EEE3097S PROGRESS REPORT 1

Admin Documents

Github link: [lukhanyoVena808/IMU_DESIGN (github.com)]

Table 1: Contribution table

CONTRIBUTORS	SUBSYSTEM
Lukhanyo Vena (VNXLUK001)	Compression
Bina Mukuyamba (MKYBIN001)	Encryption

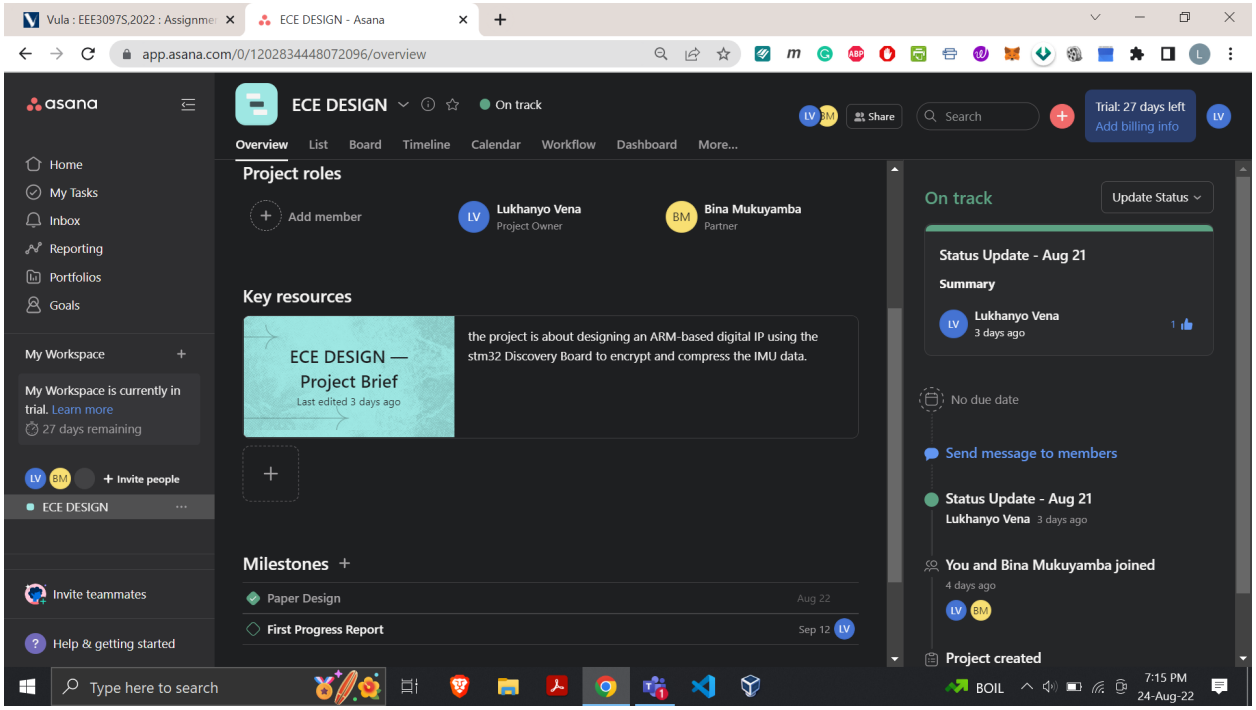

Figure 1: Overview of project management page

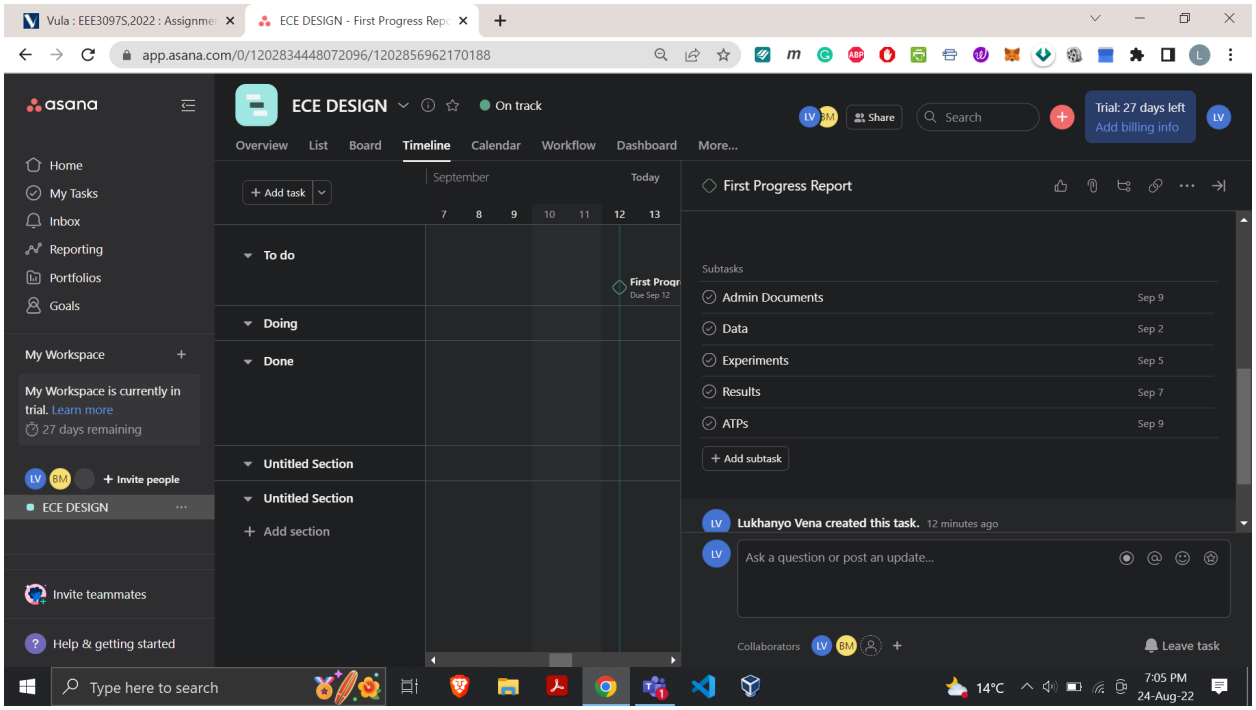

Figure 2: Timeline of the project management system

Data

Table 2: Sample of Test Data Used

Quat Z	Accel X (g)	Accel Y (g)	Accel Z (g)	Gyro X (dps)	Gyro Y (dps)	Gyro Z (dps)	Yaw	Pitch	Roll	X	Y	Z	Grav X	Grav Y	Grav Z	Temp
0	-0.001799999969	-0.002400001114	4.488900185	-0.061000006969	-0.061000006969	0	0	0.000600000285	0.000730000172	0	0.000600000285	-0.000730000172	0.000600000285	0.000730000172	0.9998999834	30.37709999
0	-0.004199999985	-0.008399999705	4.700799942	0	-0.061000006969	-0.061000006969	0	0.000699999975	0.000710000046	0	0.000699999975	-0.000710000046	0.000699999975	0.000710000046	0.9998999834	30.39469091
0	-0.003599999938	-0.010800002828	4.804399967	-0.061000006969	-0.061000006969	0	0	0.000100000047	0.000700002216	0	0.000100000047	-0.000700002216	0.000100000047	0.000700002216	0.9998999834	30.38879967
0	-0.003599999938	-0.010800002828	4.804399967	-0.061000006969	-0.061000006969	0	0	0.000100000047	0.000700002216	0	0.000100000047	-0.000700002216	0.000100000047	0.000700002216	0.9998999834	30.38879967

The data in Table 2, is the data used to test the system and is an adaptation of “EEE3097S 2022 Turntable Example Data 2” excel sheet. The following plots are of some of the sample data sets adapted from the excel file.
A Julia notebook was used to plot the data in the time and frequency domain. The data was plotted by summing up the values in each row of the data set and plotting these sums with respect to the rows they belong to.

