requirements, functional requirements and specifications of the design are defined with a list of acceptance test procedures highlighted, in which the methods are tested against the design specifications. Finally, the compression and encryption blocks are combined

and tested for correct execution prior to the full implementation of the system on the STM32 microcontroller

REQUIREMENTS ANALYSIS

Interpretation of the requirements

The user requirements were derived from the design project scope as well as the thesis on the SHARC BUOY's. Each of these requirements will be expanded into functional requirements and hence design specifications in the sections to follow. They are listed in no particular order.

User Requirement ID	Description
UR001	A motion tracking device must be used.
UR002	User requires the lowest 25% of the Fourier Coefficients as usable.
UR003	The data must be encrypted
UR004	The data must be compressed.
UR005	Minimize power usage and maximize battery life longevity.
UR006	Data must be transferred as soon as processing is done.
UR007	The design should be adaptable depending on the Antarctic environment that it is in.
UR008	System must use an on-board ARM based processor to compress and encrypt the data

Table 1: Interpretation of the User Requirements and their Description

Analysis of User Requirements

- UR001: A motion tracking device must be used.
    - System must make use of the ICM-20649
    - The ICM-20649 is a wide-range 6-axis motion tracking device for sports and other high impact applications. It has a gyroscope and accelerometer which are used to get information about the ice and wave dynamics.
      Features
    - 3-Axis gyroscope with programmable FSR of ±500 dps, ±100 dps, ±2000 dps, and ±4000 dps
    - 3-Axis accelerometer with programmable FSR of ±4g, ±8g, ±16g, and ±30g

- UR002: User requires the lowest 25% of the Fourier Coefficients as usable
    - Compressing a file reduces data size and enables the file to be sent and received quickly. This allows us to get the best out of the processing power and storage space. However, there can be a trade-off of data loss when decompressing the compressed file. The chosen compression algorithm should be able to extract at least 25% of the lower fourier coefficients of the data.
- UR003: The data must be encrypted.
    - A suitable encryption algorithm must be used to ensure security of data during transmission.
- UR004: The data must be compressed.
    - A suitable compression algorithm must be used to reduce the size of data being transferred so increase as to ensure increased data transfer speed
- UR005: Minimize the power usage and maximize battery life longevity
    - Power of the on-board processor is limited thus the need to minimize the amount of computation done by the system. The chosen compression and encryption algorithms should be efficient such that they do not require high processing power.
    - Minimal sampling should also be done to minimize the amount of computation done by the system thereby reducing power usage.
- UR006: Data must be transferred as soon as processing is done
    - Due to the limitation of storage space on a stm32 discovery board the data should be transmitted as soon as processing is done , to clear storage space for the next process.
    - Minimal sampling should be done to reduce the amount of data collected so as to overburden the limited storage capacity.
- UR007: The design should be adaptable depending on the Antarctic environment that it is in.
    - System must make use of the ICM-20649
    - The Marinebio.org(2021), report that the temperatures recorded at the Southern Ocean ranges from −2 0 C to 100 C. The ICM-20649 has an operating temperature range of −400 C to 850 C making it suitable for this application
- UR008: System must use an on-board ARM based processor to compress and encrypt

the data

- A STM32 discovery board must be used for testing implementation.
- The on-board ARM based processor that is going to be used is a STM32f051T6R8 discovery board. The IMU will be connected to the STM32 microcontroller which will be used to read data from the IMU. A compression and encryption code will be loaded on the microcontroller which will be used to compress then encrypt the read data from the IMU before transmitting it
- The board is cheap , easy to program and STM32 boards have a wide online community which helps in the know-how of how to use them.

System Requirement(s) and Specification(s)

Syst Req ID	Functional Requirement(s)	Syst Spec ID	Specifications
SR001	The system shall be able to read raw data from the IMU.	SS001	The STM32 microcontroller will communicate with the IMU using the I2C interface in order to receive the data
SR002	The algorithm shall be able to compress the raw data from IMU	SS002	We shall use the huffman-algorithm to reduce the file size containing data from the IMU.
SR003	Oceanographers have indicated that they would like to be able to extract at least 25% of the Fourier coefficients of the data	SS003	The compression algorithm should account for at least 25% of the lower fourier coefficients if there is data loss at decompression. Losses due to compression should be reduced such that at least 25% of the data can be extracted at decompression
SR003	Oceanographers have indicated that they would like to be able to extract at least 25% of the Fourier coefficients of the data	SS004	Losses due to encryption should be reduced to minimum
SR004	The subsystem must be able to encrypt the data from the IMU	SS005	AES encryption algorithm will be used.
SR005	The encryption subsystem shall be able to receive data from the compression subsystem	SS006	Use buffers to handle data being sent between the subsystems
SR006	Minimize power usage and maximize battery life longevity.	SS007	Sampling rate should be reduced to a minimum

SR006	Minimize power usage and maximize battery life longevity.	SS008	Compression time should be reduced to a minimum
SR006	Minimize power usage and maximize battery life longevity.	SS009	Encryption time should be reduced to a minimum

Table 2: Derived Functional Requirements and Specifications

Acceptance Test Procedure

Syst Spec ID	Specifications	Figure of merits
SS001	The STM32 micro controller will communicate with the IMU using the I2C interface in order to receive the data	Print out data from the IMU to prove retrieval
SS002	We shall use the huffman-algorithm to reduce the file size containing data from the IMU.	The file containing compressed data should be smaller than the one containing the data from the IMU (raw data)
SS003	The compression algorithm should account for at least 25% of the lower fourier coefficients if there is data loss at decompression. Losses due to compression should be reduced such that at least 25% of the data can be extracted at decompression	Comparing the compressed & decompressed data and raw data should show minimum or no losses
SS004	Losses due to encryption should be reduced to minimum	Data integrity should be maintained after encryption and decryption.
SS005	AES encryption algorithm will be used.	Encrypted data should be unreadable and different from the non-encrypted data
SS007	Sampling rate should be reduced to a minimum	Timers can be used to adjust the sampling rate e.g HAL_delay() in STM32CUBEIDE

Table 3: ATP's for the derived specifications

PAPER DESIGN

Comparison of some available compression and encryption algorithms

Comparison of Compression Algorithms

There are different compression available over the internet designed by different companies used for the compression data for various purposes mainly for fast transmission over networks. The key distinguishing features used by different individuals in choosing a compression algorithm which meets their needs are by comparing their compression ratios (ratio between uncompressed data and compression data) , compression speed and decompression speed or by the losses sustained by each algorithm though most are lossless. The table below shows comparisons among some of the most famous compression algorithms.

Compression Algorithm	Ratio	Compression Speed MB/s	Decompression Speed MB/s
lz4	2.10	444.69	2165.93
zstd	3.14	136.18	536.36
zlib	3.11	23.21	281.52
xz	4.31	2.37	62.97

Table 4: Comparison of Compression Algorithms[1]

As indicated in **Table 2**, there is a compromise between the compression speed and compression ratio. lz4 with the fastest speed has a lower compression ratio and xz with the largest compression ratio has the slowest speed. Zstd however has a good balance between speed and compression ratio, therefore, making it a more favourable algorithm but we chose to use a more primitive Huffman compression algorithm because it is efficient for text based files , lossless and also because it is easier for us to modify it suit needs for this project. Most famous compression application software like gzip, pkzip (winzip) and bzip2 use the huffman algorithm as base and are based upon it.

Comparison of Encryption Algorithms

Encryption Algorithm	Processing Speed (MB/s)	Time required to crack all possible keys (50 billion keys per seconds)	Security Level

AES-128	61.010	5 x 10^{21} days	Extremely High
BlowFish	64.386	3200 days	Moderate
3DES	20.783	800 days	Not secure enough

Table 5 : Comparison of Encryption Algorithms [5]

As indicated in **Table 3**, there is a compromise between the encryption speed and the level of security with AES-128 having the highest level of security needing a total of 5 x 10 21 days to crack all possible security keys, however , with a second-best encryption speed performance. With 3DES performing the worst, this leaves the BlowFish encryption offering the best encryption speed with a moderate security level(3200 days needed to crack all possible security keys). Since the project requirement did not specify the level of security needed, Blowfish and AES would be the suitable choices as they both have incredible processing speeds, minimizing the number of computations done by the processor,however AES has the highest rated security level performance which could be beneficial.

Feasibility analysis

Economic Feasibility

The project does not have a budget as most of the hardware requirements (STM32 Microcontroller, PC Laptop) and software requirements (MatLab, C) are available for free. So it is possible to design the system with the current budget.

Operational Feasibility

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

Schedule Feasibility

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

Possible bottlenecks

There are multiple trade-offs that can be noticed from the comparison of algorithms shown above. All the trade-offs involve time in one way or another . The various trade-offs we expect to encounter are listed below:

- **Compression level vs compression time**
    - The more compressed the file (the lower the compression ratio), the longer it takes to run the corresponding compression algorithm .
- **Choice of encryption algorithm vs data needing to be encrypted**
    - Different encryption algorithms have different suitabilities. Some algorithms are better suited to finite length data, while some can handle different length data streams. This choice can affect the time it takes to perform the encryption.
- **Processing time vs power used**
    - The data needs to be pre-processed to pull out only the needed data, but this processing takes time and the more accurately it is done the longer it takes. There is a trade-off between time to process and depth of processing.
- **Data collection vs processing time**
    - The amount of data that is in each transfer packet needs to be decided. How long does the code wait before it has enough data collected from the IMU to effectively process and then transfer. The larger the data collected per packet the longer it will take to process each one and then transmit it. If the wait period is too long, the data that could have been sent may be lost if the buoy only transfers occasionally and is lost before it can transfer a significant amount of data.
- **Optimization of algorithms**
    - Optimizing the algorithms to reduce computation to the minimum might compromise the integrity of the data from the sensor.

- **Time Schedule**
    - The largest limitation to the project is time constraints. The project timeline coincides with the SCALE research cruise using the winter and spring expeditions for buoy deployment thereby limiting the period for development. Additionally, the firmware development is limited to the capabilities of the selected processor.
- **Storage Space**
    - The memory size of STM32 Microcontroller is very limited making it hard for us to store any large or bulky data on it e.g compression and encryption algorithms as well as sampled data. The stm32f051r8t6 board has 64Kbytes Flash memory which is very small.
- **Environment**
    - The Ice buoys are designed to measure environmental conditions in the Antarctic which has very low temperature and humid conditions. Our design system should be able to operate in that kind of environment so it limits what we have to work on as all our system components have to have some form of resistance to cold and also its difficult to simulate that kind of environment here in South Africa as we cannot physically go there to test the system hence there will inaccuracy in our released system.

