University of Cape Town

EEE3097S

Design

Progress Report 01

Rory Schram
SCHROR002

Natasha Soldin
SLDNAT001

12/09/2022

Contents

1 Introduction **1**

2 Admin **1**

2.1 Contributions . . . 1
2.2 Project Management Tool and Timeline . . . 1
2.3 GitHub . . . 2
2.4 Demonstration . . . 2

3 Data Analysis **3**

3.1 Accelerometer Data Analysis . . . 3
3.2 Magnetometer Data Analysis . . . 4
3.3 Gyroscope Data Analysis . . . 6

4 Experiment Setup **8**

4.1 Simulations . . . 8
4.2 Compression Block . . . 8
4.3 Encryption Block . . . 9

5 Results **10**

5.1 Compression Block . . . 11
5.2 Encryption Block . . . 13

6 Acceptance Test Protocols **15**

References **18**

A Appendix **19**

A.1 Compression Python Algorithm . . . 19
A.2 Encryption Python Algorithm . . . 20
A.3 Comparison Python Algorithm . . . 21
A.4 Timing Python Algorithm . . . 22

1 Introduction

The following project design is for an ARM based digital IP using an STM32F051-microcontroller. This design aims to retrieve, compress and encrypt data from an Inertial Measurement Unit (IMU) sensor. This type of sensor includes an Accelerometer, a Gyroscope, a Magnetometer and a Barometer. This design will be implemented as a buoy installed on an ice 'pancake' in the Southern Ocean to collect data about the ice and wave dynamics.

This data will then be transmitted using the Iridium communication network, which is a global satellite communications network. However, this is extremely costly and therefore the data would need to be compressed to reduce its size. The data is also encrypted for security purposes.

The following report includes a progress report which states the simulation focused experiment or first attempt of project software implementation. It includes a data analysis, experiment setup, results and acceptance test procedures.

2 Admin

2.1 Contributions

Each team member's contributions for the following progress report are tabulated in table 1 below.

Rory Schram	Compression Algorithm Simulation
	Experiment Setup
	Results
Natasha Soldin	Encryption Algorithm Simulation
	Data Analysis
	Acceptance Test Procedure

Table 1: Contribution Table

2.2 Project Management Tool and Timeline

This project was managed using Asana as its project management software. With which a timeline was created that helped the team members stay up to date on tasks and deliverables. Screenshots of the timeline are attached in figure 2.2.1 below. The project is on track.

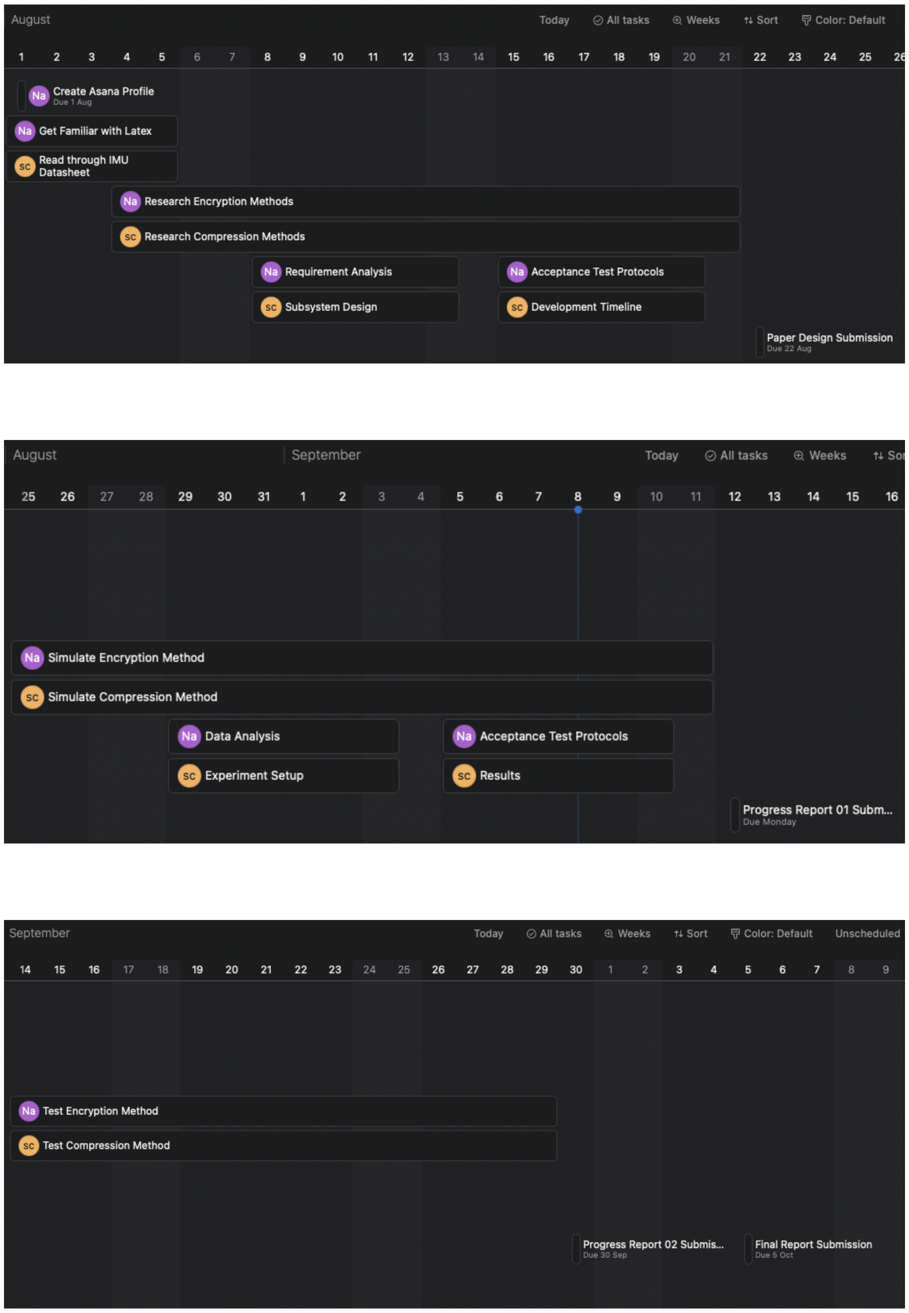


Figure 2.2.1: Asana Timeline

2.3 GitHub

The git repository can be accessed here.

2.4 Demonstration

The following was shown to the tutor: data being sent and received from the STM32F051 micro controller and a working python implementation of the encryption and compression algorithms that form the basis for the simulated portion of the project.