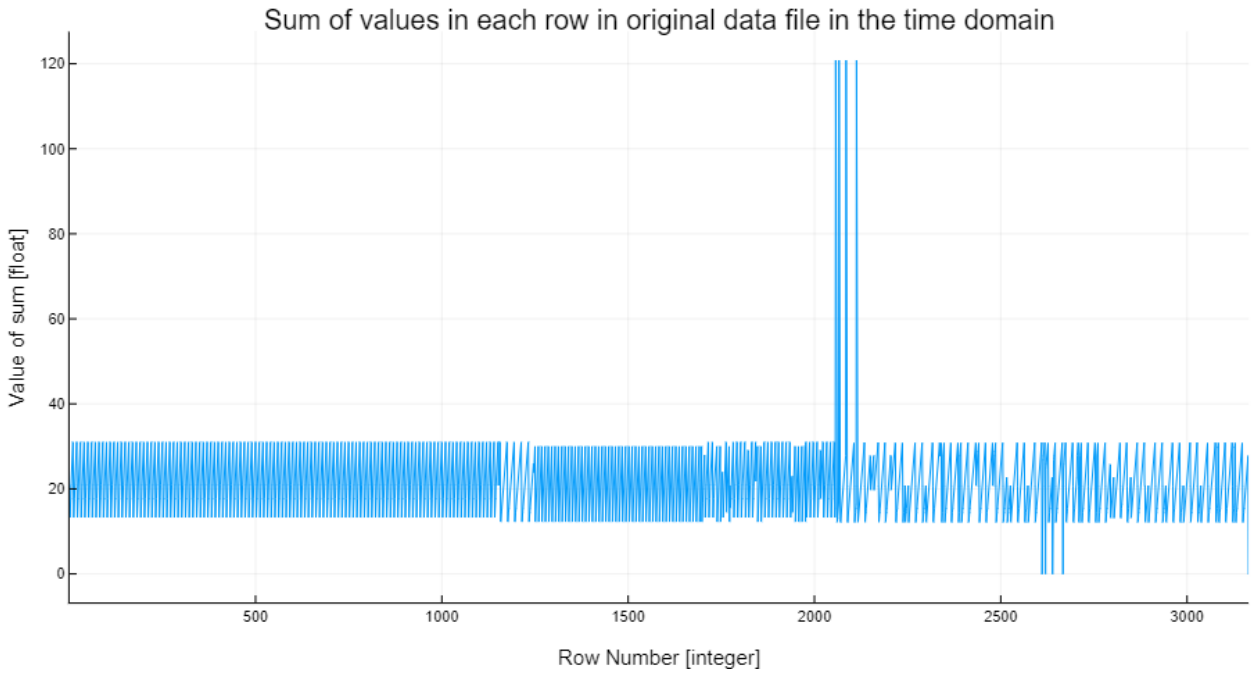

Figure 3: Original data of 777855 bytes in time-domain

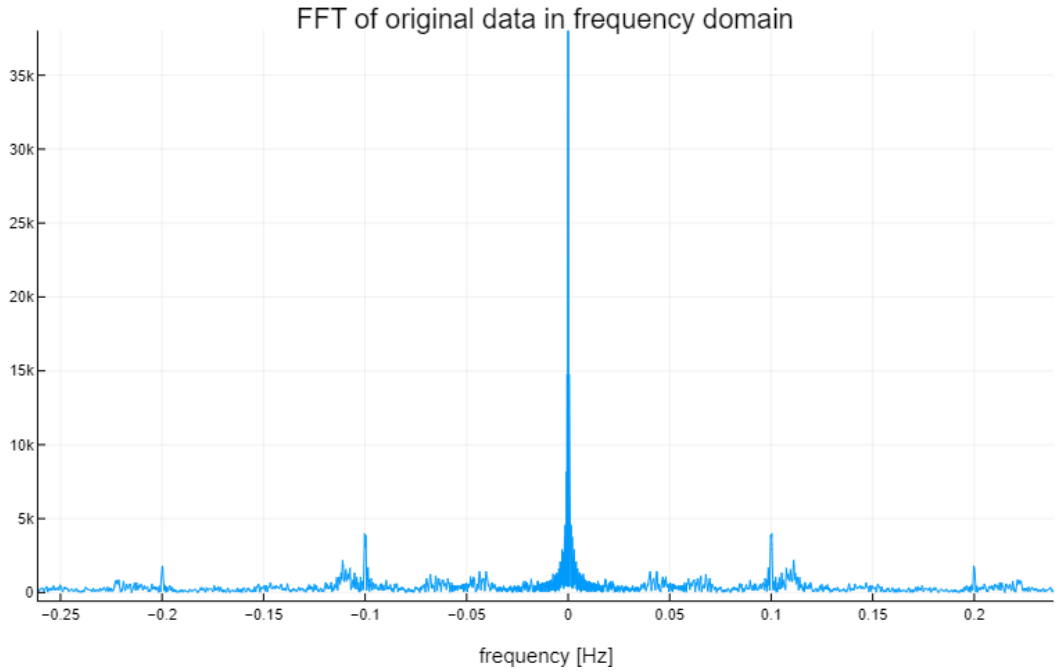

Figure 4: Original data of 777855 bytes in frequency-domain

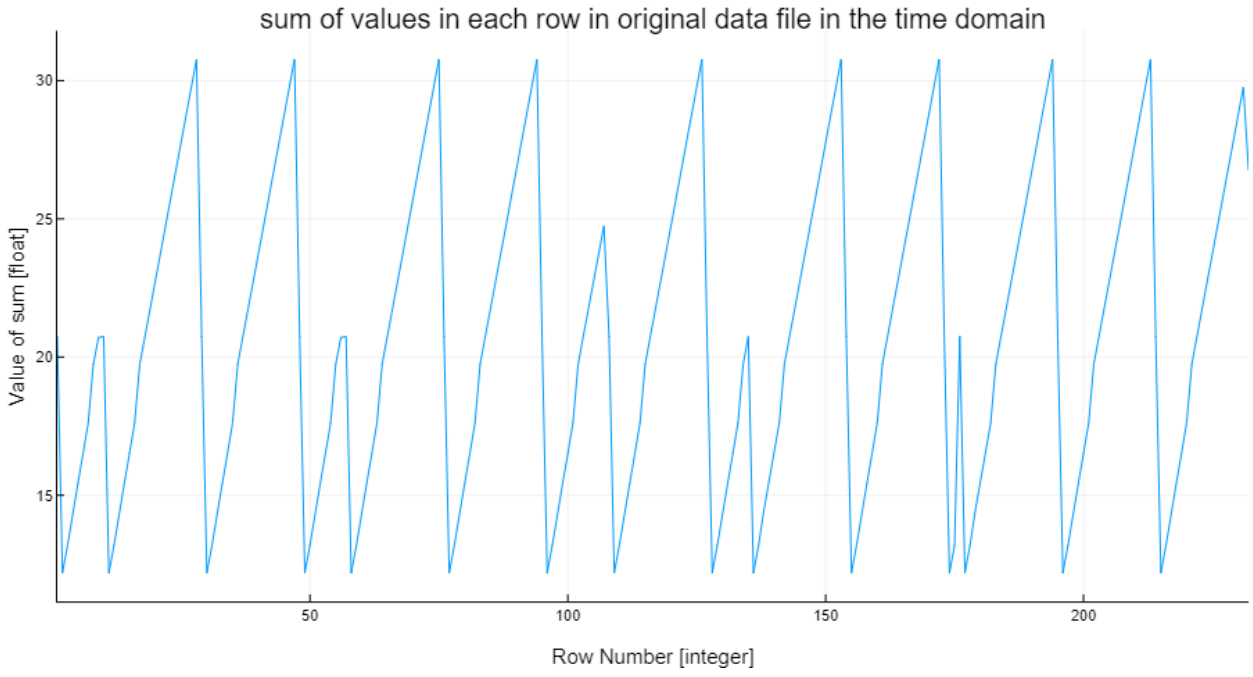

Figure 5: Original data of 11642 bytes in time-domain

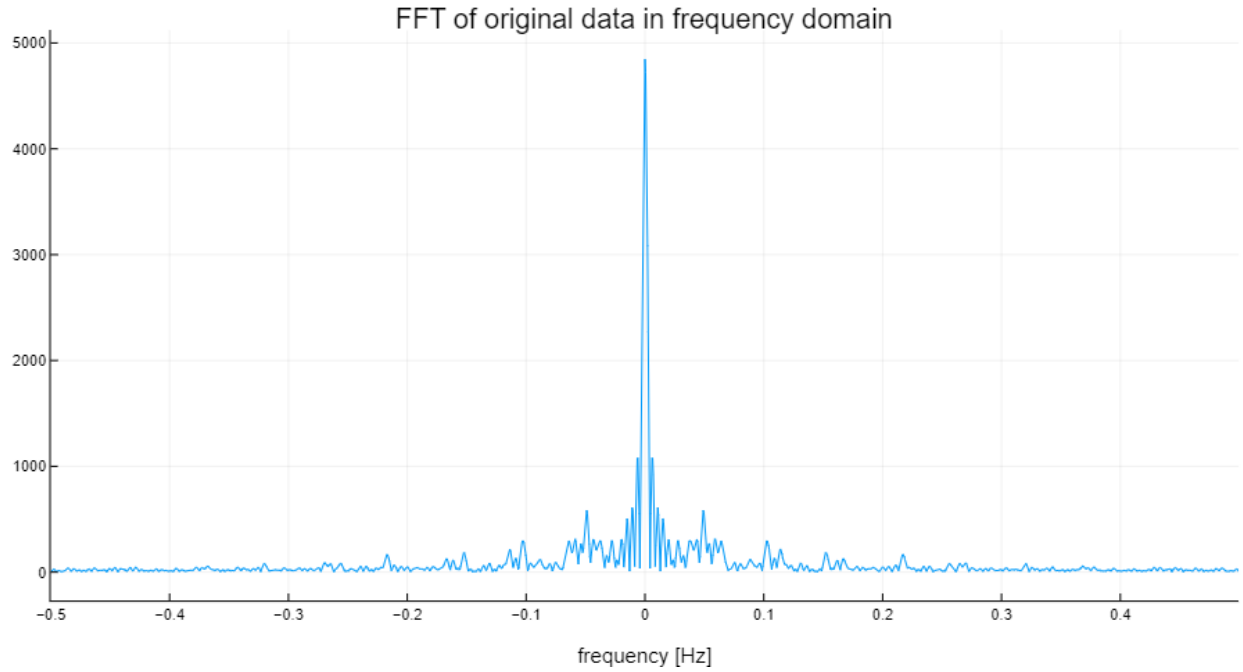

Figure 6: Original data of 11642 bytes in frequency-domain

We chose this data for the experiments because it has a similar type, size and format to the data we expect to obtain when using the actual IMU since this data was also obtained from an IMU. With this data it will be possible to quantitatively measure the ATP metrics of the system and determine if ATPs have been met. From the ATP's discussed in the Paper Design, a compression ratio of less than 90% is needed. This data set is large enough to determine if the algorithms can compress the data so that this ATP is met.The data is also in a format that can be encrypted(string of numbers) so encryption algorithms can be tested using it.

This data is random which means different frequencies can be visualized in the frequency domain which would make it easier to test whether the compressed data has at least 25% of the Fourier coefficients of the original data.

The limitation of using this data is that it was not obtained in Antarctica so there will be variation in the data values such as temperature between the test data and real data from Antarctica. This might affect how our algorithms perform since the range of values will be different.

Experimental Setup

Overall system

The overall system is comprised of these steps in the following order: compression -> encryption -> decryption -> decompression. To simulate this behavior, a string array was initialized with a portion of the sample data in the STM main file. A python script was used to read the data using serial communication from the STM to the laptop. The data was then compressed using Huffman[1] [4] then LZ77 [2] algorithm, encrypted and later decrypted using Salsa20 algorithm, and decompressed to retrieve original data. The final output after the compression and encryption process was saved in the Huffman output folder (as final.txt) when executing the Huffman algorithm and the LZ77 output folder after executing the LZ77 algorithm.

