EEE3097S
Final Report of ARM based digital IP using a Raspberry-Pi to compress and encrypt IMU data

Group 1: DSLGRE001 & ADMALI004

1. Admin Documents 3
1.1. Member Contributions 3
1.2. Project management tool 4
1.3. Link To GitHub Page 4
1.4. Project Timeline 5
2. Requirement Analysis 6
2.1. Interpretation of the requirements 6
2.2. Comparison of some available compression and encryption algorithms 6
2.2.1. Encryption 6
2.2.2. Compression 7
2.3. Feasibility analysis 9
2.4. Bottlenecks 9
2.5. Subsystem and Sub-subsystems Requirements 10
2.6. Subsystem and Sub-subsystems Specifications 13
2.7. Acceptance Test Procedure based on Specifications 19
2.7.1. Figures of merits, based on which you would validate your final design 19
2.7.2. Experiment design to test these figures of merit 19
2.7.3. Acceptable performance definition 20
3. Paper Design 20
3.1. Inter-Subsystem and Inter-Sub-subsystems Interactions 20
3.2. UML Diagram 21
4. Validation using Simulated or Old Data 21
4.1. The Need to do Simulation Based Validation 21
4.2. Experiment Setup 29
4.3. Results 38
5. Validation using a different-IMU 40
5.1. IMU Module 40
5. 2. Steps you plan to take to make sure that your testing with this IMU can be extrapolated easily to the IMU to be used in the actual buoy 42
5.3. List of validation tests 45
5. 4. Test cases to validate IMU 46
5. 5. Experiment Setup 49
5.6. Results 58

6. Consolidation of ATPs and Future Plan **73**
6.1. ATPs **73**
6.2. Future Work **74**
Conclusion **74**
References **75**

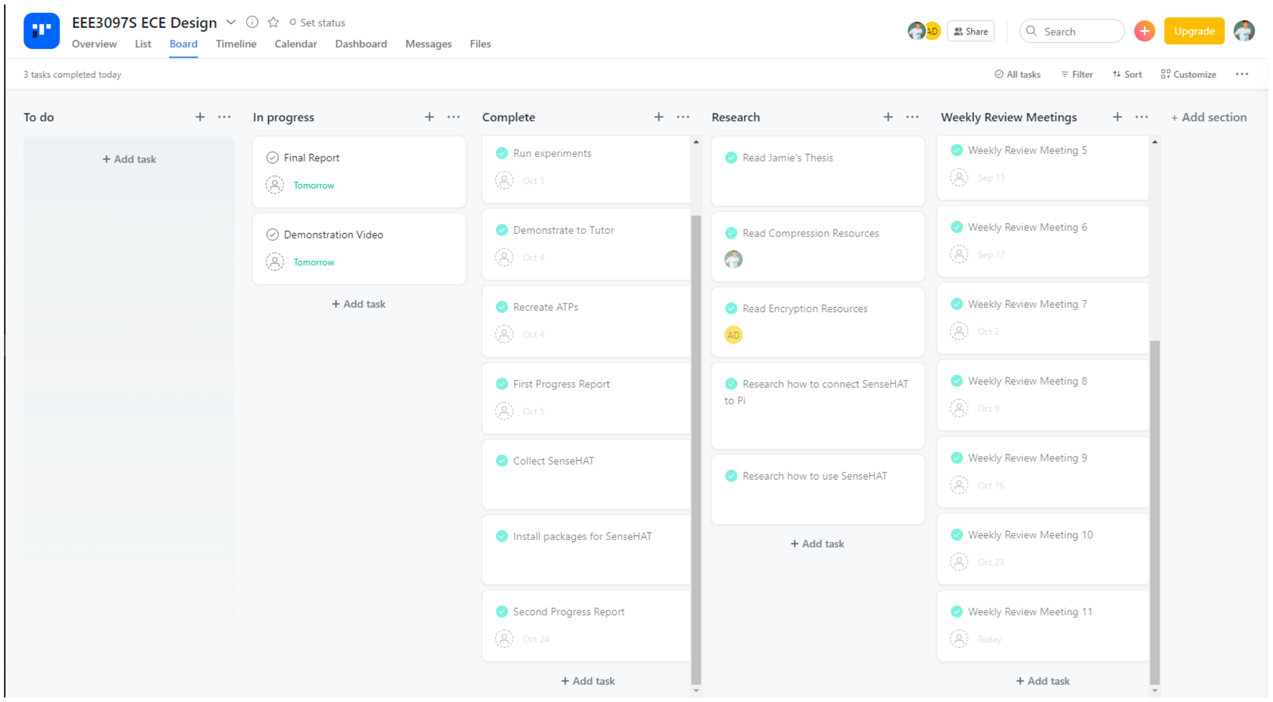
1. Admin Documents

1.1. Member Contributions

Contribution	Section Number	Page Number
DSLGRE001		
	2.2.2	6
	3.2	19
	5.1	45
	5.2	47
	5.3	50
	6.2	5.6
ADMALI004		
	2.2.1	6
	2.3	8
	Conclusion	80
	References	80
Shared	1	3
	2.1	5
	2.4	9
	2.5	10
	2.6	12
	2.7	17
	3.1	18
	4.1	19
	4.2	33
	4.3	43
	5.4	5151
	5.5	5454
	5.6	6262
	6.1	7878

Table 1. Member Contribution

1.2. Project management tool



To do

+ Add task

In progress

- Final Report — Tomorrow
- Demonstration Video — Tomorrow

+ Add task

Complete

- Run experiments — Oct 1
- Demonstrate to Tutor — Oct 4
- Recreate ATPs — Oct 4
- First Progress Report — Oct 5
- Collect SenseHAT
- Install packages for SenseHAT
- Second Progress Report — Oct 24

+ Add task

Research

- Read Jamie's Thesis
- Read Compression Resources
- Read Encryption Resources — AD
- Research how to connect SenseHAT to Pi
- Research how to use SenseHAT

+ Add task

Weekly Review Meetings

- Weekly Review Meeting 5 — Sep 11
- Weekly Review Meeting 6 — Sep 17
- Weekly Review Meeting 7 — Oct 2
- Weekly Review Meeting 8 — Oct 9
- Weekly Review Meeting 9 — Oct 16
- Weekly Review Meeting 10 — Oct 23
- Weekly Review Meeting 11 — Today

+ Add task

Figure 1. Snapshot of project management tool, Asana

1.3. Link To GitHub Page

https://github.com/TheAllyA/EEE3097S-Project

1.4. Project Timeline

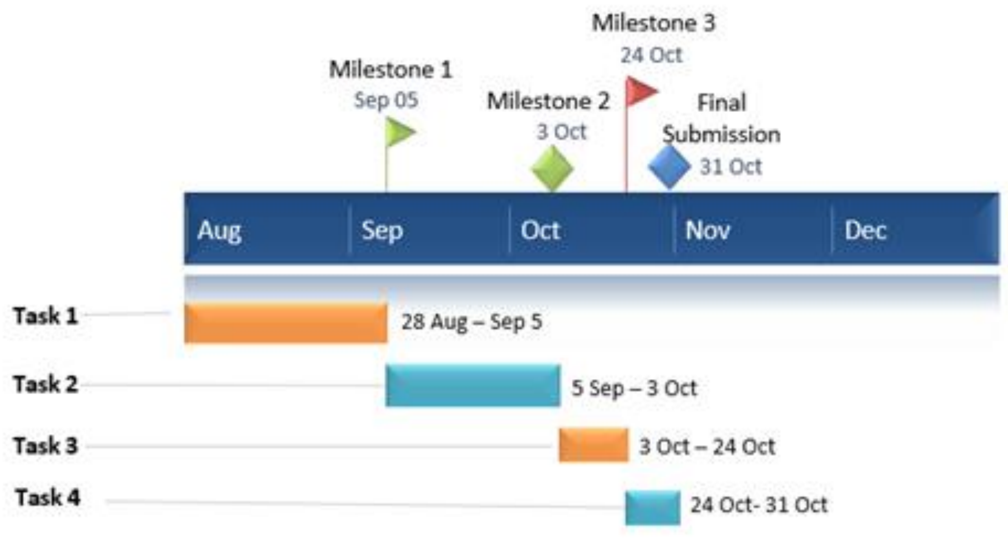


Figure 2. Snapshot of Project Timeline

No delays occurred throughout the project.

2. Requirement Analysis

2.1. Interpretation of the requirements

Requirement 1: The IMU to be used is ICM-20649. Since the IMU is not provided, we will be using a smartphone with MATLAB Mobile enabled to gather data that an IMU can provide us with, this data will be uploaded to the MATLAB cloud where we can apply our IP to. We will be designing our IP with this in mind.
Requirement 2: Oceanographers have indicated that they would like to be able to extract at least 25% of the Fourier coefficients of the data. Ensure that the compression satisfies this. The Accelerometer and Gyroscope and Maximum and Minimum ratings, if we choose an output data rate, we must ensure that 25% of the rate is used to extract the data. Matlab Mobile allows for a minimum sample rate of 0.5 - 100Hz. The 25% must fall within this sample rate.
Requirement 3: The data that is being extracted from Matlab Mobile needs to be encrypted and compressed before being transmitted using the RaspberryPi. This ensures that it is safe and not very costly.
Requirement 4: In addition to reducing the amount of data we also want to reduce the amount of processing done in the processor. Try to minimize the computation required for your IP. Practical solutions are: Switch off USB/LAN IC, turn off HDMI, Throttle CPU, Disable Wifi & Bluetooth, Disable on-board LEDs.
Requirement 5: The IP needs to be digital and ARM-based.

2.2. Comparison of some available compression and encryption algorithms

2.2.1. Encryption

When encrypting the data on a RaspberryPi there are certain methods that can be used to achieve this. A famous method is cryptography. This is a vast method that can be broken down even further into high level symmetric encryption(secret_key encryption), public_key encryption and hazardous materials layer encryption.
The high-level symmetric encryption is the easiest method to implement while still being safe. It requires using the Fernet method that can be imported onto the RaspberryPi. The Fernet method allows you to take data and encrypt it by using a self-generated key that will only be known by the people who need to decrypt the data. The Fernet method uses 3 import methods that already exist, namely, generate_key(), encrypt(data) and decrypt(data). The limitation with the data parameter is that it needs to be in bytes as the method returns a URL-safebase64-encoded byte.
An alternative method that could be used is the MultiFernet method that can also be called from the cryptography package. This takes a list of Fernet instances and rotates them. The first key is used to perform all encryptions on the data and the MultiFernet

rotation decrypts the data with each key after that. This process makes it easier to replace old keys which is an advantage as continually replacing keys could increase safety.
Another form of encryption that can be considered is to install additional hardware onto the RaspberryPi board that will ensure secure encryption takes place. The piece of hardware that can be installed is the Zymbit security module. Every security module contains a hardware root of trust that provides key generation and cryptographic functions. This hardware security module uses the ATECC508/ATECC608 secure elements from MicroChip which is secured by an additional layer of protection provided by a dedicated “security supervisor” microcontroller. This addition further prevents any attacks on the host from getting to the encrypted data. In the Zymbit security module, the encryption keys are generated inside the secure element. The private key that is used to encrypt the data never leaves the security module and the public key to decrypt the data can be found through the Zymbit API.
For autonomous operation, the security modules automatically generate a multi-factor device identity by measuring parameters of the host computer and that of the security module. These factors are combined to form a “fingerprint” that is stored inside the security module. This fingerprint is then permanently associated with the hardware.

2.2.2. Compression

LZ77 compression algorithm works by using a sliding window to find sequences of data that are repeated, and encoding each repeated sequence by a pair of numbers called a length-distance pair. The sliding window is used to examine the input data sequence, and to maintain the historical data that serve as the dictionary. The length-distance pair indicates that each of the next length characters is the same as the character exactly distance characters behind it in the original data stream.
Huffman encoding is a statistical compression method. It encodes data symbols (such as characters) with variable-length codes, and lengths of the codes are based on the frequencies of corresponding symbols. Huffman encoding, as well as other variable-length coding methods, has two properties:
Codes for more frequently occurring data symbols are shorter than codes for less frequently occurring data symbols.
Each code can be uniquely decoded. This requires the codes to be prefix codes, meaning any code for one symbol is not a prefix of codes for other symbols. For example, if code “0” is used for symbol “A”, then code “01” cannot be used for symbol “B” as code “0” is a prefix of code “01”. The prefix property guarantees when decoding there is no ambiguity in determining where the symbol boundaries are.
Huffman encoding has two steps:
Build a Huffman tree from original data symbols. A Huffman tree is a binary tree structure.
Traverse the Huffman Tree and assign codes to data symbols.
GZIP - Deflate algorithm, used in HTTP, PNG and PDF, combination of LZ77 and Huffman Coding. LZ77 is lossless, replaces repeated data with references. Three types

of data; Literals, Lengths and Distances. Huffman coding generates variable length codes. Allows for data to be uniquely decodable.
Zlib - The compression algorithm used in zlib is the deflate method. The deflate method encodes the input data into compressed data. The decompression algorithm used in zlib is the inflate method, which is the decoding process that takes a deflate bit stream for decompression and correctly produces the original full-size data or file. Deflate method is a combination of the LZ77 algorithm and Huffman encoding. zlib has 10 compression levels (0-9). Various levels have different compression performance in terms of compression ratio and speed. Level 0 means no compression, zlib will output the original data. Level 1 is the fastest, while it has low compression ratio. Level 9 gives the highest compression ratio, but the compression speed is slower.
The default compression level zlib uses is 6. zlib implements both static (fixed) Huffman encoding and dynamic Huffman encoding. For each block of LZ77 encoded data, zlib computes the number of bits in the block using both static Huffman encoding and dynamic Huffman
encoding, then choose the method which produces smaller amount of data. If the number of bits are equal using two methods, zlib chooses static Huffman encoding as the decoding process is faster.
The whole data stream can contain a mix of static and dynamic Huffman encoded data. The Huffman codes are transmitted in the deflate stream header for each block.
In summary, the Huffman encoding process in zlib consists of the following steps:
During LZ77 encoding process, collect statistics of data bytes and generate a histogram.
For each data block, build a dynamic Huffman tree using the collected statistics.
Compute and compare the encoded block lengths using a dynamic Huffman tree and a static Huffman tree, and decide whether it is worthwhile to use a dynamic or a static tree for the encoding phase.
Perform dynamic or static Huffman encoding on the block of data.

Summary:
Zlib is a data compression library that is designed to handle simple data (regardless of the source of the data).
Gzip is a file compression tool (or the compressed file format produced by the compression tool) that is designed to work with individual files. Gzip uses zlib to compress data in a file. In order to save information about file attributes, Gzip needs to save more header content in a compressed file (*.gz), and zlib do not have to consider this. But Gzip only applies to a single file, so we often see on the unix/linux of the compression packet suffix is *.tar.gz or *.tgz, that is, the first to use tar to package multiple files into a single file, and then use gzip compression results.

2.3. Feasibility analysis

Schedule:

A fixed schedule has been set out that spans over the next few weeks to complete the project by the set deadline. Every two weeks a milestone needs to be handed in that will require certain stages in the project plan to have been met. Every week a meeting is had between the people working on the project to discuss each step in the project plan and how it can be implemented. Thus, although there is time-constraints, the set-out schedule makes it possible to complete the project in time.

Technical:

We have been supplied with all the items needed to meet the requirements of the project. We will be using a smartphone to read the data extracted from the IMU as well as a RaspberryPi Zero to check the validity, compress and encrypt the data. This ensures that the IP is digital as no physical extractions need to take place as well as ARM-based since we will be using the RaspberryPi Zero. Thus, the project meets all the requirements and will be successful.

Operations:

The project will be operational as the requirements have been met and tests will be carried out to ensure that the design is operating correctly.

Compression:

Use a preset dictionary so at the beginning of the input has back reference for searching matches.

2.4. Bottlenecks

- Sensor Data from IMU on Smartphone may have different capabilities to that of the ICM-20649. This may affect simulated results.
- Matlab Mobile offers a sample rate of 0.5Hz - 100Hz. This may set limitations to how much data can be extracted through a simulation vs a real ICM-20649.
- Compression tests might fail and decompressed data differs from compressed data.
- The file may not encrypt completely or correctly and thus this will make the data vulnerable when it is being transmitted. The fernet method cannot encrypt files that are too large thus we need to ensure that the compression needs to be a sufficient size.
- The encryption and compression may be slow and thus in the end not all 25% of data will be transmitted.
- The largest limitation to the project is time constraints. The project timeline coincides with the SCALE research cruise using the winter and spring expeditions for buoy deployment thereby limiting the period for development. Additionally, the firmware development is limited to the capabilities of the selected processor.
- The firmware development is heavily constrained by the hardware selected for the platform. Peripheral drivers were written for modules that were agreed upon by the project members. Additionally, the IMU, processor, environmental sensor

and satellite modem components were pre-selected in 2018. The firmware is thus constrained by design choices originally made in 2018 and early 2019 for the first version of the system. However, these designs were revised for subsequent versions of the buoy from September 2019. As a result of these constraints, the devices were designed without previous knowledge of the environment and with a limited number of sensors. Finally, the selected processor has a limited number of communications peripherals which influenced the types of sensors that were selected. The communications network in Antarctica is severely limited and the most reliable form of remote communications is the Iridium satellite network. The amount of data that can be transmitted is limited in terms of bandwidth, data costs, packet's structure and reliability of transmission. Testing for Antarctic conditions is restricted by available testing facilities; therefore, rigorous environmental tests may only be conducted during the expeditions. The first prototype was deployed with a limited number of sensors due to development constraints. Mechanical failures resulted in the buoys ceasing operation within an hour of deployment during the 2019 SCALE Winter cruise. Further development occurred in 2020 to increase the sensing capability of the buoy. However due to the 2020 COVID-19 outbreak, all expeditions were cancelled for the year and therefore final system testing in the Antarctic MIZ was not possible. Attempts were made to deploy the devices on other expeditions from other countries. However, shipping delays were encountered preventing the device from reaching the expedition team on time. Currently, a prototype version has been sent with the SA Agulhas II to the SANAE IV base on the Antarctic continent where testing is expected to take place in late February 2021. This falls outside the period of this dissertation. Instead, the buoy will be tested on the home continent with low temperature tests being conducted in a commercial $-20°$ C freezer. Currently, two buoys are onboard the RV Polarstern and should be deployed in mid-March 2021 alongside with the Alfred Wegener Institute (AWI) buoys.

2.5. Subsystem and Sub-subsystems Requirements

R-01	**Data extraction**
Requirement	The system shall continuously measure and report specific gravity and angular rate of an object to which it is attached.
Rationale	Some academics in the department of electrical engineering work in the larger RDI consortium called SCALE. In one of the projects where we are involved, we are building a generic flexible buoy that could be installed on the pancake ice in Southern Ocean to gather environmental data. One of the crucial sensors in our buoy is the IMU. It gives information about the ice as well as wave dynamics. Ideally, we would like as much of the IMU data as possible. But, transmitting the data using Iridium is extremely costly. So, we want to compress the IMU data. We also want to encrypt the data.

Refined By	R-SS-01 R-SS-02 R-SS-03 R-SS-04 R-SS-08 R-SS-09
Verification	Requirement is verified by demonstration.

Table 2. Requirement 1

R-02	**Encryption**
Requirement	The data must be encrypted before being transferred. It can only be decrypted by a specific key.
Rationale	The data needs to be encrypted to ensure its safety and privacy
Refined By	
Verification	Requirement is verified by demonstration.

Table 3. Requirement 2

R-03	**Compression**
Requirement	The system shall compress the measured IMU data.
Rationale	We would like as much of the IMU data as possible. But, transmitting the data using Iridium is extremely costly, compressing the data allows for large data to be compressed into small data to send over large distances.
Refined By	R-SS-11 R-SS-12 R-SS-13 R-SS-19
Verification	Requirement is verified by demonstration.

Table 4. Requirement 3

R-04	**Data-integrity check**
Requirement	The system will be used to check if the extracted data is adequate and correct
Rationale	It needs to be established whether the extracted data is of an adequate type and that it is correct. This means checking if the results are correct and can be used to draw conclusions.
Refined By	
Verification	Requirement is verified by demonstration.

Table 5. Requirement 4

R-05	**Extract at least 25% of the Fourier coefficients of the data.**
Requirement	Ensure that the compression comtains at least 25% of the extracted data.
Rationale	Transmitting the data using Iridium is extremely costly. So, we want to compress the IMU data. Taking 25% of the Fourier coefficients of the data allows for enough data to be acquired while still saving costs. The communications network in Antarctica is severely limited and the most reliable form of remote communications is the Iridium satellite network. The amount of data that can be transmitted is limited in terms of bandwidth, data costs, packets structure and reliability of transmission. Testing for Antarctic conditions is restricted by available testing facilities; therefore, rigorous environmental tests may only be conducted during the expeditions.
Refined By	R-SS-05 R-SS-06 R-SS-07
Verification	Requirement is verified by demonstration.

Table 6. Requirement 5

R-06	**Reduce power consumption**
Requirement	Reduce the amount of processing done in the processor (as it takes up power which is limited). Minimize the computation required for the IP.
Rationale	The current PCB stack configuration provides too many points of failures. It is recommended to design a single, horizontally mounted PCB with all the sensors and microcontroller. Additionally, low-powered LED arrays can be implemented to provide better visual feedback on the status of the buoy. The device enclosure should be redesigned to allow for a larger power supply. In addition, the enclosure should include a mechanism for de-humidifying the internal electronics. Finally, a dedicated power board and communication module board should be designed to replace the breakout boards that the GNSS and Iridium modem modules arrive on. This will reduce the form factor and reduce the reliance on connectors that can act as points of failure.
Refined By	R-SS-08 R-SS-14 R-SS-15 R-SS-16 R-SS-17 R-SS-18
Verification	Requirement is verified by demonstration.

Table 7. Requirement 6

2.6. Subsystem and Sub-subsystems Specifications

R-SS-01	**DIGITAL MOTION PROCESSOR**
	The embedded Digital Motion Processor (DMP) within the ICM-20649 offloads computation of motion processing algorithms from the host processor. The DMP acquires data from accelerometers, gyroscopes, and additional third-party sensors such as magnetometers, and processes the data. The resulting data can be read from the FIFO. The DMP has access to the external pins, which can be used for generating interrupts. The purpose of the DMP is to offload both timing requirements and processing power from the host processor. Typically, motion processing algorithms should be run at a high rate, often around 200 Hz, to provide accurate results with low latency. This is required even if the application updates at a much lower rate; for example, a low power user interface may update as slowly as 5 Hz, but the motion processing should still run at 200 Hz. The DMP can be used to minimize power, simplify timing, simplify the software architecture, and save valuable MIPS on the host processor for use in applications. The DMP is optimized for Android Lollipop support.

Table 8. Specification 1

R-SS-02	**PRIMARY I2C AND SPI SERIAL COMMUNICATIONS INTERFACES**
	The ICM-20649 communicates to a system processor using either a SPI or an I2C serial interface. The ICM-20649 always acts as a slave when communicating to the system processor. The LSB of the of the I2C slave address is set by pin 1 (AD0).

Table 9. Specification 2

R-SS-03	AUXILIARY I 2C SERIAL INTERFACE
	The ICM-20649 has an auxiliary I 2C bus for communicating to an off-chip 3-Axis digital output magnetometer or other sensors. This bus has two operating modes: • I 2C Master Mode: The ICM-20649 acts as a master to any external sensors connected to the auxiliary I 2C bus • Pass-Through Mode: The ICM-20649 directly connects the primary and auxiliary I 2C buses together, allowing the system processor to directly communicate with any external sensors.

Table 10. Specification 3

R-SS-04	AUXILIARY I 2C BUS MODES OF OPERATION
	I 2C Master Mode: Allows the ICM-20649 to directly access the data registers of external digital sensors, such as a magnetometer. In this mode, the ICM-20649 directly obtains data from auxiliary sensors without intervention from the system applications processor. For example, in I2C Master mode, the ICM-20649 can be configured to perform burst reads, returning the following data from a magnetometer: • X magnetometer data (2 bytes) • Y magnetometer data (2 bytes) • Z magnetometer data (2 bytes) The I2C Master can be configured to read up to 24 bytes from up to 4 auxiliary sensors. A fifth sensor can be configured to work single byte read/write mode. Pass-Through Mode: Allows an external system processor to act as master and directly communicate to the external sensors connected to the auxiliary I2C bus pins (AUX_DA and AUX_CL). In this mode, the auxiliary I2C bus control logic (3rd party sensor interface block) of the ICM-20649 is disabled, and the auxiliary I2C pins AUX_CL and AUX_DA (pins 7 and 21) are connected to the main I2C bus (Pins 23 and 24) through analog switches internally. Pass-Through mode is useful for configuring the external sensors.

Table 11. Specification 4

R-SS-05	MATLAB Support Package for Android Sensors – Package to access sensor data from phone
	MATLAB® Support Package for Android™ Sensors enables you to collect sensor data from your mobile Android device, such as a phone or tablet, log it in MATLAB, and then use MATLAB to process the data. You can collect data from acceleration, angular velocity, orientation, magnetic field, and GPS sensors, as well as images from the built-in camera. This support package is functional for R2020b and beyond. To use this package, you must also install MATLAB Mobile™ on your Android device. MATLAB Support Package for Android Sensors lets you acquire and log data from the supported Android sensors to obtain the following measurements: Acceleration on 3-axes Magnetic field on 3-axes Angular velocity on 3-axes

	Azimuth, roll, and pitch Latitude, longitude, altitude, horizontal accuracy, speed, and course Images from the integrated camera With MATLAB Mobile, you can already log sensor data to MATLAB Drive™. With this support package, you can stream your sensor data over the internet to a MATLAB session. To do so, you must be logged in to MATLAB Mobile and a MATLAB session using the same MathWorks® account. You can even connect and collect data from multiple devices simultaneously. This support package is pre-installed for MATLAB Online, which means you can also stream your device data directly to MATLAB Online.

Table 12. Specification 5

R-SS-06	**Matlab Mobile – Software needed on smartphone to transmit IMU data**
	Acquire data from built-in sensors on your device. You can acquire the following data: • Acceleration on 3-axes • Angular velocity on 3-axes • Magnetic field on 3-axes • Orientation (azimuth, pitch, and roll) • Position (latitude, longitude, altitude, horizontal accuracy, speed, and course) You can stream sensor data directly to the MathWorks Cloud for analysis in MATLAB Mobile or MATLAB Online or save the data to log files when you are offline. With support packages, you can analyze the data in MATLAB and build applications that leverage the data. Sample Rate: 0.5 - 100 Hz

Table 13. Specification 6

R-SS-07	**Matlab – Process Data**	
	Operating Systems	Windows 10
	Processors	AMD x86-64 processor with four logical cores and AVX2 instruction set support
	Disk	2.9 GB of HDD space for MATLAB only, 5-8 GB for a typical installation
	RAM	8 GB
	Graphics	No specific graphics card is required.

Table 14. Specification 7

R-SS-08	**ICM-20649 ACCELEROMETER SPECIFICATIONS**		
		MIN	**MAX**
	OUTPUT DATA RATE: Low-Power Mode	0.27 Hz	562.5 Hz

	Low-Noise Mode
	ACCEL_FCHOICE = 1;
	ACCEL_DLPFCFG = x
		4.5 Hz	1.125k Hz

	Low-Noise Mode
	ACCEL_FCHOICE = 0;
	ACCEL_DLPFCFG = x
		-	4.5k Hz

Table 15. Specification 8

R-SS-09	**ICM-20649 GYROSCOPE SPECIFICATIONS**		
		MIN	**MAX**
	Acceleration (Any Axis, unpowered) ABSOLUTE MAXIMUM RATINGS:		10.000g for 0.2 ms
	OUTPUT DATA RATE Standard (duty-cycled) Mode	4.4 Hz	562.5 Hz

	Low-Noise Mode
	GYRO_FCHOICE = 1;
	GYRO_DLPFCFG = x
		4.4 Hz	1.125k Hz

	Low-Noise Mode
	GYRO_FCHOICE = 0;
	GYRO_DLPFCFG = x
			9k Hz

Table 16. Specification 9

R-SS-10	**RaspberryPi specifications**
	CPU: ARM11 RAM: 512 MB Wireless: 2.4GHz 802.11n wireless LAN GPIO: 40-pin GPIO

Table 17. Specification 10

R-SS-11	**GZIP**
	Deflate algorithm Combination of LZ77 and Huffman Coding. LZ77 is lossless, replaces repeated data with references. Three types of data; Literals, Lengths and Distances. Huffman coding generates variable length codes. Allows for data to be uniquely decodable

Table 18. Specification 11

R-SS-12	**LZ77**
	Lossless Replaces repeated data with references. Output a triple (o, l, c) where, o: offset, represents the number of positions that we would need to move backwards in order to find the start of the matching string. l: length, represents the length of the match. c: character, represents the character that is found after the match.

Table 19. Specification 12

R-SS-13	**Huffman Coding**
	lossless data compression algorithm Generates variable length codes. Allows for data to be uniquely decodable

Table 20. Specification 13

R-SS-14	**Switch OFF USB/LAN IC**
	Turn OFF USB chip: echo '1-1' |sudo tee/sys/bus/usb/drivers/usb/unbind Turn ON USB chip echo '1-1' |sudo tee /sys/bus/usb/drivers/usb/bind Saving time: Total = 7hrs 21mins

Table 21. Specification 14

R-SS-15	**Turn OFF HDMI**
	Turn OFF HDMI output sudo /opt/vc/bin/tvservice -o Turn ON HDMI output sudo /opt/vc/bin/tvservice -p Saving time: Total = 4hrs 11mins

Table 22. Specification 15

R-SS-16	**Throttle CPU**

	arm_freq_min=250
	core_freq_min=100
	sdram_freq_min=150
	over_voltage_min=0

Table 23. Specification 16

R-SS-17	**Disable Wi-Fi & Bluetooth**

	dtoverlay=pi3-disable-wifi
	
	dtoverlay=pi3-disable-bt
	
	You must reboot the Raspberry Pi for the changes to take effect.
	Saving time: Total = 4hrs 51mins

Table 24. Specification 17

R-SS-18	**Disable on-board LEDs**
	dtparam=act_led_trigger=none dtparam=act_led_activelow=on You must reboot the Raspberry Pi for the changes to take effect. The power saving here is very similar to turning off the HDMI output on the Raspberry Pi, therefore expect very similar results of a power saving around 30mA. Saving time: Total = 4hrs 11mins

Table 25. Specification 18

R-SS-19	**Python 3.9.7**
	Language used to design Compression algorithm Easy to code Free and Open Source Object-Oriented Language High-Level Language Extensible feature – Can be integrated with C/C++ Python is Portable language – Can run code on Windows and Linux/Mac Large Standard Library

Table 26. Specification 19

Subsystem	**Encryption**
Specification	pip module Cryptography AES in CBC mode with a 128-bit key PKC7 padding Base64url encoding

Table 27. Specification 20

R-SS-20	**Python 3.6+**
	Used for encryption OpenSSL 1.1.0 latest Supported on multiple OS

Table 28. Specification 21

Subsystem	**Data-integrity**
Specification	Math modules on RaspberryPi

Table 29. Specification 22

2.7. Acceptance Test Procedure based on Specifications

2.7.1. Figures of merits, based on which you would validate your final design

Decompressed Data and Raw Data have no errors or inconsistencies - This will pass the functionality test of the code. This will be done via a computer Encrypted data are inaccessible to outside parties – this will be done via computer. Extracted data is viable and maintained integrity – this will be done via computer. Choose an existing sensor on the Pi to gather data and test that the algorithms work - This will pass the test of showing that the Pi can process data from an on-board sensor.

2.7.2. Experiment design to test these figures of merit

Compress data extracted from smartphone, then decompress data, compare the two data to ensure no data is missing in decompress data. If Raw data and Decompressed data are the same, Compression is successful and Compression Algorithm is successful. The encrypted data will be sent to a device that is able to read it e.g., smartphone. The process of decrypting the data with the correct key will be tested as well decrypting it with the incorrect key. This is to ensure that only the correct key works and that the data does decrypt correctly. To ensure that the data that has been extracted from the sensor is usable it will be checked against certain parameters. It will be checked to ensure that it is a float/byte, that it is an adequate length i.e. it is not a space that is being read and that it is within the viable range of values for the readings. The Pi-Zero has an on-Board temperature sensor, by recording this data over a period of time and processing the data, followed by compressing and encrypting the data and then decrypting and decompressing the data, we can test if the algorithm works on an On-Board sensor. If the test is successful, the algorithm and Pi will have successfully met the requirements

2.7.3. Acceptable performance definition

Acceptable performance in this unit will be the satisfactory achievement of implementation and demonstration using data from computer through setting up tests to check the functionality of the code as well as implementation and demonstration using data from an On-board sensor showing that the Pi can process data from an on-board sensor.

3. Paper Design

3.1. Inter-Subsystem and Inter-Sub-subsystems Interactions

General overview of the system:
Data gets extracted onto the RaspberryPi -> Data gets proof-checked for integrity -> Data gets compressed-> Data gets encrypted -> Data gets transmitted
Data Extraction:
Smartphone (IMU) -> Matlab Mobile -> Matlab(RaspberryPi)
Data-integrity check:
Matlab(RaspberryPi) -> Data item is checked for integrity -> Data item is sent to a file
Compression:
Data file -> LZ77 Encoding -> Huffman Encoding -> Data file gets encrypted
Encryption:
Compressed file gets read -> The Fernet method is used to encrypt the data -> Data gets transmitted.

3.2. UML Diagram

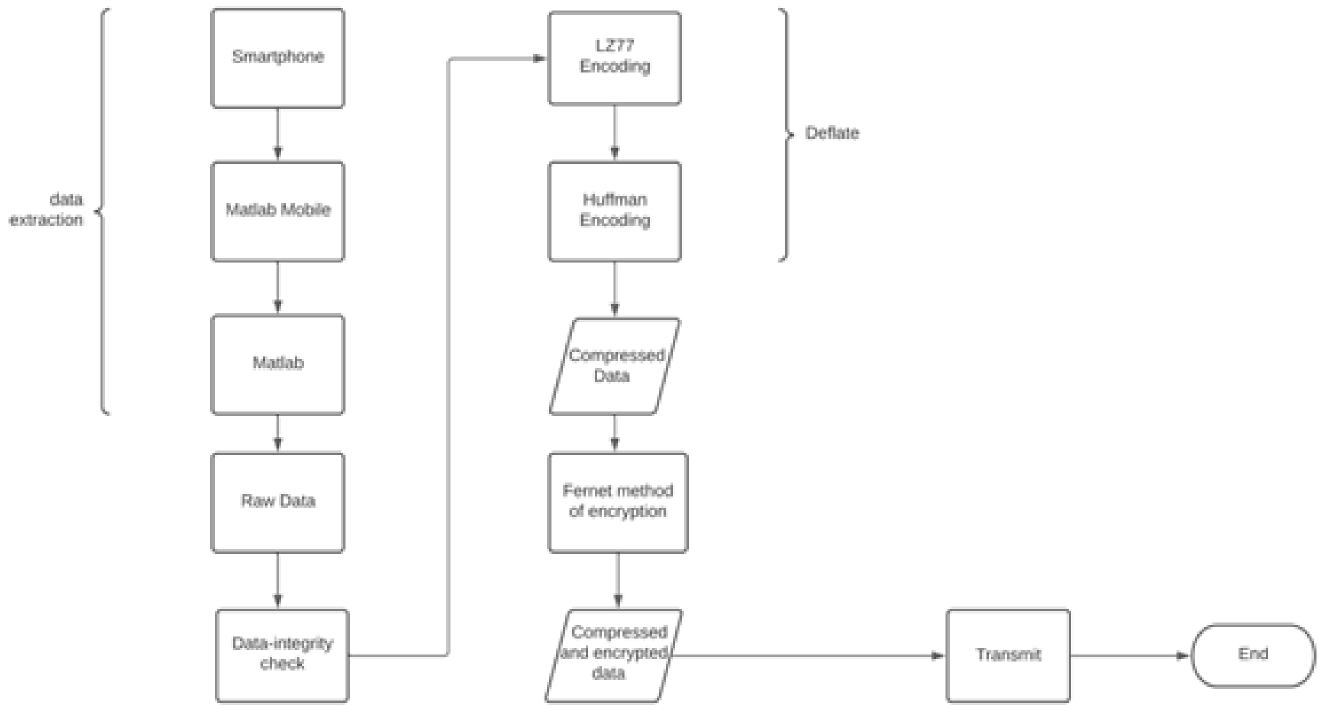


Figure 3. UML Diagram

4. Validation using Simulated or Old Data

4.1. The Need to do Simulation Based Validation

The data we will be using will be the Raw Data provided by Prof Mishra in the Week 5 Lesson's page which contained a zip file which was shared by the loken_21 paper. This data contains IMU data from another cruise (different from the one that they have reported in the paper, these are raw data form February 2021:Raw Dataset. This is clean data and all in csv files (easy to load). These IMU (and GPS) data are in "semi binary" format and can be parsed using the code at: Binary Data Parser. This data is from an IMU on a boat, so it may not be representative of other kinds of IMU uses / other vehicles etc. We are also waiting to get data sets from SHARC Buoy. More details to be provided by Prof Mishra soon. We are using this data as it best represents the type of data, we will expect to be produced by the IMU we are basing our design around. Although the dynamics of the boat and SHARC Buoy are different, the type of data that we expect to read is the same.

The data we are going to use is more than sufficient to run all ATPs, we were planning to use MATLAB Mobile to generate data from onboard sensors of a phone, this being, acceleration in x,y,z – plane, Magnetic Field in X, Y, Z – plane, orientation (Azimuth, Pitch, Roll), angular velocity in X, Y, Z –plane and Position in Latitude, Longitude, Speed, course, altitude, and horizontal accuracy.
The Raw Data provided by the course's lessons tab offers all the information that MATLAB Mobile can provide and more, with Temperature, Pressure and DCM in addition, this gives us even more accurate information that we would need to process with an actual IMU.
Although the IMU on the Boat does not generate the same Data as the IMU on the SHARC Buoy, it is still much more accurate to a real-world application compared to a phone where a person would have to physically move the phone around to gather different data over time.
The IMU on the boat is much closer to the scenario than the sensors on the phone in terms of the data we want to analyze.
MATLAB Mobile is also restricted to a sample rate of 0.5 to 100 Hz which limited our ability to capture data, the data provided by the IMU on the boat allows for a much better analysis and application of our algorithms.
The Plots of some key data below were taken from Matlab, the frequency domain plots were acquired using System Identification toolbox which did the Fourier transform on the time data.
The plots were taken over 150 seconds. 3D maps were used to represent y- and z- plane data.