3 Data Analysis

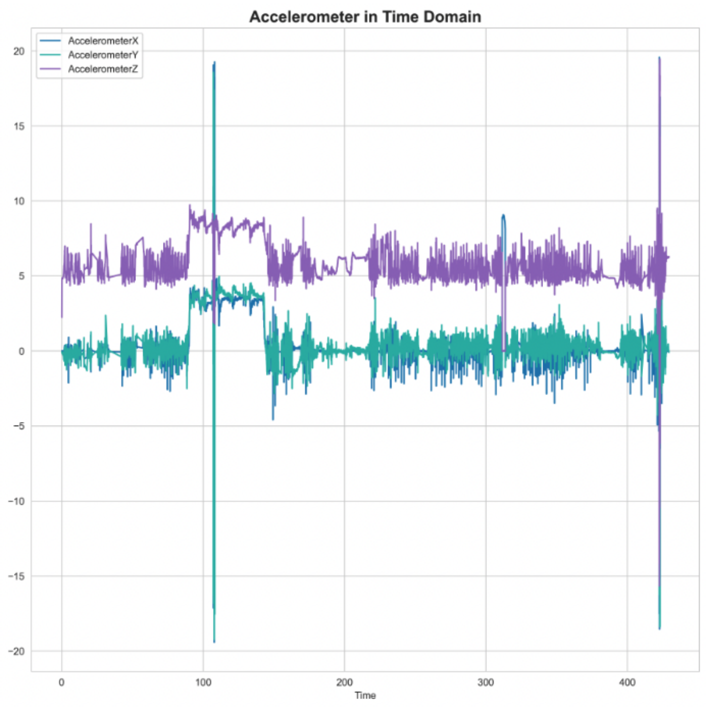
The data chosen for initial analysis is the provided 'Walking Around Example Data' file. This data was chosen as it is large in size which allows for in depth and comprehensive graphical examination of the data.

This data was chosen in default as the students have not yet received the IMU sensors and therefore cannot use the data retrieved directly from the sensor. It is expected in the future that the data come directly from the IMU connected to the STM32F051 microcontroller. This retrieved data will be verified by visual analysis to confirm that the communication link between the computer and the STM32F051 microcontroller yileded correct data. The limitations of using this 'example' data is that the first two ATPs (A1 and A2) as detailed in table 6 below cannot be tested however the remaining ATPs (A3 - A6) can be.

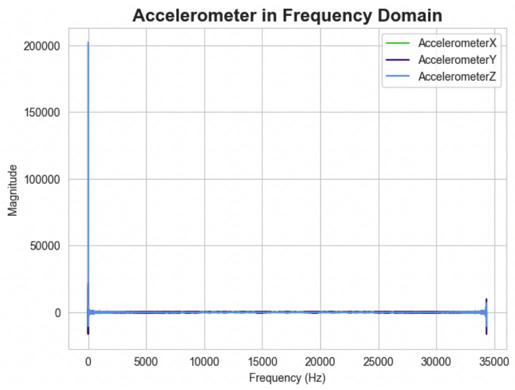
For this analysis, the data that will be focused on is that which comes from the Accelerometer, Magnetometer and Gyroscope sensors as these are the ones on the ICM-20649 IMU sensor that the project is designed to utilize. This data is specified in the x, y and z directions.

3.1 Accelerometer Data Analysis

The Accelerometer data is plotted in the time and frequency domains in figure 3.1.1 below. The frequency domain plot was obtained by taking the Fourier Transform of the data.



(a) Accelerometer Time Domain

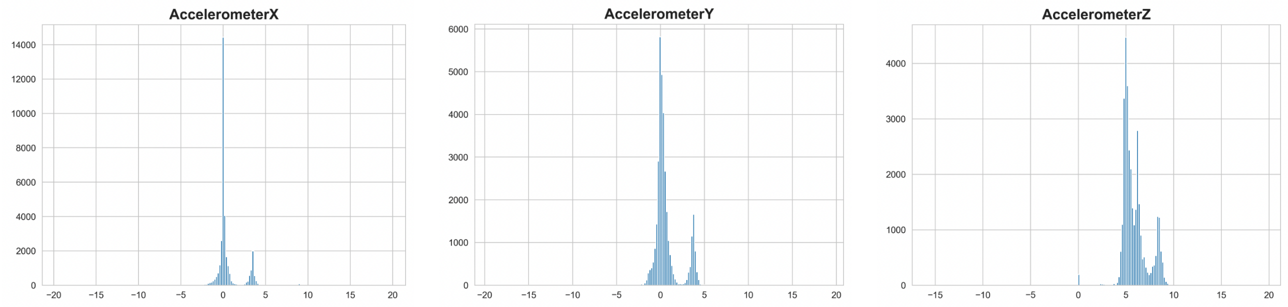


(b) Accelerometer Frequency Domain

Figure 3.1.1: Accelerometer in time and frequency domains

A histogram was plotted for the x, y and z direction time domain Accelerometer data in figure

3.1.2 below.



(a) x Accelerometer histogram (b) y Accelerometer histogram (c) z Accelerometer histogram

Figure 3.1.2: Accelerometer histograms in time domain

The properties of the Accelerometer data are tabulated in table 2 below:

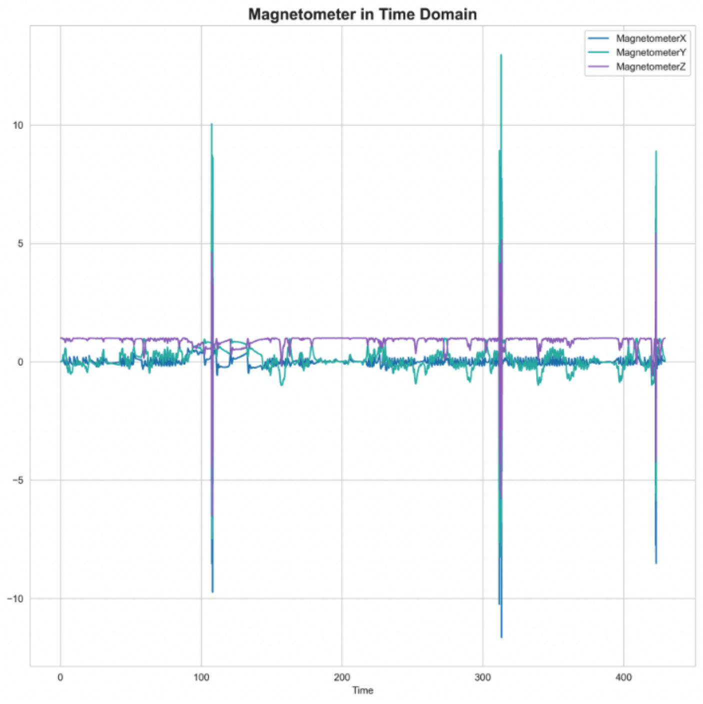
Catgeory	**AccelerometerX**	**AccelerometerY**	**AccelerometerZ**
Count	34328	34328	34328
Mean	0,532169	0,627148	5,886018
Standard Deviation	1,614881	1,580636	1,370541
Minimum	-19,410999	-19,2278	-15,6586
Maximum	19,5781	18,9123	19,457701

Table 2: Properties of Accelerometer Data

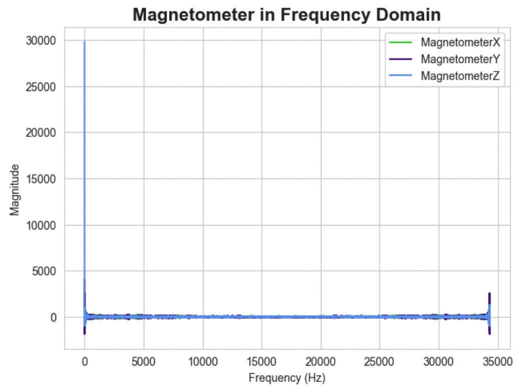
The Accelerometer data has two Gaussian distributions as shown by figure 3.1.2 above. This suggests that there are two major 'events' occurring during data retrieval. The Gaussian distributions suggests that the data is symmetric about the two means (or events) and that the data is adequately 'random'.

