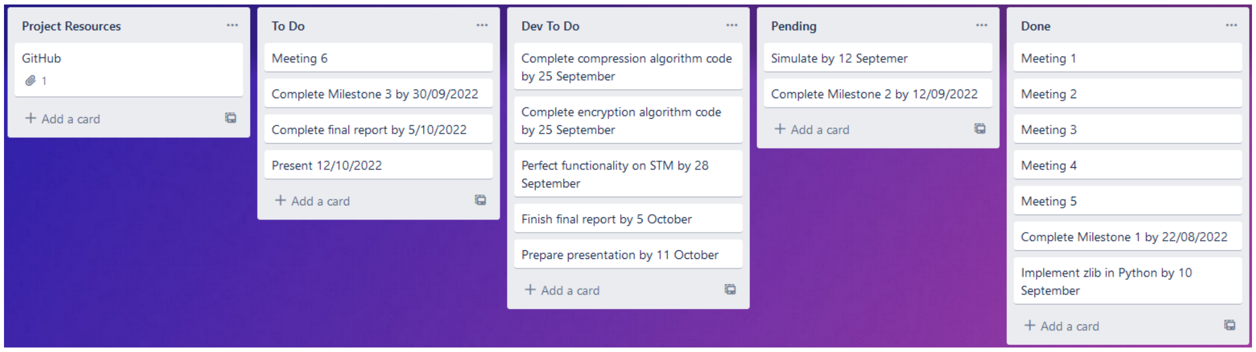
Milestone 2:
First Progress Report (Simulation)

1. Admin Documents

1.1 Contribution

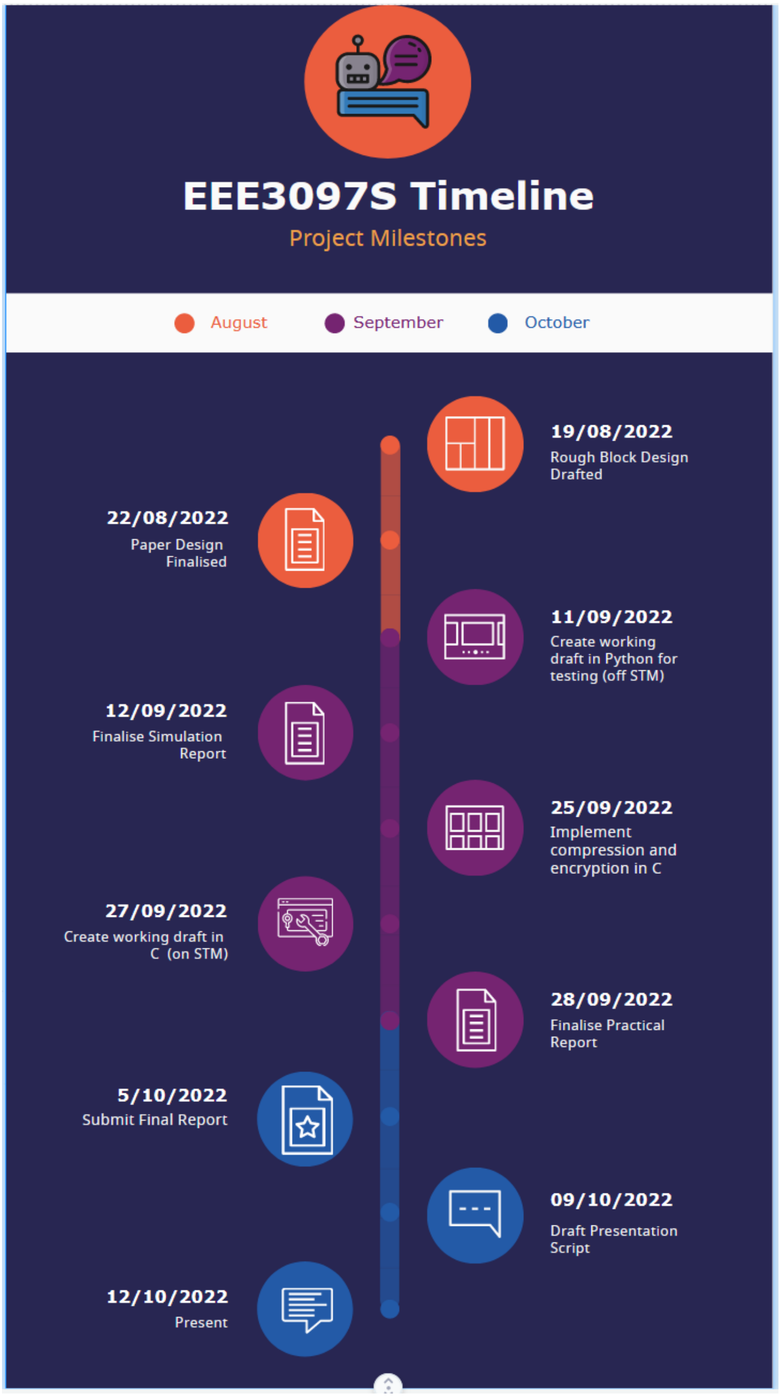
STRREI003 (Encryption)	1 3.1 & 3.3 4.1 & 4.3 5
WLSCAM007 (Compression)	1 2 3.2 4.2 5 6

