University of Cape Town

EEE3097S

Engineering Design: Electrical and Computer Engineering

Final Report

Report Authors:

Thomas Clegg, CLGTHO001

Kudzayi Samakande, SMKKUD001

Content

Admin Documents

Division of Work

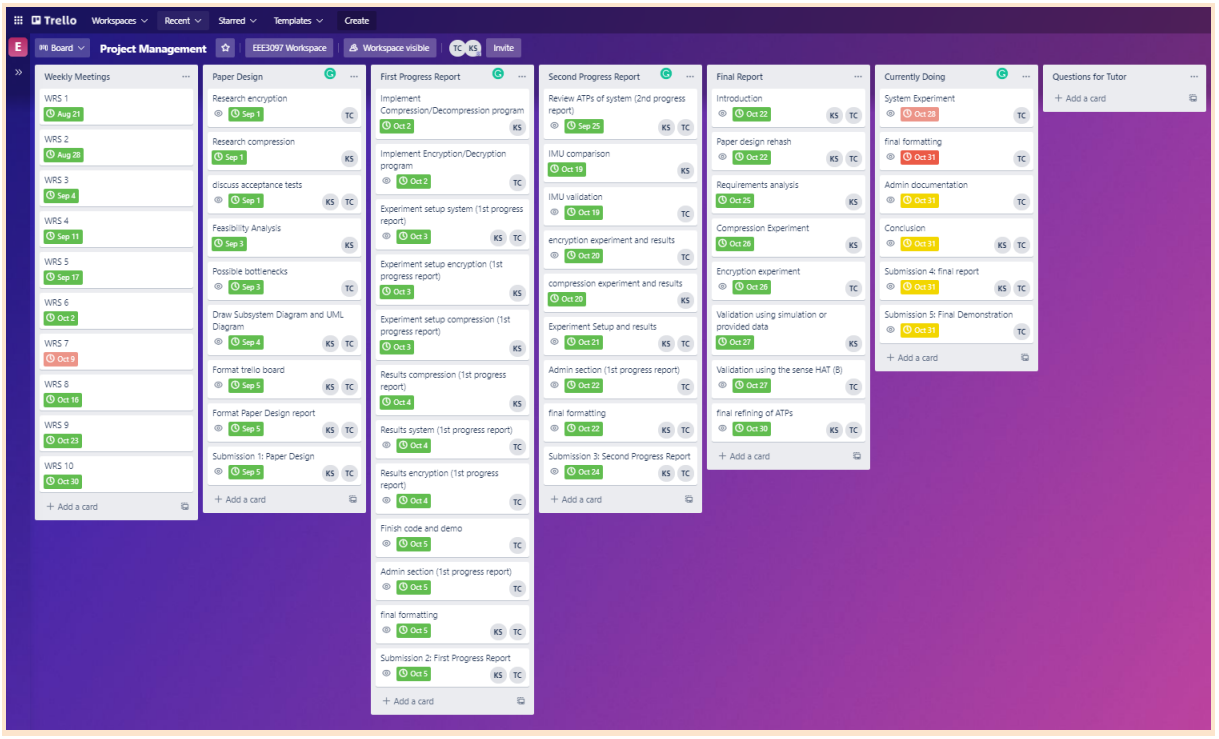
Tast		Member
Project management		Thomas
Introduction		
Requirements analysis		Kudzayi
Paper design	Requirements analysis	Both
	Feasibility analysis	Kudzayi
	Possible bottlenecks	Thomas
	Compression subsystem	Kudzayi
	Encryption subsystem	Thomas
	UML diagrams	Both
	ATPs	Both
Validation using sensor data		Kudzayi
Validation using Sense HAT (B)		Thomas
Experiment	Compression block	Kudzayi
	Encryption block	Thomas
	System	Thomas
ATPs and future plan		Both
Conclusion		Kudzayi
Demonstration video		Thomas

Github Repository

https://github.com/Tom-Clegg/EEE3097S-2021-Group6

Project Management and Timeline

We used Trello to manage our project and timeline. Below shows our Trello project management board which we used to track our tasks, who's assigned to them and when they are due.



Our timeline worked mostly to plan. Some tasks were delayed a day or two due to workload from other courses we take, but all reports were handed in on time. We missed one WRS

Introduction

There is an expedition that endeavours to study winter conditions in the Southern Ocean and Antarctica's sea ice. This is prompted by the unexpectedly extreme changes to Antarctic sea ice during 2016. Scientists intend to explore how storms affect sea ice. The proposed solution is ice-tethered buoys equipped with several sensors e.g an IMU and is installed on ice pancakes designed to communicate with scientists via satellites.

Requirements Analysis

The task at hand is to design an ARM based digital IP using the Raspberry-Pi to encrypt and compress the IMU data.

Problem Identification

Sections below give context of the problem and constraints the designers must address and adhere to.

Power Supply

A reliable power supply is needed for a remotely deployed device to extend its functionality. Advantages of robust power supply are increased processing capabilities and duration. It's impossible to power the remote buoy using outside sources hence the need of a portable power source. Batteries are a suitable solution, preferably rechargeable cells.

Cost

A compromise must be made between high speed devices and low cost devices. For example, modems of transmission that have higher bandwidth are expensive.

Remote Communication

The SHARC buoy uses an iridium satellite network with global coverage. The selected modems for this project offer limited bandwidth.

Specifications and ATPs

System Software RST as a whole are divided into 3 categories namely general constrain, functional requirements and performance requirements

1. General Constrain

Requirements	Specifications	ATP
Perform minimum computations to save power	Use rechargeable batteries and less power hungry algorithms	Connect pi to a battery source instead of a laptop and check how long it can run till it drains the battery

2. Functional requirements

Requirements	Specification	ATP
The system shall digitally sample the movement and orientation of the SHARC	Use ICM-20649 IMU to capture information and an ARM based	Subject the system to known acceleration(gravitational

BUOY. A proper sensor is needed to collect acceleration and orientation status of the BUOY periodically	microcontroller (raspberry Pi)	free fall) and orientation and check if the measured data is the same as the known values of acceleration(9.8ms/s^2)
Retain the lowest 25% Fourier coefficients of the sampled data	Algorithm to perform perform frequency analysis and discard the the upper 95% Fourier coefficients of the sampled data	Compare the sampled data and the output of the system in the frequency domain and check if the output data has the 25% Fourier coefficients
The system must pass compressed and encrypted data to the transmission module	Use compression libraries eg Zlib or GZip Use encryption libraries	Check if the file size to be transmitted is considerably lower than the file size of the sampled data Inspect if the encrypted file if difficult to read/decipher
The transmitted data should be recovered easily	Implement decompression and encryption algorithms	Compare the decrypted and decompressed file with the original file. There must be 0 differences

3. Performance Requirements

Requirements	Specifications	ATP
System`s data processing speed must keep up with the sampling rate to avoid backlogs and data losses	Compression rate plus encryption rate must be greater than the sampling rate	Processing speed in(Kb/s) = <Sampling rate(Kb/s)
There is limited memory on Pi, therefore no data should be stored on the remote device	System throughput must be equal to the transmission bandwidth	Amount of data transmitted per 25 mins must equal the amount of data encrypted and compressed in 25 mins

The respective sub-system Requirements, Specifications and ATPs of compression and encryption block are stipulated in the Paper design Section below.

Testing

Testing is very crucial towards design implementation. We are going to test out algorithms rigorously to make sure they meet all the specifications. A robust IP is needed for this application because it can be costly to rectify mistakes after deployment. Firstly we are going to validate the system by simulating the IMU and then using a different real IMU.

Simulation

For this part we have a choice to use either simulated data or old data from a boat cruise (different from the SHARC Buoy). Simulation based testing is essential in testing our system, we can test extreme conditions which may be difficult to do under real testing. The IMU can be subjected to extreme pressures which if we do with the real IMU we can damage it and it becomes costly.

Paper Design

Bellow is a copy and paste of our Paper Design submission

Requirements Analysis

The design of the subsystem in this project surrounds the collection, compression and encryption of gyroscopic accelerometer data from an IMU to be used in the SHARC buoys that will be installed on pancake ice in the Southern Ocean around Antarctica

The design criteria list three requirements for the project: the ICM-20649 IMU is to be used, at least 25% of the Fourier coefficients of the data along with data being encrypted, reducing the amount of processing done in the Raspberry Pi processor. This project will not work with the actual IMU sensor, therefore simulations and raw data provided will be used. Compression and encryption modules will be made to satisfy the second requirement and throughout the design process choices will be made to limit the processing power needed by tasks carried out on the Raspberry Pi to satisfy the third requirement

Comparison of Compression Algorithms

Compression refers to the process of reducing the data size without losing information. Compressed data is cheaper to store and transmit. In order to recover the original data from compressed data a decompressor is used. This is termed lossless data compression and decompression.

