

SHARC BUOY FINAL REPORT

Due: 31 October 2021
RAMSES TAGNE - TGNRAM001
VINCENT MUZERENGWA - MZRVIN001

1. INTRODUCTION

This document is a report that entails the complete design and simulation of an ARM based digital IP. The ARM based digital IP is a subsystem of the SHARC Buoy system. The subsystem will be responsible for compressing data collected from the IMU sensor and encrypt the data so that it can be transmitted for processing

2. REQUIREMENT ANALYSIS

This section describes the process for identifying the requirements of this system which has been broken down into two smaller subsystems: Encryption and Compression. A flow chart of the system is shown below.

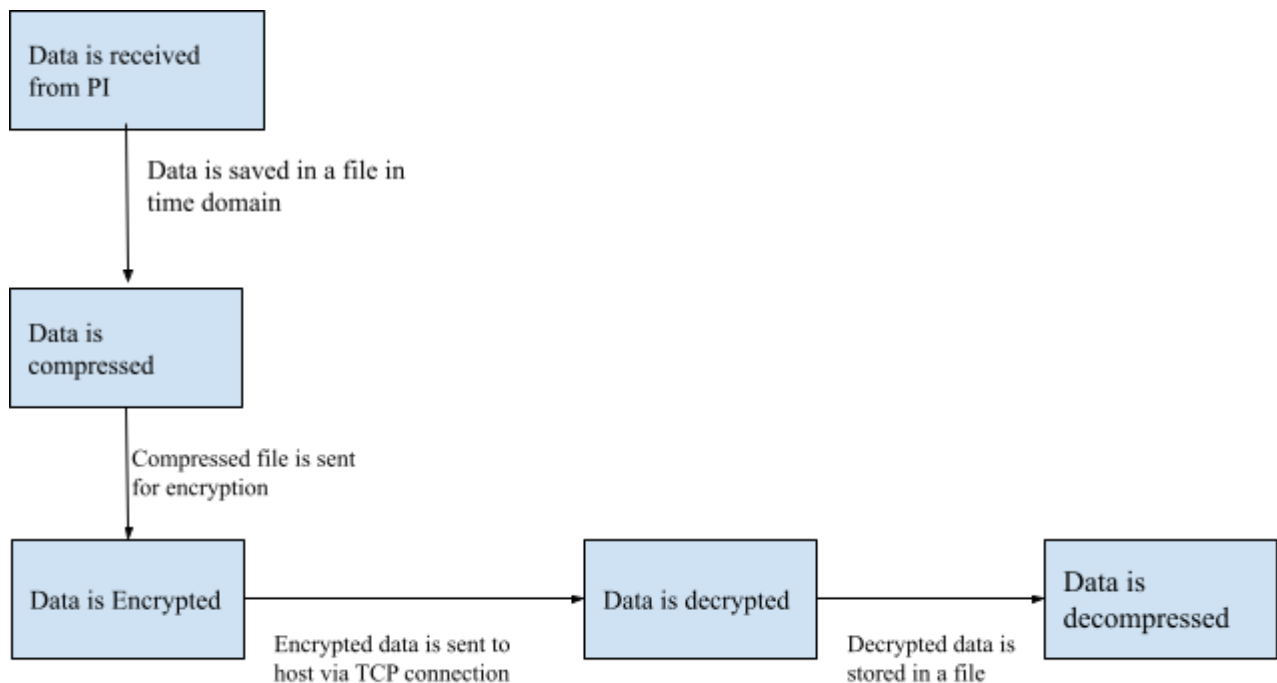


Figure 1.1 showing requirements of the system

2.1. Discussing the requirements

The aim is to design requirements and tasks to which will simplify the design process and make it possible for the system to be successfully implemented. The general requirements are shown in the table below:

ICM-20649	Compression
Requirement	The lower 25% of the Fourier coefficients of the data should be extracted without loss
Refines	A need of Sampling at low frequencies to retain only low frequencies of the data obtained. The system will require to transform data obtained in time-domain to frequency-domain in order to perform sampling to retain low frequencies only
Verification	Requirement is verified by demonstration

Table 2.1 showing requirements Compression subsystem

ICM-20649	Data Transmission
Requirement	Minimal data processing on buoy, to use minimal amount of power
Refines	The data has to be streamed to the receptor in Cape Town to minimise data processing on the buoy. The throughput of the communication channel should be equivalent or greater than the maximum throughput of the IMU. The throughput of the receptor should be equivalent or greater than the throughput of the communication channel to process data in real time.
Verification	Requirement is verified by inspection

Table 2.2 showing requirements for Transmission subsystem

ICM-20649	Encryption
Requirement	The data is only be accessible by the intended receptor and can not be intercepted and read by any other unforeseen receptor
Refines	A need for the data to only be understood and read by the intended receptor. Operation should be similar to a lock and its key where, the data is “locked” and is only accessible by a “key”
Verification	Requirement is verified by demonstration

Table 2.3 showing requirements for Encryption subsystem

2.2. Analyzing the requirements using existing research and reports

In an attempt to rise up to the requirements listed above, subsystems are designed in order to perform in a way that satisfies these requirements. A compression algorithm is developed and tested to be able to compress the data in order to retain only the lower 25% percent of Fourier coefficients without loss, and an Encryption algorithm is used to inhibit any unwanted interceptor of the data to be able to read and interpret the data. Different algorithms were considered. Comparison of the algorithms for the subsystems is discussed in the section below.

2.2.1. Comparison of some available compression and encryption algorithms

1. Compression algorithms:

For this system, three compression algorithms were considered, that is, Run length encoding (RLE), Discrete Cosine Transform and LZ4 algorithms.

RLE is a basic form of data compression that converts consecutive identical values into a code consisting of the character and the number marking the length of the run[1]. RLE algorithm is a lossless compression algorithm. Its main advantage is that it is simple to implement. The main disadvantage of this algorithm was that it required a lot of space as one cannot easily estimate the amount of space that the compressed data is going to use.

LZ4 is lossless compression algorithm, providing compression speed > 500 MB/s per core, scalable with multi-cores CPU [1]. LZ4 is extremely fast and can process large amounts of data of up to 1TB. The main disadvantage of this algorithm was that it is a very complex algorithm that uses a lot of processing power.

DCT is a mathematical operation that is used to convert a signal from one representation to another. It works by separating images into parts of differing frequencies[2]. During a step called quantization, where part of compression actually occurs, the less important frequencies are discarded. It is a lossy compression algorithm. It can offer a compression ratio of up to 1:10.

A summary of the considered algorithm' strength and weaknesses are summarized in the table below:

Algorithm	Compression ratio (Best case scenario)	Strength	Weakness
RLE	1:3	Is easy to implement	Only work well for repetitive data
LZ4	1:5	Works well large files	Uses a lot of processing power
DCT	1:10	Is easier to control	Is a lossy algorithm,

		compression ratios	therefore some data may be lost
--	--	--------------------	---------------------------------

Table 2.4 showing comparison of compression algorithms

The DCT algorithm was chosen for the subsystem because it is easier to control the correlation control ratio and the percentage of data that can be retained. It is also easier to estimate the amount of space needed when compressing the data. Another big advantage of DCT is that it can be easily implemented using java.

2. Encryption algorithms:

Three encryption algorithms were considered for this system.

Triple DES Encryption: Triple DES uses three individual keys with 56 bits each. The total key length adds up to 168 bits, but experts say that 112-bits in key strength is more like it. Though it is slowly being phased out, Triple DES is still a dependable hardware encryption solution

RSA Encryption: Unlike Triple DES, RSA is considered an asymmetric encryption algorithm because it uses a pair of keys. The public key is used to encrypt a message and a private key to decrypt it. It takes attackers quite a bit of time and processing power to break this encryption code

Advanced Encryption Standards (AES): Although it is extremely efficient in 128-bit form, AES encryption also uses keys of 192 and 256 bits for heavy-duty encryption. AES is resistant to all attacks, with the exception of brute-force attacks, which attempt to decipher messages using all possible combinations in the 128-, 192- or 256-bit cipher.

AES algorithm is selected for implementation because it is secure and its components and design principles are completely specified. Using the AES standard, this algorithm can only accept 128 bits of block, and key size can be selected from 128/192/256 bits. Based on the key size, the number of rounds will vary. Also, the AES algorithm is compatible with the transmission channel used.

3. Data Transmission:

With regards to the transmission of data from the IP to Cape Town, a series of features were taken into consideration to choose the best possible option for the transmission of our data.

It was decided that a wireless communication channel was to be used for transmission.

Different options were considered based on their compatibility with the data to be transmitted after the files had been encrypted. The selection of wireless communication protocol depends on the type of data, its timing and the transmission channel. This was done under the frame of the specification of the prototype, which is low power, real time and long-range data communication. Table 2.5 summarizes the transmission technologies that

were considered for the transmission of the data. The technologies were compared based on standard, power consumption, range and security.

Technology	Standard	Peak Data rate / Power	Range	Security
Wi-Fi	IEEE 802.11	11mbps-1 Gbps/1W	100m	WEP,WPA
Bluetooth	IEEE 802.15.1	1-3 mbps/1watt	100m	8-128bits
BLE	IoT interconnect	1 mbps/10-500mW	100m	128bit AES
Zigbee	IEEE 802.15.4	250 kbps/1mW	10m	128bit
Z-Wave	Z-wave	100 kbps/1mW	30m	Triple DES
RFID	ISO18000/29167/20248	423 kbps/1mW	1m	Available
NFC	ISO/IEC 13157	424 kbps/1mW	0.1m	Available
ANT+	ANT+	1 Mbps/1mW	100m	128bit AES
LTE	3GPP	1000Mbps/1mW	28km	SNOW 3G
EnOcean	ISO/IEC14543-3-10	125kbps/eharvesting	100m	128bit AES

Table 2.4 showing a comparison of different transmission technologies

The considered technologies shown in the table above were then compared according to the requirements of the system, i.e. low-power usage, high-speed and reliable communication. Wi-Fi communication, despite its high performance in long range connectivity, required high power usage and its cost was found to be relatively high.

Zigbee, Z-Wave and classic Bluetooth protocols, despite their low power consumptions, their peak data rate and the range of coverage were below the required range. Similarly, RFID and NFC were found unsuitable for long range application.

LTE and EnOcean were considered as excellent for long range applications. However, they were found to be expensive options for the system. They also required additional features such as a dedicated 4G mobile network for the former and energy harvesting mechanism for the latter.

From the considered wireless technologies BLE (Low Energy Bluetooth) and ANT+ fulfilled the specification requirements for high data rate, low power, long range, low cost and high reliability. Both technologies were deemed as suitable for personal-area networks of sensors and are popular in health monitoring and performance applications. Both operate in the 2.4 GHz ISM band (license free frequency band). They use short duty cycle technique and deep sleep mode to ensure low energy consumption.

BLE vs ANT+

As mentioned earlier, there are numerous similarities between ANT+ and BLE. Their peculiar differences emanate from the topologies they support, how they transfer data packages and a number of channels can be connected. Table 2 compares several other features of the two technologies. From the analysis regarding the two wireless technologies, both were found to be very competitive and each had a slight edge on the other on different features. Comparing the two, a decision was made based on two key features, which are the effective throughput and the Compatible Security algorithm feature.

With the BLE, the effective throughput is up to 1.4Mbps ,while with the ANT+ it is up to 60kbps. Noticing that the ICM-20649 has a gyroscope and accelerometer with throughputs corresponding to 75kbps, we concluded that BLE was the more efficient and appropriate method of transmission, since it could transmit at a throughput even higher than the maximum throughput of both the gyroscope and the accelerometer in the ICM-20649. Secondly, the security in the ANT+ allowed 64-bit key and 128-bit AES, while BLE was compatible with the 128-bit AES. With this info, noticing that they are both compatible with the Encryption system in place, BLE has been found preferable for our design as a means of wireless communication channel, despite equally excellent ANT+ features

Technology	ANT+	BLE
Frequency Range	2.4 to 2.483 GHz	2.4 to 2.483 GHz
Frequency	Fixed	FHSS
Band	ISM	ISM
Network Topologies	P2P, star, tree, mesh	P2P, star(Scatternet)
Modulation	GFSK	GFSK
Channel width	1 MHz	1 MHz
Protocol	Simple	Complex
Data rate	1 Mbps	125kbps to 2 Mbps
Range	100m	100 m
Number of Connections	Very high	Up to 20
Effective throughput	Up to 60kbps	Up to 1.4Mbps
Security	64-bit key, 128-bit AES	128-bit AES

Table 2.6 showing a comparison of ANT+ and BLE

BLE (Bluetooth Low Energy)

It is a standard wireless communication protocol that can be used for short to Long range applications using low power. The low power feature of BLE enables applications to operate for months with a single coin battery. Bluetooth 5 is the latest standard for BLE with a data rate of up to 2 Mbps, with long range, high sensitivity and improved broadcast capability. BLE

(Bluetooth 5) also provides the option of using four discrete transmission speed choices namely 125 Kbps, 500 Kbps, 1 Mbps and 2 Mbps. The 2 Mbps provides higher throughput capabilities.

4. Reception

It was decided that a Raspberry PI was to be used to receive transmitted data. This is because the raspberry pi had a throughput of up to 35.1 Mbps. This is sufficiently large enough to be able to receive and process all the data even at its maximum speed of transmission.

2.3. Feasibility Analysis

Technical feasibility:

- a raspberry pi with a 4GB RAM is required to design the ARM based digital IP
- A personal computer with a RAM of at least 4GB is also required to test and debug compression and encryption algorithm
- Java software and IDE environment is required to write the algorithms and debug the algorithms

Economic feasibility

- An amount of R2000 will be required to purchase the raspberry pi and a hard drive

Organizational feasibility

- A tutorial video will be made to demonstrate to the staff how the system works

Scheduling feasibility

- The system is supposed to be completed and fully tested within the next 6 weeks

Possible bottlenecks

- The buoy might experience mechanical breakdown during recording of data due to the harsh natural conditions in Antarctica

2.4. List of Requirements and Specifications

2.4.1. Compression subsystem

Requirements	Specifications
Accept data in time domain	Accept data in the form of a 2d array
Retain at least 25% of Fourier coefficients	A compression ratio of 4:1 to be used by DCT
Output data in frequency domain	DCT to perform a spectral analysis on data to obtain frequency components used for processing

2.4.2. Encryption subsystem

Requirements

- Encrypt data received from the gyroscope and accelerometer, which are 16 bits.
- Ensure inability of unauthorized reader to read the data

Specifications

- The encryption algorithm should take data of more than 16 bits.
- AES - 256 bits should be used, which is the most secured version

2.4.3. Transmission Subsystem

Requirements

- The processed data is transmitted over long distances
- The transmission speed is fast enough to transfer data as it is produced and processed in real time
- Transmission of data should be secured
- Minimal power should be used for transmission

Specifications

- The range of transmission should be over 100m
- The effective throughput of the transmission channel should be greater than 75kbps corresponding to the throughput of the IMU at maximum throughput
- The transmission channel should be compatible with 128-bit AES
- Peak data rate / power should be below 1mbps/10-500mW

2.4.4. Reception sub-system

Requirements

- The receiving device should process the data received, even at maximum speed
- The receptive device should be able to store the data

Specifications

- The throughput of the Raspberry pi should be greater than 75kbps, the transmission throughput, to enable reception even at maximum speed.
- The memory of the receptive device should be more than 1G to be able to collect and store the data as it is received.

2.5. Acceptance test procedures for specifications

2.5.1. Figures of merits

The final design will be validated if:

- When the IMU supplies data at maximum throughput, the design is able to collect the data, compress it, encrypt, transmit and recover at maximum throughput in a streaming way.
- The data is correctly encrypted at transmission and correctly decrypted at reception
- The power usage of the IMU is kept at a minimum while the system is efficiently working
- There is no loss of information along the way

2.5.2. Experiment design to test Figures of merit

To test the figures of merits, experiments were carried out in which the whole system was tested with different input samples and different forms of data. For these tests, three forms of data is used:

- Random data
- Sine waves data (periodic data)
- Fast producing data (high throughput)
- Slow throughput
- High frequency set of data with high Fourier coefficients
- Low frequency low Fourier coefficients data

Acceptable performance is defined as a situation where the data was processed by successfully compressing to at least 25% low Fourier coefficient, successful encryption, successful transmission, successful reception and decryption of the data.