1.2 Trello, GitHub, and Timeline



Project Resources	To Do	Dev To Do	Pending	Done
GitHub	Meeting 6	Complete compression algorithm code by 25 September	Simulate by 12 Septemer	Meeting 1
	Complete Milestone 3 by 30/09/2022	Complete encryption algorithm code by 25 September	Complete Milestone 2 by 12/09/2022	Meeting 2
	Complete final report by 5/10/2022	Perfect functionality on STM by 28 September		Meeting 3
	Present 12/10/2022	Finish final report by 5 October		Meeting 4
		Prepare presentation by 11 October		Meeting 5
				Complete Milestone 1 by 22/08/2022
				Implement zlib in Python by 10 September

GitHub



EEE3097S Timeline

Project Milestones

● August ● September ● October

19/08/2022
Rough Block Design Drafted

22/08/2022
Paper Design Finalised

11/09/2022
Create working draft in Python for testing (off STM)

12/09/2022
Finalise Simulation Report

25/09/2022
Implement compression and encryption in C

27/09/2022
Create working draft in C (on STM)

28/09/2022
Finalise Practical Report

5/10/2022
Submit Final Report

09/10/2022
Draft Presentation Script

12/10/2022
Present

2. Data

2.1 Data-type Discussion

For the simulated tests, the data-sets used must be applicable to all acceptance test procedures. The following requirements have been outlined. The remaining ATPs had no reference to any aspect of the data-sets.

Req. #	ATP ID	ATP Summary	Data-set requirement
1	U1.F1.S1.A1	Data sets consisting of ASCII characters must be 100% recoverable.	The data-sets must contain some ASCII characters.
2	U1.F1.S1.A2	Data sets of any format must be 100% recoverable.	The data-sets must vary in structure.

