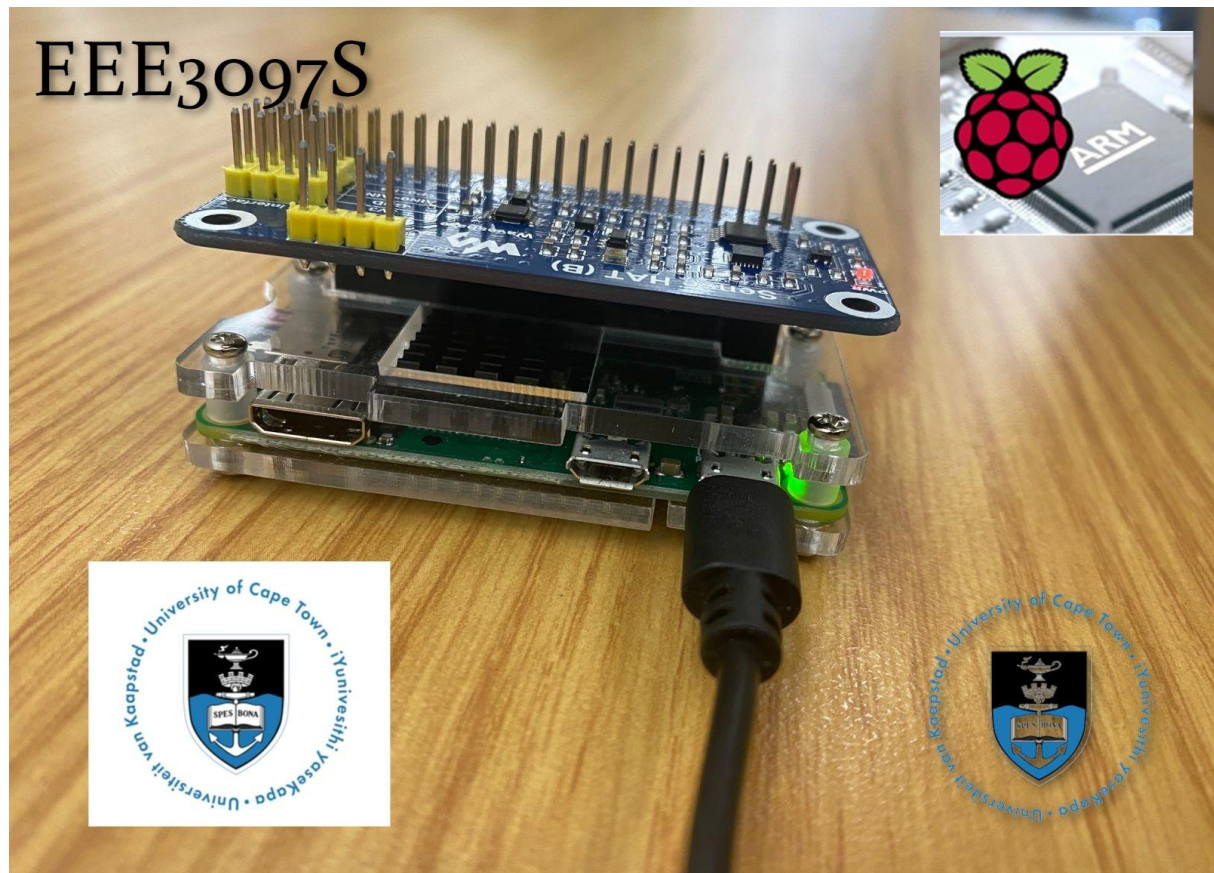


# Final Report EEE3097S



Prepared by:

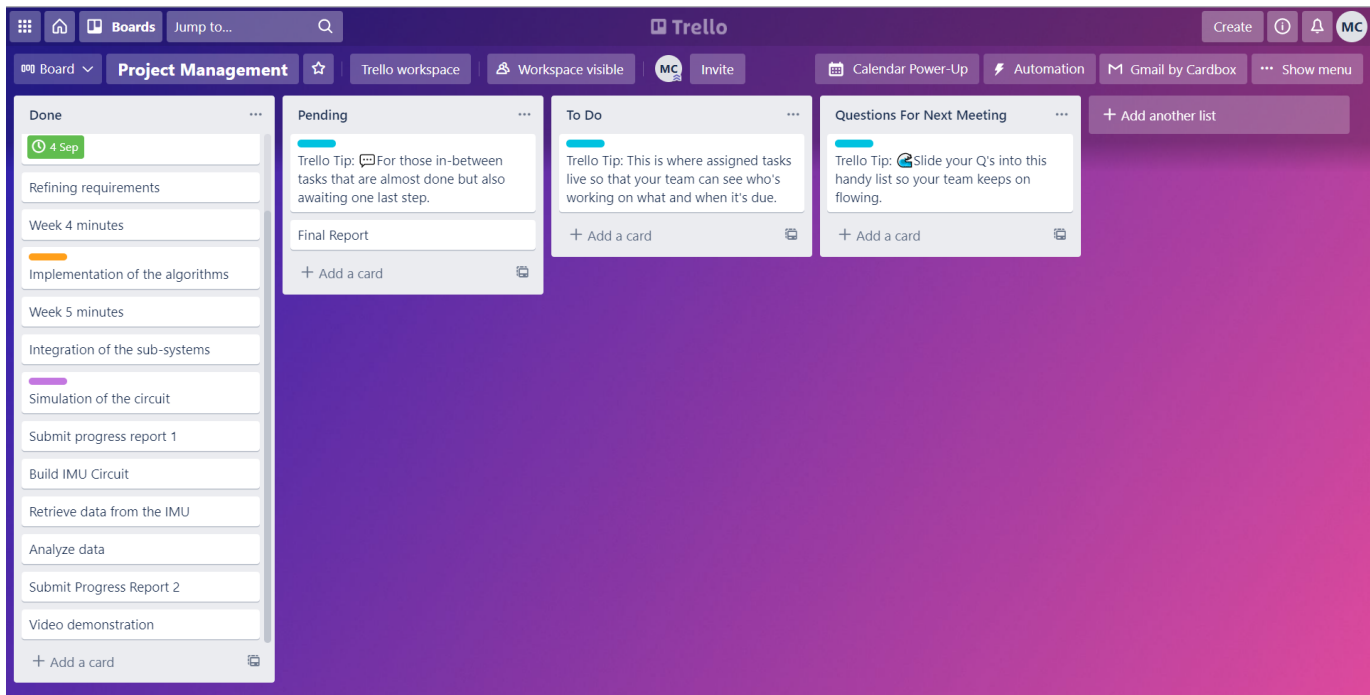
Marvin Kangangi (KNGMAR027)

Lindani Khuzwayo (KHZLIN012)

Date: October 2021

## **ADMIN**

<b>Name</b>	<b>Contribution</b>	<b>Section Number</b>	<b>Page Numbers</b>
Marvin Kangangi	Compression section	Admin, Paper Design, Requirements Interpretation ,Validation using IMU data, IMU Validation, Extrapolation, ATPs, System experiments and results	1,2,3,4,6,7,8, 9,10,11,12,13, 14,16.17,21,22 ,23,26,27,28,2 9,30,36,37,38, 39,40,41,42,43 ,47,48
Lindani Khuzwayo	Encryption section	IMU and data collection,Pap er Design, Validation using simulated data, Feasibility analysis, System experiments and results, ATPS	3,5,6,7,8,9,10 ,11,14,15,16,1 7,18,19,20,24, 25,26,30,31,32 ,33,34,35,36,3 8,39,40,41,44, 45,46



The github repo can be found [here](#).

## Timeline

- ☒ ~~Project research and planning~~
- ☒ ~~Paper design report~~
- ☒ ~~Implementing the code on the Raspberry Pi~~
- ☒ ~~Testing with sample data that resembles IMU data~~
- ☒ ~~Testing with actual data from the IMU~~
- ☒ ~~Analyzing the data~~
- ☒ ~~Second progress Report~~
- ☒ ~~Final Report~~
- ☒ ~~Demo video~~

We are on track to finish the project on the due date as stipulated.

## Introduction

This report focuses on the design of a system and development plan, implementing a prototype of this design and evaluating the design through formal testing processes. This will provide insight to understand the intricacies of real-life complex sub system design.

### **Problem statement**

The aim of this design is to come up with an ARM based digital IP using the Raspberry-Pi to encrypt and compress the IMU data.

### **Interpretation of requirements**

For this design project, the ICM-20649 IMU is to be used. This is a motion sensor that will provide data for compression and encryption. Using I2C communication, the data will be read from the IMU and manipulated to provide meaningful information.

Since oceanographers would like to extract at least 25% of the Fourier coefficients, lossless algorithms will be implemented. They will provide an even higher percentage of extraction thus allowing the oceanographers to extract meaningful data.

Due to power constraints of the IMU, a fast and efficient compression and encryption algorithm will be used. This will save power required to process the data and thus allow longer collection of data.

Optimization of the code will also be done.

### **Compression and encryption algorithms**

Compression is the process in which data is encoded and restructured in order to reduce its size. This is done in order to save both space and transmission time. There are various algorithms that can be implemented to carry this out. These algorithms can be broadly classified in two different ways. One being lossy compression and the other being lossless compression. Lossless compression algorithms allow for the original data to be reconstructed from the compressed data while lossy algorithms don't. The data received from the IMU is binary data. There are several algorithms that can be used to compress binary data. Some of the widely used compression methods include Run Length Encoding, Huffman Coding, Shannon Fano Compression and LZW Compression to mention a few. When deciding on the algorithm to implement, we took into consideration various factors such as performance, ease-of-use, proprietary concerns and ability to recover original data. Some comparisons of the algorithms can be found in Appendix A.

From the tables in the appendix, the fastest algorithm was Run Length followed by Shannon Fano and Huffman. Adapting Huffman was the slowest in execution.