3. VALIDATION USING SIMULATED OR OLD DATA

The ARM based digital IP is supposed to compress and encrypt this data so that it can be transmitted for processing. At the receiver's end, after the data is supposed to be decrypted and decompressed and at least the lower 25% of the Fourier coefficients are supposed to be recovered. A simulation of this system has been carried out, under which, different test procedures have been applied to test the performance of the system under various conditions. The results of these tests are documented in this report as well as the experiment designs that were used to carry out these tests. The test strategy for the system is shown in the diagram below:

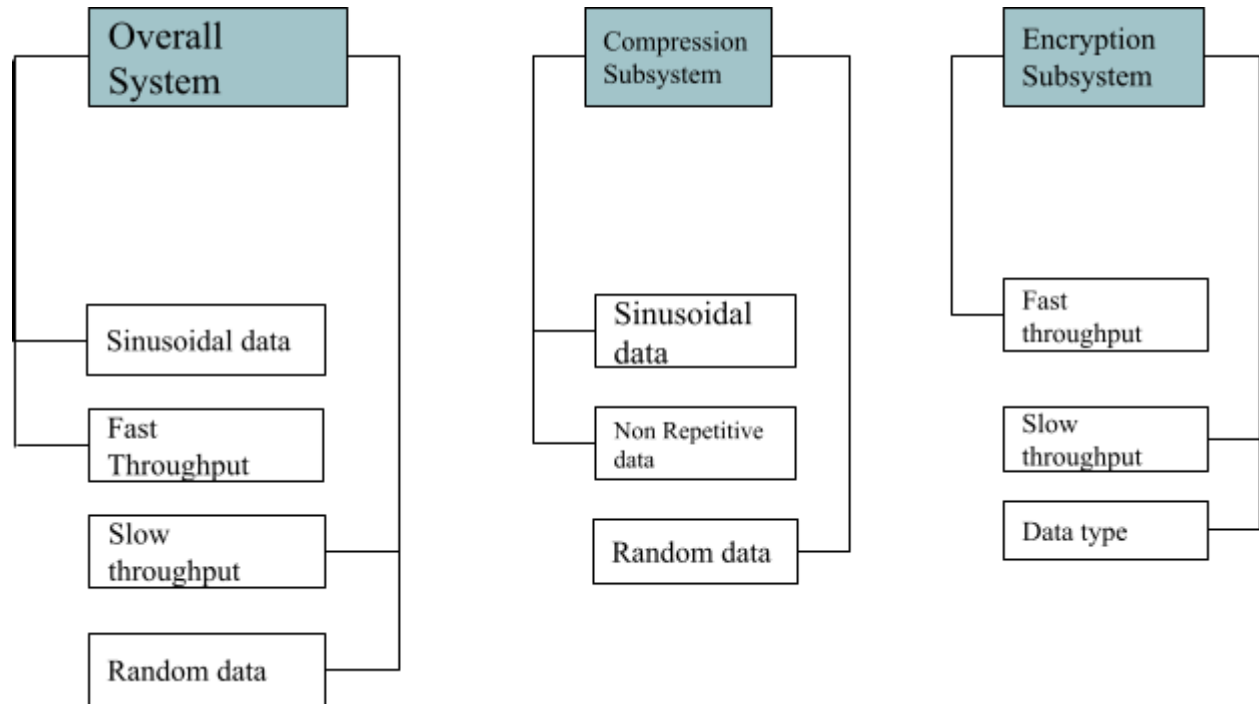


Figure 3.1 showing testing strategy for simulating system

3.1. Need for Simulation based validation

Using simulations as a means of validating the operation of the system provides the possibility to carry out a range of experimentations assuming that the system is valid and functioning as expected. This in return makes it possible to evaluate how the systems perform and handle different situations that are to be tested by the different experiments during the simulations. Also, unlike physical modelling for example, like having a copy of the IMU and the Sharc buoy, simulation modelling is computer based and allows subsystems to be tested separately.

Validation using simulations reassures that the performance of the system is in correlation to the actual purpose of the design of the actual system, which is to be used in the Sharc buoy. It provides enough information on how the system performs and handles different conditions and circumstances it might face for the task it is designed for, and therefore enables the optimization of the design to perform correctly under different conditions tested during our experimentations.

3.2. Steps

1. Decide on which ATP the experimentation will be testing
2. Determine the correct set of data that will effectively test the system and be used to validate if the system meets the ATP or not
3. Carry out the simulation experimentation
4. Record the result obtained from the simulation experimentation
5. Compare the results with the predicted output of the simulation
6. Conclude on whether the ATP was met or not. If passed, move on to the next ATP experimentation, and if failed, understand the reason why it failed, and Design additional ATP or Adjust the design to meet the ATP and retest the system with simulation for validation

3.3. Data Used for simulation

The data used for simulation is a .csv data file is shown in figure 3.2 shown below, corresponding to the format of a sample data from the IMU, which is used in the later stages of the design. Moreover, the .csv file used in this case is sample data from previous data provided by an IMU.

An ATP is that the compressed file should have a smaller size than the original file and for this to be explicitly determined, the .csv file was used because the file has to be sufficiently large such that the compression overhead does not make the compressed file larger. The .csv file used in this case is large enough to compensate and successfully meet the ATP requirements.

The data selected also made it possible to plot correct time domain waveforms, since the corresponding time for the data collection is included in the table. Therefore, proper analysis is made to determine if the system would meet the requirements.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
computer utc start 2018-09-19-03:57:21.506044																		
UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Faw, Pitch, Roll, DCM1, DCM2, DCM3, DCM4, DCM5, DCM6, DCM7, DCM8, DCM9, MagNED1, MagNED2, MagNED3																		
2018-09-19-03:57:21.71926	-0.14322271943092346	0.2232052981853485	0.123428575684561	0.15357686579227448	-0.30150956216812134	-9.795899391174316	0.00508350133895874	-0.001391227475										
2018-09-19-03:57:21.817726	-0.1420953253507614	0.21974442899227142	0.12350068542718887	0.15084362363815308	-0.30425748229026794	-9.779479026794434	0.005224071908742189	-0.001184811										
2018-09-19-03:57:21.917707	-0.14538213488429108	0.22430610918563843	0.12579116225242615	0.1569005296042398	-0.29820948046684265	-9.782461166381836	0.004827750381082296	-0.000948803										
2018-09-19-03:57:22.017733	-0.14438782632350622	0.2198091596364975	0.12465985864400864	0.1502079584121704	-0.30477917194366455	-9.774477005004883	0.005078970820294523	-0.001024232										
2018-09-19-03:57:22.117717	-0.14319761097431183	0.2232208251953125	0.1271720826625824	0.15469375252723694	-0.30872249603271484	-9.755866050720215	0.00510766853693247	-0.0009864562										
2018-09-19-03:57:22.217686	-0.14326296746730804	0.2206295630455017	0.12470496445894241	0.16080111265182495	-0.30884605646133423	-9.749424934387207	0.004865021910518408	-0.001166005										
2018-09-19-03:57:22.317733	-0.14428161084651947	0.22437554597854614	0.125805045397181647	0.15797586730063263	-0.3115788400173187	-9.74105453491211	0.004979013209670782	-0.0012970132										
2018-09-19-03:57:22.417797	-0.1465568244457245	0.2209047258234024	0.12577453255465338	0.16294534504413605	-0.3130553364753723	-9.739278793334961	0.004936043173074722	-0.001254770788										
2018-09-19-03:57:22.517726	-0.14545537531375885	0.22067420692443848	0.125833466464005548	0.16190342605113983	-0.3162236213684082	-9.728490829467773	0.00464474586077452	-0.001577646										
2018-09-19-03:57:22.617713	-0.14870640908952713	0.22311589460372925	0.12562707066535095	0.16106994450092316	-0.31497976183891296	-9.719582557678223	0.004643251188099384	-0.0008518676										
2018-09-19-03:57:22.717685	-0.148646583795547485	0.2255047273635864	0.124460887791156769	0.16258499228954315	-0.3218514621257782	-9.71753215789795	0.00438275060753689	-0.0008635037										
2018-09-19-03:57:22.817678	-0.14861410450935364	0.21983402967453003	0.12079749256372452	0.15081677174568176	-0.32272082567214966	-9.700324058532715	0.0040258122608065605	-0.00132395										
2018-09-19-03:57:22.917679	-0.14653174579143524	0.22326011955738068	0.12200228124856949	0.16515518724918365	-0.3259122567720032	-9.702316284179688	0.004099557176232338	-0.001032733										
2018-09-19-03:57:23.017675	-0.1465712934732437	0.22068377346992403	0.12327811121340613	0.163420221524006158	-0.3260989675807953	-9.694110870361328	0.003974360261413813	-0.001424150X										
2018-09-19-03:57:23.117681	-0.1454542875289917	0.22206811246086935	0.12207549365351486	0.1666800081729889	-0.31716904044151306	-9.68647494506836	0.003912205152093303	-0.001242601C										
2018-09-19-03:57:23.217676	-0.14863339607715607	0.22331994771957397	0.1218237453374672	0.17714177072048187	-0.3237636983394623	-9.685702323913574	0.00367445467049103	-0.0012060692										
2018-09-19-03:57:23.317665	-0.14878103137016296	0.2210192233324051	0.12191178649663925	0.1770520806312561	-0.32357630133628845	-9.676258087158203	0.0035345161963253	-0.0013650835C										
2018-09-19-03:57:23.417658	-0.14652030169963837	0.2213870288041528	0.12700651586055756	0.17423927783996064	-0.328918814605911885	-9.672946629933184	0.00335939361699348	-0.0015583423										
2018-09-19-03:57:23.517702	-0.14875587821006775	0.22103434801101685	0.12565478682518005	0.17514803025390635	-0.3213691711425781	-9.6646280288069429	0.0028928467340530157	-0.0016829C										
2018-09-19-03:57:23.617766	-0.14763009548187256	0.221539755963898	0.12569952011108398	0.17608954417705536	-0.3192858367960663	-9.660322189331055	0.002587320050224662	-0.001564605										
2018-09-19-03:57:23.717760	-0.1473641429424286	0.2266992824172974	0.1265038830041885	0.17370761930042535	-0.3249488986069763	-9.668174743652344	0.00274705179479127	-0.0014418582										
2018-09-19-03:57:23.817687	-0.1476208120584488	0.22158282995124	0.1289468367099762	0.17580513278484344	-0.3288641944122314	-9.657932323168457	0.0023824432864785194	-0.00177578648										
2018-09-19-03:57:23.917668	-0.14863012731075287	0.22673263983631134	0.12558147311210632	0.17027461386558075	-0.3208599060576173	-9.650251450439413	0.00209258209091187	-0.00183777C										
2018-09-19-03:57:24.017660	-0.14866749678134918	0.22332346439361572	0.12812024354904692	0.17353814840316772	-0.3253801465034485	-9.66348648071289	0.002019666361254454	-0.00188305C										

Figure 3.2 showing a sample of the simulated data

Below is the table corresponding to the different columns in the .csv file above.

X-axis acceleration	Signed 16-bit integer
Y-axis acceleration	Signed 16-bit integer
Z-axis acceleration	Signed 16-bit integer
X-axis angular velocity	Signed 16-bit integer
Y-axis angular velocity	Signed 16-bit integer
Z-axis angular velocity	Signed 16-bit integer

Table 3.1 showing size of each column in the .csv file used for simulation

3.3.1. Overall System data

To test the system, 4 types of data will be used. These four types of data will be used to access the performance of the system when both subsystems are combined. These are; fast throughput, slow throughput, sinusoidal data and random data.

1. Fast throughput

The process speed of the algorithms and transmission is tested when the data sent into the system is sent at maximum speed of transmission. This simulation is done by a python code that permits to adjust the speed of the data fed into the system, and the use of python socket programming to enable bit by bit data transmission.

2. Slow throughput

The rate at which data is being transmitted to the PI will be slowed down to check whether any changes will occur to the transmitted files. This will also later be used to measure the performance of the system.

3. Sinusoidal data

Sinusoidal data will be supplied to the system to ensure that data is compressed, encrypted and transmitted as well as decrypted and decompressed without any loss.

4. Random data

Random data will be used to check the ratio of the size of the file after it is decrypted as

compared to the size of the original file, when random data is fed at the initial stage of the system design.

3.3.2. Compression Subsystem data

Run Length Encoding (RLE) algorithm was used to compress the data [2]. This was changed from the initially suggested Discrete Cosine Transform because of the following reasons:

- RLE has a simple implementation. Therefore, it uses fewer processor power as the code contains less functions and mathematical equations
- RLE does not use third party libraries, therefore, the programmer has better control of the code and it can be easily changed as the programmer sees fit
- RLE converts compressed file to a format that can be easily used for further processing, which in this case is encryption
- RLE is a lossless compression algorithm, therefore, all data can be retrieved when the data is decrypted

In the first test, the aim was to determine the performance of the system when different types of data were used on the algorithm. 3 sets of data were to be considered, that is repetitive data, non repetitive data and random data.

1. Repetitive data (Sinusoidal Data) - Best Case Scenario

RLE counts the number of repeated data in a file and replaces all it with 2 bytes, that is, a single copy of the repeated data and a count of how many times it was repeated.

Therefore, sinusoidal data was considered to be the best case scenario for testing this algorithm as it will contain a lot of repetitive data.

2. Non Repetitive data - Average Case Scenario

On the other end of the spectrum, the worst case scenario would be data that is non repetitive. Therefore, non repetitive data was generated using a Python algorithm and the data was used to test the compression subsystem in the worst case scenario.

3. Random data - Worst Case Scenario

Realistically, it might not be possible to get perfectly sinusoidal data nor can we get non repetitive data. Therefore, random data was used to test performance of the compression algorithm. This was considered as the average case scenario.

The second test was to determine the performance of the algorithm as the size of the file increased. For this experiment, the simulated data was the closest thing to the data that will be produced by the actual IMU sensor on the Sharc buoy.

3.3.3. Encryption subsystem data:

1. Fast throughput

Having a set of data fed to the Encryption algorithm at a relatively high speed of transmission. The data is fed bit by bit using python socket programming. For this simulation, the data is transferred from the pc to the raspberry pi, and the speed of data transfer is made adjustable by a python code that performs data transfer and permits the speed to be adjusted as required for the simulation.

2. Slow throughput

Having data being fed into the Encryption algorithm at a relatively low speed of transmission. The data is fed using python socket programming using the same procedure as above to speed adjustment.

3. Data Type

Using data with different primitive data types such as strings , integers and analysing if the algorithm performs encryption correctly when the data type of the data are varied from the standard values types.

From the ATP listed above, the ATPs were as follows:

- When the IMU supplies data at maximum throughput, the design is able to collect the data, compress it, encrypt, transmit and recover at maximum throughput in a streaming way.
- The data is correctly encrypted at transmission and correctly decrypted at reception
- The power usage of the IMU is kept at a minimum while the system is efficiently working
- There is no loss of information along the way.

With these ATPs , the data selection for simulation permitted the confirmation of the validity of the ATP correspondingly as explained in the following points:

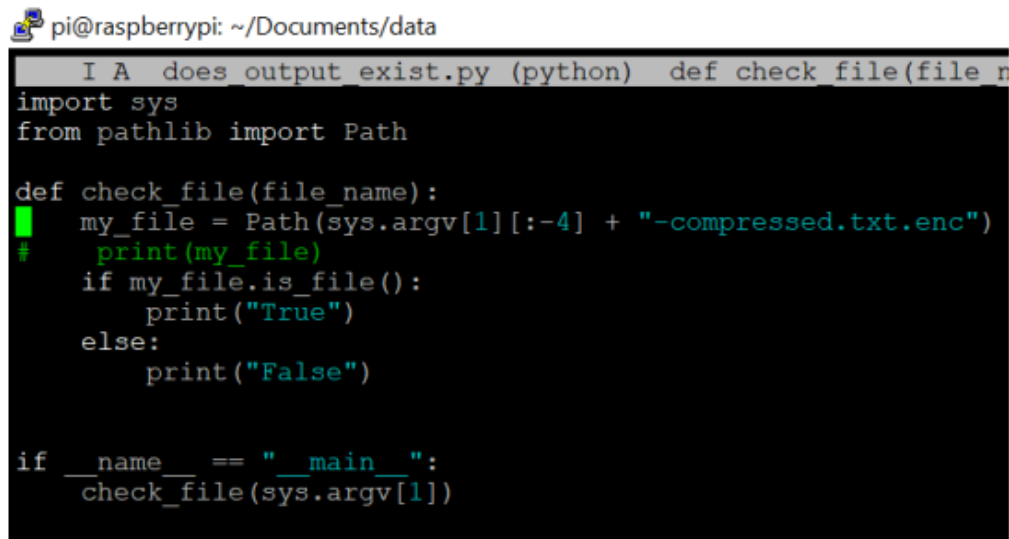
- Testing the correctness of the data manipulation when transmitted at maximum throughput throughout the system is a simulation needed to check the validity of the first ATP.
- Testing different data types for the encryption algorithm permits to identify any potential malfunction of the algorithm at the encryption stage and at the decryption stage
- The minimal number of computations performed during the full design caters for power usage. The time of performance is also recorded and analysed to compare the performance with the varying data selected.
- The data selection for the compression algorithms assures to cater any unforeseen loss of data and permits any identification of any potential data loss.

3.4. Experimental setups

3.4.1. Overall System

To check the functionality of the overall system, the system was tested with data exhibiting 4 distinct main characteristics. With regards to transmitting the data for these simulations, we used python socket programming as a way to mimic transmission of data from the sharec buoy to the pi. Using this methodology, the pc where the data is found, transmits the data bit by bit to the pi,

- Also, the latency of the system was put into test. In order to implement this, the native python time module was put into use. A timer was set before the system began and stopped right after. It should be noted that the timer was set only for the critical sections (compression and encryption) and not the trivial functions such as printing and reading files. Results were recorded and a graph plotted for a clearer representation. The output of the system was tested to make sure the data sent through the system has correctly gone through all the subsystems, and the output entails the output of all the different subsystems. The file expected at the output should have an “.compressed.txt.enc” extension at the output. A python script was implemented to do this. The input being the name of the original file



```
pi@raspberrypi: ~/Documents/data
I A does_output_exist.py (python) def check_file(file n
import sys
from pathlib import Path

def check_file(file_name):
    my_file = Path(sys.argv[1][:4] + "-compressed.txt.enc")
    # print(my_file)
    if my_file.is_file():
        print("True")
    else:
        print("False")

if __name__ == "__main__":
    check_file(sys.argv[1])
```

Figure 3.3 showing Python code to test output of system

- Sinusoidal data is used to test the overall system, to examine how the system performs with sinusoidally varying data. For this simulation, data was sent in the form of comma separated values(csv) to the system at a quite average speed. A timer is used to measure the time taken to process the data by the system till completion. The data starts off at the compression algorithm where it is compressed. Further, it goes through the encryption

algorithm, where it is encoded using the AES algorithm with key size of 256 and it is then transmitted to the pi.