Magnetic Field:

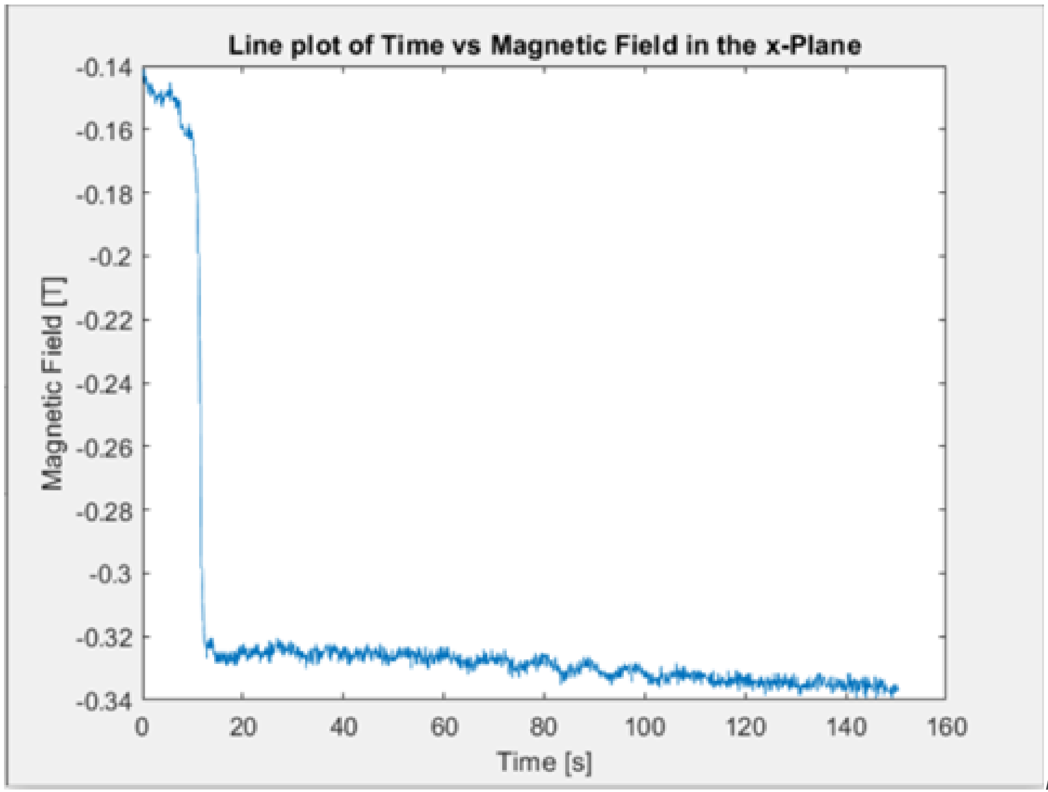


Figure 4. Magnetic Field vs Time in x plane

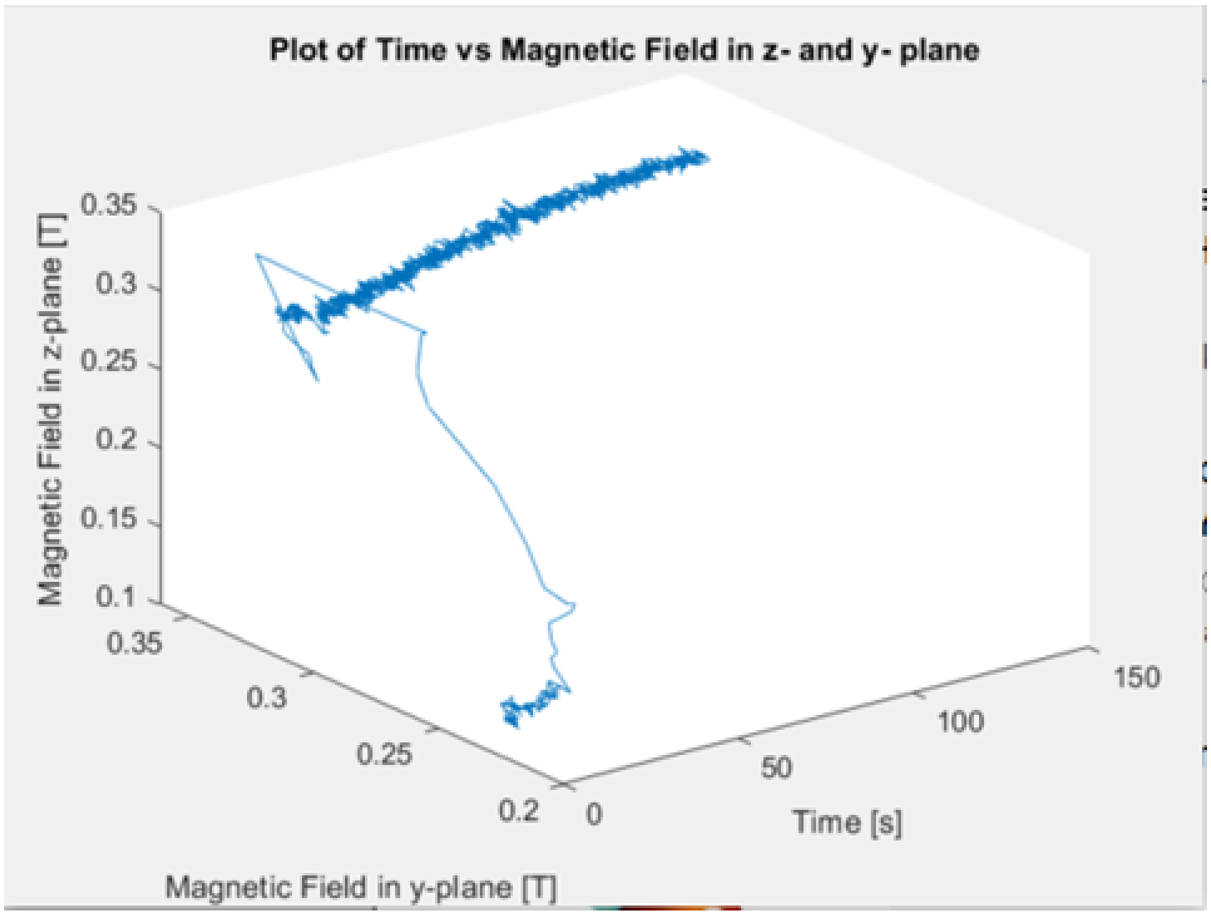


Figure 5. Magnetic Field vs Time in z and y plane

Fourier transform of x-plane data:

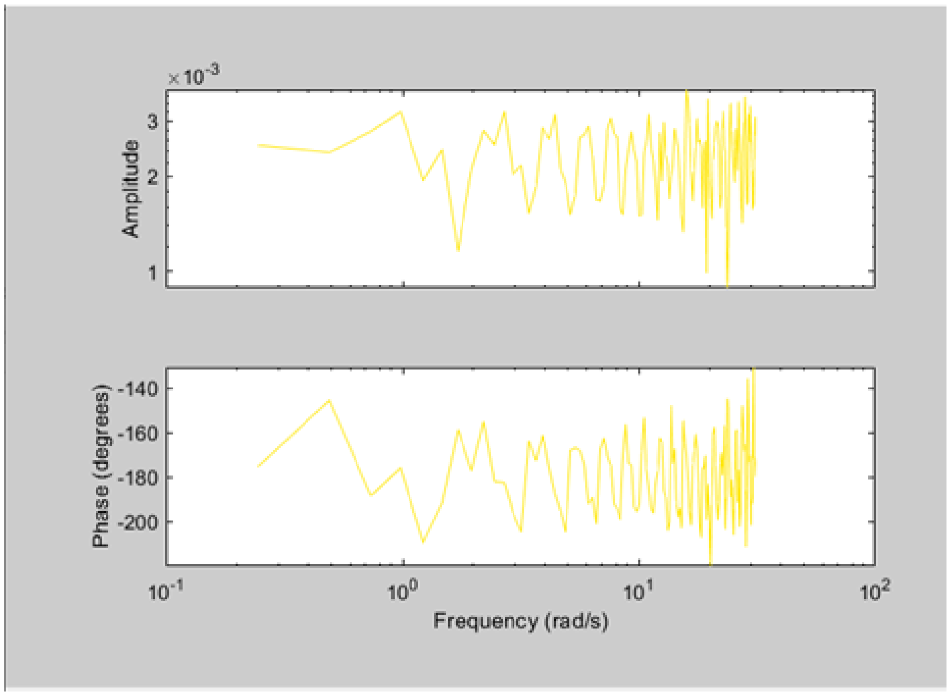


Figure 6. Fourier Transform of x plane data

Acceleration:

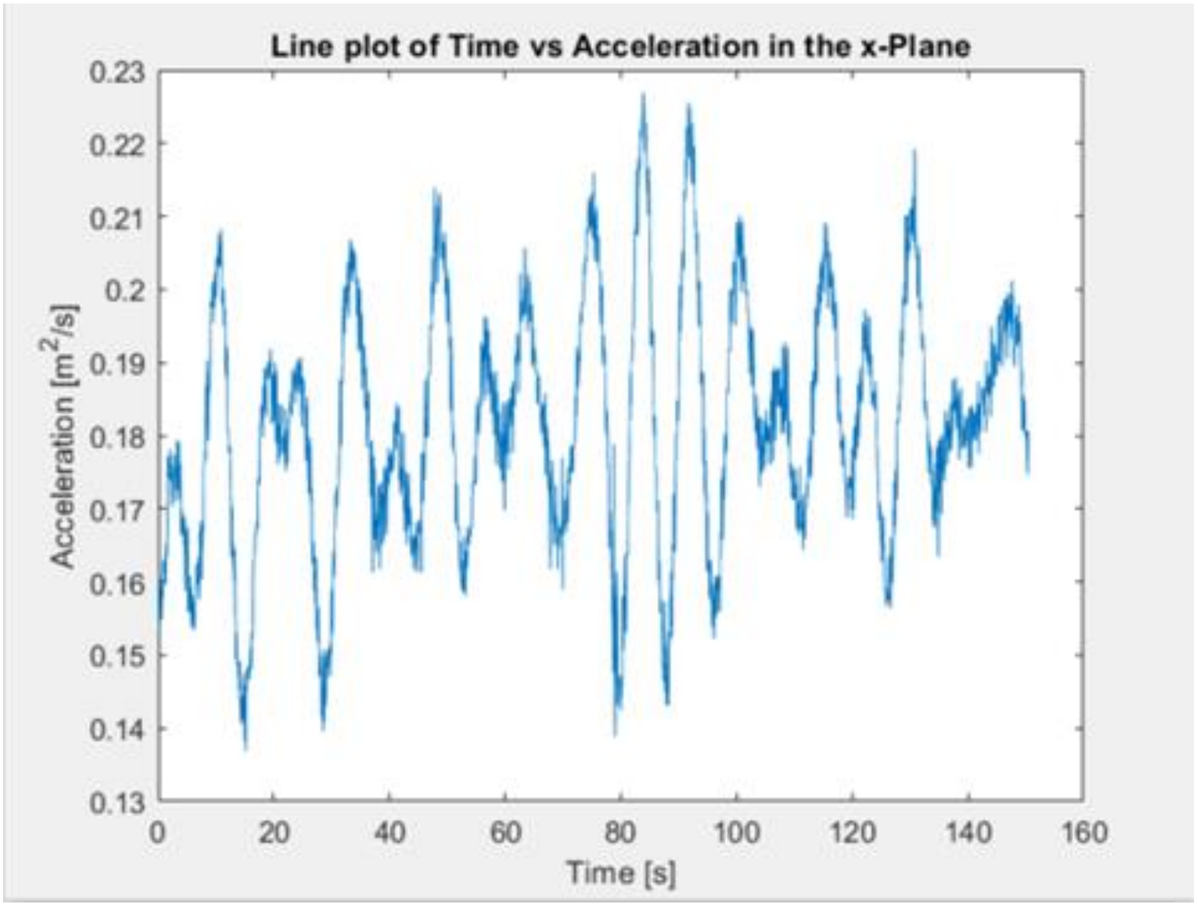


Figure 7. Graph of Acceleration vs Time in x plane

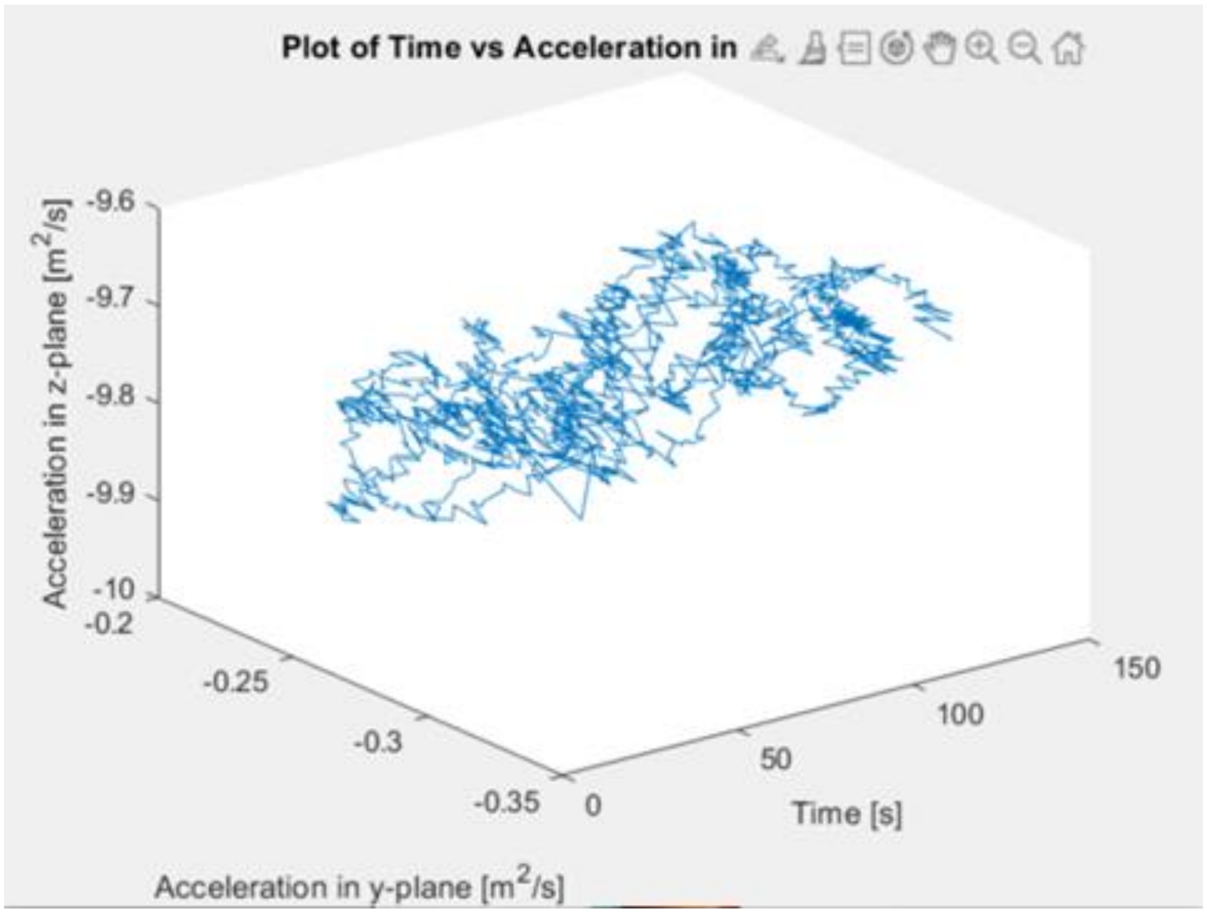


Figure 8. Acceleration vs Time in the x and y plane

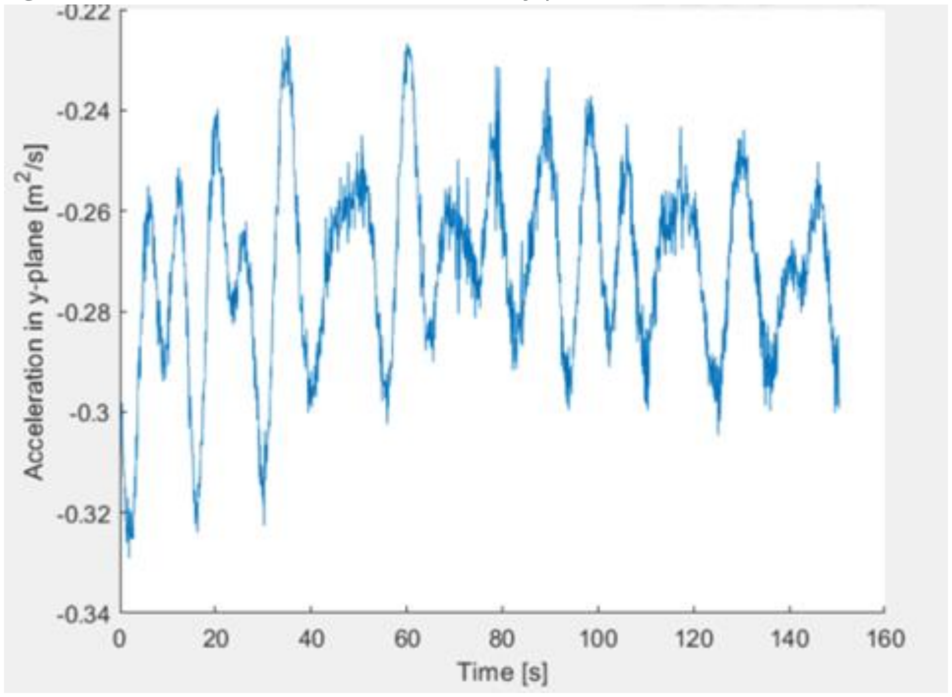


Figure 9. Acceleration vs Time in the y plane

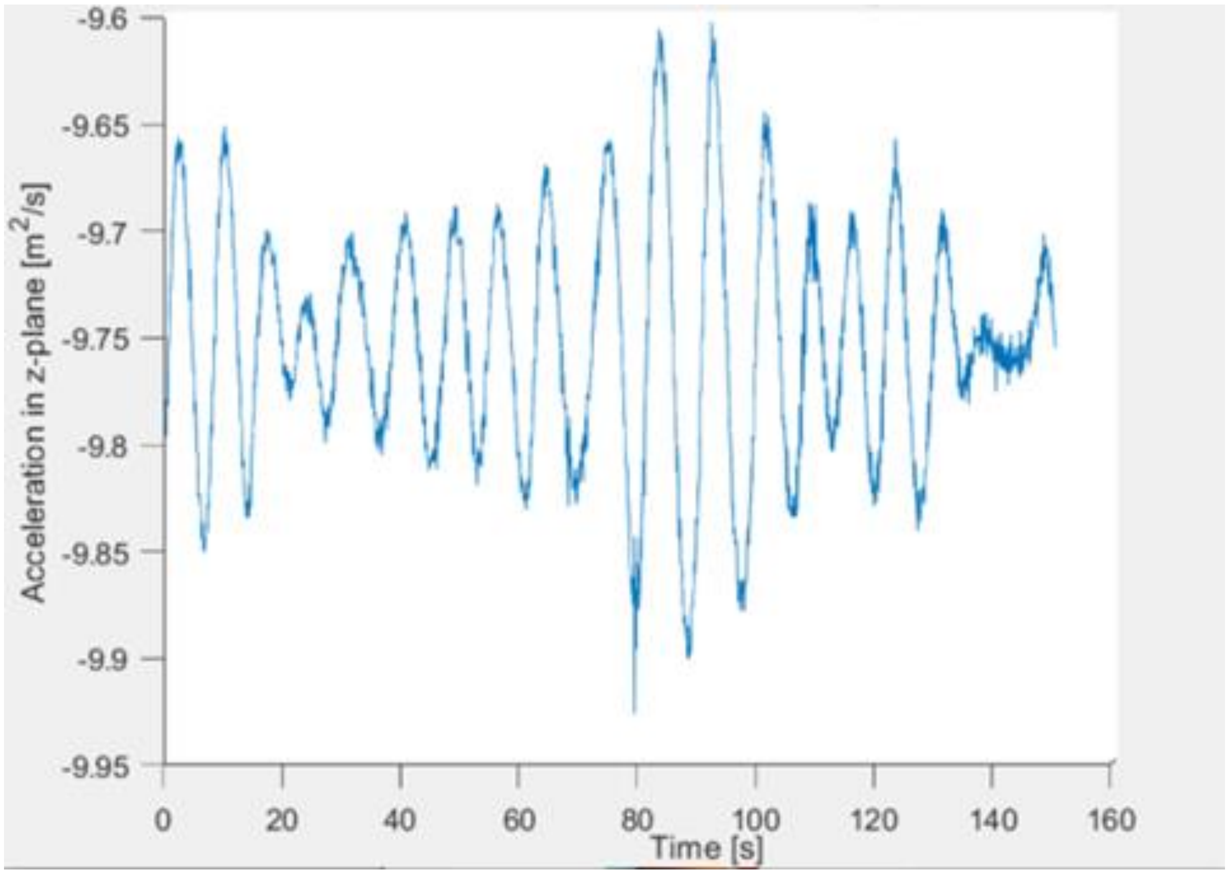


Figure 10. Acceleration vs Time in the z plane

Fourier transform of x-plane data:

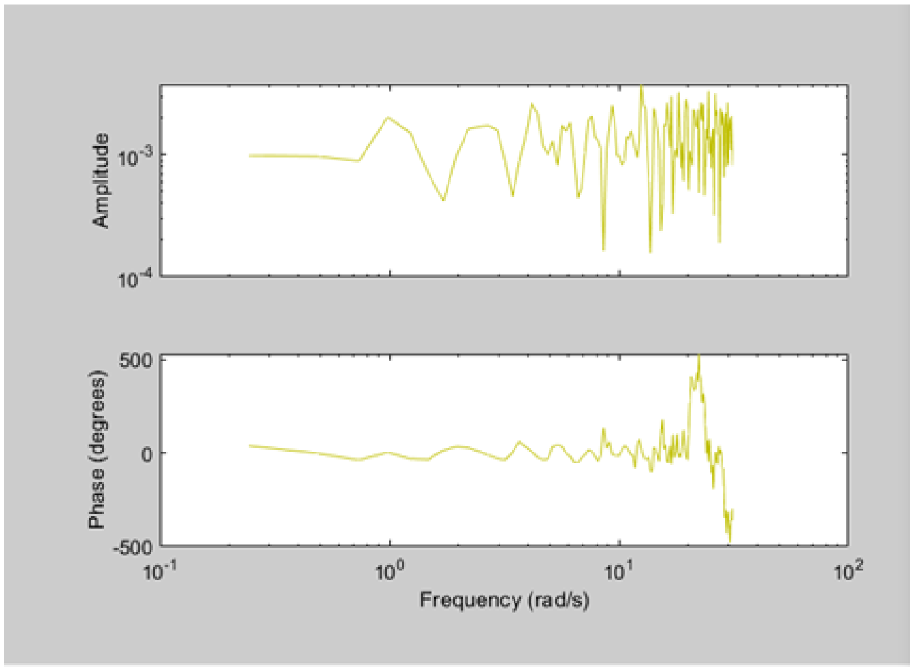


Figure 11. Fourier Transform of x plane data

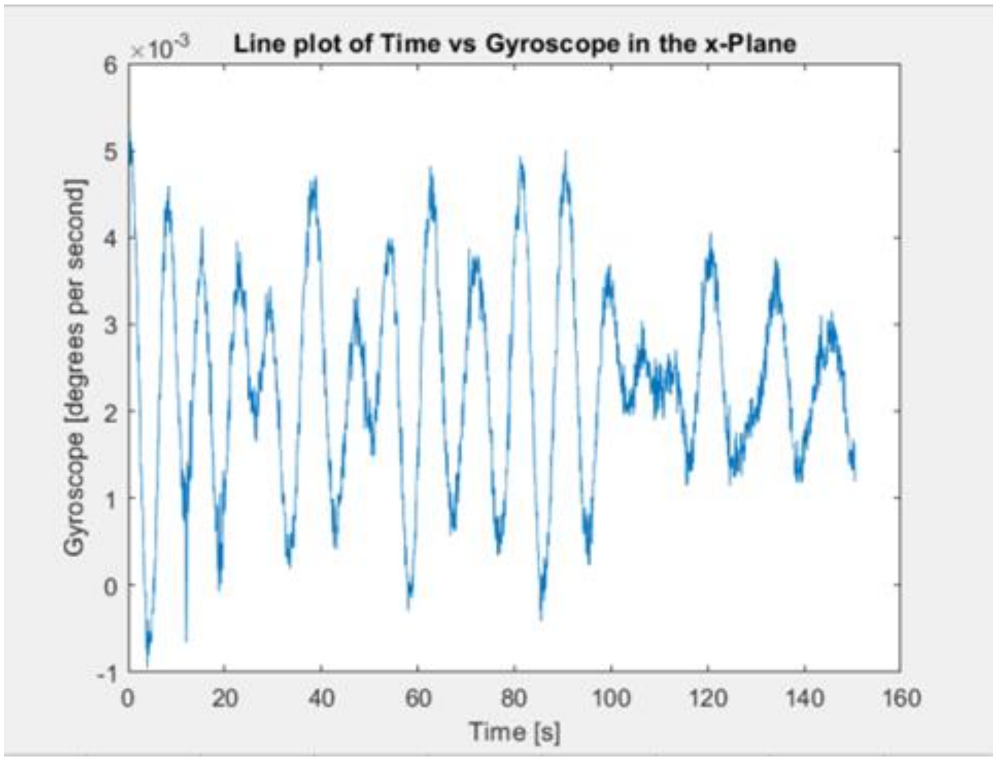


Figure 12. Gyroscope vs Time in the x plane

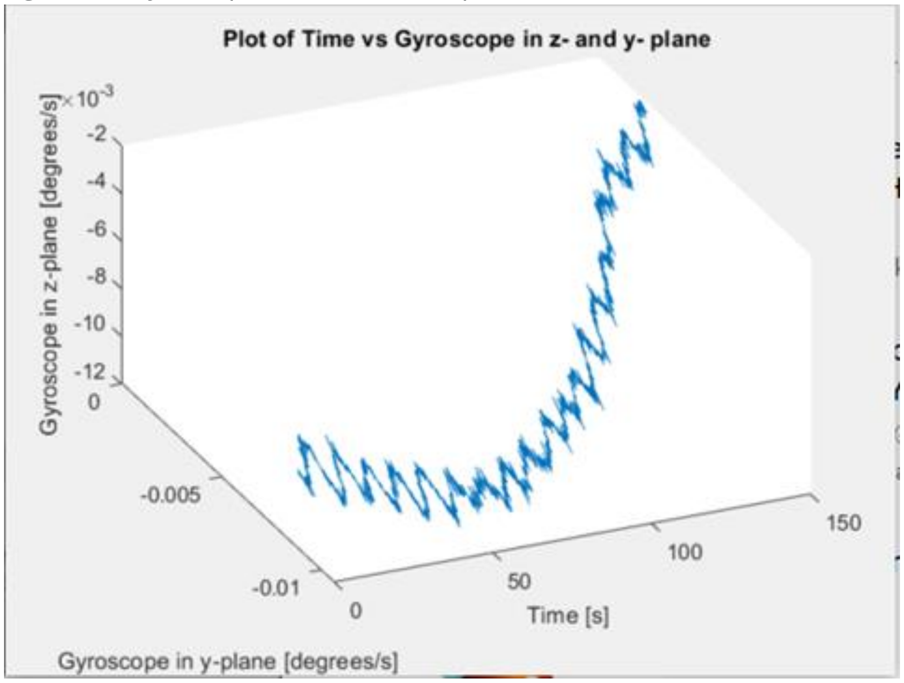


Figure 13. Gyroscope vs Time in x and y plane

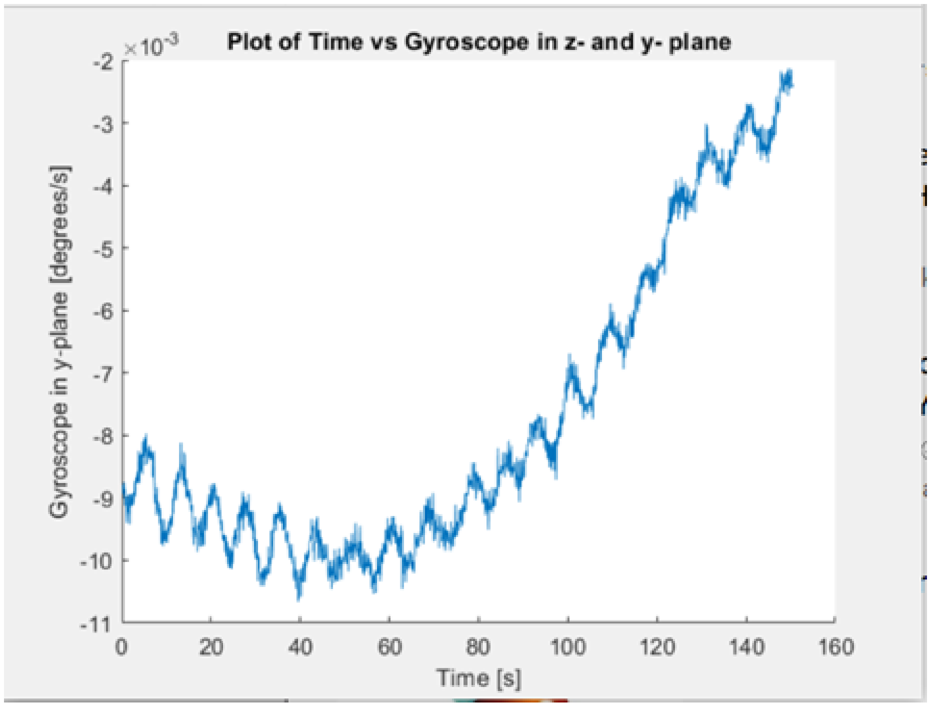


Figure 14. Gyroscope vs Time in the z and y plane

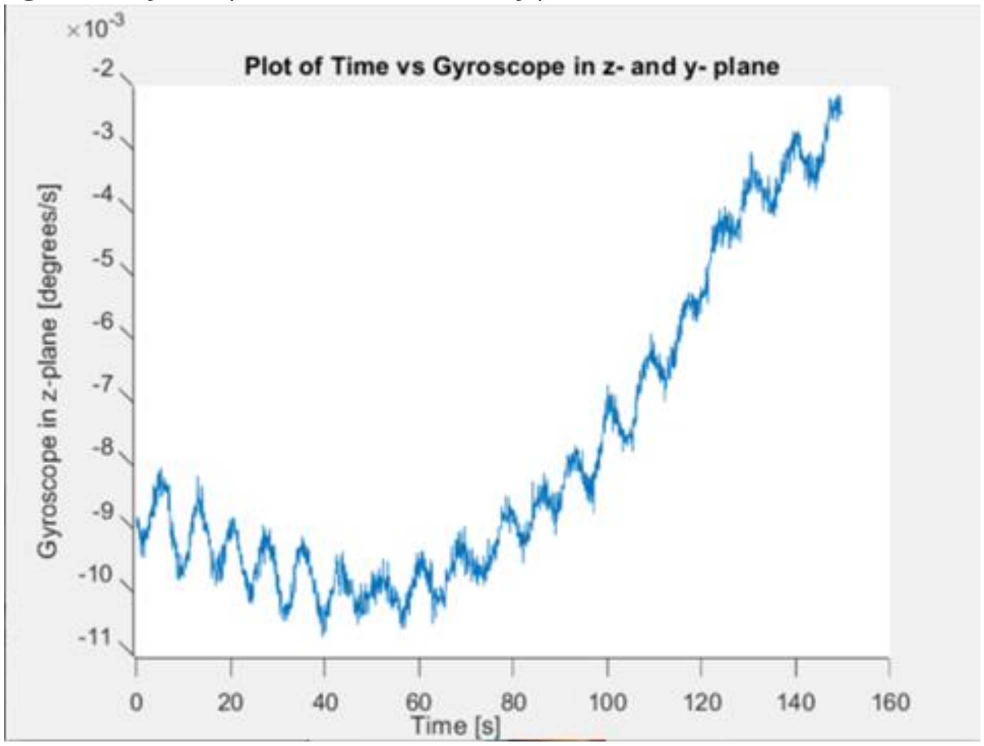


Figure 15. Gyroscope vs Time in the z plane

Fourier transform of x-plane data:

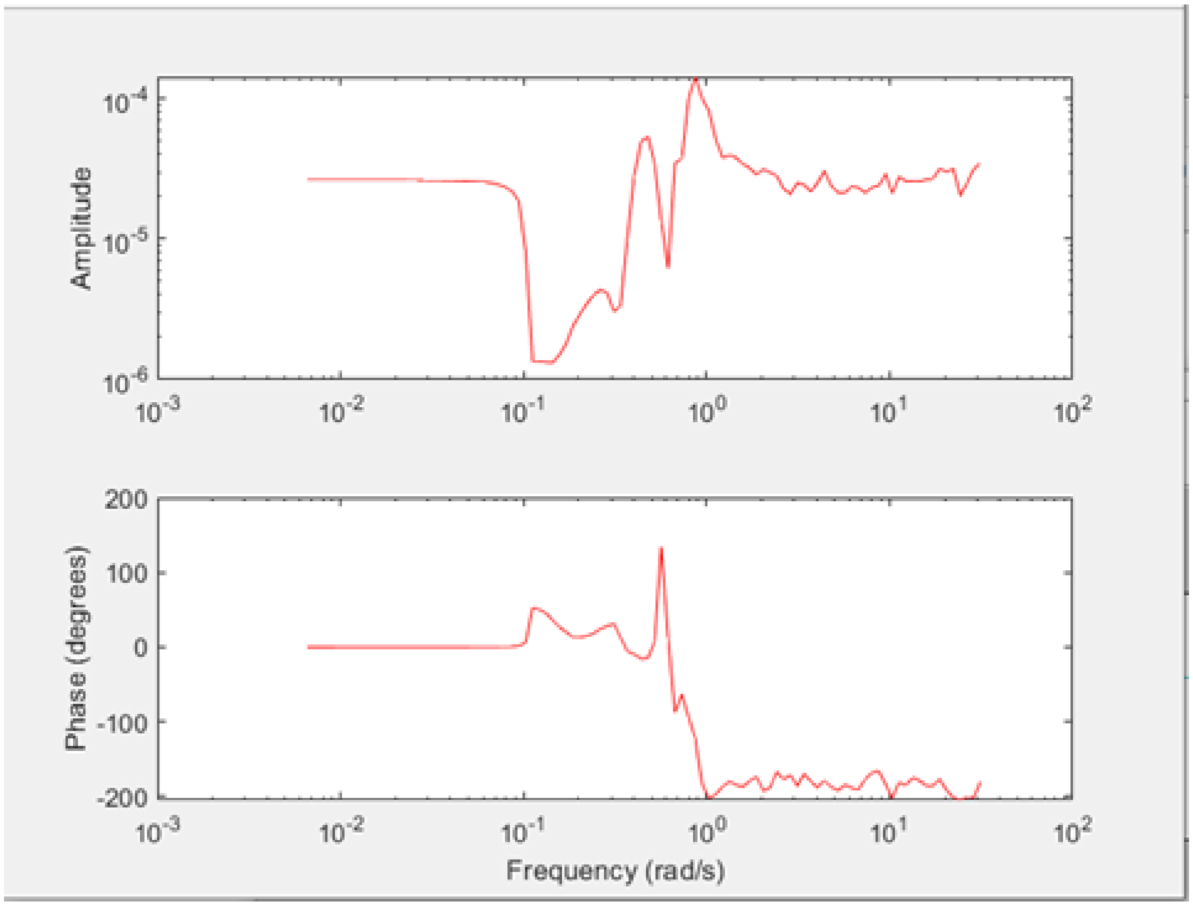


Figure 16. Fourier Transform of x plane data

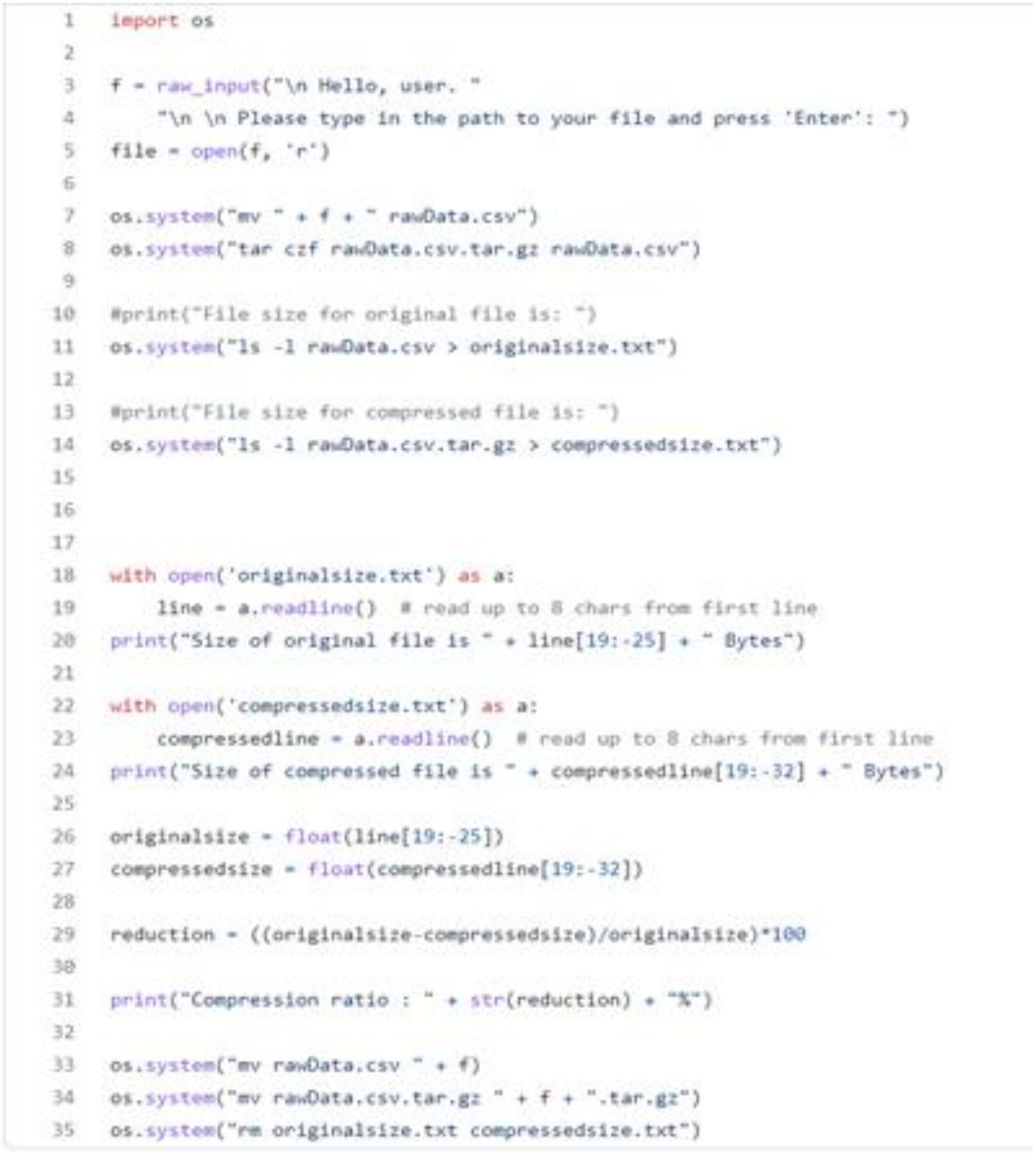
4.2. Experiment Setup

Overall:

Compress and Encrypt:

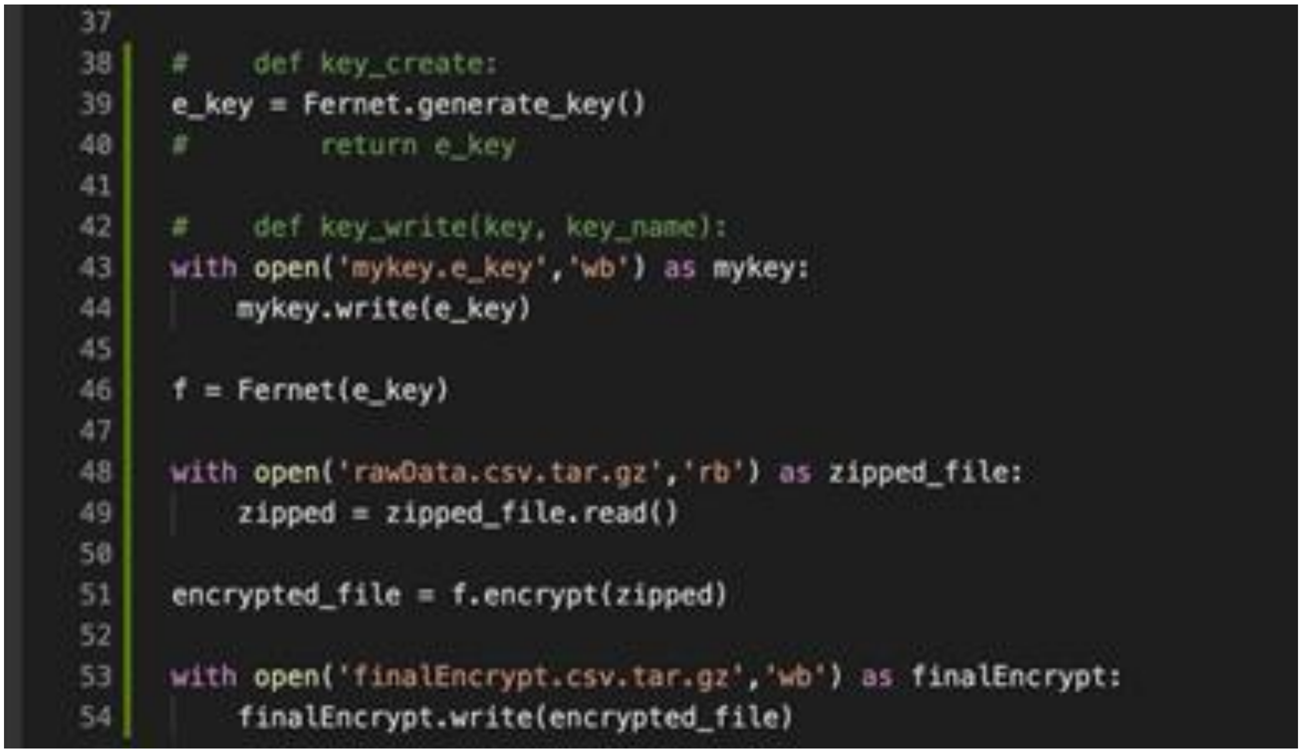
Begin by cloning the git repo https://github.com/TheAllyA/EEE3097S-Project
The first step is compression, when running the file DesignCompEncr.py (https://github.com/TheAllyA/EEE3097S-Project/blob/main/DesignCompEncr.py), this will be run with Python3.
The user will be prompted to input the path to the file that must be compressed and encrypted. This repo has CSV files that will be used in the experiment. After the user enters the file path, the program will open the file inputted by the user and then rename it to a temporary file called rawData.csv. Thereafter it runs the bash command that initiates compression, after compression it produces a compressed file called rawData.csv.tar.gz. Running the BASH command ‘ls -l’ produces the information about a file which includes its file size, using the command “>” we can send that to a text file, we do this on rawData.csv and rawData.csv.tar.gz, this will allow us to access information about the original and compressed file. Thereafter the program opens the text files that contains the file size of rawData.csv which is the original file

and using string slicing, we can access the value of the file size and output it to the user. We do the same with the compressed file size text file. After that we convert the file sizes to floats and perform a mathematical operation to produce a percentage that shows how much the original file has been reduced by. We then rename the files to the original file names that the user inputted and remove the text files that contain the sizes of the compressed and original files.



```
import os

f = raw_input("\n Hello, user. "
    "\n \n Please type in the path to your file and press 'Enter': ")
file = open(f, 'r')

os.system("mv " + f + " rawData.csv")
os.system("tar czf rawData.csv.tar.gz rawData.csv")

#print("File size for original file is: ")
os.system("ls -l rawData.csv > originalsize.txt")

#print("File size for compressed file is: ")
os.system("ls -l rawData.csv.tar.gz > compressedsize.txt")



with open('originalsize.txt') as a:
    line = a.readline()  # read up to 8 chars from first line
print("Size of original file is " + line[19:-25] + " Bytes")

with open('compressedsize.txt') as a:
    compressedline = a.readline()  # read up to 8 chars from first line
print("Size of compressed file is " + compressedline[19:-32] + " Bytes")

originalsize = float(line[19:-25])
compressedsize = float(compressedline[19:-32])

reduction = ((originalsize-compressedsize)/originalsize)*100

print("Compression ratio : " + str(reduction) + "%")

os.system("mv rawData.csv " + f)
os.system("mv rawData.csv.tar.gz " + f + ".tar.gz")
os.system("rm originalsize.txt compressedsize.txt")
```