This procedure was repeated for input data files of different sizes inside the data folder (Huffman and LZ77 folder). The python scripts also computed the execution time for the overall process of compressing and encrypting using each compression algorithm. The scripts also computed the compression ratio (size ratio of the compressed file to original data). To check if the output data retains at least 25% of the Fourier coefficients, the ratio of the decompressed data Fourier transform to the original data Fourier transform is plotted.

Compression

In order to test if the compression algorithm works, the execution time for compressing and decompressing files of different sizes were recorded for each algorithm using the python scripts. The compression ratios (compressed file /original file * 100) were also calculated. The decompressed data was also plotted in the time and frequency domain to compare it to the original data. This experiment/simulation allows us to determine the most ideal compression algorithm to use in the system, which will provide fast execution, low memory storage and accurate compression of files in the system when we use the actual sensor data.

Encryption

The encryption algorithm used was changed from AES to Salsa20 in the simulation, because the AES algorithm we used was corrupting the compressed data (info was lost) due to incompatibility with the compression algorithms chosen.

The program used for encryption/decryption was “fenc.c” which implemented the Salsa20 stream cypher algorithm [3]. The encryption subsystem had a compressed file containing the test IMU data as input(output from the compression block). Then the expected output of the subsystem was the compressed file with the contents encrypted. Similarly the expected output of the decryption was the compressed file holding plaintext(un-encrypted) data.

To test the algorithm data files of different sizes were passed into the program and the execution time was measured using the python scripts. The different times obtained for each file size were compared.
To check if encryption occurred a dummy text file was passed into the program “fenc.c” and the output written to another file. These two files were visually inspected to see if data was encrypted.

Below is a UML use case digram summarizing the experimental set up for this simulation