At the pi, the data is decrypted and then decompressed using the same algorithms, which performs the inverse effects: decryption and decompression respectively. The received and processed data is retrieved and analyzed with respect to the initial data used

- **Fast Throughput Data:** In this simulation, the data was sent in the form of a csv file, where the data was fed to the system at a really high throughput of **up to 44.5Gbits/s**. This was made possible for simulation by using a python code that inputs data to our algorithms at adjustable speed of transmission.
The data was sent to the compression algorithm, followed by the encryption algorithm and after encrypted, it was transmitted by socket programming to the Raspberry pi. At this stage, the data is decrypted and decompressed and retrieved to be compared to the initial data used for the simulation and analysed for identification of potential differences between these files, and the time taken for completion of processing the data is recorded. From our research, the system should be able to handle data encryption up to 44.5Gbits/s, however, the system should not be able to transmit at this high throughput, because the maximum throughput the Raspberry pi can handle is a maximum of **300Mbits/s**.
- **Slow throughput Data:** In this simulation, the data was sent in the form of a csv file, where the data was fed to the system at a relatively very slow throughput of **20Mbits/s**. This was made possible for simulation by use of the same python code that inputs data to our algorithms at adjustable speed of transmission.
The data was sent to the compression algorithm, followed by the encryption algorithm and after encrypted, it was transmitted by socket programming to the Raspberry pi. At this stage, the data is decrypted and decompressed and retrieved to be compared to the initial data used for the simulation and analyzed for identification of potential differences between these files, and the time taken for completion of processing the data is recorded. The system is expected to be able to process the data at this speed without hindrance.
- **Random Data:** For this simulation, random data was generated and formed in the form in a .csv format. The data was fed to the system at an average speed and analysed. It starts off at the compression algorithm where it is compressed. Further, it goes through the encryption algorithm, where it is encoded using the AES algorithm with key size of 256 and it is then transmitted to the pi.
At the pi, the data is decrypted and then decompressed. The retrieved data is analysed and compared to the initial data used at the initial stage. The time taken by the process is also recorded. The system is expected to perform correctly and successfully.

3.4.2. Encryption System

To check the performance of the encryption algorithm, it was decided to test with data consisting of two distinct general characteristics: Throughput of the data and data type, i.e data strings and integers

- Varying the data Throughput: Data was fed to the encryption algorithm using a python code that provides a method to vary the speed at which the data is fed to the algorithm. Thereby allowing the variation of the speed of transmission, that is the throughput of the data. Data at a very high throughput of **up to 44Gbits/s** was fed to the algorithm and it was encoded by the algorithm and decoded. The resulting data was analysed and compared with the initial data used. The time taken for completion of the process was also recorded.

Very low throughput data at **20Mbits/s** was then fed to the algorithm, encoded and decoded and analysed following the same procedure explained for high throughput data above. The time taken for process completion is recorded. The expected result is that the algorithm should encrypt the data at both extremes throughput since they algorithm is designed to handle these speeds.

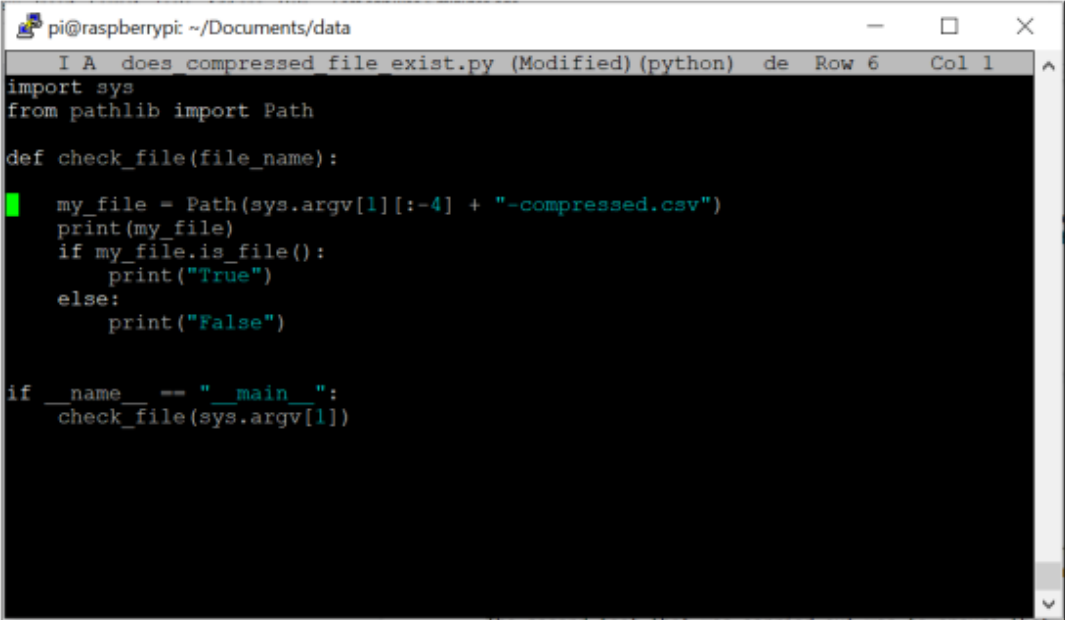
- Data type: Data of different primitive types were fed to the encryption algorithm. This data comprised of strings and integers and it is fed to the algorithm to be encrypted. The expected result of this is that the algorithm should encrypt every type of data irrespective of the data types. Thus, it should still be able to encrypt, if the data is a string or an integer. After decryption the data retrieved is compared to the initial data used.
- Some further test was made to check whether the encrypted files are reversible or not, because if not reversible, the encrypted data are lost and would not be of any use for the design.
- Checking the assertiveness of the algorithm was also undertaken. This means, making sure that no unwanted factor could decrypt the file and only the ones with the correct key could have access to the decrypted file. This is very important because if the encrypted files are accessible to be decrypted by any one, then the design would not have succeeded.
- The last test is to check the Entropy of the encrypted files. A high entropy signifies that the file is likely encrypted. We used a tool called Binwalk whose purpose is to determine the entropy of files. A file that has high entropy for instance means that the file is likely encrypted. Hidden threats in files can be identified by investigating the file's entropy. Therefore, hidden threats or suspicious scripts can be determined by measuring the randomness of the data in the file. It ranges from 0 to 8, where 0 means the file is not

random and 8 means the file is totally random, just like an encrypted file.

3.4.3. Compression System

To test for functionality of the compression algorithm 3 types of data were used to test the performance of the subsystem, that is, repetitive data, non repetitive data and random data. However, firstly, the test undertaken was to check whether the compression subsystem produced a compressed file. To carry out this procedure, the .csv file was compressed by the system and the system was made to produce an output file with the extension “compressed” suffix.

Therefore the way it operates is, a python file is used to check if there is a compressed version of any input data file, and the code will return true if the file is present, and False if no file is present. The expectation is that a file is present after it has gone through the system. Below is the python code performing this operation.



```
pi@raspberrypi: ~/Documents/data
I A does compressed file exist.py (Modified) (python) de Row 6 Col 1
import sys
from pathlib import Path

def check_file(file_name):
    my_file = Path(sys.argv[1][:-4] + "-compressed.csv")
    print(my_file)
    if my_file.is_file():
        print("True")
    else:
        print("False")

if __name__ == "__main__":
    check_file(sys.argv[1])
```

Figure 3.4 showing Code to check for compress file

After the check, the algorithm was tested with the following data types:

1. Repetitive Data

To test with repetitive data, a .csv file containing periodic sinusoidal data was obtained from a physics experiment that was carried out in a different course. The compression algorithm was applied to the data and the size of the compressed file was recorded. The compressed file was decompressed and the size of the file was recorded. An analysis was

then performed to calculate the compression ratio of the data by dividing the size of the compressed file with the size of the initial file. The size of the decompressed file was also compared with the size of the original file to see if any losses had occurred.

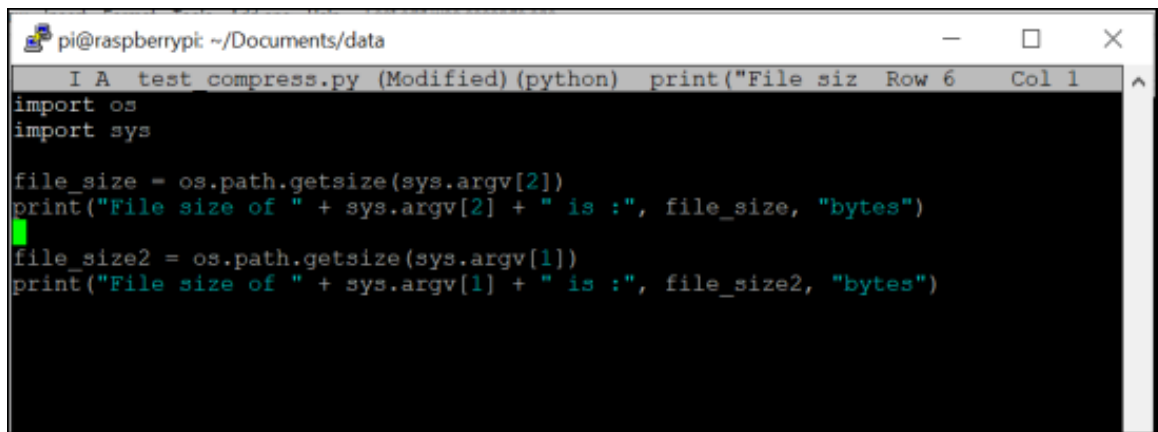
2. Non repetitive data

To test with non repetitive data, a python code was used to generate a list of distinct numbers between 1 and 100. The generated numbers were stored in a csv and the compression algorithm was applied to the file. The same procedure as in the repetitive data was used to obtain the compression ratio and compare the size of the decompressed file with that of the original file.

3. Random data

To test random data, a python code was used to generate a list of random numbers. The numbers were added to a csv file. The RLE algorithm was applied to the file. The same procedure as in the repetitive data was used to obtain the compression ratio and compare the size of the decompressed file with that of the original file.

4. A test was carried out to test whether the compressed output file was smaller in size than the original file. To verify this hypothesis, a python function in the test_comp.py file was used to compare the sizes of the files in bytes. The compressed file is expected to be significantly smaller.

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~/Documents/data'. The terminal shows the execution of a Python script named 'test_compress.py'. The code imports 'os' and 'sys' modules. It then uses 'os.path.getsize(sys.argv[2])' to get the size of a file specified by the second command-line argument, and prints it. Next, it uses 'os.path.getsize(sys.argv[1])' to get the size of a file specified by the first command-line argument, and prints it. The terminal output shows the file sizes in bytes.

```
pi@raspberrypi: ~/Documents/data
I A test_compress.py (Modified) (python) print("File siz Row 6 Col 1
import os
import sys

file_size = os.path.getsize(sys.argv[2])
print("File size of " + sys.argv[2] + " is :", file_size, "bytes")

file_size2 = os.path.getsize(sys.argv[1])
print("File size of " + sys.argv[1] + " is :", file_size2, "bytes")
```

Figure 3.5 showing Python code to calculate file for size comparison

In the second experiment, 5 data sets from Accelerometer readings were considered for the experiment. The data sets were compressed. The python code was used to record the time taken to compress and decompress the file. The initial file size, compressed file size and decompressed file size were recorded and used for analysis to determine performance of the system as the amount of data increased.

3.5. Results

3.5.1. Overall System:

1. Speed of System

For different data sizes, the following speeds recorded are shown in the table below.

Data size (Bytes)	Time(s)
5709	0.027382789
56890	0.15383947
628688	0.6727837297
1276838	1.27097848
7973489	6.5698748

Table 3.2 showing speed results of overall system

The following graphs were obtained from the graph above

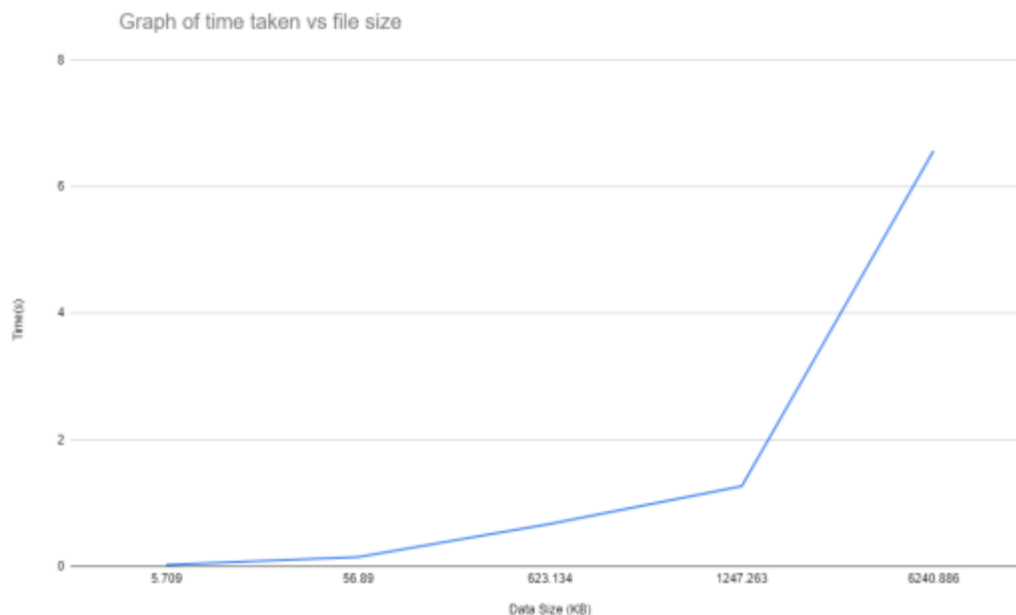
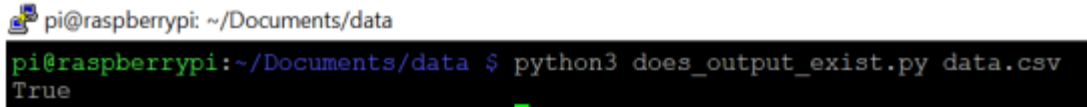


Figure 3.6 showing time taken vs file size of compressed file

From the graph above, it was observed that there was an increase in process time as the size of the data set increases when put through the overall system.

2. Output present:

Using the python script, we checked to determine if the system effectively produces an output. Using the command below:

A terminal window on a Raspberry Pi. The prompt is 'pi@raspberrypi: ~/Documents/data'. The command 'python3 does_output_exist.py data.csv' has been executed, and the output 'True' is displayed on the next line.

```
pi@raspberrypi: ~/Documents/data
pi@raspberrypi:~/Documents/data $ python3 does_output_exist.py data.csv
True
```

Figure 3.7 showing that the system produced an output file

Therefore, the system produced an output indeed.

3. Sinusoidal Data:

The data retrieved from the system after simulating with sinusoidally varying data was identical to the initial data inserted at the initial stage of the system. The time taken to process data was roughly identical to the time to process standard data.

4. Fast throughput

When data is fed to the system at a high throughput of 300Mbps/s, the system performs well as expected and transmits the data at the corresponding throughput. Beyond this throughput, the processing speed of the system did not increase, neither did the transmission rate. However, it was observed that the subsystems performed with little or no change at very high throughputs, though the transmission and reception of the data did not exceed 300Mbps/s.

5. Slow throughput

When data is fed to the system at a low throughput of around 20Mbps/s, the system performs well as expected and transmits the data at the corresponding throughput. The system did not seem to have any minimal throughput. As slow as the data can be received by the system, the data is still going to be processed regardless of how slow data is fed into the system.

6. Random data

When the data used is random data, the system still performs well and fulfill its duty to completion successfully. Comparing the recorded standard time with the time when there is random data, it is observed that the time taken is roughly the same.