Subsystem Design

There are three main subsystems that have been identified and are relevant to the main STM32 micro-processor. Each of these systems are listed below along with their respective functional requirements and design specifications. The systems have been listed in the order in which the submodules interface with the external system as well as other subsystems. After the list of subsystem requirements,inter-subsystem interactions and UML diagrams will provide an enhanced visual understanding of how each system and subsystem interface with each other.

Extraction & Storing of Data

This system primarily focuses on the extraction of the data generated by the IMU (a motion tracking device) and storing of it on the STM32 Microcontroller. We are constrained by the memory on the micro controller for the simulations that we will perform.

Functional Requirement ID	Description	User Requirement Addressed
FR001	A 6-axis MEMs motion tracking device IMU is to be used to collect environmental data	UR001
FR002	A buffer is used to store data from the IMU to be processed by other subsystems	UR008
FR003	The sampling rate must not exceed the minimum.	UR005

Table 6: Functional Requirements of the Extraction and Storing of Data

Design Specification ID	Description	Functional Requirement Addressed
DS001	An ICM-20649 motion tracking IMU device is used.	FR001
DS002	The IMU supports I2C communication and these comm ports will be enabled on the STM32 microcontroller.	FR001
DS003	A large enough buffer will be used to store the IMU data for later processing by the subsystems	FR002
DS004	Sampling frequency of 50ms was used on the IMU	

Table 7: Design Specifications of the Extraction and Storing of Data

Compression of Data

The compression of data is done so by using one of the many compression algorithms that are available on the internet.There have been companions made between the various algorithms in which the compression ratio and speed is used to determine the most efficient algorithm for the SHARC BUOY application.The compression of data precedes the encryption of the data for numerous reasons . The most prominent answer is that once data has been encrypted the file generates a random stream of data , which becomes near impossible to compress. Additionally , compression depends on finding a compressible pattern to reduce the over sized data. The SHARC BUOY project requires the collected data, from the buoy, to be compressed and

transmitted through the existing Antarctic transmission infrastructure. This transmission procedure is limited by the satellite network it uses. This network has high data transfer costs, inconsistent transmission reliability, bandwidth and data structure specifications which all therefore limit the aspect of transmission. Therefore, compression of the data will make it cheaper to transfer, as there is less to transfer, and will increase the likelihood of a complete transfer as the transfer can take place in a shorter period.

Functional Requirement ID	Description	User Requirement Addressed
FR005	Both data sent to the microcontroller and received from the micro controller must be compressed using an appropriate algorithm	UR004
FR006	The subsystem shall do minimum computations to save power	UR005
FR007	User requires the lowest 25% of the Fourier Coefficients as usable	U002

Table 8: Functional Requirements of the Compression of Data

Design Specification ID	Description	Functional Requirement Addressed
DS003	We shall use a huffman compression algorithm	FR005
DS004	We shall use a simplified version of a huffman encoder/decoder which is not too complex. It is also fast	FR006
DS005	Huffman algorithm is lossless	FR007

Table 9: Design Specifications of the Compression of Data

Encryption of Data

Encryption is a method of securing information in which only authorized individuals have access to its contents. The method of encryption involves converting the original data into an alternative, unreadable form, and it is only those who are authorized that can decipher the encrypted form using a security key to gain its contents There exist many encryption algorithms with different levels of security and speed of execution, however having a high level of security and fast speed of execution are mutually exclusive. In the case of the SHARC BUOY Project,, the level of security is unspecified however a firm requirement is to reduce the amount of

processing done within the processor, which means the project requirement is more in favor of the speed of encryption than a high level of security.

Functional Requirement ID	Description	User Requirement Addressed
FR008	The IMU's compressed contents to be sent to the encryption subsystem to generate an encrypted file version ready for transmission	UR003
FR009	The subsystem to limit the amount of processing done in the processor and to minimize the number of computations	UR005

Table 10: Functional Requirements of the Encryption of Data

Design Specification ID	Description	Functional Requirement Addressed
DS006	A possible choice of encryption could be the use of the algorithm BlowFish which has a high level of speed and a moderate level of security.	FR008
DS006	BlowFish speed of execution is one of the fastest algorithms available today thus limiting the number of processes happening in the processor	FR009

Table 11: Design Specifications of the Encryption of Data

Inter-Subsystem and Inter-Sub-subsystems Interactions

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

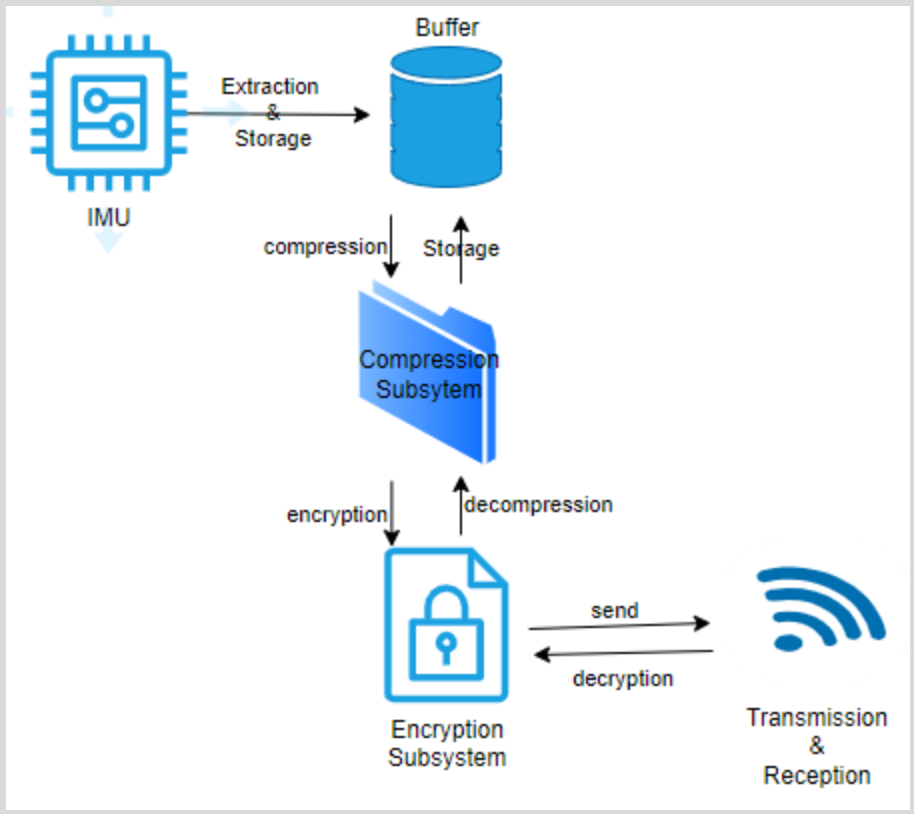

Figure 1: Block diagram showing the Interaction of Different Subsystems

UML Diagram

The use case diagram below focuses on the two main buoy subsystem blocks which are namely the compression and encryption blocks and how they interact with the IMU, its data and the data transmission system.

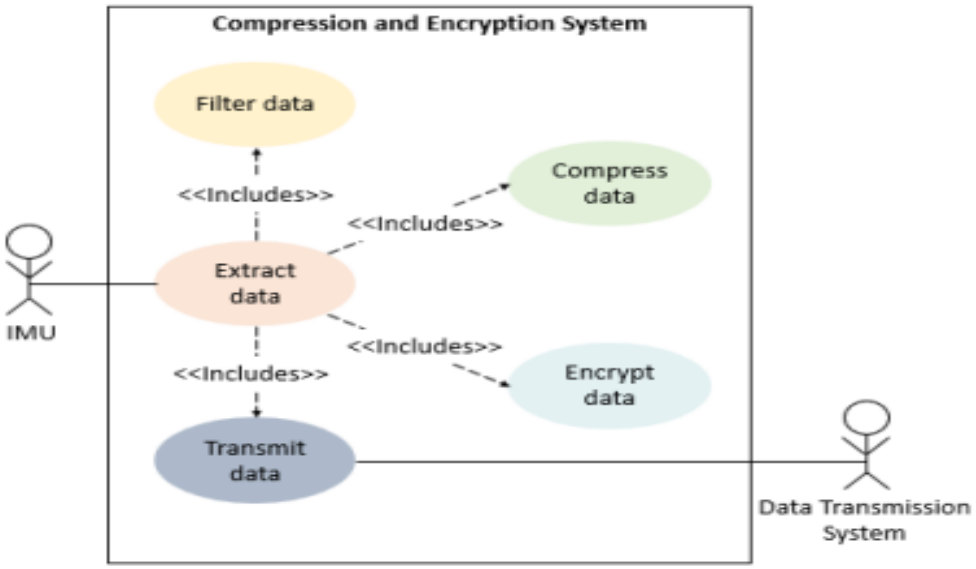

Figure 2: Use case diagram for system

Acceptance Test Procedure

Specification(s) ID	Specification(s)	Figure of Merits
DS001	An ICM-20649 motion tracking IMU device is used.	We can send the data collected to a computer to prove we are collecting the correct information using an ICM-20948
DS003	We shall use the Huffman encoding algorithm to compress our data from the IMU	It is lossless so that we are able to retrieve at least 25% of the Fourier transform coefficients
DS004	We shall use a simplified version of a huffman encoder/decoder which is not too complex. It is also fast	Fast as it is efficient in compressing text or program files
DS002	The microcontroller will communicate with the IMU in order to receive the data & a serial connection between the microcontroller and PC will be established	A serial connection was successfully established between the microcontroller and PC, allowing for data packets to be stored on and received from the microcontroller
DS006	The use AES library to encrypt and decrypt the compressed IMU data and ensure security.	AES-128 was used to successfully encrypt and decrypt the compressed .csv data ensuring 100% security

DS007	Choosing an encryption algorithm that limits the number of processors done on the microcontroller.	The AES library provided by kokke[11] is a simplified version of the general AES library, intended for areas where memory and processing power are constrained. Efficiency testing is yet to commence

Table 12: Figures of Merit of the Overall System

VALIDATION USING SIMULATED DATA

Data Used & Justification

Simulation is a helpful process which allows for entire systems to be tested and validated before expensive and important hardware can be involved. With simulation, one can use input data, similar to that of the real world, which allows the user to reassuringly process the data and determine the effects and response of the developing system. Simulation aids in solving system based issues early in the development cycle. Simulation validation is a necessary step as it allows users to verify if the system meets the requirements before the hardware can be implemented. With system validation, once that system has passed these requirements, extrapolation the system to include real world hardware and components becomes much simpler.