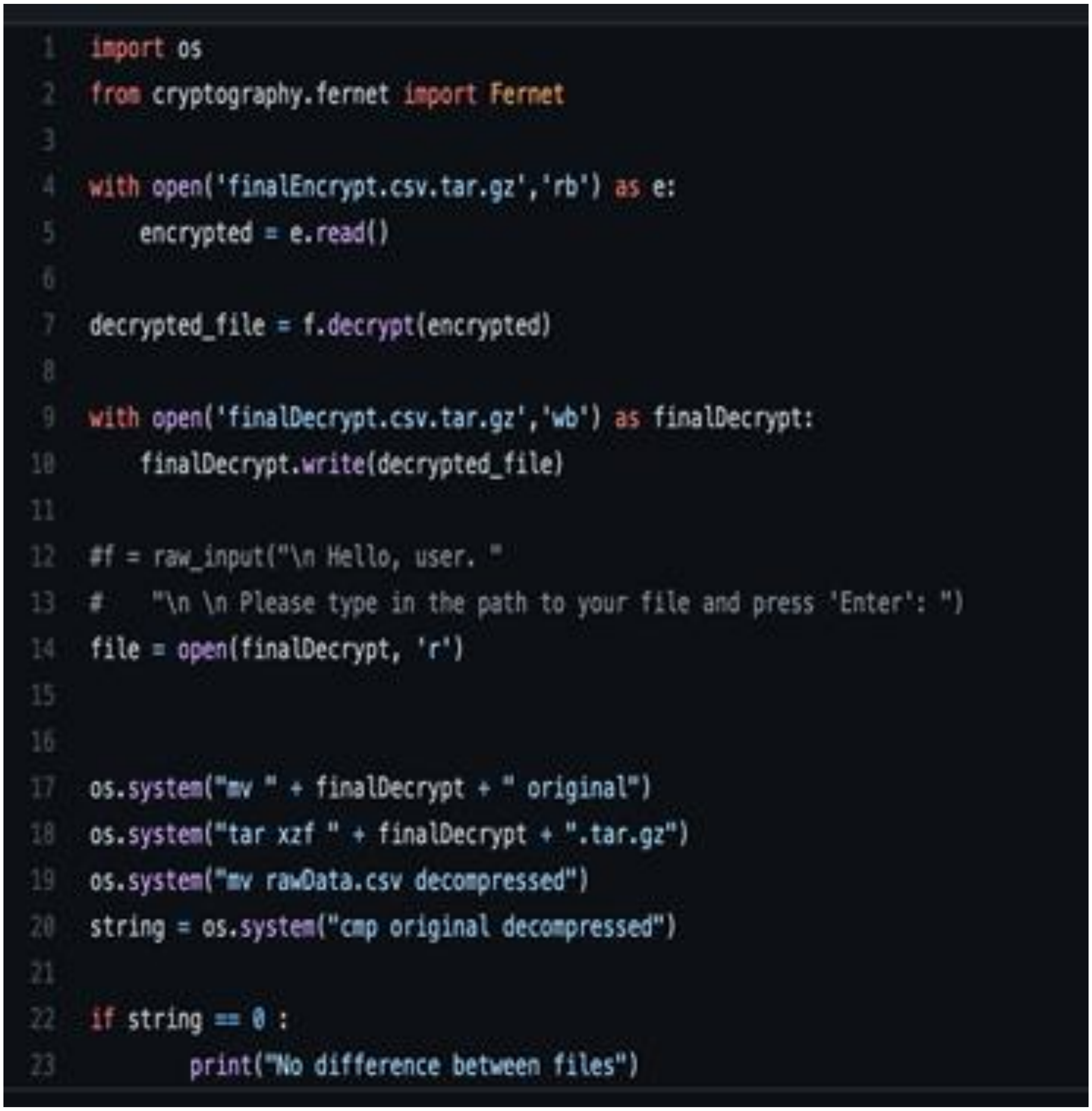
Figure 17. Compression Algorithm



```
#    def key_create:
e_key = Fernet.generate_key()
#        return e_key

#    def key_write(key, key_name):
with open('mykey.e_key','wb') as mykey:
    mykey.write(e_key)

f = Fernet(e_key)

with open('rawData.csv.tar.gz','rb') as zipped_file:
    zipped = zipped_file.read()

encrypted_file = f.encrypt(zipped)

with open('finalEncrypt.csv.tar.gz','wb') as finalEncrypt:
    finalEncrypt.write(encrypted_file)
```

Figure 18. Encryption Algorithm

Decrypt and Decompress:



```python
import os
from cryptography.fernet import Fernet

with open('finalEncrypt.csv.tar.gz','rb') as e:
    encrypted = e.read()

decrypted_file = f.decrypt(encrypted)

with open('finalDecrypt.csv.tar.gz','wb') as finalDecrypt:
    finalDecrypt.write(decrypted_file)

#f = raw_input("\n Hello, user. "
#    "\n \n Please type in the path to your file and press 'Enter': ")
file = open(finalDecrypt, 'r')


os.system("mv " + finalDecrypt + " original")
os.system("tar xzf " + finalDecrypt + ".tar.gz")
os.system("mv rawData.csv decompressed")
string = os.system("cmp original decompressed")

if string == 0 :
        print("No difference between files")
```

Figure 19. Decryption Algorithm

The first step in the DesignDecrypDecomp.py (https://github.com/TheAllyA/EEE3097S-Project/blob/main/DesignDecrypDecomp.py) is to decrypt the file. For the decryption, the encrypted file is taken as input to be read. After being read, the file is decrypted and written to a separate file that will contain the decrypted data items.

The second step is to decompress the decrypted file. For decompression, the program will use the file indicated in the decryption block to be decompressed and then will execute the decompression BASH command to decompress the compressed file. We will then use the 'cmp' command in BASH to compare the decompressed file to the original file, if no output is observed, the files are the same and no data has been lost.

Block 1

Data acquisition:

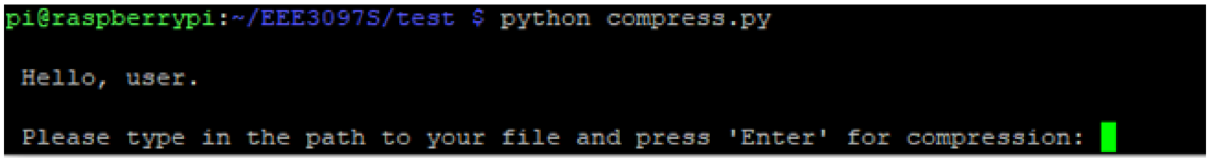
To acquire the data on our pi's, we downloaded the Raw Data on our lessons tab from week 5 and then uploaded it to github where we then cloned the repository throughout pi in order to access the IMU data. Github Link: https://github.com/TheAllyA/EEE3097S-Project

Block 2

Compression:

The compression algorithm was set up using python language. The program imports os in order to interact with the Pi shell which operates on linux.

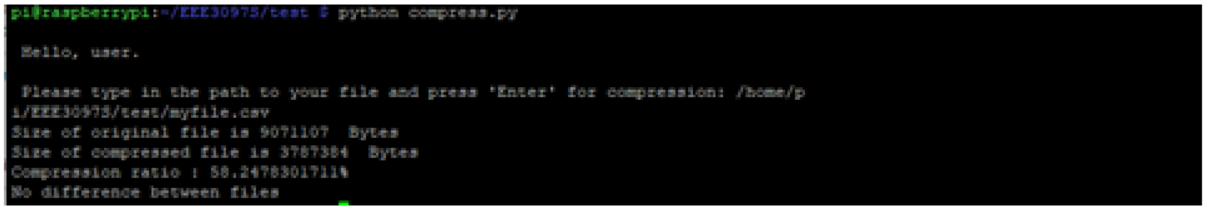
When the file compress.py is run using 'python compress.py', the program prompts the user to enter the path to the file that is indented to be compressed.



```
pi@raspberrypi:~/EEE3097S/test $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: 
```

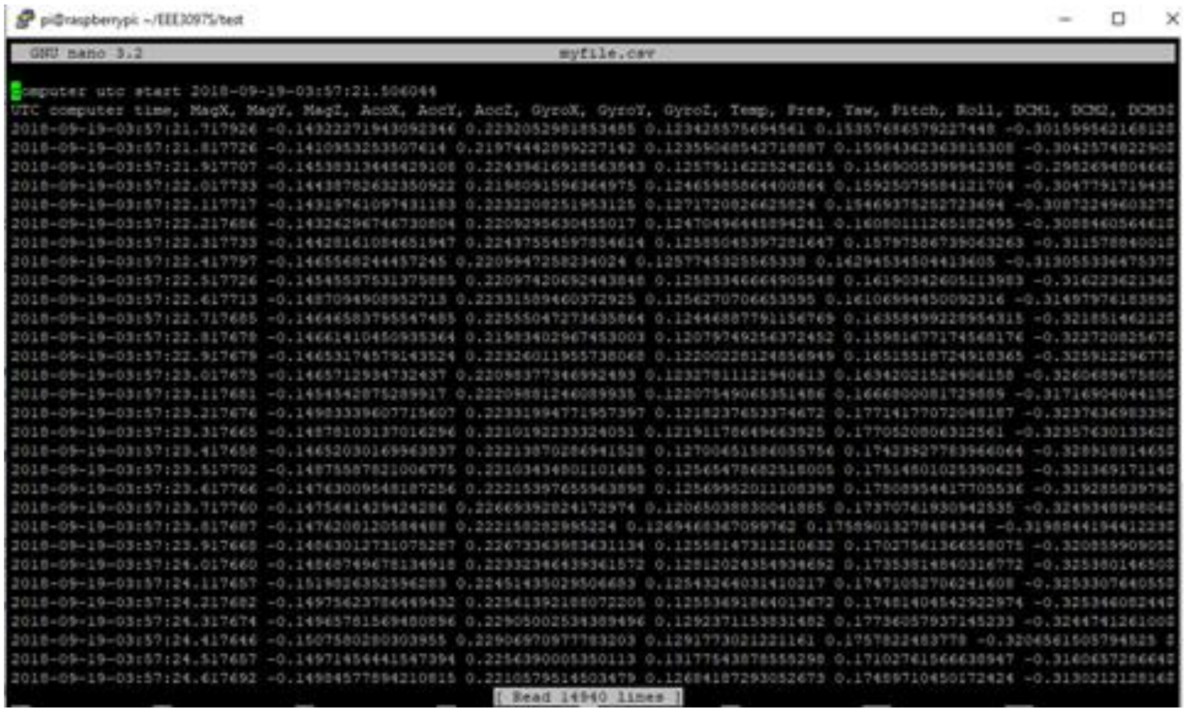
Figure 20. Snapshot of prompt from compression algorithm

In the case below, the file 'myfile.csv' was the file chosen to be compressed



```
pi@raspberrypi:~/EEE3097S/test $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: /home/p
i/EEE3097S/test/myfile.csv
Size of original file is 9071107  Bytes
Size of compressed file is 3787384  Bytes
Compression ratio : 58.24783017111%
No difference between files
```

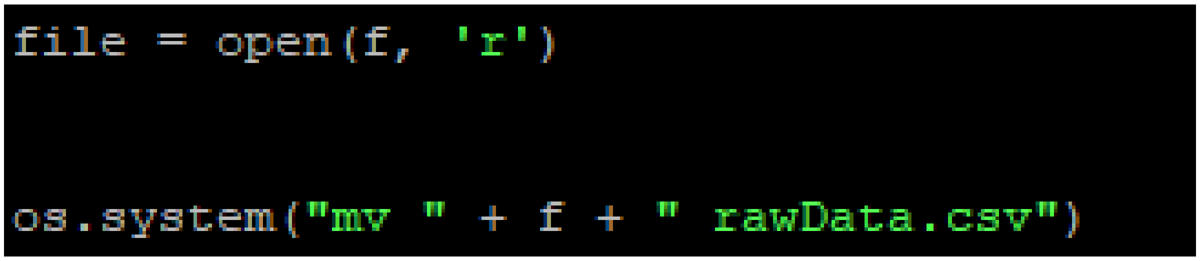
Figure 21. Output from running compression algorithm



```
pi@raspberrypi: ~/EEE3097S/test

  GNU nano 3.2                              myfile.csv

computer utc start 2018-09-19-03:57:21.506044
UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM2, DCM3$
2018-09-19-03:57:21.717926 -0.14322271943092346 0.22320529818534 85 0.12342855756945 61 0.15357686579227448 -0.30159956216812$
2018-09-19-03:57:21.817726 -0.14109532535076 14 0.21974442099227142 0.12355906854271 8087 0.15984362363815308 -0.30425740229 0$
2018-09-19-03:57:21.917707 -0.14538313448429108 0.22439616918563 843 0.1257911622524 2615 0.15690053999 42398 -0.29826948046 6$
2018-09-19-03:57:22.017733 -0.14438782632350922 0.21980915963649 75 0.12465985866440 0864 0.15925079558412 1704 -0.30477917194 3$
2018-09-19-03:57:22.117717 -0.14319761097431183 0.22332082519531 25 0.12717208266258 24 0.15469375252723694 -0.30872249603 27$
2018-09-19-03:57:22.217686 -0.14326296746730804 0.22092956304550 17 0.12470496448894241 0.16080111265182495 -0.30884605646 1$
2018-09-19-03:57:22.317733 -0.14428161084651947 0.22437554597854614 0.12555045397281647 0.15797586739063263 -0.31157884001$
2018-09-19-03:57:22.417797 -0.14655682444572 45 0.22099472582340 24 0.12577453255653 38 0.16294534504413605 -0.31305533647537$
2018-09-19-03:57:22.517726 -0.14545537531375885 0.22097420692443848 0.12583346664905548 0.16190342605113983 -0.31622362136$
2018-09-19-03:57:22.617713 -0.14870949089527 13 0.22331589460037 2925 0.12562707066535 95 0.16106994450092316 -0.31497976183 89$
2018-09-19-03:57:22.717685 -0.14646583795547485 0.22555047273635864 0.12444688779115 6769 0.16355499228954315 -0.32185146212$
2018-09-19-03:57:22.817678 -0.14661410450935364 0.21983402967453003 0.12079749256372452 0.15981677174568176 -0.32272082567$
2018-09-19-03:57:22.917679 -0.14653174579143524 0.22326011955738068 0.12200228124856949 0.16515518724918365 -0.32591229677$
2018-09-19-03:57:23.017675 -0.14657129347324 37 0.22095377346992493 0.12327811121940613 0.16342021524906158 -0.32606896758$
2018-09-19-03:57:23.117681 -0.14545420875289917 0.22209881246089935 0.12207549065351486 0.16668000081729889 -0.31716904044 15$
2018-09-19-03:57:23.217676 -0.14983339607715607 0.22331994771957397 0.12182376533746 72 0.17714177072048187 -0.32376369833 9$
2018-09-19-03:57:23.317665 -0.14878103137016296 0.22101922333240 51 0.12191178649663925 0.17705208066312561 -0.32357630133 62$
2018-09-19-03:57:23.417658 -0.14652030169963837 0.22213870286941528 0.12700465158405 5756 0.17423927783966064 -0.32891881465$
2018-09-19-03:57:23.517702 -0.14875587821006775 0.22103434801101685 0.12565478682518005 0.17514801025390625 -0.32136917114$
2018-09-19-03:57:23.617766 -0.14763009548187256 0.22215397655963898 0.12569952011108398 0.17808954417705536 -0.31928583979$
2018-09-19-03:57:23.717760 -0.14756441429424286 0.22669392024172974 0.12065038830041885 0.17370761930942535 -0.32493489980 6$
2018-09-19-03:57:23.817687 -0.14762001205444 88 0.22215828295224 0.12694668367099762 0.17589013278484344 -0.31988441944 1223$
2018-09-19-03:57:23.917668 -0.14863012731075287 0.22673363983631134 0.12558147311210632 0.17027561366558075 -0.32085590905$
2018-09-19-03:57:24.017660 -0.14860749678134918 0.22332346439361572 0.12812024354934692 0.17353814840316772 -0.32538014650$
2018-09-19-03:57:24.117657 -0.15190263525562 83 0.22451435029506683 0.12543264031410217 0.17471052706241608 -0.32533076405 5$
2018-09-19-03:57:24.217682 -0.14975623786449432 0.22561392188072205 0.12553691864013672 0.17481404542922974 -0.32534608244$
2018-09-19-03:57:24.317674 -0.14965781569480896 0.22905002534389496 0.12923711533831482 0.17736057937145233 -0.32447412610 0$
2018-09-19-03:57:24.417646 -0.15075802803039 55 0.22906970977783203 0.12917730212221161 0.1757822483778 -0.32065615057945 25 $
2018-09-19-03:57:24.517657 -0.14971456441547394 0.22563900053501 13 0.13177543878555298 0.17102761566638947 -0.31606572866 4$
2018-09-19-03:57:24.617692 -0.14984577594210815 0.22105795145034 79 0.12684187293052673 0.17489710450172424 -0.31302121281 6$
                                                  [ Read 14940 lines ]
```

Figure 22. Output from raw Data

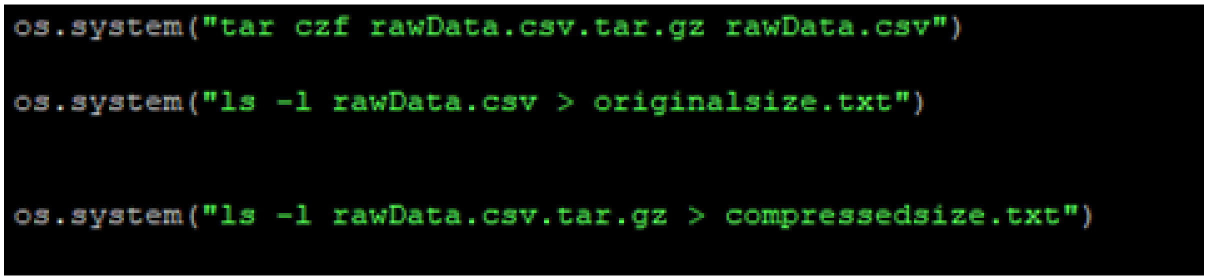
The program 'compress.py' opens the file 'myfile.csv' and then renames it to a temporary file called rawData.csv,



```
file = open(f, 'r')


os.system("mv " + f + " rawData.csv")
```

Figure 22. Snapshot of compression algorithm

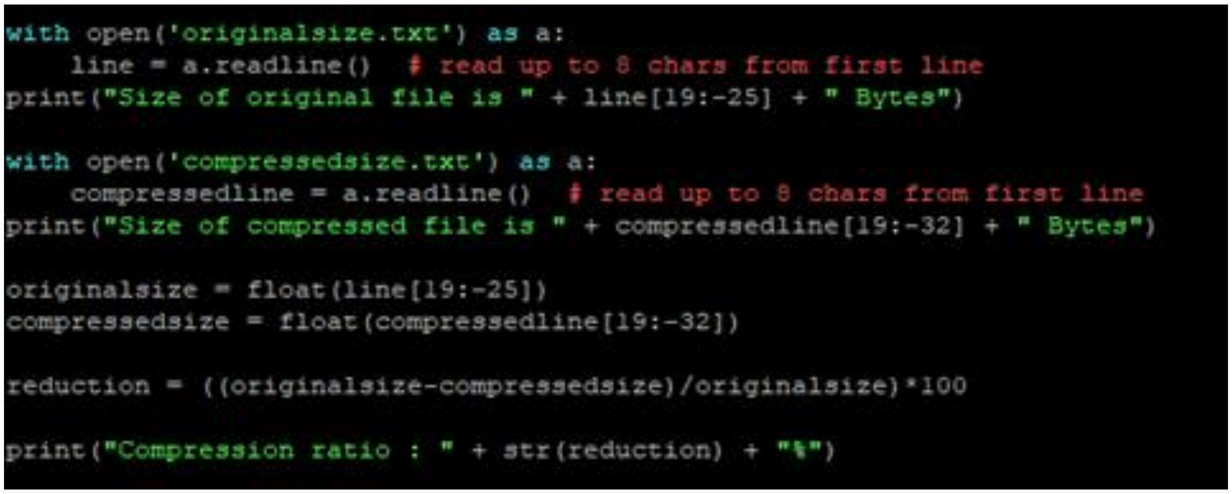
Thereafter it runs the bash command that initiates compression, after compression it produces a compressed file called rawData.csv.tar.gz. Running the BASH command 'ls -l' produces the information about a file which includes its file size, using the command ">" we can send that to a text file, we do this on rawData.csv and rawData.csv.tar.gz, this will allow us to access information about the original and compressed file.



```
os.system("tar czf rawData.csv.tar.gz rawData.csv")

os.system("ls -l rawData.csv > originalsize.txt")


os.system("ls -l rawData.csv.tar.gz > compressedsize.txt")
```

Figure 23. Snapshot of compression algorithm

Thereafter the program opens the text files that contains the file size of rawData.csv which is the original file and using string slicing, we can access the value of the file size and output it to the user. We do the same with the compressed file size text file. After that we convert the file sizes to floats and perform a mathematical operation to produce a percentage that shows how much the original file has been reduced by.

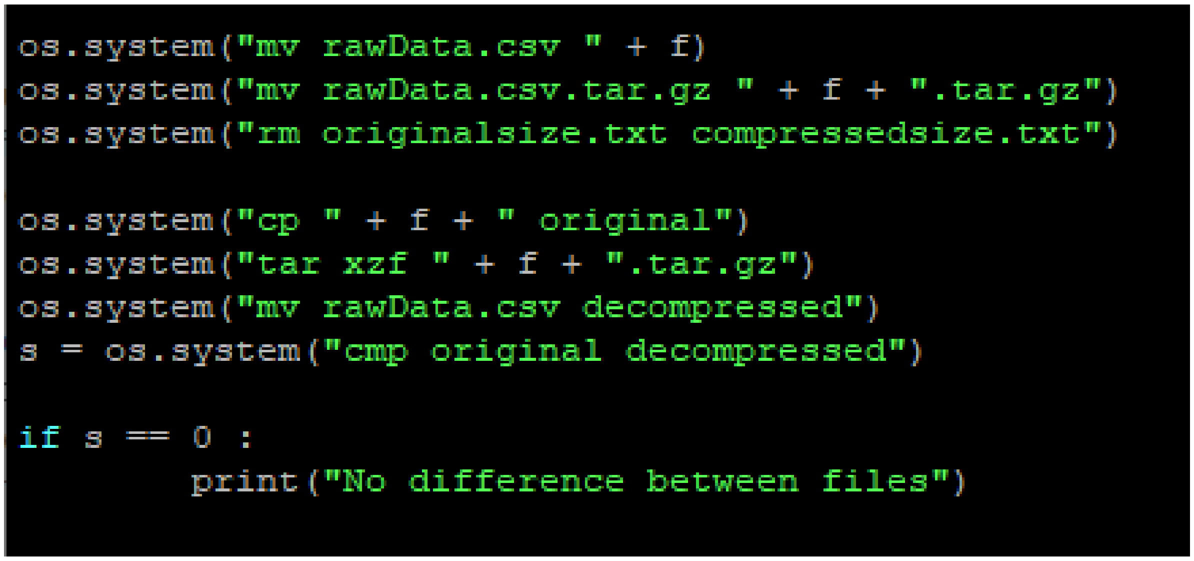


```
with open('originalsize.txt') as a:
    line = a.readline()  # read up to 8 chars from first line
print("Size of original file is " + line[19:-25] + " Bytes")

with open('compressedsize.txt') as a:
    compressedline = a.readline()  # read up to 8 chars from first line
print("Size of compressed file is " + compressedline[19:-32] + " Bytes")

originalsize = float(line[19:-25])
compressedsize = float(compressedline[19:-32])

reduction = ((originalsize-compressedsize)/originalsize)*100

print("Compression ratio : " + str(reduction) + "%")
```

Figure 24. Snapshot of compression algorithm

We then rename the files to the original file names that the user inputted and remove the text files that contain the sizes of the compressed and original files.

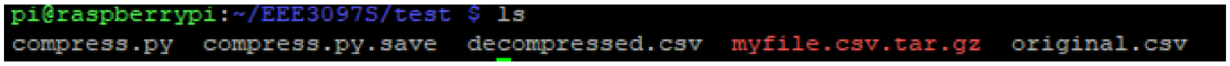
We then make a copy of the original file before compression and name the file original. We then run a decompression BASH command and produce a decompressed file and rename that decompressed.
Using the ‘cmp’ command we compare the original and decompressed file and if no differences are found between the files the program outputs ‘No difference between files’



```
os.system("mv rawData.csv " + f)
os.system("mv rawData.csv.tar.gz " + f + ".tar.gz")
os.system("rm originalsize.txt compressedsize.txt")

os.system("cp " + f + " original")
os.system("tar xzf " + f + ".tar.gz")
os.system("mv rawData.csv decompressed")
s = os.system("cmp original decompressed")

if s == 0 :
        print("No difference between files")
```

Figure 25. Snapshot of compression algorithm

By using the BASH command ls, we can see the files in the local directory, the files contained are original.csv and decompressed.csv. These two files represent the file before compression and the file after compression.



```
pi@raspberrypi:~/EEE3097S/test $ ls
compress.py  compress.py.save  decompressed.csv  myfile.csv.tar.gz  original.csv
```

Figure 26. Snapshot of files in the directory after running compression algorithm

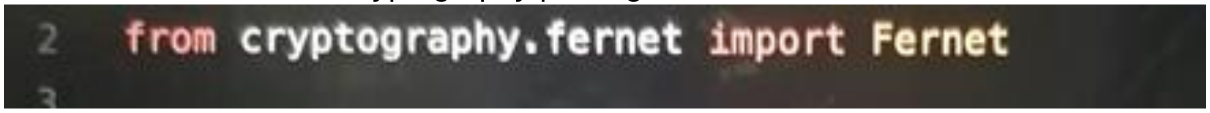
Using the ‘cmp’ BASH command we can compare the contents of original.csv with decompressed.cdv, doing this results in an empty output, illustrating that no difference between the data and that original.csv is exactly the same as decompressed.csv.
This ‘cmp’ command is already included in the compress.py program and outputs the sentence “No difference between files” if ‘cmp’ returns an empty output. This proves that the experiment is successful and the ATP has been met in the case where decompressed data is the same as compressed data and no data has been lost in the process.
The compressed file was then uploaded to github in order to go through the encryption experiment.

Block 3

Encryption:

The encryption program was set up by using Python3 as the programming language of choice. The encryption program makes use of a cryptography module which is only available and effective using Python3. The installation of cryptography also required the

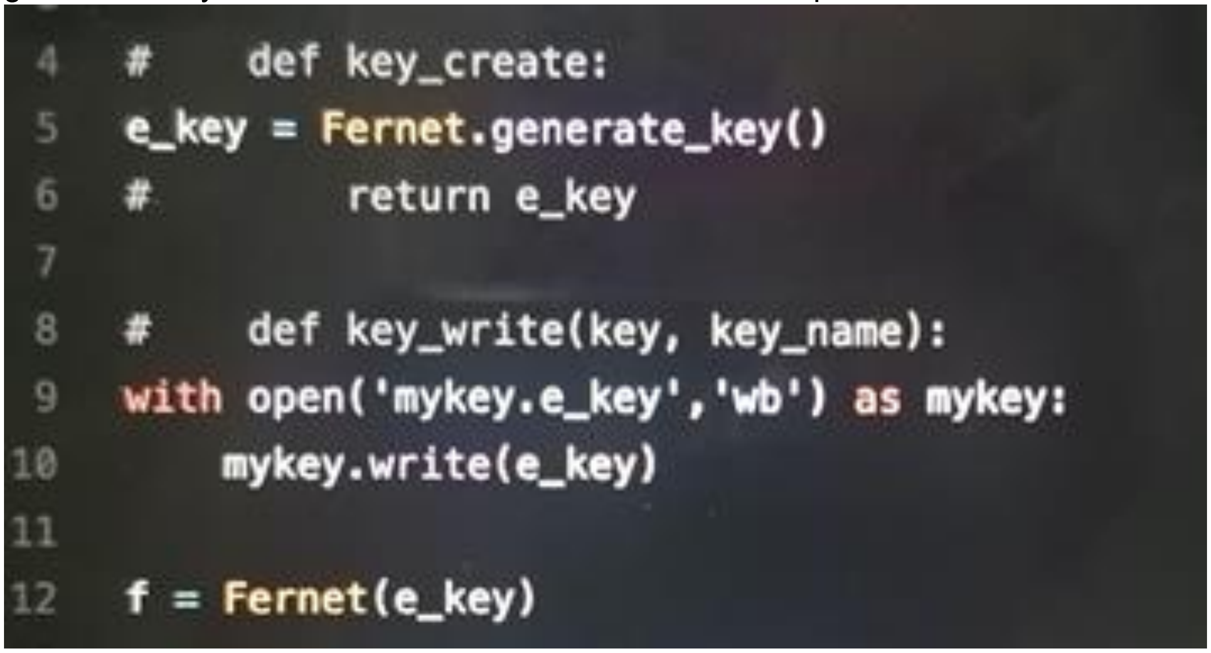
“pip” command to be used thus in total the encryption program required an additional installation of python3, pip.py and cryptography.
There was one program created, namely, encrypted.py (encrypted) that imports the Fernet module from the cryptography package.



```
from cryptography.fernet import Fernet
```

Figure 27.

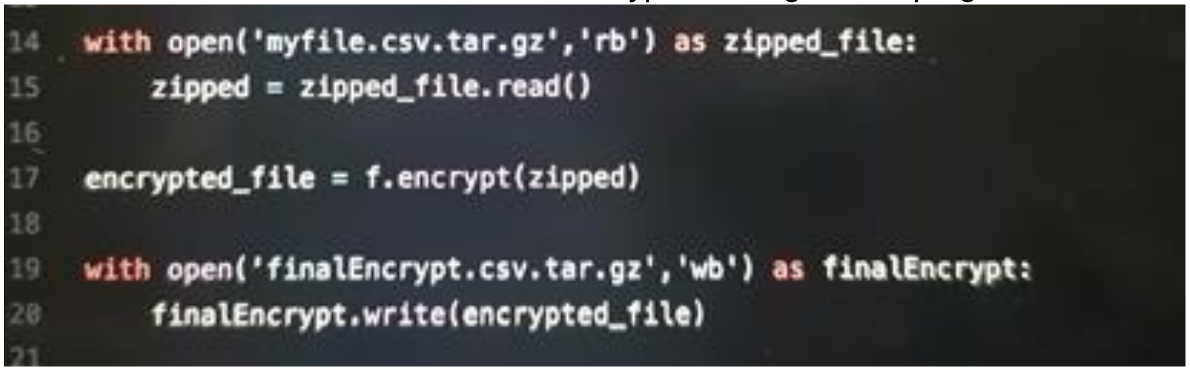
This module contains all the methods and functions needed for the encryption and decryption, namely, generate(), encryt and decrypt. The generate() method is used to generate a key that will be used to “lock” and “unlock” a specified file.



```
#    def key_create:
e_key = Fernet.generate_key()
#        return e_key

#    def key_write(key, key_name):
with open('mykey.e_key','wb') as mykey:
    mykey.write(e_key)

f = Fernet(e_key)
```

Figure 28.

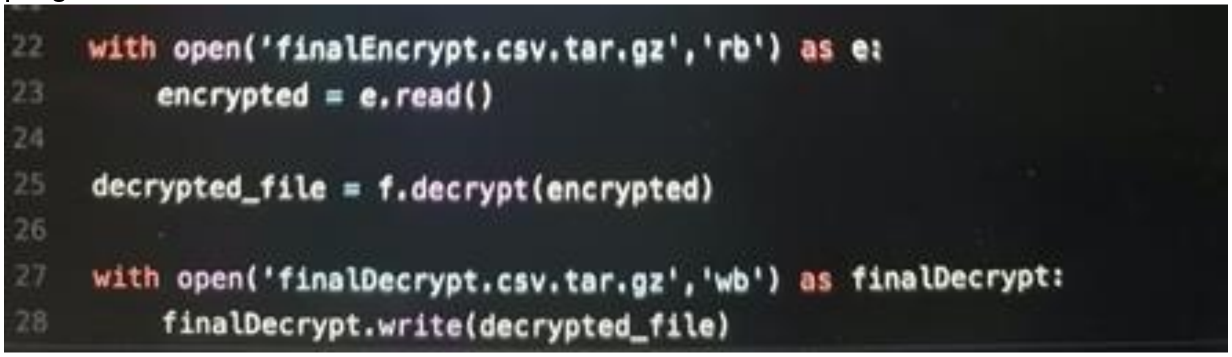
The encrypt function is then used to read the file, encrypt it and write the encrypted data to a new file. This new file is called finalEncrypt.csv.tar.gz in the program.



```
with open('myfile.csv.tar.gz','rb') as zipped_file:
    zipped = zipped_file.read()

encrypted_file = f.encrypt(zipped)

with open('finalEncrypt.csv.tar.gz','wb') as finalEncrypt:
    finalEncrypt.write(encrypted_file)
```

Figure 29.

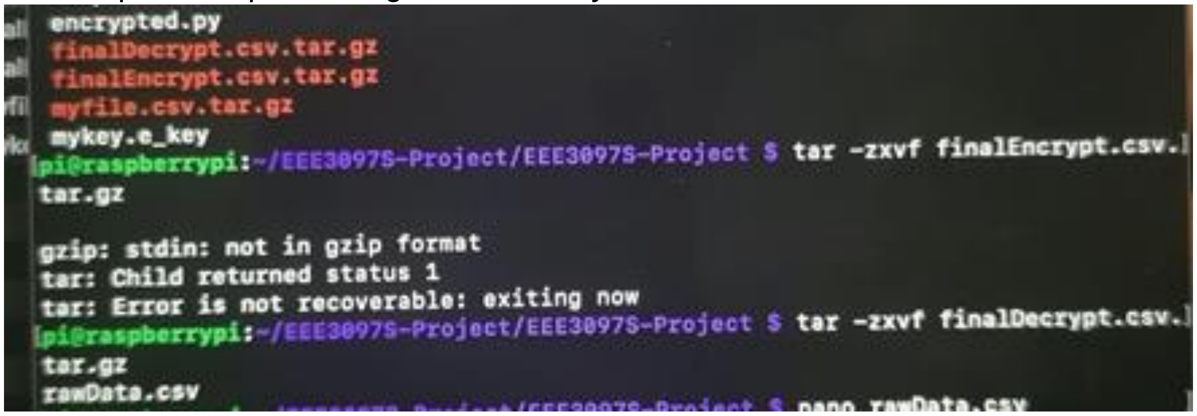
The decrypt function is used to read the encrypted file, decrypt it and write the decrypted data to a new file. This new file is called finalDecrypt.csv.tar.gz in the program.



```
with open('finalEncrypt.csv.tar.gz','rb') as e:
    encrypted = e.read()

decrypted_file = f.decrypt(encrypted)

with open('finalDecrypt.csv.tar.gz','wb') as finalDecrypt:
    finalDecrypt.write(decrypted_file)
```

Figure 30.

The file that was used in the program to be encrypted and decrypted was obtained from the compression processing block, namely, the rawData.csv file.



```
encrypted.py
finalDecrypt.csv.tar.gz
finalEncrypt.csv.tar.gz
myfile.csv.tar.gz
mykey.e_key
pi@raspberrypi:~/EEE3097S-Project/EEE3097S-Project $ tar -zxvf finalEncrypt.csv.
tar.gz

gzip: stdin: not in gzip format
tar: Child returned status 1
tar: Error is not recoverable: exiting now
pi@raspberrypi:~/EEE3097S-Project/EEE3097S-Project $ tar -zxvf finalDecrypt.csv.
tar.gz
rawData.csv
                  /EEE3097S-Project/EEE3097S-Project $ nano rawData.csv
```

Figure 31.

(worked on by ADMALI004)

Block 4

Data integrity:

The data integrity program was set up using Python3 as the programming language of choice. The data integrity program makes use of the csv module that had to be imported.

There was one program created, namely, dInteg.py that makes use of all the methods and functions that are available in the csv module. The open() method is first used to open up the file that contains the data that was extracted from the IMU. The reader object is then used to read the file and the readlines() method is used to read each line within the file. The data items within each line gets further extracted and checked to ensure that it is a float point value and that falls within the lowest 25% of Fourier coefficients. If both checks return true then the program does not get flagged. However,

if one of the checks returns false then an error message shows and the program stops. This process occurs within a for loop to ensure that each data item is checked.
The program only sends the file through to get compressed if both checks return true for each data item.
(worked on by ADMALI004)

4.3. Results

Overall:
The results for the overall experiment were successful, when testing the original file with the decompressed file, no differences between the data was observed on the receiver end which indicates that the file the Pi sent to the laptop was the original data and this data was compressed and decompressed successfully. The compressed file was then encrypted and could not be decompressed which proves that the file cannot be used without the key. The encrypted compressed file was alternatively decrypted and decompressed which produced the rawData.csv file. This was the file that was used at the beginning of the program thus since it is being produced at the end of the program, it proves that the program is successful.

Data acquisition:
Acquiring the data was successful, through the use of git on the pi and github on the laptop, we are able to send Raw Data to the pi to test our algorithms on.

Compression:
The compression experiment proved to be successful in generating a file that is a compressed version of the original file. In the experiment it was found that the compressed file was reduced by 58% compared to the original file. This is particularly important to the experiment as we are trying to reduce the file size as much as possible. After decompressing the compressed file, a test was run to ensure that no data was lost in the process. This was shown to be successful as well, the test was run on all 9 csv files provided by the lessons tab. No differences were found in all the files and a 60% average compression rate was found for all 9 csv files. A test was run in the demo with our tutor and met the requirements that were necessary in this experiment. Seeing that our tutor approved the test, we can conclude that the compression block is successful.

Encryption:
The encryption was successfully executed and produced the expected results. The data items that were generated in the file were encrypted and thus could not be understood. The compressed file was encrypted and when trying to decompress this file it gave an error. This was expected as it is an encrypted file thus is cannot be opened without the generated key. The compressed file was then decrypted and decompressed and it successfully did so. The decrypted and decompressed file was the rawData.csv file which is expected since that is the file that was used at the beginning of the program. This proves that the encryption program was executed successfully.

Data integrity:
The data integrity was partly successful in execution. Each data item was checked to ensure that it met both the requirements, namely, it is a float point value and it is within

the range of the lowest 25% Fourier coefficients. The float point value check was executed successfully, but the check for the Fourier coefficients was unsuccessful and thus it still needs to be improved.

Try to check the change in the performance when you change the data:

Changing the data only affects the compression part of the experiment, not the encryption. When reducing the size of the file it was noted that the percentage of compression reduced aswell, on the other hand, the time taken to compress the file increased remarkably with an almost instant result when the file was 7 kB whereas it took 13 seconds to compress the original file with a size of 9 MB. The graph below shows that for large files, (1 – 9 MB), the compression percentage stays relatively constant at 58%, and the time taken to compress is approximately a few seconds, with 13 seconds as a maximum for a 9MB file and instantaneous for a small file of a couple of kilobytes.