3.5.2. Encryption System

1. Recovery test:

In the picture below you can see the original data before it is encrypted

```

1 computer utc start 2018-09-19-04:22:31.529971
2 UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM
3 2018-09-19-04:22:31.793573 -0.34008175134658813 0.35862863063812256 0.30060529708862305 0.23097454011440277 -0.3260
4 2018-09-19-04:22:31.893383 -0.342180460691452 0.36213183403015137 0.3067106604576111 0.23146139085292816 -0.3308394
5 2018-09-19-04:22:31.993358 -0.34227075192596436 0.35860307625672485 0.3042473795829785 0.23394353067765214 -0.32825
6 2018-09-19-04:22:32.093380 -0.33891722559928894 0.3597724735736847 0.3056625723838806 0.21971283853054047 -0.331705
7 2018-09-19-04:22:32.193370 -0.34226855635643005 0.3586834967136383 0.30424585938453674 0.21944978833198547 -0.32590
8 2018-09-19-04:22:32.293361 -0.34003153443336487 0.36091920733451843 0.3005756437778473 0.21391789615154266 -0.32591
9 2018-09-19-04:22:32.393351 -0.3410453200340271 0.3643854856491089 0.3029782474040985 0.21461999416351318 -0.3265402
10 2018-09-19-04:22:32.493357 -0.3410690426826477 0.363239586353302 0.3029923439025879 0.21591301262378693 -0.31855210
11 2018-09-19-04:22:32.593398 -0.34011411666870117 0.3563460409641266 0.30314040184020996 0.19787898659706116 -0.32240
12 2018-09-19-04:22:32.693342 -0.34121447801509966 0.3552418649196625 0.3081103265285492 0.2025754451751709 -0.3199371
13 2018-09-19-04:22:32.793340 -0.3389686048030853 0.35748231410980225 0.30569320917129517 0.19914020597934723 -0.31245
14 2018-09-19-04:22:32.893385 -0.34227919578552246 0.3575487434864044 0.30676835775375366 0.18888604640960693 -0.31485
15 2018-09-19-04:22:32.993407 -0.3422193229198456 0.36097484827041626 0.3042171597480774 0.18875494599342346 -0.318521
16 2018-09-19-04:22:33.093386 -0.3422207236289978 0.36097633838653564 0.30421796441078186 0.18590101599693298 -0.31363
17 2018-09-19-04:22:33.193342 -0.3411988317966461 0.35638344287872314 0.3068430423736572 0.18076957762241364 -0.310221
18 2018-09-19-04:22:33.293366 -0.3390343189239502 0.3551810681819916 0.3032151758670807 0.18188047409057617 -0.3065259
19 2018-09-19-04:22:33.393342 -0.34009069204330444 0.35749325156211853 0.30312711000442505 0.17157606780529022 -0.3034
20 2018-09-19-04:22:33.493335 -0.3388703763484955 0.36319708824157715 0.3018614947795868 0.1723051518201828 -0.3019212
21 2018-09-19-04:22:33.593378 -0.3401644825935364 0.35518717765808105 0.29939576983451843 0.1633971929550171 -0.297162
22 2018-09-19-04:22:33.693340 -0.33793994784355164 0.35402682423591614 0.3057979345321655 0.16524161398410797 -0.28876
23 2018-09-19-04:22:33.793317 -0.3401374816894531 0.3540785312652588 0.30818718671798706 0.15837541222572327 -0.294530
24 2018-09-19-04:22:33.893314 -0.33904027938842773 0.35405370593070984 0.30699390172958374 0.15909089148044586 -0.2921
25 2018-09-19-04:22:33.993331 -0.33902448415756226 0.3551957309246063 0.30572694540023804 0.1572961062192917 -0.283139
26 2018-09-19-04:22:34.093327 -0.3379184305667877 0.35517433285713196 0.3057858347892761 0.1518937349319458 -0.2851741
27 2018-09-19-04:22:34.193365 -0.3390256464481354 0.35519564151763916 0.3057270646095276 0.15245196223258972 -0.283565
28 2018-09-19-04:22:34.293412 -0.33785420656204224 0.358607679605484 0.3044903874397278 0.14724959433078766 -0.2662251
29 2018-09-19-04:22:34.393352 -0.33904117345809937 0.3563167154788971 0.2994458079338074 0.15416912734508514 -0.271390
30 2018-09-19-04:22:34.493330 -0.3378465175628662 0.3586127758056123 0.3057440221300662 0.14544525742530823 -0.2621203

```

Figure 3.8 showing data before it was encrypted

Using the python code for encryption below, the data set is encoded.

```

import sys
from cryptor import Encryptor

#key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02]j\xdf\
key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02]j\xdf\xcb\xc4\x94\x9d(\x9e'
encryptor1 = Encryptor(key)

encryptor1.encrypt_file(sys.argv[1])

```

Figure 3.8 showing python code to begin encoding data

And the resulting encrypted file is shown below,



figure 3.9 showing encoded data

Now, running the encoded file through the decryption algorithm to check for reversibility, the code for decryption is seen in the picture below



figure 3.10 showing python code to start decrypting data

And the file obtained from decrypting the encrypted file is shown below


```

1 | computer utc start 2018-09-19-04:22:31.529971
2 | UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM
3 | 2018-09-19-04:22:31.793573 -0.34008175134658813 0.35862863063812256 0.30060529708862305 0.23097454011440277 -0.33260
4 | 2018-09-19-04:22:31.893383 -0.342180460691452 0.36213183403015137 0.3067106604576111 0.23146139085292816 -0.3308394
5 | 2018-09-19-04:22:31.993358 -0.34227075192596436 0.35860507625672485 0.3042473795829785 0.23394355067765214 -0.32825
6 | 2018-09-19-04:22:32.093380 -0.33891722559928894 0.3597724735736847 0.3056625723838806 0.21971283853054047 -0.331705
7 | 2018-09-19-04:22:32.193370 -0.34226855635643005 0.3586834967136383 0.30424585938453674 0.21944978833198547 -0.32590
8 | 2018-09-19-04:22:32.293361 -0.34003153443336487 0.36091920733451843 0.3005756437778473 0.21391789615154266 -0.32591
9 | 2018-09-19-04:22:32.393351 -0.3410453200340271 0.3643854856491089 0.3029782474040985 0.21461999416351318 -0.3265402
10 | 2018-09-19-04:22:32.493357 -0.3410690426826477 0.363239586353302 0.3029923439025879 0.21591301262378693 -0.31855210
11 | 2018-09-19-04:22:32.593398 -0.34011411666870117 0.3563460409641266 0.30314040184020996 0.19787898659706116 -0.32240
12 | 2018-09-19-04:22:32.693342 -0.34121447801509966 0.3552418649196625 0.3081103265285492 0.2025754451751709 -0.3199371
13 | 2018-09-19-04:22:32.793340 -0.3389686048030853 0.35748231410980225 0.30569320917129517 0.19914020597934723 -0.31245
14 | 2018-09-19-04:22:32.893385 -0.34227919578552246 0.3575487434864044 0.30676835775375366 0.18888604640960693 -0.31485
15 | 2018-09-19-04:22:32.993407 -0.3422193229198456 0.36097484827041626 0.3042171597480774 0.18875494599342346 -0.318521
16 | 2018-09-19-04:22:33.093386 -0.3422207236289978 0.36097633838653564 0.30421796441078186 0.18590101599693298 -0.31363
17 | 2018-09-19-04:22:33.193342 -0.3411988317966461 0.35638344287872314 0.3068430423736572 0.18076957762241364 -0.310221
18 | 2018-09-19-04:22:33.293366 -0.3390343189239502 0.3551810681819916 0.3032151758670807 0.18188047409057617 -0.3065259
19 | 2018-09-19-04:22:33.393342 -0.34009069204330444 0.35749325156211853 0.30312711000442505 0.17157606780529022 -0.3034
20 | 2018-09-19-04:22:33.493335 -0.3388703763484955 0.36319708824157715 0.3018614947795868 0.1723051518201828 -0.3019212
21 | 2018-09-19-04:22:33.593378 -0.3401644825935364 0.35518717765808105 0.29939576983451843 0.1633971929550171 -0.297162
22 | 2018-09-19-04:22:33.693340 -0.33793994784355164 0.35402682423591614 0.3057979345321655 0.16524161398410797 -0.28876
23 | 2018-09-19-04:22:33.793317 -0.3401374816894531 0.35407853112652588 0.30818718671798706 0.15837541222572327 -0.294530
24 | 2018-09-19-04:22:33.893314 -0.33904027938842773 0.35405370593070984 0.30699390172958374 0.15909089148044586 -0.2921
25 | 2018-09-19-04:22:33.993331 -0.33902448415756226 0.3551957309246063 0.30572694540023804 0.1572961062192917 -0.283139
26 | 2018-09-19-04:22:34.093327 -0.3379184305667877 0.35517433285713196 0.3057858347892761 0.1518937349319458 -0.2851741
27 | 2018-09-19-04:22:34.193365 -0.3390256464481354 0.35519564151763916 0.3057270646095276 0.15245196223258972 -0.283565
28 | 2018-09-19-04:22:34.293412 -0.33785420656204224 0.358607679605484 0.3044903874397278 0.14724959433078766 -0.2662251
29 | 2018-09-19-04:22:34.393352 -0.33904117345809937 0.3563167154788971 0.2994458079338074 0.15416912734508514 -0.271390
30 | 2018-09-19-04:22:34.493338 -0.3378465175628662 0.3586127758026123 0.3057440221300662 0.14544525742530823 -0.2621203

```

Figure 3.11 showing decrypted data

And from the picture above, the decrypted file corresponds to the original file with no change of data.

2. The integrity of the key

an attempt was made to decrypt the encrypted file with a key not corresponding to the encrypting key. The new key is shown below

```

key = b'0123456789987654'
enc = Encryptor(key)

```

Figure 3.12 showing an attempt to decrypt using different key

Below is a picture showing the original encrypted file



Figure 3.13 showing encrypted file before decrypting with different key

And the python code used to decrypt the file containing the new and incorrect key is shown in the picture below

```

IMU_prac-master > decrypt.py > ...
1  import sys
2  from encryptor import Encryptor
3
4  #key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02}j\xdf\
5
6  key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02}j\xdf\xcb\xc4\x94\x9d(\x9e'
7  encryptor1 = Encryptor(key)
8
9  encryptor1.decrypt_file(sys.argv[1])
10

```

Figure 3.14 showing file being decrypted with different key

And the corresponding output file from the decryption algorithm with the new key is shown below.



Figure 3.15 showing data after being decrypted with different key

Therefore, the file has not been decrypted with an incorrect key as designed and expected.

3. Varying data throughput:

When the data is transferred a bit at a time to the encryption algorithm, the algorithm performs correctly under maximum throughput of 44Gbits/s, and encrypts and decodes at the corresponding speed. However, it is observed that when the algorithm receives large chunks of data at very high speed, the encryption algorithms might not sustain the speed of encryption and eventually takes time to encode and decode large chunks of data at very high speed. Therefore, the performance of the algorithm at high throughput of data varies as a function of the size of the chunks of data sent to the encryption algorithm, as shown in the graph of data versus time for the overall system.

Furthermore, at slow speed, that is low throughput of about 20Mbits/s, the algorithm performs correctly at the corresponding throughput.

4. Data Type

When inserting data of different primitive types such as integers and strings the algorithm performs well regardless of the data type of the input data, and performs at the same rate with standard data csv values

5. Entropy test

The results obtained for entropy and Randomness online for a normal csv file and an encrypted csv file are shown below., First, the picture of the standard csv file :

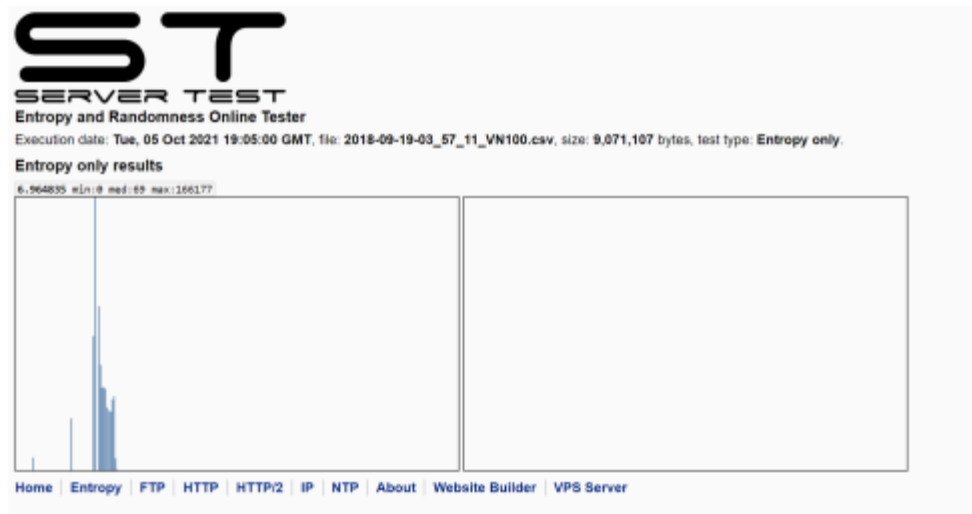


Figure 3.16 showing Entropy for .csv file

Following is the picture for the encrypted csv file

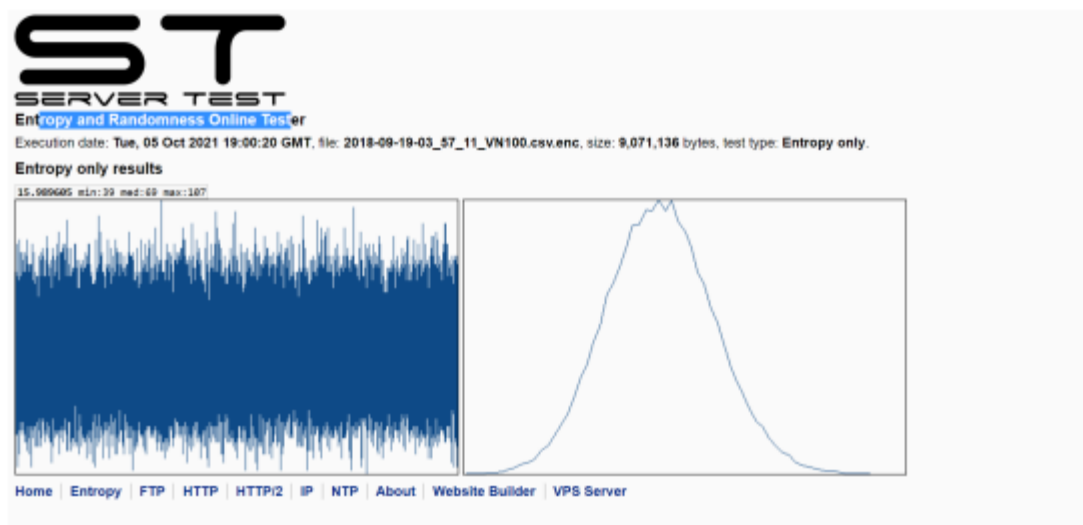
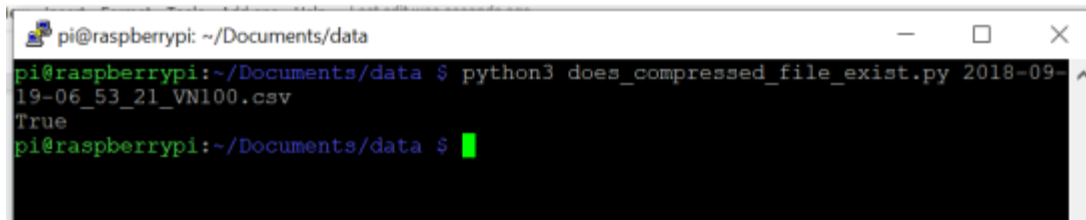


Figure 3.17 showing Entropy for encrypted csv file

3.5.3. Compression system

1. Presence of Compressed File

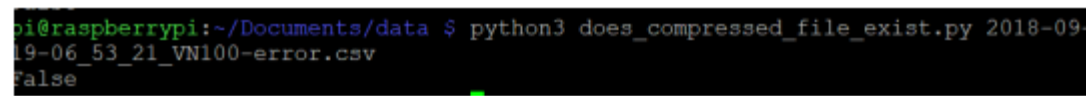
A python code was used to check whether a compression file was present. A screenshot of the command on the command line is shown below.

A terminal window on a Raspberry Pi. The prompt is 'pi@raspberrypi: ~/Documents/data'. The command 'python3 does_compressed_file_exist.py 2018-09-19-06_53_21_VN100.csv' is entered. The output is 'True'.

```
pi@raspberrypi: ~/Documents/data
pi@raspberrypi:~/Documents/data $ python3 does_compressed_file_exist.py 2018-09-19-06_53_21_VN100.csv
True
pi@raspberrypi:~/Documents/data $
```

Figure 3.18 showing a presence check for a compressed file

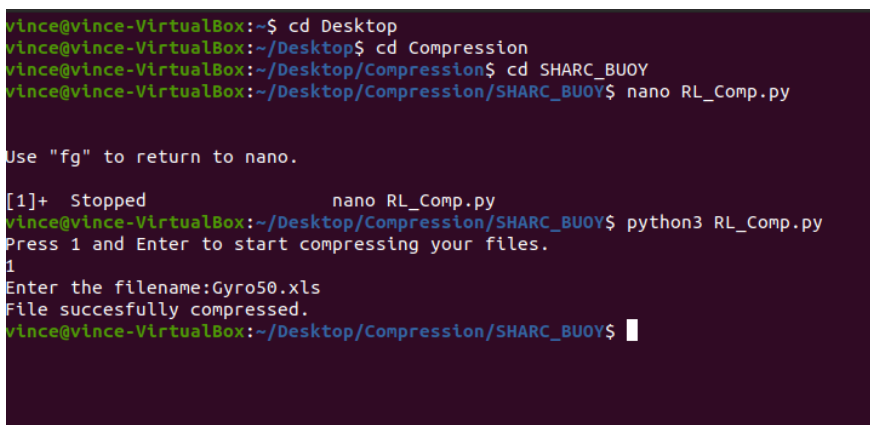
The output of the code is True, indicating the presence of a compressed file. Furthermore, running the python when there is no compressed file, the picture below illustrates the output of the code.

A terminal window on a Raspberry Pi. The prompt is 'pi@raspberrypi: ~/Documents/data'. The command 'python3 does_compressed_file_exist.py 2018-09-19-06_53_21_VN100-error.csv' is entered. The output is 'False'.

```
pi@raspberrypi:~/Documents/data $ python3 does_compressed_file_exist.py 2018-09-19-06_53_21_VN100-error.csv
False
```

Figure 3.19 showing an absent compressed file

From the figure above, it is shown that the system will produce False if the compressed file is absent.