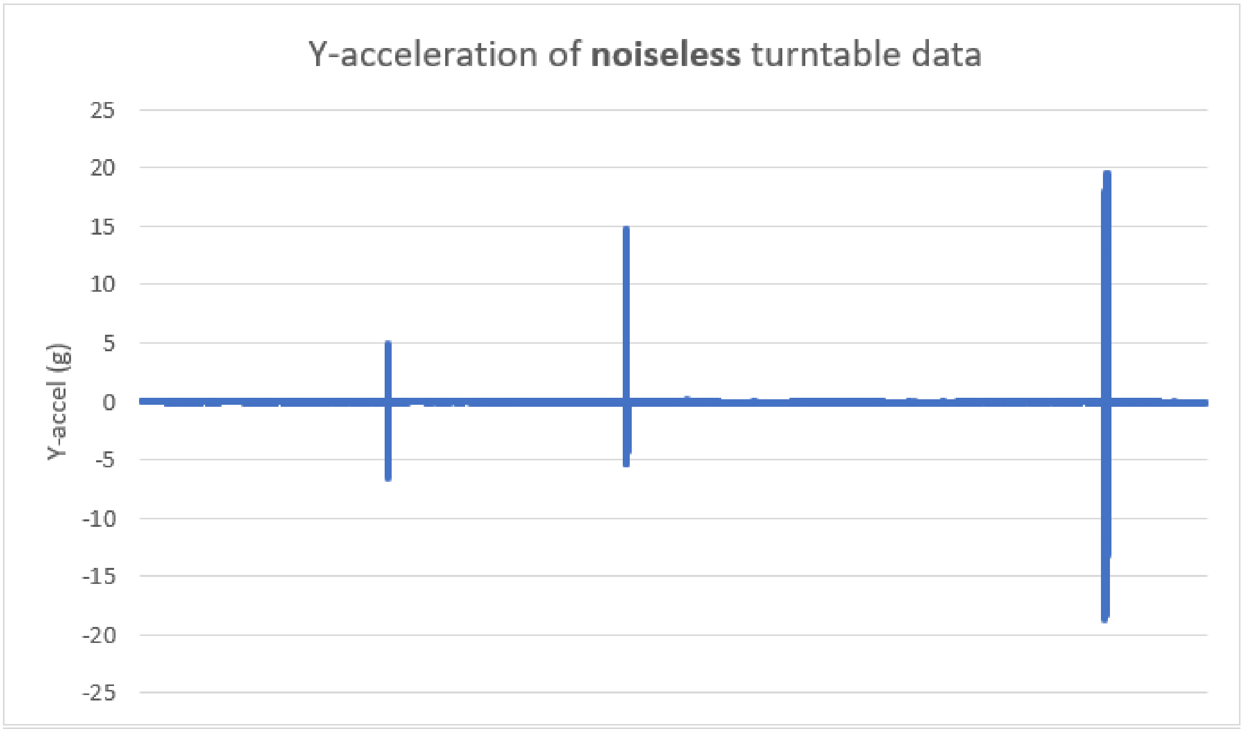
From the given example IMU data-sets [1], it is clear that IMU data is usually stored in a comma-separated-value file (.csv extension). The data-sets used will therefore include the two example IMU csv files (renamed to turntable.csv and walking.csv).