The Deflate algorithm has two stages i.e. LZ77 and Huffman encoding as shown in figure 1.[1]

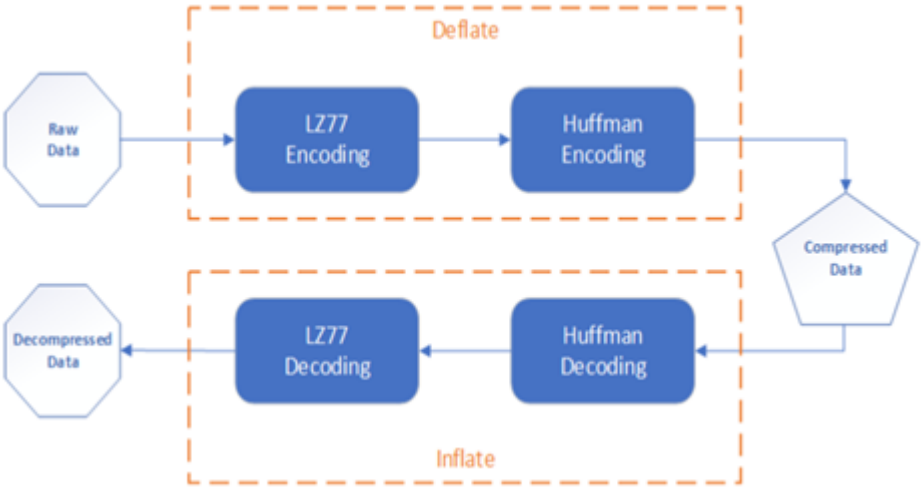


Figure 1. Breakdown of the deflate technique into two algorithms.

LZ77

Replace the repeated occurrences of the data stream with references. To find matches the algorithm keeps track of recently read data (search buffer) and input data (look ahead buffer) in a sliding window.

HUFFMAN ENCODING

The algorithm cleverly transforms fixed binary representation of data symbols (e.g. ASCII) to variable length codes based on frequency of appearance. The rule of thumb is that shorter codes are assigned to more frequently occurring data symbols. Each code is uniquely decoded and this is made possible by making each code a prefix code.

Here is a comparison of different libraries and tools that implement deflate algorithms [3].

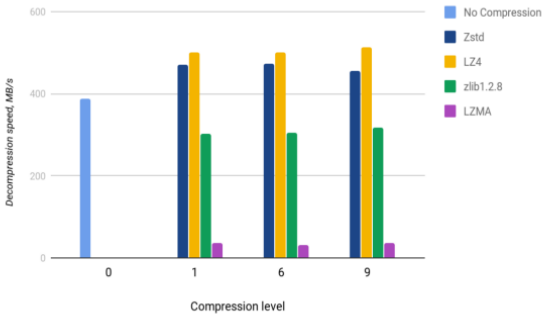


Figure 2:Diagram demonstrating the compression speed

We have chosen to use the GZip library which implements the two techniques discussed above. The GZip compression tool has the best trade-off between speed and compression ratio. GZip has a wide range of support in many applications[3].

Comparison of Encryption Algorithms

Current techniques of encryption are symmetric and asymmetric. Symmetric encryption requires that the sender and receiver have access to the same key, meaning the recipient needs to have access to the key before the message is received. Symmetric encryption algorithms include Data Encryption Standard (DES), Triple Data Encryption Standard (3DES/Triple DES), Advanced Encryption Standard (AES), Blowfish, Twofish, International Data Encryption Algorithm (IDEA), etc [4]. Asymmetric encryption on the other hand uses two keys, one public key and one private key that are mathematically linked to each other. Asymmetric encryption algorithms include Rivest Shamir Adleman (RSA), Elliptical Curve Cryptography (ECC), Diffie-Hellman exchange method, Digital Signature Standard (DSS), Digital Signature Algorithm (DSA), etc [4].

We have chosen to use symmetric encryption because it is faster than asymmetric and requires less computational power [5]. There are downfalls with symmetric encryption such as using a single secret key and potentially having to transmit this key to the receiving decryption entity, however given that we want to optimise speed and lower computational power required, symmetric encryption is best suited for our use case.

Of the available symmetric encryption methods, we have chosen to use AES. AES is quickly becoming the industry standard, it is widely used and trusted, and it is considered invulnerable to all attacks except for brute force (albeit this would take many years to break). Other forms of symmetric encryption are either outdated (such as DES), not as robust (such as Triple DES or IDEA) or not as widely adopted (such as Blowfish and Twofish), therefore making AES the best option to integrate in our subsystem [4, 5].

Feasibility Analysis

1. Technical Feasibility: The current hardware and software supports the implementation of the digital IP using the Raspberry-Pi to encrypt and compress the IMU data. Existing compression and encryption libraries will be used. The team has the technical skills for the project development

2. Schedule Feasibility: Deadlines will be met timely because we will be recycling implemented algorithms for compression and encryption.

3. Economic Feasibility: The project is not economically feasible. Funds are not available to purchase the expensive sensors(IMU).

4. Operational Feasibility: All the requirements have been turned into testable specifications which are attainable. Hence the project has a greater chance of satisfying the user requirements.

Therefore, the project can go ahead regardless of the unavailability of the sensor. As described below this challenge will be overcome by using simulations.

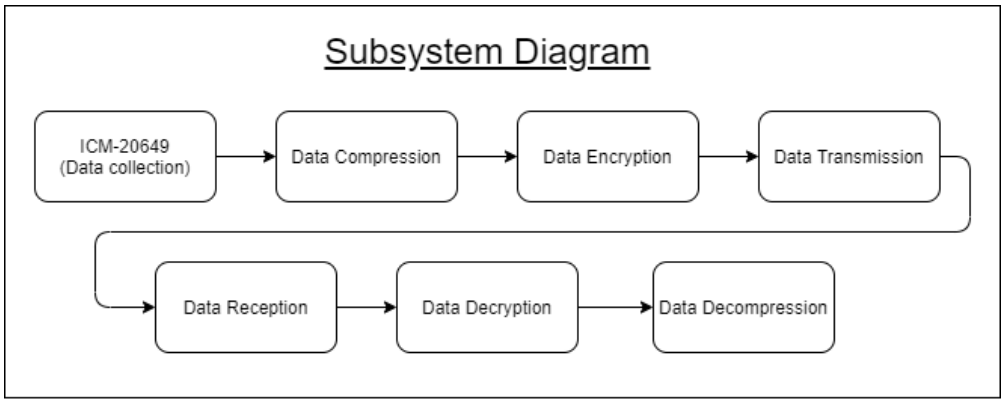
Possible Bottlenecks

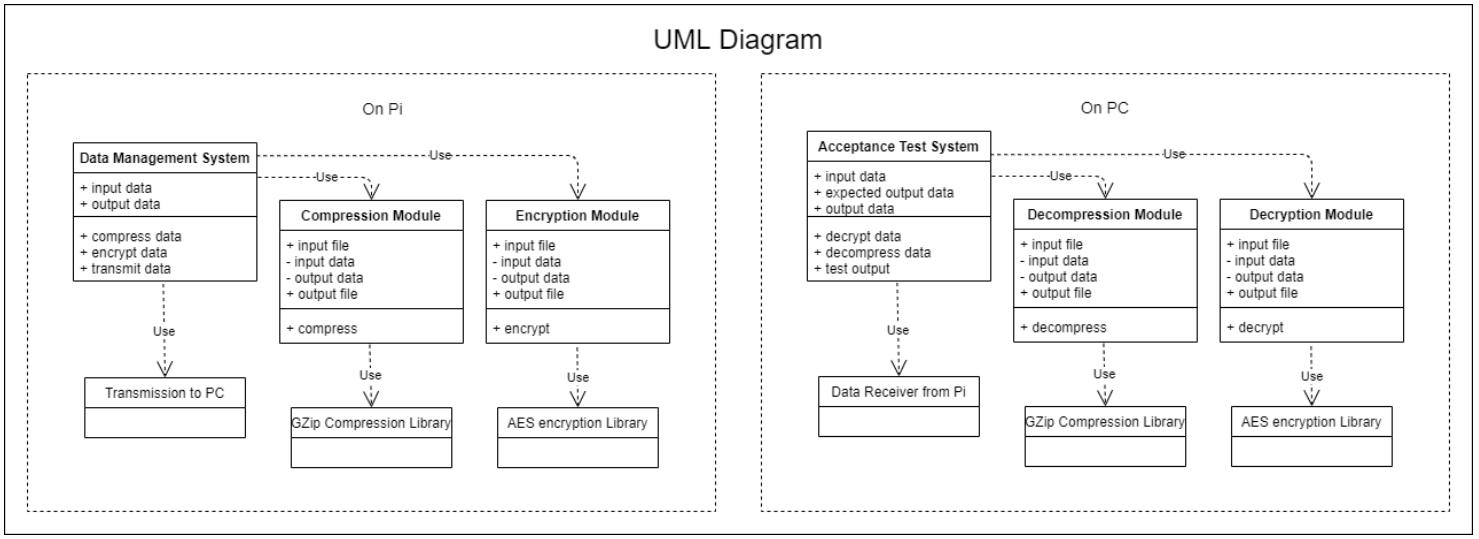
This project is still in the design phase and implementation has not begun. Faults could arise in our use of compression and encryption as well as transmitting the data between the different sub-subsystems. These bottlenecks will be dealt with if these problems occur, this may mean we need to continually update our design and methods used.