3.2 Magnetometer Data Analysis

The Magnetometer data is plotted in the time and frequency domains in figure 3.2.1 below. The frequency domain plot was obtained by taking the Fourier Transform of the data.



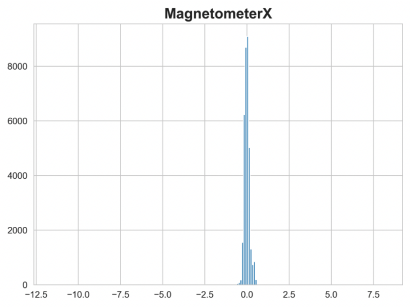
(a) Magnetometer Time Domain



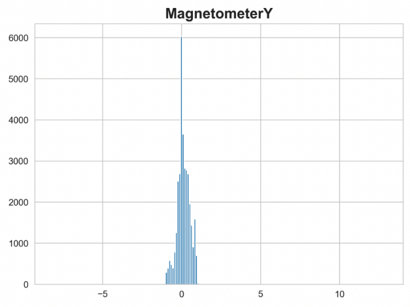
(b) Magnetometer Frequency Domain

Figure 3.2.1: Magnetometer in time and frequency domains

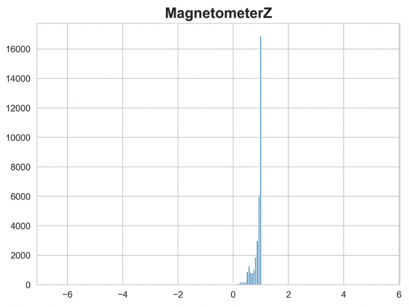
A histogram was plotted for the x, y and z direction time domain Magnetometer data in figure 3.2.2 below.



(a) x Magnetometer histogram



(b) y Magnetometer histogram



(c) z Magnetometer histogram

Figure 3.2.2: Magnetometer histograms in time domain

The properties of the Magnetometer data are tabulated in table 3 below:

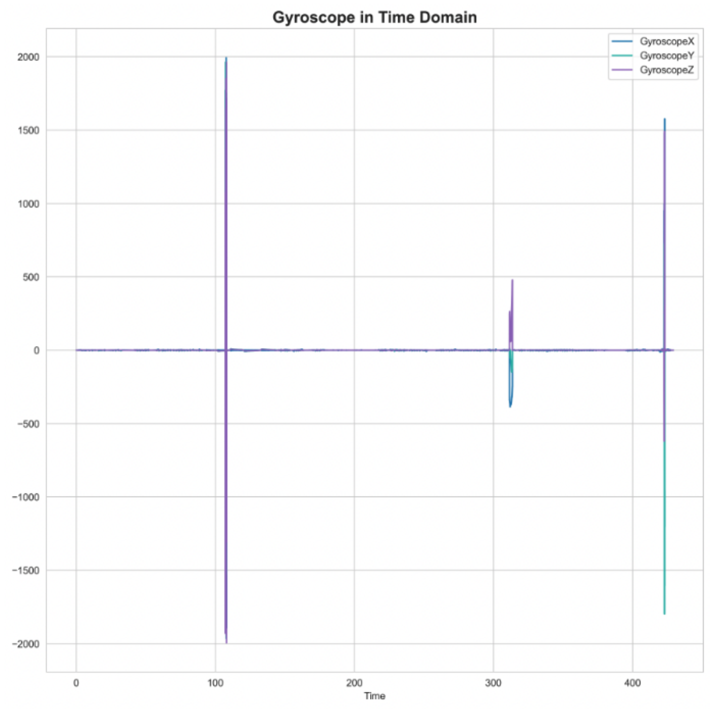
Catgeory	**MagnetometerX**	**MagnetometerY**	**MagnetometerZ**
Count	34328	34328	34328
Mean	-0,006781	0,117376	0,868291
Standard Deviation	0,427628	0,57007	0,317025
Minimum	-11,645	-8,2363	-6,5104
Maximum	8,2309	12,9682	5,4401

Table 3: Properties of Magnetometer Data

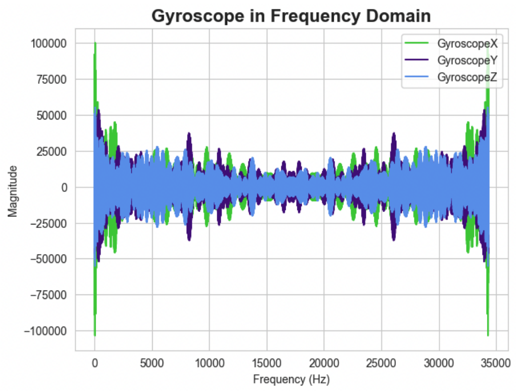
The Magnetometer data follows a Gaussian distribution as shown by figure 3.2.2 above. This suggests that the data is relatively symmetric about the mean and that the data is adequately 'random'. The low standard deviations of data suggest that the data does not deviate greatly from the mean.

3.3 Gyroscope Data Analysis

The Gyroscope data is plotted in the time and frequency domains in figure 3.3.1 below. The frequency domain plot was obtained by taking the Fourier Transform of the data.



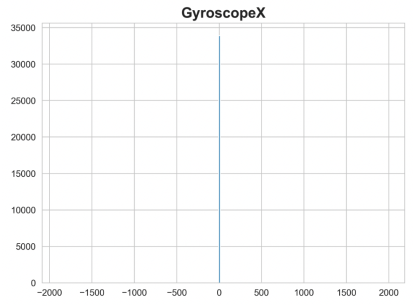
(a) Gyroscope Time Domain



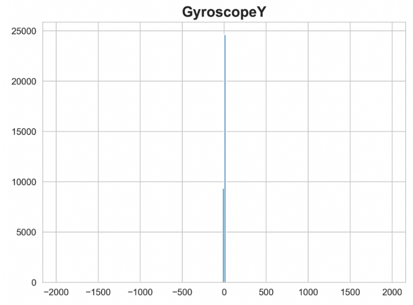
(b) Gyroscope Frequency Domain

Figure 3.3.1: Gyroscope in time and frequency domains

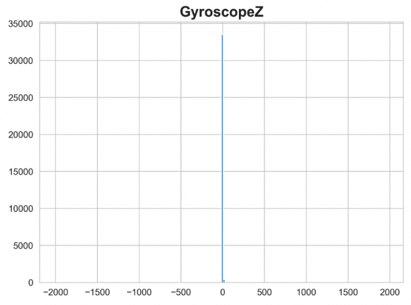
A histogram was plotted for the x, y and z direction time domain Gyroscope data in figure 3.3.2 below.