Given two sets of data: the first, a data collection of an IMU placed in the pocket of the user, whilst walking around, sitting and standing for five minutes and the second , a data collection of an IMU placed on a fixed rotation program, rotating about one of its axes for over ten minutes. We have decided on using the first data set with the IMU in a user's pocket (reading accelerometer and gyroscope data) as it contains more complexities (i.e walking,sitting, standing) compared to a fixed rotation device.Though we do not have a dataset that resembles that of an IMU in the arctic (which is where are our project will be adapted), the arctic conditions and complexities are unknown and thus we have decided it would be wise to deal with a data set with more gyroscope and accelerometers complexities such that of an IMU in a pocket. Since the user will be walking and stopping to sit down, the movements may be irregular and may induce rotations about all axes which can be comparable to a degree to the unknown movement of an IMU in the arctic.

This data set contains a large number of entries which are suitable to run our compression and encryption speed ATPs. We will be able to vary the size of the data and use this varying data to perform compression and encryption speed, compression ratio tests and allow us to improve the efficiency of our subsystems accordingly. The data set also contains noise which is expected from our given IMU, allowing us to demonstrate our compression test and determine if at least 25% of non-noisy data may be extracted.

Data Analysis

As the rest of the project's focus is on the compression and encryption of the data set, therefore the data within is not critical if it fits the expected data that the IMU on the buoy will output. The reason for this is as both the compression and encryption methods used are lossless and therefore the specification that the minimum 25% of Fourier coefficients cannot be lost is met

Time Domain

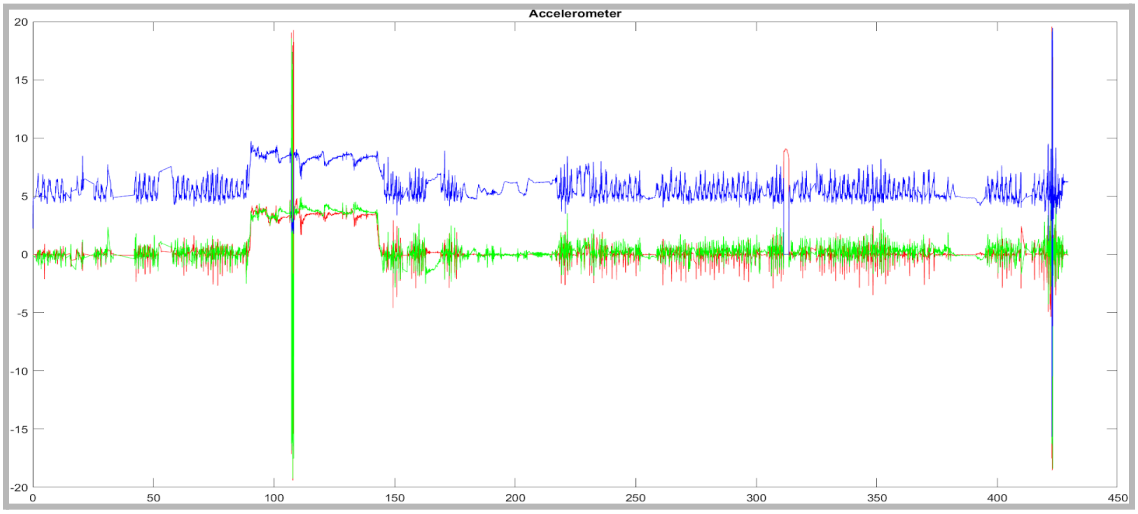

Figure 3: Accelerometer in the time domain with x-axis in red, y-axis in green, z-axis in blue

Below are the graphs of the data set, showcasing the accelerometer and gyroscope data in the time domain. .

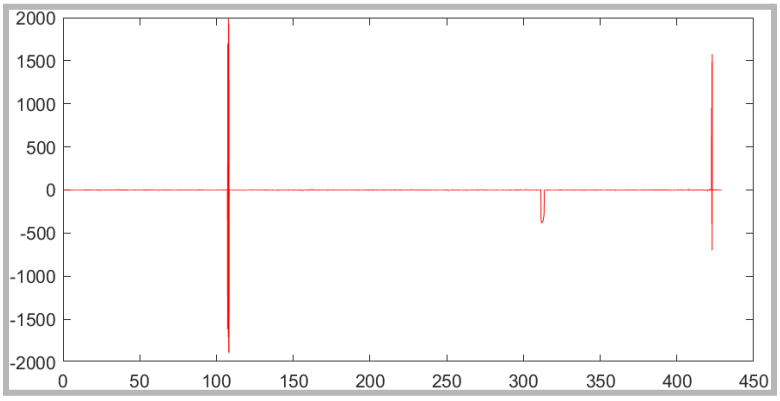

Figure 4: Gyroscope x-axis data in the time domain

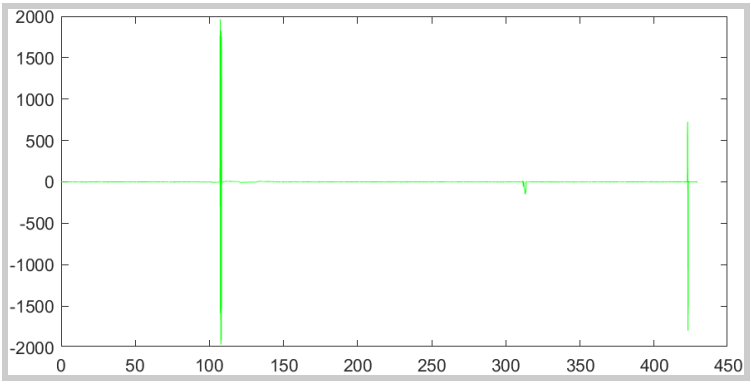

Figure 5: Gyroscope y-axis data in the time domain

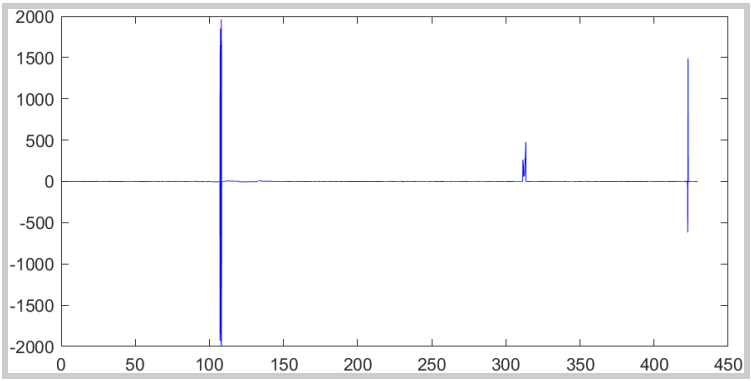

Figure 6: Gyroscope z-axis data in the time domain

Frequency Domain

Below are the graphs of the data set, showcasing the accelerometer and gyroscope data in the frequency domain.

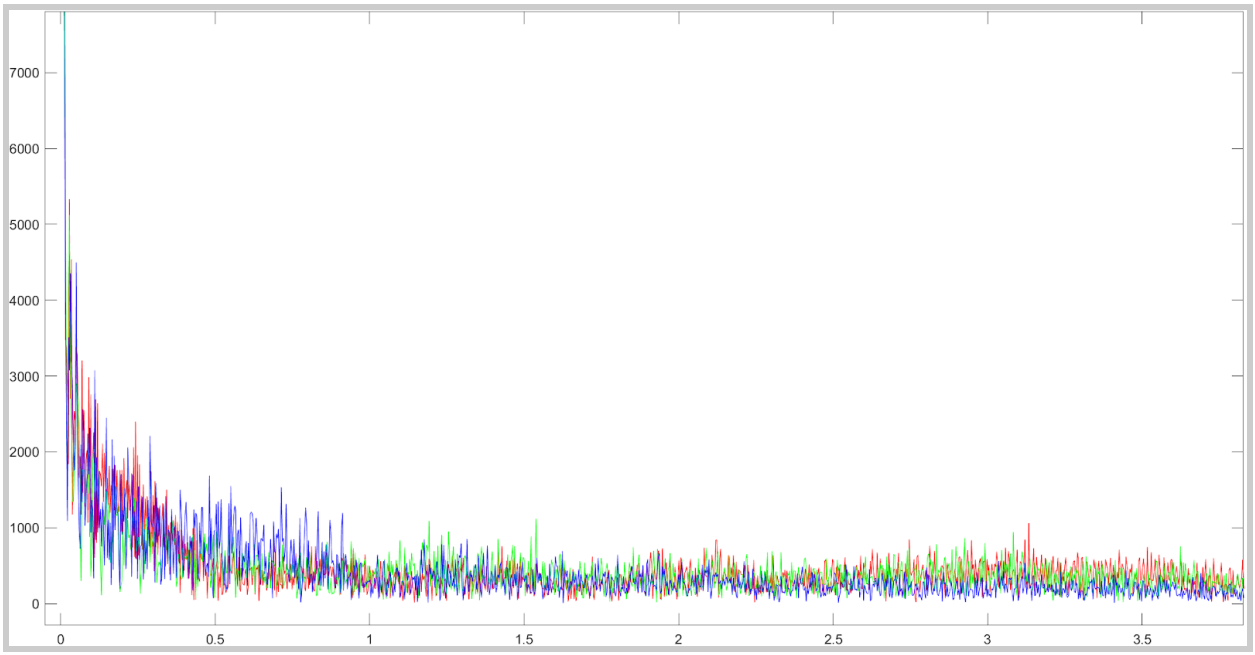

Figure 7: Accelerometer in the frequency domain with x-axis in red, y-axis in green, z-axis in blue