Another bottleneck may be in simulating the IMU dataset. The simulations will need to represent the actual output of the sensor as close as possible. If the simulation is not representative of the real world use, this subsystem will not be valid. To combat this, raw data from a similar sensor will be used to test the validity of our design alongside the simulation.

Subsystem Design

Our subsystem is made up of the following sub-subsystems; Data Collection, Data Compression, Data Encryption, Data Transmission, Data Reception, Data Decryption, Data Decompression.





- Data Collection

This unit is the first stage of the subsystem. Since we will not be working with the ICM-20649 sensor, the data collection phase will happen through datasets of simulations and raw data provided. In the simulations, the signal data will need to correspond as close to the sensor data as possible. The sampling rate will be the same as the actual unit and it will have to be realistic.

Additionally, this stage involves filtering of the data to extract at least 25% of the Fourier coefficients of the data.

- Data Compression RSTs

Requirements	**Specifications**	**Acceptance Test Procedure**
Fast compression	Compression speed >>sampling rate	Verify by testing several file sizes
Great compression ratio	compression ratio > 1.5	Compare compressed file size and compare to the original file size.
Recover the original file	Lossless Compression and Decompression	The decompressed file must be identical to the original file.

The output from this subsystem is a gzip compliant file.

- Data Encryption

Once the data has been compressed correctly it will then be encrypted in this module. This will be done using the AES encryption library for the Raspberry Pi.

Requirements	Specifications	Acceptance Test Procedure
Use encryption library	AES encryption library for Raspberry Pi	Decrypt the transmitted data and compare it to raw input
Must be able to receive output from compression as input and must be able to send output after encryption to pc	Use python scripts to handle data being sent between the sub-subsystems	Data files need to be transmitted properly without fault.

- Data Transmission

The final system would require that the dataset is sent via satellite link. We will only be working with the subsystem of capturing, compressing and encrypting the data. Therefore, this Data Transmission unit will send the final compressed and encrypted dataset from the Raspberry Pi to a pc from which our acceptance tests can be run to validate the functionality of the subsystem.

On the other side of transmission the data is decrypted and decompressed to the original format.

Intra-subsystem Interaction

The modules of our subsystem will communicate and pass data files through python scripts. The Encryption module needs to be designed to take in gzip complaint files and the output from the decryption module will be a gzip compliant file which will then be decompressed by the gzip decompressor.

Validation Using Simulated or Old Data

Choice of Data

We have decided to use IMU raw data from another cruise different from SHARC BUOY. However the data sample has readings from other sensors such as temperature, humidity, pressure, magnetometer, etc. The IMU that we will eventually use will out X-, Y-, and Z-axis accelerometer and X-, Y-, and Z-axis angular rate sensors (gyroscopes)

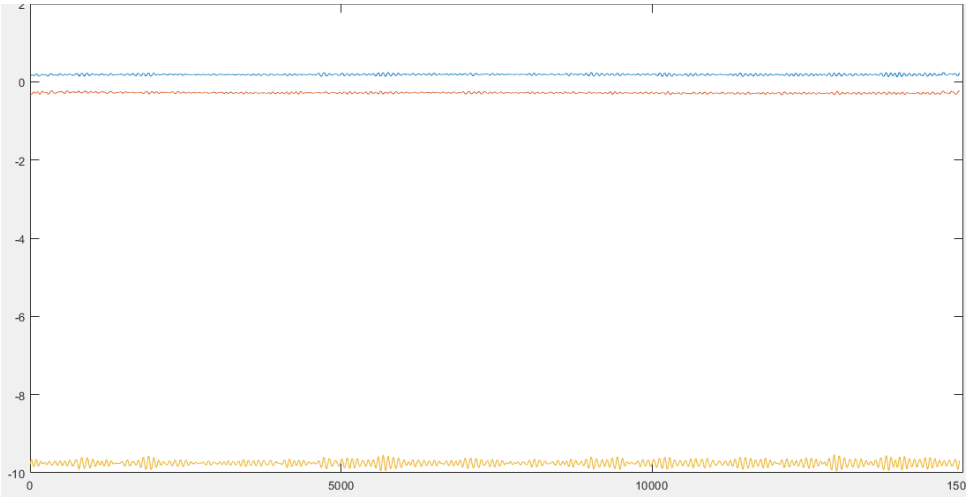
Justification for using Raw data

1. Data set is very large, this allow us to test our system performance under extreme conditions
2. Raw data represent the actual sampling rate of the IMU we are going to use.
3. More importantly the raw data is in the same format as the data that will be produced by the actual IMU.
4. Using old data saves time. It's simple and easy to implement, no worries about writing simulation algorithms that match the IMU sampling rate. This allows us to meet our deadlines.

Analysis of Data

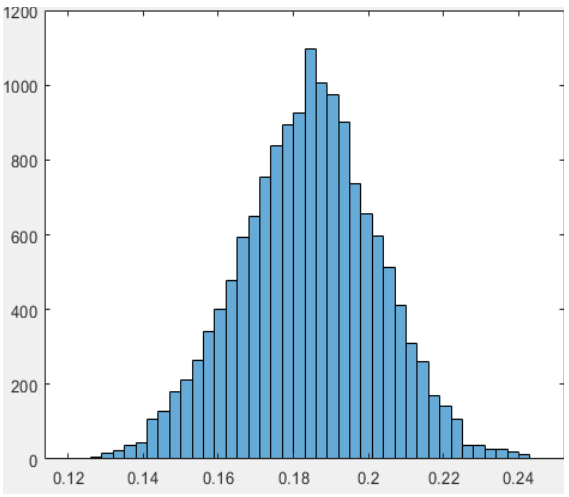
For analysis i`m using the file “2018-09-19-03_57_11_VN100.csv”

AccXYZ time Analysis

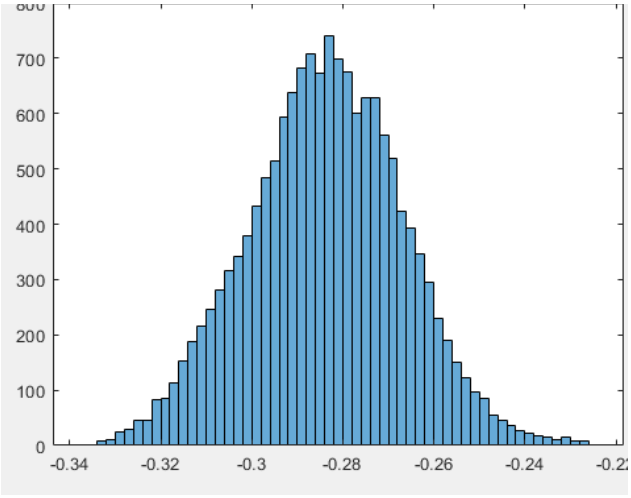


Blue=X, Red=Y, Orange=Z

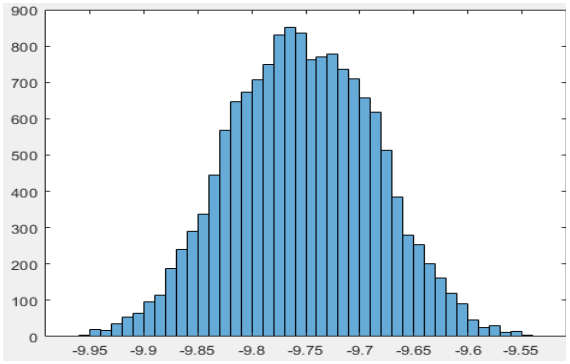
In the X and Y direction acceleration is close to 0