A terminal window in a VirtualBox environment. The user navigates through directories: Desktop, Compression, and SHARC_BUOY. They use nano to edit RL_Comp.py. The program prompts for a filename, and 'Gyro50.xls' is entered. The output is 'File succesfully compressed.'.

```
vince@vince-VirtualBox:~$ cd Desktop
vince@vince-VirtualBox:~/Desktop$ cd Compression
vince@vince-VirtualBox:~/Desktop/Compression$ cd SHARC_BUOY
vince@vince-VirtualBox:~/Desktop/Compression/SHARC_BUOY$ nano RL_Comp.py

Use "fg" to return to nano.

[1]+  Stopped                  nano RL_Comp.py
vince@vince-VirtualBox:~/Desktop/Compression/SHARC_BUOY$ python3 RL_Comp.py
Press 1 and Enter to start compressing your files.
1
Enter the filename:Gyro50.xls
File succesfully compressed.
vince@vince-VirtualBox:~/Desktop/Compression/SHARC_BUOY$
```

Figure 3.20 showing the input to compression algorithm

```

1 | "-171,-422,-712"
2 | "2"
3 | "2"
4 | "-134,52,-454"
5 | "2"
6 | "2"
7 | "1,-68,467"
8 | "2"
9 | "2"
10 | "-436,-932,-593"
11 | "2"
12 | "2"
13 | "127,393,61"
14 | "2"
15 | "-15,72,365"
16 | "2"
17 | "2"
18 | "41,-72,-28"
19 | "2"
20 | "2"
21 | "2"
22 | "510,-260,-1059"
23 | "2"
24 | "2"
25 | "149,-707,-618"
26 | "2"
27 | "2"
28 | "-363,927,324"

```

figure 1.21 showing compressed file

```

vince@vince-VirtualBox:~$ cd Desktop
vince@vince-VirtualBox:~/Desktop$ cd Compression
vince@vince-VirtualBox:~/Desktop/Compression$ cd SHARC_BUOY
vince@vince-VirtualBox:~/Desktop/Compression/SHARC_BUOY$ nano RL_Comp.py

Use "fg" to return to nano.

[1]+  Stopped                  nano RL_Comp.py
vince@vince-VirtualBox:~/Desktop/Compression/SHARC_BUOY$ python3 RL_Comp.py
Press 1 and Enter to start compressing your files.
1
Enter the filename:Gyro50.xls
File succesfully compressed.
vince@vince-VirtualBox:~/Desktop/Compression/SHARC_BUOY$ python3 RL_decomp.py
Press 1 and Enter to start decompressing your files.
1
Enter the filename:Compressed.csv
File succesfully decompressed.
vince@vince-VirtualBox:~/Desktop/Compression/SHARC_BUOY$

```

figure 3.21 showing input to decompression algorithm

2. Comparisons of Files sizes

A python code was run to compare the file size of the compressed file with that of the original. A sample of the output of the code is shown in the figure below;

```

pi@raspberrypi: ~/Documents/data
pi@raspberrypi:~/Documents/data $ python3 test_compress.py mlb_players.csv mlb_p
layers-compressed.csv
File size of mlb_players-compressed.csv is : 15446 bytes
File size of mlb_players.csv is : 56890 bytes

```

Figure 3.22 showing a screenshot of file comparison algorithm

The picture above illustrates the difference in size of the compressed files and the original file. Their sizes are different, with the compressed file being of a smaller size than the original file's size. Data was used to calculate the compression ratio of the files which was used to measure performance of the algorithm.

After the experiment was performed for repetitive data, the following results were obtained:

Size of original file = 4.3kb

Size of compressed file = 3.2kb

Size of decompressed file = 3.5kb

$$\text{Compression ratio} = \frac{\text{original file}}{\text{compressed file}} = \frac{4.3}{3.2} = 1.34$$

Comment: File was compressed to a size 74% to that of the original size and the decompressed file retained 78% of the original data.

3. Non repetitive data

ARepetitive Data

fter the experiment was performed for non-repetitive data, the following results were obtained:

Size of original file = 3.1kb

Size of compressed file = 3.0kb

Size of decompressed file = 3.0kb

$$\text{Compression ratio} = \frac{\text{original file}}{\text{compressed file}} = \frac{3.1}{3.0} = 1.03$$

Comment: File was compressed to a size 96.77% to that of the original size and the decompressed file retained 96.77% of the original data.

4. Random data

After the experiment was performed for random data, the following results were obtained:

Size of original file = 8.7kb

Size of compressed file = 7.6kb

Size of decompressed file = 8.1kb

$$\text{Compression ratio} = \frac{\text{original file}}{\text{compressed file}} = \frac{8.7}{7.6} = 1.14$$

Comment: File was compressed to a size 87% to that of the original size and the decompressed file retained 93% of the original data

5. Time taken to compress data

Using different data file sizes, the algorithm's performance and speed was taken and compared. The following table represents the time taken by the compression algorithm to compress files in a range of sizes.

Data Set	Initial File Size	Compressed file Size	Time	Compression Ratio
50	5 736 bytes	5 612 bytes	0.00257979s	1.02209551
500	10 352 bytes	10 016 bytes	0.123458774s	1.033546326
1000	30 564 bytes	29 764 bytes	0.61289873s	1.026878108
2000	63 245 bytes	64 782 bytes	1.16898979s	0.9762742737
5000	124 376 bytes	145 393 bytes	6.18987734s	0.855446961

Table 3.4 showing file size changes, compression ratio and time taken to compress data

The graph below represents how the time taken for compression varies with the file's size

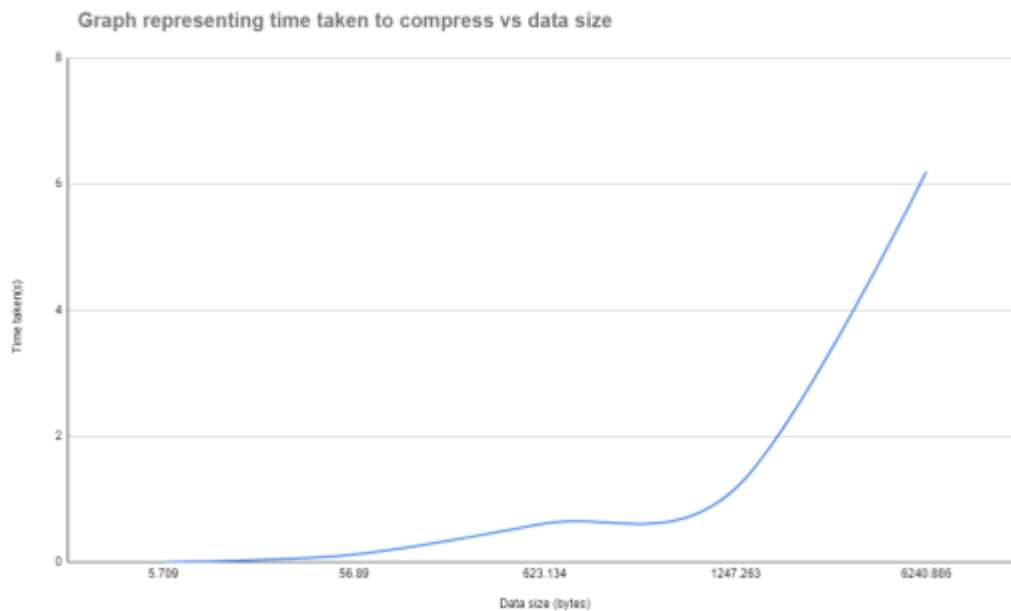


Figure showing time taken to compress data vs data size

From the graph, we observe that the time taken for compressing a file by the compression algorithm varies with size of the file. We notice an increase in the time taken for compression as the size increases .

Scope and Limitation of experiment:

Theoretically, RLE is a lossless algorithm, therefore, 100% of the data was supposed to be retained. However, the decompressed data was within acceptance range, therefore, the reason as to why all the data was not covered in this writing

4. VALIDATION USING A DIFFERENT-IMU

This section will focus on sending data from an IMU sensor to the compression and encryption subsystem. An ICM 20948 sensor will be used to create data by performing different motions on the sensor. This data will be used to measure the performance of the system and can then be extrapolated to fit that of the actual ICM 20649 sensor which will be used on the SHARC BUOY. The data will be passed to the system, that is, the compression and encryption subsystem, and the performance of these subsystems will be measured by varying the size and throughput of the data.

4.1. Need for hardware-based validation

Hardware validation is necessary because it helps experiment our design with an actual target unit, which may be a combination of sensors, combination of IP, transfer of data from one device to another, and therefore helps confirm that the target device has a high chance of working effectively as designed to do. It serves as an early window for functional interoperability and represents a prototype of the final design.

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. 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 20948	ICM 20649
Is a 9-axis device	Is a 6-axis device

3-Axis Compass with a wide range to ± 4900	No 3-axis compass
Has Android support	Has a 4kB FIFO buffer enables the applications processor to read the data in bursts
Auxiliary I2C interface for external sensors	Host interface: 7 MHz SPI or 400 kHz I2C

Table 4.1 showing the differences between ICM 20649 and ICM 20948

4.2. Steps

This system is designed to be used in the poles in Sharc buoys to capture various forms of data such as wave dynamics. The 2 conditions in these environments are pretty rough. The sensors would be in constant movement due to the wave, ice and wind movement. The first aspect is the harsh weather.

- Tests were done with the IMU in a freezer. Although the temperatures may not be the same, this would give an almost ideal representation of the weather in the poles.
- The second aspect was the vigorous movements that are common in the poles. Tests were done while shaking the IMU to mimic the movements. The IMU responded well by giving accurate results on the accelerometer and gyroscope that reflected that movement.
- The third aspect was the magnetic forces present in the poles. For this, we drove an electromagnet with current to create a magnetic field. This would give an idea of the magnetic forces that the IMU would have to deal with. The data produced by the IMU accurately represented this.

4.3. Data and steps followed

The decided to use actual data from the IMU for the experiments we will further carry out. This is because it helps us in two aspects: Making sure we get correct values from the IMU, thereby confirming us that the IMU is in good state and produces accurate data, and secondly, Confirms us that there is proper communication between the IMU and Raspberry pi so as not to be exposed to unforeseen miscommunications between the two hardware device, since we would have had confirmation already.

In an attempt to collect data from the IMU connected to the Raspberry pi, a series of validation tests were taken to confirm the correct performance of the IMU and the IMU + Raspberry pi system in general. The validation test are as follows:

1. Connection to the Raspberry pi

To check whether the IMU was properly connected to the RPI, the following command was run on the RPi:

```
sudo i2cdetect -y 1
```

Figure 4.1 showing that the PI was connected

The following result was obtained:

```
pi@raspberrypi:~ $ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  29  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  48  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  5c  --  --  --  --
60:  --  --  --  --  --  --  --  68  --  --  --  --  --  --  --
70: 70  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

Figure 4.2 showing that the IMU was connected and the PI was receiving data

This confirmed that the IMU was indeed connected to the RPi and that data would be collected from the surrounding.

2. Testing the sensor

The IMU is going in a buoy in the ocean. Therefore, to create a similar motion, the sensor was oscillated in various positions to create the same motion as ocean waves as data was being recorded. This data was used to test the system as it could easily be extrapolated to data from the actual IMU on the SHARC BUOY.

3. Testing IMU+PI

To test if the sensor was working, data from some of the sensors was collected. The IMU was connected to the Rpi and was configured using the LPS22HB.python library that was supplied in the IMU datasheet.

The output produced was then examined to ensure that the right temperature was recorded. Since the IMU-20949 was inside a room, the expected temperature was to be around room temperature of 22-25 degrees Celsius. The temperature was around the desired temperature. This confirmed that the IMU worked as expected

Moreover, the values obtained from the gyroscope and accelerometer were obtained from the IMU by running the code shown below. As shown, the output was formatted to be produced in a sort of .csv file that could then output redirect the values to a file and plot the values obtained from the simulations. This is shown below:

```
File ICM20948.py not changed so no update needed
pi@raspberrypi:~/EEE3097/Python-File-Encryptor/IMU/Sensor-IMU- $ python3 ICM20948.py