(a) x Gyroscope histogram



(b) y Gyroscope histogram



(c) z Gyroscope histogram

Figure 3.3.2: Gyroscope histograms in time domain

The properties of the Gyroscope data are tabulated in table 4 below:

Catgeory	**GyroscopeX**	**GyroscopeY**	**GyroscopeZ**
Count	34328	34328	34328
Mean	-0,533702	-0,52153	1,172048
Standard Deviation	79,11129	76,232833	73,234824
Minimum	-1893,536621	-1964,573242	-1993,78064
Maximum	1993,292847	1963,902344	1963,841431

Table 4: Properties of Gyroscope Data

The Gyroscope data follows a Gaussian distribution as shown by figure 3.3.2 above. This suggests that the data is relatively symmetric about the mean and that the data is adequately 'random'.

4 Experiment Setup

Before experiments were performed, the decision of order of encryption and compression of the data needed to be made. Compression relies on patterns and predictability of data and encryption randomises the data resulting in a reduction in patterns and predictability which would reduce the efficiency of compression. Therefore, data should first be compressed and then encrypted. [1]

4.1 Simulations

For the simulations, python programs were used. Although the project requires code in C/C++ for STM32F051 microcontroller compatibility, python allows for easier and more flexible testing to occur. Python is appropriate at this phase of the project as the main focus is simulation based and will not involve the use of the STM32F051 yet. The main algorithms are the compression algorithm as stated in Appendix A.1 [2] below and the encryption algorithm as stated in Appendix A.2 [3] below. A third comparison algorithm is also used as stated in Appendix A.3 [4] below to compare data to ensure minimal loss, this algorithm outputs a percentage difference between two inputted files.

The main pythons script is set up such that the compression algorithm accepts a varying number of lines of data from a csv file containing sample data. This data set of variable size is then compressed and passed to the encryption method to be encrypted. Decryption and subsequent decompression take place before the data is compared to the original data to test for data loss. The time taken for compression, encryption, decompression and decryption was measured using a timing algorithm as stated in Appendix A.4 [5].

4.2 Compression Block

The aim of the compression experiment will be to determine in the compression algorithm meets the specifications and corresponding ATPs. The following method will be followed to test the compression algorithm:

- The compression algorithm A.1 will run on a dataset of known size and a time measurement will be taken for the compression part of the program's runtime.
- This will be repeated on 23 datasets of varying sizes. The varying sizes of data input are determined by the number of lines inputted into the program. This ranges from 10 lines to 33 010 lines.
- A graph will be plotted to determine the time complexity of the compression algorithm. This graph will plot dataset size vs. runtime.
- The compression ratio will be calculated for each of the 23 compression cases and this will also be plotted.
- The compressed file will then be decompressed for the 23 compression cases and a graph will be plotted to determine the time complexity of the decompression algorithm. This graph will plot dataset size vs. runtime.

- The decompressed data will then be compared to the original data using the comparison algorithm A.3 to test for signs of data loss.
- The Fourier transform of the original data and the decompressed data will be verified to have the same first 25% fourier coefficients.

The compression block expects to receive raw data and produce compressed data of reduced size.

4.3 Encryption Block

The aim of the encryption experiment will be to determine in the encryption algorithm meets the specifications and corresponding ATPs. The following method will be followed to test the encryption algorithm:

- The encryption algorithm A.2 will run on a compressed dataset of known size and a time measurement will be taken for the program's runtime.
- This will be repeated on 23 datasets of varying sizes. The varying sizes of data input are determined by the number of lines inputted into the compression program. This ranges from 10 lines all the way up to 34 010 lines.
- A graph will be plotted to determine the time complexity of the encryption algorithm. This graph will plot dataset size vs. runtime.
- The encrypted file will then be decrypted for the 23 compression cases and a graph will be plotted to determine the time complexity of the decryption algorithm. This graph will plot dataset size vs. runtime.
- The encrypted file will then be decrypted and compared to the original data using the comparison algorithm A.3 to test for signs of data loss or tampering.

The encryption block expects to receive compressed data and produce encrypted data of increased randomization.

5 Results

The results of the above mentioned experimental setups for the compression and encryption blocks are shown below.

The program was setup such that the dataset size (i.e. number of lines of csv file) was varied and the compression, encryption, decryption and decompression algroithms were automated. The program automatically records (in csv format): the number of lines in the csv file (i.e. the dataset size), the timing of these four methods, the size of the original data, the size of the compressed data, the compression ratio and a boolean variable stating whether or not the decompressed data matches the original data. Table 5 below shows the results of the experiment.