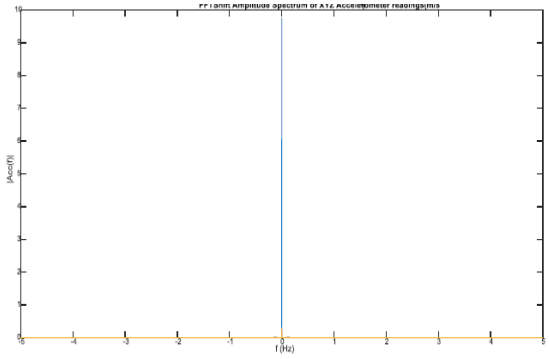
Histogram for AccX



Histogram for AccY



Histogram for AccY

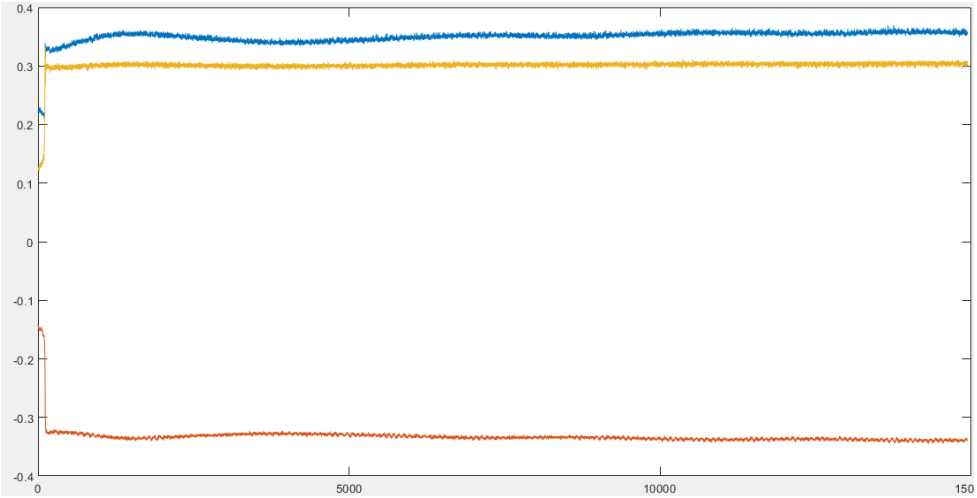


Frequency Spectrum of Acceleration

Variable	Mean	Mode	Std
AccX	0.1841	0.1476	0.0183
AccY	-0.2837	-0.3060	0.0170
AccZ	-9.7528	-9.7897	0.0674

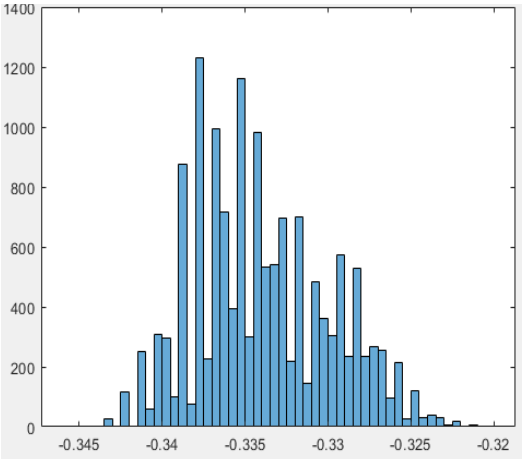
All three histograms show that the data is normally distributed about the mean.
Acceleration in the Z direction is interesting because it is equal to the gravitational acceleration of 9.8m/s^2. The assumption here is that Z represents the vertical motion, and the boat is somewhere in a free fall.
The variables closely follow the mean, small standard deviation
The frequency spectrum shows that the dominant component has a frequency of 0 which means acceleration values are fairly constant within the mean and this is confirmed by small standard deviation.

MagXYZ time Analysis

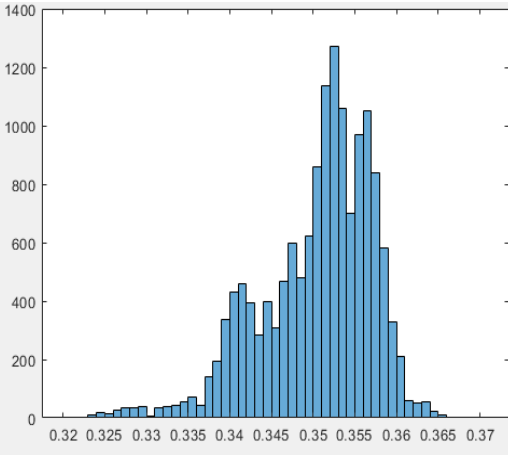


Blue=Y, Red=X, Orange=Z

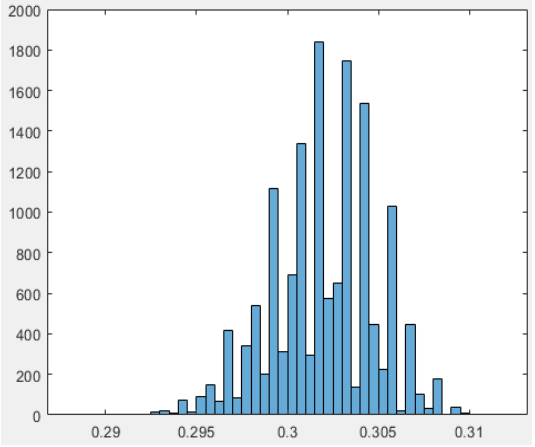
For the Histogram, I removed the first 150 elements because they are outliers, they will hinder proper data analysis.
The outliers might be a result of the boat changing direction during the initial time recording. Constant magnetic field thereafter shows that the boat is moving in a straight line



Histogram for MagX



Histogram for MagY

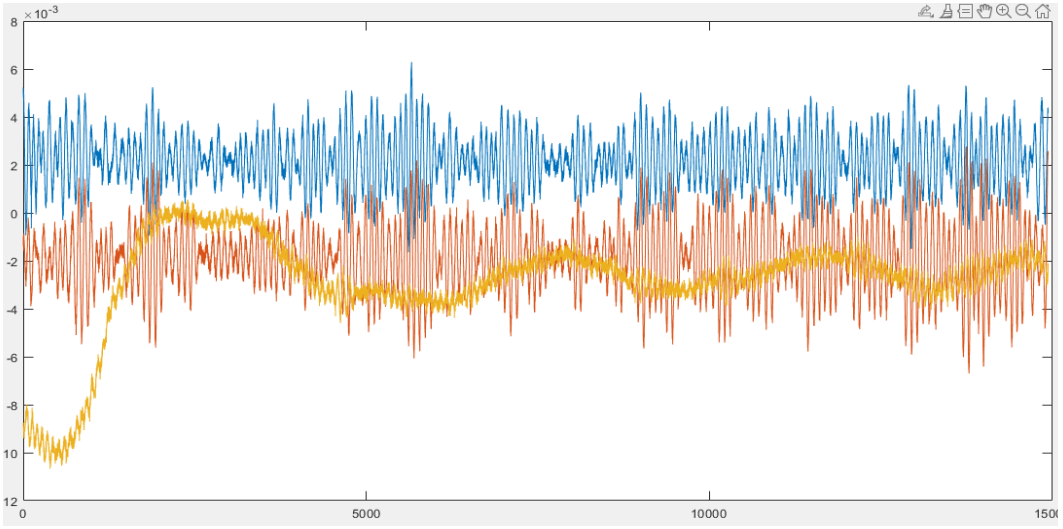


Histogram for MagZ

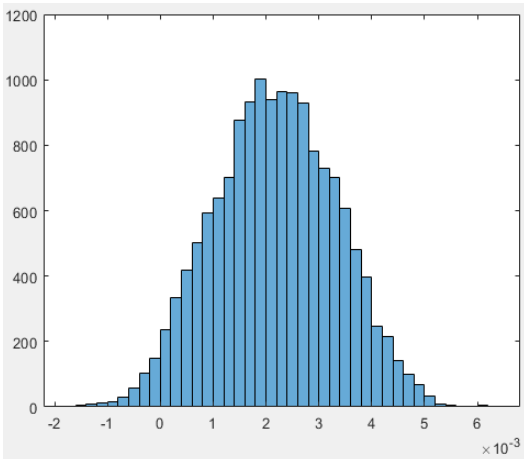
Variable	Mean	Mode	Std
MagX	-0.3339	-0.3390	0.0160
MagY	0.3504	0.3502	0.0067
MagZ	0.3020	0.3044	0.0029

MagX is positively skewed
The MagY is negatively skewed.
MagZ follows gaussian distribution , its occurrence is random, no skweness.