In terms of compressed file size, Adaptive Huffman provided the smallest file followed by LZW and Huffman Encoding. Run Length Encoding produced the largest compressed file which is not ideal. In terms of code efficiency, Huffman provided the highest code efficiency. It also had the second highest saving percentage.

Putting all these factors into consideration, it was clear that the Huffman Encoding was the best option. After the comparisons, we decided to use GZIP compression for the data provided by the IMU. GZIP compression was desirable due its high speed of compression and decompression, free implementation that avoids patents and a good compression ratio. GZIP implements two different kinds of algorithms. L277 which generates a statistical model for the data provided and Huffman coding which maps the input data to bit strings with the help of the medal. The output provided by the Huffman code is a variable-length code where common symbols are represented using fewer bits than less common symbols.

Encryption is the process in which data is encoded in such a way that **only** authorized parties are able to access it. Encryption is important for various reasons. The main reasons being that encryption provides a means of securing information and prevents interception of data by unauthorized parties. It also allows for authentication. Encryption can be broadly categorized into two types. One being asymmetric encryption and the other being symmetric encryption. In asymmetric encryption, the encryption key is available to anyone to use while the decryption key can only be accessed by the receiving party. In symmetric encryption however, the encryption and decryption keys are the same thus both the sending and receiving party need to have the same key. There are a number of encryption algorithms that are widely used today. Some include Triple DES Encryption, RSA

Encryption, Twofish encryption algorithm just to mention a few. When deciding on an algorithm to use, we took into consideration simplicity, computational power required and size of data. With this, we decided to use the AES algorithm. A figure below shows the comparison between DES and AES

	DES	AES
Developed	1977	2000
Key Length	56 bits	128, 192, or 256 bits
Cipher Type	Symmetric block cipher	Symmetric block cipher
Block Size	64 bits	128 bits
Security	Proven inadequate	Considered secure

figure 1 : Comparison of AES and DES

Our choice of the AES was based on the fact that AES is proven to be more secure and powerful compared to the DES from the Block size, length to security provided by the two. AES is drastically better than ASE and most encryption algorithms out there

### **Feasibility analysis**

Looking at technical feasibility, the IMU is readily available. The software is also available. Algorithms that will be implemented are not patented and

therefore we will have the freedom to use them for our design. This also makes it legally feasible. This project is also economically feasible, the IMU can be obtained for as low as R230. This is within our capabilities. As mentioned earlier, the algorithms to be implemented are not patented and therefore we will not incur any costs in their implementation.

In terms of time, the project is feasible since the goal can be achieved within the time constraints of approximately two months. With the comprehensive timeline and well-spaced milestone checkpoints, the design can be completed, testing done and deployment completed.

Finally, in terms of operational feasibility, this project will help simplify the data collection and transmission process for oceanographers. It also provides important learning points and real-world exposure for students designing it.

### **Possible bottlenecks**

One of the possible bottlenecks we may face is integration of the algorithms with the Pi. The algorithm may not behave as expected. For this we would have to debug and implement comprehensive testing in order to fix it.

Another bottleneck may be unavailability of the IMU closeby. In addition to placing an order as soon as possible, we may have to generate sample data that is similar to the data provided by the IMU.



## **SUBSYSTEM DESIGN**

This section focuses on the design and implementation of the subsystems, sub subsystems and inter subsystems with their requirements and specifications. This section continues to describe the interaction of these building blocks of the project and visually represents their interactions through UML diagrams.

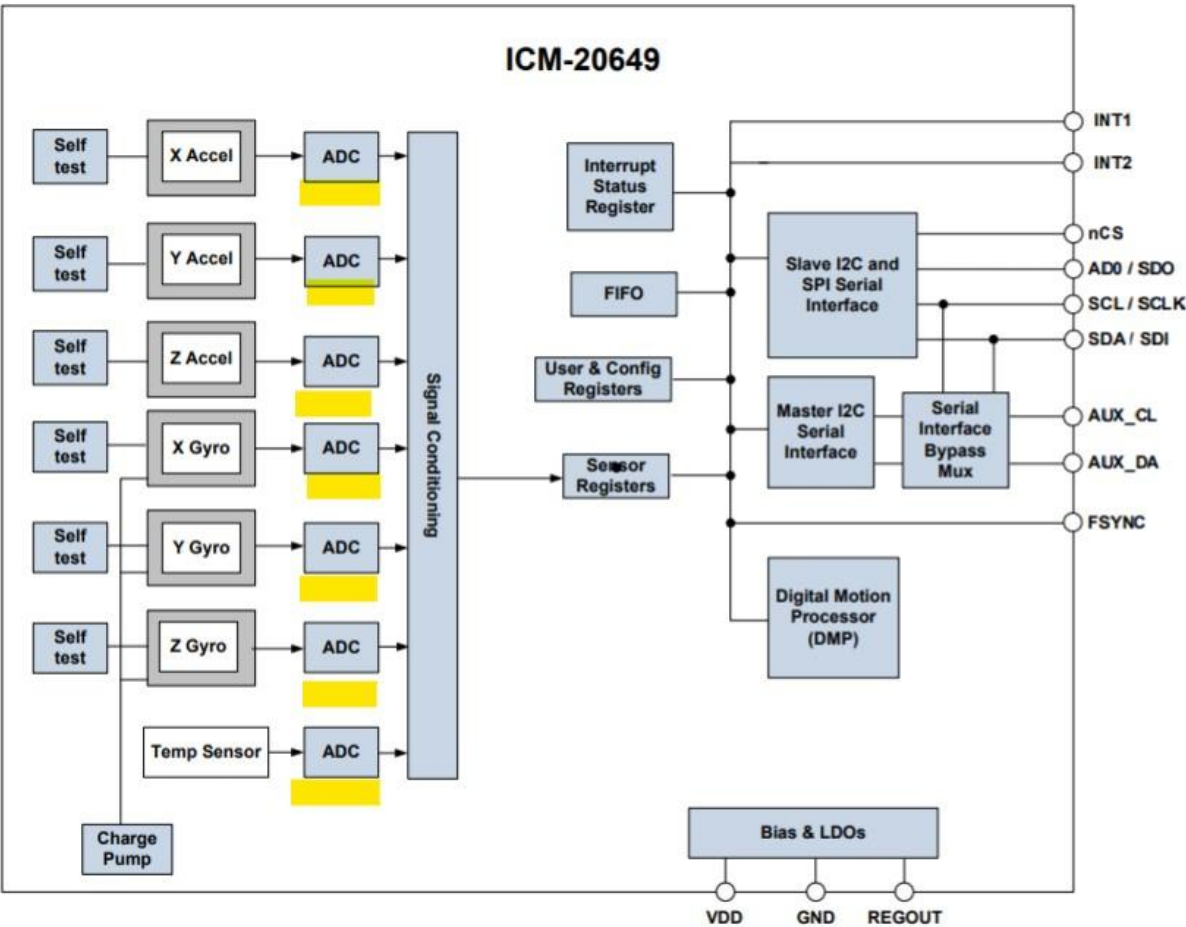
The main subsystems of the project are The IMU for data collection, Compression, and encryption of the collected data.their respective sub subsystems are mentioned below

### **1. The IMU and data collection**

This section entails the measurement of wave data using the inertial measurement unit .Wave parameter to focus on are wave heights and wave frequencies,dominant frequencies thus this device incorporate on board sensors that measure vertical acceleration, roll and pitch these parameter are measured using accelerometer and gyroscope thus all these functions are integrated in IC's called IMU's for this projects requirement the [ICM-20649](#) . The sampling rate for this application is chosen to be 20Hz which satisfies the Nyquist sampling criterion which simply states that the sampling frequency must be

at least twice the highest frequency thus for underwater application.

The IMU measures inertial axis reading which is digitised using 16 bit AD for each axis of the accelerometer and gyroscope as shown below in yellow



The data from the ADC are then stored in a register and a burst read operation is done to read the data in the sensor registers. Which then equals to the output of the [ICM-20649](#) the table below describes the output format of the [ICM-20649](#)

Table 1

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

This output from the IMU module is then read by the Master/ Raspberrypi to perform further processes on the data as per requirement of this project the I2c communication interface is established between the two for moving to the Raspberry Pi.

Requirement ID	description
IMUR001	System must be able to transmit data remotely without additional infrastructure or user input.
IMUR002	System must be capable of measuring ice floe dynamics and environmental conditions surrounding sea ice formation.
IMUR003	System must be able to

	store and process data in an organised manner
--	---

## **Specifications**

Specifications ID	description
IMUS001	Subsystems to be rated for 3:3 V to 5 V power.
IMUS002	Enclosure built using thermal resistant plastic.
IMUS003	Device to have a temperature operating range of 40 C to 20 C with 1 C uncertainty.

## **1. COMPRESSION**

Compression algorithms help optimize file size. Different algorithms provide different results. The compression sub-system takes input directly from the IMU through the I2C communication protocol. The data format provided by the IMU is text containing signed 16-bit numbers. The data is sampled at 100Hz.

GZIP compression is made up of both LZ77 and Huffman Coding.

LZ77 replaces repeated instances of data with references. The reference's format is as follows: <num1, num2> where num1 is the number of positions back the original data was and num2 is the length of the original data. This reference is easily decoded by the decoder. After the LZ77 has compressed, there are three types of data: literals(the data which could not be referenced), lengths and distances

The Huffman Coding uses bits(codes) to represent symbols using variable-length codes. The data is from the compressed output of the LZ77 algorithm. More frequent characters are assigned to shorter codes. The codes have a prefix which helps to identify the code before it. This makes decoding of the code easier.

This compression **outputs** two tables. One containing literals and lengths and the other containing distances.

### Requirements

CR001	Speed
Requirement	The compression should take the minimum possible time to reduce processor time
Verification	This will be verified by timing the circuit and

	comparing it with different algorithms.
--	---

CR002	<b>Data Retrieval</b>
Requirement	The algorithm should allow for at least 25% of the Fourier Coefficients to be retrieved.
Verification	This will be verified during the decryption of the encrypted data.