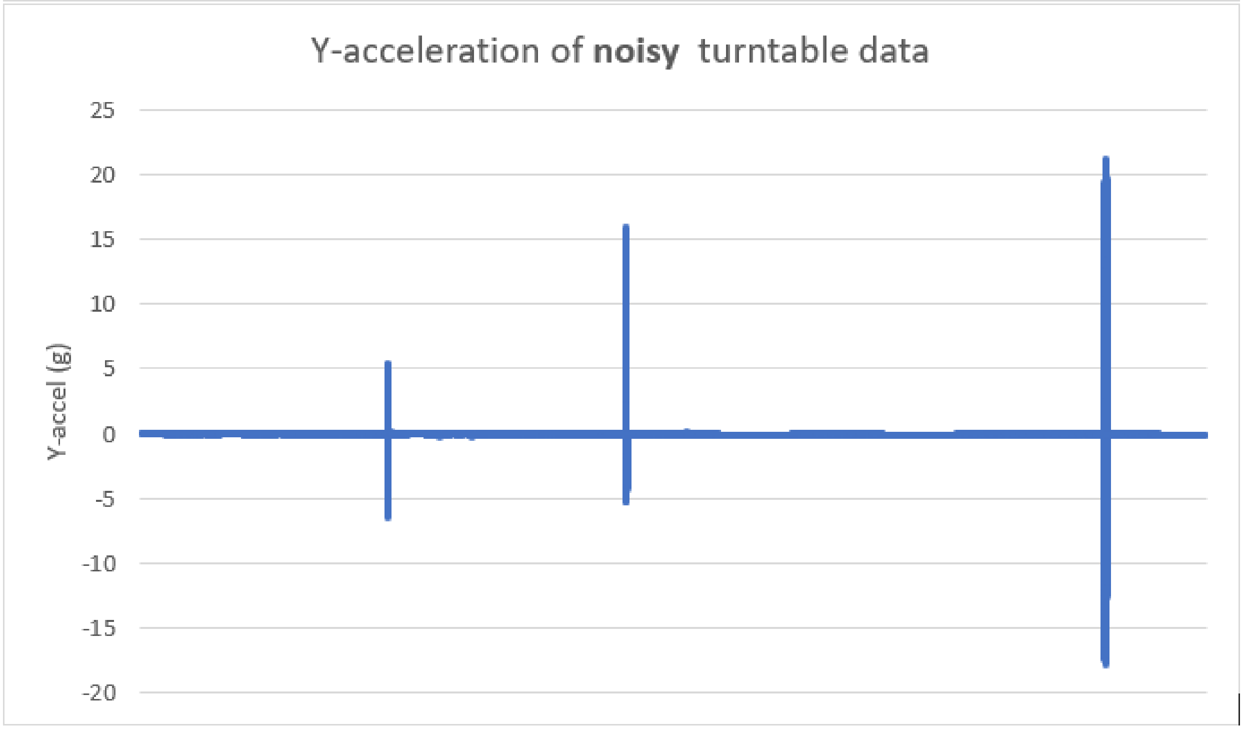
To satisfy data-type Requirement 2, generic extensionless files will also be used. These files are data logs [2] sourced from week 2's Vula resources page. These data logs differ in both file type and structure compared to the example IMU data, thus satisfying Requirement 2.

The two IMU data sets consist entirely of ASCII, whereas the data logs contain unicode characters, some of which are also ASCII characters. This variance is suitable to satisfy and exceed Requirement 1, as the algorithms are being tested on data-sets they are unlikely to interface with.

These files are all much larger than the STM's 64 KB flash memory, but that will be addressed when progress is made in implementing this simulation onto the discovery board. For the time being, these data sets will be suitable.

To investigate the effects of noise on the system's performance, a copy of turntable.csv with Gaussian noise (0-mean, standard deviation = 0.05) was created. A comparison of the two is given below, using the Y-acceleration values as an example. The entire data-set is similarly noisy.


Y-acceleration of **noiseless** turntable data
25
20
15
10
5
0
-5
-10
-15
-20
-25
Y-accel (g)


Y-acceleration of **noisy** turntable data
25
20
15
10
5
0
-5
-10
-15
-20
Y-accel (g)

3. Experimental Setup

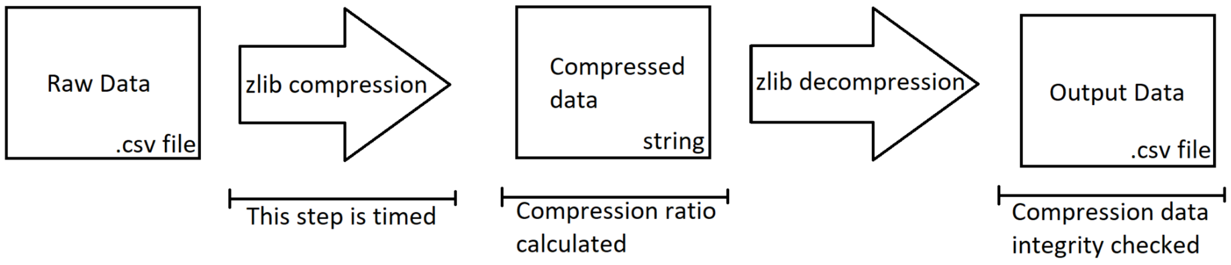
3.1 Overall Functionality

The data collected from the relative sensors is first compressed. The compression algorithm then outputs strings of compressed data which is fed into the encryption algorithm and subsequently encrypted using a constant key and transmitted. The system is designed to be small, effective and simple. When the signal is received on the other side the same key is used to decrypt the data which after being decompressed will be readable. We ran a full compression and decryption test case using the tester.csv data file we created to ensure that our compression and encryption algorithms work together as expected.