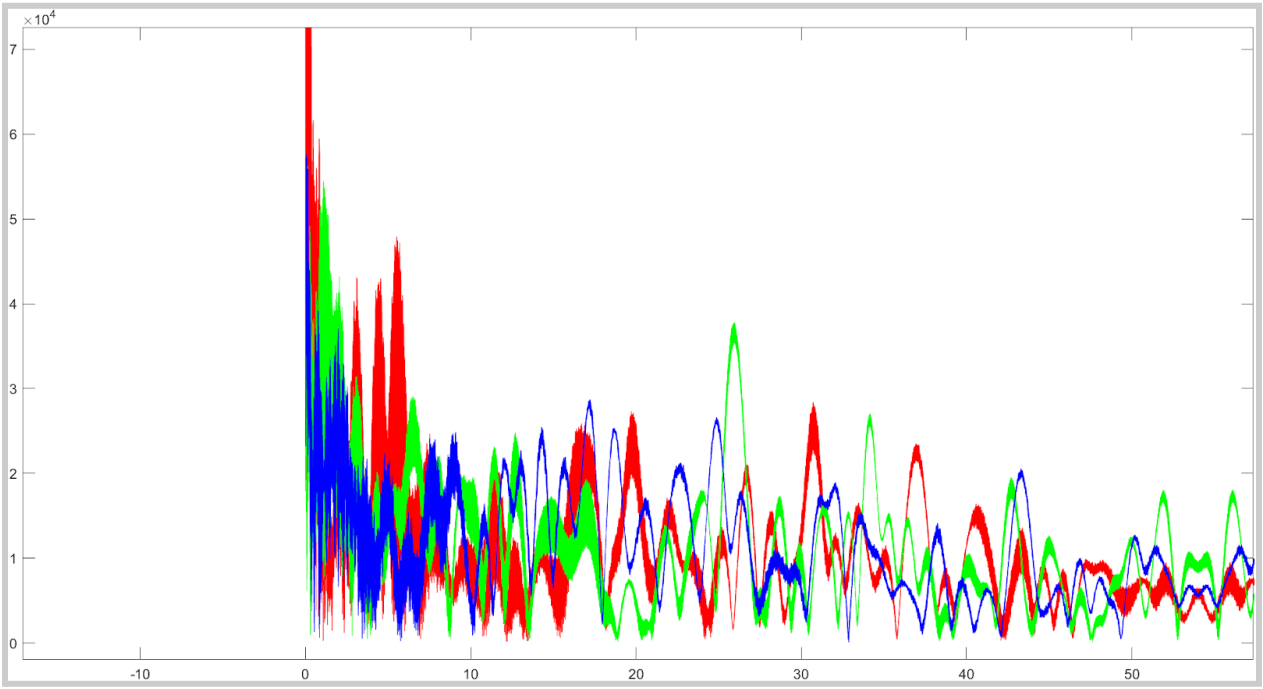

Figure 8: Gyroscope in the frequency domain with x-axis in red, y-axis in green, z-axis in blue

Experiment Setup

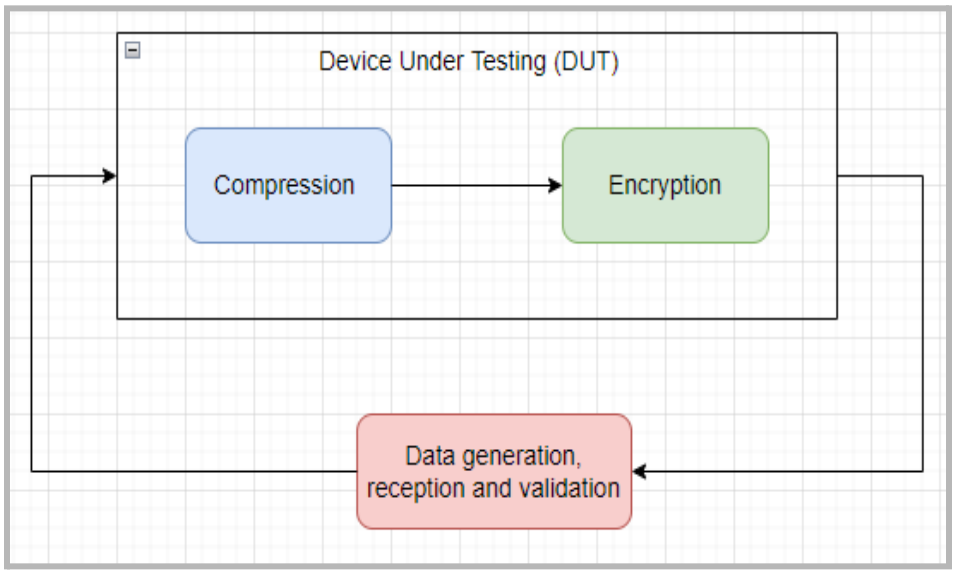

Figure 9: The loop-in test that was used in this experiment

Given an stm32f0 microcontroller and usb-to-uart module, we were able to form a serial connection between our PC and the microcontroller. The serial connection allows for our IMU data to be sent to the microcontroller for compression and encryption processing, as well as to be retrieved. Our IMU data is stored within a .csv file located on a PC however, unlike the raspberry pi's ability to store user files, the micro-controller does not have a file I/O system in place for us to store files, and as a result we have opted to send each line from the .csv file as a string for processing. This method of input will not be a problem as this method will most likely be similar to when we sample sensor data from an attached IMU. Below we will describe how we aim to run our experiments to test our overall system functionality, as well as discuss the experiments set for the compression and encryption submodules.

Overall System Functionality

In order to test the system functionality of the simulated system, we created a subset of data consisting of a few rows of records and tested the compression and encryption subblocks separately and finally collectively. We utilized a program called CoolTerm which allowed us to send string lines of data to our microcontroller as well as capture the data output from our microcontroller to which we can then save as a text file. Using CoolTerm we sent each line of our IMU subset data into our microcontroller buffer via the serial port connection. The buffer allows for the storage of our input lines of data whilst allowing us to process each string of data simultaneously[3].The text file containing the subset file was sent for compression on a pc, to which we then retrieved it's compressed version and sent each string line of data to the microcontroller. Each string line of data was sent to the encryption block where each line will be encoded and transmitted to the serial port where CoolTerm captured and stored the encoded data in a text file for encryption analysis. We then reversed the process, inputting data from the

encrypted back into the microcontroller to retrieve the decrypted data. After decryption, we sent the file for decompression on a pc and compared the results.

Compression Subsystem

This subsystem will have 3 tests i.e compression, decompression and data Integrity. During compression we will read the .csv file data, Huffman encodes it, store the compressed data in an array and then output the data into a .txt file on the PC for analysis.
For decompression it's the reverse. We will read the .txt file data, Huffman decodes it, store the decompressed data in an array and then output the data a .csv on the PC for analysis.
During analysis, we simply compare the file sizes of the input and output files to prove that has occurred. The data Integrity test also takes place during this analysis where we will check if the same data is within the test .csv file and the compressed and decompressed .csv file. The compression and decompression speed was measured. A test was also carried out by compressing the file using different compression algorithms as well as the measuring of the compression time , decompression time and compression ratio

Encryption Subsystem

The subsystem consisted of three experiments, namely the encryption test, decryption test and efficiency test. Within the encryption test, we tested if the input data had indeed been transformed into ciphertext. From CoolTerm we input string lines of data from the compression text file, into the microcontroller where each will be stored in the buffer created for the subblock. The subblock will then process each data stored in the buffer at a time, performing AES-128 encryption to which will then be placed in another buffer for serial data retrieval. Once the data has been captured with CoolTerm and saved in a text file, a comparison was made with the results and that of the original input data text file. On our PC, an online tool or python script was used to compare each line from the input and output text file and to determine if there were similarities. For successful encryption, the online tool needs a result of 0% similarities for each line compared.

Within the decryption test, we reversed the process, using our output encrypted data as input, expecting the compressed data to be retrieved. We used the same online tool to compare the newly decrypted text file and compression text file, and for a successful decryption, the online tool needs a result of 100% similarities for each line compared.

The decryption test also had a security test to determine if the encrypted data was 100% secured. In this test, the security key used to decrypt the data was changed, simulating a malicious user trying to decrypt the data. Once the data was decrypted using this incorrect key, the decrypted data was compared with the compression text file. The online tool would need a result of 0% similarities for each line compared, for the security test to be deemed succesful

Results

Overall System Functionality

The serial connection and communication between our PC and microcontroller was successful. The microcontroller was able to receive data lines from csv and text file subset data as expected and was able to store these contents in the created buffer without any loss of contents. Both the encryption and compression subblocks were able to function individually, and linking the subblocks to perform the compression, encryption,decryption and decompression steps collectively was successful.After the encryption process and the separate decompression process, data from the buffer was transmitted to the serial port and CoolTerm was able to capture the data and save these datasets as textfiles for compression and encryption result analysis.The measured time for compression, encryption ,decryption and decompression are shown below using python script.

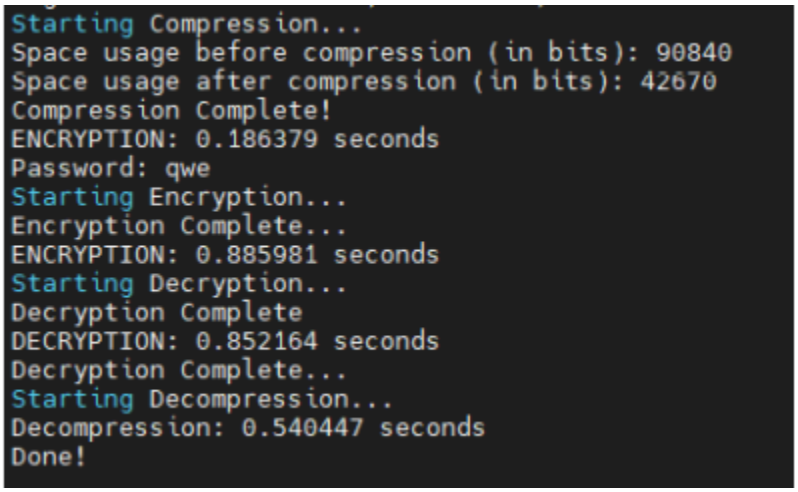

Figure 10: Overall system performance

Compression Subsystem

For the compression test, the output .txt file size is smaller than the input .csv file proof that compression has occurred.