Lines of Data	**Compress Time (s)**	**Encrypt Time (s)**	**Decrypt Time (s)**	**Decompress Time (s)**	**Original Size (B)**	**Compressed Size (B)**	**Compression Ratio**	**Data Comparison**
10	5.698204040527344e-05	0.008265018463134766	0.008541345596313477	5.412101745605469e-05	4004	1407	2.845771144278607	TRUE
1510	0.0014219284057617188	1.2346031665802002	1.1993489265441895	0.0005030632019042969	636341	199383	3.191550934633344	TRUE
3010	0.0030820369720458984	2.483834981918335	2.5102078914642334	0.0009360313415527344	1260976	407195	3.096737435381083	TRUE
4510	0.008454084396362305	3.7066709995269775	3.6556930541992188	0.001850128173828125	1882197	619049	3.0404652943466512	TRUE
6010	0.011243820190429688	5.106395721435547	5.0341410636901855	0.00374603271484375	2502724	827160	3.0256830601092894	TRUE
7510	0.015943050384521484	6.29496693611145	6.266692876815796	0.004539966583251953	3116791	1037100	3.005294571401022	TRUE
9010	0.012837886810302734	7.821256875991821	7.764508962631226	0.005251884460449219	3732423	1246056	2.9953894528014793	TRUE
10510	0.010243654251098633	8.91141414642334	8.88274598121643	0.0071642398834228516	4348083	1451452	2.995678120943717	TRUE
12010	0.01190495491027832	9.457523107528687	9.51429033279419	0.007005214691162109	4975697	1662961	2.9920707701503524	TRUE
13510	0.014448165893554688	11.50040602684021	11.365385293960571	0.007726192474365234	5605845	1869210	2.999045051117852	TRUE
15010	0.01681685447692871	11.94100308418274	12.004706144332886	0.009037017822265625	6242389	2049173	3.0462967255570907	TRUE
16510	0.017476797103881836	13.139831066131592	13.046177864074707	0.011100292205810547	6879158	2219491	3.099430455000719	TRUE
18010	0.019752979278564453	15.192198991775513	15.078176975250244	0.01158905029296875	7517537	2393168	3.1412491726447955	TRUE
19510	0.024883031845092773	15.257004737854004	15.120185852050781	0.01331782341003418	8145650	2603725	3.1284601868476893	TRUE
21010	0.02616596221923828	17.117594718933105	17.084326028823853	0.013182878494262695	8775760	2807400	3.1259385908669945	TRUE
22510	0.027812957763671875	18.642353057861328	18.54386305809021	0.013921022415161133	9401424	3018094	3.115020274385092	TRUE
24010	0.02629399299621582	21.60689091682434	23.784772872924805	0.014039993286132812	10028875	3228572	3.106288167028643	TRUE
25510	0.02746891975402832	21.42377495765686	21.103504180908203	0.009458780288696289	10646412	3438397	3.096330063107896	TRUE
27010	0.02765512466430664	20.83461093902588	20.83729362487793	0.015753984451293945	11274051	3645814	3.0923275295996997	TRUE
28510	0.0314488410949707	23.77382493019104	23.65444588661194	0.016793251037597656	11897055	3856139	3.0852246249422026	TRUE
30010	0.030972957611083984	24.934611320495605	25.514904260635376	0.011677980422973633	12526243	4061942	3.0838064649864525	TRUE
31510	0.035909175872802734	26.269142150878906	25.221153020858765	0.019836902618408203	13161134	4223094	3.1164672157427704	TRUE
33010	0.041114091873168945	26.270262002944946	25.94036602973938	0.0164029598236084	13786535	4429737	3.1122694191551328	TRUE

Table 5: Experimental Data

5.1 Compression Block

Data pertaining to the compression block of the experiment from table 5 above is plotted below.

Figure 5.1.1 below shows how the runtime of the compression algorithm changes over varying dataset sizes.

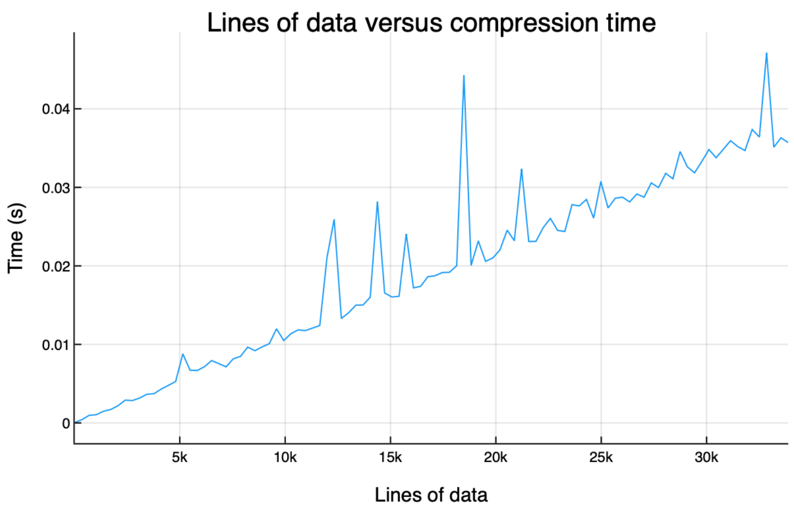


Figure 5.1.1: Graph showing the time required to compress varying sizes of input data.

It can concluded from figure 5.1.1 above that the lz4 compression algorithm implemented in this particular setup has a time complexity of order n [O(n)]. This is to be expected from the lz4 compression library.

Figure 5.1.2 below shows how the runtime of the decompression algorithm changes over varying dataset sizes (number of lines of data).

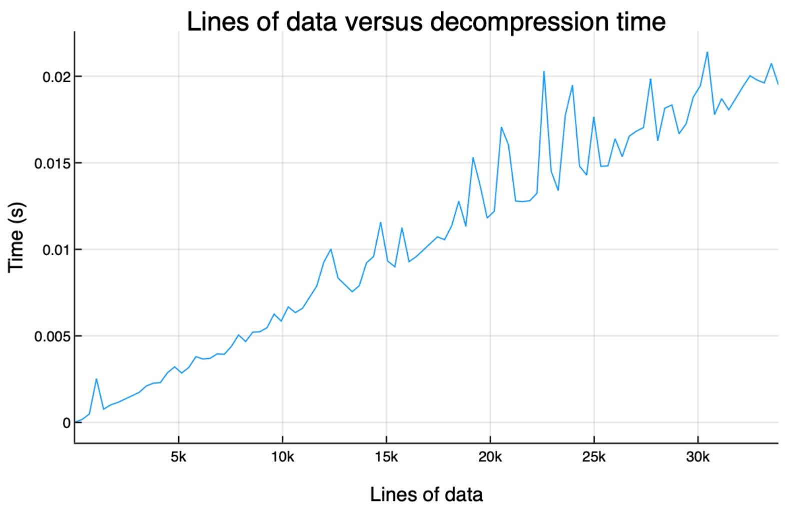


Figure 5.1.2: Graph showing the time required to decompress varying sizes of input data.

As with the results from the compression times, it can concluded from figure 5.1.2 above that the time complexity of the decompression algorithm is also order n [O(n)].

Figure 5.1.2 below shows the change in compression ratio over varying dataset sizes.

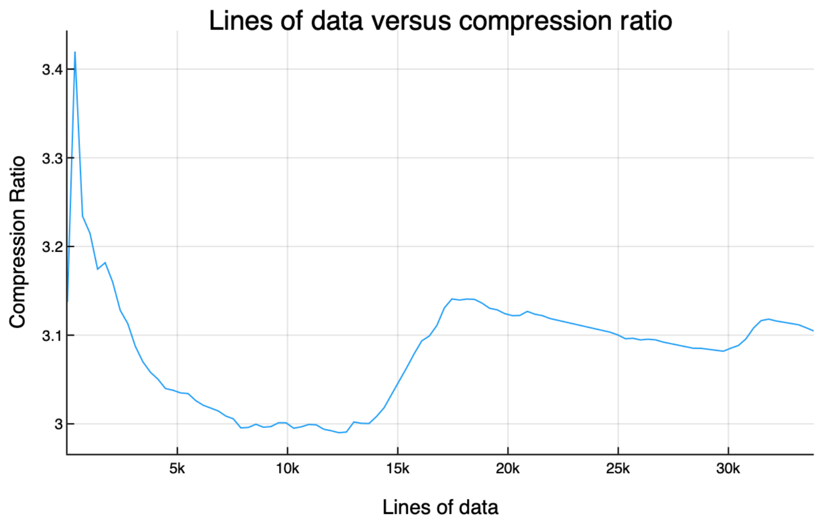
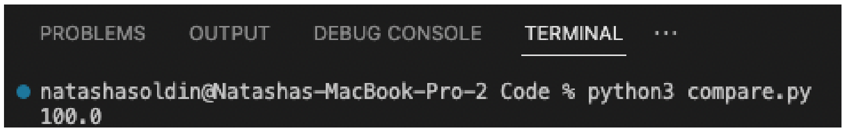


Figure 5.1.3: Graph showing the compression ratio over varying sizes of input data.