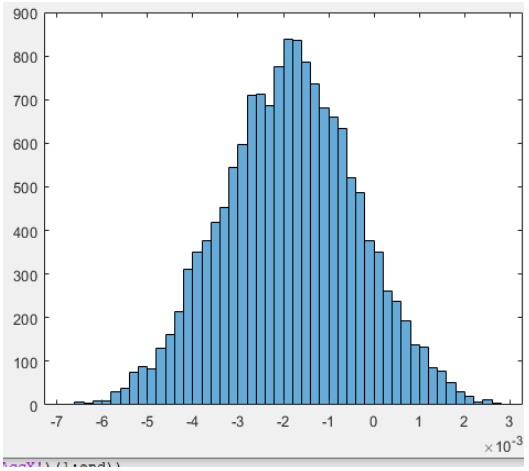
GyroXYZ time Analysis



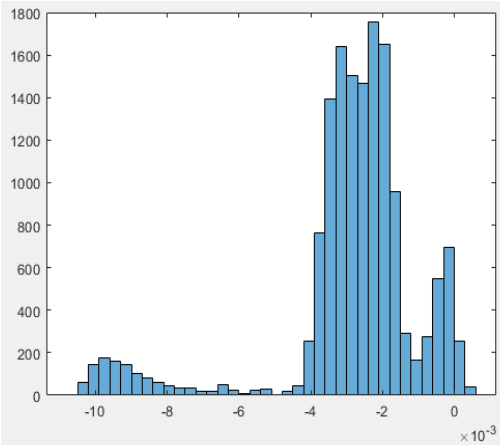
Blue=X, Red=Y, Orange=Z



Histogram for GyroX



Histogram for GyroY



Histogram for GyroZ

Variable	Mean	Mode	Std
GyroX	0.0022	0.0011	0.0012
GyroY	-0.0019	-0.0028	0.0015
GyroZ	-0.0029	-0.0037	0.0021

GyroX and GyroY are normally distributed
The occurrence of GyroZ is not random.

Experiment

Compression Block

The objective of the experiment is to test the compression ratios if they meet the specifications. The experiment seeks to check if the decompressed files are the same as the original file. The experiment will check the best compression level/algorithm to archive speeds required to avoid backlog and keep up with the transmission speeds.

Method

The zlib library we are using has 9 different compression levels
Choose a file large enough to represent the required throughput/bandwidth.
For each level run the compression and record time it takes to complete compression
Calculate speed by diving original file size with time
Calculate compression ratio by dividing original file size with compressed file size
The above steps are done by running the method *compress(fineName)*
Run the decompression algorithm on the compressed file, use the method *decompress(fileName)*
Decompress the compressed file and check for differences with the original file by running *comparison(file1, file2)*
There is no need to test decompression speeds because this process is not done with the IP on the buoy.
Repeat the procedure with different data samples.

Results

Table#:2018-09-19-03_57_11_VN100.csv

Level	File Size (Kb)	Compressed File Size(Kb)	Compression Time(s)	Speed(Kb/s)	Compression Ratio	Differences registered
3	8859	5133	0.54	16402	1.726	0
5	8859	4962	0.9	9843	1.785	0
7	8859	4923	1.3	6814	1.799	0
9	8859	4913	2.0	4430	1.803	0

Table# : 2018-09-19-09_49_31_VN100.csv

Level	File Size (Kb)	Compressed File Size(Kb)	Compression Time(s)	Speed(Kb/s)	Compression Ratio	Differences registered
3	8919	5126	0.46	19319	1.74	0

5	8919	4943	0.72	12423	1.80	0
7	8919	4896	1.09	8176	1.82	0
9	8919	4885	1.76	5072	1.83	0

Table #: combined two file ie 2018-09-19-03_57_11_VN100.csv and 2018-09-19-09_49_31_VN100.csv

Level	File Size (Kb)	Compressed File Size(Kb)	Compression Time(s)	Speed(Kb/s)	Compression Ratio	Differences registered
3	18678	10270	0.77	24306	1.82	0
5	18678	9895	1.26	14874	1.89	0
7	18678	9800	1.8	10370	1.906	0
9	18678	9770	2.79	6695	1.911	0

Input and output of the compression block

The input is a csv file
The output is also csv file

Input and output of the decompression block

The input is a csv file
The output is also csv file

Calculating the Sampling rate

The sample period(T) = 0.1s
Fs = 1/0.1 = 10Hz
Average file size = 8901 Kb contains 14935 samples
Each sample size = 8901/1435 = 0.6Kb
Sampling rate = 0.6Kb/s

Comments

Compression ratio increases with increasing compression level. This in turn increases time to complete compression(speed decreases) because the algorithms are using larger windows to find matches to compress the file even smaller.
With increasing file size the compression speed and ratios increase for each level this is attributed to the fact that larger files have greater chances of matches, there are many

matches therefore the compression ratios are larger.

Compression Blocks ATPs

ATP	Met
Compression Speed >= Sampling Speed(0.6Kb/s)	✓
Compression ratio > 1.5	✓
Decompressed file identical to the original file	✓

Encryption Block

We used AES encryption from the pycrypto library and an algorithm adapted from M.A. Zia [6]. The encryptor class contains all the methods necessary to encrypt and decrypt files. This program recursively runs through each file in the same directory (except for the files containing the python code). When the encrypt_all_files method is called, a timer is run until every file has been encrypted. A second timer is used similarly for the decrypt_all_files method. Our ATPs for the encryption block are encryption/decryption speeds < 10% of data recording speed, and all files must remain exactly the same after encrypting and decrypting them.

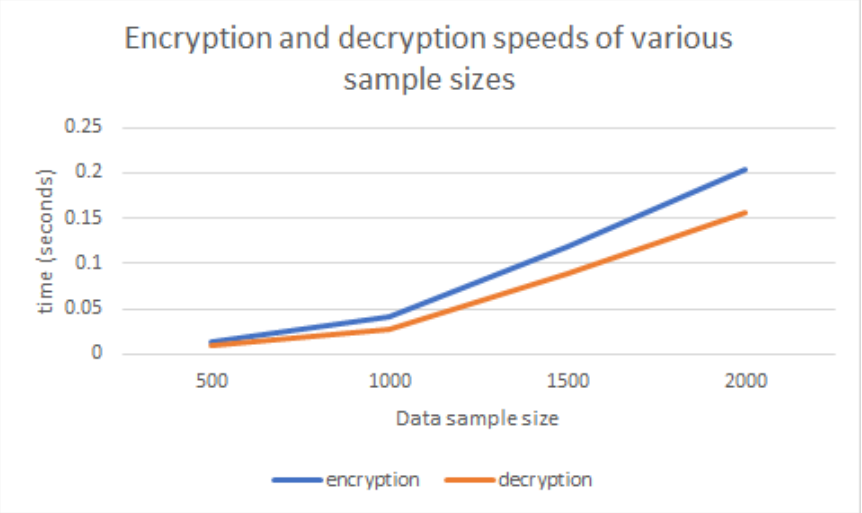
Input and Output of the encryption block

The inputs are all the files of any type in the same directory as the program
The outputs are similarly named files except they will have a .enc extension appended to the end and the original files are deleted.

When decrypting, .enc files are deleted and the files are all saved again with their original names.

Results

Encryption and decryption was done using the sense hat data. This includes the comparison between 500, 1000, 1500 and 2000 samples of data we recorded from the HAT.



Comment on results

Testing the encryption and decryption speeds on the different sample sizes produced expected results. Each increase in sample size produced an increase in both encryption and decryption. At a sample size of 500, the speeds were very similar. As the sample size increases these times diverge too with encrypting taking longer decrypting.

These results are only limited to sample sizes tested. The sense HAT can only record in roughly 0.2-0.3 second intervals so we could not run tests on larger file sizes as it would take too long to produce the larger files. However, we can extrapolate that our system will still function the same. Even at 2000 samples, taking roughly 10 minutes to record, encryption and decryption speeds are still very quick, encryption taking 0.2046 seconds and decryption taking 0.1564 seconds. For larger files, encryption and decryption speeds should remain quick compared to the time taken to record, meaning no bottleneck will occur between batches of data when encrypting.

Comparing file to see if it matches after decryption:

A comparison test was done on each sample size of data to test that after decrypting the file remains the same as before decryption (comparison function explained in Experiment, subheading System)

Test	File remains exactly the same
500 samples	✓
1000 samples	✓
1500 samples	✓
2000 samples	✓

Encryption Block ATPs

ATP	Met
Processing Speed < 10% of data recording speed (per sample size)	✓
Decrypted file identical to the original file	✓

The encryption block has met all the ATPs we designed for.

System

The system is a combination of 2 two subsystems which are compression and encryption blocks. Our system is designed to run linearly through recording, compressing, encrypting, decrypting, decompressing and finally a comparison of the data before and after. The objective of the experiment is to test how fast our system performs as a whole and to check if the end file is identical to the original file.