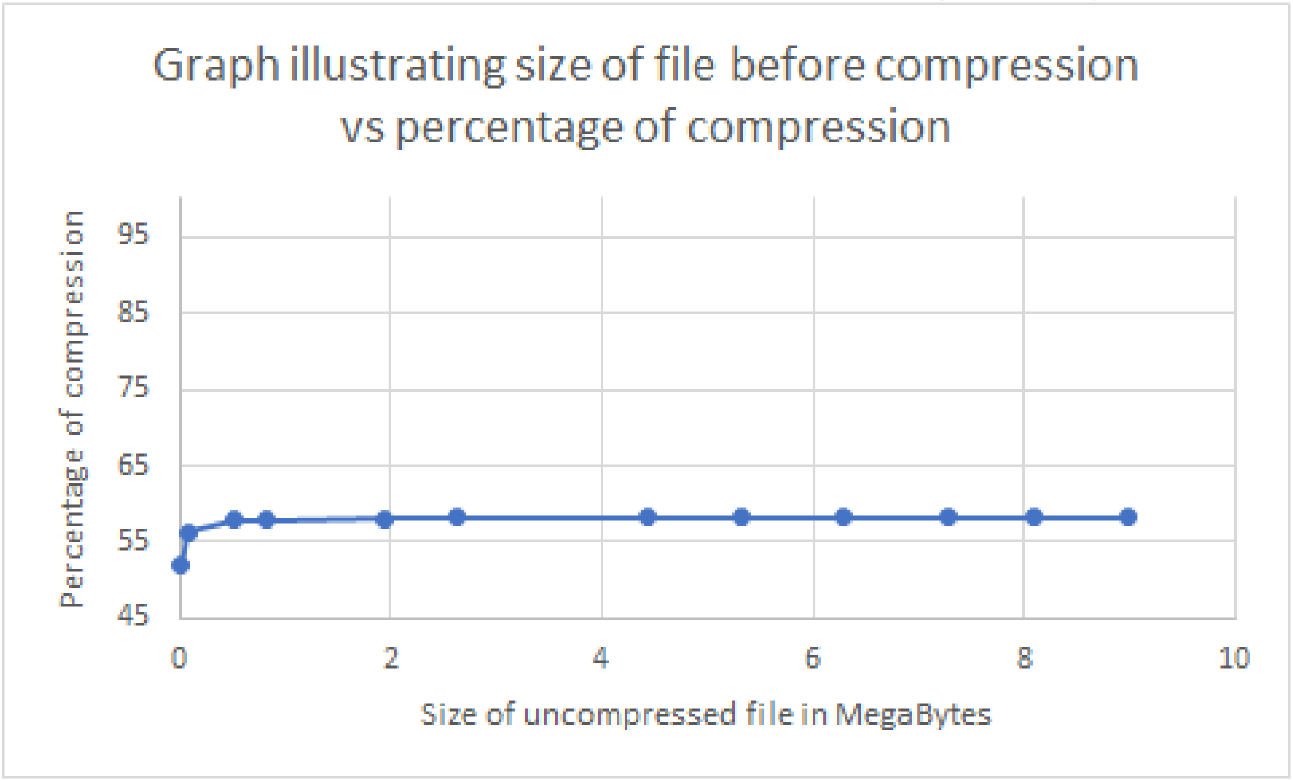


Figure 33. Graph illustrating size differences in compression

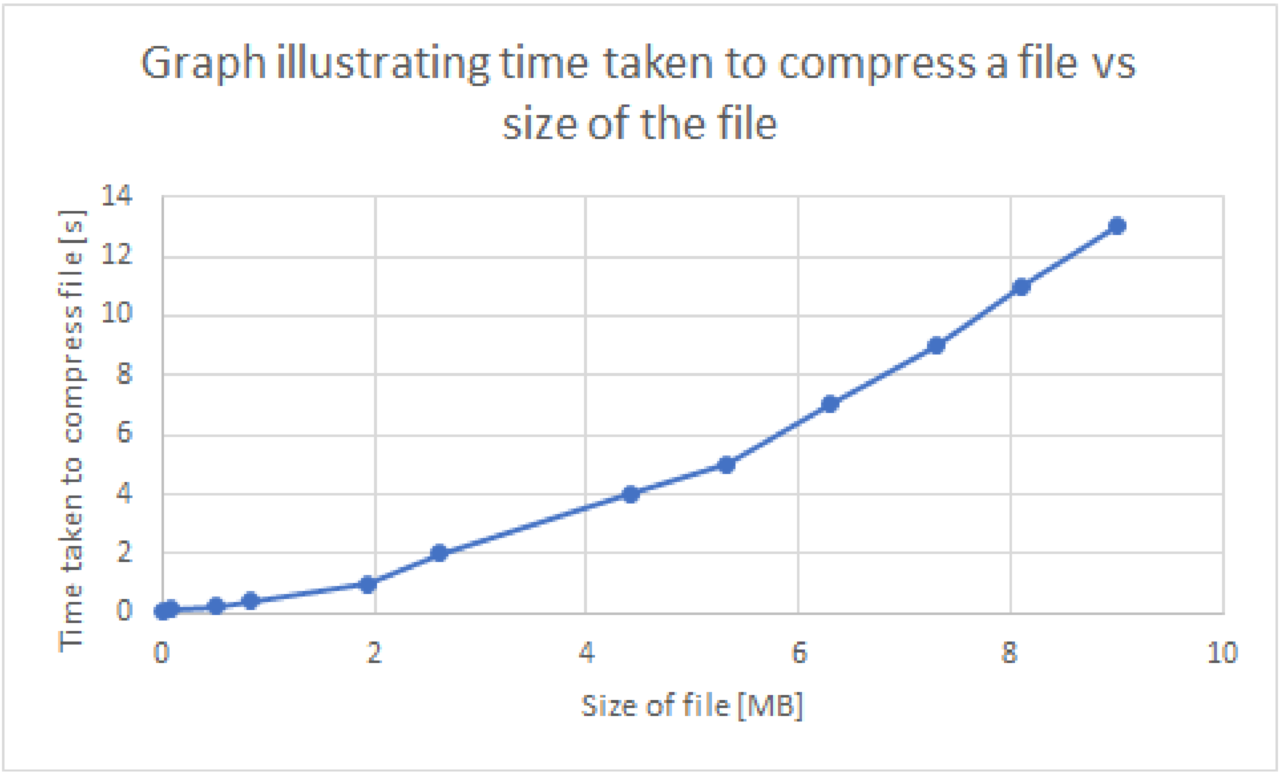


Figure 34. Graph illustrating time differences in compression

5. Validation using a different-IMU

5.1. IMU Module

Discussion of Salient features and how they differ:

	SenseHAT ICM-20948	SHARC ICM-20649
APPLICATIONS	• Smartphones and Tablets • Wearable Sensors • IoT Applications	• Sports • Wearable Sensors • High Impact Applications
FEATURES	• 3-Axis Gyroscope with Programmable FSR of ±250 dps, ±500 dps, ±1000 dps, and ±2000 dps • 3-Axis Accelerometer with Programmable FSR of ±2g, ±4g, ±8g, and ±16g • On-Chip 16-bit ADCs and Programmable Filters • 7 MHz SPI or 400 kHz Fast Mode I²C • Digital-output temperature sensor • VDD operating range of 1.71V to 3.6V • MEMS structure hermetically sealed and bonded at wafer level • RoHS and Green compliant • Onboard Digital Motion Processor (DMP) • Lowest Power 9-Axis Device at 2.5 mW • 3-Axis Compass with a wide range to ±4900 µT • Android support • Auxiliary I2 C interface for external sensors	• 3-Axis gyroscope with programmable FSR of ±500 dps, ±100 dps, ±2000 dps, and ±4000 dps • 3-Axis accelerometer with programmable FSR of ±4g, ±8g, ±16g, and **±30g** • On-Chip 16-bit ADCs and Programmable Filters • Host interface: 7 MHz SPI or 400 kHz I2C • Digital-output temperature sensor • VDD operating range of 1.71V to 3.6V • MEMS structure hermetically sealed and bonded at wafer level • RoHS and Green compliant • DMP Enabled: - SMD, Step Count, Step Detect, Activity Classifier, RV, GRV - Calibration of accel/gyro/compass • User-programmable interrupts • Wake-on-motion interrupt for low power operation of applications processor • 4kB FIFO buffer enables the applications processor to read the data in bursts

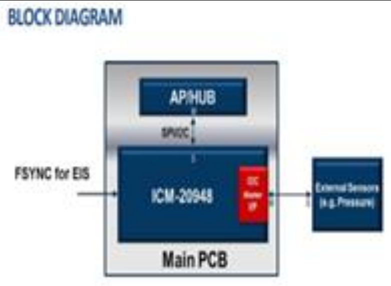
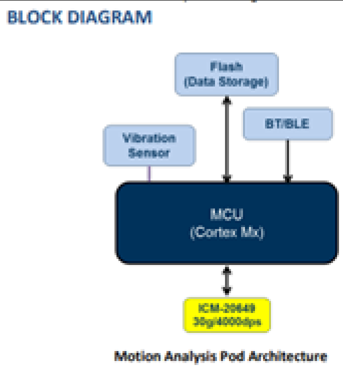
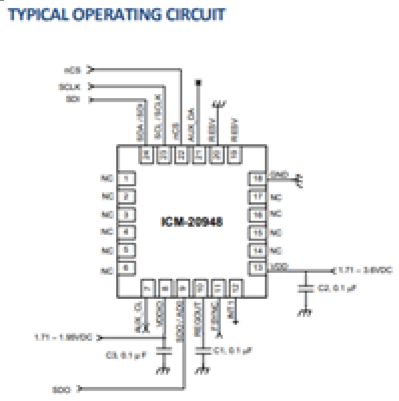
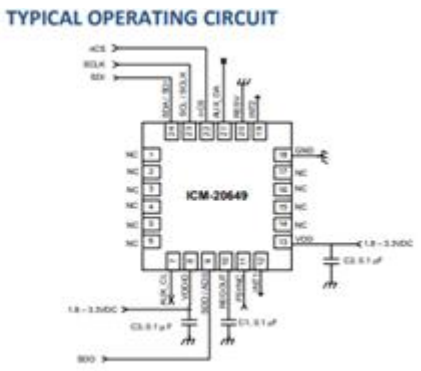
Block Diagram	 BLOCK DIAGRAM AP/HUB SPI/I2C FSYNC for EIS ICM-20948 External Sensors (e.g. Pressure) Main PCB	 BLOCK DIAGRAM Flash (Data Storage) Vibration Sensor BT/BLE MCU (Cortex Mx) ICM-20649 30g/4000dps Motion Analysis Pod Architecture
Typical Operating Circuit		

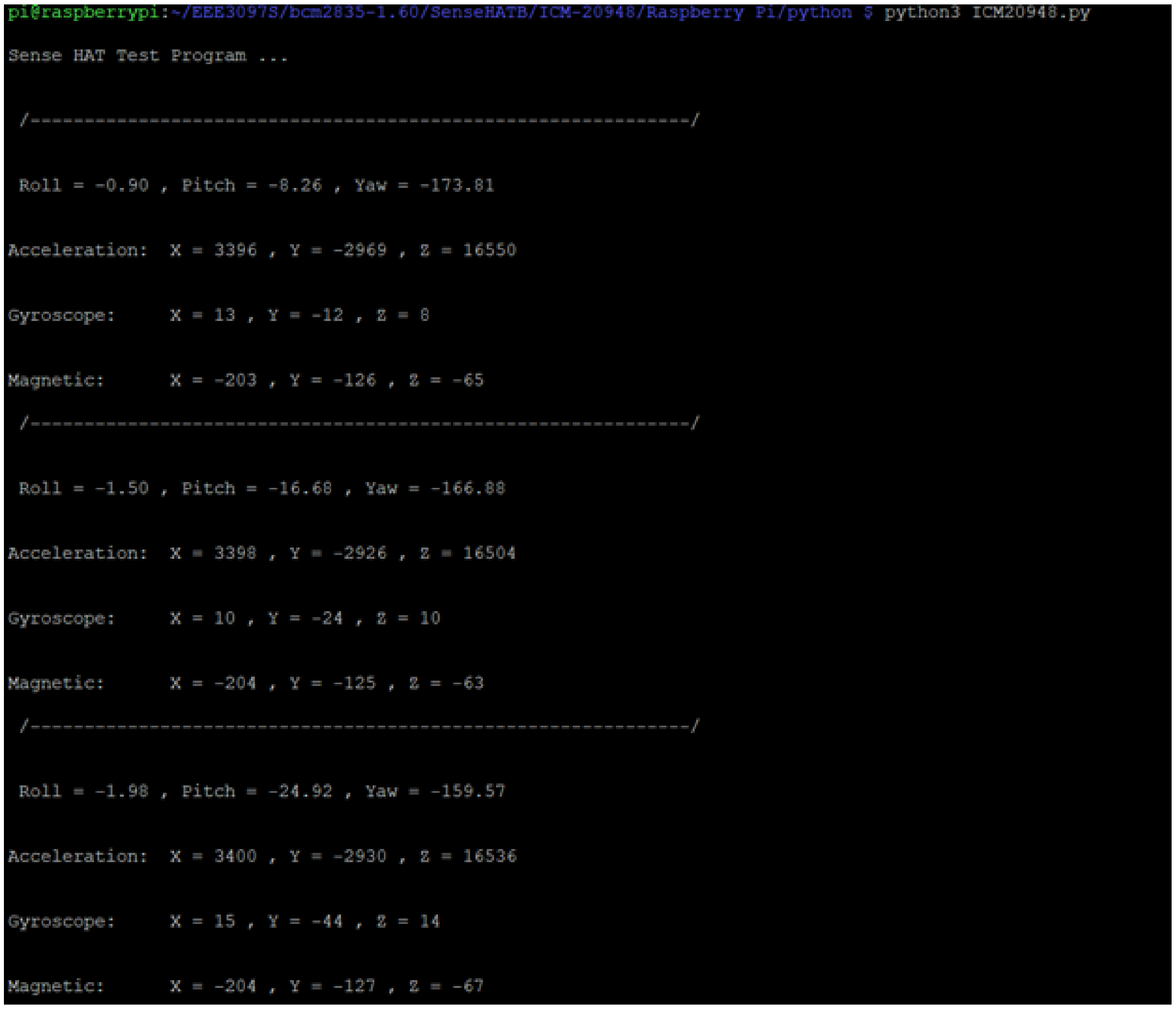
Table 30. Comparison of SenseHAT IMU and Buoy IMU

From the table above we can see the comparison between the ICMs used, one on the buoy (SHARC ICM-20649) and one on our Raspberry Pi using a SenseHAT(ICM-20948)

Applications for the SenseHAT are different to that of the actual Buoy with one shared application in wearable sensors. There are a lot of features that both ICM's share, these are in black text in the table above whereas the differences are indicated in red text. The Buoy ICM's Accelerometer can handle a maximum of ±30g whereas the maximum for the SenseHAT we have is only ±16g. The block diagram and typical operating circuit is shown for both Modules and show the slight differences between the two modules.

5. 2. Steps you plan to take to make sure that your testing with this IMU can be extrapolated easily to the IMU to be used in the actual buoy

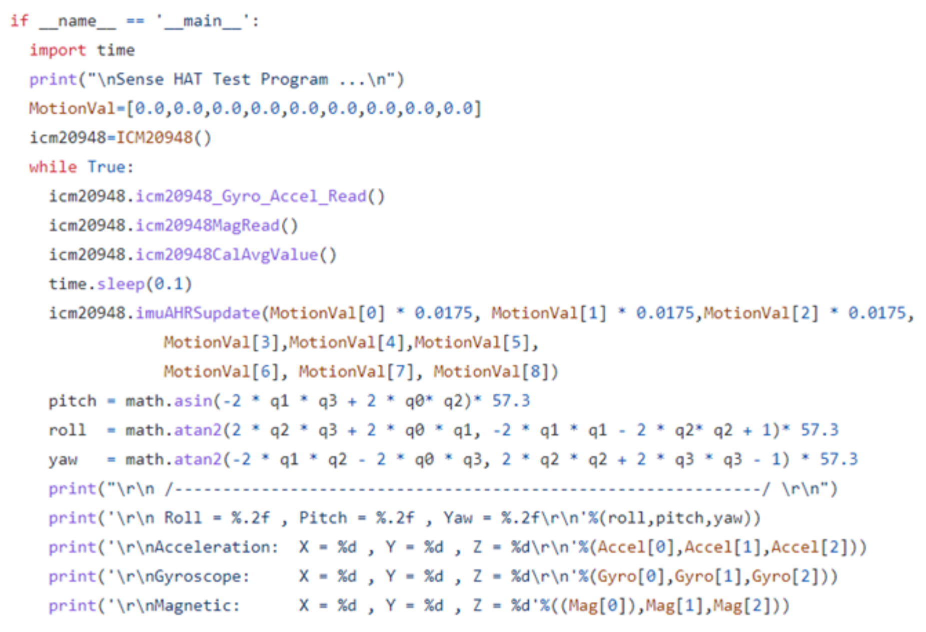
The python file from Waveshare to read data from the sensors produce the following output:



```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python3 ICM20948.py

Sense HAT Test Program ...


 /-------------------------------------------------------------/


 Roll = -0.90 , Pitch = -8.26 , Yaw = -173.81


Acceleration:  X = 3396 , Y = -2969 , Z = 16550


Gyroscope:     X = 13 , Y = -12 , Z = 8


Magnetic:      X = -203 , Y = -126 , Z = -65
 /-------------------------------------------------------------/


 Roll = -1.50 , Pitch = -16.68 , Yaw = -166.88


Acceleration:  X = 3398 , Y = -2926 , Z = 16504


Gyroscope:     X = 10 , Y = -24 , Z = 10


Magnetic:      X = -204 , Y = -125 , Z = -63
 /-------------------------------------------------------------/


 Roll = -1.98 , Pitch = -24.92 , Yaw = -159.57


Acceleration:  X = 3400 , Y = -2930 , Z = 16536


Gyroscope:     X = 15 , Y = -44 , Z = 14


Magnetic:      X = -204 , Y = -127 , Z = -67
```

Figure 35. Snapshot of output from ICM python file

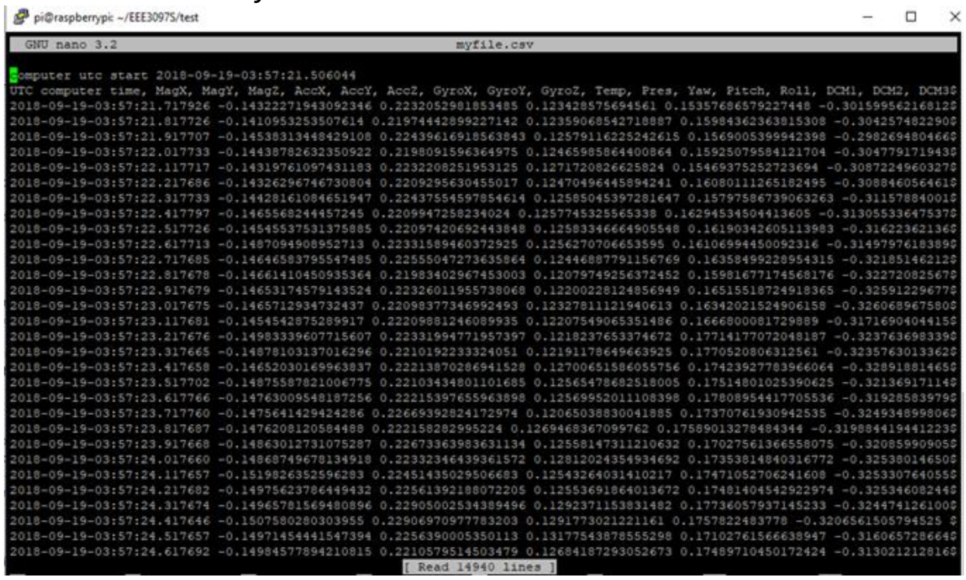
The output above is gathered from the following code snippet from the python file given by Waveshare:



```
if __name__ == '__main__':
  import time
  print("\nSense HAT Test Program ...\n")
  MotionVal=[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
  icm20948=ICM20948()
  while True:
    icm20948.icm20948_Gyro_Accel_Read()
    icm20948.icm20948MagRead()
    icm20948.icm20948CalAvgValue()
    time.sleep(0.1)
    icm20948.imuAHRSupdate(MotionVal[0] * 0.0175, MotionVal[1] * 0.0175,MotionVal[2] * 0.0175,
                MotionVal[3],MotionVal[4],MotionVal[5],
                MotionVal[6], MotionVal[7], MotionVal[8])
    pitch = math.asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3
    roll  = math.atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3
    yaw   = math.atan2(-2 * q1 * q2 - 2 * q0 * q3, 2 * q2 * q2 + 2 * q3 * q3 - 1) * 57.3
    print("\r\n /-------------------------------------------------------------/ \r\n")
    print('\r\n Roll = %.2f , Pitch = %.2f , Yaw = %.2f\r\n'%(roll,pitch,yaw))
    print('\r\nAcceleration:  X = %d , Y = %d , Z = %d\r\n'%(Accel[0],Accel[1],Accel[2]))
    print('\r\nGyroscope:     X = %d , Y = %d , Z = %d\r\n'%(Gyro[0],Gyro[1],Gyro[2]))
    print('\r\nMagnetic:      X = %d , Y = %d , Z = %d'%((Mag[0]),Mag[1],Mag[2]))
```

Figure 36. Snapshot of ICM python file

The RawData given by the course's lessons tab produces the following output which is formatted in such a way that it is easy to analyze on excel when the data is transmitted to a user. This format is the same format that would be outputted from the IMU module on the actual Buoy.

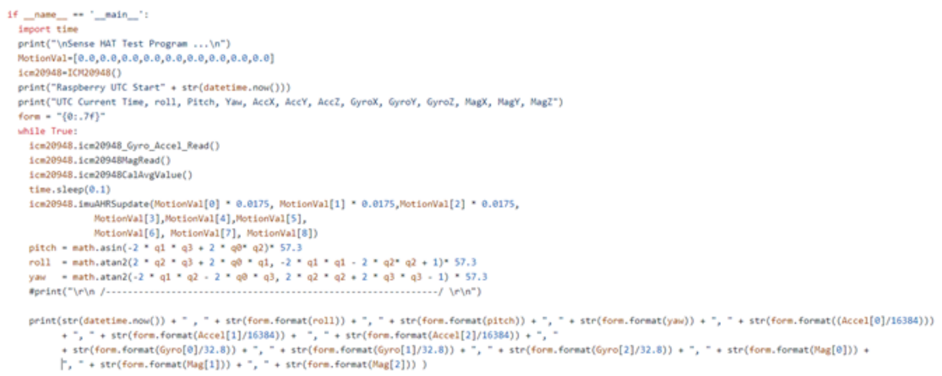


```
pi@raspberrypi: ~/EEE3097S/test
  GNU nano 3.2                          myfile.csv

computer utc start 2018-09-19-03:57:21.506044
UTC computer time, MagX, MagY, MagZ, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, Temp, Pres, Yaw, Pitch, Roll, DCM1, DCM2, DCM3$
2018-09-19-03:57:21.717926 -0.14322271943092346 0.2232052981853485 0.123428575694561 0.15357686579227448 -0.30159956216812$
2018-09-19-03:57:21.817726 -0.1410953253507614 0.21974442899227142 0.12359068542718887 0.15984362363815308 -0.304257482290$
2018-09-19-03:57:21.917707 -0.14538313448429108 0.22439616918563843 0.12579116225242615 0.15690053999423398 -0.298269480466$
2018-09-19-03:57:22.017733 -0.14438782632350922 0.21980915963649755 0.12465985864400864 0.15925079584121704 -0.304779171943$
2018-09-19-03:57:22.117717 -0.14319761097431183 0.22322082519531125 0.12717208266258824 0.15469375252723694 -0.30872249603278$
2018-09-19-03:57:22.217686 -0.14326296746730804 0.22092956304550017 0.12470496445894241 0.16080111265182495 -0.308846056461$
2018-09-19-03:57:22.317733 -0.14428161084651947 0.22437554597854614 0.12585045397281647 0.15797586739063263 -0.311578840015$
2018-09-19-03:57:22.417797 -0.1465568244457245 0.22099472582340024 0.12577453255653338 0.16294534504413605 -0.31305533647537$
2018-09-19-03:57:22.517726 -0.14545537531375885 0.22097420692443848 0.12583346664905548 0.16190342605113983 -0.31622362136$
2018-09-19-03:57:22.617713 -0.1487094908952713 0.22331589460372925 0.12562707066535595 0.16106994450092316 -0.31497976183898$
2018-09-19-03:57:22.717685 -0.14646583795547485 0.22555047273635864 0.12446887791156769 0.16358499228954315 -0.32185146212$
2018-09-19-03:57:22.817678 -0.14661410450935364 0.21983402967453003 0.12079749256372452 0.15981677174568176 -0.32272082567$
2018-09-19-03:57:22.917679 -0.14653174579143524 0.22326011955738068 0.12200228124856949 0.16515518724918365 -0.32591229677$
2018-09-19-03:57:23.017675 -0.1465712934732437 0.22098377346992493 0.12327811121940613 0.16342021524906158 -0.326068967580$
2018-09-19-03:57:23.117681 -0.1454542875289917 0.22209881246089935 0.12207549065351486 0.1666800081729889 -0.31716904044155$
2018-09-19-03:57:23.217676 -0.14983339607715607 0.22331994771957397 0.12182376533746672 0.17714177072048187 -0.323763698339$
2018-09-19-03:57:23.317665 -0.14878103137016296 0.2210192233324051 0.12191178649663925 0.1770520806312561 -0.32357630133362$
2018-09-19-03:57:23.417658 -0.14652030169963837 0.22213870286941528 0.12700651586055756 0.17423927783966064 -0.328918814655$
2018-09-19-03:57:23.517702 -0.14875587821006775 0.22103434801101685 0.12565478682518005 0.17514801025390625 -0.321369171145$
2018-09-19-03:57:23.617766 -0.14763009548187256 0.22215397655963898 0.12569952011108398 0.17808954417705536 -0.319285839795$
2018-09-19-03:57:23.717760 -0.1475641429424286 0.22669392824172974 0.12065038830041885 0.17370761930942535 -0.324934899806$
2018-09-19-03:57:23.817687 -0.1476208120584488 0.222158282995224 0.1269468367099762 0.17589013278484344 -0.31988441944122 3$
2018-09-19-03:57:23.917668 -0.14863012731075287 0.22673363983631134 0.12558147311210632 0.17027561366558075 -0.320859909055$
2018-09-19-03:57:24.017660 -0.14868749678134918 0.22332346439361572 0.12812024354934692 0.17353814840316772 -0.325380146505$
2018-09-19-03:57:24.117657 -0.1519826352596283 0.22451435029506683 0.12543264031410217 0.17471052706241608 -0.325330764055$
2018-09-19-03:57:24.217682 -0.14975623786449432 0.22561392188072205 0.12553691864013672 0.17481404542922974 -0.325346082445$
2018-09-19-03:57:24.317674 -0.14965781569480896 0.22905002534389496 0.12923711538314820 0.17736057937145233 -0.324474126100$
2018-09-19-03:57:24.417646 -0.1507580280303955 0.22906970977783203 0.1291773021221161 0.1757822483778 -0.3206561505794525 $
2018-09-19-03:57:24.517657 -0.14971454441547394 0.22563900053500113 0.13177543878555298 0.17102761566638947 -0.316065728664$
2018-09-19-03:57:24.617692 -0.14984577894210815 0.2210579514503479 0.12684187293052673 0.17489710450172424 -0.313021212816$
                                    [ Read 14940 lines ]
```

Figure 37. Snapshot of ICM Output

To get the same output from our IMU Module we edited the python file for the ICM:
We included the date and time in order to know when the data was recorded, this was introduced to follow the same format of the RawData given by the course's lessons tab.

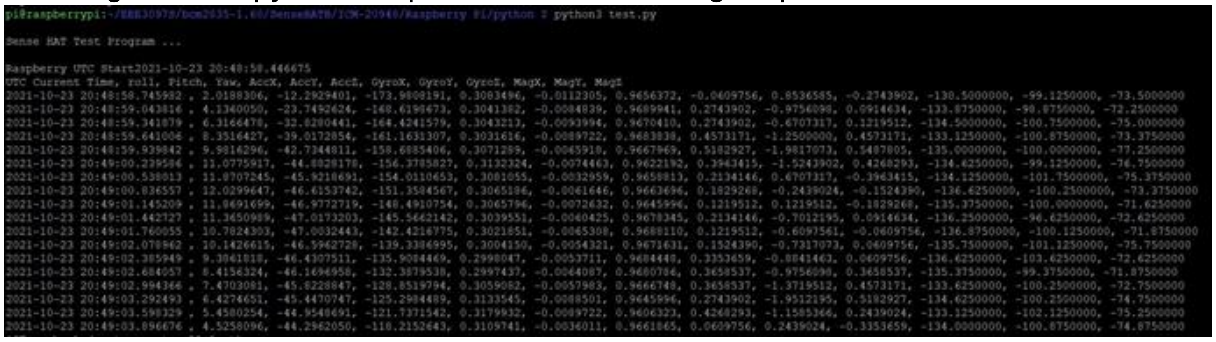
Since most ICM's list Acceleration in g's, we kept Acceleration in g's unlike in the raw data.



```
if __name__ == '__main__':
  import time
  print("\nSense HAT Test Program ...\n")
  MotionVal=[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
  icm20948=ICM20948()
  print("Raspberry UTC Start" + str(datetime.now()))
  print("UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX, MagY, MagZ")
  form = "{0:.7f}"
  while True:
    icm20948.icm20948_Gyro_Accel_Read()
    icm20948.icm20948MagRead()
    icm20948.icm20948CalAvgValue()
    time.sleep(0.1)
    icm20948.imuAHRSupdate(MotionVal[0] * 0.0175, MotionVal[1] * 0.0175,MotionVal[2] * 0.0175,
                MotionVal[3],MotionVal[4],MotionVal[5],
                MotionVal[6], MotionVal[7], MotionVal[8])
    pitch = math.asin(-2 * q1 * q3 + 2 * q0* q2)* 57.3
    roll  = math.atan2(2 * q2 * q3 + 2 * q0 * q1, -2 * q1 * q1 - 2 * q2* q2 + 1)* 57.3
    yaw   = math.atan2(-2 * q1 * q2 - 2 * q0 * q3, 2 * q2 * q2 + 2 * q3 * q3 - 1) * 57.3
    #print("\r\n /-------------------------------------------------------------/ \r\n")

    print(str(datetime.now()) + " , " + str(form.format(roll)) + ", " + str(form.format(pitch)) + ", " + str(form.format(yaw)) + ", " + str(form.format((Accel[0]/16384)))
          + ", " + str(form.format(Accel[1]/16384)) +  ", " + str(form.format(Accel[2]/16384)) + ", "
          + str(form.format(Gyro[0]/32.8)) + ", " + str(form.format(Gyro[1]/32.8)) + ", " + str(form.format(Gyro[2]/32.8)) + ", " + str(form.format(Mag[0])) +
          ", " + str(form.format(Mag[1])) + ", " + str(form.format(Mag[2])) )
```

Figure 38. Snapshot of edited ICM python file

We added some changes to the outputs, the values for acceleration were read in least significant bits, to fix this, we divided the acceleration output value by 16384, this converts it from the bit value into g's. It is reading acceleration in terms of its inertial value. We know this is right as the x direction should be close to 1 g to signify 9.8 m/s^2 which is acceleration due to gravity when it is stationary. We also changed the Gyro output by dividing the output value by 32.8 in order to get an output in degrees. Running the new python file produces the following output:



```
pi@raspberrypi:~/EEE30978/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python3 test.py

Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:48:58.446675
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX, MagY, MagZ
2021-10-23 20:48:58.745982 , 2.0188306, -12.2929401, -173.9808191, 0.3083496, -0.0112305, 0.9656372, -0.0609756, 0.8536585, -0.2743902, -130.5000000, -99.1250000, -73.5000000
2021-10-23 20:48:59.043816 , 4.1340050, -23.7492624, -168.6198673, 0.3041382, -0.0084839, 0.9689941, 0.2743902, -0.9756098, 0.0914634, -133.8750000, -98.8750000, -72.2500000
2021-10-23 20:48:59.341879 , 6.3166478, -32.0280441, -164.4241579, 0.3043213, -0.0093994, 0.9670410, 0.2743902, -0.6707317, 0.1219512, -134.5000000, -100.7500000, -75.0000000
2021-10-23 20:48:59.641006 , 8.3516427, -39.0172854, -161.1631307, 0.3031616, -0.0089722, 0.9683838, 0.4573171, -1.2500000, 0.4573171, -133.1250000, -100.8750000, -73.3750000
2021-10-23 20:48:59.939842 , 9.9816296, -42.7344811, -158.6885406, 0.3071289, -0.0065918, 0.9667969, 0.5182927, -1.9817073, 0.5487805, -135.0000000, -100.0000000, -77.2500000
2021-10-23 20:49:00.239586 , 11.0775917, -44.8828178, -156.3785827, 0.3132324, -0.0074463, 0.9622192, 0.3963415, -1.5243902, 0.4268293, -134.6250000, -99.1250000, -76.7500000
2021-10-23 20:49:00.538013 , 11.8707245, -45.9218691, -154.0110653, 0.3081055, -0.0032959, 0.9658813, 0.2134146, 0.6707317, -0.3963415, -134.1250000, -101.7500000, -75.3750000
2021-10-23 20:49:00.836557 , 12.0299647, -46.6153742, -151.3584567, 0.3065186, -0.0061646, 0.9663696, 0.1829268, -0.2439024, -0.1524390, -136.6250000, -100.2500000, -73.3750000
2021-10-23 20:49:01.145209 , 11.8691699, -46.9772719, -148.4910754, 0.3065796, -0.0072632, 0.9645996, 0.1219512, 0.1219512, -0.1829268, -135.3750000, -100.0000000, -71.6250000
2021-10-23 20:49:01.442727 , 11.3650989, -47.0173203, -145.5662142, 0.3039551, -0.0060425, 0.9678345, 0.2134146, -0.7012195, 0.0914634, -136.2500000, -96.6250000, -72.6250000
2021-10-23 20:49:01.760055 , 10.7824303, -47.0032443, -142.4216775, 0.3021851, -0.0065308, 0.9688110, 0.1219512, -0.6097561, -0.0609756, -136.8750000, -100.1250000, -71.8750000
2021-10-23 20:49:02.078962 , 10.1426615, -46.5962728, -139.3386995, 0.3004150, -0.0054321, 0.9671631, 0.1524390, -0.7317073, 0.0609756, -135.7500000, -101.1250000, -75.7500000
2021-10-23 20:49:02.385949 , 9.3061818, -46.4307511, -135.9084469, 0.2998047, -0.0053711, 0.9684448, 0.3353659, -0.8841463, 0.0609756, -136.6250000, -103.6250000, -72.6250000
2021-10-23 20:49:02.684057 , 8.4156324, -46.1696958, -132.3879538, 0.2997437, -0.0064087, 0.9680786, 0.3658537, -0.9756098, 0.3658537, -135.3750000, -99.3750000, -71.8750000
2021-10-23 20:49:02.994366 , 7.4703081, -45.8228847, -128.8519794, 0.3059082, -0.0057983, 0.9666748, 0.3658537, -1.3719512, 0.4573171, -133.6250000, -100.2500000, -72.7500000
2021-10-23 20:49:03.292493 , 6.4274651, -45.4470747, -125.2984489, 0.3133545, -0.0088501, 0.9645996, 0.2743902, -1.9512195, 0.5182927, -134.6250000, -100.2500000, -74.7500000
2021-10-23 20:49:03.598329 , 5.4580254, -44.9548691, -121.7371542, 0.3179932, -0.0089722, 0.9606323, 0.4268293, -1.1585366, 0.2439024, -133.1250000, -102.1250000, -75.2500000
2021-10-23 20:49:03.896676 , 4.5258096, -44.2962050, -118.2152643, 0.3109741, -0.0036011, 0.9661865, 0.0609756, 0.2439024, -0.3353659, -134.0000000, -100.8750000, -74.8750000
```

Figure 39. Snapshot of output from edited ICM python file

5.3. List of validation tests

Switch on sequence:
The SenseHAT(B) has been set up in such a way that is directly above the RaspberryPi and connected to it using the 40 GPIO pins. Thus, the RaspberryPi is activated the SenseHAT(B) will be activated as well due to this. The SenseHAT(B) has an onboard LED that turns red whenever it is on thus this is an indication that the SenseHAT(B) is on and activated.
Bringing a magnet nearby produces the following behavior:

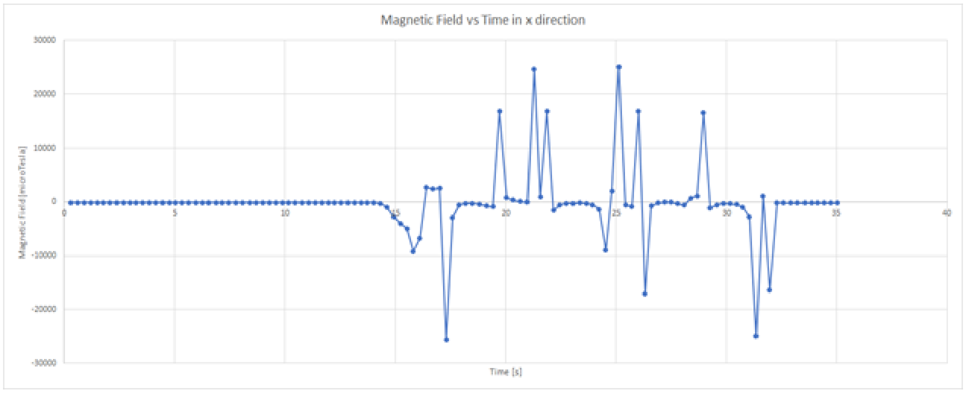


Figure 40. Graph of Magnetic Field vs Time in the x direction

Initially the SenseHAT and Pi were at rest with no nearby objects, just before 15 seconds, we brought a magnet nearby to the HAT and moved it around in all directions. We recorded this data and plotted it in the graph above. The output results produced were expected, since so magnetic objects were around initially, the reading should be 0 uT, when the magnet was brought closer, the reading changed which was expected. By moving the magnet around the SenseHAT, we expect that the value should not be constant which is proved in the graph.

Shaking motion:

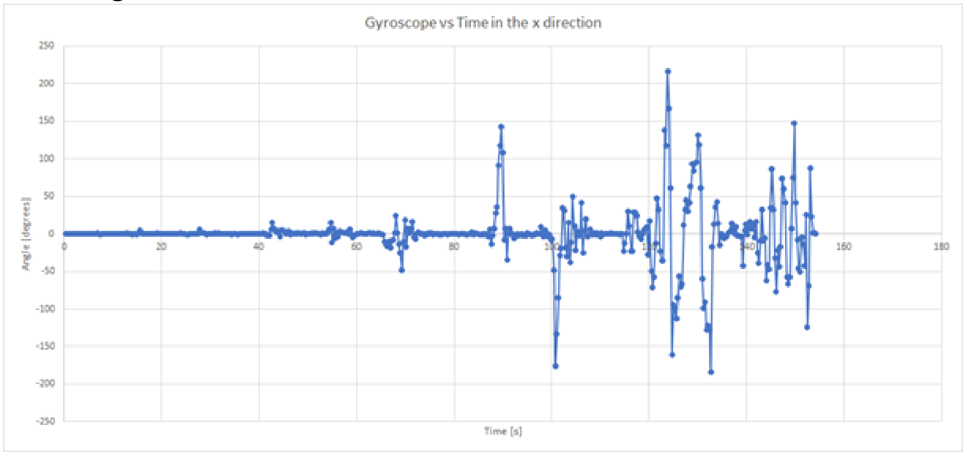


Figure 41. Graph of Gyroscope vs Time in the x direction

Initially, the Pi was placed on a desk at rest, every few seconds we would shake the table to produce a shaking motion on the Pi, when moving the pi with our hands in a "sinusoidal" shape, we also shook our hands to ensure it was accounting for this motion. These results were recorded and the graph can be witnessed above. For the first 80 seconds we can see that the Pi is not completely still, there are movements recorded which is the effect of shaking the table. When we move the Pi with a shaking motion, we can see that the graph does not move perfectly up and down, it shows some additional movements, caused by the shaking which affected its ideal motion.

5. 4. Test cases to validate IMU

Dropping The IMU from height and measuring the vertical acceleration.