It can be seen that the compression ratio for the system starts off fluctuating and as the dataset increases in size, the compression ratio stabilizes to around 3,1.

The comparison algorithm as stated in appendix A.3 outputted a percentage of 100% as shown by the output in figure 5.1.4 below. This suggest that there is no difference between the original data and the decompressed data. This further infers that the Fourier transforms of the two datasets will be the same and therefore the first 25% of the fourier coefficients were retained successfully.



```
natashasoldin@Natashas-MacBook-Pro-2 Code % python3 compare.py
100.0
```

Figure 5.1.4: Output of the comparison algorithm

5.2 Encryption Block

Data pertaining to the encryption block of the experiment from table 5 above is plotted below.

Figure 5.2.1 below shows how the runtime of the encryption algorithm changes over varying dataset sizes.

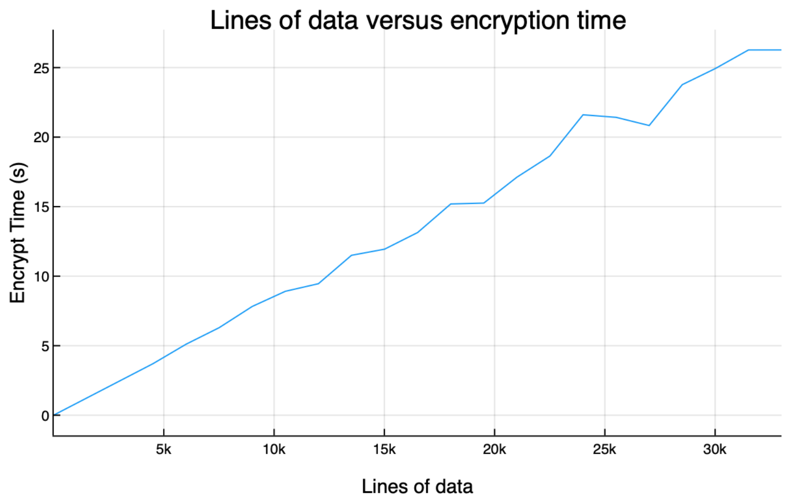


Figure 5.2.1: Graph showing the time required to encrypt varying sizes of input data.

It can concluded from figure 5.2.1 above that the AES encryption algorithm implemented in this particular setup has a time complexity of order n [O(n)].

Figure 5.2.2 below shows how the runtime of the decryption algorithm changes over varying dataset sizes (number of lines of data).

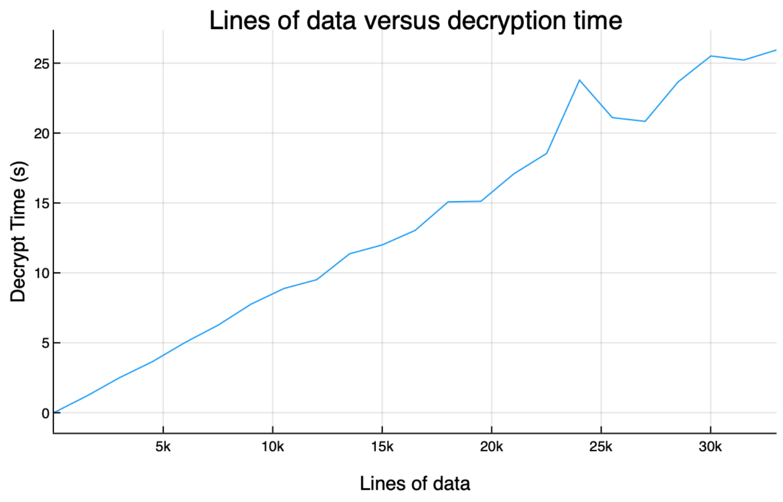


Figure 5.2.2: Graph showing the time required to decryption varying sizes of input data.

As with the results from the encryption times, it can concluded from figure 5.1.2 above that the time complexity of the decryption algorithm is also order n [O(n)].

It is relevant to note that the time taken for the encryption algorithm is larger than initially anticipated. This can be attributed to the fact that the input into the encryption algorithm is in the form of a string of substantial size which renders the algorithms inefficient. This is not expected to be an issue in the future because C/ C++ is faster than python as it is a compiled language while Python is an interpreted language [6]. In addition, the sample sizes of data that will be sent into the encryption algorithm will be considerably smaller which is more efficient to encrypt. Therefore, in the next phase of the project when code is implemented in C/C++, the encryption algorithm is expected be faster.

6 Acceptance Test Protocols

The following table 6 summarises the project's requirements, specifications and acceptance test protocols (ATPs). It also highlights which correspond to which.

Requirements	**Specifications**	**Acceptance Test Protocols**
R1	S1	A1
The project should be designed to utilize the ICM-20649 IMU sensor even though that sensor is not available to use during the project's planning stage.	The design should adhere to the electrical specifications of the ICM-20649 IMU.	Ensure that the data obtained from the sensor hat closely follows the structure of the sample data provided as this comes from the design's intended sensor (ICM-20649 IMU).
R2	S2	A2
The project should be designed to utilize the STM32F051 microcontroller.	Both algorithms needs to be coded in C/C++ in STM32CUBEIDE to be compatible with the STM32F051 microcontroller.	Run the algorithms in a C/C++ IDE as well as on the STM32F051 to test that it works.
R3	S3.1	A3.1
The data obtained from the IMU sensor should be compressed to reduce the cost of transmission because the transmission of data using Iridium is extremely costly.	The compression and decompression of the IMU data will be done using a dictionary compression method.	Check that the file size of the compressed data is considerably less than the file size of the original data.
	S3.2	A3.2
	The compression ratio should be between 40% and 60%	Calculate the compression ratio between the uncompressed and compressed data.