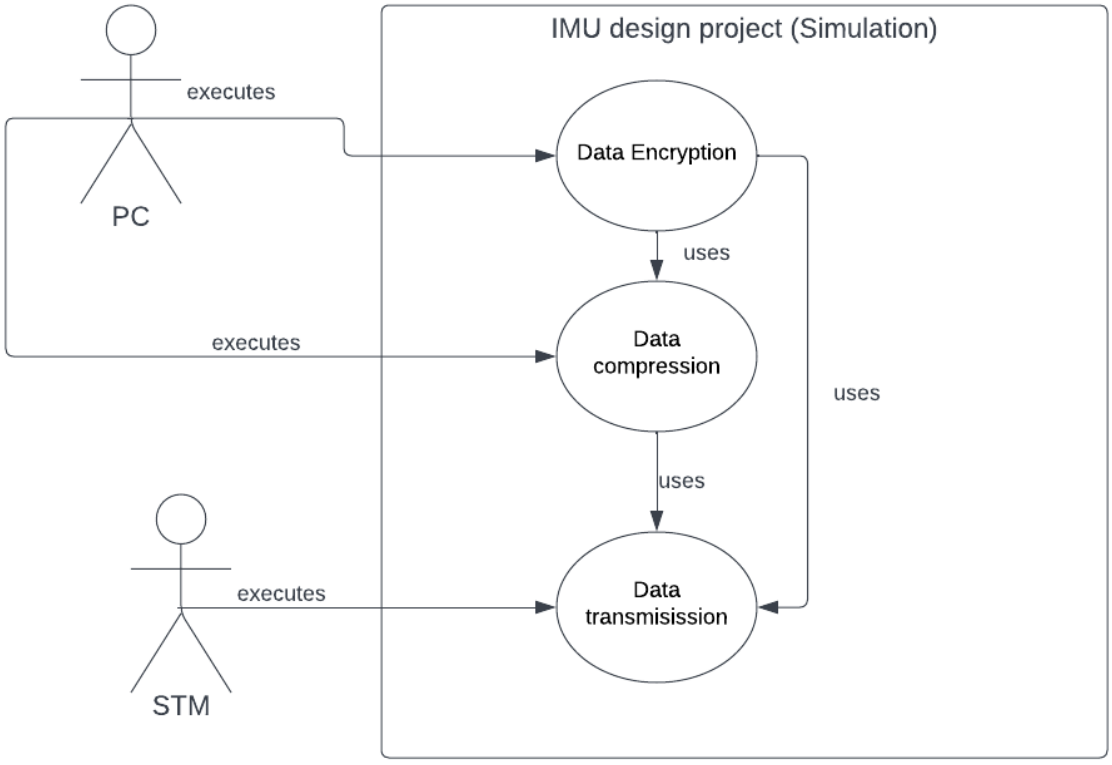

Figure 7: UML use case diagram

Results

Using the above methods the tables below were drawn to show the relationship between file size and execution time for each sub-system and the overall system for different input file sizes. It was observed that the Huffman compression and Salsa encryption algorithm combination was faster than the Lz compression and Salsa encryption combination.

Overall System

Table 3: Time executions of overall system at different data sets

	Time execution of the whole system [seconds]	
Size of Input file [Bytes]	Using Huffman Algorithm	Using LZ77 Algorithm
777855	2.680	4.205
11642	2.376	2.371
58561	2.466	2.473
93723	2.376	2.586
152513	2.498	2.610
211195	2.518	2.874
269877	2.567	2.937
410232	2.530	3.243
492506	2.486	3.476
971733	2.714	4.575