CR003	<b>Output data size</b>
Requirement	The algorithm should significantly reduce the size of the data
Verification	This can be verified by viewing the properties of the compressed data

## Specifications

CS001	<b>Data Retrieval</b>
Specification	The algorithm to be implemented will be GZIP which involves LZ77 and Huffman Coding which are lossless.

CS002	<b>Speed</b>
Specification	The algorithm to be implemented will be GZIP which involves LZ77 and Huffman Coding which is fast and efficient.

CS003	<b>Output Data Size</b>
Specification	The bit size of the output should be significantly lower than that of the input.

### **3.Encryption.**

The Advanced Encryption Standard (AES) is a block cipher, a form of shared secret encryption that was selected by the National Bureau of Standards as an official Federal Information Processing Standard (FIPS) which has subsequently enjoyed widespread use internationally. It is based on a symmetric-key algorithm that uses a 128-bit, 192-bit or 256-bit key. Making it exponentially stronger than 56bit key DES. The code has been written to provide readability and easy understanding of the algorithm.

This protection of data is to ensure security in case there's a third party and to ensure confidentiality in this project symmetric encryption is implemented which uses a shared secret key . The encryption strategy is a substitution permutation with multiple rounds to produce ciphertext where the number of rounds depends on the key size being used. On each round the following takes place, substitution of the bytes,

shifting the rows,mixing the columns and adding the round key.The algorithm can be accessed [here](#)

A bitwise operation at the core of encryption is where the plaintext in this case Raw data is XOR'd with the key to produce a ciphertext. A reverse operation can be done to obtain the original data by performing a bitwise XOR operation between the cipher text and the key. This can be understood as decryption lastly the cipher text in bits is safely stored with its corresponding key

Requirement ID	description
ER001	Device to contain sufficient memory for data storage.
ER002	Device to contain a processing unit to control sensors and process data.
ER003	Protect against unauthorized access to information

## Specifications

Specifications ID	description
ES001	System will use the Raspberry pi as a



	microcontroller. For managing resources and processes memory of the pi is sufficient
ES002	System will use the Raspberry pi as a microcontroller. For managing resources and processes
ES003	Encryption protocols thus the use of DES

#### **4.The System Overview**

The main subsystems of the project are The IMU for data collection, Compression, and encryption of the collected data.their respective sub subsystems are visually represented below which clearly shows the systems interaction with one another

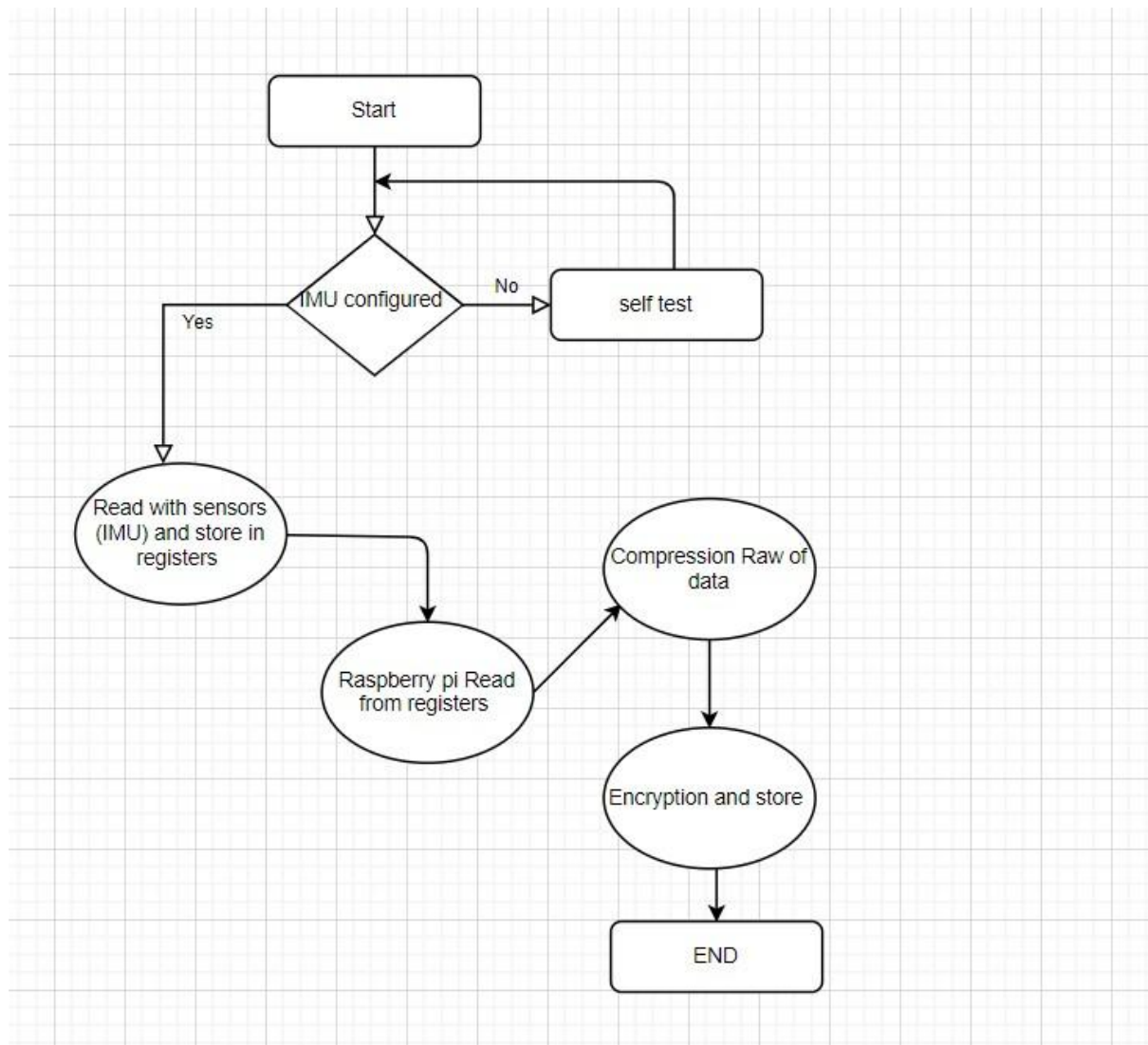


Figure 2

From this it's clear how the data flows. It's read by the Pi over the I2C protocol. The data is then fed into the compression subsystem where it is compressed. The compressed data is then fed into the encryption subsystem where it is encrypted and stored awaiting transmission.

#### Validation using Simulated or Old Data

Validation is the process of determining the degree to which a simulation model and its associated data are an accurate representation of the real world from the perspective of the intended uses of the model. This section discusses the validation process of the project, thus validation ensures that the system built is the right model. This inturn is a process focusing on determining whether the result of each step in the development of the simulation model.

### **Data Analysis**

For now, we will use sample csv data shown below, that resembles the actual data from the IMU that will be used in later stages of the project. We decided to use this because it is sampled from previous data provided by an IMU. However, once an IMU is present for use, the data from the IMU will be used.

The data was sufficient for various ATPs to be met.

One of the ATPs was that the compressed file should be smaller than the original file. For this to happen the data has to be sufficiently large such that the compression overhead does not make the compressed file larger. The data to be used is large enough for this ATP to be met.

Another ATP is that the algorithms should be as efficient as possible. For this to happen, the data needs to be ordered and organized in table form. The data to be used is in this form and it will therefore give us efficient results.

The data used also made it possible to plot appropriate time domain waveforms and from that, frequency domain waveforms because the time when the data was collected is also tabulated. From this, proper analysis can be made.