1,1,3

1,0,0

0,-2,0

2,-2,1

0,0,1

3,0,2

3,-2,2

3,0,1

2,-3,3

2,-1,2

3,-2,1

1,0,-1
```

Figure 4.3 showing output of the sensor after ICM20498.py was ran

After establishing that the sensor was connected correctly and sending some data to the PI, the next was to test if the sensors were working. Two sensors were used for this test, Accelerometer and Gyroscope. The expected output was 3-axis data for both sensors. The IMU is known to automatically handle motion profiles especially using the accelerometer which measures proper acceleration it experiences relative to freefall and 4 acceleration felt by people and objects. To do this, the acceleration sensor was put to use. A python script, ICM20948.py, was used to collect data from using the acceleration sensor and the data as separated such that the first run was when the sensors were stationary or at rest then the second test was when the sensor was exposed to some movement (i.e shaking) graph were drawn from the generated or collected data as shown below.

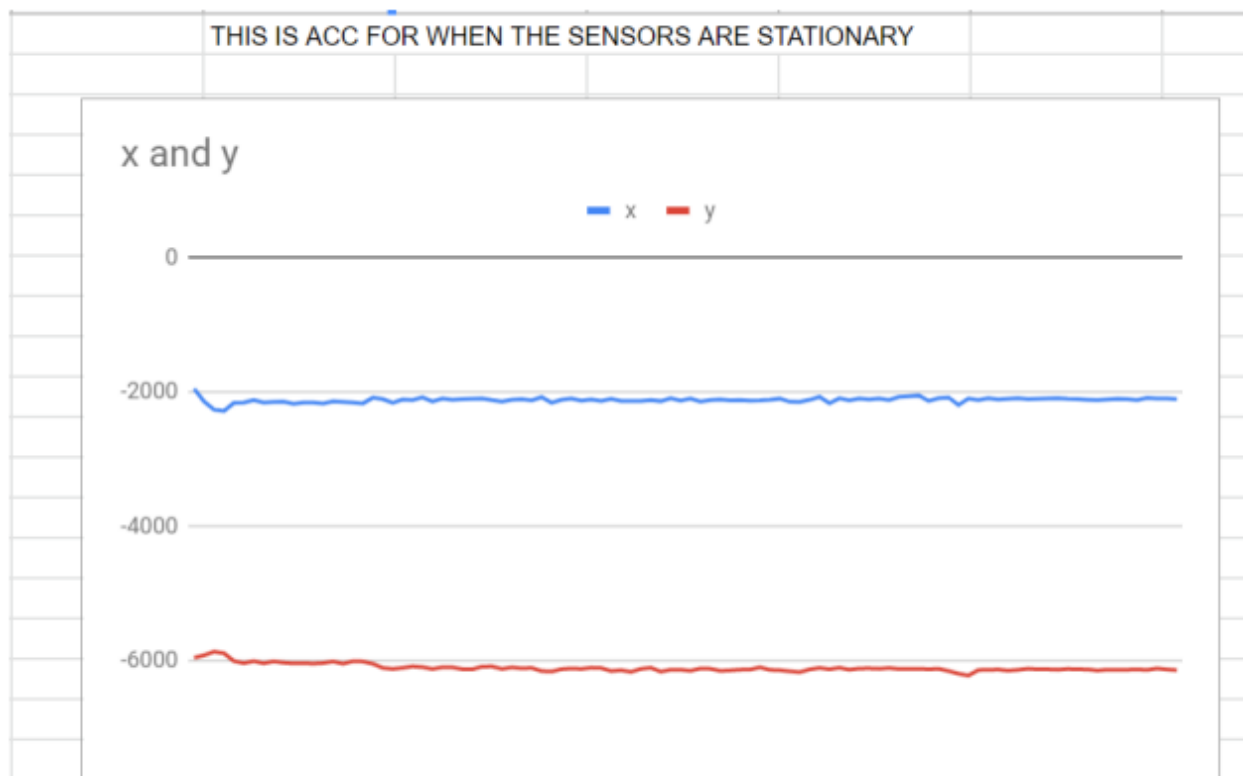


Figure 4.4 showing a plot Accelerometer readings for the X, Y and Z axis

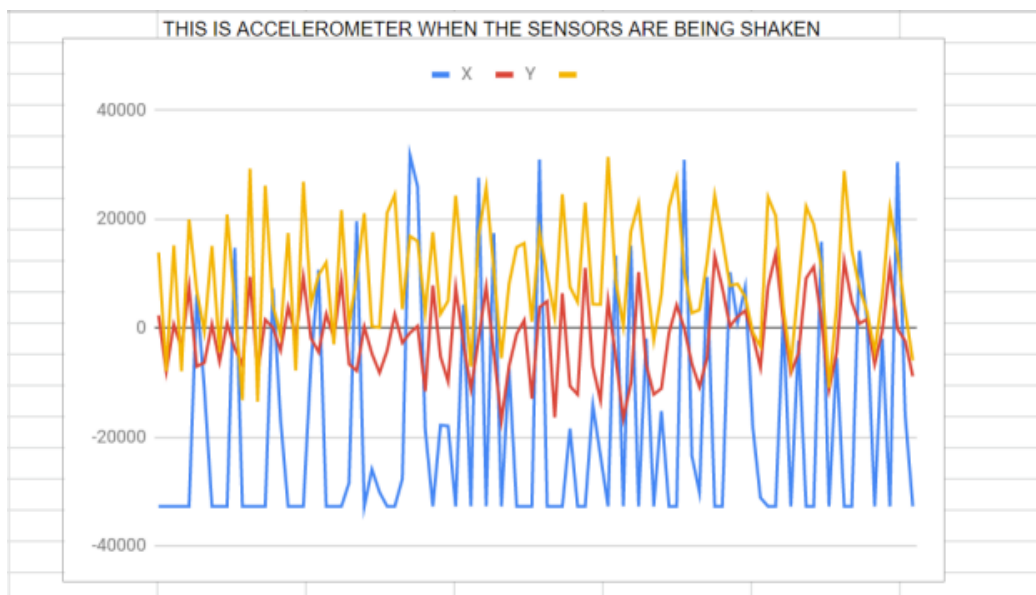


Figure 4.5 showing Accelerometer readings after sensor was shaken

From the figures above one can conclude that the acceleration sensor can automatically detect movement stimulus and convert it to digital representation.

Below is the graph obtained from the gyroscope.

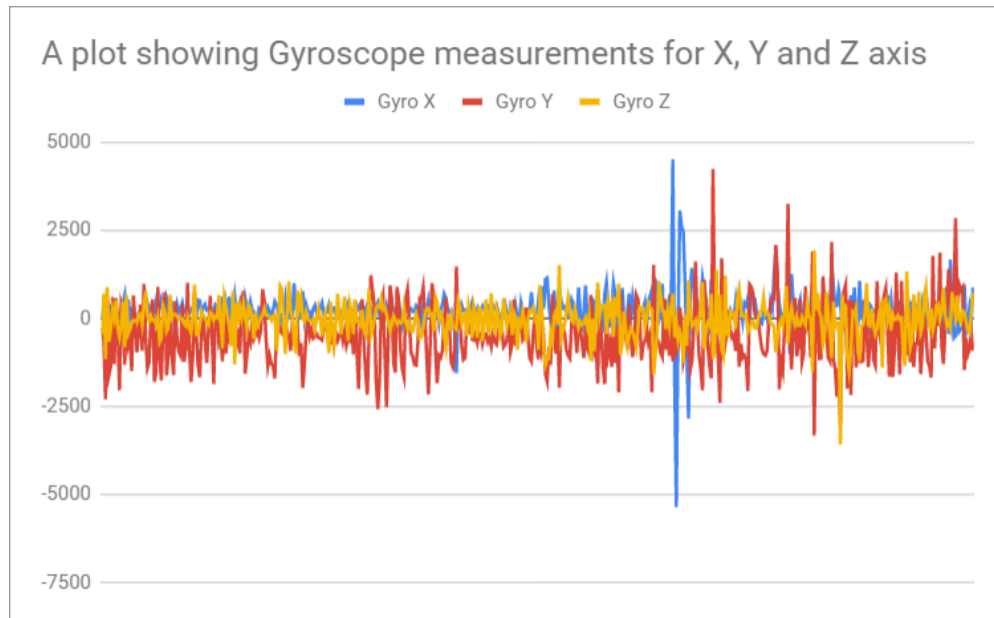


Figure 4.6 showing a plot for Gyroscope readings for the X,Y and Z axis

From the graphs shown above, it was concluded that the PI was receiving data from the IMU.

To check if the received data would be fit for the experiment, a sample of 1000 readings was taken from the supplied data of the Accelerometer. The data was plotted and compared with the received data to see if there were any similarities in the data. The plotted data is shown in the figures below

A graph showing supplied Accelerometer measurements for X, Y and Z axis

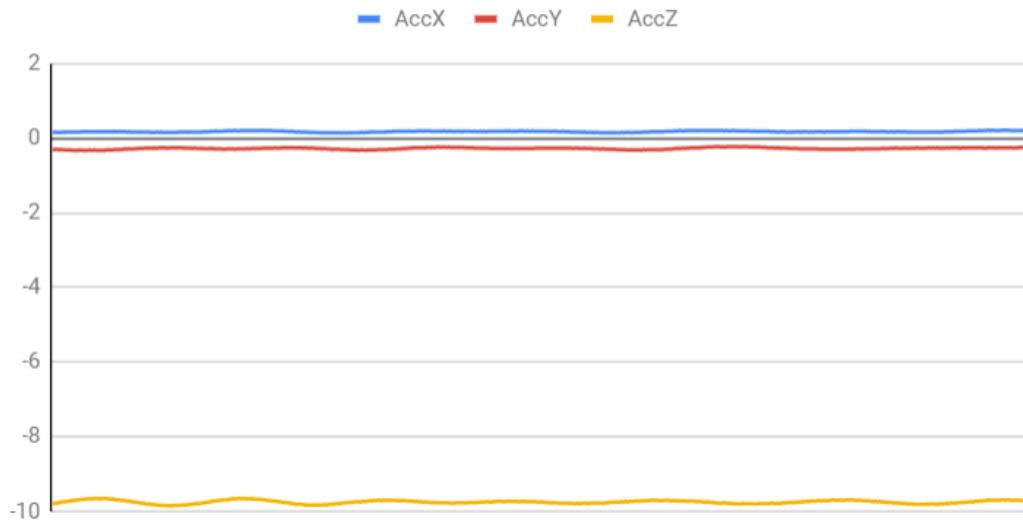


Figure 4.7 showing Accelerometer measurements for X, Y and Z axis of the supplied data

The graph above was compared to the graph of the Accelerometer from the received data. It was seen that the graphs were similar, therefore the data was considered as acceptable for the experiment.

4. The IMU's Temperature Detection

From the figures above one can conclude that the acceleration sensor can automatically detect movement stimulus and convert it to digital representation. The fourth test done was on the digital-output temperature sensor. To do this effectively, the IMU was placed in normal room conditions and thereafter in the refrigerator. It was expected that the temperature readings provided by the IMU would be significantly different for the two different scenarios. As expected, the temperature readings in the normal room conditions was around 25 degrees 6 Celsius while the temperature readings in the refrigerator was around 16 degrees Celsius. We believe that if the sensor was left in the refrigerator for a longer period, it would give an accurate reading of around 4 degrees. However, the temperature difference was enough to confirm that the sensor was working.

4.4. Experimental Setup

The system is designed to process data received from the sensor in the following sequence;

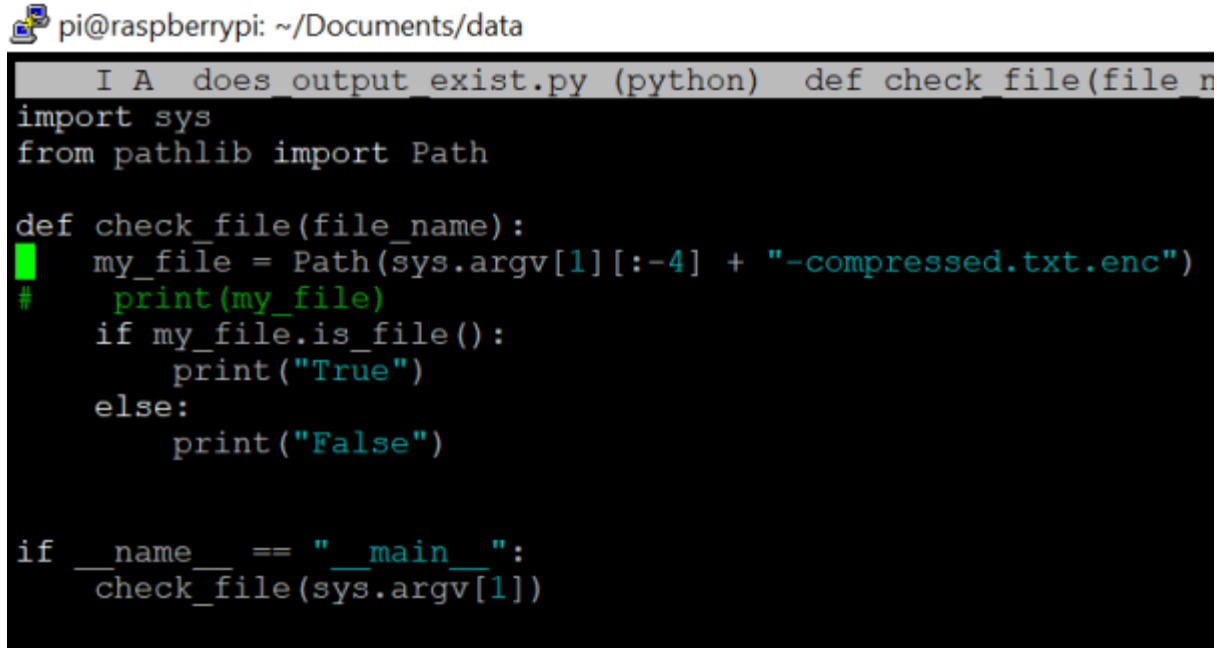
1. Compression - data is received from IMU and compressed bit by bit

2. Encryption - compressed data is read and encrypted. The encrypted data is transmitted for processing via TCP socket
3. Decryption - data is streamed from TCP socket and decrypted. The decrypted data is stored in a .txt file
4. Decompression - decrypted file is decompressed into a .csv file

The aim of the experiment is to assess the performance of each subsystem as the amount of data being processed is increased. This test will be carried out using two main criterias, that is, compression ratios to test for efficiency of the compression algorithm and data throughput to test for efficiency of the encryption algorithm.

I. Overall System

The latency of the system was put into test. In order to implement this, the native python time module was put into use. A timer was set before the system began and stopped right after. It should be noted that the timer was set only for the critical sections (compression and encryption) and not the trivial functions such as printing and reading files. Results were recorded and a graph plotted for a clearer representation. A test was carried out to test the output of the system. A .csv file was fed into the system and the output was examined for correctness. It was expected that an “.compressed.txt.enc” file would be present in the output after running the system. A python script was implemented to do this. The input being the name of the original file

A terminal window on a Raspberry Pi. The prompt is 'pi@raspberrypi: ~/Documents/data'. The code being entered is a Python script named 'does_output_exist.py'. The code imports 'sys' and 'Path' from 'pathlib'. It defines a function 'check_file(file_name)' that takes a file name as an argument, constructs a path by appending '-compressed.txt.enc' to the input, prints the path, and then checks if it is a file. If it is, it prints 'True'; otherwise, it prints 'False'. The script has a main block that calls 'check_file(sys.argv[1])' if run directly.

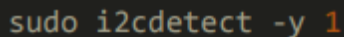
```
pi@raspberrypi: ~/Documents/data
I A does_output_exist.py (python) def check_file(file n
import sys
from pathlib import Path

def check_file(file_name):
    my_file = Path(sys.argv[1][: -4] + "-compressed.txt.enc")
    # print(my_file)
    if my_file.is_file():
        print("True")
    else:
        print("False")

if __name__ == "__main__":
    check_file(sys.argv[1])
```

Figure 4.8 showing python code to record time for program to execute

A test was done to ensure that the IMU and the Pi were connected well. This was done by using the I2C commands available in the Pi. To do this, the following command was run on the RPi

A terminal window showing the command 'sudo i2cdetect -y 1' being entered.

```
sudo i2cdetect -y 1
```

Figure 4.9 showing a test to detect if IMU is connected to PI

The test carried out was to ensure that there is proper communication between the compression and encryption. This was done by making sure the encryption system had input from the compression system at all times. Socket programming was used for the transmission.

A test was carried out to evaluate the power consumption of the overall system, to determine the measures required to keep the power to minimum values. Although some complications were faced as we tried to measure the power using software, we came to knowledge that best practices for measuring power consumption of a raspberry pi is by using Hardwares like USB voltmeter, which we would have had to buy. So, in an attempt to have some estimate of which technique has minimal power consumption, we decided to compare the time of execution of the overall system when the data is stored and transmitted as first case, and when the data is streamed bit by bit. The expected output is that streaming data bit by bit consumes less power from the overall system and the time of execution is also less. This is based on the results obtained during simulation of the algorithms and sub systems where we observed a greater time taken for processing as the size of the files processed increased.

II. Compression subsystem

The main criteria used to test the compression subsystem was compression ratio. Five data sets were used to carry out the experiment, that is, 50, 500, 1000, 2000 and 5000 readings. RLE.py was invoked. The user was asked to enter the name of the file that is to be compressed. A file called "Compression.csv " was created. The data and the size of the compressed file was recorded. The compressed file was decompressed and saved to a file called "Decompressed.csv". The size of the file was recorded.

An analysis was then performed to calculate the compression ratio of the data by dividing the size of the compressed file with the size of the initial file. The size of the decompressed file was also compared with the size of the original file to see if any losses had occurred.

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

A second test was carried out after the data had been received. The aim of the experiment is to ascertain whether significant losses had occurred along the way. To do this, The first test was done to determine whether the two files were of the same size. The second test that was done was to determine whether the files had the same length and finally content. It iterates through the contents of the files and compares the two. If there is a discrepancy in any of the three tests, the script returns an error and halts operation of the whole experiment. With these three parameters, we believed that the lossless nature of the algorithm would be adequately put to test.

A test was finally carried to test the speed of the subsystem. This was done using the native python time module. A timer was set before the function call and stopped after the function. This gave us the execution time of the function. The time was tested on various sets of data of different sizes and formats in order to draw accurate conclusions. Appropriate graphs were drawn to provide a visual. It should be noted that the timer was only set for the critical parts of the code i.e the compression and not other trivial functions like printing and opening the files

III. Encryption subsystem

I. Firstly, to test the encryption algorithm the data throughput was considered. Similar to the compression algorithm, 5 data sets were used to test the subsystem. The data from the Gyroscope sensor was encrypted. Data was send to the algorithm at a maximum throughput of **up to 44.5Gbits/s** to validate its operation. The Algorithm's throughput was calculated using Encryption time measured using the time library in a python file which measured only specifically the time taken by the algorithm to perform to completion.

ii. Checking whether the encryption 11 mechanism is reversible or not which is a very important part of this project. That is because failure of reversing the encrypted file means the origins data is lost or not accessible

iii. making sure that no one else can access the original data without the decryption script with the correct key, as that will defeat the purpose of encryption. This is successfully done by changing using a random key on the decryption script. Also making sure that the key is not known. This is completely dependent on the type of key used in this case a 258 bit key is used which is generally difficult to figure out.

iv. Looking at the entropy of the file. If the entropy is high, then it's likely encrypted. You can use tools like binwalk to determine the entropy. A consistent, high entropy indicates that the file is likely encrypted. To investigate files for hidden threats, you can look at file entropy values. File entropy measures the randomness of the data in a file and is used to determine whether a file contains hidden data or suspicious scripts. The scale of randomness is from 0, not random, to 8, totally random, such as an encrypted file. The more a unit can be compressed, the lower the entropy value; the less a unit can be compressed, the higher the entropy value.

IV. Transmission subsystem

A python code was written to create a TCP connection between the PI and a host on which the data will be decompressed and decrypted. A socket will be made on the PI and on the host computer, through which data will be transmitted. The encrypted data will be streamed.

A test was carried out to measure the maximum throughput of transmission. To this goal, data was sent with a throughput of 300Mbits/s. This was the maximum throughput attainable because it corresponds to the maximum throughput the raspberry pi is designed to transmit data.

4.5. Results

Entire System

1. Speed of the system

The tests explained earlier were implemented and the following results were obtained

	Data size (Bytes)	Speed(s)
1	5709	0.02376864
2	56890	0.15765757
3	623134	0.67576876
4	1247263	1.27062843

5	6240886	6.5621285
---	---------	-----------

Table 4.2 showing speed of the system

A graph of data size was plotted against the speed of computation, in an attempt to better visualize how these two quantities vary as function of each other.

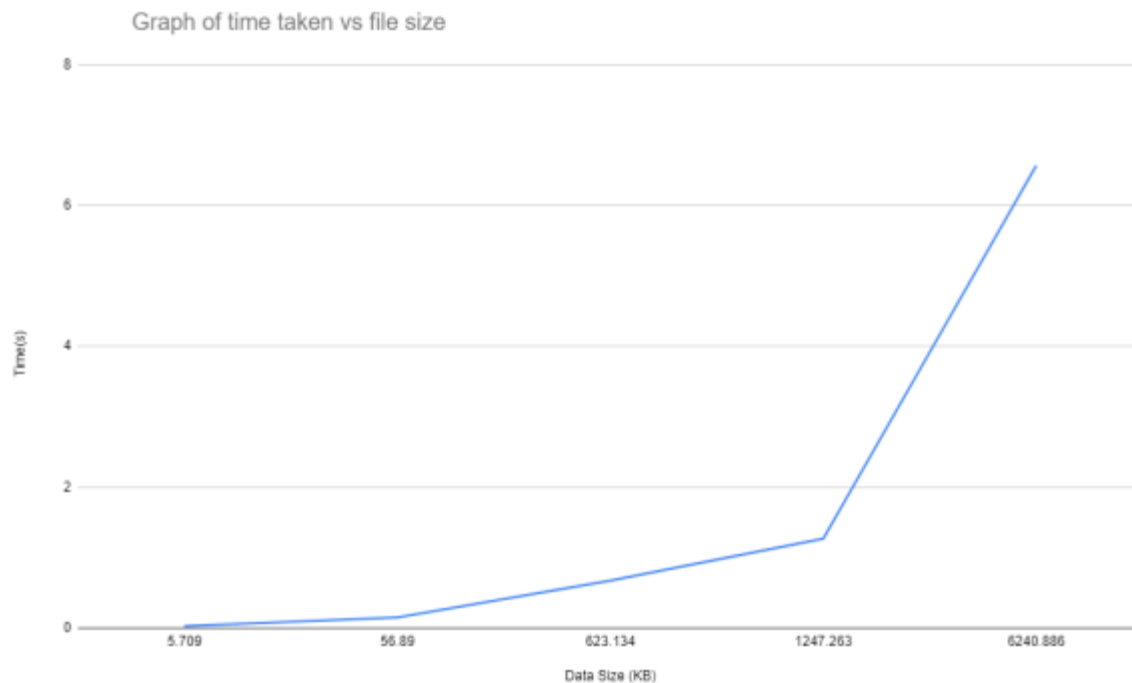


figure 4.10 showing file size vs time taken

Therefore, from the graph above, the time taken increased exponentially as the size of the file increased.

2. Data transmission

The system ran and produced an output indicating that it indeed had an input and therefore there was proper communication between the two systems

3. IMU connectivity to the Pi

After running the commands detailed in the experiment setup, the following output was produced.