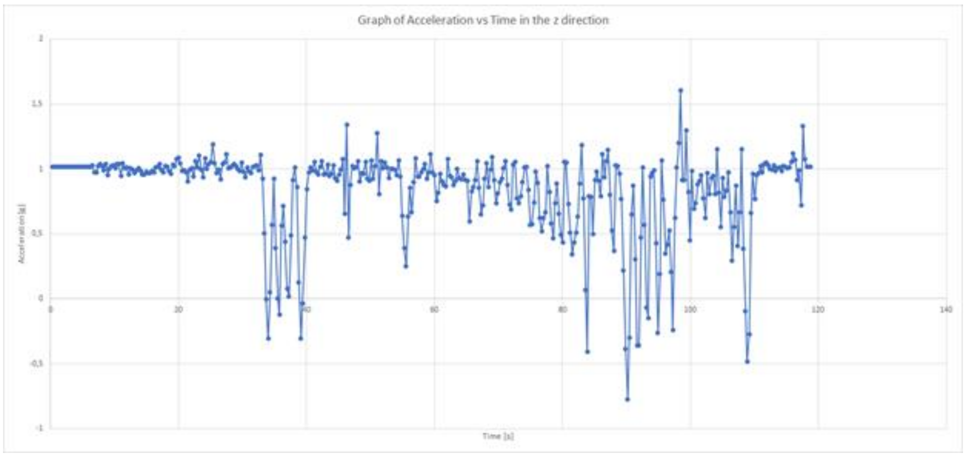


Figure 42. Graph of Acceleration vs Time in the z direction

From the above graph we see that the acceleration measured is 1 g, this signifies gravitational acceleration of 9.8 m/s^2. When the SenseHAT is dropped from some height, we expect the acceleration to drop to 0 which we observe multiple times when the device was dropped. Through this test we can validate the IMU.

Keeping the IMU Module still before moving it in x and y plane, we expect the graph to be at 0 initially since no forces are acting on the module and then when motion occurs, the resulting force records a change in acceleration.

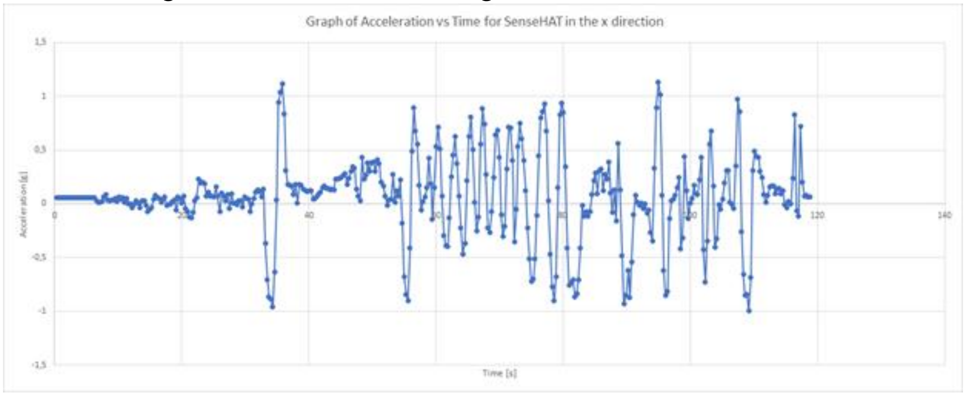


Figure 43. Graph of Acceleration vs Time in the x direction

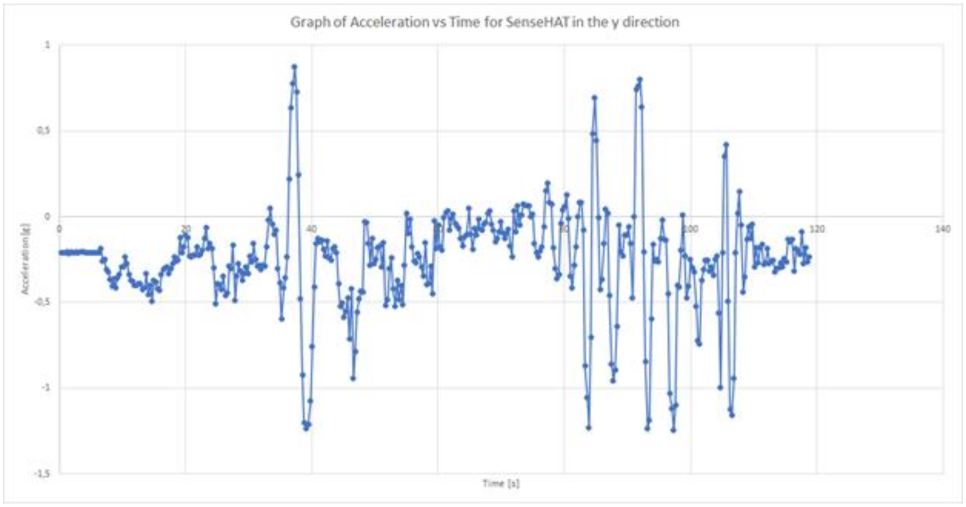


Figure 44. Graph of Acceleration vs Time in the y direction

Moving the IMU in all three planes after being at rest produces the following graphs. We expect it to be centered at 0 first and then begin to move due to motion taking place.

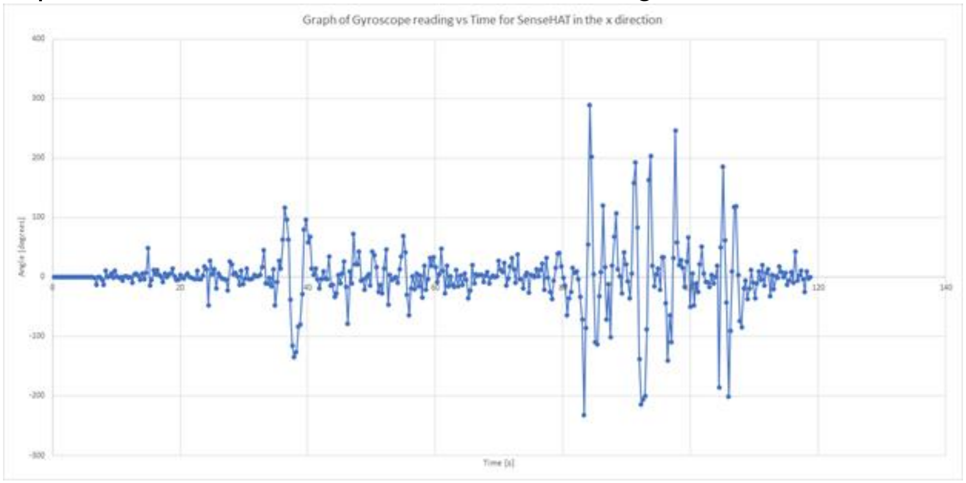


Figure 45. Graph of Gyroscope vs Time in the x direction

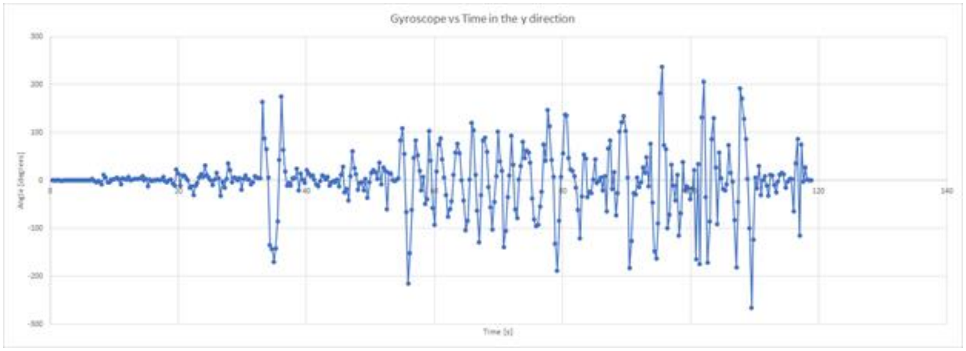


Figure 46. Graph of Gyroscope vs Time in the y direction

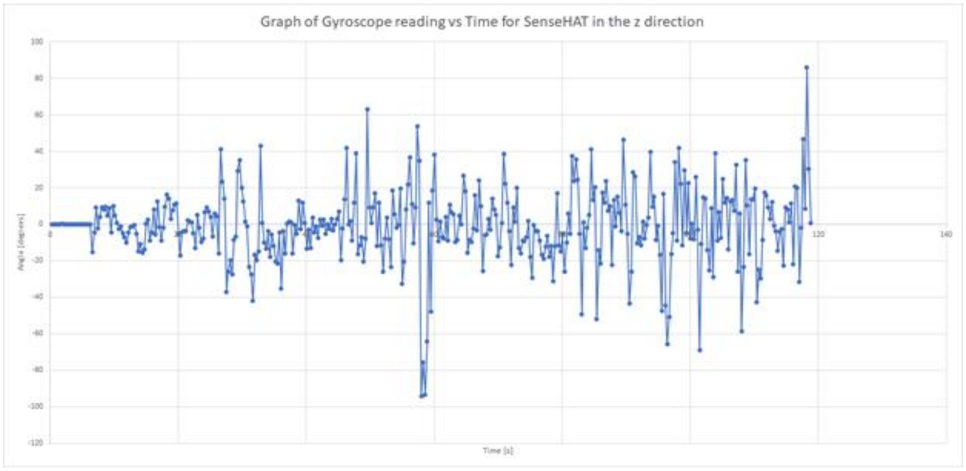


Figure 47. Graph of Gyroscope vs Time in the z direction

Moving the IMU module away and towards a smartphone produces the following output, we expect this output due to the phones own magnetic field that is generated.

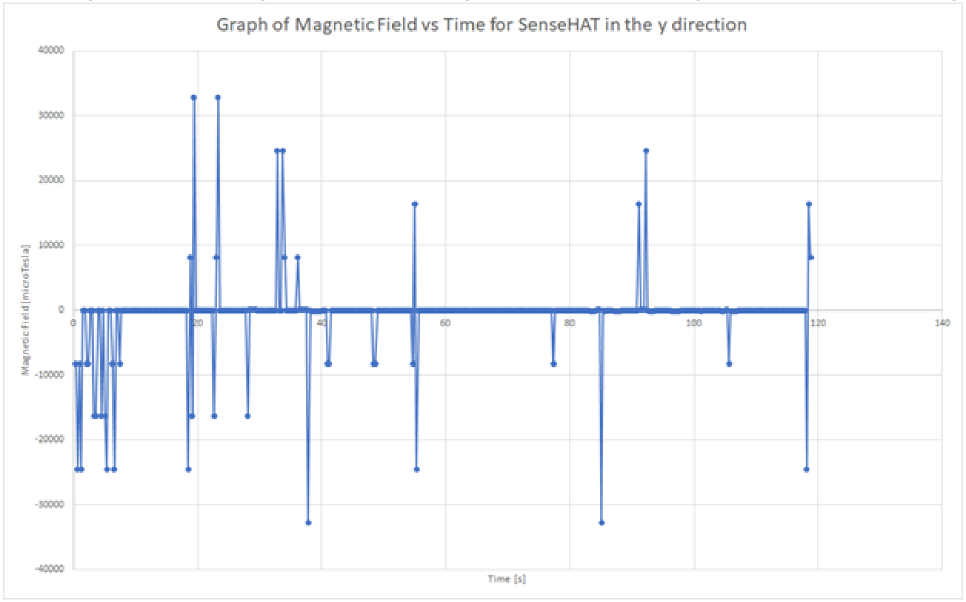


Figure 48. Graph of Magnetic Field vs Time in the y direction

5. 5. Experiment Setup

<u>**Overall Functionality**</u>

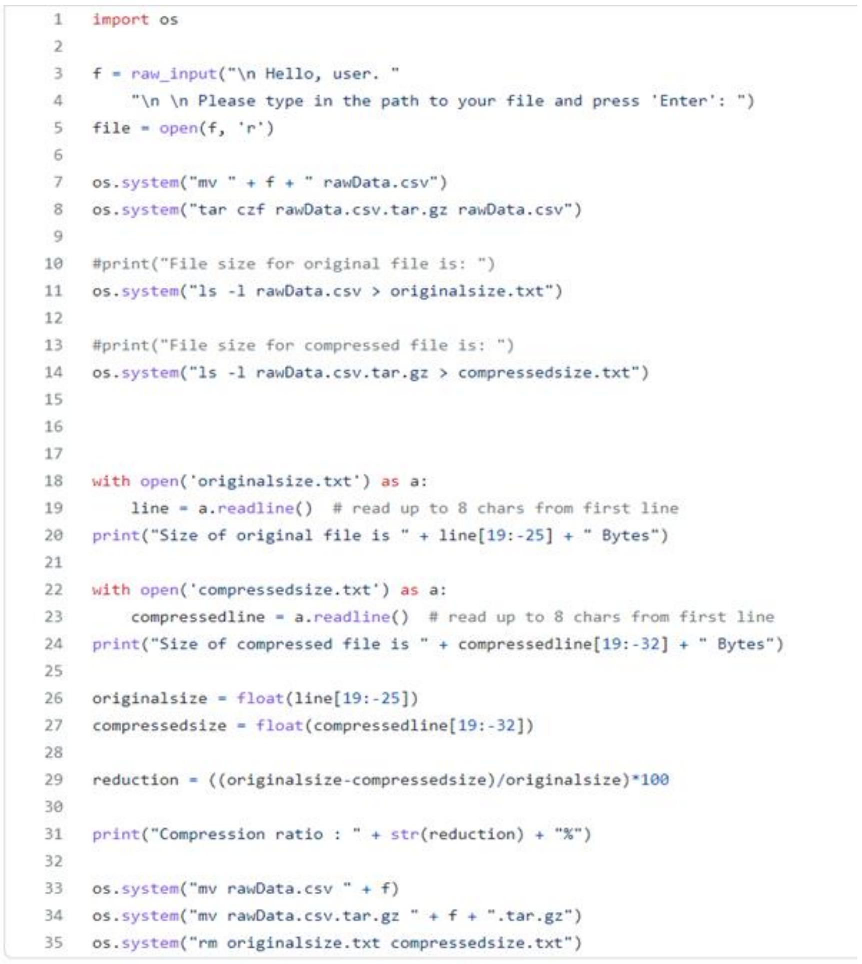
The experimentation of the overall system started with individually experimenting with the SenseHAT(B) and running various tests on the ICM that mimic real life experiences such as noise, movement, to name a few. Based on these tests, data was collected and stored in a csv file.

This csv file was then extracted onto the RaspberryPi from the SenseHAT(B) and used as input into the data integrity block. The data integrity block checked every data item in

the file to ensure that it matched certain parameters and to validate the data items. Once this has been it checked the file then proceeds to the compression block.
The compression block is used to compress the file to a size that is significantly smaller than the original file. The significance in compression is dependent on the size of the file as there are differences in compression sizes for each data file that was tested. However, the compression size was still adequately small enough to proceed to the encryption block.
The encryption block read in the file, encrypted it using a key and then sent it to the relevant user.
In order to ensure that the file received by the user is valid and correct, we created a decrypt and decompress algorithm. This algorithm read in the compressed and encrypted file, decrypted it, and then decompressed it. After the entire process, the contents of the file were compared to the original file. It was found that the contents had not changed and thus the overall functionality of our program is successful.

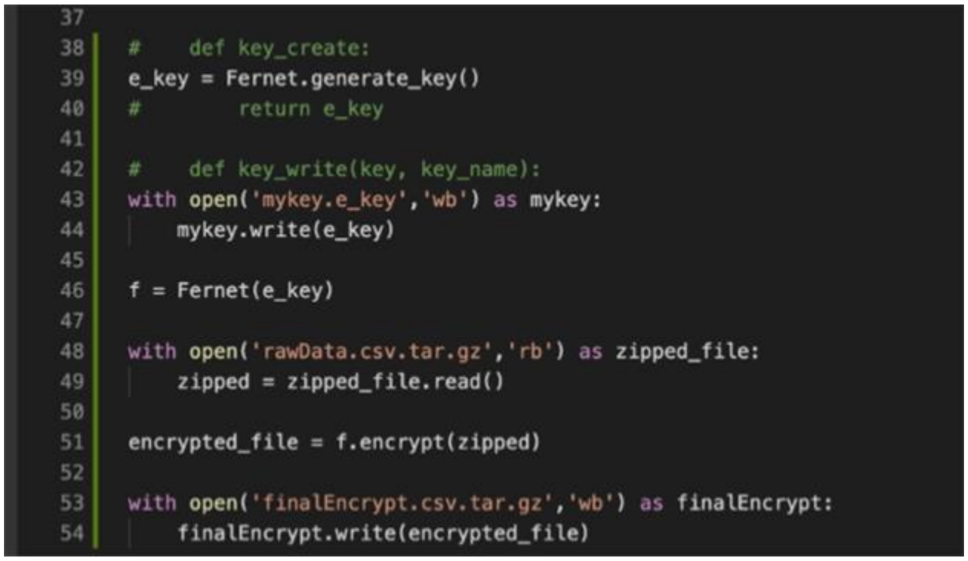
Below is a further in-depth explanation of the above:
Begin acquiring data from the SenseHAT
The first step is to read in the file that contains the data using a csv reader. The data within the file will then be checked against certain parameters to ensure that it is valid and may be used further. If the data items are valid this file will then be sent to the DesignCompEncr.py program.
The second step is compression, when running the file DesignCompEncr.py (https://github.com/TheAllyA/EEE3097S-Project/blob/main/DesignCompEncr.py), this will be run with Python3.
The user will be prompted to input the path to the file that must be compressed and encrypted. The Pi has CSV files that will be used in the experiment, acquired from the SenseHAT. After the user enters the file path, the program will open the file inputted by the user and then rename it to a temporary file called rawData.csv. Thereafter it runs the bash command that initiates compression, after compression it produces a compressed file called rawData.csv.tar.gz. Running the BASH command 'ls -l' produces the information about a file which includes its file size, using the command ">" we can send that to a text file, we do this on rawData.csv and rawData.csv.tar.gz, this will allow us to access information about the original and compressed file. Thereafter the program opens the text files that contains the file size of rawData.csv which is the original file and using string slicing, we can access the value of the file size and output it to the user. We do the same with the compressed file size text file. After that we convert the file sizes to floats and perform a mathematical operation to produce a percentage that shows how much the original file has been reduced by. We then rename the files to the original file names that the user inputted and remove the text files that contain the sizes of the compressed and original files. The compressed file, rawData.csv.tar.gz then gets passed to the encryption algorithm which takes it as input. The file contents get read into a different file that will be edited. A key is generated at the beginning of the program using the Fernet method and it is used to encrypt the new file. This is a

security measure to ensure that the file can only be decrypted with this key and thus prevents any sort of outside interference. Once the file is encrypted, it is written to a finalEncrypt.csv.tar.gz file that can only be read and not written to. This is to ensure the data from the IMU cannot be changed or manipulated.



```
import os

f = raw_input("\n Hello, user. "
    "\n \n Please type in the path to your file and press 'Enter': ")
file = open(f, 'r')

os.system("mv " + f + " rawData.csv")
os.system("tar czf rawData.csv.tar.gz rawData.csv")

#print("File size for original file is: ")
os.system("ls -l rawData.csv > originalsize.txt")

#print("File size for compressed file is: ")
os.system("ls -l rawData.csv.tar.gz > compressedsize.txt")



with open('originalsize.txt') as a:
    line = a.readline()  # read up to 8 chars from first line
print("Size of original file is " + line[19:-25] + " Bytes")

with open('compressedsize.txt') as a:
    compressedline = a.readline()  # read up to 8 chars from first line
print("Size of compressed file is " + compressedline[19:-32] + " Bytes")

originalsize = float(line[19:-25])
compressedsize = float(compressedline[19:-32])

reduction = ((originalsize-compressedsize)/originalsize)*100

print("Compression ratio : " + str(reduction) + "%")

os.system("mv rawData.csv " + f)
os.system("mv rawData.csv.tar.gz " + f + ".tar.gz")
os.system("rm originalsize.txt compressedsize.txt")
```

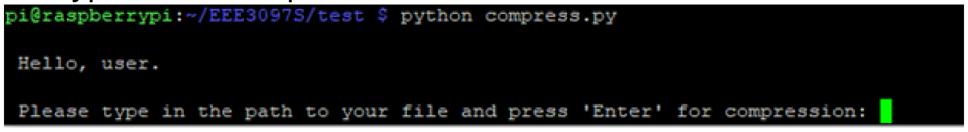
Figure 49. Snapshot of compression algorithm



```
#    def key_create:
e_key = Fernet.generate_key()
#         return e_key

#    def key_write(key, key_name):
with open('mykey.e_key','wb') as mykey:
    mykey.write(e_key)

f = Fernet(e_key)

with open('rawData.csv.tar.gz','rb') as zipped_file:
    zipped = zipped_file.read()

encrypted_file = f.encrypt(zipped)

with open('finalEncrypt.csv.tar.gz','wb') as finalEncrypt:
    finalEncrypt.write(encrypted_file)
```

Figure 50. Snapshot of encryption algorithm

Decrypt and Decompress:



```
pi@raspberrypi:~/EEE3097S/test $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression:
```

Figure 51. Snapshot of user entry in compression algorithm

The first step in the DesignDecrypDecomp.py (https://github.com/TheAllyA/EEE3097S-Project/blob/main/DesignDecrypDecomp.py) is to decrypt the file. For the decryption, the encrypted file is taken as input to be read. After being read, the file is decrypted and written to a separate file that will contain the decrypted data items.

The second step is to decompress the decrypted file. For decompression, the program will use the file indicated in the decryption block to be decompressed and then will execute the decompression BASH command to decompress the compressed file. We will then use the 'cmp' command in BASH to compare the decompressed file to the original file, if no output is observed, the files are the same and no data has been lost.

Data Acquisition:

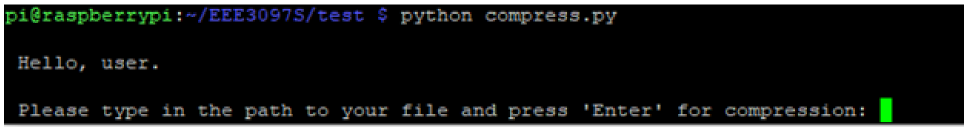
To acquire the data on our pi's we ran the edited python file acquired by waveshare which can be found here https://github.com/TheAllyA/EEE3097S-Project/edit/main/SenseHAT/SenseHAT.py, running this on the pi produces the data from the sensors. We then send the output from the ICM into a text file and name it in a csv format.

Compression:

Worked on by Greg Da Silva (DSLGRE001)

The compression algorithm was set up using python language. The program imports os in order to interact with the Pi shell which operates on linux.

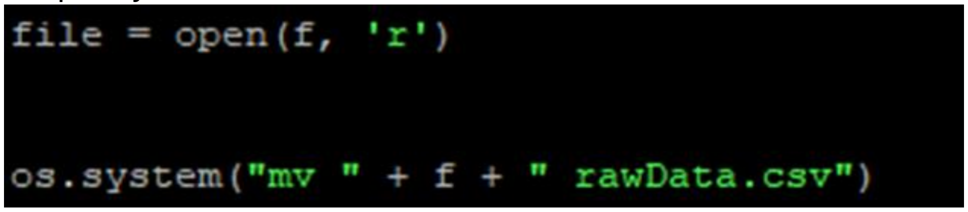
When the file compress.py is run using 'python compress.py', the program prompts the user to enter the path to the file that is indented to be compressed.



```
pi@raspberrypi:~/EEE3097S/test $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression:
```

Figure 52. Snapshot of prompt from compression algorithm

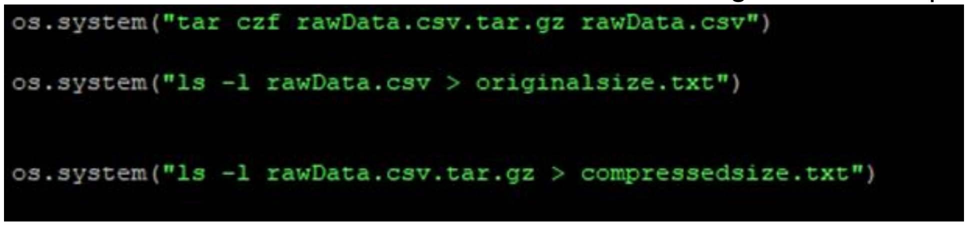
The program ‘compress.py’ opens the file inputted by the user and then renames it to a temporary file called rawData.csv,



```
file = open(f, 'r')


os.system("mv " + f + " rawData.csv")
```

Figure 53. Snapshot of compression algorithm

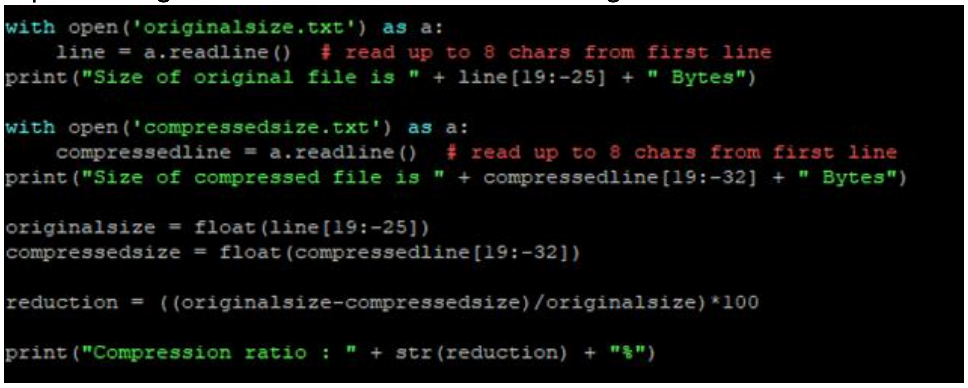
Thereafter it runs the bash command that initiates compression, after compression it produces a compressed file called rawData.csv.tar.gz. Running the BASH command ‘ls -l’ produces the information about a file which includes its file size, using the command “>” we can send that to a text file, we do this on rawData.csv and rawData.csv.tar.gz, this will allow us to access information about the original and compressed file.



```
os.system("tar czf rawData.csv.tar.gz rawData.csv")

os.system("ls -l rawData.csv > originalsize.txt")


os.system("ls -l rawData.csv.tar.gz > compressedsize.txt")
```

Figure 54. Snapshot of compression algorithm

Thereafter the program opens the text files that contains the file size of rawData.csv which is the original file and using string slicing, we can access the value of the file size and output it to the user. We do the same with the compressed file size text file. After that we convert the file sizes to floats and perform a mathematical operation to produce a percentage that shows how much the original file has been reduced by.

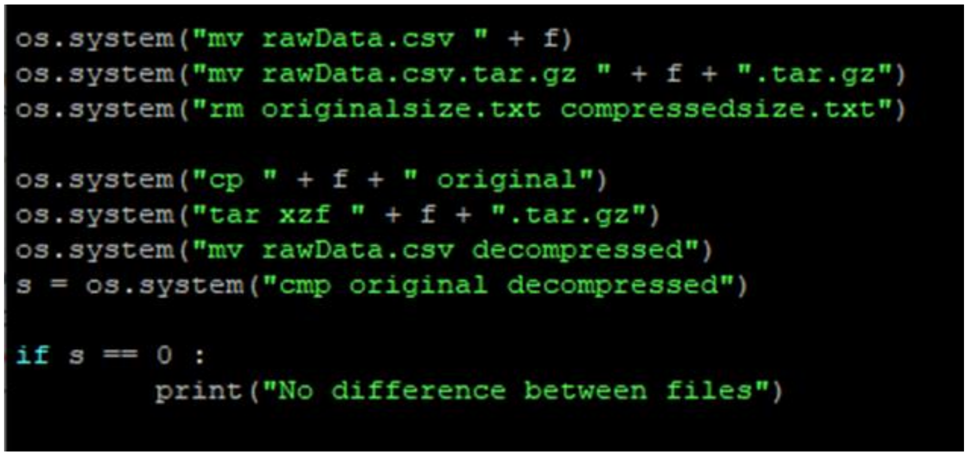


```
with open('originalsize.txt') as a:
    line = a.readline()  # read up to 8 chars from first line
print("Size of original file is " + line[19:-25] + " Bytes")

with open('compressedsize.txt') as a:
    compressedline = a.readline()  # read up to 8 chars from first line
print("Size of compressed file is " + compressedline[19:-32] + " Bytes")

originalsize = float(line[19:-25])
compressedsize = float(compressedline[19:-32])

reduction = ((originalsize-compressedsize)/originalsize)*100

print("Compression ratio : " + str(reduction) + "%")
```

Figure 55. Snapshot of compression algorithm

We then rename the files to the original file names that the user inputted and remove the text files that contain the sizes of the compressed and original files.

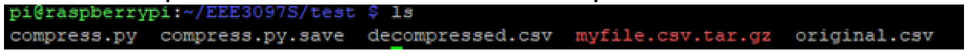
We then make a copy of the original file before compression and name the file original. We then run a decompression BASH command and produce a decompressed file and rename that decompressed.
Using the ‘cmp’ command we compare the original and decompressed file and if no differences are found between the files the program outputs ‘No difference between files’



```
os.system("mv rawData.csv " + f)
os.system("mv rawData.csv.tar.gz " + f + ".tar.gz")
os.system("rm originalsize.txt compressedsize.txt")

os.system("cp " + f + " original")
os.system("tar xzf " + f + ".tar.gz")
os.system("mv rawData.csv decompressed")
s = os.system("cmp original decompressed")

if s == 0 :
        print("No difference between files")
```

Figure 56. Snapshot of compression algorithm

By using the BASH command ls, we can see the files in the local directory, the files contained are original.csv and decompressed.csv. These two files represent the file before compression and the file after compression.



```
pi@raspberrypi:~/EEE3097S/test $ ls
compress.py  compress.py.save  decompressed.csv  myfile.csv.tar.gz  original.csv
```

Figure 57. Snapshot of files in the directory after running compression algorithm

Using the ‘cmp’ BASH command we can compare the contents of original.csv with decompressed.cdv, doing this results in an empty output, illustrating that no difference between the data and that original.csv is exactly the same as decompressed.csv.
This ‘cmp’ command is already included in the compress.py program and outputs the sentence “No difference between files” if ‘cmp’ returns an empty output. This proves that the experiment is successful and the ATP has been met in the case where decompressed data is the same as compressed data and no data has been lost in the process.
The experiment starts by running the compression python file and will ask the user which file they want to compress, after the user has inputted the csv file, the compression algorithm, will take the csv file generated by the SenseHAT after running the python file and compress it. In this experiment we ran the SenseHAT file 4 times with a sleep time of 0.1s. We took the 4 csv files and used them in the compression algorithm. We then took note of the compression ratio and then decompressed the file and compared the original csv file to the decompressed file. If there are no differences, then the compression algorithm will output “No difference between files”. If there are differences, the algorithm will output them. The four different files obtained from the SenseHAT will have various data sizes in order to compare the compression ratio for each case.

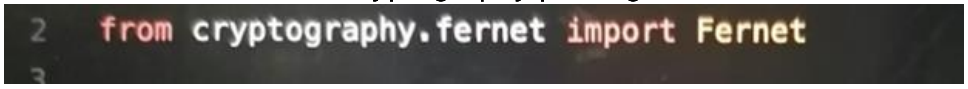
After this, we repeated the experiment while changing the sleep time from 0.01s to 0.05s and 0.5s and ran the compression algorithm on the csv files.

Encryption:

Worked on by ADMALI004

The encryption program was set up by using Python3 as the programming language of choice. The encryption program makes use of a cryptography module which is only available and effective using Python3. The installation of cryptography also required the "pip" command to be used thus in total the encryption program required an additional installation of python3, pip.py and cryptography.

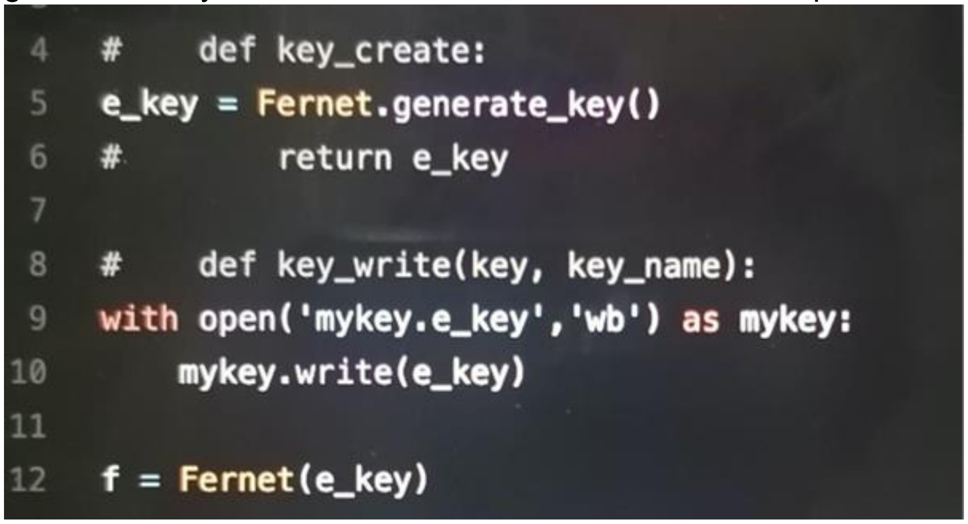
There was one program created, namely, encrypted.py (encrypted) that imports the Fernet module from the cryptography package.



```
from cryptography.fernet import Fernet
```

Figure 58. Module used for encryption

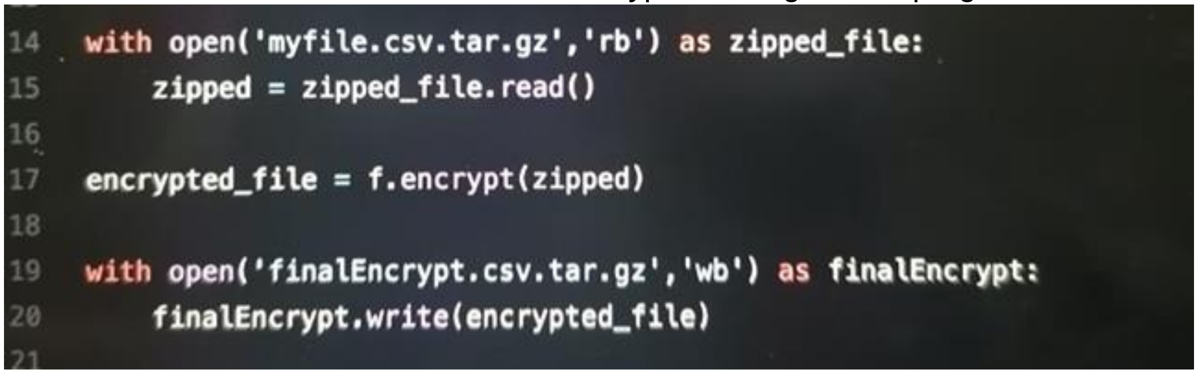
This module contains all the methods and functions needed for the encryption and decryption, namely, generate(), encryt and decrypt. The generate() method is used to generate a key that will be used to "lock" and "unlock" a specified file.



```
#    def key_create:
e_key = Fernet.generate_key()
#        return e_key

#    def key_write(key, key_name):
with open('mykey.e_key','wb') as mykey:
    mykey.write(e_key)

f = Fernet(e_key)
```

Figure 59. Snapshot of encryption algorithm

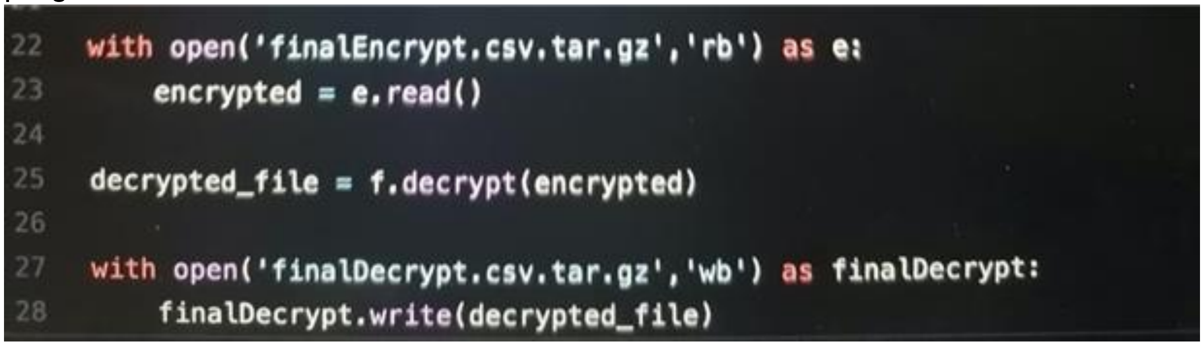
The encrypt function is then used to read the file, encrypt it and write the encrypted data to a new file. This new file is called finalEncrypt.csv.tar.gz in the program.



```
with open('myfile.csv.tar.gz','rb') as zipped_file:
    zipped = zipped_file.read()

encrypted_file = f.encrypt(zipped)

with open('finalEncrypt.csv.tar.gz','wb') as finalEncrypt:
    finalEncrypt.write(encrypted_file)
```

Figure 60. Snapshot of encryption algorithm

The decrypt function is used to read the encrypted file, decrypt it and write the decrypted data to a new file. This new file is called finalDecrypt.csv.tar.gz in the program.



```
with open('finalEncrypt.csv.tar.gz','rb') as e:
    encrypted = e.read()

decrypted_file = f.decrypt(encrypted)

with open('finalDecrypt.csv.tar.gz','wb') as finalDecrypt:
    finalDecrypt.write(decrypted_file)
```

Figure 61. Snapshot of decryption algorithm

The files that were used in the program to be encrypted and decrypted was obtained from the compression processing block, namely Data_1.csv, Data_2.csv, Data_3.csv and Data_4.csv. These files all contain different data that was extracted from the SenseHAT(B), however, proved to behave in the same manner as the rawData.csv file that was given to us.

Each file was compared to its original decompressed file. Each file was obtained from the compression block, encrypted, decrypted and then decompressed to produce the data was within the file. This data was then compared to the original contents of the file, and it was proven to be exactly the same. This ensures that encrypting the file does not cause any data to be loss and is successful in encrypting and decrypting the file safely.

Data integrity:

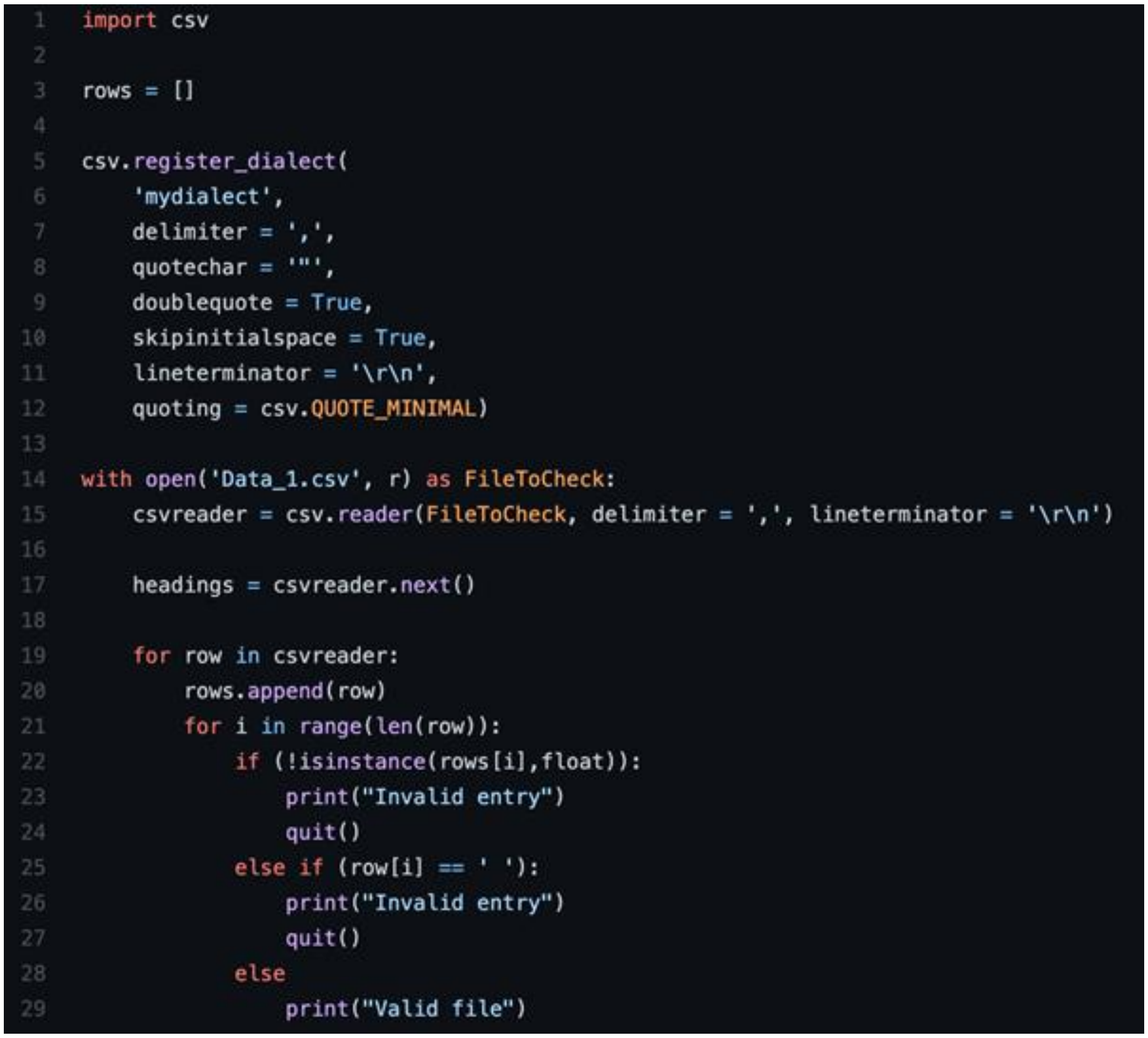
Worked on by ADMALI004

The data integrity program (DataInt) was set up using Python3 as the programming language of choice. The data integrity program makes use of the csv module that had to be imported.