3.2 Compression (WLSCAM007)

Making use of all 3 data sets (turntable, turntablenoise, and walking), the following steps were followed:

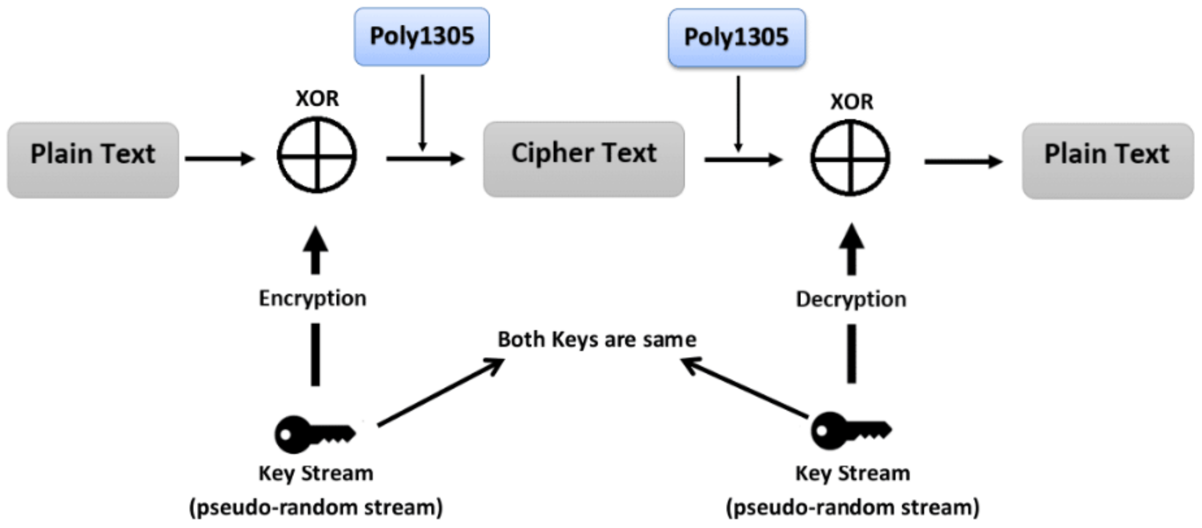
- The data sets were encrypted at each of zlib's compression levels (0 - 9).
- Timers were used to monitor the execution speed of each compression.
- The overall compression ratio was calculated for each compression.
- The compressed output data was decompressed and checked against the original.



The above steps were implemented using zlib's Python module. The timer data and compression ratios were then saved in tables and plotted later in this report.

3.3 Encryption (STRREI003)

The chacha20 encryption algorithm was tested on all 3 sets of supplied data (turntable, turntablenoise and walking)



These experiments were carried out in python simply to display the functionality of the algorithm. When executed on the STM board the encryption will be coded in C. The python cryptodome libraries were installed and the chacha20 object created. The encryption from plaintext to ciphertext was timed using the Python timer libraries.

To establish the relationship between the amount of data needing to be processed and the time taken I varied the size of the input data and recorded the time taken to encrypt and decrypt. This was then plotted using python against the relative size of the data to display its relationship. I ran a test case using a custom file tester.csv to test and ensure the integrity of the algorithm by comparing the input plain text to the decrypted plain text. I also analyzed the cipher text to ensure it was indecipherable. The key used in the simulations is generated randomly however on the actual board the key will be a constant construct which is shared between the receiver and the sender.