Method

To do this, three classes are used; one to handle compression and all its methods, one to handle encryption and all its methods, and one to compare a file's data after the first four stages. Below shows the terminal message when running the program:

```
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to
post-decryption/decompression.
7. Press '7' to exit.
```

When '1' is pressed, it will ask the user how many samples they want in the batch. For testing our program we limited the data files to sample sizes of 500, 1000, 1500 and 2000. This is because it takes between 0.2-0.3 seconds per recording. Using really large sample sizes would take a long time to capture, thus we decreased this to speed up testing and validating of the system. If this program were implemented for the buoy, the sample size can be selected to any desired value. The sampling rate can also be decreased by adding "time.sleep(x)" to the code, however the maximum sampling rate is still limited to roughly 0.2 seconds between readings.

It is designed such that each number is pressed sequentially, although this does allow for one to try only compressing and decompressing (by entering 2 then 5) or only encrypting and decrypting (by entering 3 then 4). Comparisons can still be done when only compressing/decompressing or only encrypting/decrypting. After each process, the time it took is printed to the terminal. When the program functions for compression, encryption, decryption and decompression are executed, it will act on every other file in the same directory except for the python files.

File Comparison Check

The comparison class is used to test that the files remain the exact same as the raw data data files after running through the system. To do this, the comparison class opens every file and saves them as text in a variable (pre_text). This is done before the user is able to input to the program. Once the user has compressed, encrypted, decrypted and then decompressed the other files, the comparison test can be initiated by pressing 6. This will open up the files again and save them as text in a new variable (post_text). The pre_text and post_text variables are then compared. If there is any discrepancy the program will tell the user that there was loss and print to the terminal "There was loss after decryption and decompression." If there is no difference found, the program will print to the screen "Files are the same after decryption and decompression."

This comparison method can be adapted in the code to test individually per file or if you only want to test pre/post encryption or pre/post compression. With our methods of compression and encryption used, it is expected that no loss will occur.

System Results

The system was designed to test compression then encryption. Our code cannot compress files that have already been encrypted. Below shows the system results for compression then encryption.

Table#: Results from running compression followed by encryption

File Size(Kb)	Encryption Time(s)	Compression Time(s)	Output File Size(Kb)	Total Time(s)	Processing Speed(KB/s)	Differences Registered
44 (500 samples)	0.0152	0.2021	23	0.2173	202.49	0
90 (1000 samples)	0.0374	0.4679	43	0.5053	178.11	0
134 (1500 samples	0.1097	0.6350	64	0.7447	179.94	0
179 (2000 samples)	0.2103	0.8971	83	1.1074	161.64	0

As expected, times increase for larger file sizes however the processing speed remains somewhat constant. At small file size, i.e. 500 samples, this processing speed is the fastest, however it is highly unlikely that real implementation would require batches of such small sizes. Processing speed of 1500 samples is slightly greater than that of 1000 samples, this probably due to random variations on the pi processor, it is not evident of any result. At 2000 samples, the processing speed slows down. It is unclear if this is the case for increasing the sample size further or if it was due to random variations on the pi processor. It is expected that using much larger files will slow down this processing speed.

Output of tests after decrypting and decompressing

```
File is the same after decryption and decompression
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

This shows that our system maintains all data after processing it. The system we are going to implement will start by compressing data, encrypt it and on the other hand after transmission there will be decryption 1st followed by decompression.

System ATPs

ATP	Met
Processing Speed > sampling rate(0.6kB/s)	✓
System throughput = < modem broadband(amount of data that the communication lines can handle at a time)	✓
Encryption Time =<Compression time (encryption must be faster or equal to the rate at which files are being compressed to avoid bacolod)	✓
CPU time //check procession power	Not checked
Compressed and Encrypted file at most ⅔ (67%) the size of the original file	✓
Decrypted and decompressed file identical to the original file.	✓

Validation Using ICM20948 Sense HAT (B) IMU

Hardware based validation of our software system is very important and needful to address the limitations of simulation. Hardware based validation will expose the system to real world constraints such as power and processor speed constraints. Validating the system using the actual pi instead of a laptop to run algorithms gives accurate speed measurements of compression and encryption. The Sense HAT(B) is going to give an insight of how the data is actually stored.

With hardware validation we can test and configure other things which are assumed during simulation such as the I2C communication between the pi and the HAT.

Feature comparison of the ICM-20649 and Sense HAT (B)

ICM-20649	**Sense HAT (B)**
3-Axis accelerometer, FSR of ±4g, ±8g, ±16g, and ±30g	3-axis accelerometer, FSR of ±2/4/8/16 g
3-Axis gyroscope, FSR of ±500 dps, ±100 dps, ±2000 dps, and ±4000 dps	3-axis gyroscope,FSR of ±250/500/1000/2000 dps
N/A	3-axis magnetometer
N/A	Barometer, Temperature and Humidity
On-Chip 16-bit ADCs	Resolution: 12-bits

Host interface: 7 MHz SPI or 400 kHz I2C	I2C

Justification for using the sense HAT (B)

Sense HAT(B) is a board with a collection of sensors i.e. ICM20948, SHTC3, LPS22HB, TCS34725 and other processing chips. For this project we will make use of ICM20948 which closely resembles the ICM-20649 IMU to be used in the actual buoy.

- Pressure, temperature, humidity and magnetic readings are going to be ignored.
- Will use a force sensitive resistor(FSR) of ±16g for the accelerometer since it is common to both IMUs and to better insure that critical data is not lost at the point of high impact
- Will use a force sensitive resistor(FSR) of ±2000dps for the accelerometer since it is common to both IMUs and to better insure that critical data is not lost at the point of high speed rotation
- The sense HAT ((B) has lower ADC resolution, meaning the digital values obtained are not as accurate as what would have been observed when using ICM-20649. However, for testing purposes it can be ignored.

Validating the IMU

The sense HAT was validated by running operational tests using the 'ICM20948.py' file provided by Waveshare [7]. These tests include the switch-on sequence (to test that all sensors record and used as a baseline for comparing the changes in the other tests), gyroscopic motion, acceleration test, roll pitch, and yaw and magnetic field test. For each test, other recordings are not considered; we only want to compare outputs for the specific test running.

Test Procedures

1. Switch on sequence
    - Method: Run the ICM20948.py file while the sensor remains motionless
    - Expectations: program should run and start displaying readings
2. Gyroscopic Motion
    - Method: Lift the HAT vertically upwards and downwards in varying motions, Slide the HAT across a table in different directions
    - Expectations: Gyroscope readings should change
3. Rotation Test
    - Method: rotate the HAT along its three axes
    - Expectations: Roll, Pitch and Yaw readings should change.
4. Acceleration Test
    - Method: Shake the sensor haphazardly
    - Expectations: Acceleration readings should increase
5. Magnetic field test
    - Method: Bring a strong magnet closer to the HAT
    - Expectations: Increase in magnetic readings

Test Results

Test:		Switch on Sequence	Gyroscope	Rotation	Acceleration	Magnetic
Gyroscope	X	1	-137	~	~	~
	Y	-1	226	~	~	~
	Z	0	-832	~	~	~
Rotation	Roll	-0.60	~	32.42	~	~
	Pitch	0.96	~	17.13	~	~
	Yaw	-33.43	~	-67.61	~	~
Acceleration	X	-248	~	~	-9116	~
	Y	-190	~	~	-32768	~
	Z	16518	~	~	7876	~
Magnetic	X	-122	~	~	~	1175
	Y	-82	~	~	~	-2749
	Z	99	~	~	~	-2398
Test Passed		✓	✓	✓	✓	✓

Experiment setup

System

Introduction

Our system is designed to run linearly through sampling, compressing, encrypting, decrypting, decompressing and finally a comparison of the data before and after. To do this, two files are used. One for the main program and one that is used to record data from the sense HAT. The code to record data, 'ICM20948.py' is adapted from the waveshare wiki [1]. In the main program, three classes are used; one to handle compression and all its methods, one to handle encryption and all its methods, and one to compare a file's data after the first four stages. Below shows the terminal message when running the program:

```
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to
post-decryption/decompression.
7. Press '7' to exit.
```

When ‘1’ is pressed, it will ask the user how many samples they want in the batch. For testing our program we limited the data files to sample sizes of 500, 1000, 1500 and 2000. This is because it takes between 0.2-0.3 seconds per recording. Using really large sample sizes would take a long time to capture, thus we decreased this to speed up testing and validating of the system. If this program were implemented for the buoy, the sample size can be selected to any desired value. The sampling rate can also be decreased by adding “time.sleep(x)” to the code, however the maximum sampling rate is still limited to roughly 0.2 seconds between readings.

It is designed such that each number is pressed sequentially, although this does allow for one to try only compressing and decompressing (by entering 2 then 5) or only encrypting and decrypting (by entering 3 then 4). Comparisons can still be done when only compressing/decompressing or only encrypting/decrypting. After each process, the time it took is printed to the terminal. When the program functions for compression, encryption, decryption and decompression are executed, it will act on every other file in the same directory except for the two python files.

Method

Testing the combined compression and encryption speed.
Run the compression algorithm on the sampled data keeping track of time. Pass the compressed file to the encryption clock while still keeping track of time.
Measure file sizes and calculate speed and throughput

Compression Block Experiment Setup

The objective of the experiment is to test the compression ratios if they meet the specifications. The experiment seeks to check if the decompressed files are the same as the original file. The experiment will check the best compression level/algorithm to archive speeds required to avoid backlog and keep up with the transmission speeds.

Method

The zlib library we are using has 9 different compression levels
The files are generated by the IMU. Set the IMU sampling rate to 10Hz and save the file as a text file.
Pass the file to the compression module
For each level run the compression and record time it takes to complete compression
Calculate speed by diving original file size with time
Calculate compression ratio by dividing original file size with compressed file size
The above steps are done by running the method *compress(fileName)*
Run the decompression algorithm on the compressed file, use the method *decompress(fileName)*
Decompress the compressed file and check for differences with the original file by running *comparison(file1, file2)*
There is no need to test decompression speeds because this process is not done with the IP on the buoy.
Sample another set of time for a longer period and repeat the expriment.

Results

Table#:data_1000.txt

Level	File Size (Kb)	Compressed File Size(Kb)	Compression Time(s)	Speed(Kb/s)	Compression Ratio	Differences registered
3	91	40	0.54	16402	1.726	0
5	91	39	0.9	9843	1.785	0
7	91	38	1.3	6814	1.799	0
9	91	37	2.0	4430	1.803	0

Table# : data_1500.txt

Level	File Size (Kb)	Compressed File Size(Kb)	Compression Time(s)	Speed(Kb/s)	Compression Ratio	Differences registered
3	8919	5126	0.46	19319	1.74	0
5	8919	4943	0.72	12423	1.80	0
7	8919	4896	1.09	8176	1.82	0
9	8919	4885	1.76	5072	1.83	0

Table #: combined two file ie 2018-09-19-03_57_11_VN100.csv and 2018-09-19-09_49_31_VN100.csv

Level	File Size (Kb)	Compressed File Size(Kb)	Compression Time(s)	Speed(Kb/s)	Compression Ratio	Differences registered
3	18678	10270	0.77	24306	1.82	0
5	18678	9895	1.26	14874	1.89	0
7	18678	9800	1.8	10370	1.906	0
9	18678	9770	2.79	6695	1.911	0

Input and output of the compression block
The input is a txt file
The output is also txt file

Input and output of the decompression block
The input is a txt file
The output is also txt file

Sample rate is set at 10Hz

Comments
Compression ratio increases with increasing compression level. This in turn increases time to complete compression(speed decreases) because the algorithms are using larger windows to find matches to compress the file even smaller.
With increasing file size the compression speed and ratios increase for each level this is attributed to the fact that larger files have greater chances of matches, there are many matches therefore the compression ratios are larger.

Compression Blocks ATPs

ATP	Met
Compression Speed >= Sampling Speed(0.6Kb/s)	✓
Compression ratio > 1.5	✓
Decompressed file identical to the original file	✓

Encryption Block

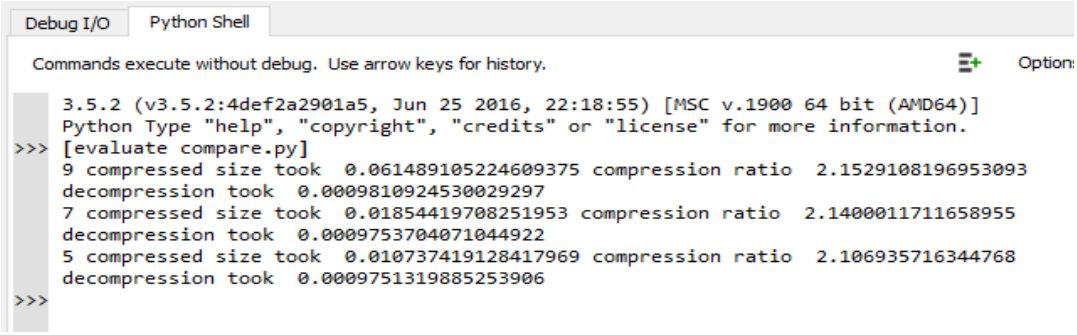
We used AES encryption from the pycrypto library and an algorithm adapted from M.A. Zia [2]. The encryptor class contains all the methods necessary to encrypt and decrypt files. This program recursively runs through each file in the same directory (except for the files containing the python code). When the encrypt_all_files method is called, a timer is run until every file has been encrypted. A second timer is used similarly for the decrypt_all_files method

Input and Output of the encryption block
The inputs are all the files of any type in the same directory as the program
The outputs are similarly named files except they will have a .enc extension appended to the end and the original files are deleted.
When decrypting, .enc files are deleted and the files are all saved again with their original names.

Results

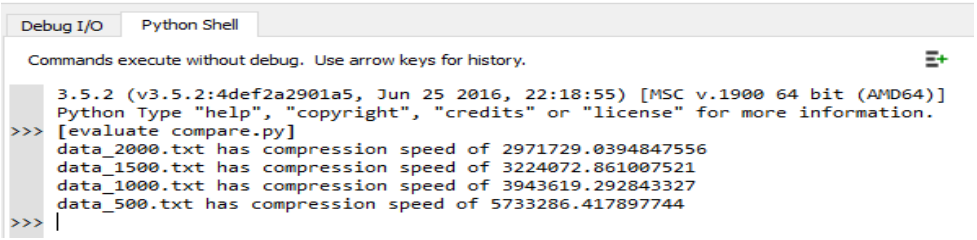
Sampling rate = 0.3 seconds per set of readings

Compression Block



Effects of different compression levels.

As observed with increasing compression levels (from 5 to 9) , the compression time increases however on the upside the compression ratio increases.
A compromise has to be made between compression ratio and speed.
Decompression is very quick and remains the same at 0.0009s for all data sets regardless of size.



Effects of different file sizes on compression speeds.

As observed, a sample of 500 readings has the greatest compression speed of 5733286bits/s compared to 2000 data samples which have a speed of 2971729b/s under the same compression level of 9..

Comparing The Original and The Decompressed

Windows command line is used to check if the original and the decompressed files are the same. The command is fc decompressed.txt "data_2000.txt"

Conclusion on the Compression Block

Tests performed are sufficient to conclude that the compression decompression blocks are accurate. Several data files were used to give a true indication of real data that the software will be processing.

```
C:\Users\Kudzai Samakande\Desktop\src\IMU>fc decompressed.txt "data_2000.txt"
Comparing files decompressed.txt and DATA_2000.TXT
FC: no differences encountered


C:\Users\Kudzai Samakande\Desktop\src\IMU>
```

No differences were encountered

Encryption Block

Encrypting data file of 500 samples:

```
Encryption time: 0.10529756546020508
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Decrypting data file of 500 samples:

```
Decryption time: 0.011373281478881836
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Encrypting data file of 1000 samples:

```
Encryption time: 0.07234740257263184
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Decrypting data file of 1000 samples:

```
Decryption time: 0.018003225326538086
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Encrypting data file of 1500 samples:

```
Encryption time: 0.10439467430114746
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Decrypting data file of 1500 samples:

```
Decryption time: 0.08768796920776367
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Encrypting data file of 2000 samples:

```
Encryption time: 0.228165864944458
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Decrypting data file of 2000 samples:

```
Decryption time: 0.048346757888793945
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Encryption and decryption times were very fast compared to previous submission when testing with data files provided. This is because the recorded data is much smaller in size. Times increase with increasing file size except for 500 samples where the encryption and decryption took longer than the next test of 1000 samples.

System

Since the system runs sequentially from user input, the total time is calculated from the summation of times of each process. For the program to compress, encrypt, decrypt and decompress all four of the datasets, it takes an average of between 1.5-2.0 seconds. Below shows one example of running the program sequentially.