There was one program created, namely, DataInt.py that makes use of all the methods and functions that are available in the csv module. The open() method is first used to open up the file that contains the data that was extracted from the IMU. The reader object is then used to read the file and the next() method is used to read each line within the file. The data items within each line gets further extracted and checked to ensure that it is a float point value and that it is not an arbitrary value such as a space. If both checks return true then the program does not automatically quit and it returns “Valid file”. However, if one of the checks returns false then an error message shows and the program stops. This process occurs within a for loop to ensure that each data item in each row is checked.

The program only sends the file through to get compressed if both checks return true for each data item.

Below is the code for data integrity program:


```
import csv

rows = []

csv.register_dialect(
    'mydialect',
    delimiter = ',',
    quotechar = '"',
    doublequote = True,
    skipinitialspace = True,
    lineterminator = '\r\n',
    quoting = csv.QUOTE_MINIMAL)

with open('Data_1.csv', r) as FileToCheck:
    csvreader = csv.reader(FileToCheck, delimiter = ',', lineterminator = '\r\n')

    headings = csvreader.next()

    for row in csvreader:
        rows.append(row)
        for i in range(len(row)):
            if (!isinstance(rows[i],float)):
                print("Invalid entry")
                quit()
            else if (row[i] == ' '):
                print("Invalid entry")
                quit()
            else
                print("Valid file")
```
Figure 62. Snapshot of data integrity algorithm

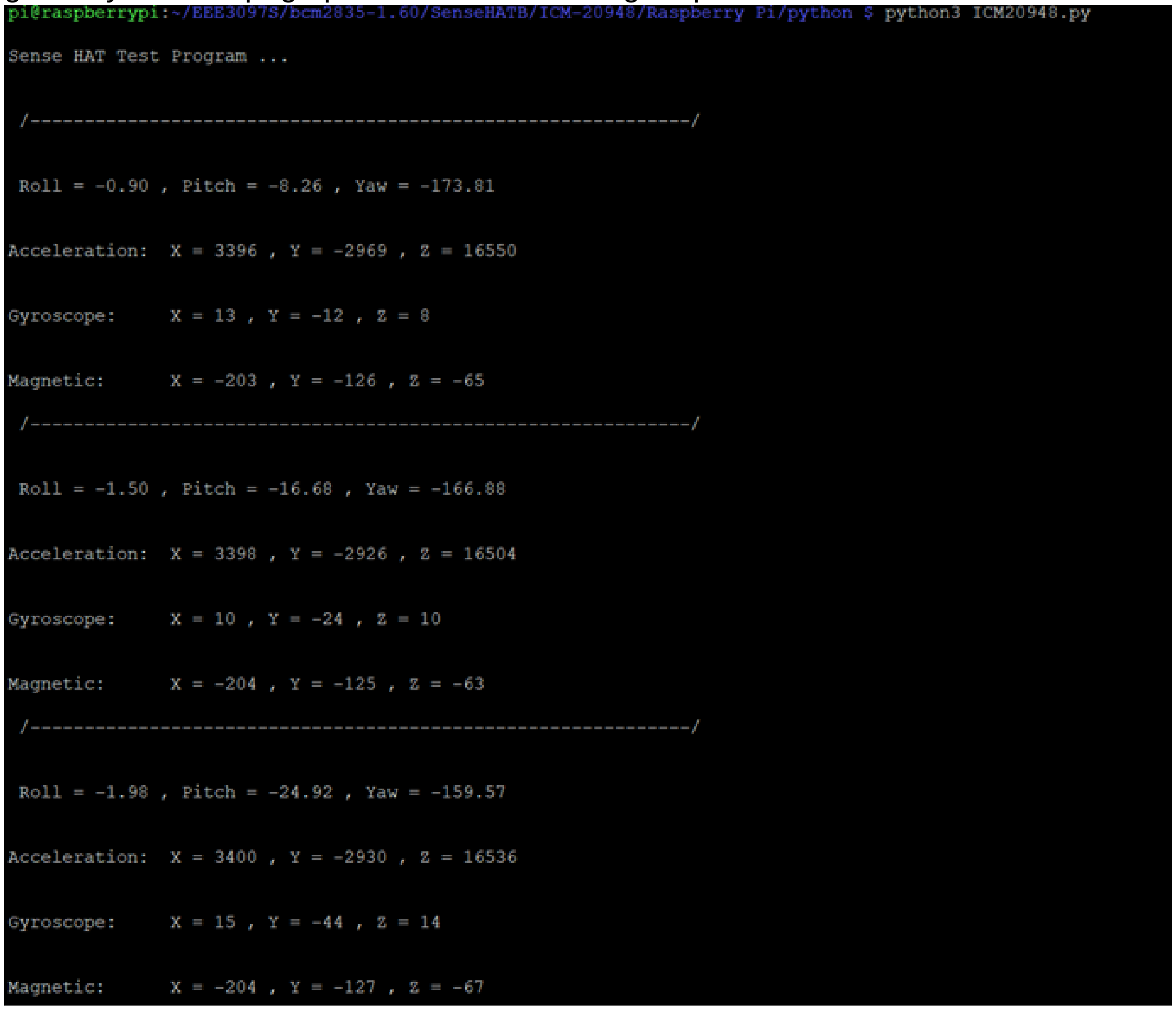
Data Each Block is Expected to get in and give out

A csv file is read into the data integrity algorithm. This file is then thoroughly checked to ensure certain parameters are met for the file to be valid. If this is successful then the csv file enters the compression block, this is a file that contains RawData from the IMU Module in String format. It is then compressed and the file produced after that is a compressed file which contains compressed data. This compressed file has a smaller size than the original csv file. This compressed file is then sent into the encryption block where it is encrypted with a key that gets generated specifically for the file. This encrypted file will then get sent off to the relevant parties that will make use of the data. The generated key will be needed to decrypt the file.

5.6. Results

Overall Functionality

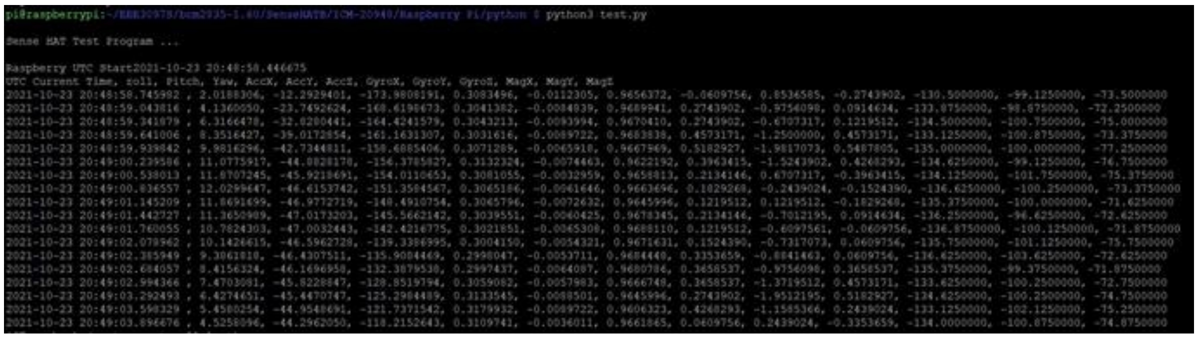
We used a wiki page [1] to set up the SenseHAT on our Raspberry Pi. The python file given by the wiki page produced the following output from the ICM when it was run:



```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python3 ICM20948.py

Sense HAT Test Program ...

 /-------------------------------------------------------------/

 Roll = -0.90 , Pitch = -8.26 , Yaw = -173.81

Acceleration:  X = 3396 , Y = -2969 , Z = 16550

Gyroscope:     X = 13 , Y = -12 , Z = 8

Magnetic:      X = -203 , Y = -126 , Z = -65
 /-------------------------------------------------------------/

 Roll = -1.50 , Pitch = -16.68 , Yaw = -166.88

Acceleration:  X = 3398 , Y = -2926 , Z = 16504

Gyroscope:     X = 10 , Y = -24 , Z = 10

Magnetic:      X = -204 , Y = -125 , Z = -63
 /-------------------------------------------------------------/

 Roll = -1.98 , Pitch = -24.92 , Yaw = -159.57

Acceleration:  X = 3400 , Y = -2930 , Z = 16536

Gyroscope:     X = 15 , Y = -44 , Z = 14

Magnetic:      X = -204 , Y = -127 , Z = -67
```

Figure 63. Screenshot of output from original python file

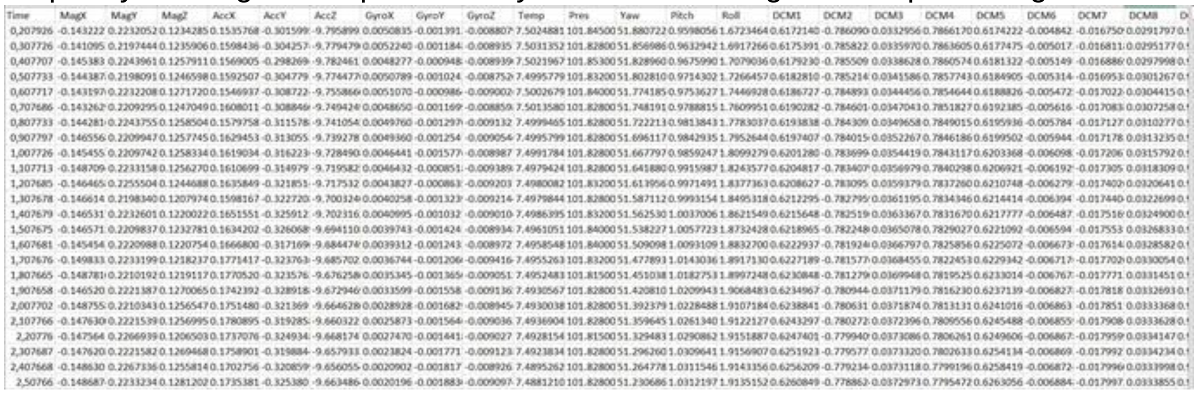
We decided to change the format of the output in order to save space and make it easier to analyse the data once it is sent to the database for analysis. After doing this, our new edited python file produced the following output:



```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHAT B/ICM-20948/Raspberry Pi/python $ python3 test.py

Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:48:58.446675
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX, MagY, MagZ
2021-10-23 20:48:58.745982 , 2.0188306, -12.2929401, -173.9808191, 0.3083496, -0.0112305, 0.9656372, -0.0609756, 0.8536585, -0.2743902, -130.5000000, -99.1250000, -73.5000000
2021-10-23 20:48:59.043816 , 4.1360050, -23.7492624, -168.6198673, 0.3041382, -0.0084839, 0.9689941, 0.2743902, -0.9756098, 0.0914634, -133.8750000, -98.8750000, -72.2500000
2021-10-23 20:48:59.341879 , 6.3166478, -32.0280441, -164.4241579, 0.3043213, -0.0093994, 0.9670410, 0.2743902, -0.6707317, 0.1219512, -134.5000000, -100.7500000, -75.0000000
2021-10-23 20:48:59.641006 , 8.3516427, -39.0172854, -161.1631307, 0.3031616, -0.0089722, 0.9683838, 0.4573171, -1.2500000, 0.4573171, -133.1250000, -100.8750000, -73.3750000
2021-10-23 20:48:59.939842 , 9.9816296, -42.7344811, -158.6085406, 0.3071289, -0.0065918, 0.9667969, 0.5182927, -1.9817073, 0.5487805, -135.0000000, -100.0000000, -77.2500000
2021-10-23 20:49:00.239586 , 11.0775917, -44.8828170, -156.3785827, 0.3132324, -0.0074463, 0.9622192, 0.3963415, -1.5243902, 0.4268293, -134.6250000, -99.1250000, -76.7500000
2021-10-23 20:49:00.538013 , 11.8707245, -45.9218691, -154.0110653, 0.3081055, -0.0032959, 0.9658813, 0.2134146, 0.6707317, -0.3963415, -134.1250000, -101.7500000, -75.3750000
2021-10-23 20:49:00.836557 , 12.0299647, -46.6153742, -151.3584567, 0.3065186, -0.0061646, 0.9663696, 0.1829268, -0.2439024, -0.1524390, -136.6250000, -100.2500000, -73.3750000
2021-10-23 20:49:01.145209 , 11.8691699, -46.9772719, -148.4910754, 0.3065796, -0.0072632, 0.9645996, 0.1219512, 0.1219512, -0.1829268, -135.3750000, -100.0000000, -71.6250000
2021-10-23 20:49:01.442727 , 11.3650909, -47.0173203, -145.5662142, 0.3039551, -0.0060425, 0.9678345, 0.2134146, -0.7012195, 0.0914634, -136.2500000, -96.6250000, -72.6250000
2021-10-23 20:49:01.760055 , 10.7824303, -47.0032443, -142.4216775, 0.3021851, -0.0065308, 0.9688110, 0.1219512, -0.6097561, -0.0609756, -136.8750000, -100.1250000, -71.8750000
2021-10-23 20:49:02.078962 , 10.1426615, -46.5962728, -139.3386995, 0.3004150, -0.0054321, 0.9671631, 0.1524390, -0.7317073, 0.0609756, -135.7500000, -101.1250000, -75.7500000
2021-10-23 20:49:02.385949 , 9.3061810, -46.4307511, -135.9084469, 0.2998047, -0.0053711, 0.9684440, 0.3353659, -0.8841463, 0.0609756, -136.6250000, -103.6250000, -72.6250000
2021-10-23 20:49:02.684057 , 8.4156324, -46.1696958, -132.3879538, 0.2997437, -0.0064087, 0.9680786, 0.3658537, -0.9756098, 0.3658537, -135.3750000, -99.3750000, -71.8750000
2021-10-23 20:49:02.994366 , 7.4703081, -45.8228847, -128.8519794, 0.3059082, -0.0057983, 0.9666748, 0.3658537, -1.3719512, 0.4573171, -133.6250000, -100.2500000, -72.7500000
2021-10-23 20:49:03.292493 , 6.4274651, -45.4470747, -125.2984489, 0.3133545, -0.0088501, 0.9645996, 0.2743902, -1.9512195, 0.5182927, -134.6250000, -100.2500000, -74.7500000
2021-10-23 20:49:03.598329 , 5.4580254, -44.9548691, -121.7371542, 0.3179932, -0.0089722, 0.9606323, 0.4268293, -1.1585366, 0.2439024, -133.1250000, -102.1250000, -75.2500000
2021-10-23 20:49:03.896676 , 4.5258096, -44.2962050, -118.2152643, 0.3109741, -0.0036011, 0.9661865, 0.0609756, 0.2439024, -0.3353659, -134.0000000, -100.8750000, -74.8750000
```

Figure 64. Snapshot of output from edited python file

From the above screenshot we can see that the results produced by the ICM produce an output such that it is easier to open the file in Excel and analyze it, we also included a time and date output to ensure that the data can be categorized and can be analyzed better as opposed to have data without knowing the time or date. We followed the same convention that of the RawData csv files we were given in the course's lessons tab. We added some changes to the outputs, the values for acceleration were read in least significant bits, to fix this, we divided the acceleration output value by 16384, this converts it from the bit value into g's. It is reading acceleration in terms of its inertial value. We know this is right as the x direction should be close to 1 g to signify 9.8 m/s^2 which is acceleration due to gravity when it is stationary. We also changed the Gyro output by dividing the output value by 32.8 in order to get an output in degrees.



Time	MagX	MagY	MagZ	AccX	AccY	AccZ	GyroX	GyroY	GyroZ	Temp	Pres	Yaw	Pitch	Roll	DCM1	DCM2	DCM3	DCM4	DCM5	DCM6	DCM7	DCM8	D
0,207926	-0.143222	0.2232052	0.1234285	0.1535768	-0.301599	-9.795899	0.0050835	-0.001391	-0.008807	7.5024881	101.84500	51.880722	0.9598056	1.6723464	0.6172140	-0.786090	0.0332956	0.7866170	0.6174222	-0.004842	-0.016750	0.0291797	0.
0,307726	-0.141095	0.2197444	0.1235906	0.1598436	-0.304257	-9.779479	0.0052240	-0.001184	-0.008935	7.5031352	101.82800	51.856986	0.9632942	1.6917266	0.6175391	-0.785822	0.0335970	0.7863605	0.6177475	-0.005017	-0.016811	0.0295177	0.
0,407707	-0.145383	0.2243961	0.1257911	0.1569005	-0.298269	-9.782461	0.0048277	-0.000948	-0.008939	7.5021967	101.85300	51.828960	0.9675990	1.7079036	0.6179230	-0.785509	0.0338628	0.7860574	0.6181322	-0.005149	-0.016886	0.0297998	0.
0,507733	-0.144387	0.2198091	0.1246598	0.1592507	-0.304779	-9.774477	0.0050789	-0.001024	-0.008752	7.4995779	101.83200	51.802810	0.9714302	1.7266457	0.6182810	-0.785214	0.0341586	0.7857743	0.6184905	-0.005314	-0.016953	0.0301267	0.
0,607717	-0.143197	0.2232208	0.1271720	0.1546937	-0.308722	-9.755866	0.0051070	-0.000986	-0.009002	7.5002679	101.84000	51.774185	0.9753627	1.7446928	0.6186727	-0.784893	0.0344456	0.7854644	0.6188826	-0.005472	-0.017022	0.0304415	0.
0,707686	-0.143262	0.2209295	0.1247049	0.1608011	-0.308846	-9.749424	0.0048650	-0.001169	-0.008859	7.5013580	101.82800	51.748191	0.9788815	1.7609951	0.6190282	-0.784601	0.0347043	0.7851827	0.6192385	-0.005616	-0.017083	0.0307258	0.
0,807733	-0.144281	0.2243755	0.1258504	0.1579758	-0.311578	-9.741054	0.0049760	-0.001297	-0.009132	7.4999465	101.82800	51.722213	0.9813843	1.7783037	0.6193838	-0.784309	0.0349658	0.7849015	0.6195936	-0.005784	-0.017127	0.0310277	0.
0,907797	-0.146556	0.2209947	0.1257745	0.1629453	-0.313055	-9.739278	0.0049360	-0.001254	-0.009054	7.4995799	101.82800	51.696117	0.9842935	1.7952644	0.6197407	-0.784015	0.0352267	0.7846186	0.6199502	-0.005944	-0.017178	0.0313235	0.
1,007726	-0.145455	0.2209742	0.1258334	0.1619034	-0.316223	-9.728490	0.0046441	-0.001577	-0.008987	7.4991784	101.82800	51.667797	0.9859247	1.8099279	0.6201280	-0.783699	0.0354419	0.7843117	0.6203368	-0.006098	-0.017206	0.0315792	0.
1,107713	-0.148709	0.2233158	0.1256270	0.1610699	-0.314979	-9.719582	0.0046432	-0.000851	-0.009389	7.4979424	101.82800	51.641880	0.9915987	1.8243577	0.6204817	-0.783407	0.0356979	0.7840298	0.6206921	-0.006192	-0.017305	0.0318309	0.
1,207685	-0.146465	0.2255504	0.1244688	0.1635849	-0.321851	-9.717532	0.0043827	-0.000863	-0.009203	7.4980082	101.83200	51.613956	0.9971491	1.8377363	0.6208627	-0.783095	0.0359379	0.7837260	0.6210748	-0.006279	-0.017402	0.0320641	0.
1,307678	-0.146614	0.2198340	0.1207974	0.1598167	-0.322720	-9.700324	0.0040258	-0.001323	-0.009214	7.4979844	101.82800	51.587112	0.9993154	1.8495318	0.6212295	-0.782795	0.0361195	0.7834346	0.6214414	-0.006394	-0.017440	0.0322699	0.
1,407679	-0.146531	0.2232601	0.1220022	0.1651551	-0.325912	-9.702316	0.0040995	-0.001032	-0.009010	7.4986395	101.83200	51.562530	1.0037006	1.8621549	0.6215648	-0.782519	0.0363367	0.7831670	0.6217777	-0.006487	-0.017516	0.0324900	0.
1,507675	-0.146571	0.2209837	0.1232781	0.1634202	-0.326068	-9.694110	0.0039743	-0.001424	-0.008934	7.4961051	101.84000	51.538227	1.0057723	1.8732428	0.6218965	-0.782248	0.0365078	0.7829027	0.6221092	-0.006594	-0.017553	0.0326833	0.
1,607681	-0.145454	0.2220988	0.1220754	0.1666800	-0.317169	-9.684474	0.0039312	-0.001243	-0.008972	7.4958548	101.84000	51.509098	1.0093109	1.8832700	0.6222937	-0.781924	0.0366797	0.7825856	0.6225072	-0.006673	-0.017614	0.0328582	0.
1,707676	-0.149833	0.2233199	0.1218237	0.1771417	-0.323763	-9.685702	0.0036744	-0.001206	-0.009416	7.4955263	101.83200	51.477893	1.0143036	1.8917130	0.6227189	-0.781577	0.0368455	0.7822453	0.6229342	-0.006717	-0.017702	0.0330054	0.
1,807665	-0.148781	0.2210192	0.1219117	0.1770520	-0.323576	-9.676258	0.0035345	-0.001365	-0.009051	7.4952483	101.81500	51.451038	1.0182753	1.8997248	0.6230848	-0.781279	0.0369948	0.7819525	0.6233014	-0.006767	-0.017771	0.0331451	0.
1,907658	-0.146520	0.2221387	0.1270065	0.1742392	-0.328918	-9.672946	0.0033599	-0.001558	-0.009136	7.4930567	101.82800	51.420810	1.0209943	1.9068483	0.6234967	-0.780944	0.0371179	0.7816230	0.6237139	-0.006827	-0.017818	0.0332693	0.
2,007702	-0.148755	0.2210343	0.1256547	0.1751480	-0.321369	-9.664628	0.0028928	-0.001682	-0.008945	7.4930038	101.82800	51.392379	1.0228488	1.9107184	0.6238841	-0.780631	0.0371874	0.7813131	0.6241016	-0.006863	-0.017851	0.0333368	0.
2,107766	-0.147630	0.2221539	0.1256995	0.1780895	-0.319285	-9.660322	0.0025873	-0.001564	-0.009036	7.4936904	101.82800	51.359645	1.0261340	1.9122127	0.6243297	-0.780272	0.0372396	0.7809556	0.6245488	-0.006855	-0.017908	0.0333628	0.
2,20776	-0.147564	0.2266939	0.1206503	0.1737076	-0.324934	-9.668174	0.0027470	-0.001441	-0.009027	7.4928154	101.81500	51.329483	1.0290862	1.9151887	0.6247401	-0.779940	0.0373086	0.7806261	0.6249606	-0.006867	-0.017959	0.0334147	0.
2,307687	-0.147620	0.2221582	0.1269468	0.1758901	-0.319884	-9.657933	0.0023824	-0.001771	-0.009123	7.4923834	101.82800	51.296260	1.0309641	1.9156907	0.6251923	-0.779577	0.0373320	0.7802633	0.6254134	-0.006869	-0.017992	0.0334234	0.
2,407668	-0.148630	0.2267336	0.1255814	0.1702756	-0.320859	-9.656055	0.0020902	-0.001817	-0.008926	7.4895262	101.82800	51.264778	1.0311546	1.9143356	0.6256209	-0.779234	0.0373118	0.7799196	0.6258419	-0.006872	-0.017996	0.0333998	0.
2,50766	-0.148687	0.2233234	0.1281202	0.1735381	-0.325380	-9.663486	0.0020196	-0.001883	-0.009097	7.4881210	101.82800	51.230686	1.0312197	1.9135152	0.6260849	-0.778862	0.0372973	0.7795472	0.6263056	-0.006884	-0.017997	0.0333855	0.

Figure 65. Snapshot of data acquired from course's lessons tab

Above is the data taken from the csv file provided by the course's lessons tab. When comparing the data we see that the values are within range of each other. The ICM on our senseHAT does not provide data on temperature and pressure, to get that we would need to use another sensor simultaneously. The SenseHAT also does not provide DCIM data which can't be compared to the rawData file above.

Data Captured from the IMU:

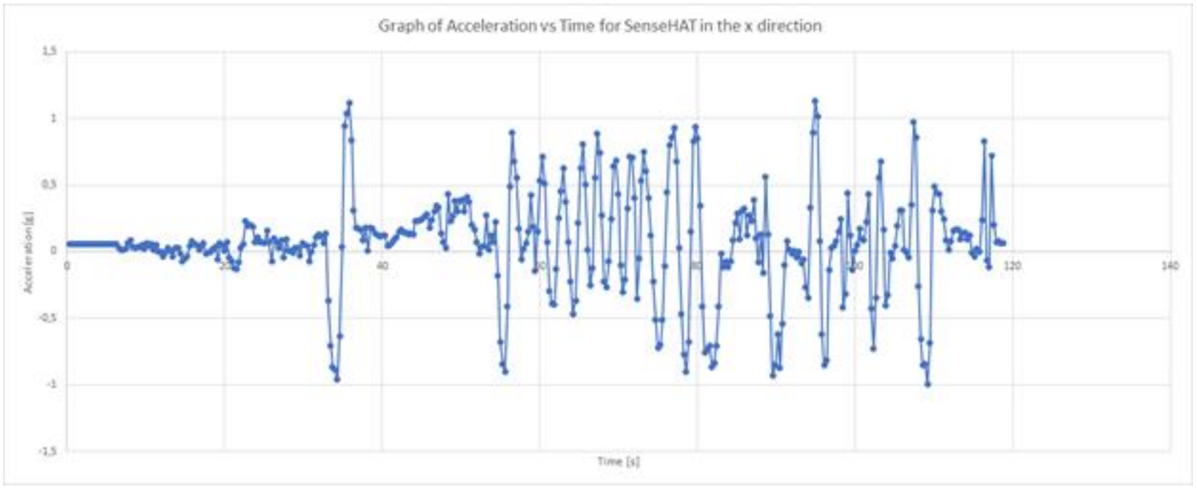


Figure 66. Acceleration vs Time in x direction

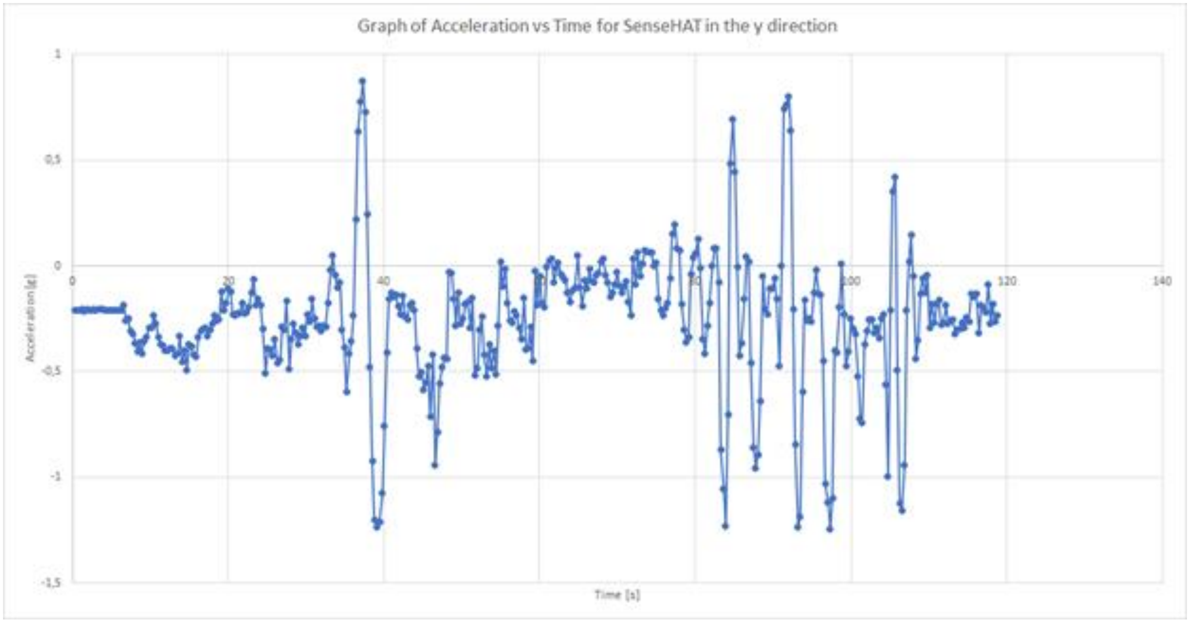


Figure 67. Acceleration vs Time in y direction

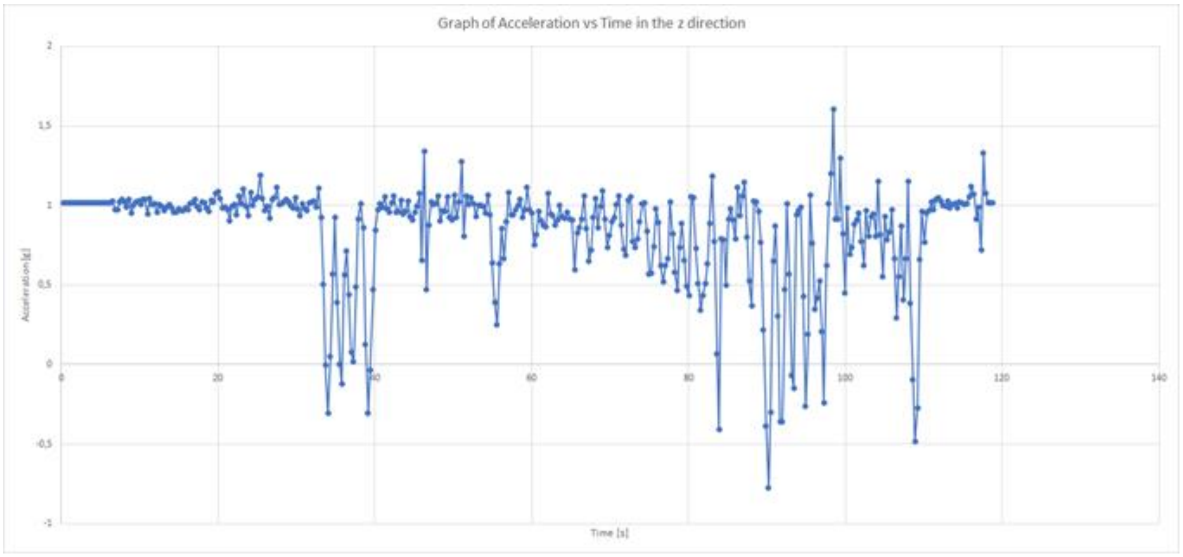


Figure 68. Acceleration vs Time in z direction

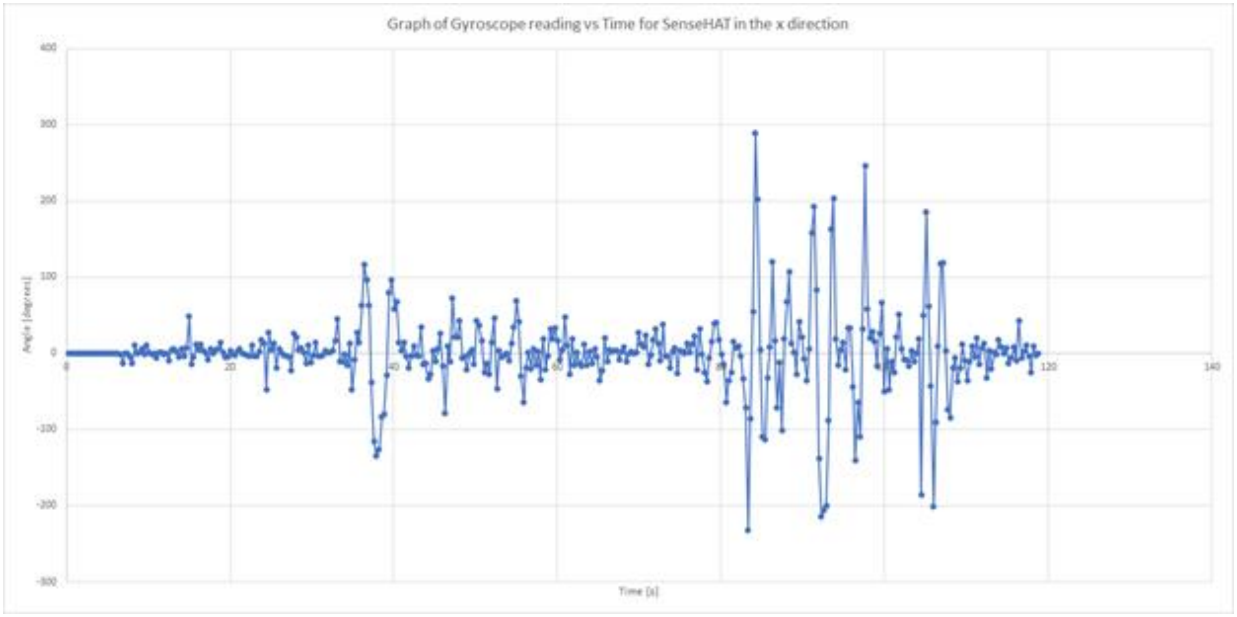


Figure 69. Gyroscope vs Time in x direction

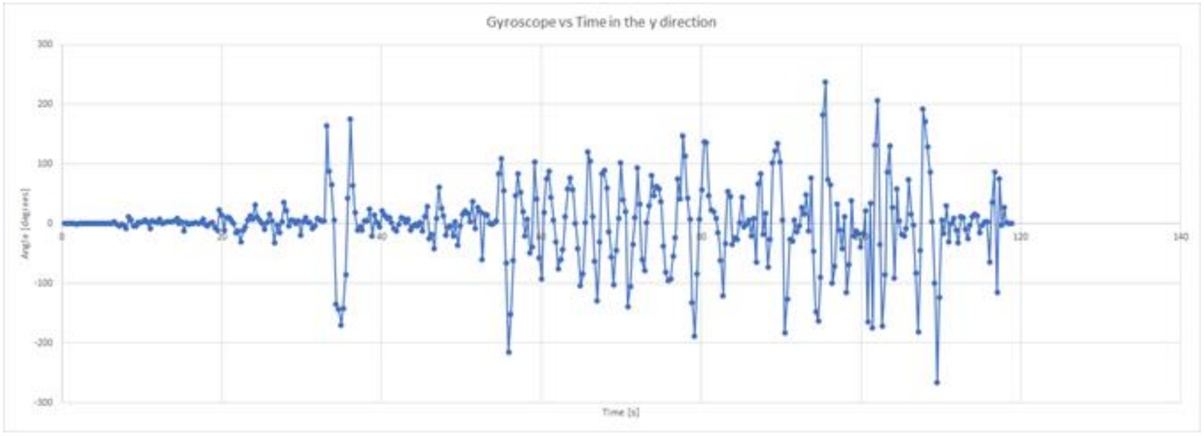


Figure 70. Gyroscope vs Time in y direction

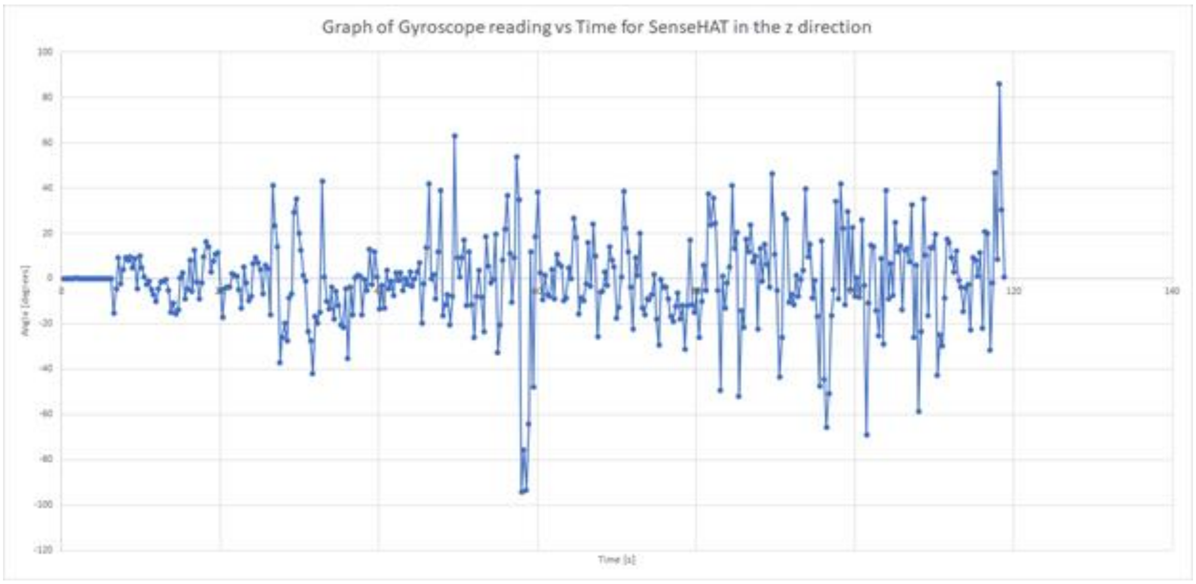


Figure 71. Gyroscope vs Time in z direction

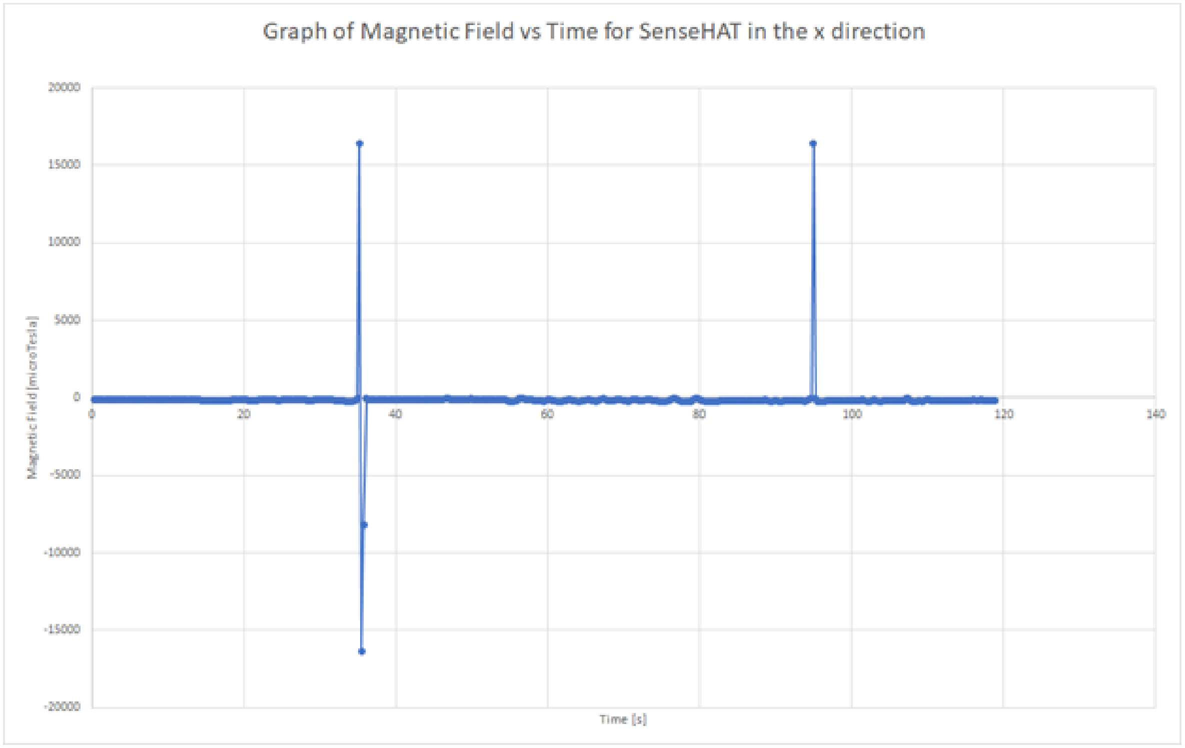


Figure 72. Magnetic Field vs Time in x direction

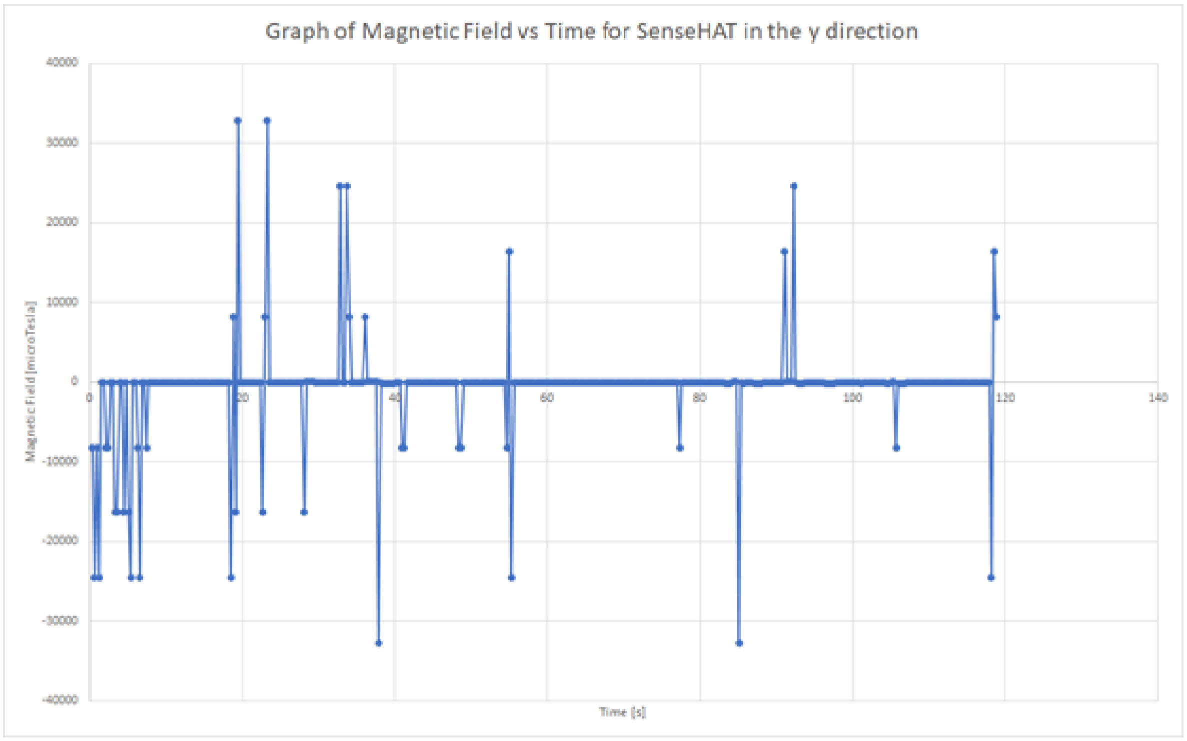


Figure 73. Magnetic Field vs Time in y direction

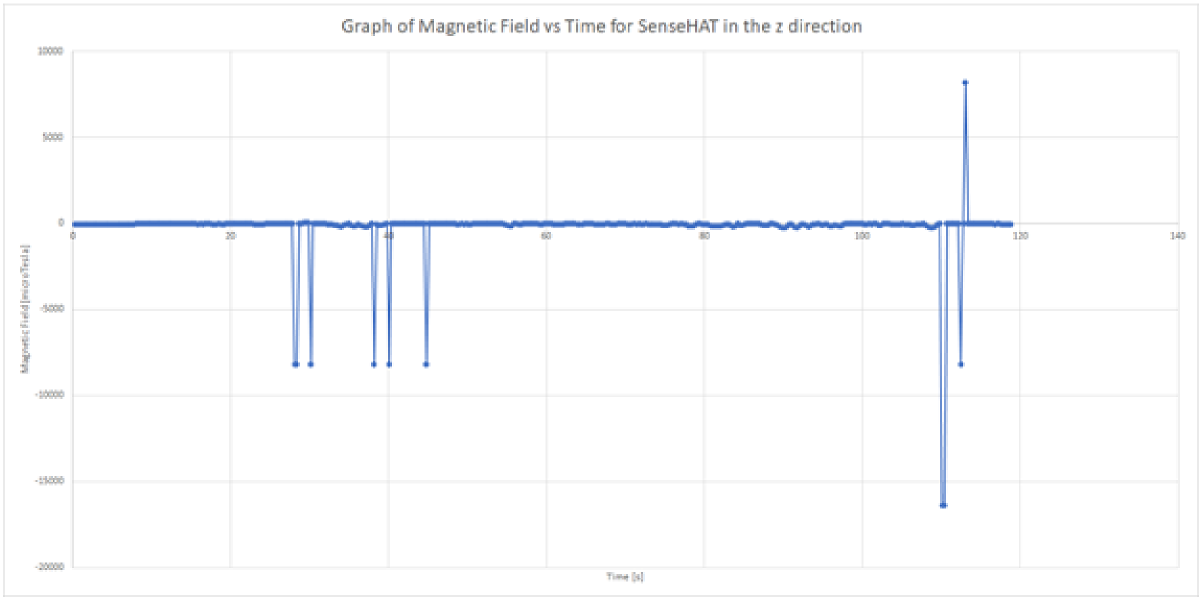


Figure 74. Magnetic Field vs Time in z direction

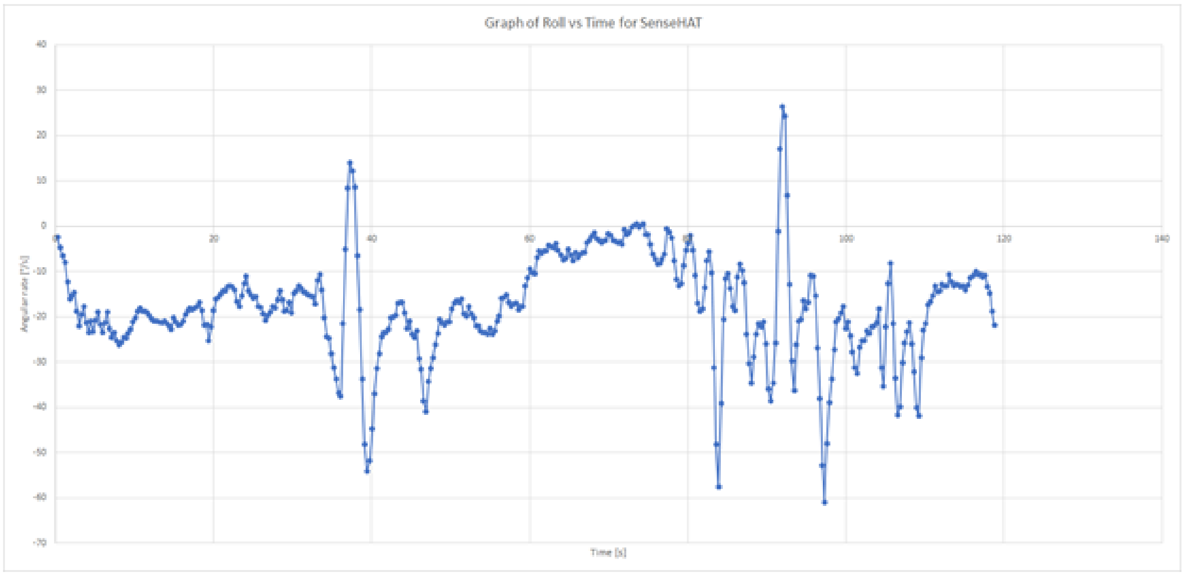


Figure 75. Roll vs Time

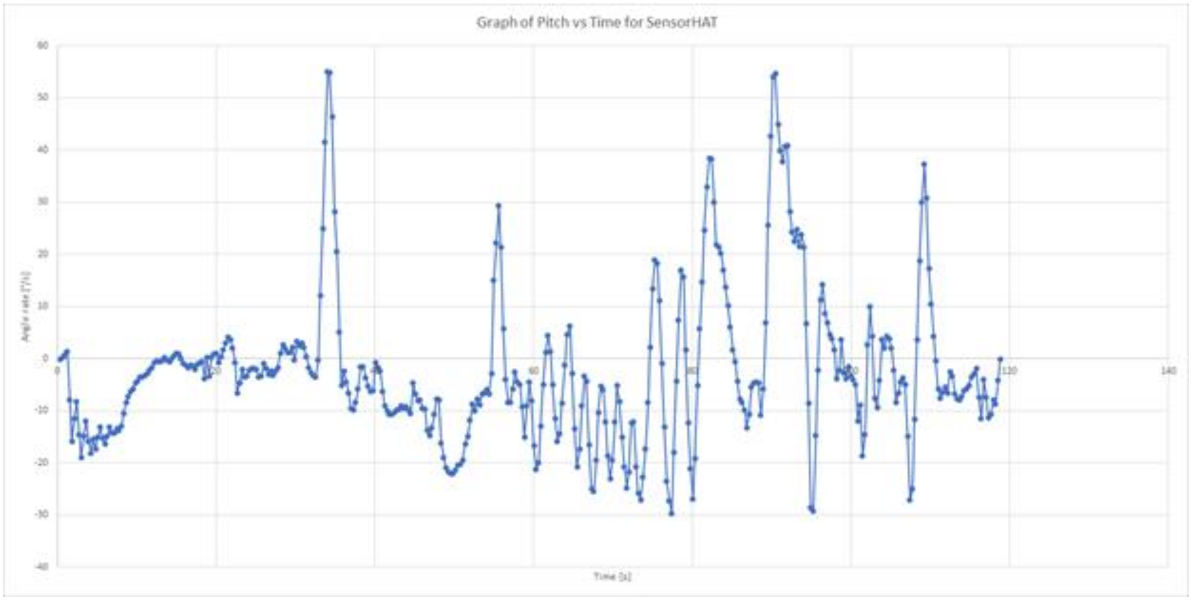


Figure 76. Pitch vs Time

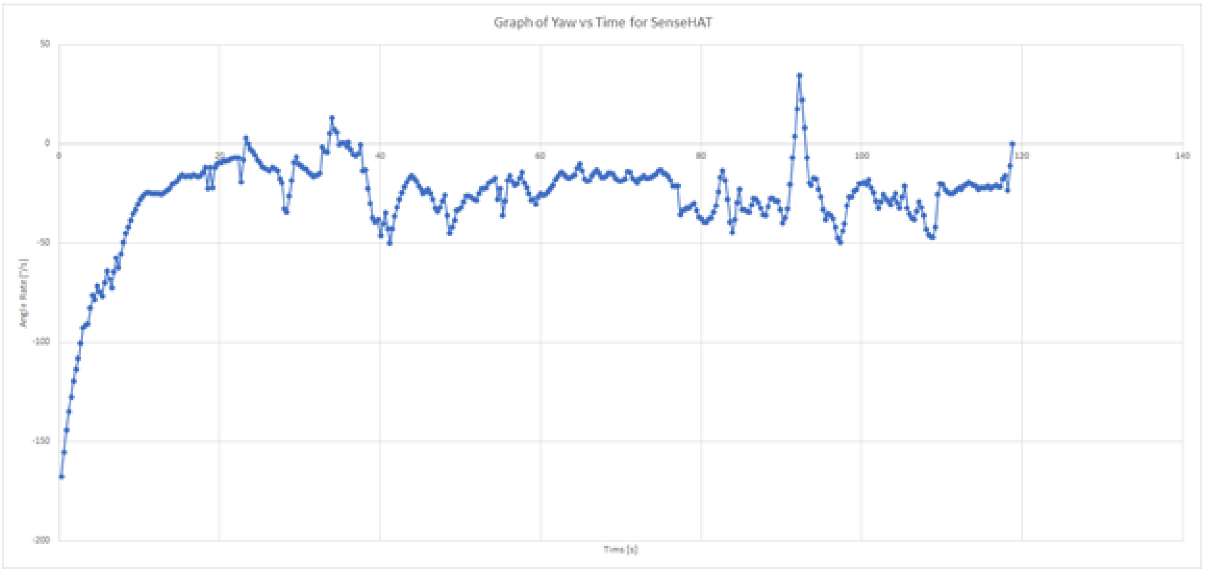
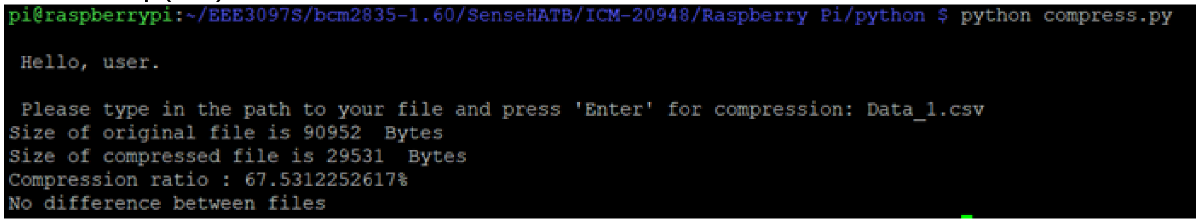


Figure 77. Yaw vs Time

Overall, the program is functioning as expected as it is meeting all the requirements of the project. The IMU data is being read correctly and then converted into a csv file that can be read by our compression and encryption program. Once the file has been compressed and encryption it gets sent to the relevant user.

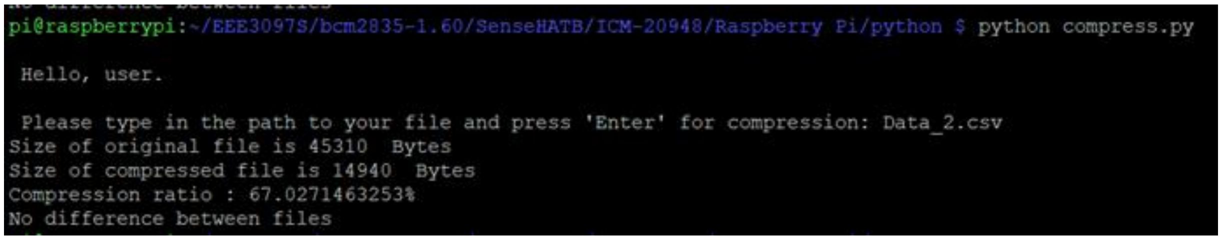
Compression:

Worked on by Greg Da Silva (DSLGRE001)
4 files named Data_1.csv to Data_4.csv were made to store readings from the SenseHAT, these files were then used in the compression algorithm to run our tests on the data.
The following outputs were observed after running the compression algorithm on the various data recorded from the ICM on the SenseHAT:
Time.sleep(0.1)



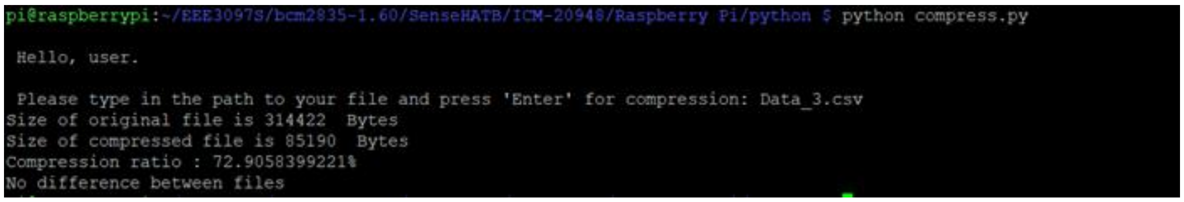
```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: Data_1.csv
Size of original file is 90952  Bytes
Size of compressed file is 29531  Bytes
Compression ratio : 67.5312252617%
No difference between files
```

Figure 78. Output from running compression algorithm



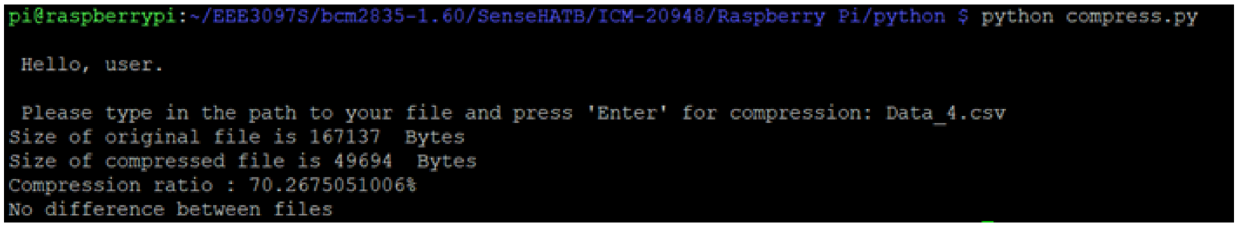
```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: Data_2.csv
Size of original file is 45310  Bytes
Size of compressed file is 14940  Bytes
Compression ratio : 67.0271463253%
No difference between files
```

Figure 79. Output from running compression algorithm



```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: Data_3.csv
Size of original file is 314422  Bytes
Size of compressed file is 85190  Bytes
Compression ratio : 72.9058399221%
No difference between files
```

Figure 80. Output from running compression algorithm

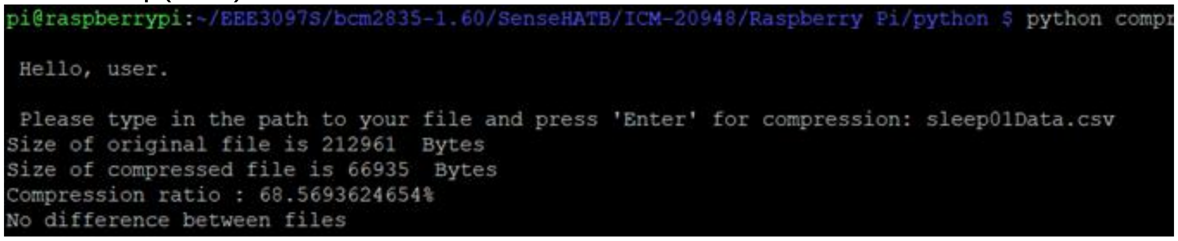


```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python compress.py

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: Data_4.csv
Size of original file is 167137  Bytes
Size of compressed file is 49694  Bytes
Compression ratio : 70.2675051006%
No difference between files
```

Figure 81. Output from running compression algorithm

Varying the data by recording for longer/shorter times changed the compression ratio of the compression algorithm with a best case of 72.9% for a file of 314422 Bytes and worst case of 67.02% for a file of 45310 Bytes. From this we conclude that the compression algorithm is successful on the ICM data provided by the SenseHAT. Every compressed file was tested for lost data and all tests returned with “No difference between files” indicating that the decompressed file and original file had no differences, therefore no data was lost in the process of compressing the files.

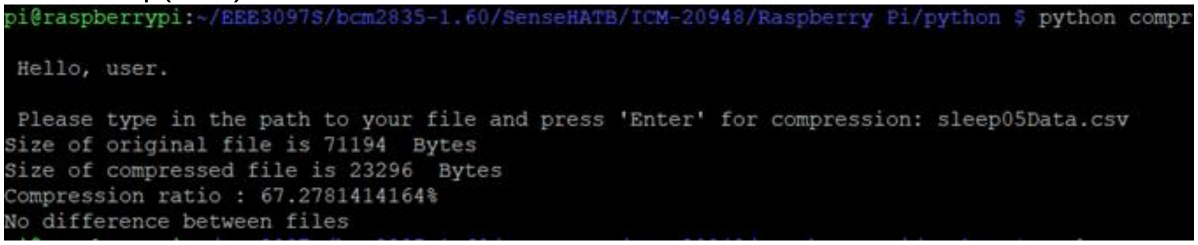
Time.sleep(0.01):



```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python compr

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: sleep01Data.csv
Size of original file is 212961  Bytes
Size of compressed file is 66935  Bytes
Compression ratio : 68.5693624654%
No difference between files
```

Figure 82. Output from running compression algorithm

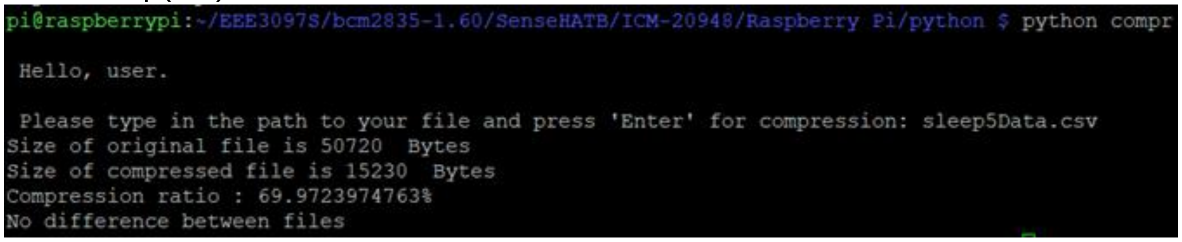
Time.sleep(0.05)



```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python compr

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: sleep05Data.csv
Size of original file is 71194  Bytes
Size of compressed file is 23296  Bytes
Compression ratio : 67.2781414164%
No difference between files
```

Figure 83. Output from running compression algorithm

Time.sleep(0.5):



```
pi@raspberrypi:~/EEE3097S/bcm2835-1.60/SenseHATB/ICM-20948/Raspberry Pi/python $ python compr

 Hello, user.

 Please type in the path to your file and press 'Enter' for compression: sleep5Data.csv
Size of original file is 50720  Bytes
Size of compressed file is 15230  Bytes
Compression ratio : 69.9723974763%
No difference between files
```

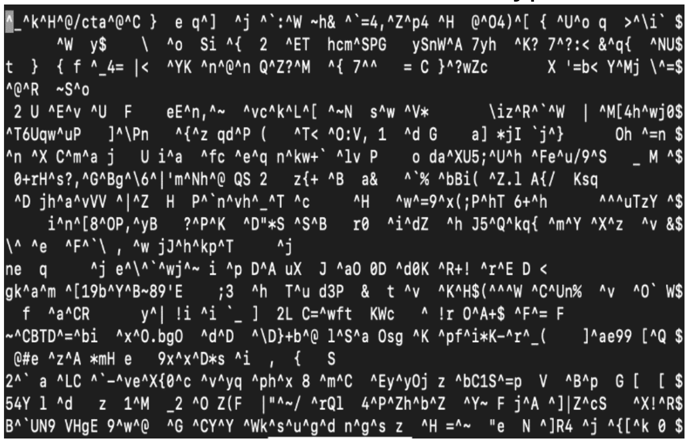
Figure 84. Output from running compression algorithm

Encryption

Worked on by ADMALI004

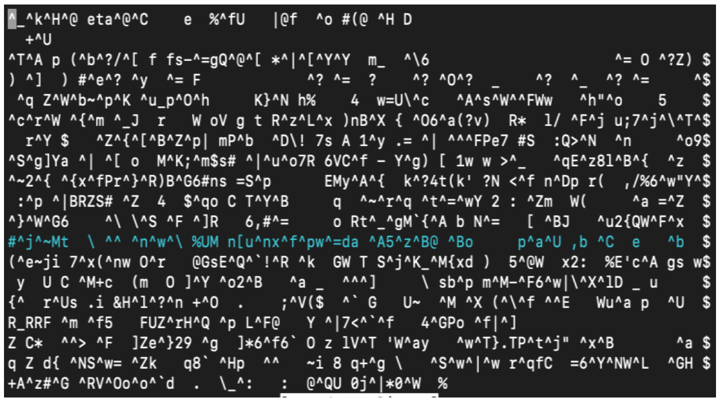
The encryption was successfully executed and produced the expected results. The data items that were generated in the file were encrypted and thus could not be understood. The compressed file was encrypted and when trying to decompress this file it gave an error. This was expected as it is an encrypted file thus it cannot be opened without the generated key. The compressed file was then decrypted and decompressed and it was successfully. The decrypted and decompressed file was the rawData.csv file which is expected since that is the file that was used at the beginning of the program. The same procedure was followed for the four data csv files above that were extracted from the SenseHAT(B). At the end of decrypting and decompressing the files that were produced for each test were Data_1.csv, Data_2.csv, Data_3.csv and Data_4.csv which are all renamed to rawData.csv. This proves that the encryption program was executed successfully.

The screenshots below are the encrypted files for each data file:



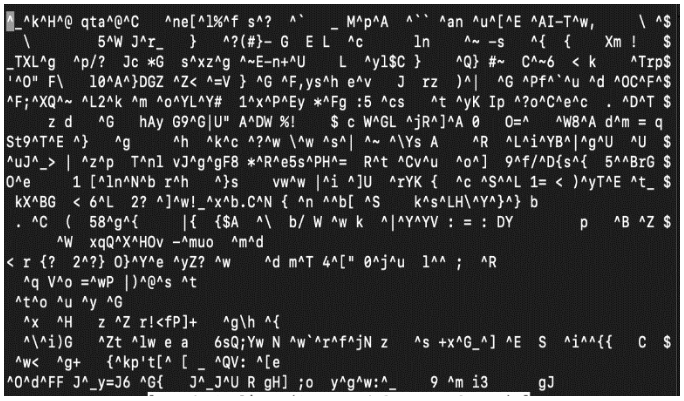
```
^_^k^H^@/cta^@^C }  e q^]  ^j ^`:^W ~h& ^`=4,^Z^p4 ^H  @^O4)^[ { ^U^o q  >^\i` $
     ^W  y$    \  ^o  Si ^{  2  ^ET  hcm^SPG   ySnW^A 7yh  ^K? 7^?:< &^q{  ^NU$
t  }  { f ^_4= |<  ^YK ^n^@^n Q^Z?^M  ^{ 7^^   = C }^?wZc       X '=b< Y^Mj \^=$
^@^R  ~S^o
 2 U ^E^v ^U  F    eE^n,^~  ^vc^k^L^[ ^~N  s^w ^V*       \iz^R^`^W  | ^M[4h^wj0$
^T6Uqw^uP   ]^\Pn   ^{^z qd^P (   ^T< ^O:V, 1  ^d G    a] *jI `j^}      Oh ^=n $
^n ^X C^m^a j   U i^a  ^fc ^e^q n^kw+` ^lv P    o da^XU5;^U^h ^Fe^u/9^S   _ M ^$
 0+rH^s?,^G^Bg^\6^|'m^Nh^@ QS 2   z{+ ^B  a&   ^`% ^bBi( ^Z.l A{/  Ksq
 ^D jh^a^vVV ^|^Z  H  P^`n^vh^_^T ^c     ^H   ^w^=9^x(;P^hT 6+^h      ^^^uTzY ^$
     i^n^[8^OP,^yB   ?^P^K  ^D"*S ^S^B   r0  ^i^dZ  ^h J5^Q^kq{ ^m^Y ^X^z  ^v &$
\^ ^e  ^F^`\ , ^w jJ^h^kp^T     ^j
ne  q     ^j e^\^`^wj^~ i ^p D^A uX  J ^aO 0D ^d0K ^R+! ^r^E D <
gk^a^m ^[19b^Y^B~89'E    ;3  ^h  T^u d3P  &  t ^v  ^K^H$(^^^W ^C^Un%  ^v  ^O` W$
  f  ^a^CR       y^| !i ^i `_ ]  2L C=^wft  KWc   ^ !r O^A+$ ^F^= F
~^CBTD^=^bi  ^x^O.bgO  ^d^D  ^\D}+b^@ l^S^a Osg ^K ^pf^i*K-^r^_(    ]^ae99 [^Q $
 @#e ^z^A *mH e   9x^x^D*s ^i  ,  {   S
2^` a ^LC ^`-^ve^X{0^c ^v^yq ^ph^x 8 ^m^C  ^Ey^yOj z ^bC1S^=p  V  ^B^p  G [  [ $
54Y l ^d   z  1^M  _2 ^O Z(F  |"^~/ ^rQl  4^P^Zh^b^Z  ^Y~ F j^A ^]|Z^cS   ^X!^R$
B^`UN9 VHgE 9^w^@  ^G ^CY^Y ^Wk^s^u^g^d n^g^s z  ^H =^~  "e  N ^]R4 ^j ^{[^k 0 $
```

Figure 85. Encrypted Data_4.csv file



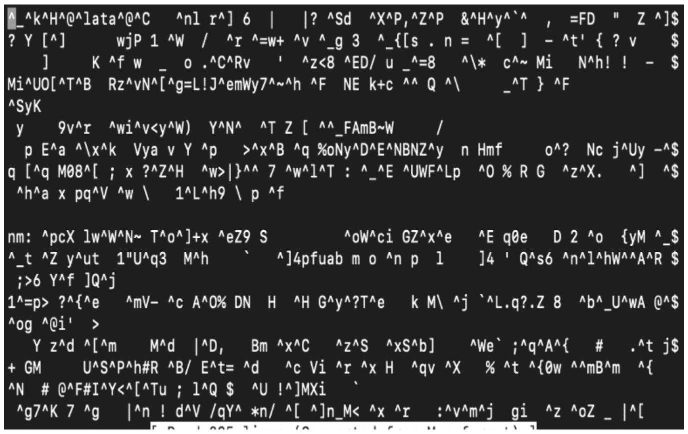
```
^_^k^H^@ eta^@^C    e  %^fU   |@f  ^o #(@ ^H D
  +^U
^T^A p (^b^?/^[ f fs-^=gQ^@^[ *^|^[^Y^Y  m_  ^\6                      ^= O ^?Z) $
) ^]  ) #^e^? ^y  ^= F             ^? ^=  ?    ^? ^O^?  _    ^?  ^_  ^? ^=     ^$
 ^q Z^W^b~^p^K ^u_p^O^h     K}^N h%    4  w=U\^c   ^A^s^W^^FWw   ^h"^o    5    $
^c^r^W ^{^m ^_J  r   W oV g t R^z^L^x )nB^X { ^O6^a(?v)  R*  l/ ^F^j u;7^j^\^T^$
  r^Y $   ^Z^{^[^B^Z^p| mP^b  ^D\! 7s A 1^y .= ^| ^^^FPe7 #S  :Q>^N  ^n     ^o9$
^S^g]Ya ^| ^[ o  M^K;^m$s# ^|^u^o7R 6VC^f - Y^g) [ 1w w >^_   ^qE^z8l^B^{  ^z  $
^~2^{ ^{x^fPr^}^R)B^G6#ns =S^p      EMy^A^{  k^?4t(k' ?N <^f n^Dp r(  ,/%6^w"Y^$
 :^p ^|BRZS# ^Z  4  $^qo C T^Y^B     q  ^~^r^q ^t^=^wY 2 : ^Zm  W(     ^a =^Z  $
^}^W^G6    ^\ \^S ^F ^]R   6,#^=     o Rt^_^gM`{^A b N^=   [ ^BJ   ^u2{QW^F^x  $
#^j^~Mt  \ ^^ ^n^w^\ %UM n[u^nx^f^pw^=da ^A5^z^B@ ^Bo     p^a^U ,b ^C  e   ^b  $
(^e~ji 7^x(^nw O^r   @GsE^Q^`!^R ^k  GW T S^j^K_^M{xd )  5^@W  x2:  %E'c^A gs w$
 y  U C ^M+c  (m  O ]^Y ^o2^B   ^a _  ^^^]     \ sb^p m^M-^F6^w|\^X^lD _ u     $
{^  r^Us .i &H^l^?^n +^O  .    ;^V($  ^` G   U~  ^M ^X (^\^f ^^E   Wu^a p  ^U  $
R_RRF ^m ^f5   FUZ^rH^Q ^p L^F@   Y ^|7<^`^f   4^GPo ^f|^]
Z C*  ^^> ^F  ]Ze^}29 ^g  ]*6^f6` O z lV^T 'W^ay   ^w^T}.TP^t^j" ^x^B       ^a $
q Z d{ ^NS^w= ^Zk   q8` ^Hp  ^^   ~i 8 q+^g \   ^S^w^|^w r^qfC  =6^Y^NW^L  ^GH $
+A^z#^G ^RV^Oo^o^`d  .  \_^:   :  @^QU 0j^|*0^W  %
```

Figure 86. Encrypted Data_3.csv file



```
^_^k^H^@ qta^@^C   ^ne[^l%^f s^?  ^`   _ M^p^A  ^`` ^an ^u^[^E ^AI-T^w,     \ ^$
  \        5^W J^r_   }   ^?(#}- G  E L  ^c      ln    ^~ -s   ^{  {    Xm !   $
_TXL^g  ^p/?  Jc *G  s^xz^g ^~E-n+^U    L  ^yl$C }    ^Q} #~  C^~6  < k    ^Trp$
'^O" F\   l0^A^}DGZ ^Z< ^=V } ^G ^F,ys^h e^v   J  rz  )^|  ^G ^Pf^`^u ^d ^OC^F^$
^F;^XQ^~ ^L2^k ^m ^o^YL^Y#  1^x^P^Ey *^Fg :5 ^cs   ^t ^yK Ip ^?o^C^e^c  . ^D^T $
     z d   ^G   hAy G9^G|U" A^DW %!    $ c W^GL ^jR^]^A 0   O=^   ^W8^A d^m = q
St9^T^E ^}   ^g     ^h  ^k^c ^?^w \^w ^s^| ^~ ^\Ys A    ^R  ^L^i^YB^|^g^U  ^U  $
^uJ^_> | ^z^p  T^nl vJ^g^gF8 *^R^e5s^PH^=  R^t ^Cv^u  ^o^]  9^f/^D{s^{  5^^BrG $
O^e     1 [^ln^N^b r^h   ^}s    vw^w |^i ^]U  ^rYK {  ^c ^S^^L 1= < )^yT^E ^t_ $
 kX^BG  < 6^L  2? ^]^w!_^x^b.C^N { ^n ^^b[ ^S    k^s^LH\^Y^}^} b
 . ^C  (  58^g^{     |{  {$A  ^\  b/ W ^w k  ^|^Y^YV : = : DY        p   ^B ^Z $
     ^W  xqQ^X^HOv -^muo  ^m^d
< r {?  2^?} O}^Y^e ^yZ? ^w    ^d m^T 4^[" 0^j^u  l^^ ;  ^R
  ^q V^o =^wP |)^@^s ^t
 ^t^o ^u ^y ^G
  ^x  ^H   z ^Z r!<fP]+   ^g\h ^{
  ^\^i)G   ^Zt ^lw e a   6sQ;Yw N ^w`^r^f^jN z   ^s +x^G_^] ^E  S  ^i^^{{   C  $
 ^w<  ^g+   {^kp't[^ [ _ ^QV: ^[e
^O^d^FF J^_y=J6 ^G{   J^_J^U R gH] ;o  y^g^w:^_    9 ^m i3      gJ
```

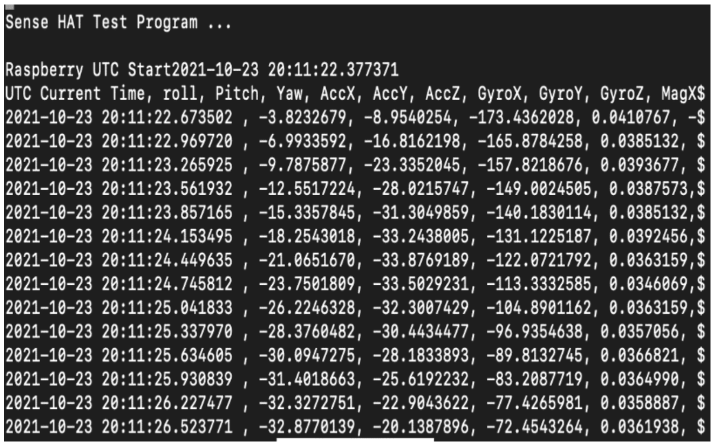
Figure 87. Encrypted Data_2.csv file



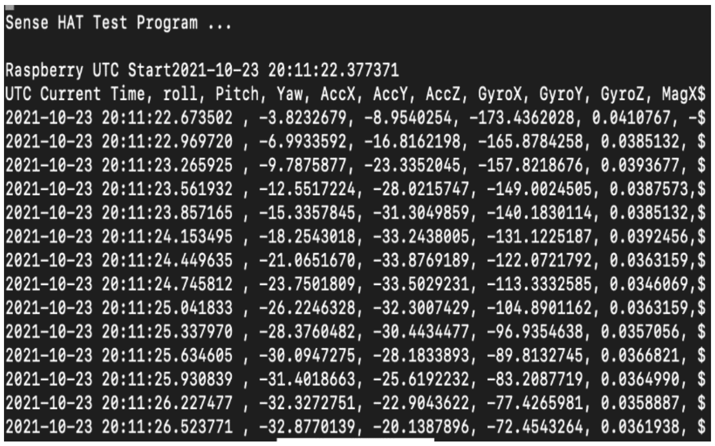
```
^_^k^H^@^lata^@^C   ^nl r^] 6  |   |? ^Sd  ^X^P,^Z^P  &^H^y^`^  ,  =FD  "  Z ^]$
? Y [^]      wjP 1 ^W  /  ^r ^=w+ ^v ^_g 3  ^_{[s . n =  ^[  ]  - ^t' { ? v    $
    ]     K ^f w  _  o .^C^Rv   '  ^z<8 ^ED/ u _^=8   ^\*  c^~ Mi   N^h! !  -  $
Mi^UO[^T^B  Rz^vN^[^g=L!J^emWy7^~^h ^F  NE k+c ^^ Q ^\     _^T } ^F
^SyK
 y    9v^r  ^wi^v<y^W)  Y^N^  ^T Z [ ^^_FAmB~W     /
  p E^a ^\x^k  Vya v Y ^p   >^x^B ^q %oNy^D^E^NBNZ^y  n Hmf     o^?  Nc j^Uy -^$
q [^q M08^[ ; x ?^Z^H  ^w>|}^^ 7 ^w^l^T : ^_^E ^UWF^Lp  ^O % R G  ^z^X.   ^]  ^$
 ^h^a x pq^V ^w \   1^L^h9 \ p ^f

nm: ^pcX lw^W^N~ T^o^]+x ^eZ9 S         ^oW^ci GZ^x^e   ^E q0e   D 2 ^o  {yM ^_$
^_t ^Z y^ut  1"U^q3  M^h    `   ^]4pfuab m o ^n p  l    ]4 ' Q^s6 ^n^l^hW^^A^R $
 ;>6 Y^f ]Q^j
1^=p> ?^{^e   ^mV- ^c A^O% DN  H  ^H G^y^?T^e   k M\ ^j `^L.q?.Z 8  ^b^_U^wA @^$
^og ^@i'  >
   Y z^d ^[^m    M^d  |^D,   Bm ^x^C   ^z^S  ^xS^b]    ^We` ;^q^A^{   #   .^t j$
+ GM     U^S^P^h#R ^B/ E^t= ^d   ^c Vi ^r ^x H  ^qv ^X   % ^t ^{0w ^^mB^m  ^{
^N  # @^F#I^Y<^[^Tu ; l^Q $  ^U !^]MXi   `
 ^g7^K 7 ^g   |^n ! d^V /qY^ *n/ ^[ ^]n_M< ^x ^r   :^v^m^j  gi  ^z ^oZ _ |^[
```

Figure 88. Encrypted Data_1.csv file

The code above was run on each data file used in the compression to produce the following results:

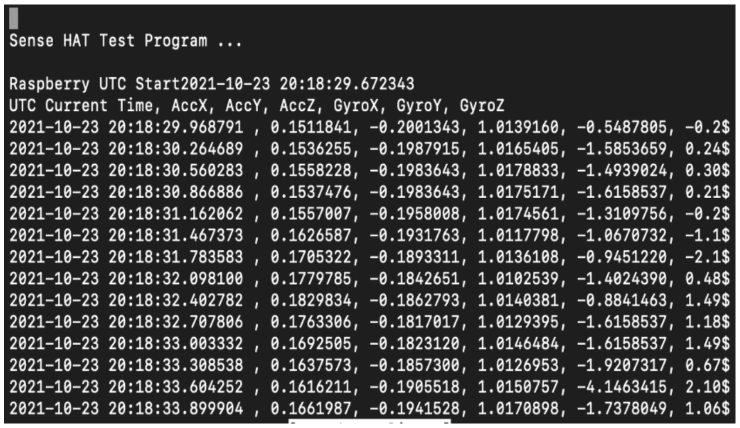


```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:11:22.377371
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX$
2021-10-23 20:11:22.673502 , -3.8232679, -8.9540254, -173.4362028, 0.0410767, -$
2021-10-23 20:11:22.969720 , -6.9933592, -16.8162198, -165.8784258, 0.0385132, $
2021-10-23 20:11:23.265925 , -9.7875877, -23.3352045, -157.8218676, 0.0393677, $
2021-10-23 20:11:23.561932 , -12.5517224, -28.0215747, -149.0024505, 0.0387573,$
2021-10-23 20:11:23.857165 , -15.3357845, -31.3049859, -140.1830114, 0.0385132,$
2021-10-23 20:11:24.153495 , -18.2543018, -33.2438005, -131.1225187, 0.0392456,$
2021-10-23 20:11:24.449635 , -21.0651670, -33.8769189, -122.0721792, 0.0363159,$
2021-10-23 20:11:24.745812 , -23.7501809, -33.5029231, -113.3332585, 0.0346069,$
2021-10-23 20:11:25.041833 , -26.2246328, -32.3007429, -104.8901162, 0.0363159,$
2021-10-23 20:11:25.337970 , -28.3760482, -30.4434477, -96.9354638, 0.0357056, $
2021-10-23 20:11:25.634605 , -30.0947275, -28.1833893, -89.8132745, 0.0366821, $
2021-10-23 20:11:25.930839 , -31.4018663, -25.6192232, -83.2087719, 0.0364990, $
2021-10-23 20:11:26.227477 , -32.3272751, -22.9043622, -77.4265981, 0.0358887, $
2021-10-23 20:11:26.523771 , -32.8770139, -20.1387896, -72.4543264, 0.0361938, $
```