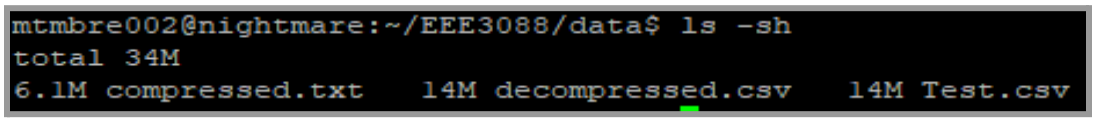

Figure 11: Input and Output file

Input file = Test.csv, Output file = compressed.txt

Compression ratio = $\frac{Original\ file}{compressed\ file} = \frac{14M}{6.1M} = 2.3$

Note: File was compressed to 44% of its original size and still retained 100% of its original data. The same applies to the decompression test as well. The input file (compressed.txt) is smaller than the output file (decompressed.csv).

As for the results of the last test the data integrity test, the subsystem passed as there is no difference between the original data and the data that went through compression and decompression

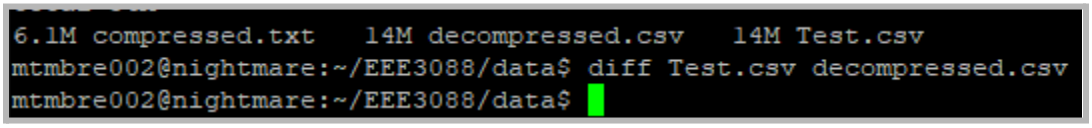

Figure 12: differences between the Original file and processed data file

This result shows that the compression algorithm is lossless thereby retaining data integrity.

Compression Algorithm	Compression Ratio	Compression Speed (MBps)	Decompression Speed(MBps)
huffman	2.3	0.094	0.034
gzip	2.62	0.497	0.588
zlib	3.22	1.99	14.21
zstd	2.22	0.331	0.446
lz4	1.69	0.479	0.539

Table 13: Performance different compression algorithm

Initially, zstd compression was the chosen compression algorithm for the subsystem. Based on the results of the experiment obtained, shown in Table 18 above but due to its complexity and its large size it was not perfect for our system , we needed to minimize computational resources and also save the very limited storage space hence we chose the smaller huffman encode/decoder algorithm.

Encryption Subsystem

The encryption surpassed 2 of the 3 experiments. With the encryption test, I was able to extract the ciphertext from CoolTerm, and the comparison test between the ciphertext and the original compression text file resulted in 0% comparison in each line. Within the decryption test, I was also able to extract the decrypted compression data, and according to our simulations, the decrypted text file had a 100% match to the original compression text file. I can conclude that AES-128 is secure, as changing decryption keys did not result in a perfect decryption. The results of the encryption performance showcasing each of the experiments above will be recorded and may be viewed within our video demonstration submission.

As stated, only 2 of the 3 experiments succeeded. At this moment I have not been able to perform an efficiency test and perform any necessary speedups, as the functionality of the encryption algorithm has been the primary focus. It is expected that AES-128 will produce the fastest rate of encryption[5] however testing this theory will come at a later stage.

VALIDATION USING IMU

Hardware testing is a helpful process which allows for entire systems to be tested and validated using real world components, after simulation. Hardware based validation is the most important step in the development cycle as it allows users to make sure each component or hardware is functional and determines whether your system is ready to be released and implemented in the real world. In hardware validation, it is crucial to test and reduce any unknown issues as this may affect the system function as a whole. A system is a collection of subsystems and without hardware based validation, you risk the chance of subsystem failure, system failure and reputational damage.

IMU Module

An ICM 20948 will be used to collect data which will be used to test the system. The IMU is a low power 9-axis device with a 3- axis gyroscope, 3-axis accelerometer, 3-axis compass, a humidity sensor, temperature sensor, among other components. An ICM 20649 will be used on the SHARC BUOY. It is a 6-axis device with a 3-axis gyroscope, 3-axis accelerometer, humidity sensor, temperature sensor, among other components. The two devices are similar: they both have a VDD operating range of 1.71V to 3.6V and both utilize I2C and SPI communication protocols. They both have on-Chip 16-bit ADCs and Programmable Filters. with a few differences. Below is a table showing some of the differences between ICM 20649 and ICM 20948.

ICM 20649	ICM 20948
Is a 6-axis device	Is a 9-axis device
offers precise analysis of contact sports applications by providing continuous motion sensor data before, during, and after impact	integrates multi powerful sensors such as gyroscope, accelerometer, magnetometer, barometer, temperature and humidity sensor, etc.
No Axis Compass	3-Axis Compass with a wide range to ±4900
Has a 4kB FIFO buffer enables the applications processor to read the data in bursts	Has android support

Host interface: 7 MHz SPI or 400 kHz I2C	Auxiliary I2C interface for external sensors. 7 MHz SPI or 400 kHz Fast Mode I²C.
Extended FSR for gyroscope: ±4000 dps	±2000 dps FSR for gyroscope
Extended FSR for accelerometer: ±30g	±16g FSR for accelerometer

Table 14: Comparison between ICM-20649 and ICM-20948 [7][8]

In order to verify our IMU is in working condition, we derived test conditions to examine and validate the accelerometer and gyroscope. Since the 3-Axis compass is not available on the SHARC BUOY, we decided not to test its function. Before beginning the tests, we first displayed the IMU readings to the console at a continuous rate to examine the data from the IMU and its changes. We decided to choose a sampling rate of 50ms as it was sufficient to examine the changes in the accelerometer and gyroscope readings. We also decided to use the maximum scaling factor for both the accelerometer and gyroscope as this would increase the sensitivity of these sensors to make their varying readings easily examinable. The IMU rested on a breadboard to ensure its stability before the tests were conducted.

Testing the accelerometer consisted of applying oscillating forces to the IMU, namely side-side push and pull forces for the x axis, forward and backward push and pull forces for the y axis, raising and lowering forces for the z axis. By applying these forces, an acceleration would occur within these axes which would reflect in our console readings. Each axis was tested for approximately 10 seconds which can be seen in Figure 1 which illustrates the readings collected from the accelerometer as a result of the oscillating force testing. When testing the x-axis, we used a gradual motion resulting in its meander shape. When testing the y-axis, we wanted to see what would happen if we applied impact forces, such as flicking the breadboard with a finger, resulting in a shape with a number of large spikes. In testing the z-axis, we pulsated our raising and lowering of the breadboard at a fast rate, which can be seen in the resulting shape. Though the z-axis responds to changes in axis motion, it can be seen to have an offset of around 2000 which can be corrected in software.

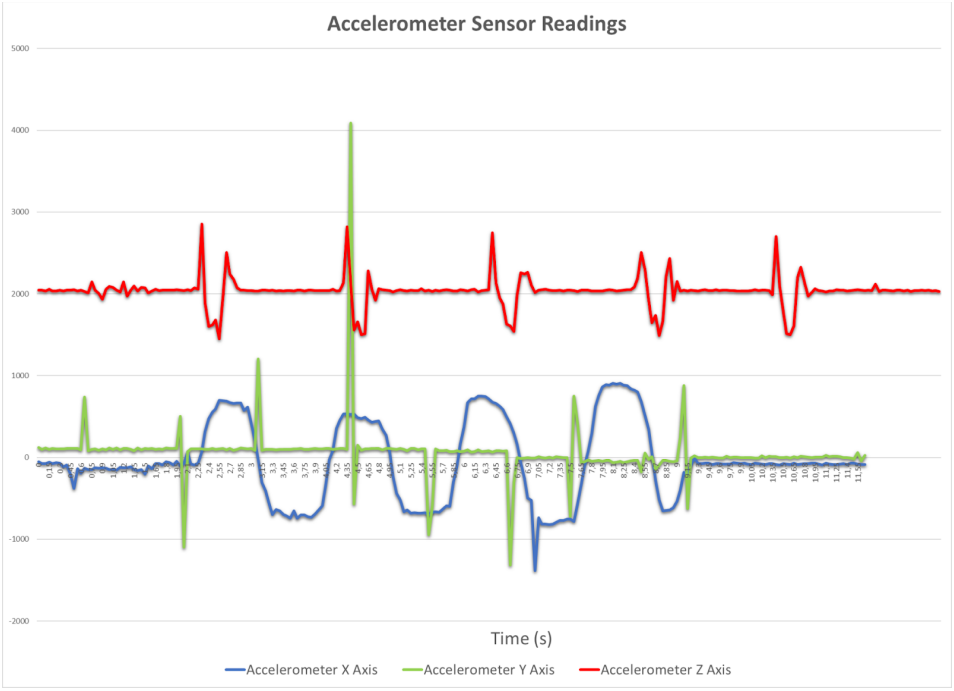

Figure 13:Accelerometer Sensor Readings

We followed the same approach when testing the gyroscope, where each axis would be tested for approximately 10 seconds. Testing the gyroscope consisted of applying oscillating rotations within the x, y and z axes. Gyroscope sensors measure angular velocity and by applying these oscillating rotations, varying positive and negative readings should occur within our console readings. Figure 4 illustrates the readings collected from the gyroscope sensor after applying these oscillating rotations, demonstrating that the angular velocity is varying and that the gyroscope sensor is working.