```
Compression time: 1.4341986179351807
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

```
Encryption time: 0.3458676338195801
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

```
Decryption time: 0.045719623565673833
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

```
Decompression time: 0.12929511070251465
1. Press '1' to record samples.
2. Press '2' to compress all files in the directory.
3. Press '3' to encrypt all files in the directory.
4. Press '4' to decrypt all files in the directory.
5. Press '5' to decompress all files in the directory.
6. Press '6' to compare pre-compression/encryption to post-decryption/decompression.
7. Press '7' to exit.
```

Our system can now record data (of any given input sample size) and save outputs to txt files, compress all the files, encrypt all the files, decrypt all the files and decompress all the files. From the previous submission, comparisons showed that no data is lost. This is expected from the methods of compression and encryption we used. Our system preserves all data.

ATPs

Previous submission's ATPs:

Compression speed < 10 seconds for entire dataset	ATP met
Compression ratio >= 1.5	ATP met
Decompression speed < 10 seconds for entire dataset	ATP met
Encryption speed < 10 seconds for entire dataset	ATP met
One block should not operate more than 5 times slower than the other so that bottleneck does not occur	ATP met

Our ATPs needed to be redesigned from the last submission. Below depicts a table of our ATP and whether they were met or not. This table now includes the requirements and specifications to go along with the ATPs.

Requirements	Specifications	ATP	Decision
Power consumption should be minimized	Ip is power by 11mA rechargeable batteries		Not tested
Fast compressions and decompression	Compression speed >> sampling rate	Compression speed < 10 seconds for entire dataset Decompression speed < 10 seconds for entire dataset	ATP met
Reduced file sizes for transmission	Compressed file<iridium bandwidth	Compression ratio >= 1.5	ATP met
Fast encryption and decryption	Encryption speed =< Compression Speed (making use of AES encryption)	Encryption speed < 10 seconds for entire dataset	ATP met
Secure files for transmission	The system should be able to compress then encrypt the files with no bottleneck	One block should not operate more than 5 times slower than the other so that bottleneck does not occur	ATP met
Minimal data loss	Data should be preserved after decryption and decompression	No data loss	ATP met

All our ATPs were met except for power consumption. This was not tested for our system nor did we define what our power consumption should be. This was not the main concern however, we wanted our compression and encryption blocks to work seamlessly, results show this was satisfied.

ATP Consolidation

System Software ATP

Requirements	Specifications	ATP	Decision
System`s data processing speed must keep up with the sampling rate to avoid backlogs and data losses	Compression rate plus encryption rate must be greater than the sampling rate	Processing speed in(Kb/s) = <Sampling rate(Kb/s)	Test passed
There is limited memory on Pi, therefore no data should be stored on the remote device	System throughput must be equal to the transmission bandwidth	Amount of data transmitted per 25 mins must equal the amount of data encrypted and compressed in 25 mins	Test Passed
Perform minimum computations to save power		Connect pi to a battery source instead of a laptop and check how long it can run till it drains the battery	Not tested
The system shall digitally sample the movement and orientation of the SHARC BUOY. A proper sensor is needed to collect acceleration and orientation status of the BUOY periodically	Use ICM-20649 IMU to capture information and an ARM based microcontroller (raspberry Pi)	Subject the system to known acceleration(gravitational free fall) and orientation and check if the measured data is the same as the known values of acceleration(9.8m s/s^2)	Test Passed
Retain the lowest 25% Fourier coefficients of the sampled data	Algorithm to perform perform frequency analysis and discard the the upper 95% Fourier coefficients of the sampled data	Compare the sampled data and the output of the system in the frequency domain and check if the output data has the 25% Fourier coefficients	Test Failed
The system must pass compressed	Use compression libraries eg Zlib or	Check if the file size to be	Test passed

and encrypted data to the transmission module	GZip Use encryption libraries	transmitted is considerably lower than the file size of the sampled data Inspect if the encrypted file if difficult to read/decipher	
The transmitted data should be recovered easily	Implement decompression and encryption algorithms	Compare the decrypted and decompressed file with the original file. There must be 0 differences	Tests Passed

Below is an extract from our Sub Systems ATPs from paper design

Data compression

Requirements	**Specifications**	**Acceptance Test Procedure**	**Decision**
Fast compression and decompression	Compression speed greater than 50MB/s	The decompressed data must be identical to the original raw data.	Test Passed
Great compression ratio	compression ratio > 1.5		

Data encryption

Requirements	**Specifications**	**Acceptance Test Procedure**	**Decision**
Use encryption library	AES encryption library for Raspberry Pi	Decrypt the transmitted data and compare it to raw input	Test Passed
Must be able to receive output from compression as input and must be able to send output after encryption to pc	Use python scripts to handle data being sent between the sub-subsystems	Data files need to be transmitted properly without fault.	Test Passed

These ATPs needed to be redesigned because they were vague and very limited. They were also not organised properly in a coherent ATP section

Below is an extract from our first progress report ATPS

ATP	Met
Compression speed < 10 seconds for entire dataset	ATP met
Compression ratio >= 1.5	ATP met
Decompression speed < 10 seconds for entire dataset	ATP met
Encryption speed < 10 seconds for entire dataset	ATP met
One block should not operate more than 5 times slower than the other so that bottleneck does not occur	ATP met

Our ATPs were more defined but still not exact for our system. They still lacked detail and further explanation.

Below is an extract from our second progress report ATPs

Requirements	Specifications	ATP	Decision
Power consumption should be minimized	Ip is power by 11mA rechargeable batteries		Not tested
Fast compressions and decompression	Compression speed >> sampling rate	Compression speed < 10 seconds for entire dataset Decompression speed < 10 seconds for entire dataset	ATP met
Reduced file sizes for transmission	Compressed file<iridium bandwidth	Compression ratio >= 1.5	ATP met
Fast encryption and decryption	Encryption speed =< Compression Speed (making use of AES encryption)	Encryption speed < 10 seconds for entire dataset	ATP met
Secure files for transmission	The system should be able to compress then encrypt the files with no bottleneck	One block should not operate more than 5 times slower than the other so that bottleneck does not occur	ATP met
Minimal data loss	Data should be preserved after decryption and decompression	No data loss	ATP met

These ATPs are much more defined, with included requirements and specifications for each ATP. As we progressed through our design, we needed to refine our acceptance tests. This helped us focus on defining our system more, testing for specific requirements and presenting the results in an easy to read format.

Future Plan

We wanted to design a modular system that is easy to read, adapt and implement. This was our purpose in designing our program to run sequentially from user input. The user can select the sample size of the data to record, the user can select to compress and/or encrypt, and the user can easily modify the code to run their own test. Therefore, in future use of this program the user will be able to incorporate our work into the system it's being used for with little difficulty. The program can be modified to run all the operations one after the other more automatically and the number of samples in one batch can be set to the desired size for use on the SHARC buoy.

Conclusion

The project went on according to the timeline. All Deadlines were met and each individual delivered their subsystems as promised. All the anticipated challenges/bottlenecks were overcomed. The systems integrated well and performed up to speed.

The software system as a whole met all its objectives which are to compress, encrypt, decrypt and decompress files.

On compression great compression ratios were achieved, and the encrypted files are very secure for transmission.

Tests were performed and the end file was identical to the original file.

In conclusion the designed sub system is ready for the next stage, integration with other subsystems of the SHARC BUOY and then deployment.

References

[1] (2019). Understanding Zlib [Online]. Available: https://www.euccas.me/zlib/

[2] Raul Fraile: How GZIP compression works | JSConf EU 2014. Available: https://www.youtube.com/watch?v=wLx5OGxOYUc&ab_channel=JSConf

[3] O Shadura and B Bockelman 2020 J. Phys.: Conf. Ser. 1525 012049. Available: https://iopscience.iop.org/article/10.1088/1742-6596/1525/1/012049/pdf.

[4] B. Daniel, “Symmetric vs. Asymmetric Encryption: What's the Difference?,” Trenton Systems Blog, 4 May 2021. [Online]. Available: https://www.trentonsystems.com/blog/symmetric-vs-asymmetric-encryption. [Accessed 4 September 2021].

[5] J. Thakkar, “Types of Encryption: 5 Encryption Algorithms & How to Choose the Right One,” The SSL Store, 22 May 2020. [Online]. Available: https://www.thesslstore.com/blog/types-of-encryption-encryption-algorithms-how-to-choose-the-right-one/. [Accessed 4 September 2021].

[6] M. A. Zia, "Python File Encryptor," Javapocalypse, 20 January 2018. [Online]. Available: https://github.com/the-javapocalypse/Python-File-Encryptor/blob/master/script.py. [Accessed 30 September 2021].

[7] Waveshare, "Sense HAT (B)," Waveshare, 21 July 2021. [Online]. Available: https://www.waveshare.com/wiki/Sense_HAT_(B). [Accessed 20 October 2021].