4. Results

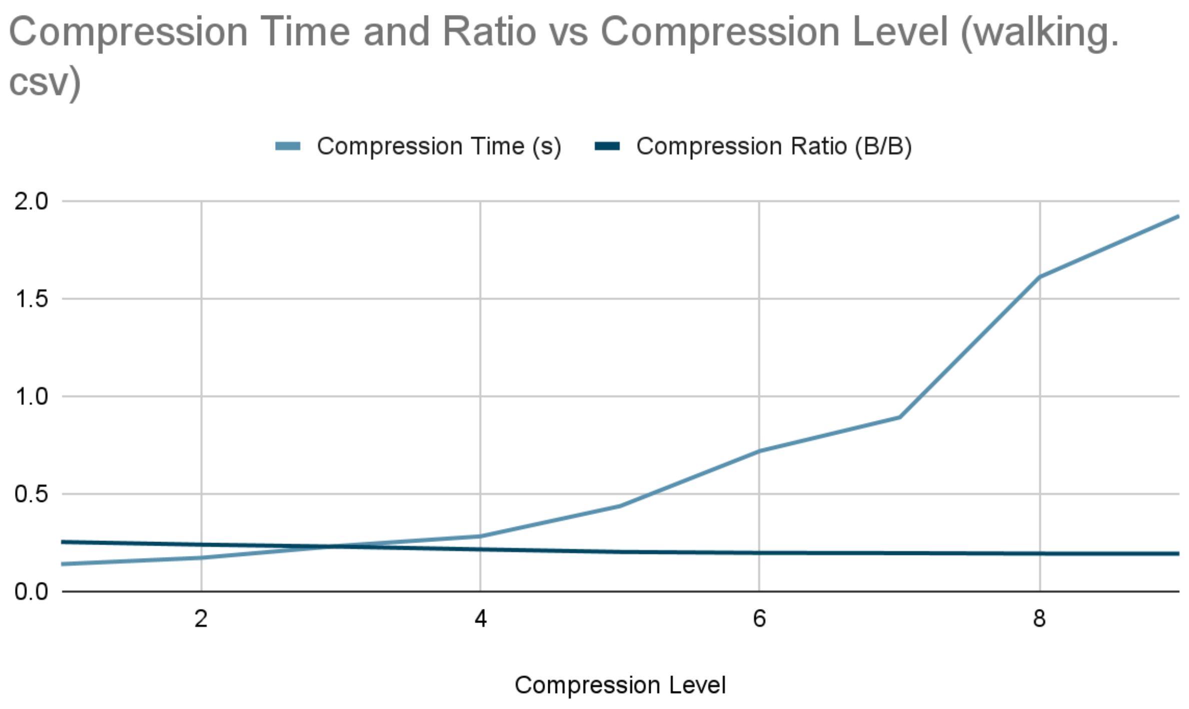
4.1 Overall Functionality

The compression and encryption algorithms are both 100% lossless, small and fast meeting all the expectations and requirements we layed out. The compression algorithm at the chosen level 1 runs on average at 0.391 seconds and the encryption algorithm for the same data set was 0.86 seconds. Therefore the total time estimate to compress and encrypt turntablenoise.csv file is 1.251 falling far below the specified requirement of 10 seconds. Due to both algorithms being 100% lossless we will receive all the Fourier Coefficients.