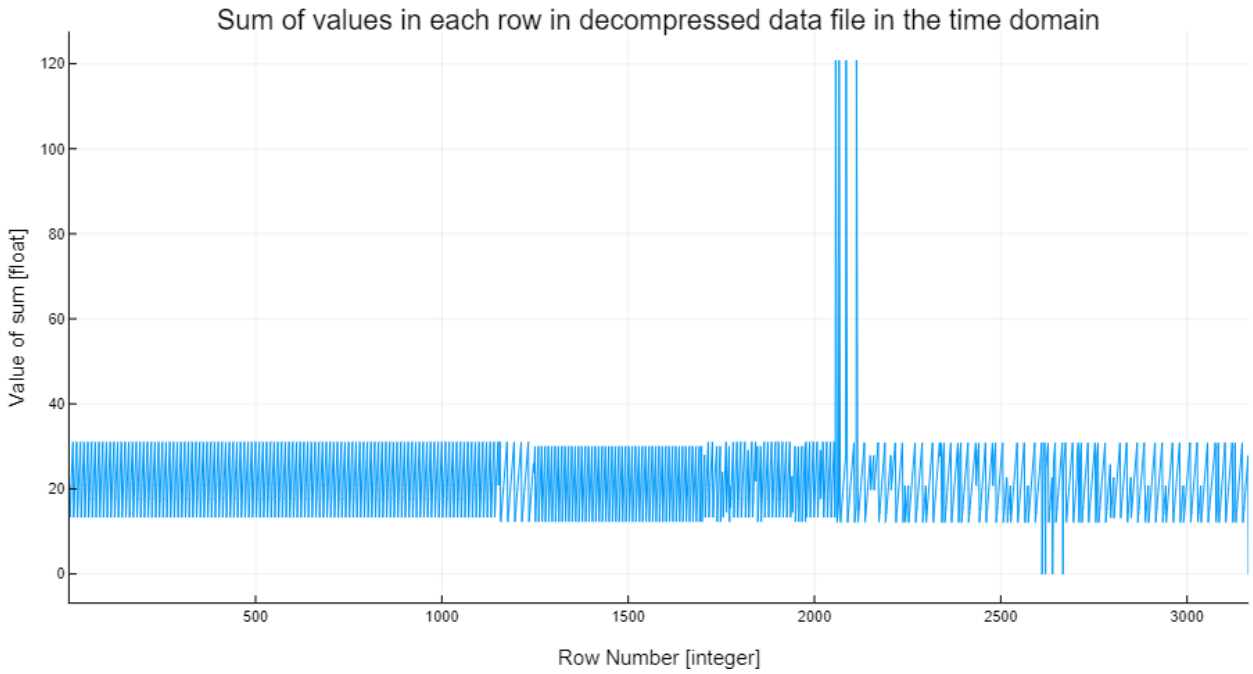

Figure 8: Decompressed data of 777855 bytes in time-domain

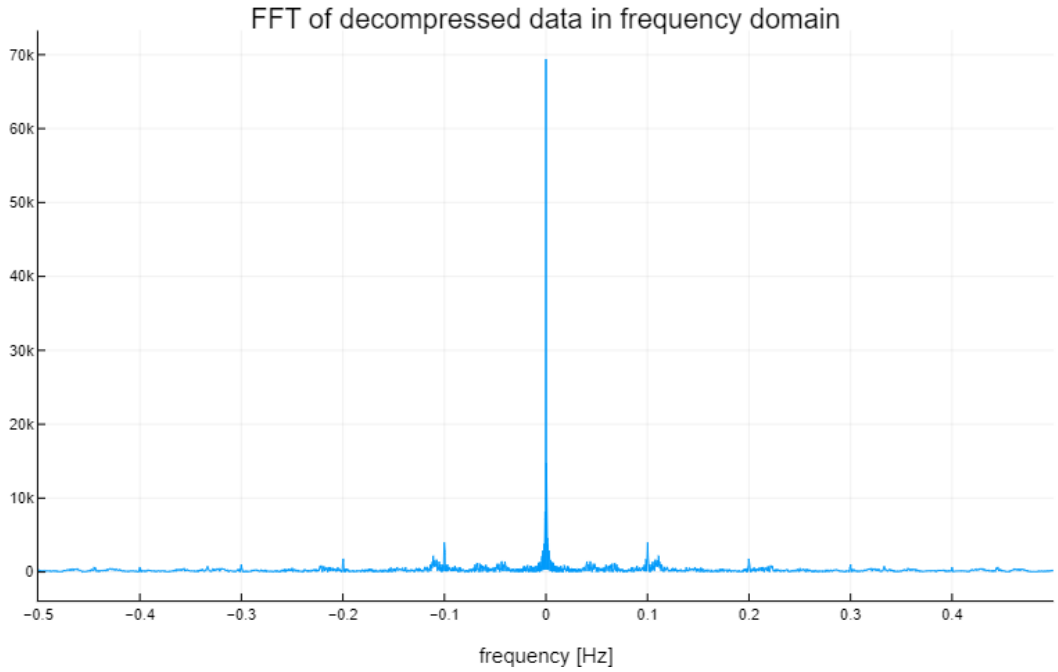

Figure 9: Decompressed data of 777855 bytes in frequency-domain

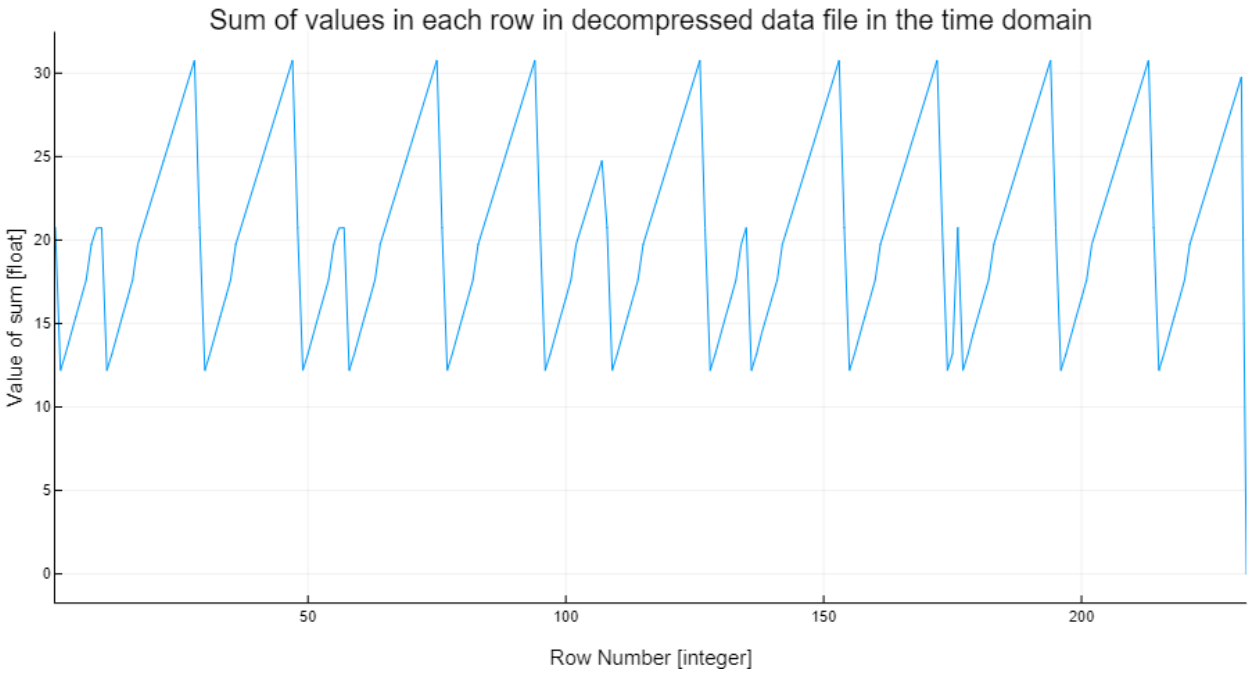