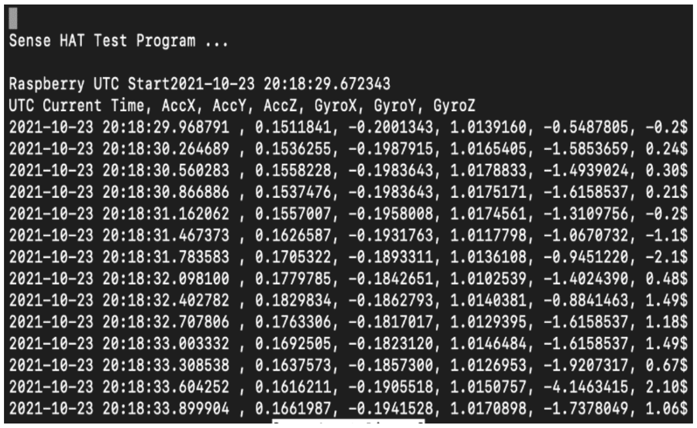
```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:11:22.377371
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX$
2021-10-23 20:11:22.673502 , -3.8232679, -8.9540254, -173.4362028, 0.0410767, -$
2021-10-23 20:11:22.969720 , -6.9933592, -16.8162198, -165.8784258, 0.0385132, $
2021-10-23 20:11:23.265925 , -9.7875877, -23.3352045, -157.8218676, 0.0393677, $
2021-10-23 20:11:23.561932 , -12.5517224, -28.0215747, -149.0024505, 0.0387573,$
2021-10-23 20:11:23.857165 , -15.3357845, -31.3049859, -140.1830114, 0.0385132,$
2021-10-23 20:11:24.153495 , -18.2543018, -33.2438005, -131.1225187, 0.0392456,$
2021-10-23 20:11:24.449635 , -21.0651670, -33.8769189, -122.0721792, 0.0363159,$
2021-10-23 20:11:24.745812 , -23.7501809, -33.5029231, -113.3332585, 0.0346069,$
2021-10-23 20:11:25.041833 , -26.2246328, -32.3007429, -104.8901162, 0.0363159,$
2021-10-23 20:11:25.337970 , -28.3760482, -30.4434477, -96.9354638, 0.0357056, $
2021-10-23 20:11:25.634605 , -30.0947275, -28.1833893, -89.8132745, 0.0366821, $
2021-10-23 20:11:25.930839 , -31.4018663, -25.6192232, -83.2087719, 0.0364990, $
2021-10-23 20:11:26.227477 , -32.3272751, -22.9043622, -77.4265981, 0.0358887, $
2021-10-23 20:11:26.523771 , -32.8770139, -20.1387896, -72.4543264, 0.0361938, $
```

Figure 89. Data from decompressed file in encryption *Figure 89. Data from Data_1.csv file*



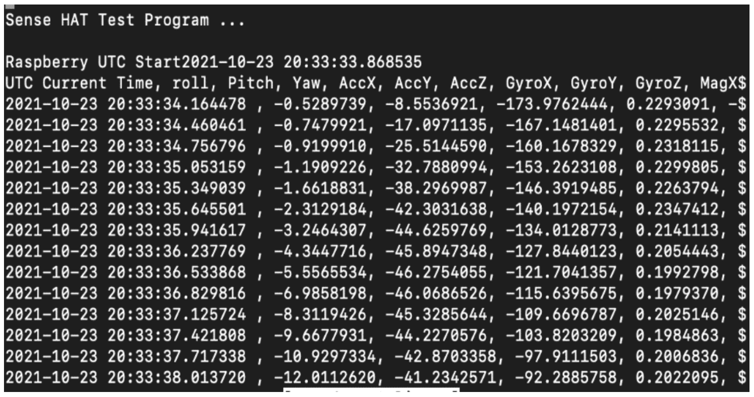
```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:18:29.672343
UTC Current Time, AccX, AccY, AccZ, GyroX, GyroY, GyroZ
2021-10-23 20:18:29.968791 , 0.1511841, -0.2001343, 1.0139160, -0.5487805, -0.2$
2021-10-23 20:18:30.264689 , 0.1536255, -0.1987915, 1.0165405, -1.5853659, 0.24$
2021-10-23 20:18:30.560283 , 0.1558228, -0.1983643, 1.0178833, -1.4939024, 0.30$
2021-10-23 20:18:30.866886 , 0.1537476, -0.1983643, 1.0175171, -1.6158537, 0.21$
2021-10-23 20:18:31.162062 , 0.1557007, -0.1958008, 1.0174561, -1.3109756, -0.2$
2021-10-23 20:18:31.467373 , 0.1626587, -0.1931763, 1.0117798, -1.0670732, -1.1$
2021-10-23 20:18:31.783583 , 0.1705322, -0.1893311, 1.0136108, -0.9451220, -2.1$
2021-10-23 20:18:32.098100 , 0.1779785, -0.1842651, 1.0102539, -1.4024390, 0.48$
2021-10-23 20:18:32.402782 , 0.1829834, -0.1862793, 1.0140381, -0.8841463, 1.49$
2021-10-23 20:18:32.707806 , 0.1763306, -0.1817017, 1.0129395, -1.6158537, 1.18$
2021-10-23 20:18:33.003332 , 0.1692505, -0.1823120, 1.0146484, -1.6158537, 1.49$
2021-10-23 20:18:33.308538 , 0.1637573, -0.1857300, 1.0126953, -1.9207317, 0.67$
2021-10-23 20:18:33.604252 , 0.1616211, -0.1905518, 1.0150757, -4.1463415, 2.10$
2021-10-23 20:18:33.899904 , 0.1661987, -0.1941528, 1.0170898, -1.7378049, 1.06$
```

Figure 90. Data from decompressed file in encryption



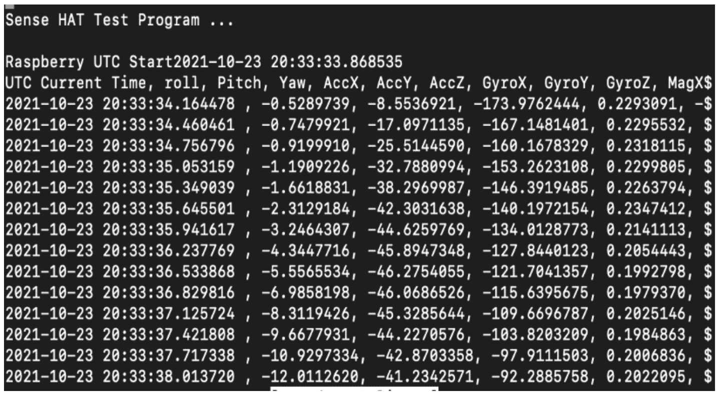
```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:18:29.672343
UTC Current Time, AccX, AccY, AccZ, GyroX, GyroY, GyroZ
2021-10-23 20:18:29.968791 , 0.1511841, -0.2001343, 1.0139160, -0.5487805, -0.2$
2021-10-23 20:18:30.264689 , 0.1536255, -0.1987915, 1.0165405, -1.5853659, 0.24$
2021-10-23 20:18:30.560283 , 0.1558228, -0.1983643, 1.0178833, -1.4939024, 0.30$
2021-10-23 20:18:30.866886 , 0.1537476, -0.1983643, 1.0175171, -1.6158537, 0.21$
2021-10-23 20:18:31.162062 , 0.1557007, -0.1958008, 1.0174561, -1.3109756, -0.2$
2021-10-23 20:18:31.467373 , 0.1626587, -0.1931763, 1.0117798, -1.0670732, -1.1$
2021-10-23 20:18:31.783583 , 0.1705322, -0.1893311, 1.0136108, -0.9451220, -2.1$
2021-10-23 20:18:32.098100 , 0.1779785, -0.1842651, 1.0102539, -1.4024390, 0.48$
2021-10-23 20:18:32.402782 , 0.1829834, -0.1862793, 1.0140381, -0.8841463, 1.49$
2021-10-23 20:18:32.707806 , 0.1763306, -0.1817017, 1.0129395, -1.6158537, 1.18$
2021-10-23 20:18:33.003332 , 0.1692505, -0.1823120, 1.0146484, -1.6158537, 1.49$
2021-10-23 20:18:33.308538 , 0.1637573, -0.1857300, 1.0126953, -1.9207317, 0.67$
2021-10-23 20:18:33.604252 , 0.1616211, -0.1905518, 1.0150757, -4.1463415, 2.10$
2021-10-23 20:18:33.899904 , 0.1661987, -0.1941528, 1.0170898, -1.7378049, 1.06$
```

Figure 90. Data from Data_2.csv file



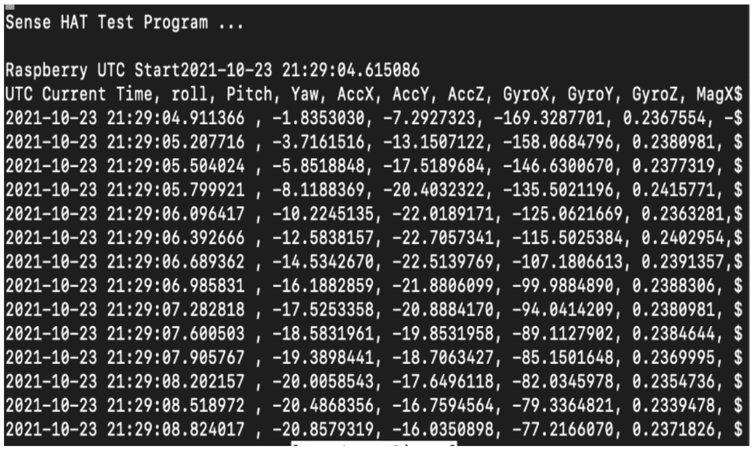
```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:33:33.868535
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX$
2021-10-23 20:33:34.164478 , -0.5289739, -8.5536921, -173.9762444, 0.2293091, -$
2021-10-23 20:33:34.460461 , -0.7479921, -17.0971135, -167.1481401, 0.2295532, $
2021-10-23 20:33:34.756796 , -0.9199910, -25.5144590, -160.1678329, 0.2318115, $
2021-10-23 20:33:35.053159 , -1.1909226, -32.7880994, -153.2623108, 0.2299805, $
2021-10-23 20:33:35.349039 , -1.6618831, -38.2969987, -146.3919485, 0.2263794, $
2021-10-23 20:33:35.645501 , -2.3129184, -42.3031638, -140.1972154, 0.2347412, $
2021-10-23 20:33:35.941617 , -3.2464307, -44.6259769, -134.0128773, 0.2141113, $
2021-10-23 20:33:36.237769 , -4.3447716, -45.8947348, -127.8440123, 0.2054443, $
2021-10-23 20:33:36.533868 , -5.5565534, -46.2754055, -121.7041357, 0.1992798, $
2021-10-23 20:33:36.829816 , -6.9858198, -46.0686526, -115.6395675, 0.1979370, $
2021-10-23 20:33:37.125724 , -8.3119426, -45.3285644, -109.6696787, 0.2025146, $
2021-10-23 20:33:37.421808 , -9.6677931, -44.2270576, -103.8203209, 0.1984863, $
2021-10-23 20:33:37.717338 , -10.9297334, -42.8703358, -97.9111503, 0.2006836, $
2021-10-23 20:33:38.013720 , -12.0112620, -41.2342571, -92.2885758, 0.2022095, $
```

Figure 91. Data from decompressed file in encryption



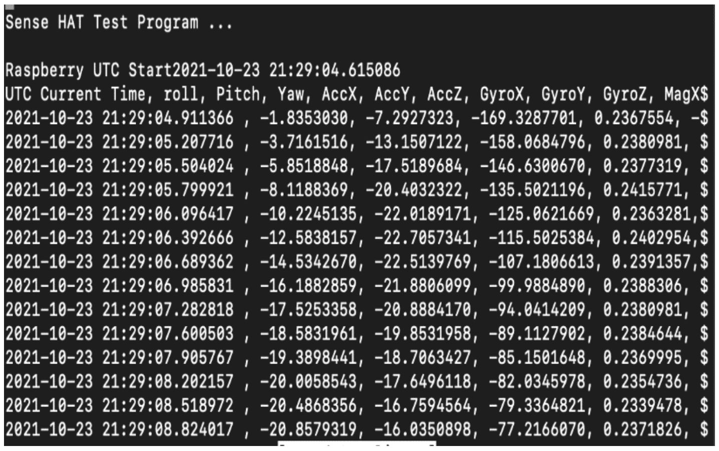
```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 20:33:33.868535
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX$
2021-10-23 20:33:34.164478 , -0.5289739, -8.5536921, -173.9762444, 0.2293091, -$
2021-10-23 20:33:34.460461 , -0.7479921, -17.0971135, -167.1481401, 0.2295532, $
2021-10-23 20:33:34.756796 , -0.9199910, -25.5144590, -160.1678329, 0.2318115, $
2021-10-23 20:33:35.053159 , -1.1909226, -32.7880994, -153.2623108, 0.2299805, $
2021-10-23 20:33:35.349039 , -1.6618831, -38.2969987, -146.3919485, 0.2263794, $
2021-10-23 20:33:35.645501 , -2.3129184, -42.3031638, -140.1972154, 0.2347412, $
2021-10-23 20:33:35.941617 , -3.2464307, -44.6259769, -134.0128773, 0.2141113, $
2021-10-23 20:33:36.237769 , -4.3447716, -45.8947348, -127.8440123, 0.2054443, $
2021-10-23 20:33:36.533868 , -5.5565534, -46.2754055, -121.7041357, 0.1992798, $
2021-10-23 20:33:36.829816 , -6.9858198, -46.0686526, -115.6395675, 0.1979370, $
2021-10-23 20:33:37.125724 , -8.3119426, -45.3285644, -109.6696787, 0.2025146, $
2021-10-23 20:33:37.421808 , -9.6677931, -44.2270576, -103.8203209, 0.1984863, $
2021-10-23 20:33:37.717338 , -10.9297334, -42.8703358, -97.9111503, 0.2006836, $
2021-10-23 20:33:38.013720 , -12.0112620, -41.2342571, -92.2885758, 0.2022095, $
```

Figure 91. Data from Data_3.csv file



```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 21:29:04.615086
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX$
2021-10-23 21:29:04.911366 , -1.8353030, -7.2927323, -169.3287701, 0.2367554, -$
2021-10-23 21:29:05.207716 , -3.7161516, -13.1507122, -158.0684796, 0.2380981, $
2021-10-23 21:29:05.504024 , -5.8518848, -17.5189684, -146.6300670, 0.2377319, $
2021-10-23 21:29:05.799921 , -8.1188369, -20.4032322, -135.5021196, 0.2415771, $
2021-10-23 21:29:06.096417 , -10.2245135, -22.0189171, -125.0621669, 0.2363281,$
2021-10-23 21:29:06.392666 , -12.5838157, -22.7057341, -115.5025384, 0.2402954,$
2021-10-23 21:29:06.689362 , -14.5342670, -22.5139769, -107.1806613, 0.2391357,$
2021-10-23 21:29:06.985831 , -16.1882859, -21.8806099, -99.9884890, 0.2388306, $
2021-10-23 21:29:07.282818 , -17.5253358, -20.8884170, -94.0414209, 0.2380981, $
2021-10-23 21:29:07.600503 , -18.5831961, -19.8531958, -89.1127902, 0.2384644, $
2021-10-23 21:29:07.905767 , -19.3898441, -18.7063427, -85.1501648, 0.2369995, $
2021-10-23 21:29:08.202157 , -20.0058543, -17.6496118, -82.0345978, 0.2354736, $
2021-10-23 21:29:08.518972 , -20.4868356, -16.7594564, -79.3364821, 0.2339478, $
2021-10-23 21:29:08.824017 , -20.8579319, -16.0350898, -77.2166070, 0.2371826, $
```

Figure 92. Data from decompressed file in encryption



```
Sense HAT Test Program ...

Raspberry UTC Start2021-10-23 21:29:04.615086
UTC Current Time, roll, Pitch, Yaw, AccX, AccY, AccZ, GyroX, GyroY, GyroZ, MagX$
2021-10-23 21:29:04.911366 , -1.8353030, -7.2927323, -169.3287701, 0.2367554, -$
2021-10-23 21:29:05.207716 , -3.7161516, -13.1507122, -158.0684796, 0.2380981, $
2021-10-23 21:29:05.504024 , -5.8518848, -17.5189684, -146.6300670, 0.2377319, $
2021-10-23 21:29:05.799921 , -8.1188369, -20.4032322, -135.5021196, 0.2415771, $
2021-10-23 21:29:06.096417 , -10.2245135, -22.0189171, -125.0621669, 0.2363281,$
2021-10-23 21:29:06.392666 , -12.5838157, -22.7057341, -115.5025384, 0.2402954,$
2021-10-23 21:29:06.689362 , -14.5342670, -22.5139769, -107.1806613, 0.2391357,$
2021-10-23 21:29:06.985831 , -16.1882859, -21.8806099, -99.9884890, 0.2388306, $
2021-10-23 21:29:07.282818 , -17.5253358, -20.8884170, -94.0414209, 0.2380981, $
2021-10-23 21:29:07.600503 , -18.5831961, -19.8531958, -89.1127902, 0.2384644, $
2021-10-23 21:29:07.905767 , -19.3898441, -18.7063427, -85.1501648, 0.2369995, $
2021-10-23 21:29:08.202157 , -20.0058543, -17.6496118, -82.0345978, 0.2354736, $
2021-10-23 21:29:08.518972 , -20.4868356, -16.7594564, -79.3364821, 0.2339478, $
2021-10-23 21:29:08.824017 , -20.8579319, -16.0350898, -77.2166070, 0.2371826, $
```

Figure 92. Data from Data_4.csv file

The screenshots in the left column all represent the data that was contained in the encrypted file. This encrypted file was then decrypted, decompressed, and opened to reveal the data inside using the code in block above. Each row represents a different data file that was used i.e. row 1 used Data_1.csv.tar.gz as input to the encryption block, row 2 used Data_2.csv.tar.gz as input to the encryption block, and so forth. The screenshots in the right column all represent the data in the compressed file from the compression block that was then decompressed without being encrypted. This is to validate that the decrypted and decompressed file in equivalent to the original decompressed file thus no data was lost.

Data integrity

Worked on by ADMALI004

The data integrity was successful in execution. Each data item was checked to ensure that it met both the requirements, namely, it is a float point value and that it is not an arbitrary value. The checks were executed successfully as the correct print statements were printed out at the correct time. The file does not get passed to the compression block algorithm without passing all the checks.

Change in performance when data is changed

Changing the time.sleep() in order to change the sample rate:
Varied from 0.1s to 0.05s to 0.01s

Sample time = 0.1s produced the following:

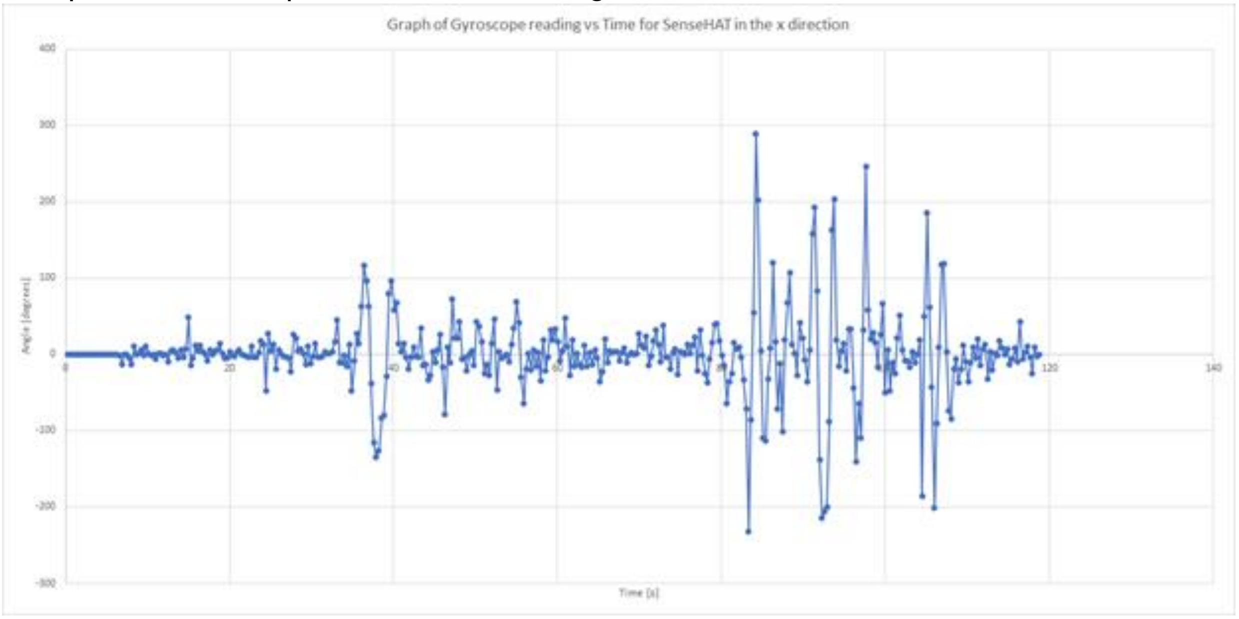


Figure 93. Gyroscope vs Time in x direction

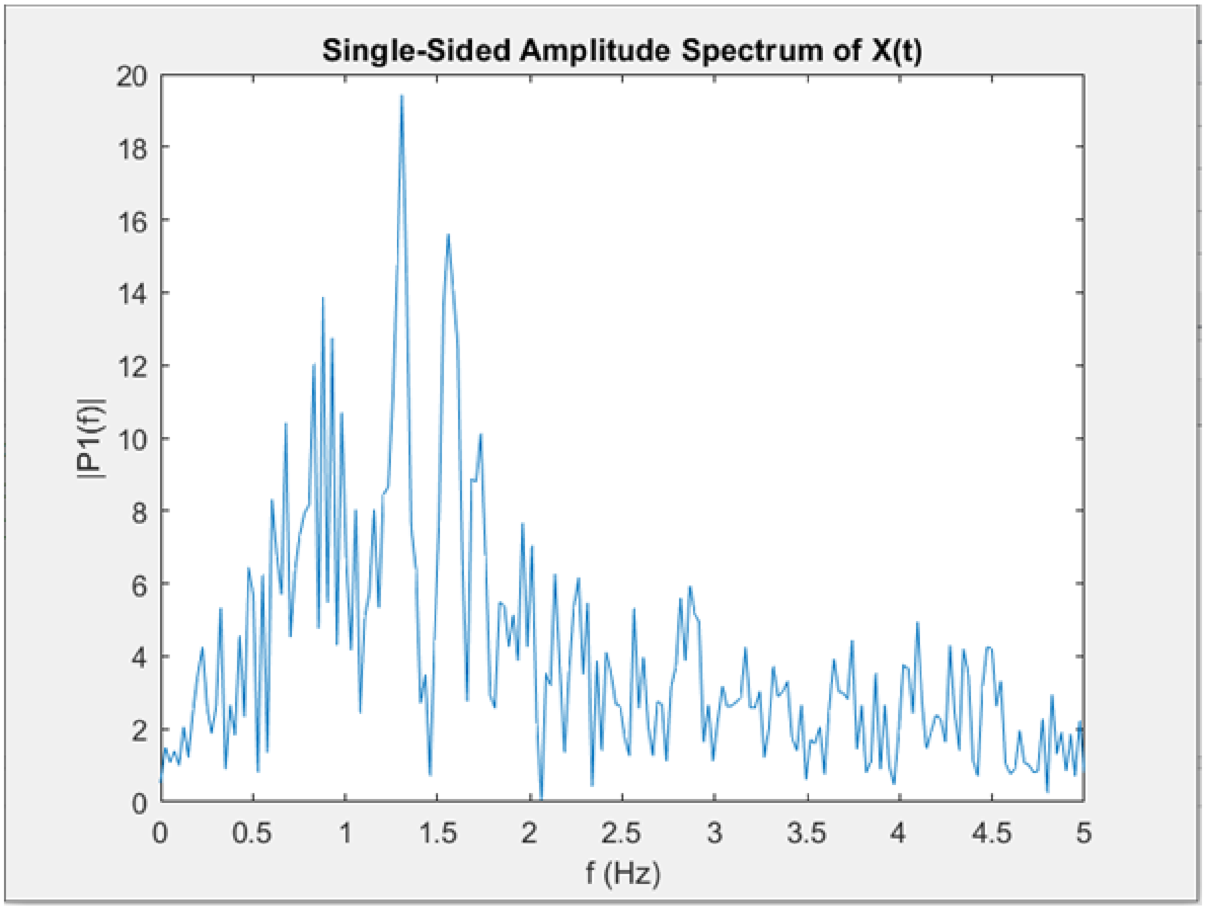


Figure 94. Fourier Transform of Gyroscope vs Time in x direction

Sample time = 0.05s produced the following:

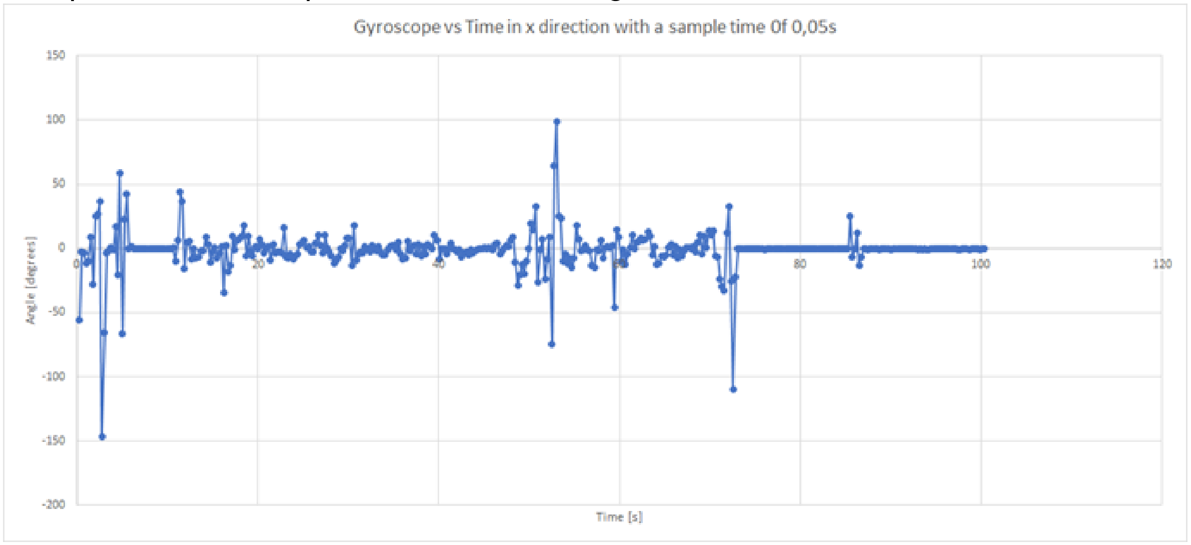


Figure 95. Gyroscope vs Time in x direction

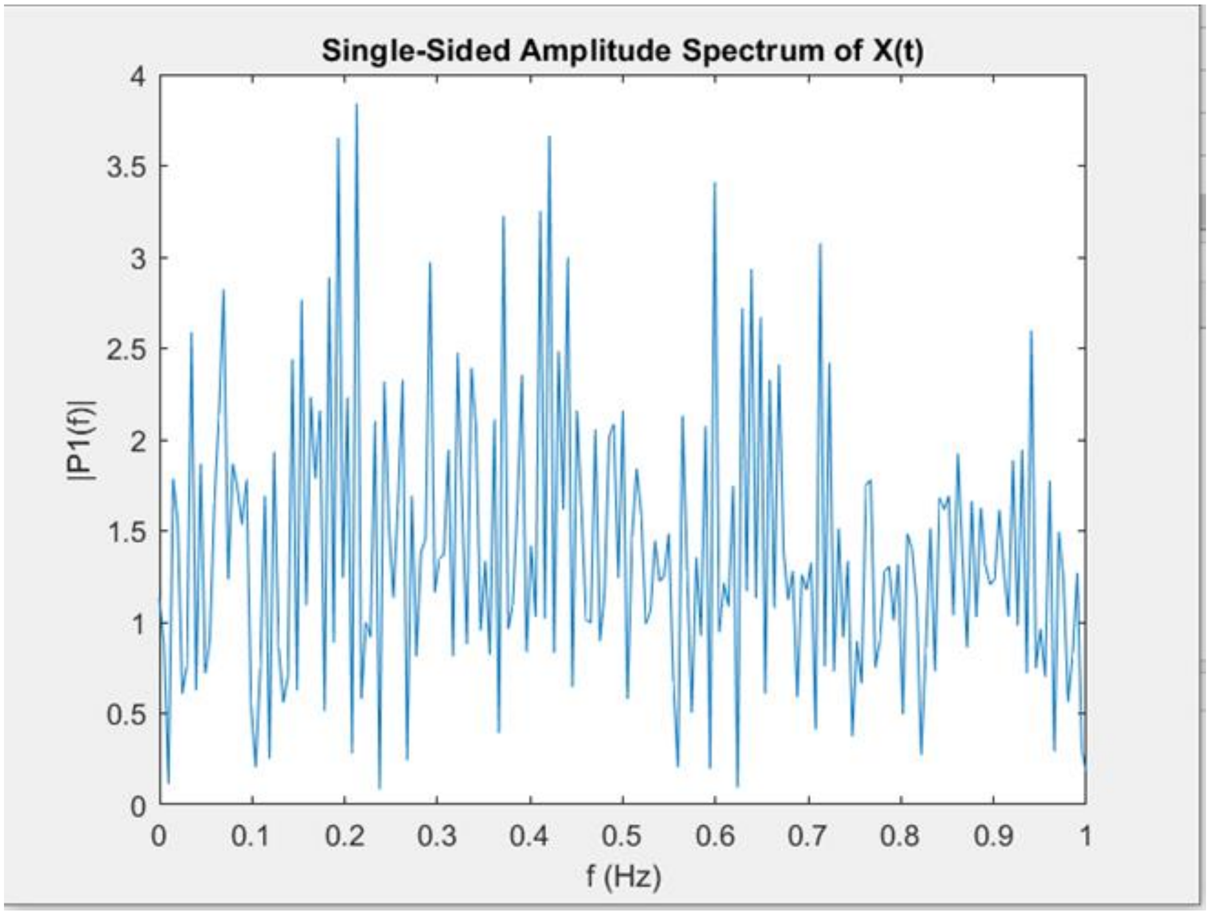


Figure 96. Fourier Transform of Gyroscope vs Time in x direction

Sample Time = 0.01s produced the following:

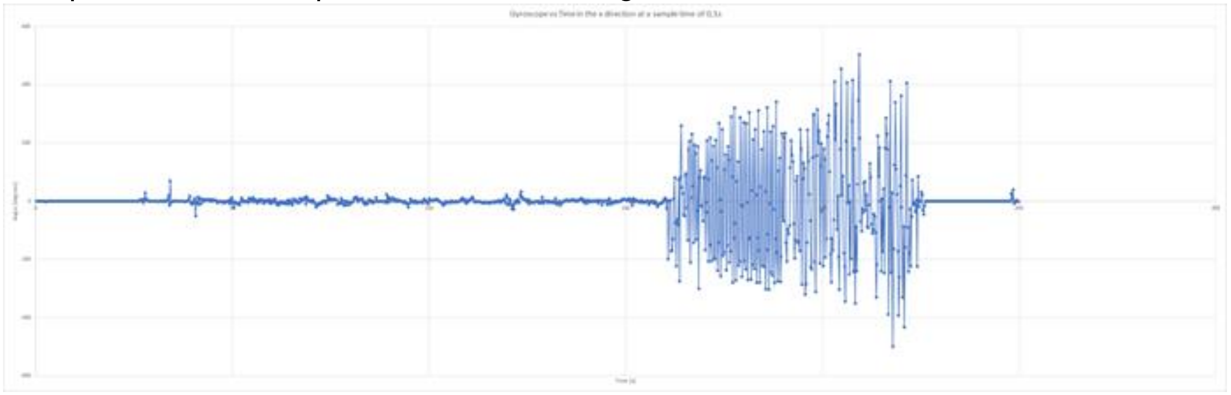


Figure 97. Gyroscope vs Time in x direction

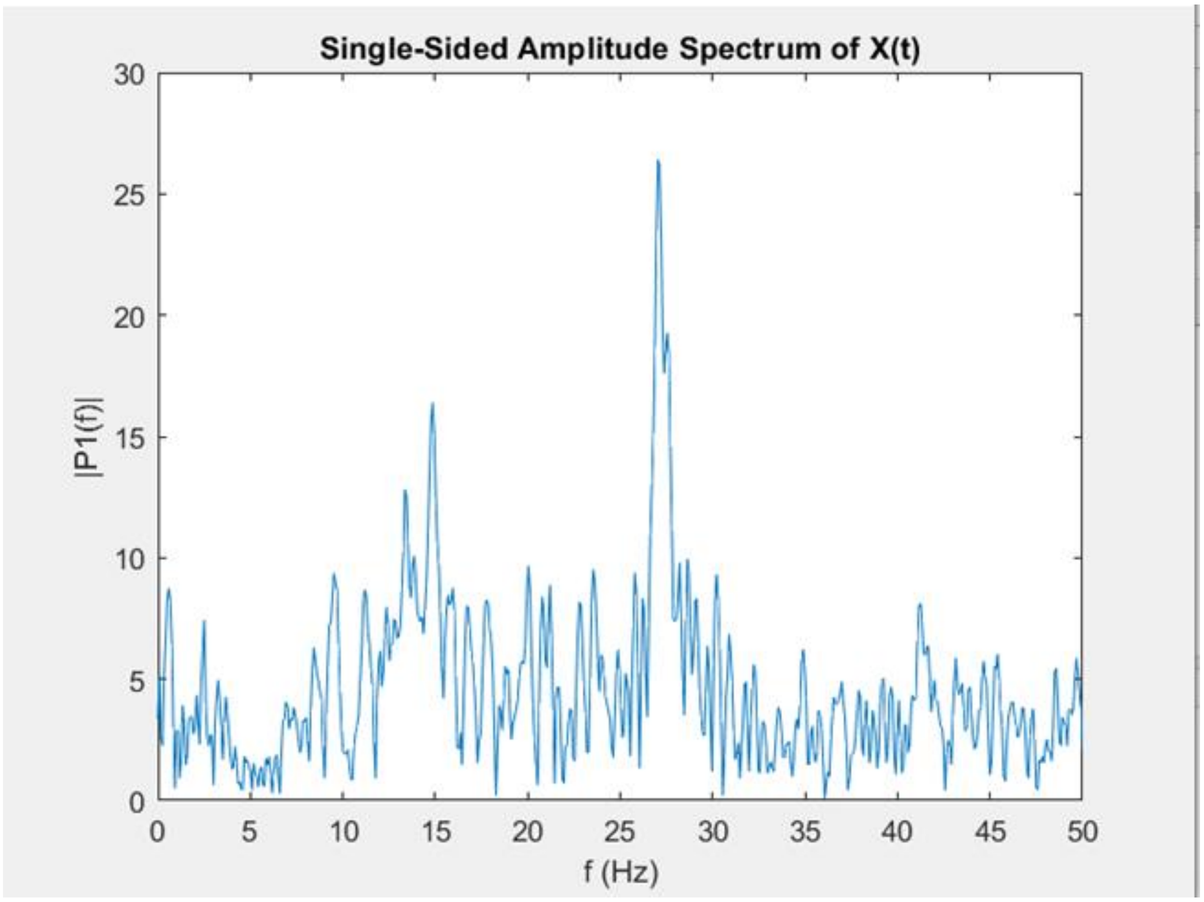


Figure 98. Fourier Transform of Gyroscope vs Time in x direction

As we reduce the sample time, we see more samples are taken, hence more noise is found and included in the readings. As the sample time decreased, the readings became more sensitive to movement and any disturbances. Due to using our hand to move the IMU Module, it will be difficult to produce a consistent periodic motion, so the Fourier Transform graphs above are expected, there are some dominant frequencies which represent the periodic motion of moving left and right consistently.

6. Consolidation of ATPs and Future Plan

6.1. ATPs

IMU Module:

The SenseHAT(B) will be tested in three different ways, namely the switch on sequence, increase in magnetic field and increase in shaking motion.

The SenseHAT(B) has an onboard LED that turns red whenever it is on thus this is an indication that the SenseHAT(B) is on and activated.

The magnetic field will be tested with nothing surrounding the SenseHAT, but gradually a magnetic will be brought closer to it over time and the change in data will be recorded. This will then be graphed to observe the differences.

The shaking motion will be tested in a similar manner to the magnetic field. To begin with it will lay dormant but overtime it will be moved gradually in a sinusoidal manner as if to mimic a wave motion. Thereafter, noise will be introduced by shaking the SenseHAT while moving it and the data will be recorded. These effects will then be graphed.

Compression:

Compress data extracted from smartphone, then decompress data, compare the two data to ensure no data is missing in decompress data. If the raw data and decompressed data are the same then the compression is successful and hence, the compression algorithm is successful. This was done using the rawData.csv file given to us.

The same procedure was followed to test the data that was extracted from the SenseHAT(B). The extracted data was placed in four different files and these files were all tested using the compression code. If there were no differences found between the decompressed file and the original file then the compression algorithm is successful.

Encryption:

The encrypted data will be sent to a device that is able to read it e.g., smartphone. The process of decrypting the data with the correct key will be tested as well decrypting it with the incorrect key. This is to ensure that only the correct key works and that the data does decrypt correctly. The file that this will be tested on will be obtained from the compression block. Different files will be tested as for the compression block as well.

Data-integrity:

To ensure that the data that has been extracted from the sensor is usable it will be checked against certain parameters. It will be checked to ensure that it is a float, that it is an adequate length i.e. it is not a space that is being read and that it is within the viable range of values for the readings. We decided to not include the check to make sure it is within 25% of the Fourier coefficients as the compression algorithm already covers this. The compression algorithm extracts the lowest 25% of Fourier coefficients.

Time	**ATP met (Y/N)**
IMU Module	Y
Compression	Y
Decompression	Y
Encryption	Y
Decryption	Y
Data-integrity	Y

Table 31. ATP Table

6.2. Future Work

We would like to implement an algorithm that runs in one go with no need for in-between pauses. The user would be able to indicate the number of samples they would like and the algorithm would be able to store that data immediately in a csv file and then the file would get compressed and then encrypted. We would like to implement a GUI where a graph can be displayed as data is read from the IMU, while it is being stored on a csv file, the algorithm can also visually show the data it is tracking.

Conclusion

Over the past few months, we have been actively working on a design that extracts data from a SHARC buoy's IMU module, compresses and encrypts the data before sending it off to be used by the relevant user. The design was separated into many different parts before coming together. The parts consisted of testing the extraction of the data from the IMU module and placing it into a file, checking the integrity of the data items, compressing and decompressing the file and encrypting and decrypting the file. Each part was tested and had to meet specific criteria before being used. The different parts of the design worked exactly as expected and thus when it was cascaded it produced the results that were desired. It can be concluded that the design has met all the requirements stipulated in the project handout.

References

[1] "Sense HAT (B)" [Online]. Available: https://www.waveshare.com/wiki/Sense_HAT_(B). [Accessed 02 10 2021].

[2] "FFT of Signal in MATLAB," [Online]. Available: FFT of Signal in MATLAB | Fast Fourier Transform in MATLAB | MATLAB Tutorial for Beginners . [Accessed 02 10 2021].

[3] "Python Tutorial: Working with CSV file," [Online]. Available: https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/. [Accessed 03 10 2021].

[4] "How to install pip in macOS: GeeksforGeeks," [Online]. Available: https://www.geeksforgeeks.org/how-to-install-pip-in-macos/. [Accessed 03 10 2021].

[5] "Python| os.system() method," [Online]. Available: https://www.geeksforgeeks.org/python-os-system-method/ . [Accessed 01 10 2021].

[6] "Encrypt and Decrypt Files using Python - Python Programming," [Online]. Available: https://levelup.gitconnected.com/encrypt-and-decrypt-files-using-python-python-programming-pyshark-a67774bbf9f4. [Accessed 03 10 2021].

[7] "Add title and axis labels to chart," [Online]. Available: https://ch.mathworks.com/help/matlab/creating_plots/add-title-axis-labels-and-legend-to-graph.html. [Accessed 04 10 2021].