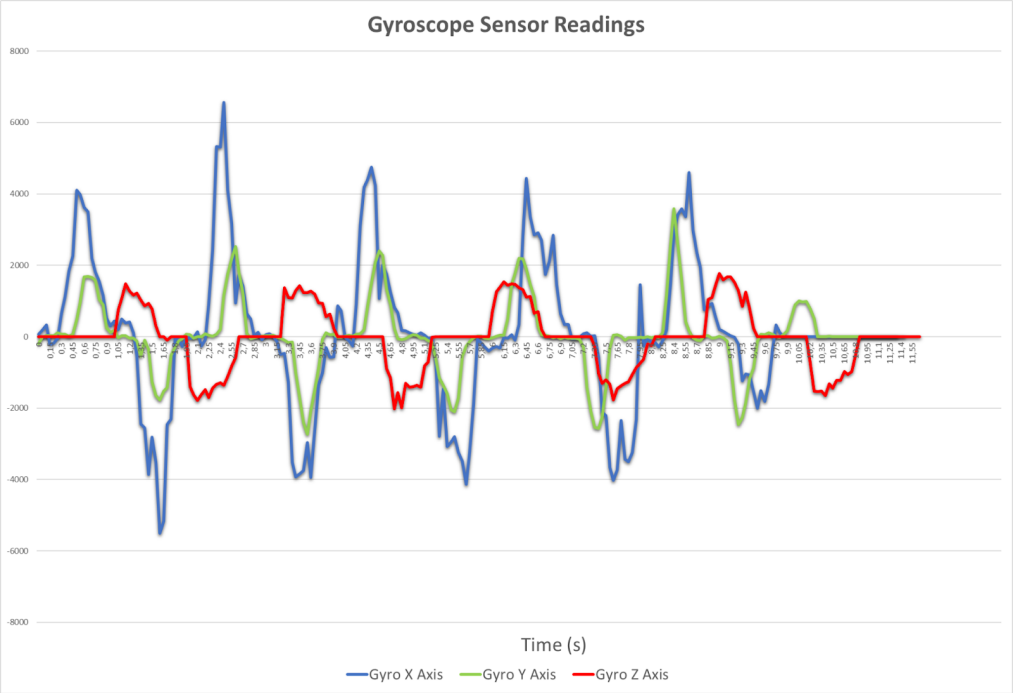

Figure 14:Gyroscope Sensor Readings

Even though we could not test and confirm that the accelerometer and gyroscope readings are in fact accurate, we can confirm that the accelerometer and gyroscope sensors are indeed functional, and that the IMU is functional.

Experiment Setup

Overall Functionality of the System

Implementing the IMU into the system was fairly straight forward. After soldering the pins and following the pin out guide provided by mokhwasomssi[12] from were able to see what seemed to be a working IMU, based on its lit LEDs. We then had to test if the IMU was indeed functional, and after following the source code provided by mokhwasomssi we were able to collect and continuously display our IMU data readings to our communication terminal, shown in figure 14. We then proceeded to test if the data readings in the terminal would respond, when applying forces and rotations to the IMU and took note of these changes. Once satisfied with our ability to collect and transmit IMU data, we then implemented compression and encryption to function with the IMU. We followed a similar approach to that of our simulation where we created a file containing a subset of IMU generated data, sent this file for compression and input the compressed data lines into the microcontroller for the data to be encrypted and transmitted back to the communication channel for encryption analysis. The data was then decrypted and used for compression analysis as well.

Compression Subsystem

The main criteria used to test the compression subsystem was compression ratio. For compression, we only tested the compression submodule by only compressing the IMU data and sending it to the computer where it is sent back for decompression. In the end, we compare the file size of the raw data and compressed data to ensure that data compression has occurred, also by comparing the uncompressed data and the raw data we can check the data losses. We also tested the time it takes for the compression and transmission of data as well as the decompression of data of the system. By increasing and decreasing the sampling frequency we were also able to determine whether the compression ratio is affected by the volume of data being operated on.

Encryption Subsystem

In this experiment we tested the IMU and encryption pair, determining if the IMU data can be collected, encrypted and decrypted. We used CoolTerm to display our encrypted and decrypted data. For a successful encryption subsystem, we first needed to collect the IMU data within the buffer and process the buffer, sending the string buffer into our encryption function and determine if the data inside the buffer had indeed been encrypted. The plan was to collect a subset of the IMU data (displaying the subset as well, to the console using CoolTerm for future reference) and concatenate the subset into a buffer. Once completed we sent the buffer to the operational encryption function, and waited for the program to print the encrypted buffer data to the console serial port. Since the buffer before encryption was displayed to the console earlier, we were able to compare the non encrypted data and encrypted data, determining if there was no match between their data. After encryption, the program initiates a decryption process where it takes the encrypted buffer data and sends it to the operational decryption function, displaying the decrypted buffer data to the console. We expected the decrypted displayed data should match the IMU data collected before.

Results

The data collection using the ICM-20948 was successful. In figure 15 below were able to collect data from the IMU and transmit that data to the communication terminal. In figures 13 and 14, we can see the IMU responds to applied forces and axial rotations.

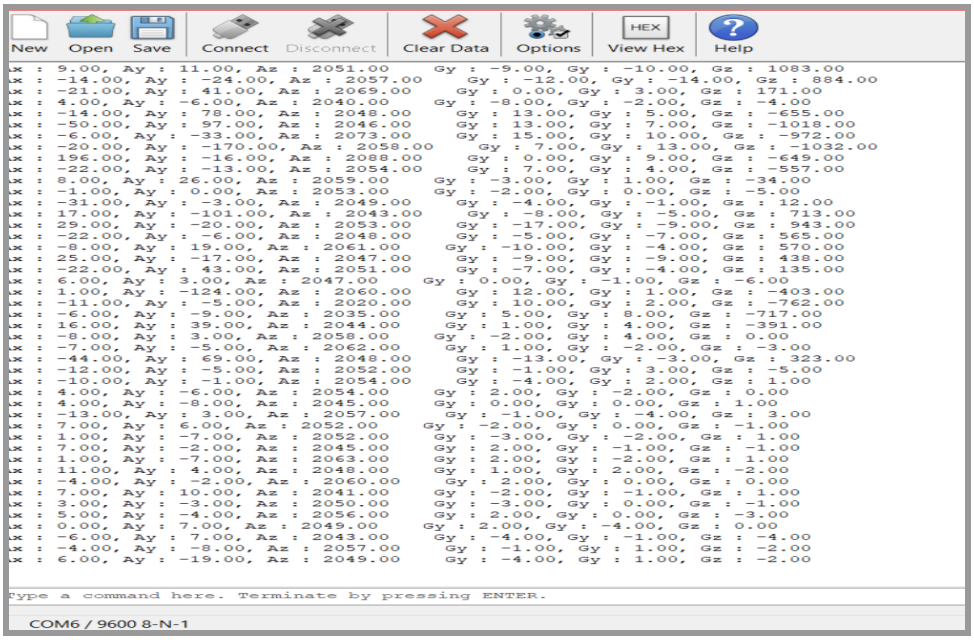

Figure 15: IMU Data Readings using CoolTerm

Though we were not able to combine compression and encryption within the STM32, we decided the best option was to perform compression on a local pc and transmit the compressed data into the microcontroller for encryption. With this in mind, we decided on collecting IMU data and transmitting it to a pc using CoolTerm, illustrated in figure 15, and storing the data into a text file. From here, we used a c program to encode our IMU text file using Huffman principles to compress the file, which can be seen in figure 16. After successfully compressing the file, we decided to use python to read in the compressed file and transmit each line into the microcontroller for it to be encrypted using AES-128 and transmitted back simultaneously. The compressed file to be encrypted and the program source code can be found in figure 19 and 18 respectfully. As python loads the data into the microcontroller, it would simultaneously listen for incoming transmissions and store the retrieved encrypted data into another text file, illustrated in figure 20. We followed the same process with decryption, using python to load the encrypted and retrieve the decrypted data, storing it in another file. Once decryption was complete, we then used our c program to decompress our decrypted data to which we then compared if data loss occurred. Though we were not able to combine the subsystems within the microcontroller, our compression and encryption algorithms are both functional. Our compression algorithm allowed for a +50% reduction in file size, as can be seen in figure 16. Our compression algorithm is also lossless which is demonstrated in figure 17. In figure 17, calling the function “diff” compares two files, namely our decompressed and raw IMU data. After calling the function, if there were to be differences between the files, an error message would occur, however as can be seen in the figure, no errors occurred. Our encryption algorithm was able to convert our compressed data in figure 19 into hex cipher text in figure 20, and by visual comparison we can see that they differ. We were also able to decrypt our data using the algorithm, resulting in 100% matches between the raw daw and decrypted data.

```
~/Compression$  ./huffman -e AccelGyroData.csv CompressedAccelGyroData.csv
File size = 10030
Table size = 27
~/Compression$ ls -l
total 38
-rw-r--r-- 1 user user 10030 Oct 16 11:23 AccelGyroData.csv
-rw-r--r-- 1 user user  4484 Oct 16 11:28 CompressedAccelGyroData.csv
```

Figure 16: IMU text file undergoing compression, illustrating file size reduction in “yellow”

```
able size = 12
/Compression$ ./huffman -d DecryptAccelGyro.csv  decompressedData.csv
/Compression$ diff AccelGyroData.csv decompressedData.csv
/Compression$
```

Figure 17: Decompression comparison test, with function call “diff”