Figure 10: Decompressed data of 11642 bytes in time-domain

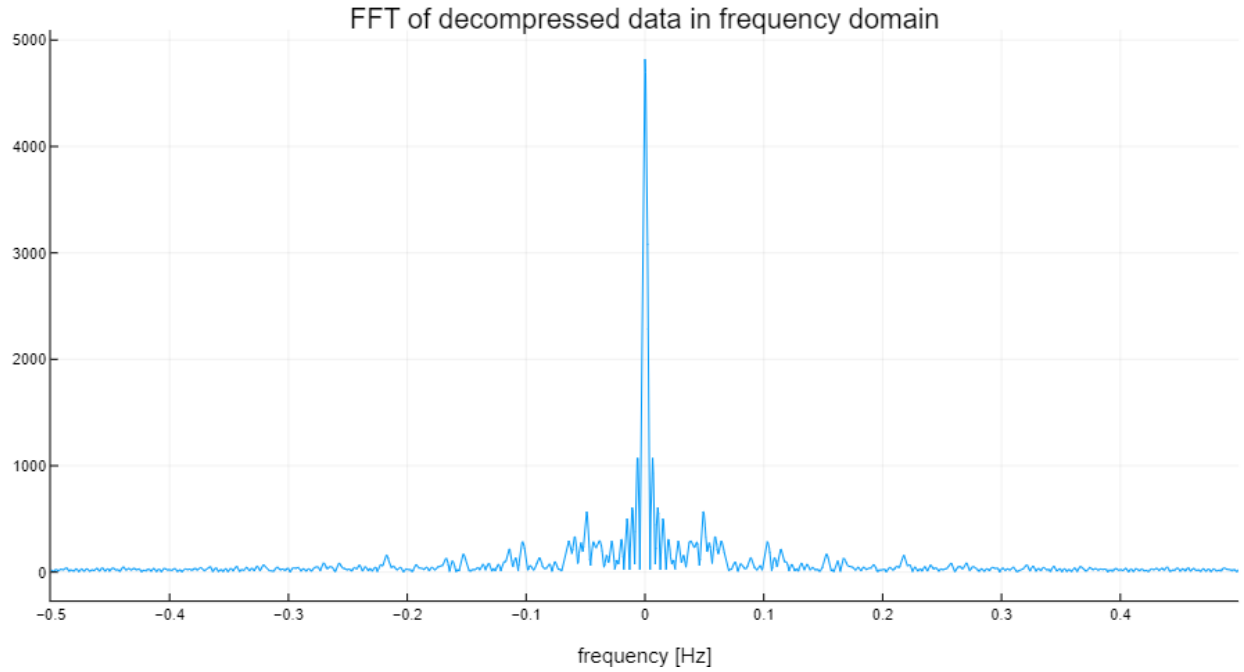

Figure 11: Decompressed data of 11642 bytes in frequency-domain

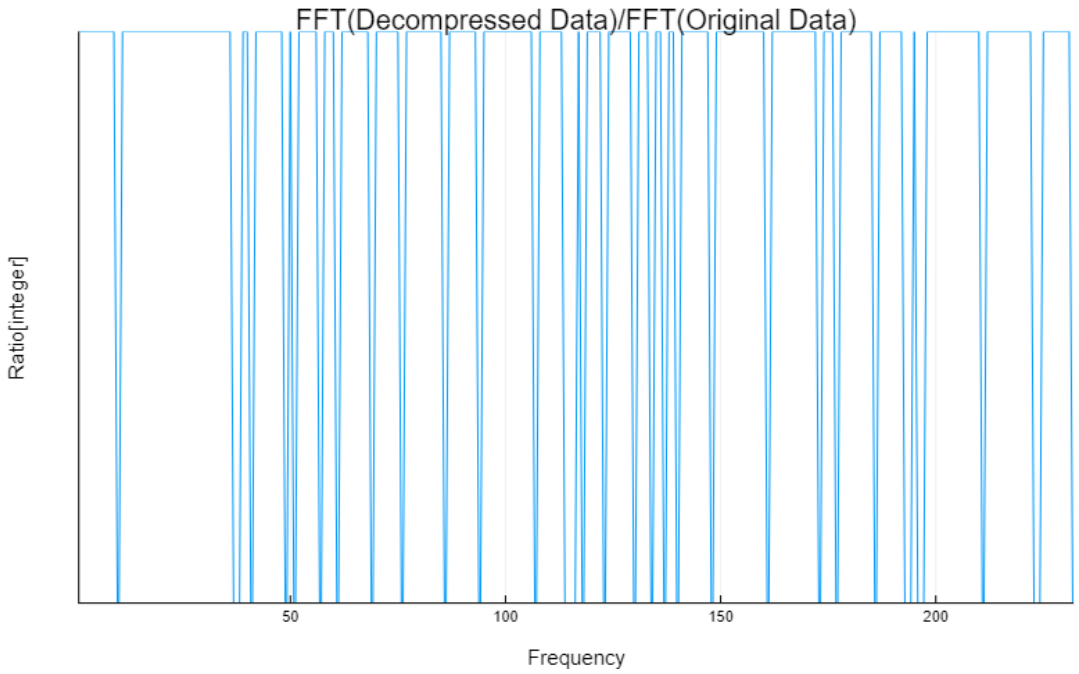