File	Home	Insert	Page Layout	Formulas	Data	Review	View	Help	Power Pivot	Tell me what you want to do	Share	
A1												
computer utc start 2018-09-19-03:57:21.506044												
UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM2, DCM3, DCM4, DCM5, DCM6, DCM7, DCM8, DCM9, MagNED1, MagNED2, MagNED3												
2018-09-19-03:57:21.717926 -0.14322271943092346 0.2232052981853485 0.123428575694561 0.15357686579227448 -0.30159956216812134 -9.795899391174316 0.00508350133895874 -0.001391227473												
2018-09-19-03:57:21.817726 -0.1410953253507614 0.21974442899227142 0.12359068542718887 0.15984362363815308 -0.30425748229026794 -9.779479026794434 0.005224071908742189 -0.001184811												
2018-09-19-03:57:21.917707 -0.14538313448429108 0.22439616918563843 0.12579116225242615 0.1569005399942398 -0.29826948046684265 -9.782461166381836 0.004827750381082296 -0.000948803												
2018-09-19-03:57:22.017733 -0.14438782632350922 0.2198091596364975 0.12465985864400864 0.15925079584121704 -0.30477917194366455 -9.774477005004883 0.005078970920294523 -0.001024233												
2018-09-19-03:57:22.117717 -0.14319761097431183 0.2232208251953125 0.1271720826625824 0.15469375252723694 -0.30872249603271484 -9.755866050720215 0.005107069853693247 -0.0009864562												
2018-09-19-03:57:22.217686 -0.14326296746730804 0.2209295630455017 0.1247049645894241 0.16080111265182495 -0.30884605646133423 -9.749424934387207 0.004865021910518408 -0.001169905												
2018-09-19-03:57:22.317733 -0.14428161084651947 0.22437554597854614 0.12585045397281647 0.15797586739063263 -0.3115788400173187 -9.74105453491211 0.004976013209670782 -0.0012970133												
2018-09-19-03:57:22.417797 -0.1465568244457245 0.2209947258234024 0.1257745325565338 0.16294534504413605 -0.3130553364753723 -9.739278793334961 0.004936043173074722 -0.001254770786												
2018-09-19-03:57:22.517726 -0.14545537531375885 0.22097420692443848 0.12583346664905548 0.16190342605113983 -0.3162236213684082 -9.728490829467773 0.004644147586077452 -0.001577646												
2018-09-19-03:57:22.617713 -0.1487094908952713 0.22331589460372925 0.1256270706653595 0.16106994450092316 -0.31497976183891296 -9.719582557678223 0.004643251188099384 -0.0008518678												
2018-09-19-03:57:22.717685 -0.14646583795547485 0.22555047273635864 0.12446887791156769 0.16358499228954315 -0.3218514621257782 -9.71753215789795 0.004382750950753689 -0.0008635037												
2018-09-19-03:57:22.817678 -0.14661410450935364 0.21983402967453003 0.12079749256372452 0.15981677174568176 -0.32272082567214966 -9.700324058532715 0.0040258122608065605 -0.00132335												
2018-09-19-03:57:22.917679 -0.14653174579143524 0.22326011955738068 0.12200228124856949 0.16515518724918365 -0.3259122967720032 -9.702316284179688 0.004099557176232338 -0.001032733												
2018-09-19-03:57:23.017675 -0.1465712934732437 0.22098377346992493 0.12327811121940613 0.16342021524906158 -0.3260689675807953 -9.694110870361328 0.003974369261413813 -0.0014241506												
2018-09-19-03:57:23.117681 -0.1454542875289917 0.22209881246089935 0.12207549065351486 0.1666800081729889 -0.31716904044151306 -9.68447494506836 0.003931220155209303 -0.0012432601C												
2018-09-19-03:57:23.217676 -0.14983339607715607 0.22331994771957397 0.121823765374672 0.17714177072048187 -0.3237636983394623 -9.685702323913574 0.003674454987049103 -0.0012060992												
2018-09-19-03:57:23.317665 -0.14878103137160296 0.2210192233324051 0.12191178649663925 0.1770520806312561 -0.32357630133628845 -9.676258087158203 0.00353451631963253 -0.0013650835C												
2018-09-19-03:57:23.417658 -0.14652030169963837 0.22213870286941528 0.12700651586055756 0.17423927783966064 -0.32891881465911865 -9.67294692993164 0.00335993361659348 -0.0015583423												
2018-09-19-03:57:23.517702 -0.14875587821006775 0.22103434801101685 0.12565478682518005 0.17514801025390625 -0.3213691711425781 -9.664628028869629 0.0028928497340530157 -0.00168295												
2018-09-19-03:57:23.617766 -0.14763009548187256 0.22215397655963898 0.12569952011108398 0.17808954417705536 -0.3192858397960663 -9.660322189331055 0.002587320050224662 -0.001564605												
2018-09-19-03:57:23.717760 -0.1475641429424286 0.22669392824172974 0.12065038830041885 0.17370761930942535 -0.3249348998069763 -9.668174743652344 0.002747015794739127 -0.0014418582												
2018-09-19-03:57:23.817687 -0.1476208120584488 0.222158282995224 0.1269468367099762 0.17589013278484344 -0.31988441944122314 -9.657933235168457 0.0023824432864785194 -0.0017717864E												
2018-09-19-03:57:23.917668 -0.14863012731075287 0.22673363983631134 0.12558147311210632 0.17027561366558075 -0.3208599090576172 -9.656055450439453 0.002090258290991187 -0.00181777C												
2018-09-19-03:57:24.017660 -0.14868749678134918 0.22332346439361572 0.12812024354934692 0.17353814840316772 -0.3253801465034485 -9.66348648071289 0.0020196966361254454 -0.00188305E												
2018-09-19-03 57 11 VN100												

The table below from the paper design specifies the expected data from the IMU which corresponds to the special axial components specified below. This data is then compressed and encrypted from one point for example locally (PC) which prepares for remote transfer of the files which are already secured, thus for the purpose of transmitting. Compression and encryption won't be necessary if there was no remote transmission of data.

Table 1


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

## EXPERIMENTAL SETUP

### Entire 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

 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])
```

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:

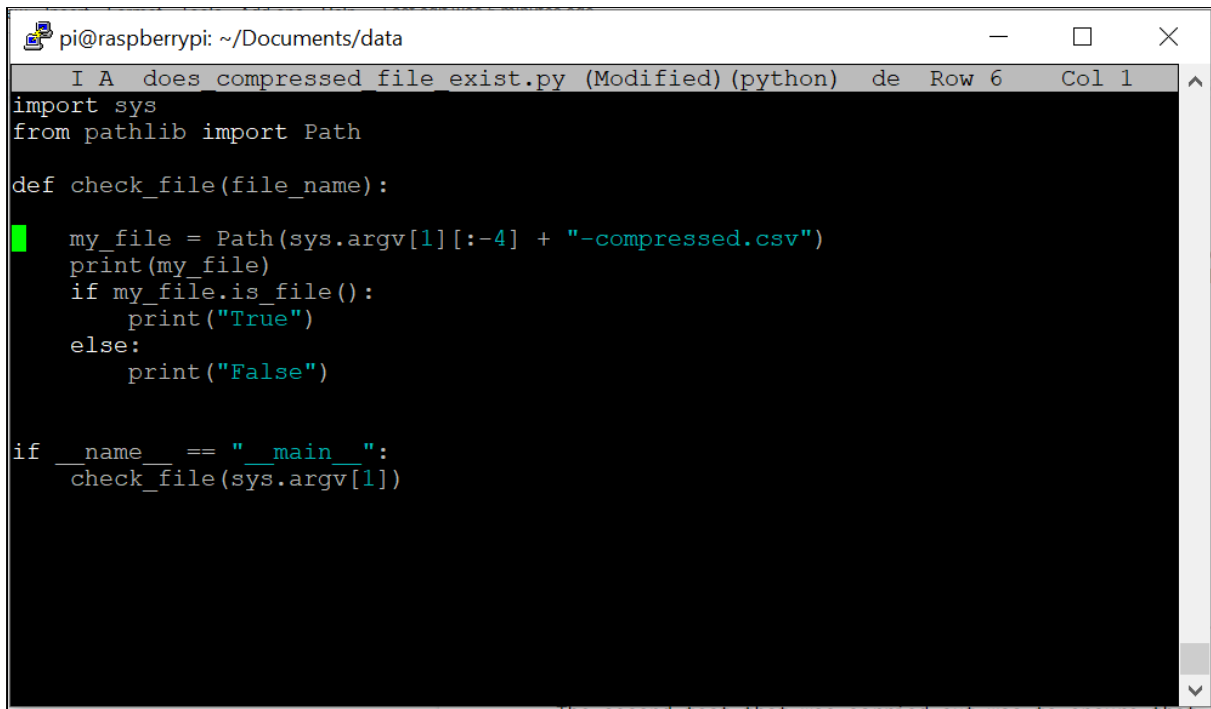
```
sudo i2cdetect -y 1
```

The final 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.

### **Compression**

For compression, a couple of tests were carried out to validate the subsystem. For the purposes of this project, the input was in csv format from the IMU. The output was both in csv and text. The csv was to provide a good comparison between it and the original while the text file was the preferred input of the encryption system.

The first test that was carried out was to check whether the compression subsystem actually produces a compressed file. To do this, the data csv file was fed into the subsystem. The subsystem will then create a csv and text file with a "compressed" suffix. A python script (does\_compressed\_file\_exist.py) was written to implement this. If there is a compressed version of the file, it will return true and false otherwise. It's expected that a file is present after the compression algorithm is run. The name of the file being tested with is "2018-09-19-06\_53\_21\_VN100.csv"

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~/Documents/data'. The code editor shows a Python script named 'does\_compressed\_file\_exist.py'. The code imports 'sys' and 'Path' from 'pathlib'. It defines a function 'check\_file(file\_name)' that constructs a path 'my\_file' by appending '-compressed.csv' to the first command-line argument. It then checks if 'my\_file' exists and prints 'True' or 'False'. A main block checks if the script is being run directly and calls 'check\_file(sys.argv[1])'.

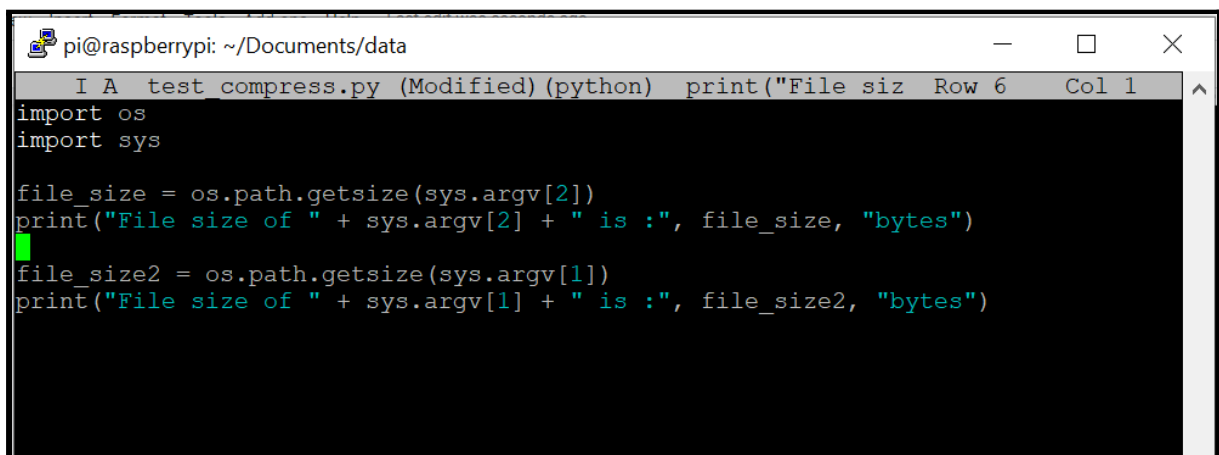
```
pi@raspberrypi: ~/Documents/data
I A does_compressed_file_exist.py (Modified) (python) de Row 6 Col 1
import sys
from pathlib import Path

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 2.1

The second test that was carried out was to ensure that the compressed file was smaller in size as compared to the original file. To effectively carry this out, native python functions in the test\_compress.py file were used to compare the size of the two files in terms of bytes and a comparison was carried out. It's expected that the compressed file will be significantly smaller.