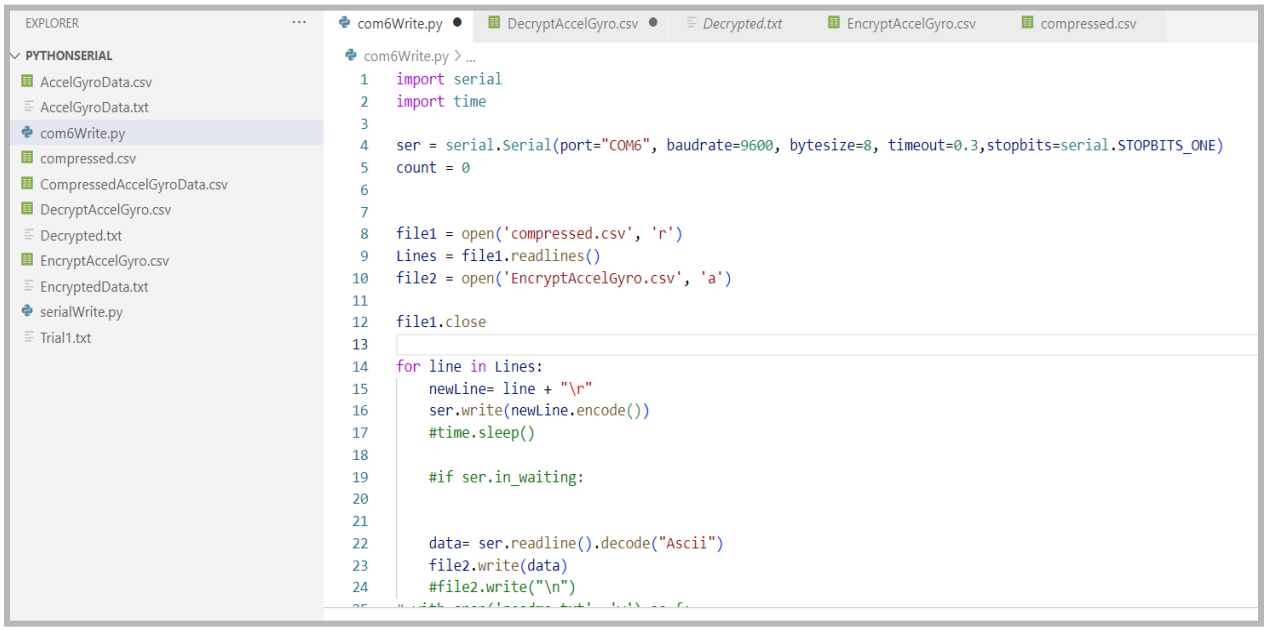

Figure 18: Python Script to Transmit and Retrieve STM Encryption Data

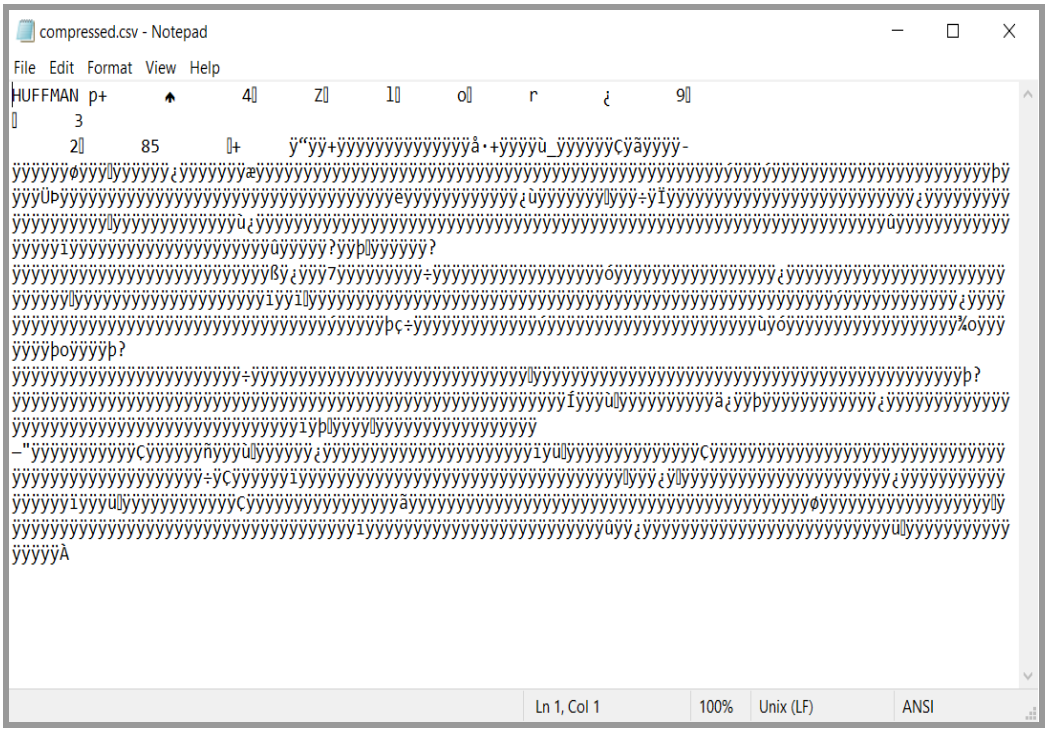

Figure 19: Compressed File Contents compressed.csv

Figure 20: Python generate encrypted text file, encrypting compressed.csv

Additionally, we can validate that the IMU and encryption subsystem pair is deemed functional. As mentioned in our encryption experiment, we would first store our IMU subset data in our buffer and display the buffer contents to the console and as can be seen in Figure 21 below. The first 7 lines on the CoolTerm console consists of the IMU data where Ax,Ay and Az represent the accelerometer readings in its 3 axes and Gx, Gy, and Gz represent the gyroscope readings in its 3 axes. Our buffer size was set to store 1024 bytes worth of IMU data readings and as a result only 10 accelerometer and gyroscope data pairs could be stored at a time. The buffer is expandable, as our stm board still has around 2.5kb left worth of RAM, thus an extra 1024 would be able to store an additional 10 and possibly more after removing unnecessary

“Ax,Ay,Ax” labels and brackets. The 10 samples were easily encryptable as can be seen in the lines 8-19 in Figure 8 below, representing hex cipher text of the IMU data. It can also be seen that the hex cipher text and the original IMU data do not match. The initial experiment was to see if the entire 1024 byte concatenated buffer could be sent for encryption however, the encryption algorithm had an issue, dealing with that much data and as a result, the concatenated buffer string had to be broken into chunks of 256 bytes, where each chunks would be encrypted and displayed to the console. Similarly, decrypting the cipher could only be done in 256 byte chunks, however the decryption was still deemed functional as can be seen from line 20-26 in the figure below, representing the decrypted cipher text which can be seen to match the original IMU data in lines 1-7.

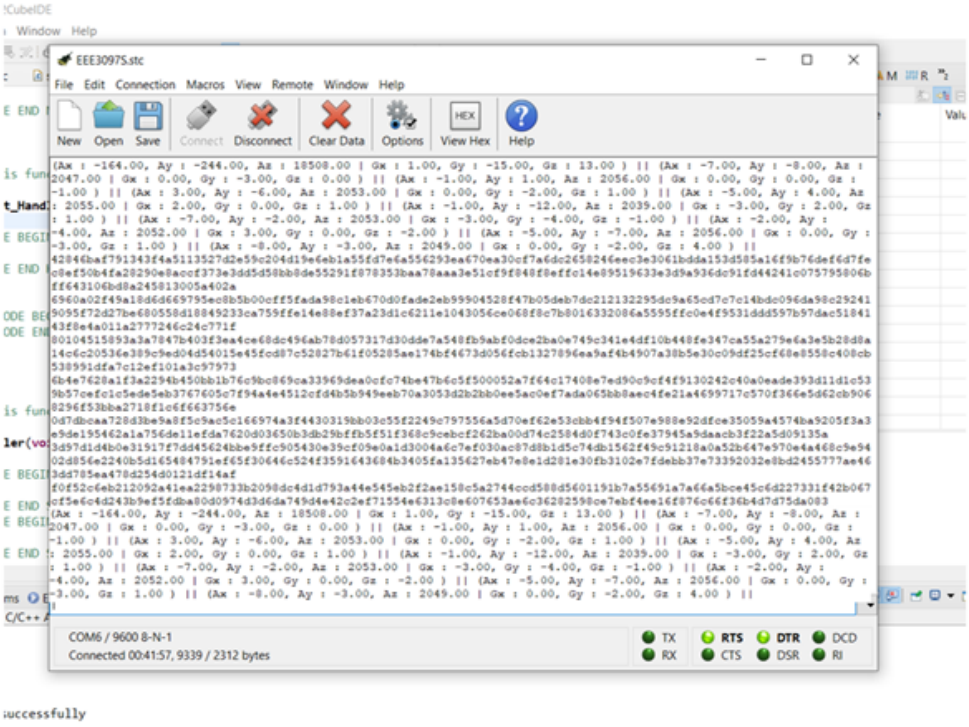

Figure 21: Encryption & Decryption Experiment Result

CONSOLIDATION OF ATP’s & FUTURE PLANS

Specification(s) ID	Specification(s)	Figure of Merits
DS001	An ICM-20649 motion tracking IMU device to be used.	We can send the data collected to a computer to prove we are collecting the correct information using a similar IMU, an ICM-20948

DS003	We shall use the Huffman encoding algorithm to compress our data from the IMU	The algorithm is supposed to compress the data from the simulations so that we will be able to retrieve at least 25% of the data
DS004	We shall use a simplified version of a huffman encoder/decoder which is not too complex. It is also fast	Fastest time of execution
DS002	The microcontroller will communicate with the IMU in order to receive the data & a serial connection between the microcontroller and PC will be established	Print out data from the IMU to prove retrieval
DS006	The use of an AES library to encrypt and decrypt the compressed IMU data and ensure security.	Sending an unencrypted file to the microcontroller and retrieving it, seeing if the input data matches the yield data. The data should not match
DS007	Choosing an encryption algorithm that limits the number of processors done on the microcontroller.	Time how long it takes for the file to be encrypted and returned from the microcontroller, generating an average speed

Table 15: ATPs from Paper Design

Specification(s) ID	Specification(s)	Review
DS001	An ICM-20649 motion tracking IMU device to be used.	IMU ICM-20948 is functional, responding to applied forces and rotations about it's axes
DS003	We shall use the Huffman encoding algorithm to compress our data from the IMU	It is lossless so that we are able to retrieve at least 25% of the Fourier transform coefficients
DS004	We shall use a simplified version of a huffman encoder/decoder which is not too complex. It is also fast	Fast as it is efficient in compressing text or program files
DS002	The microcontroller will communicate with the IMU in order to receive the data & a serial connection between the microcontroller and PC will be	A serial connection was successfully established between the microcontroller and PC, allowing for data packets to be stored on and received from

	established	the microcontroller
DS006	The use of an AES library to encrypt and decrypt the compressed IMU data and ensure security.	AES-128 was used to successfully encrypt and decrypt the compressed .csv data ensuring 100% security
DS007	Choosing an encryption algorithm that limits the number of processors done on the microcontroller.	The AES library provided by kokke[11] is a simplified version of the general AES library, intended for areas where memory and processing power are constrained.