Requirements	**Specifications**	**Acceptance Test Protocols**
R4	S4.1	A4.1
The data obtained from the IMU sensor should be encrypted to increase security of the data.	The encryption and decryption of the IMU data will be done using an AES encryption method.	Compare the encrypted data to the un-encrypted data to ensure minimal similarities which indicates reasonable level of encryption.
	S4.2	A4.2
	The encryption key should be either 128, 192 or 256 bits to ensure adequate security.	Ensure decrypted data is exactly the same as the data before encryption to make sure that no data loss has occured during the encryption phase.
R5	S5	A5
Minimal loss/ damage should apply to the decrypted and decompressed data.	The decrypted and decompressed data should reflect 25% of the Fourier coefficients of the original IMU data.	The fourier transform of the decrypted and decompressed data should be compared to the fourier transform of the original data to test if 25% of the lower fourier co-efficients have been retained.
R6	S6	A6
Limit power consumption by reducing the amount of processing done in the processor and minimizing the computation required.	The C/C++ programs should be of time complexity order n [O(n)] to reduce the amount of processing done and thus the power consumption.	Test the time complexity of the compression and encryption programs by running multiple tests with different dataset sizes and plot the correlation.

Table 6: Requirements, Specifications and ATPs

From the results in section 5 above, the ATPs can be tested to determine whether they have been met or not. This is tabulated in table 7 below:

ATP Number	**Has the ATP been met?**	**Reason**
A1	N/A	The student have not yet received the IMU sensor and therefore cannot obtain data from it. Within this progress report the provided 'Walking Around' IMU data was used as it is large in size and allows for multiple tests to be performed on it.
A2	N/A	This progress report does not yet require code in C/C++ as it is simulation focused. Therefore python was used for easier and more flexible testing to be perfromed.
A3.1	✓	A file size of 13786535B compressed to a file size of 4429737B. This is considerably less and therefore the compression had the desired affect in terms of size reduction.
A3.2	✓	The compression ratio was calculated to be around 3,1. This is adequate as the project required a compression ratio between 40\% and 60\%
A4.1	✓	The similarity between the pre-encrypted data and post-encypted data is minimal. This enough to consider the encryption to be of adequate randomisation.
A4.2	✓	The pre-encrypted data is compared to the post-encrypted data and shows no loss of data
A5	✓	The lower 25% of the fourier co-efficients have been retained when comparing the fourier transform of the original data to the decrypted and decompressed data
A6	✓	The time complexity of the algorithms is O(n)

Table 7: ATP Meeting and Reasoning

The ATPs and corresponding specification were not changed. Although the first two ATPs (A1 and A2) were not met, it is because they were not applicable to this portion of the project as the IMU sensors are not available to the student for data retrieval and verification. Use of the sample data allows for testing of the remaining ATPs (A3 - A6)

References

[1] A. Ahmad, “Encryption and compression of data,” Feb 1960. https://security.stackexchange.com/questions/19969/encryption-and-compression-of-data.

[2] “Lz4 compression library bindings for python.”

[3] “Simple aes-ctr example.” https://cryptobook.nakov.com/symmetric-key-ciphers/aes-encrypt-decrypt-examples.

[4] user9323924user9323924, “How to compare and count the differences between two files line by line?,” Aug 1965. https://stackoverflow.com/questions/49659668/how-to-compare-and-count-the-differences-between-two-files-line-by-line.

[5] “How do i get time of a python program’s execution?.” https://stackoverflow.com/questions/1557571/how-do-i-get-time-of-a-python-programs-execution/12344609#12344609, journal=Stack Overflow, author=PaulMc, year=1957, month=Feb.

[6] “Difference between c and python,” Jun 2022. https://www.interviewbit.com/blog/difference-between-c-and-python/.

A Appendix

A.1 Compression Python Algorithm

The following code was used to implement dictionary compression in python [2]:

```python
def compress(data):
    # Convert imported csv data to bytes format.
    data_as_bytes = str.encode(data)

    # Compress the data.
    Timing.startlog()
    compressed = lz4.frame.compress(data_as_bytes)
    global iTimeToCompress
    iTimeToCompress = Timing.endlog()

    # Get the sizes of the data, compressed and not compressed.
    global iSizeOriginal
    iSizeOriginal = getsizeof(data_as_bytes)

    global iSizeCompressed
    iSizeCompressed = getsizeof(compressed)

    global iCompressionRatio
    iCompressionRatio = iSizeOriginal/iSizeCompressed

    # Write the binary compressed data to a file.
    with open('compressed_data.txt', 'wb') as f:
        f.write(compressed)
```

```python
def decompress(data):
    # Decompress the data.
    Timing.startlog()
    global decompressed
    decompressed = lz4.frame.decompress(data)
    global iTimeToDecompress
    iTimeToDecompress = Timing.endlog()

    #Convert back from byte array.
    decompressed = decompressed.decode()

    # Write decompressed data to output file.
    with open('decompressed_decrypted_data.csv', 'w') as f:
        f.write(decompressed)
```

A.2 Encryption Python Algorithm

The following code was used to implement AES encryption in python [3]:

```
def encrypt(data):
    # Derive a 256-bit AES encryption key from the password
    password = "hello"
    passwordSalt = os.urandom(16)
    global key
    key = pbkdf2.PBKDF2(password, passwordSalt).read(32)

    # Encrypt the plaintext with the given key:
    #   ciphertext = AES-256-CTR-Encrypt(plaintext, key, iv)
    global iv
    iv = secrets.randbits(256)
    aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))

    # Encrypt the data.
    Timing.startlog()
    ciphertext = aes.encrypt(data)
    global iTimeToEncrypt
    iTimeToEncrypt = Timing.endlog()

    # Write the binary compressed encrypted data to a file.
    with open('compressed_encrypted_data.txt', 'wb') as f:
        f.write(ciphertext)
```

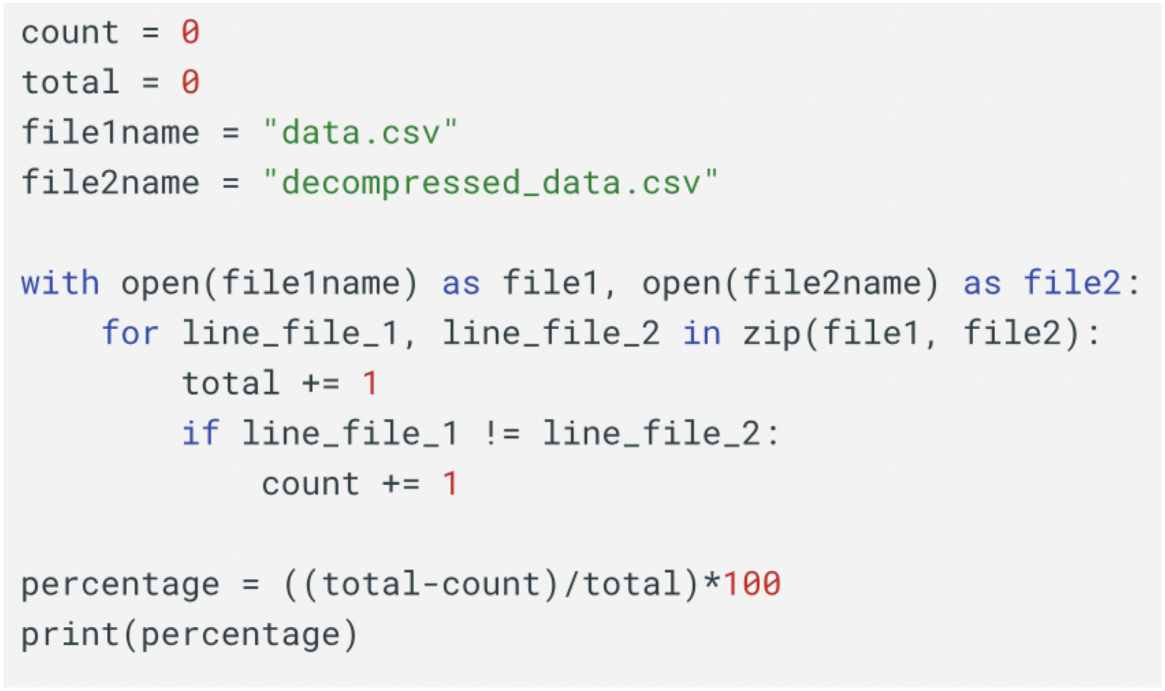
```
def decrypt(data):
    # Decrypt the ciphertext with the given key:
    # plaintext = AES-256-CTR-Decrypt(ciphertext, key, iv)
    aes = pyaes.AESModeOfOperationCTR(key, pyaes.Counter(iv))

    # Decrypt the data.
    Timing.startlog()
    decrypted = aes.decrypt(data)
    global iTimeToDecrypt
    iTimeToDecrypt = Timing.endlog()

    # Write the binary compressed decrypted data to a file.
    with open('compressed_decrypted_data.txt', 'wb') as f:
        f.write(decrypted)
```