A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~/Documents/data'. The code editor shows a Python script named 'test\_compress.py'. The code imports 'os' and 'sys'. It gets the size of the file at 'sys.argv[2]' and prints it. Then it gets the size of the file at 'sys.argv[1]' and prints it. The code is partially visible.

```
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 2.2

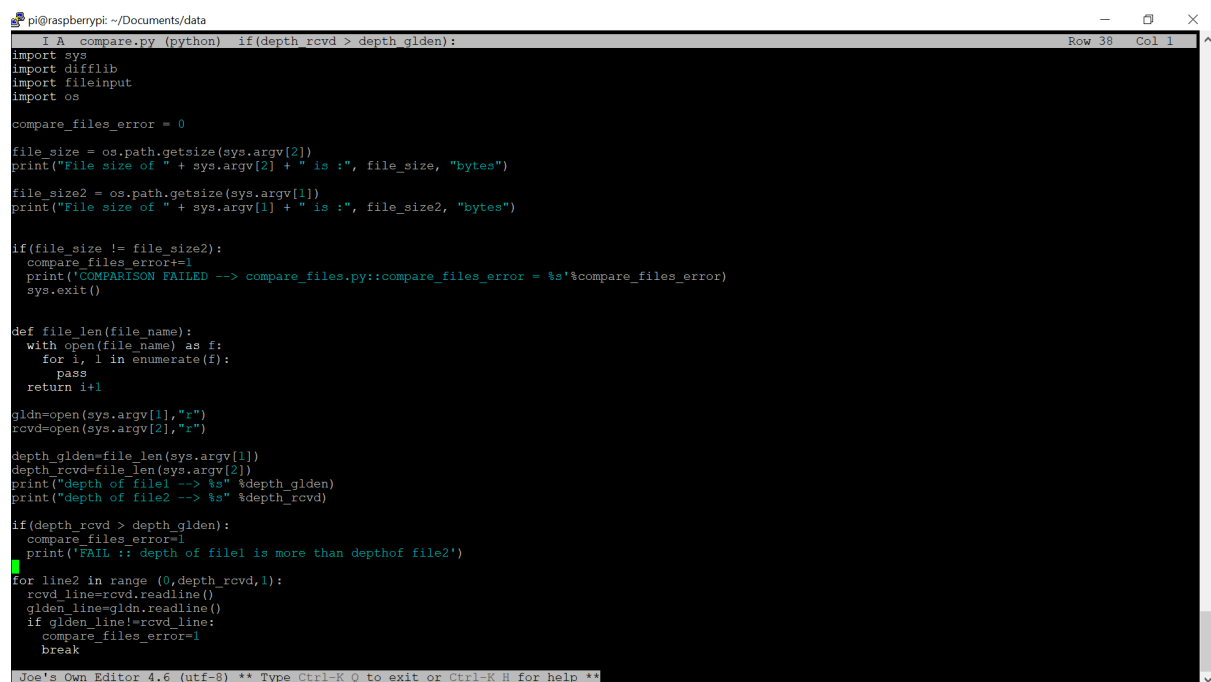
After getting the file sizes of both the original data and the compressed file, the compression ratio of the algorithm was calculated. The following formula was used:

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

Figure 2.3

This was tested for different sizes and types of data and the results tabulated. It's expected that the compression ratio should be similar or deviate by a small factor. It's also expected that the compression ratio will be small, within the range of 1 to 4 due to the lossless nature of the algorithm.

Another test was carried out to make sure that the decompressed file was the same as the original file thus ensuring no loss of data. To do this, three different tests were carried out in the same python script (compare.py). 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. It's expected that there will be no loss in the data since the gzip algorithm used is lossless. The code is shown in Figure 2.4.



```

pi@raspberrypi: ~/Documents/data
I A compare.py (python) if(depth_rcvd > depth_gldn):
import sys
import difflib
import fileinput
import os

compare_files_error = 0

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")

if(file_size != file_size2):
    compare_files_error+=1
    print("COMPARISON FAILED --> compare_files.py::compare_files_error = %s"%compare_files_error)
    sys.exit()

def file_len(file_name):
    with open(file_name) as f:
        for i, l in enumerate(f):
            pass
        return i+1

gldn=open(sys.argv[1],"r")
rcvd=open(sys.argv[2],"r")

depth_gldn=file_len(sys.argv[1])
depth_rcvd=file_len(sys.argv[2])
print("depth of file1 --> %s" %depth_gldn)
print("depth of file2 --> %s" %depth_rcvd)

if(depth_rcvd > depth_gldn):
    compare_files_error=1
    print("FAIL :: depth of file1 is more than depth of file2")

for line2 in range(0,depth_rcvd,1):
    rcvd_line=rcvd.readline()
    gldn_line=gldn.readline()
    if gldn_line!=rcvd_line:
        compare_files_error=1
        break

Joe's Own Editor 4.6 (utf-8) ** Type Ctrl-R Q to exit or Ctrl-R H for help **

```



*Figure 2.4*

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.

### **Encryption and Decryption :**

Below is a set of steps taken to check whether a file was successfully encrypted/decrypted or not. Data used is this section as input a text file and output text.enc file for the encryption part. Lastly input for decryption is a text.enc file and outputs the original txt file :

1. First volition was checking whether the encryption 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
2. Secondly, making sure that no one else can access the

original data without the decruping 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.

3. Thirdly, 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.

Just examining the file using tools like "strings" will also give you a clue, but won't be as definitive as binwalk. If you find actual readable strings in the file, it's not encrypted. However, some forms of compression will often look like encryption. If there's a readable header in the file, then it's likely compressed and not encrypted.

## 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	0.027378985
2	56890	0.15123934

3	623134	0.6729736324
4	1247263	1.27062843
5	6240886	6.5621285

Table 2.1

Using the data above, the graph below was plotted:

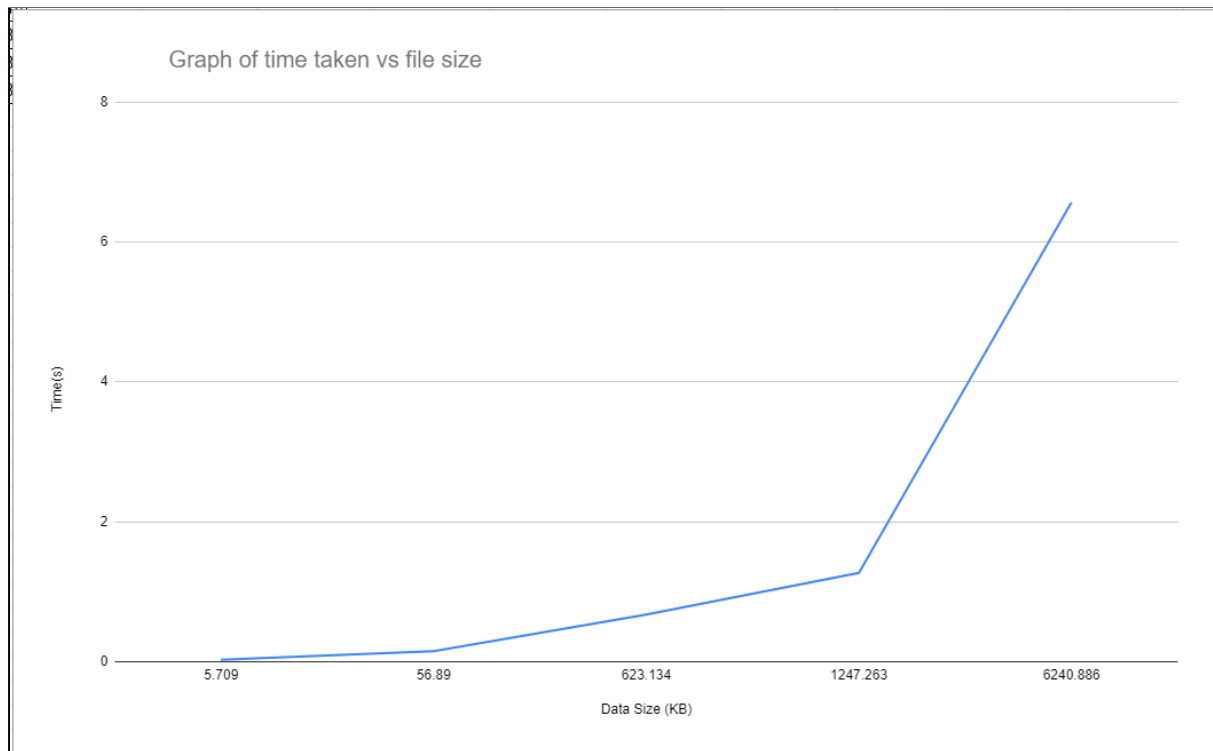


Figure 3.1

It was seen that the time increased exponentially as the file size increased.

## 2. Desired output test

Using the python script, a check was done to determine w