Table 16: ATP Paper Design Review

Before our project can be implemented on the SHARC BUOY,we first need to address our limitations.The project report speaks on wireless file transmission using Iridium, however we have only tested with a serial connection to a PC. We therefore need to research wireless file transmission with the stm32 microcontroller and determine if it can be implemented into the project. We also had difficulties combining compression and encryption in our microcontroller, and as a result, solving this issue should be our primary focus. A necessary addition to the SHARC BUOY team, would be to research and implement a form of file storage, The STM32 contains 8KB of RAM, of which 4-5KB is usable. Since each character is considered one byte, we'd only be able to store 4000 characters at a time before we need to transmit and flush the data. 4000 characters translates to around 200 lines of IMU data readings (±20 characters per sampled IMU reading), and if we're sampling at a rate of 50ms, we'd only be able to collect 10 seconds worth of data at a time. This means we would possibly be using Iridium every 10 seconds which could be quite expensive. Finding a solution to file storage will allow for more data to be captured and stored, and less frequent Iridium transmission times, reducing the cost.

CONCLUSION

A STM32 Microcontroller, is used to compress, encrypt, and transmit data from an inertial measurement unit (IMU) for further analysis of the environments surrounding the Antarctic region. This STM32 Microcontroller design system is one of many subsystems that make up the SHARC BUOY designed by Jamie Jacobson.

Over the course of this semester we have been actively working on this design system , it was separated into many different parts before coming together. The parts consisted of testing the extraction of the data from the IMU module and placing it into a file, compressing , decompressing the file and encrypting and decrypting the file and checking the integrity of the processed data. Each part was tested and had to meet specific criteria before being used. The different parts of the design worked exactly as expected and thus when it was cascaded it

produced the results that were desired except the failure of implementing compression on the microcontroller , It can be concluded that the design has met most of the requirements stipulated in the project handout.

ADMIN DOCUMENTS

Distribution of Work

Work	Member
Admin documents	MTMBRE002 & STLMOU001
Requirements Analysis	MTMBRE002
Paper Design	MTMBRE002
Validation using simulated data	STLMOU001
Validation using IMU	STLMOU001
Consolidation of ATPs and Future Plans	MTMBRE002 & STLMOU001
Conclusion	MTMBRE002 & STLMOU001

Table 17: contribution of each of the team members

Project Management Tools

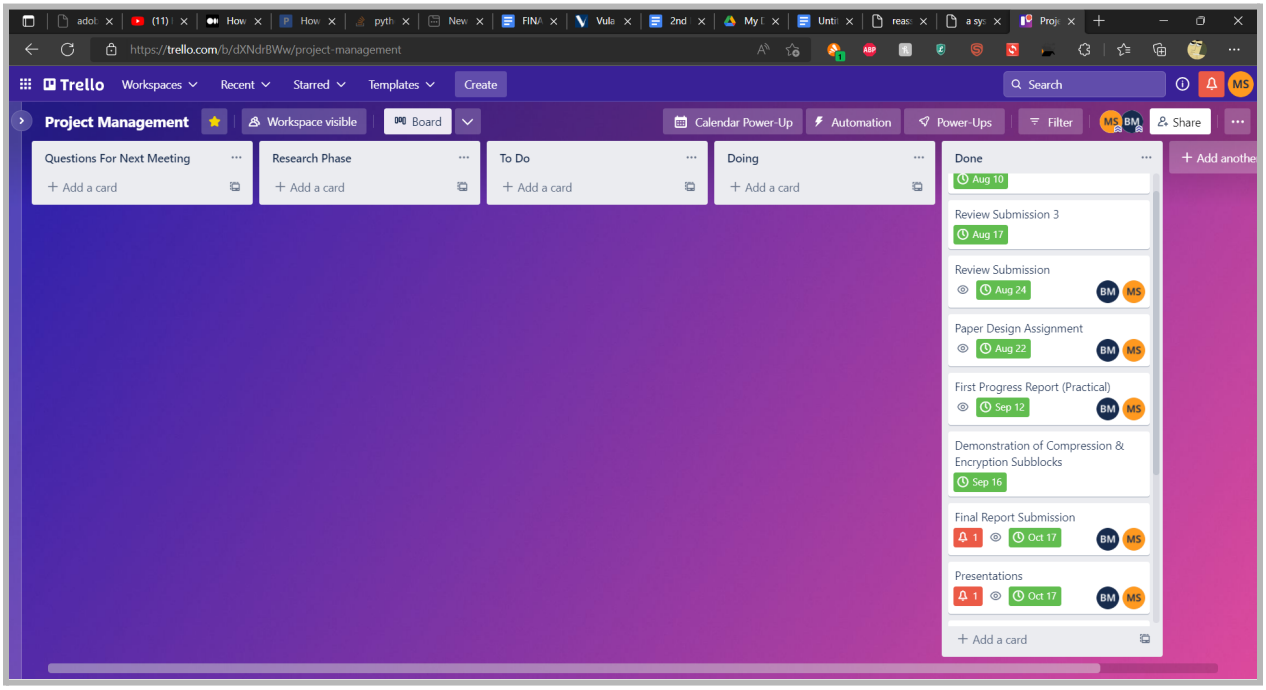

Figure 22: Our Project Management Tool

- Here is the link to our github repository here

Development timeline

The project was planned to be completed within the course timeline. Changes were made when the Second progress report got an extension causing the deadlines for the final report and presentation to be pushed back and were scheduled to be due on the same day

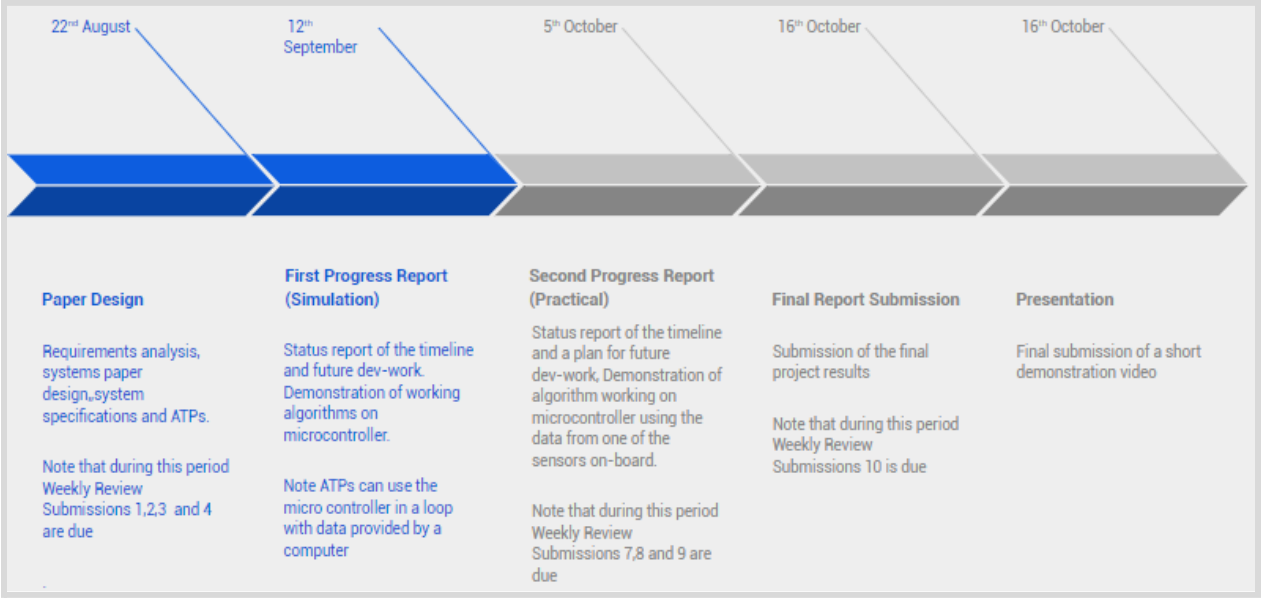

Figure 21: Our Project Development Timeline

REFERENCES

[1] Import User, “Smaller and faster data compression with Zstandard,” Engineering at Meta, 31-Aug-2016. [Online]. Available: https://engineering.fb.com/2016/08/31/core-data/smaller-and-faster-data-compression-with-zstandard/. [Accessed: 08-Oct-2022].

[2] “Sense HAT (B) for Raspberry Pi, Onboard Multi Powerful Sensors ..” https://www.waveshare.com/sense-hat-b.htm (accessed: Oct. 05, 2022).

[3] Lones and k5054, “File compression and decompression and C,” CodeProject, 22-Dec-2021. [Online]. Available: https://www.codeproject.com/Questions/5320638/File-compression-and-decompression-and-Cv. [Accessed: 05-Oct-2022].

[4]“ICM-20649 Datasheet | TDK.” https://invensense.tdk.com/download-pdf/icm-20649-data-sheet/ (accessed: Oct. 05, 2022).

[5] Admin, “How to use UART to transmit data in STM32 || poll || inetrrupt || DMA,” ControllersTech, 14-Dec-2021. [Online]. Available: https://controllerstech.com/uart-transmit-in-stm32/. [Accessed: 05-Oct-2022].

[6]A. Al Tamimi, "Performance Analysis of Data Encryption Algorithms", Cs.wustl.edu, 2022. [Online]. Available: https://www.cs.wustl.edu/~jain/cse567-06/ftp/encryption_perf/. [Accessed: 22- Aug- 2022].
[7]“Sense HAT (B) for Raspberry Pi, Onboard Multi Powerful Sensors ..” https://www.waveshare.com/sense-hat-b.htm (accessed: Oct. 05, 2022).

[8] Lones and k5054, “File compression and decompression and C,” CodeProject, 22-Dec-2021. [Online]. Available: https://www.codeproject.com/Questions/5320638/File-compression-and-decompression-and-Cv. [Accessed: 05-Oct-2022].

[9]“ICM-20649 Datasheet | TDK.” https://invensense.tdk.com/download-pdf/icm-20649-data-sheet/ (accessed: Oct. 05, 2022).

[10] Admin, “How to use UART to transmit data in STM32 || poll || inetrrupt || DMA,” ControllersTech, 14-Dec-2021. [Online]. Available: https://controllerstech.com/uart-transmit-in-stm32/. [Accessed: 05-Oct-2022].

[11]Kokke(2020) Tiny AES in C[Source Code]. GitHub - kokke/tiny-AES-c: Small portable AES128/192/256 in C

[12]MokhwaSomssi(2021) STM32_hal_icm20948 [Source Code]. GitHub - mokhwasomssi/stm32_hal_icm20948: ICM-20948 library with STM32 HAL driver