Figure 12: Ratio of Decompressed data FFT to Original Data FFT of 11642 bytes in frequency-domain

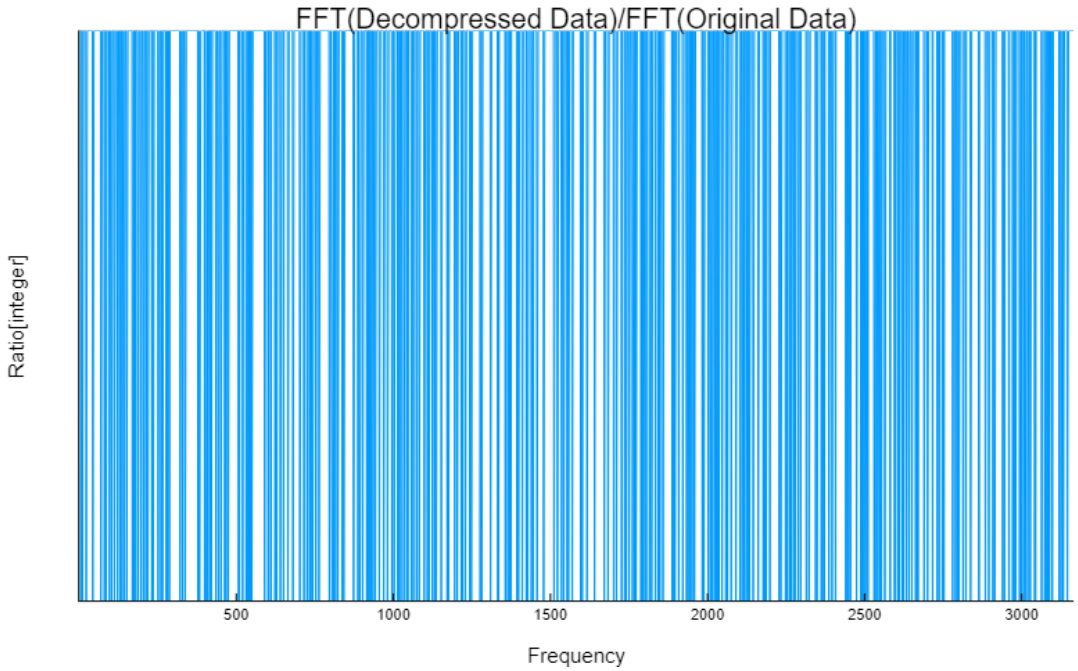

Figure 13: Ratio of Decompressed data FFT to Original Data FFT of 777855 bytes in frequency-domain