```
pi@raspberrypi: ~/Documents/data
```

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

This confirmed that indeed the required output was produced by the system. The validity of the output was tested in the later stages of the tests.

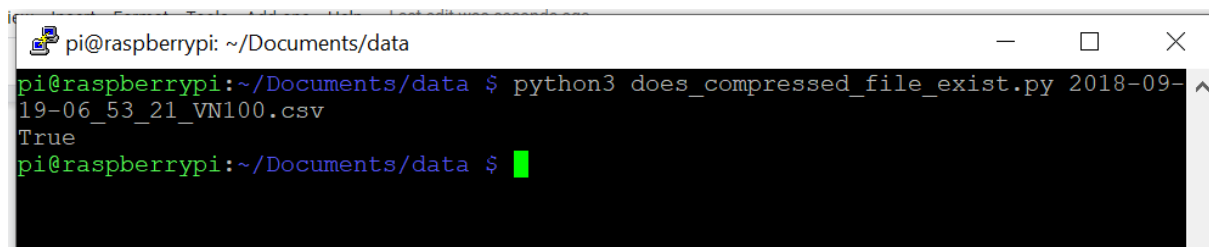
### **3. Data transmission test**

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

#### **Compression:**

##### **1. Compressed file is produced**

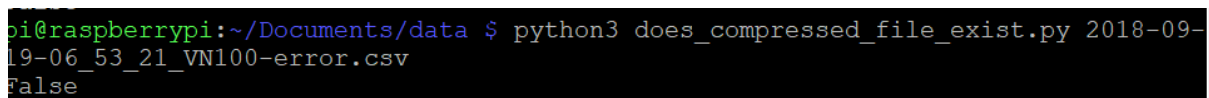
After running the script, the following result was obtained:

A terminal window on a Raspberry Pi with the title bar 'pi@raspberrypi: ~/Documents/data'. The command 'python3 does\_compressed\_file\_exist.py 2018-09-19-06\_53\_21\_VN100.csv' is entered and executed. 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 3.2

To test for when there's no file, the suffix "-error" was added to the file name in the command line arguments.

A terminal window on a Raspberry Pi with the title bar 'pi@raspberrypi: ~/Documents/data'. The command 'python3 does\_compressed\_file\_exist.py 2018-09-19-06\_53\_21\_VN100-error.csv' is entered and executed. The output is 'False'.

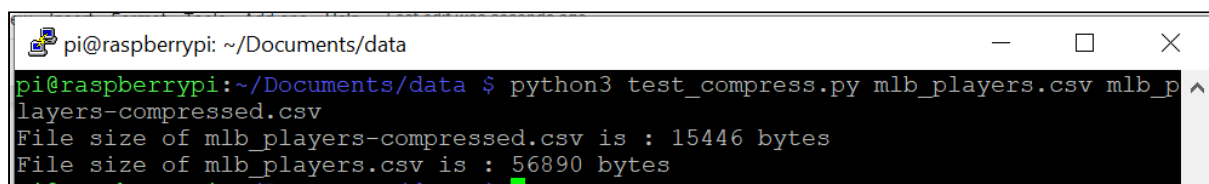
```
pi@raspberrypi:~/Documents/data $ python3 does_compressed_file_exist.py 2018-09-19-06_53_21_VN100-error.csv
False
```

Figure 3.3

This shows that the compression algorithm did indeed produce a file. This file will be tested further to ensure its validity.

##### **2. The size of the compressed file**

With the script shown in Figure 2.2, the following result was obtained.

A terminal window on a Raspberry Pi with the title bar 'pi@raspberrypi: ~/Documents/data'. The command 'python3 test\_compress.py mlb\_players.csv mlb\_players-compressed.csv' is entered and executed. The output shows the file sizes in bytes.

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

Figure 3.4

The size of the original file was 56.890 KB and the compressed file is 15.446 KB. This led to the conclusion that the compression algorithm indeed compressed the file as required.

### **3. Compression ratio**

The formula shown below was used to determine the compression ratio of different data sizes and results were tabulated.

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

	<b>Data size</b>	<b>Compression ratio</b>
1	5709	3.0859459
2	56890	3.6831542
3	623134	3.160918
4	1247263	3.175062
5	6240886	3.187314

Table 3.1

The compression ratio of the algorithm with different sets and sizes of data was **3.2584**.

The compression ratio is not high due the algorithm's lossless nature. The compression ratio of lossless algorithms range from 1:1 to 4:1. The higher the compression ratio, the smaller the file size. However, this will take significantly more resources to decompress back to the original file. Thus a compression ratio of 3.25 provides a good balance between size of compressed file and ease of decompression.

### **4. Loss of data**

The algorithm used is a lossless algorithm and therefore it was expected that the decompressed file would be the same in

size and contents. Running the tests explained earlier, we obtained the following results.(Name of the original file is data.csv and the name of the decompressed file is data-v2.csv)

```
pi@raspberrypi:~/Documents/data $ python3 compare.py data.csv data-v2.csv
File size of data-v2.csv is : 9071107 bytes
File size of data.csv is : 9071107 bytes
depth of file1 --> 14940
depth of file2 --> 14940
COMPARISION SUCCESSFUL --> compare_files.py::compare_files_error = 0
```

Figure 3.5

It was noted that the files had the same size, length and content and therefore 0 errors present.

In order to fully test it, a bit was added to data-v2.csv to check whether the script would return an error. This is what was observed.

```
pi@raspberrypi:~/Documents/data $ python3 compare.py data.csv data-v2.csv
File size of data-v2.csv is : 9071108 bytes
File size of data.csv is : 9071107 bytes
COMPARISION FAILED --> compare_files.py::compare_files_error = 1
```

Figure 3.6

Upon adding a bit, the script returned an error. This further confirmed that the original file and the compressed file were equal and there was no data loss

## 5. Time taken to compress data

The algorithm was tested using different data sizes and the results were recorded in Table 1 below and a corresponding graph was plotted.

	File size (bytes)	Time taken(s)
1	5709	0.002556085
2	56890	0.12375044
3	623134	0.614487886

4	1247263	1.1698408
5	6240886	6.1944408

Table 3.2

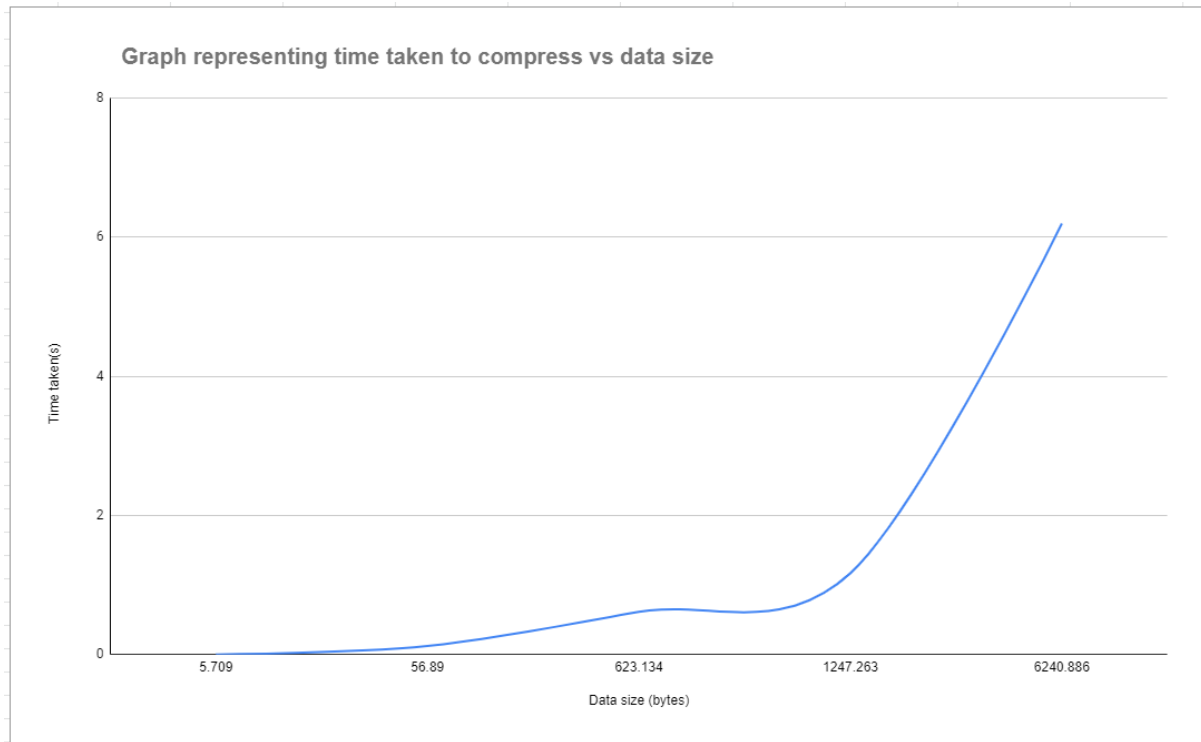


Figure 3.7

From the results above, we can see that the time taken increases exponentially and not linearly as the data size grows thus the smaller the data size, the faster the compression

## Encryption:

### 1. Recovery test

The results are analyzed based on the implementation discussed above. figure 2.1 below is 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 -0.34008175134658813 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 0.18888604640960693 -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 0.3586127758026123 0.3057440221309662 0.14544525742530823 -0.2623203

```

Figure 4.1:

The file shown in figure 2.1 is encrypted through the script shown in figure 2.2 below. The resulting encrypted file is shown in figure 2.3

```

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]])

```

Figure 4.2 : encrypt.py script





Figure 4.3: content of the encrypted file

For the purpose of this experiment, checking if the encryption mechanism is reversible or not the very same file shown above figure 2.3 is run through the decryption script namely decrypt.py shown in figure 2.4 the results from that process is shown in figure 2.5 this does not only prove successful encryption but also decryption.



figure 4.4:decrypt.py

```

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 0.18888604640960693 -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 0.3586127758026123 0.3057440221309562 0.14544525742530823 -0.2623203

```

figure 4.5

## 2. Key integrity testing

The outcome from changing the key when decrypting clearly shows that one cannot recover the original data. Above the figure 2.1 was encrypted with the key show in figure 2.6 and later decrypted with a different 16it key shown in figure 2.8 and the outcome is represented by figure 2.9 This proves and validates security as promised by the AES algorithm

```

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

```

4.6: new encryption 16 bit key



4.7:result from figure 2.6

```
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
```

4.8:decrypting key(original key for the projet)