```

pi@raspberrypi:~ $ sudo i2cdetect -y 1
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  29  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  48  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  5c  --  --  --  --
60:  --  --  --  --  --  --  68  --  --  --  --  --  --  --  --  --
70: 70  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --

```

Figure 4.11 showing that IMU an PI are connected

This is a confirmation that the raspberry pi and the IMU were connected.

4.5.1. Results for compression and decompression

Compression Ratio = Initial file size/Compressed file size

The results of the test on the compression subsystem are shown in the table below

Data Set	Initial File Size	Compressed file Size	Decompressed File Size	Compression Ratio
50	1 052 bytes	1 012 bytes	1 051 bytes	1.0009514
500	10 743 bytes	9 878 bytes	10 743 bytes	1.0875683
1000	21 910 bytes	21 864 bytes	21 910 bytes	1.0020134
2000	42 982 bytes	42 782 bytes	42 9812 bytes	1.0046749
5000	91 346 bytes	90 393 bytes	91 346 bytes	1.0105428

Table 4.3 showing results of compression subsystems.

From the table above, it can be shown that increasing the amount of data has little effect on the compression subsystem as seen by the small differences in the compression ratios as the size of the files increased.

It was also observed that some of the files had some losses. The decompressed data was plotted and compared to the initial file to see the effect of the losses. The plots are shown below:

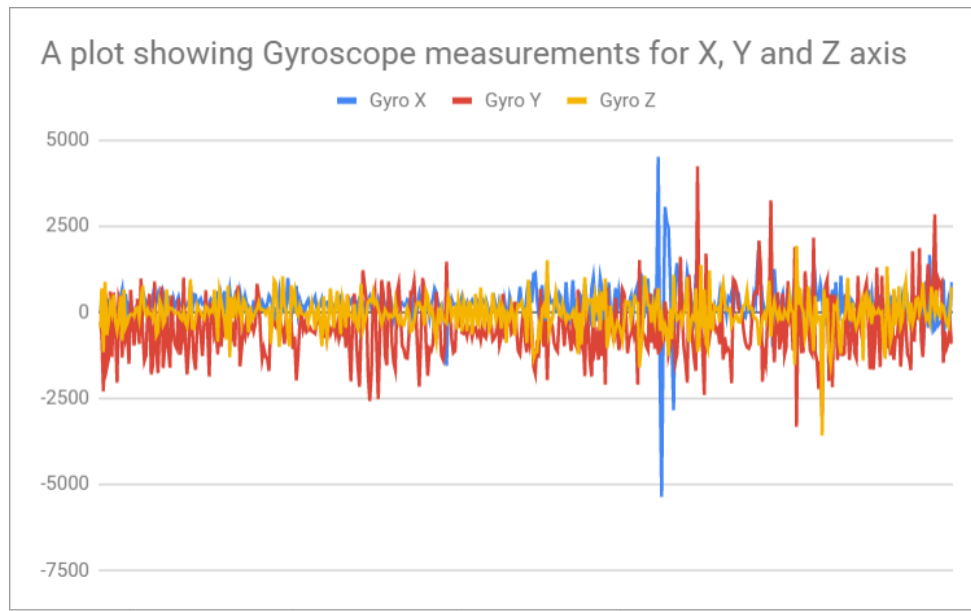


Figure 4.12 showing initial data for Gyroscope

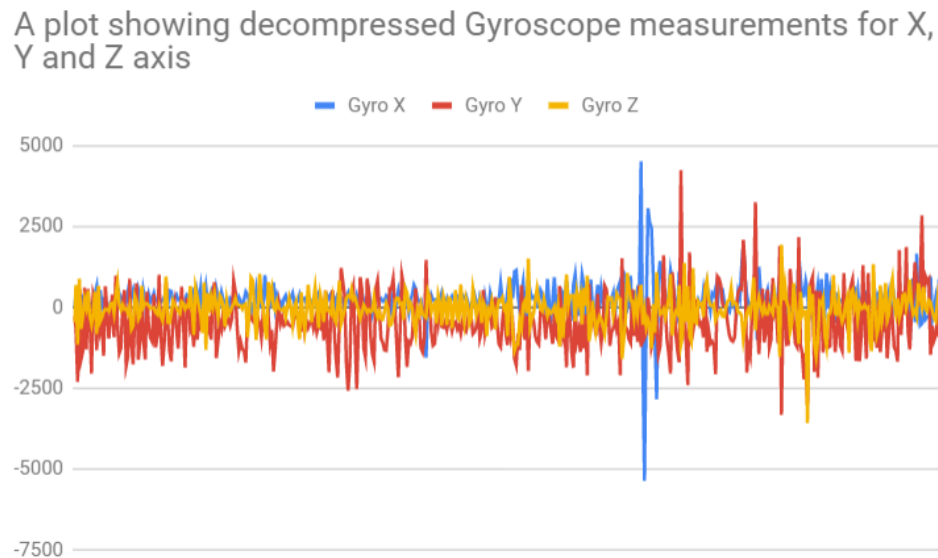


Figure 4.13 showing decompressed Gyroscope measurements

Comparing the two graphs shown above, it can be seen that the effect of the losses is negligible.

Bottleneck

Run Length Encoding algorithm works well with repetitive data. The data collected by the sensor did not have a lot of repetitive data, therefore the compression was not as efficient.

Limitation of experiment

Run Length Encoding algorithm is a lossless algorithm. However, it was observed after decompression that some of the files had some losses. The cause of these losses could not be determined and are beyond the scope of the experiment.

4.5.2. Results for encryption and decryption

1. Testing the data performance

The results of the experiment are shown in the table below

Data Set	File Size	Encryption Time	Decryption Time
50	1 012 bytes	1.02753286463823	0.005865376100944
500	9 878 bytes	1.23320823468323	0.0112780901068963
1000	20 864 bytes	1.88784933671840	0.0193009342276422
2000	42 982 bytes	3.74537283166363	0.0373652019920321
5000	91 346 bytes	6.5621285736773	0.183654629303672

Table 4.4 showing results of encryption subsystem

To measure the performance of the system, a graph of time vs data set was plotted for both encryption and decryption time. The graphs are shown in the figures below:

A graph showing Encryption time vs Data set

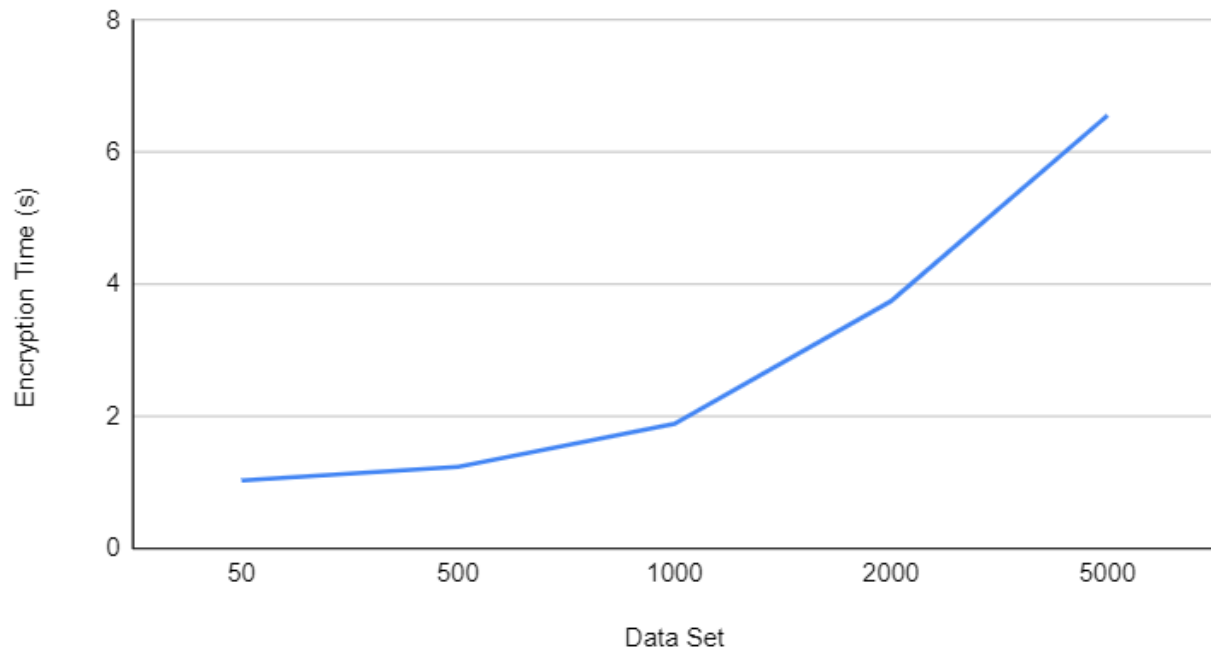


Figure 4.14 showing Encryption Time vs Data set

From the graph above, it can be shown that encryption time will increase exponentially as the amount of data being encrypted increases.

A graph showing Decryption Time vs. Data Set

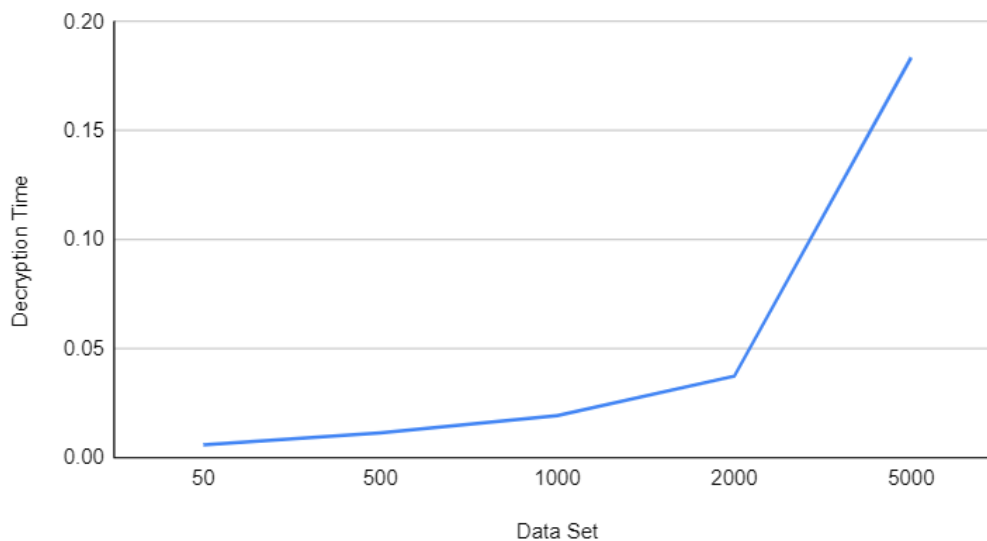


Figure 4.15 showing Decryption Time vs Data set

From the graph shown above, it can be seen that as the amount of readings taken increases, the time taken to decrypt the data will increase exponentially.

1. Recovery test:

The results are analyzed based on the implementation discussed above. figure 2.1 below shows the original data before it is encrypted

```

1 computer utc start 2018-09-19-04:22:31.529971
2 UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM
3 2018-09-19-04:22:31.793573 -0.34088175134658813 0.35862863063812256 0.38060529708862385 0.23097454011440277 -0.3260
4 2018-09-19-04:22:31.893383 -0.342188460691452 0.36213183403015137 0.3067106604576111 0.23146139085292816 -0.3308394
5 2018-09-19-04:22:31.993358 -0.34227073192596436 0.35868507623672485 0.3842473793029785 0.23394353687763214 -0.32825
6 2018-09-19-04:22:32.093380 -0.33891722559928894 0.3597724735736847 0.3056625723838806 0.21971283853054047 -0.331785
7 2018-09-19-04:22:32.193370 -0.34226855635643005 0.3586834967136383 0.30424585938453674 0.21944978833198547 -0.32590
8 2018-09-19-04:22:32.293361 -0.34003153443336487 0.36091920733451843 0.3005756437778473 0.21391789615154266 -0.32591
9 2018-09-19-04:22:32.393351 -0.3410453200340271 0.3643854856491089 0.3029782474040985 0.21461999416351318 -0.3265402
10 2018-09-19-04:22:32.493357 -0.3410690426826477 0.363239586353302 0.3029923439025879 0.21591301262378693 -0.31855210
11 2018-09-19-04:22:32.593398 -0.34011411666870117 0.3563460409641266 0.30314040184020996 0.19787898659706116 -0.32249
12 2018-09-19-04:22:32.693342 -0.34121447801589966 0.3552418640196625 0.3081103265285492 0.2025754451751709 -0.3199371
13 2018-09-19-04:22:32.793340 -0.3389686048030853 0.35748231410980225 0.30569320917129517 0.19914020597934723 -0.31245
14 2018-09-19-04:22:32.893385 -0.34227919578552246 0.3575487434864044 0.30676835775375366 0.18888604640960693 -0.31485
15 2018-09-19-04:22:32.993407 -0.3422193229198456 0.36097484827041626 0.3042171597480774 0.18875494599342346 -0.318521
16 2018-09-19-04:22:33.093386 -0.3422207236289978 0.36097633838653564 0.30421796441078186 0.18590101599693298 -0.31363
17 2018-09-19-04:22:33.193342 -0.3411988317966461 0.35638344287872314 0.3068430423736572 0.18076957762241364 -0.310221
18 2018-09-19-04:22:33.293366 -0.3390343189239502 0.3551810681819916 0.3032151758670807 0.18188047409057617 -0.3065259
19 2018-09-19-04:22:33.393342 -0.34009069204330444 0.35749325156211853 0.30312711000442505 0.17157606780529022 -0.3034
20 2018-09-19-04:22:33.493335 -0.3388703763484955 0.36319708824157715 0.3018614947795868 0.1723051518201828 -0.3019212
21 2018-09-19-04:22:33.593378 -0.3401644825935364 0.35518717765808105 0.29939576983451843 0.1633971929550171 -0.297162
22 2018-09-19-04:22:33.693340 -0.33793994784355164 0.35402682423591614 0.3057979345321655 0.16524161398410797 -0.28876
23 2018-09-19-04:22:33.793317 -0.3401374816894531 0.3540785312652588 0.30818718671798706 0.15837541222572327 -0.294530
24 2018-09-19-04:22:33.893314 -0.33904027938842773 0.35405370593070984 0.30699390172958374 0.1590089148044586 -0.2921
25 2018-09-19-04:22:33.993331 -0.33902448415756226 0.3551957309246063 0.30572694540023804 0.1572961062192917 -0.283139
26 2018-09-19-04:22:34.093327 -0.3379184305667877 0.35517433285713196 0.3057858347892761 0.1518937349319458 -0.2851741
27 2018-09-19-04:22:34.193365 -0.3390256464481354 0.35519564151763916 0.3057270646095276 0.15245196223258972 -0.283565
28 2018-09-19-04:22:34.293412 -0.33785420656204224 0.358607679605484 0.3044903874397278 0.14724959433078766 -0.2662251
29 2018-09-19-04:22:34.393352 -0.33904117345809937 0.3563167154788971 0.2994458079338074 0.15416912734508514 -0.271390
30 2018-09-19-04:22:34.493330 -0.3378465175628662 0.3586127758026123 0.3057402211306662 0.14544525742330823 -0.2623303

```

The python file used to encrypt these data is shown below,

```

import sys
from encryptor import Encryptor

#key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02]j\xdf\
key = b'[EX\xc8\xd5\xbfI{\xa2$\x05(\xd5\x18\xbf\xc0\x85)\x10nc\x94\x02]j\xdf\xcb\xc4\x94\x9d(\x9e'
encryptor1 = Encryptor(key)

encryptor1.encrypt_file(sys.argv[1])

```

And below is the resulting correctly encrypted file



Furthermore, the file was tested for decryption, to compare how different the output file is to the original data set file. Firstly, below is shown the python code ran for implementing decryption



And below is the decrypted file