4.2 Compression (WLSCAM007)

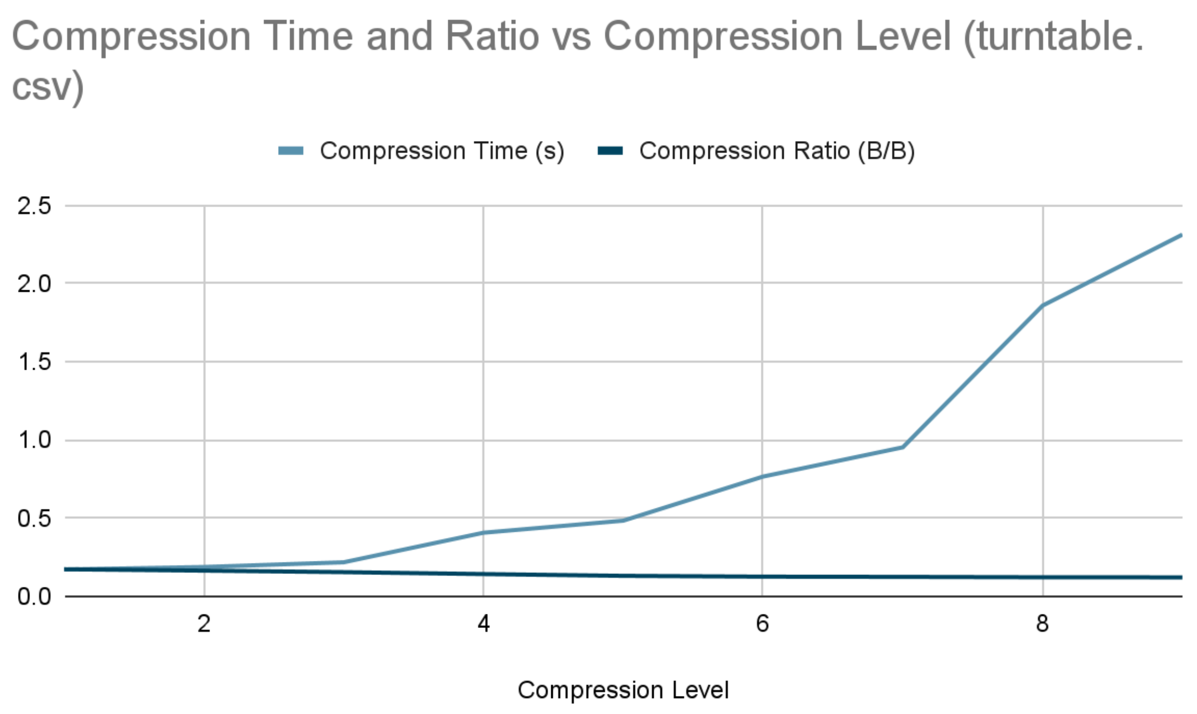
4.2.1 Walking.csv

Compression Level	Time (s)	Compression Ratio (B/B)
0 (no compression)	0.016	1.0001
1	0.14	0.2533
2	0.172	0.2397
3	0.234	0.2293
4	0.282	0.2152
5	0.437	0.2024
6	0.719	0.1975
7	0.891	0.1963
8	1.609	0.1938
9	1.922	0.1932