4.9: result from figure 2.8

### 3. 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

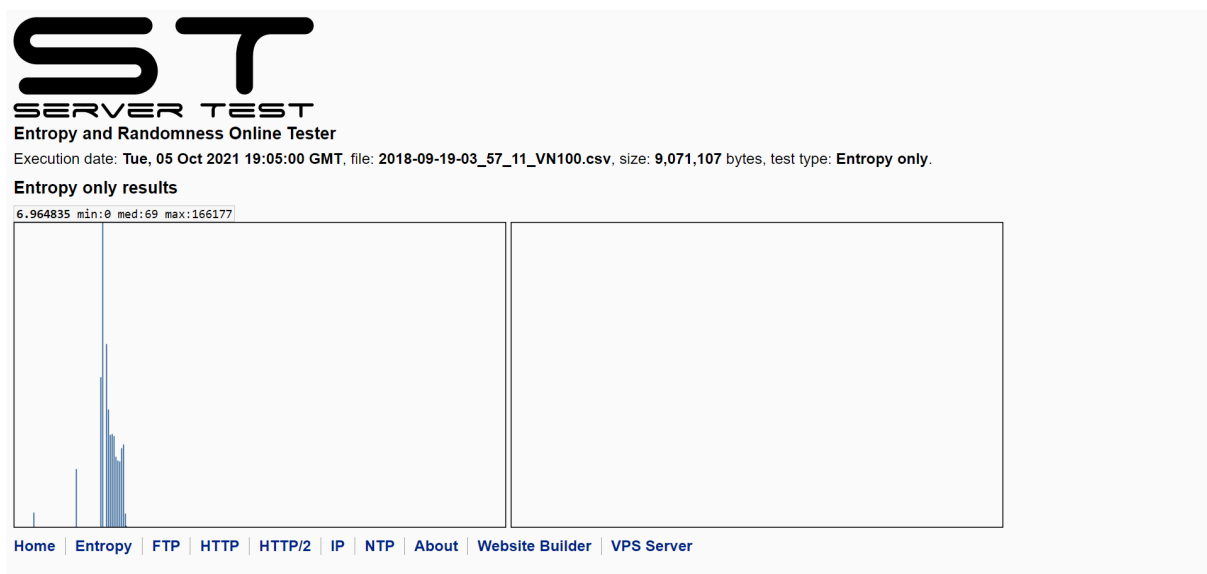


Figure 5.1: file2018-09-19-03\_57\_11\_VN100.CSV

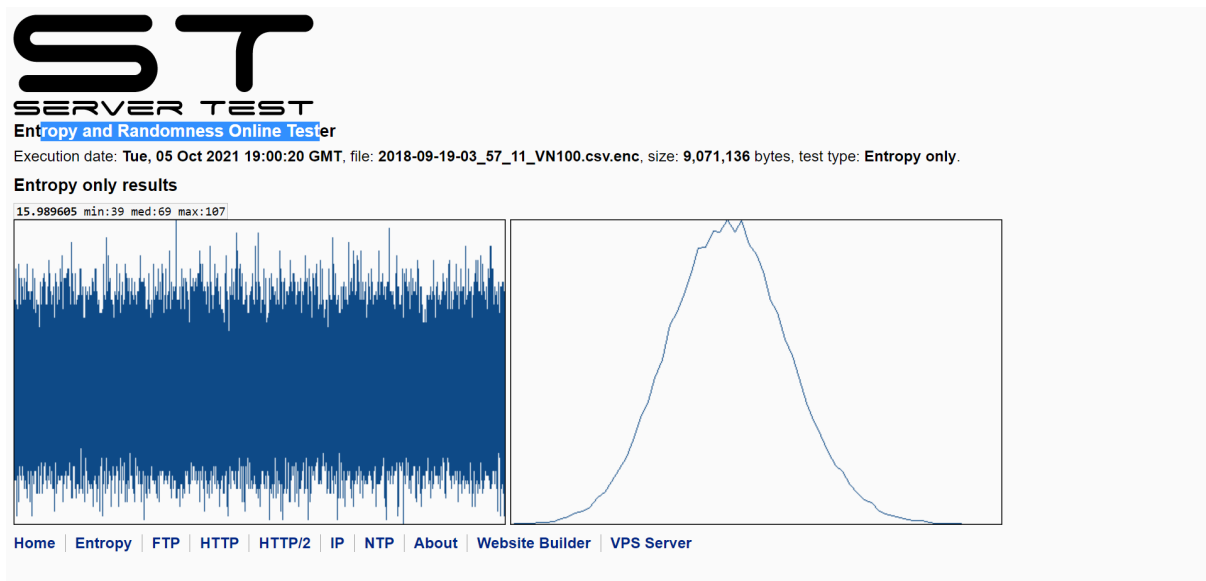


Figure 5.2:file 2018-09-19-03\_57\_11\_VN100.CSV.ENC

### Validation using a different-IMU

Hardware based validation and testing is function focused thus allowing testing and validation of each and every hardware's function in respective subsystems. Hardware based validation is also an early stage integrator which recognizes any short falls and inadequacies of the function based approach allowing early stage improvements in the process of development.

### IMU Module

The IMU used for this practical is the IMU-20948. It is a low power 9-axis motion tracking device. It has several components that make up a comprehensive tracking device: 3-axis gyroscope that detects the deviation of an object from a desired orientation, 3-axis accelerometer to measure linear and angular acceleration, 3-axis compass for general direction, Digital Motion Processor™ to detect motion effectively, a



digital-output temperature sensor to measure the surrounding temperature. It's because of these that the IMU-20948 has several applications such as in mobile phones, wearable devices, drones, IoT devices just to mention a few.

The device used in the shark-buoy is the IMU-20649. It's similar to the one used here albeit with a few differences. It also has a 3-axis gyroscope, 3-axis accelerometer, digital-output temperature sensor. They both have a VDD operating range of 1.71V to 3.6V. They both have on-Chip 16-bit ADCs and Programmable Filters.

In terms of differences, the IMU-20649 is a 6-axis motion tracking device while the IMU\_20948 is a 9-axis motion tracking device. The IMU-20948 therefore offers a bit more data.

### **Extrapolation**

The final project would be used in the poles in shark buoys to capture various forms of data such as wave dynamics. The 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.

### **Validation tests**

A couple of tests were carried out to ensure that the IMU-20949 was working as expected

### **a. The IMU is connected to the Raspberry Pi(RPi)**

A couple of tests were carried out to ensure that the IMU-20949 was working as expected

The first test that was done was to check whether the IMU was properly connected to the RPi. This was done so as to ensure that the required data would actually be collected from the surroundings by the IMU.

To do this, the following command was run on the RPi:

```
sudo i2cdetect -y 1
```

After running this, the following result 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 1.

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

### **b. The IMU produces data output**

The second test was to make sure that the IMU actually produced the desired output. To do this, the temperature sensor was put to use. A python script, LPS22HB.py, was used to collect data from using the temperature sensor. The script can be found [here](#).

The following command was run to run the script and copy the contents in a file, pressure\_and\_temp.csv.

```
$ python LPS22HB.py >> pressure_and_temp.csv
```

The file 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 file produced, which can be found [here](#), contains the pressure in the first column and temperature in the second column. The temperature was around the desired temperature. This confirmed that the IMU worked as expected.

### **c. The IMU's response to movements**

The IMU is known to automatically handle motion profiles especially using the accelerometer which measures proper acceleration it experiences relative to freefall and 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