```

1 computer utc start 2018-09-19-04:22:31.529971
2 UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM
3 2018-09-19-04:22:31.793573 -0.34008175134658813 0.35862863063812256 0.30060529708862305 0.23097454011440277 -0.3260
4 2018-09-19-04:22:31.893383 -0.342180460691452 0.36213183403015137 0.3067106604576111 0.23146139085292816 -0.3308394
5 2018-09-19-04:22:31.993358 -0.34227073192596436 0.35868507623672485 0.3042473793029785 0.23394353687763214 -0.32825
6 2018-09-19-04:22:32.093380 -0.33891722559928894 0.3597724735736847 0.3056625723838806 0.21971283853054047 -0.331705
7 2018-09-19-04:22:32.193370 -0.34226855635643005 0.3586834967136383 0.30424585938453674 0.21944978833198547 -0.32590
8 2018-09-19-04:22:32.293361 -0.34003153443336487 0.36091920733451843 0.3005756437778473 0.21391789615154266 -0.32591
9 2018-09-19-04:22:32.393351 -0.3410453200340271 0.3643854856491089 0.3029782474040985 0.21461999416351318 -0.3265402
10 2018-09-19-04:22:32.493357 -0.3410690426826477 0.363239586353302 0.3029923439025879 0.21591301262378693 -0.31855210
11 2018-09-19-04:22:32.593398 -0.34011411666870117 0.3563460409641266 0.30314040184020996 0.19787898659706116 -0.32249
12 2018-09-19-04:22:32.693342 -0.34121447801589966 0.3552418649196625 0.3081103265285492 0.2025754451751709 -0.3199371
13 2018-09-19-04:22:32.793340 -0.3389686048030853 0.35748231410980225 0.30569320917129517 0.19914020597934723 -0.31245
14 2018-09-19-04:22:32.893385 -0.34227919578552246 0.3575487434864044 0.30676835775375366 0.1888604640960693 -0.31485
15 2018-09-19-04:22:32.993407 -0.3422193229198456 0.36097484827041626 0.3042171597480774 0.18875494599342346 -0.318521
16 2018-09-19-04:22:33.093386 -0.3422207236289978 0.36097633838653564 0.30421796441078186 0.18590101599693298 -0.31363
17 2018-09-19-04:22:33.193342 -0.3411988317966461 0.35638344287872314 0.3068430423736572 0.18076957762241364 -0.310221
18 2018-09-19-04:22:33.293366 -0.3390343189239502 0.3551818681819916 0.3032151758670807 0.18188047409057617 -0.3065259
19 2018-09-19-04:22:33.393342 -0.34009069204330444 0.35749325156211853 0.30312711000442505 0.17157606780529022 -0.3034
20 2018-09-19-04:22:33.493335 -0.3388703763484955 0.36319708824157715 0.3018614947795868 0.1723051518201828 -0.3019212
21 2018-09-19-04:22:33.593378 -0.3401644825935364 0.35518717765808105 0.29939576983451843 0.1633971929550171 -0.297162
22 2018-09-19-04:22:33.693340 -0.33793994784355164 0.35402682423591614 0.3057979345321655 0.16524161398410797 -0.28876
23 2018-09-19-04:22:33.793317 -0.3401374816894531 0.3540785312652588 0.30818718671798706 0.15837541222572327 -0.294530
24 2018-09-19-04:22:33.893314 -0.33904027938842773 0.35405370593070984 0.30699390172958374 0.15909089148044586 -0.2921
25 2018-09-19-04:22:33.993331 -0.33902448415756226 0.3551957309246063 0.30572694540023804 0.1572961062192917 -0.283139
26 2018-09-19-04:22:34.093327 -0.3379184305667877 0.35517433285713196 0.3057858347892761 0.1518937349319458 -0.2851741
27 2018-09-19-04:22:34.193365 -0.3390256464481354 0.35519564151763916 0.3057270646095276 0.15245196223258972 -0.283565
28 2018-09-19-04:22:34.293412 -0.33785420656204224 0.358607679605484 0.3044903874397278 0.14724959433078766 -0.2662251
29 2018-09-19-04:22:34.393352 -0.33904117345809937 0.3563167154788971 0.2994458079338074 0.15416912734508514 -0.271390
30 2018-09-19-04:22:34.493310 -0.3378465175628662 0.3586127758876123 0.3057440271109662 0.14544525742558823 -0.2621203

```

1. Testing the Key correctness

It is shown that using a different key to decrypt the file has shown to be unsuccessful, and therefore one can not recover the original data. The above encrypted file was tried to be decrypted with a !6 bit key, but the output did not correspond to the original data set.

1. Entropy Test

Below are results from Entropy and Randomness Online Tester of the original file 2018-09-19-03_57_11_VN100.CSV and the encrypted file 2018-09-19-03_57_11_VN100.CSV.ENC

ST

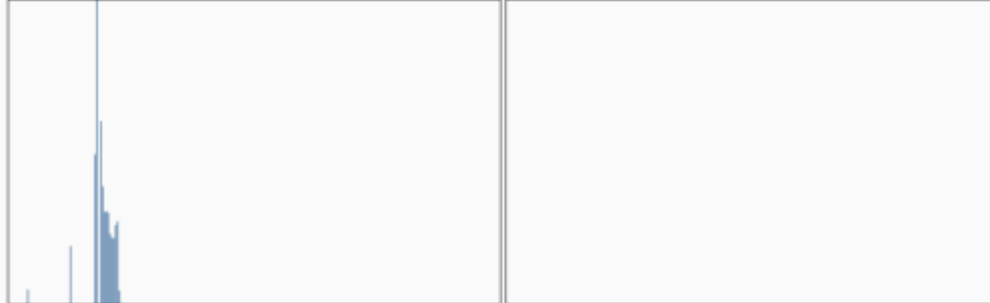
SERVER TEST

Entropy and Randomness Online Tester

Execution date: Tue, 05 Oct 2021 19:05:00 GMT, file: 2018-09-19-03_57_11_VN100.csv, size: 9,071,107 bytes, test type: Entropy only.

Entropy only results

6.964835 min:0 med:69 max:166177



[Home](#) | [Entropy](#) | [FTP](#) | [HTTP](#) | [HTTP/2](#) | [IP](#) | [NTP](#) | [About](#) | [Website Builder](#) | [VPS Server](#)

ST

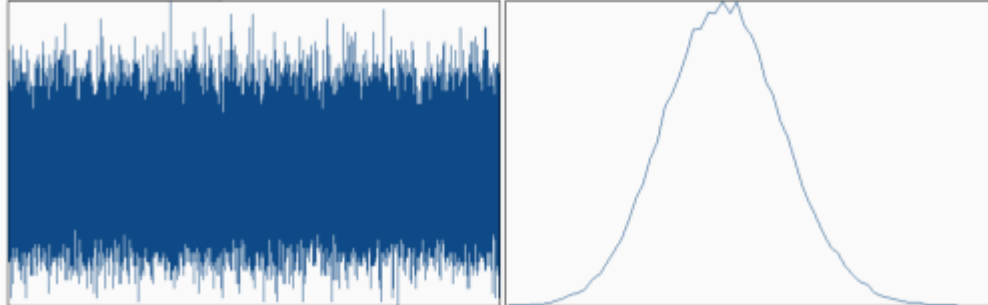
SERVER TEST

Entropy and Randomness Online Tester

Execution date: Tue, 05 Oct 2021 19:00:20 GMT, file: 2018-09-19-03_57_11_VN100.csv.enc, size: 9,071,136 bytes, test type: Entropy only.

Entropy only results

15.989605 min:39 med:69 max:107



[Home](#) | [Entropy](#) | [FTP](#) | [HTTP](#) | [HTTP/2](#) | [IP](#) | [NTP](#) | [About](#) | [Website Builder](#) | [VPS Server](#)

4.5.3. Results for transmission

Data used to test compression and encryption algorithms was sent through the TCP connection. From the results above, little to no losses occurred during transmission. The system sustain transfer of data at 300Mbits/s , which is the pi maximum throughput performance, using socket programming with a host(In this case, the host represents the device on which decryption and decompression wiill happen in case of the real Byoc sharc).

5. CONSOLIDATION OF ATPs AND FUTURE PLAN

ATP

ATP	Configuration of the slave
Evaluation type	Hardware
Target	I2C connection
Test protocol	<ul style="list-style-type: none">• Connect the IMU to the master, read and write from IMU registers• Additional self test
Pass condition	Start and stop condition generation
Fail condition	No response from the IMU
Test result	Pass

ATP	Desired output produced
Evaluation type	Software
Target	Entire system
Test protocol	<ul style="list-style-type: none">• Feed a file into the system.• Examine the output produced•
Pass condition	<ul style="list-style-type: none">• A compressed and encrypted file is produced.

Fail condition	A file that is not both compressed and encrypted is produced
Test result	Pass

Compression:

ATP	Compressed file size test
Evaluation type	Software
Target	Compression subsystem
Test protocol	<ul style="list-style-type: none"> ● Feed the sample data into the subsystem ● Determine and compare the <ul style="list-style-type: none"> - size of the output with the original
Pass condition	The output file's size is smaller than the input
Fail condition	The output file's size is equal or bigger than the original file.
Test result	Pass

ATP	Data loss test
Evaluation type	Software

Target	Compression subsystem
Test protocol	<ul style="list-style-type: none"> ● Feed the sample data into the subsystem ● Extract the output of the subsystem and decompress it. ● Compare this with the original file ●
Pass condition	The decompressed file is the same as the original file
Fail condition	The decompressed file differs from the original file
Test result	Pass

Encryption and Decryption

ATP	Encryption/decryption algorithm correctness and successful execution
Evaluation type	Software
Target	Encryption algorithm
Test protocol	<ul style="list-style-type: none"> ● Simulation of the encryption and decryption process using the open source code here
Pass condition	<ul style="list-style-type: none"> ● Successfully encrypt the given data

	<ul style="list-style-type: none"> • Successfully decryption of the given encrypted file with a key back to the original data
Fail condition	<ul style="list-style-type: none"> • Non recovery of the original data
Test result	Pass

Overall System

ATP	Maximum Throughput
Evaluation type	Performance
Target	Overall system performance
Test protocol	<ul style="list-style-type: none"> • Data sent at very high throughput up to a maximum of 44Gbits/s
Pass condition	Data successfully compressed, encrypted, transmitted, received at maximum throughput
Fail condition	Data is processed incorrectly and output is inconsistent
Test result	Pass

ATP	Minimal Power usage
Evaluation type	Hardware

Target	IMU battery life
Test protocol	Performed experiment to test the amount of power consumed by the IMU
Pass condition	Power usage is kept at a minimal level and only directed towards vital task like data collection
Fail condition	Unnecessary power consumption at the level of the IMU
Test result	Pass

Added ATP

ATP	Entropy and Randomness Test
Evaluation type	Software
Target	Encrypted files
Test protocol	<ul style="list-style-type: none"> • Entropy formula measures the equiprobability regardless of real <u>unpredictability</u>. • Tool can be found <u>here</u>
Pass condition	<ul style="list-style-type: none"> • High Entropy score • Dense Entropy graph is more desirable for a pass condition
Fail condition	<ul style="list-style-type: none"> • Low entropy score • Non dense Entropy graph

Test result	Pass
-------------	------

ATP	Encryption/decryption algorithm secured key
Evaluation type	Software
Target	Encryption and decryption algorithm
Test protocol	<ul style="list-style-type: none"> • Simulation of the encryption and decryption process using a different key for these stages
Pass condition	<ul style="list-style-type: none"> • Non recovery of the original data • See under experiment results 2
Fail condition	<ul style="list-style-type: none"> • Successfully encrypt the given data • Successfully decryption of the given encrypted file with a key back to the original data
Test result	Pass

Necessary Future work before Usage

Before usage of our module, some understanding about Socket programming is necessary. This is so that we are able to configure the IMU to send the collected data to the required address and that has to be done manually, and therefore requires some knowledge of IP addresses and a little bit of the system of operation of Socket programming.

6. CONCLUSION

In conclusion, the designed system met all requirements. The system can successfully retrieve data from the IMU sensor and save it to a file. The data in the file was then compressed using Run Length encoding and encrypted using AES encoding. The data was transmitted to host via TCP connection for processing. Upon receipt, the data was decrypted and decompressed. The decompressed data was then stored in a .csv file and was ready for processing. It was also shown that the data had small losses, however, the losses were very small and the decompressed data matched the initial data. This means that the system can retain more 25% of Fourier Transforms. The system's performance depends on throughput. If more data is being sent to the system, the system's performance would decrease exponentially. Because the compression algorithm had a simple implementation, the system used little processing power. A recommendation would be that the compression algorithm be changed to LZ4, as it compresses data at a faster rate as compared to RLE and is suitable for compressing large size file.

7. REFERENCES

1. BBC Bitesize. 2021. Codecs and compression algorithms - Encoding audio and video - GCSE Computer Science Revision - BBC Bitesize. [online] Available at: [Accessed 01 September 2021].
2. Ameer, I., 2020. The Effect of Re-Use of Lossy JPEG Compression Algorithm on the Quality of Satellite Image. *Neuroquantology*, 18(5), pp.17-25.
3. Rao, K., 2021. Discrete cosine transform : algorithms, advantages, applications in SearchWorks catalog. [online] Searchworks.stanford.edu. Available at: [Accessed 01 September 2021].
4. Dudhagara, C. and Patel, H., 2021. Performance Analysis of Data Compression Using Lossless Run Length Encoding. [online] *Oriental Journal of Computer Science and Technology*. Available at: <http://www.computerscijournal.org/vol10no3/performance-analysis-of-data-compression-using-lossless-run-length-encoding> [Accessed 02 October 2021].

8. APPENDIX

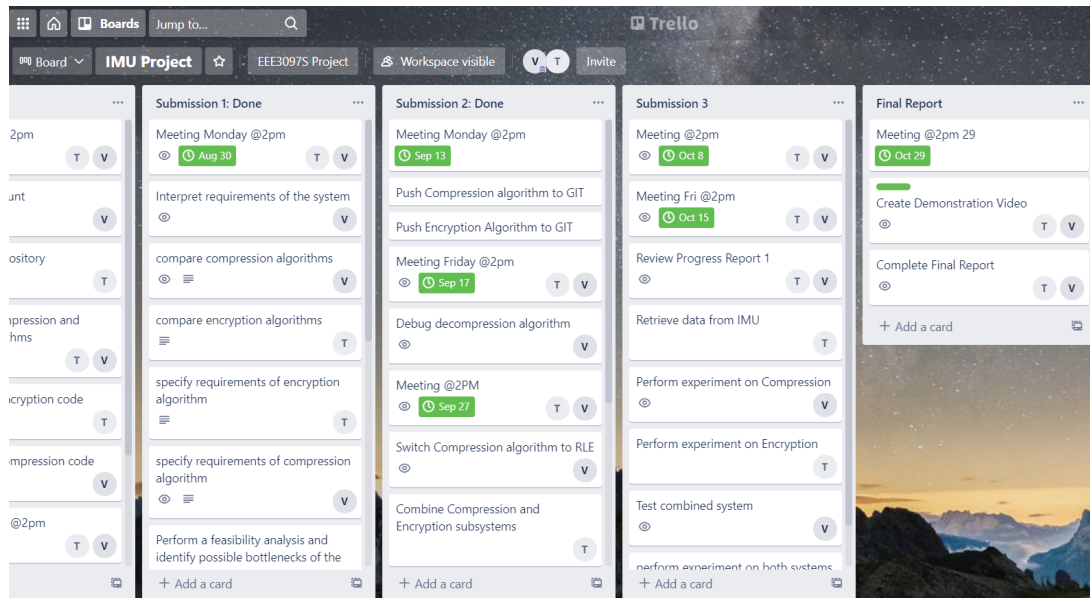
Table of contributions

Group Member	Contribution	Section Number	Page number
MZRVIN001	1.Compression	2.1	2
	● Design data test	2.2.1	4
	● Simulate subsystem	2.4.1	8
	● Analyze results of	3.3.1	9
	compression	3.3.2	13
	subsystem	3.4.3	14
	3. Identify system	3.5.1	19
	requirements and	3.5.3	21
	specifications	4.3	29
	4.Design test cases for	4.4	33
	overall system	4.5.1	39
TGNRAM001	5.compare IMUs	5	43
	6.Management of		
	development timeline,		
	and Trello page		
	1.Encryption	2.2	2
	● Design data test	2..4.3	5
	● Simulate subsystem	2.4.4	9
	● Analyze results of	3.3	12
	subsystem	3.3.3	15
	2.Transmission	3.4.1	17
	Subsystem	3.4.2	18
	3. Configuring PI	3.5.2	19
	Retrieving data from	4.2	34
	IMU	4.3	34
	4.Design test for IMU	4.5.2	43
	functionality	5	53
	5.Update ATPs		

Link to git repository

https://gitlab.com/vincent_muzerengwa/sharc-buoy-subsystem-project.git

Trello page



Development Timeline

Paper Design

WBS	Task	Start	End
0.1	Identify Requirements	Mon 17-08-21	Mon 23-08-21
0.2	Design System Specifications	Tue 25-08-21	Fri 28-08-21
0.3	Subsystem Design	Fri 28-08-21	Mon 20-08-21
0.4	Acceptance Test Procedures	Thur 02-09-21	Thur 21-09-21
0.5	Paper Design writing	Fri 03-10-21	Sat 04-10-21

Progress Report 1

WBS	Task	Start	End
1.1	Algorithm Design	Mon 06-09-21	Fri 17-09-21
1.2	Subsystem Combining	Mon 20-09-21	Fri 24-09-21
1.3	System Simulation	Mon 09-20-21	Fri 10-24-21
1.4	ATP testing and System Debugging	Mon 27-09-21	Fri 10-01-21
1.5	Progress Report 2 Writing	Fri 01-10-21	Mon 04-08-21

Progress Report 2

WBS	Task	Start	End
2.1	Retrieving data from IMU to PI	Mon 18-10-21	Mon 18-10-21
2.2	Testing retrieved data on each subsystem	Tue 19-10-21	Tue 19-10-21
2.3	Testing data on combined subsystem	Wed 20-10-21	Wed 20-10-21
2.4	ATP testing	Thur 21-10-21	Thur 21-10-21
2.5	Progress Report 2 Writing	Fri 22-10-21	Sat 23-10-21

Final Report

WBS	Task	Start	End
3.1	Record Demonstration vide	Mon 26-10-21	Mon 26-10-21
3.2	Final Report Writing	Tue 26-10-21	Fri 29-10-21

The project was completed according to schedule.