A.3 Comparison Python Algorithm

The following code was used to implement data comparison in python [4]



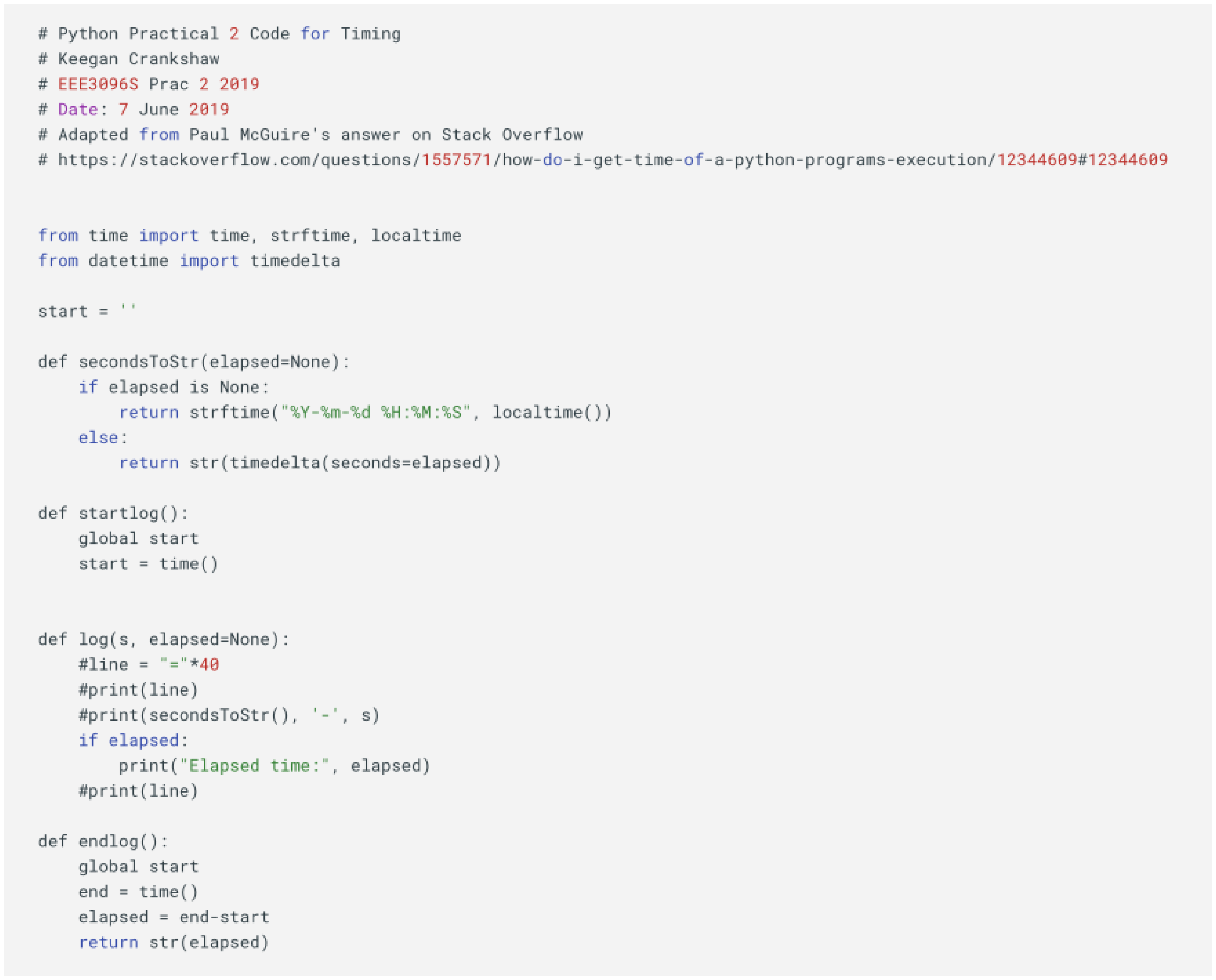
```python
count = 0
total = 0
file1name = "data.csv"
file2name = "decompressed_data.csv"

with open(file1name) as file1, open(file2name) as file2:
    for line_file_1, line_file_2 in zip(file1, file2):
        total += 1
        if line_file_1 != line_file_2:
            count += 1

percentage = ((total-count)/total)*100
print(percentage)
```

A.4 Timing Python Algorithm

The following code was used to implement relative timing measurement in python [5]:



```python
# Python Practical 2 Code for Timing
# Keegan Crankshaw
# EEE3096S Prac 2 2019
# Date: 7 June 2019
# Adapted from Paul McGuire's answer on Stack Overflow
# https://stackoverflow.com/questions/1557571/how-do-i-get-time-of-a-python-programs-execution/12344609#12344609


from time import time, strftime, localtime
from datetime import timedelta

start = ''

def secondsToStr(elapsed=None):
    if elapsed is None:
        return strftime("%Y-%m-%d %H:%M:%S", localtime())
    else:
        return str(timedelta(seconds=elapsed))

def startlog():
    global start
    start = time()


def log(s, elapsed=None):
    #line = "="*40
    #print(line)
    #print(secondsToStr(), '-', s)
    if elapsed:
        print("Elapsed time:", elapsed)
    #print(line)

def endlog():
    global start
    end = time()
    elapsed = end-start
    return str(elapsed)
```