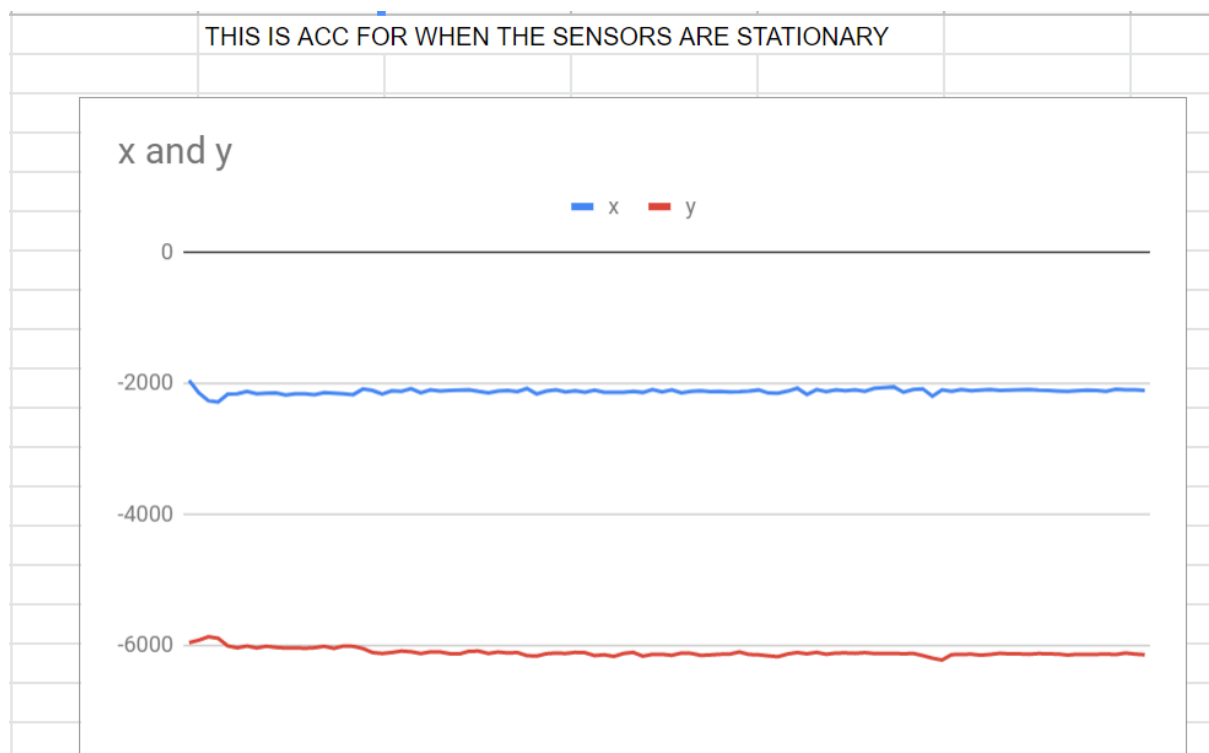


figure 1



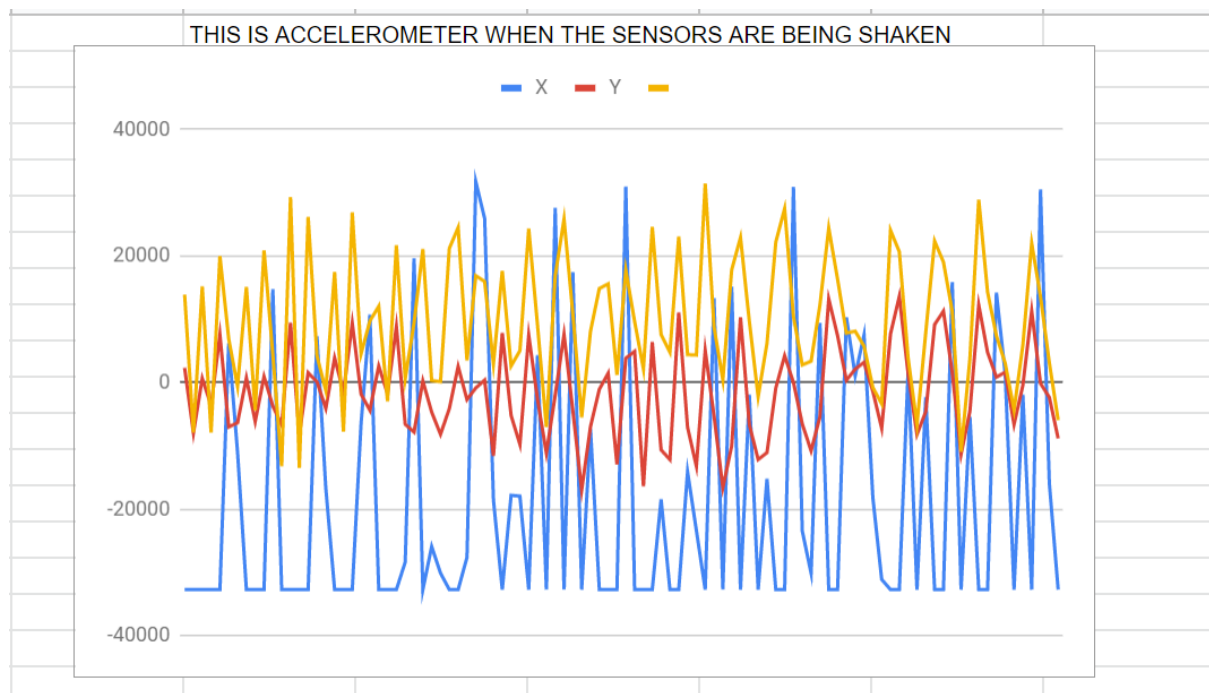


figure 2

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

#### **d. The IMU's response to temperature**

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, which can be found [here](#), was around 25 degrees Celsius while the temperature readings in the refrigerator, which can be found [here](#), 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.

The data used is the data collected from the IMU. It contains data on different aspects of the environment such as temperature, pressure and acceleration.

### **ATPS**

#### **Entire System:**

<b>AT001</b>	<b>Configuration of the Slave</b>
Evaluation type	Hardware
Target	I2C connection
Test protocol	<ul style="list-style-type: none"> <li>• Connect the IMU to the master, read and write from IMU registers</li> <li>• Additional self test</li> </ul>
Pass condition	<ul style="list-style-type: none"> <li>• Start and stop condition generation</li> <li>•</li> </ul>
Fail condition	No response from the IMU
Test result	Pass

<b>AT002</b>	<b>Desired output produced</b>
Evaluation type	Software
Target	Entire system
Test protocol	<ul style="list-style-type: none"> <li>• Feed a file into the system.</li> </ul>

	<ul style="list-style-type: none"> <li>Examine the output produced</li> </ul>
Pass condition	<ul style="list-style-type: none"> <li>A compressed and encrypted file is produced.</li> </ul>
Fail condition	A file that is not both compressed and encrypted is produced.
Test result	Pass

### Compression:

<b>AT003</b>	<b>Compressed file size test</b>
Evaluation type	Software
Target	Compression subsystem
Test protocol	<ul style="list-style-type: none"> <li>Feed the sample data into the subsystem</li> <li>Determine and compare the size of the output with the original</li> </ul>
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

<b>AT004</b>	<b>Data loss test</b>
Evaluation type	Software
Target	Compression subsystem
Test protocol	<ul style="list-style-type: none"> <li>• Feed the sample data into the subsystem</li> <li>• Extract the output of the subsystem and decompress it.</li> <li>• Compare this with the original file</li> </ul>
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

#### **Added ATPs**

<b>AT006</b>	<b>File produced test</b>
Evaluation type	Software
Target	Compression subsystem
Test protocol	<ul style="list-style-type: none"> <li>• Feed the sample data into the subsystem</li> <li>• Determine whether an output is produced by the system</li> </ul>
Pass condition	A file is produced by the subsystem
Fail condition	No file is produced
Test result	Pass

All the previous ATPs were met and therefore the

specifications do not change.

### Encryption and Decryption :

Results of the experiments done above are compared with our ATPs in this section.

AT007	Encryption/decryption algorithm correctness and successful execution
Evaluation type	Software
Target	Encryption algorithm
Test protocol	<ul style="list-style-type: none"><li>• Simulation of the encryption and decryption process using the open source code <a href="#">here</a></li></ul>
Pass condition	<ul style="list-style-type: none"><li>• Successfully encrypt the given data</li><li>• Successfully decryption of the given encrypted file with a key back to the original data</li></ul>
Fail condition	<ul style="list-style-type: none"><li>• Non recovery of the original data</li></ul>
Test results	pass

AT004 was successfully executed and it follows that

the it is met

#### Added ATPs

AT008	Entropy and Randomness Test
Evaluation type	Software
Target	Encrypted files
Test protocol	<ul style="list-style-type: none"><li>• Entropy formula measures the equiprobability regardless of real <a href="#">unpredictability</a>.</li><li>• Tool can be found <a href="#">here</a></li></ul>
Pass condition	<ul style="list-style-type: none"><li>• High Entropy score</li><li>• Dense Entropy graph is more desirable for a pass condition</li><li>• See figure 3.1</li></ul>
Fail condition	<ul style="list-style-type: none"><li>• Low entropy score</li><li>• Non dense Entropy graph</li></ul>
Test results	pass

AT009	Encryption/decryption algorithm secured key
Evaluation type	Software
Target	Encryption and decryption algorithm
Test protocol	<ul style="list-style-type: none"> <li>• Simulation of the encryption and decryption process using a different key for these stages</li> </ul>
Pass condition	<ul style="list-style-type: none"> <li>• Non recovery of the original data</li> <li>• See under experiment results 2</li> </ul>
Fail condition	<ul style="list-style-type: none"> <li>• Successfully encrypt the given data</li> <li>• Successfully decryption of the given encrypted file with a key back to the original data</li> </ul>
Test results	Pass

For the module to be used, the following things will be done to it:

1. Test with large amounts of data

2. Data transfer system in real-time
3. Test with different data formats.
4. Protect the module from damage due to the harsh conditions.

## Summary

In conclusion, the main aim of this project was to develop a compression and encryption system that would be used in shark buoys in collecting data on wave dynamics in the poles. This would be made possible by an IMU connected to a Raspberry Pi. The IMU is a motion sensor which provides data on various environmental aspects. For the project, the actual environment that the IMU would be used in was recreated in various ways. The two subsystems for this project were the compression subsystem and encryption subsystem. For the compression, the GZIP method was found to be efficient and therefore implemented. For the encryption, AES encryption was used. Various tests, as stipulated in the ATPs, were run on these subsystems to validate their implementation and to ensure that data was not lost in the process. After integrating the subsystems, various tests were also done to ensure that both work together in harmony. All the requirements were met and the project timeline and schedule was kept

## References

1. 2019. [online] Available at:  
<<https://maker.pro/raspberry-pi/tutorial/how-to-interface-an-imu-sensor-with-a-raspberry-pi>> [Accessed 25 October 2021].
2. Kangangi, M. and Khuzwayo, L., 2021.  
*EEE3097S\_2021\_Progress Report1\_Group14\_KHZLIN012\_KNGMAR027*. [online] Available at:  
<<https://docs.google.com/document/d/1ZqVSIy0hdStPsbXiPyGjvoC58lTP5-Lu1gdcwspcIME/edit?usp=sharing>> [Accessed 25 October 2021].
3. n.d. *World's Lowest Power 9-Axis MEMS MotionTracking™ Device*. [ebook] TDK Invensense. Available at:  
<<https://3cfeqx1hf82y3xcoull08ihx-wpengine.netdna-ssl.com>



/wp-content/uploads/2021/07/DS-000189-ICM-20948-v1.4.pdf>  
[Accessed 25 October 2021].

4. Kangangi, M. and Khuzwayo, L., 2021.

*EEE3097S\_2021\_Progress*

*Report2\_Group14\_KHZLIN012\_KNGMAR027*. [online] Available  
at:

<[https://docs.google.com/document/d/1vuWEgAUddn1rEPrixRu-wPTdunZmMlP25Zyn2jVa\\_K0/edit](https://docs.google.com/document/d/1vuWEgAUddn1rEPrixRu-wPTdunZmMlP25Zyn2jVa_K0/edit)> [Accessed 25 October 2021].

5. Kangangi, M. and Khuzwayo, L., 2021.

*EEE3097S\_2021\_PAPER\_DESIGN\_Group14\_KHZLIN012\_KNGMAR027*.

[online] Available at:

[https://docs.google.com/document/d/15DTUUm3UsI5eIc0xjPCQBPFpSTGmCIU3CPde\\_II20w4/edit](https://docs.google.com/document/d/15DTUUm3UsI5eIc0xjPCQBPFpSTGmCIU3CPde_II20w4/edit)