Compression sub-system

Huffman Compression and decompression

Table 4: Results of Huffman Algorithm

Size of Input file [Bytes]	Time execution of compression Algorithm [seconds]	Time execution of decompression Algorithm [seconds]	Compression Ratio [%]
777855	0.171	0.125	44.478
11642	0.015	0.000	49.811
58561	0.016	0.015	46.179
93723	0.031	0.015	45.904
152513	0.063	0.047	45.439

211195	0.062	0.062	45.392
269877	0.077	0.046	45.367
410232	0.110	0.078	45.182
492506	0.109	0.078	45.166
971733	0.219	0.172	45.095

LZ77 Compression and decompression

Table 5: Results of LZ77 Algorithm

Size of Input file [Bytes]	Time execution of compression Algorithm [seconds]	Time execution of decompression Algorithm [seconds]	Compression Ratio [%]
777855	1.641	0.125	90.779
11642	0.015	0.000	91.582
58561	0.156	0.000	91.098
93723	0.203	0.015	91.088
152513	0.328	0.031	90.903
211195	0.484	0.046	90.921
269877	0.577	0.078	90.931
410232	0.859	0.094	90.862
492506	1.063	0.078	90.877
971733	2.062	0.187	90.832

Encryption

Encrypting Huffman compressed files

Table 6: Results of Salsa20 Algorithm for Huffman files

Size of Input file [Bytes]	Time execution of encryption Algorithm	Time execution of decryption Algorithm

	[seconds]	[seconds]
777855	0.000	0.000
11642	0.000	0.015
58561	0.000	0.000
93723	0.000	0.000
152513	0.000	0.000
211195	0.000	0.000
269877	0.000	0.015
410232	0.000	0.015
492506	0.000	0.015
971733	0.000	0.015

Encrypting LZ77 compressed files

Table 7: Results of Salsa20 Algorithm for LZ77 files

Size of Input file [Bytes]	Time execution of encryption Algorithm [seconds]	Time execution of decryption Algorithm [seconds]
777855	0.000	0.000
11642	0.000	0.000
58561	0.000	0.000
93723	0.000	0.015
152513	0.015	0.015
211195	0.016	0.000
269877	0.016	0.000
410232	0.000	0.015
492506	0.000	0.000
971733	0.000	0.000

Note: times such as 0.000s were very small and couldn't be displayed by script i.e. 0.0000….

DISCUSSIONS

In the overall system, the Huffman algorithm is faster. In table 3 it can be observed that as the file sizes increases, the execution time of the LZ77 algorithm is higer than Huffman. This makes it less efficient as longer execution time means a lot more computational power is used by the system.

Figures 8 and 9 show the output of the 777855 bytes file at the end of the system. By observation, the figures look identical to the original data figures 3 and 4. Figure 13, confirms that the output data has over 25% of the original data FFT. Similarly, figures 10 and 11 look identical to the original data figures 5 and 6,since the peaks in the frequency domain occur at the same frequencies. Figure 12 shows that the data of the 11642 bytes file is retained in the decompressed file, as the ratio of the decompressed file to original data is 1.

In table 4, the compression ratio of the Huffman algorithm is way below the 90% set in the ATP's. This makes it very efficient, as it can compress a file to half its size which saves space, and computational power for the IMU system. The LZ77 algorithm results shown in table 5, however, does not efficiently compress a file to save a lot of space, a lot of memory would still be used up even after compression.

ATPs

Table 8: ATP summary

ATP	ATP Met (Y/N)	Why ATP hasn't been met	Changes
When the data is compressed, the compression ratio must be at least under 90%.	Y, We get 50% for Huffman But get >90 for LZ77	The LZ77 does not compress file smaller than 90% of the size of the original data, as the files increase in size	Choose Huffman in final design because its compression ratio is small enough to save space in the IMU Design.
The compression data should have at least 25% of the Fourier coefficients of the data.	Y	-	-

Compression does not take more than 90 seconds to run.	Y, takes less than 10s	-	-
Test if the data is sent from the microcontroller to the PC and vice-versa	Y, can see communication via putty	-	-
Test if communication between IMU and microcontroller is working	N	We don't have IMU yet	Acquire an IMU
Encryption does not take more than 90s to run	Y, sub-system takes less than 10s	-	-
Encryption uses block cypher	N	AES algorithm in C didn't work	Used Salsa20 stream cypher algorithm
Ciphertext different from plain text	Y	-	-
Original data obtained after decryption	Y	-	-
all the original data is recovered after encryption, decryption compression and decompression	Y	-	-

References

[1] Bestoa, "Bestoa/Huffman-codec: Very simple 8 bits Huffman encoder/decoder with pure C.," *GitHub*. [Online]. Available: https://github.com/Bestoa/huffman-codec [Accessed: 12-Sep-2022].

[2] Favrito, “Favrito/LZ77 at 54fe795a983ac84ab615e16c2b0a6194c84825be,” *GitHub*. [Online]. Available: https://github.com/Favrito/LZ77/tree/54fe795a983ac84ab615e16c2b0a6194c84825be [Accessed: 12-Sep-2022].

[3] Adamierymenko, “Adamierymenko/FENC: Fenc: A utility to just encrypt a file,” *GitHub*. [Online]. Available: https://github.com/adamierymenko/fenc [Accessed: 12-Sep-2022]

[4] “Huffman coding: Greedy Algo-3,” *GeeksforGeeks*, 06-Sep-2022. [Online]. Available: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/ [Accessed: 12-Sep-2022].

[5] D. Budhrani, “How data compression works: Exploring LZ77,” *Medium*, 28-Dec-2019. [Online]. Available: https://towardsdatascience.com/how-data-compression-works-exploring-lz77-3a2c2e06c097 [Accessed: 12-Sep-2022].