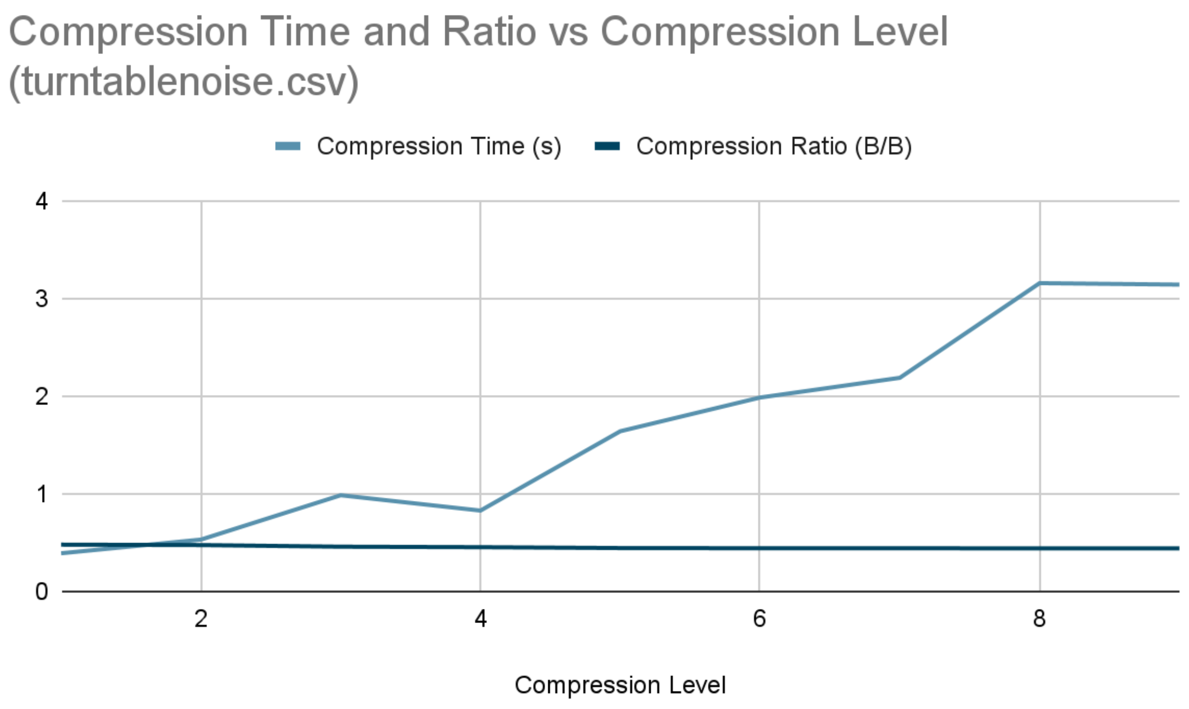
4.2.2 Turntable.csv

Compression Level	Time (s)	Compression Ratio (B/B)
0 (no compression)	0.015	1.0001
1	0.172	0.1734
2	0.188	0.1646
3	0.218	0.155
4	0.407	0.1427
5	0.484	0.1314
6	0.766	0.1262
7	0.953	0.1248
8	1.859	0.1222
9	2.313	0.1217



4.2.3 Turntablenoise.csv

Compression Level	Time (s)	Compression Ratio (B/B)
0 (no compression)	0.015	1.0001
1	0.391	0.4793
2	0.531	0.4751
3	0.985	0.4584
4	0.828	0.4534
5	1.64	0.4445
6	1.985	0.4427
7	2.187	0.4422
8	3.156	0.4418
9	3.141	0.4418



4.2.4 Discussion

The above figures highlight the trade-offs present in data compression. The use of higher compression levels decreases the compression ratio, resulting in smaller output files, but at the cost of significantly increased processing time. Zlib recommends a compression level of 6, and uses this as its default.

Using a compression level of 0 results in an output file that is slightly larger than the original. This is entirely useless for our purposes.

For our limited-power, real-time system, a compression level of 1 will be used. The compression benefits of using higher values are marginal compared to the time saved. Moving from compression level 1 to 9, the average compression ratio improvement is 23% while the processing time increases by up to 1345%

The inclusion of noise resulted in compression that was, on average, 3.62 times less effective. This is not believed to be a reflection of the random noise's effects on compression. The additive noise was not bound to the same number of decimal spaces as the original data, meaning the noisy values were significantly longer and significantly more unique. This decreases the effectiveness of all compression algorithms in general. Care will be taken to ensure that the data being compressed is rounded to as few decimal places as possible.

It is worth noting that, due to the STM’s insufficient memory, the files to be compressed will be significantly smaller than the ones investigated above. This will result in larger compression ratios. That said, the above observations are still valid and will be applied to our physical prototype.

4.3 Encryption (STRREI003)

4.3.1 Encrypt and Decrypt walking.csv

Encryption Table 1: To show the relationship between the size of the data and the speed of the encryption process for the walking.csv file.

Percentage of walking.csv ,(Number of lines)	Time (seconds)			
	Run 1	Run 2	Run 3	Average
25%, (8580)	0.145153	0.146703	0.141259	0.144372
50%, (17160)	0.272702	0.306346	0.296087	0.291712
75%, (25740)	0.414782	0.399274	0.398927	0.404328
100%, (34320)	0.556912	0.517832	0.532069	0.535604

4.3.2 Encrypt and Decrypt turntable.csv

Encryption Table 2: To show the relationship between the size of the data and the speed of the encryption process for the turntable.csv file.

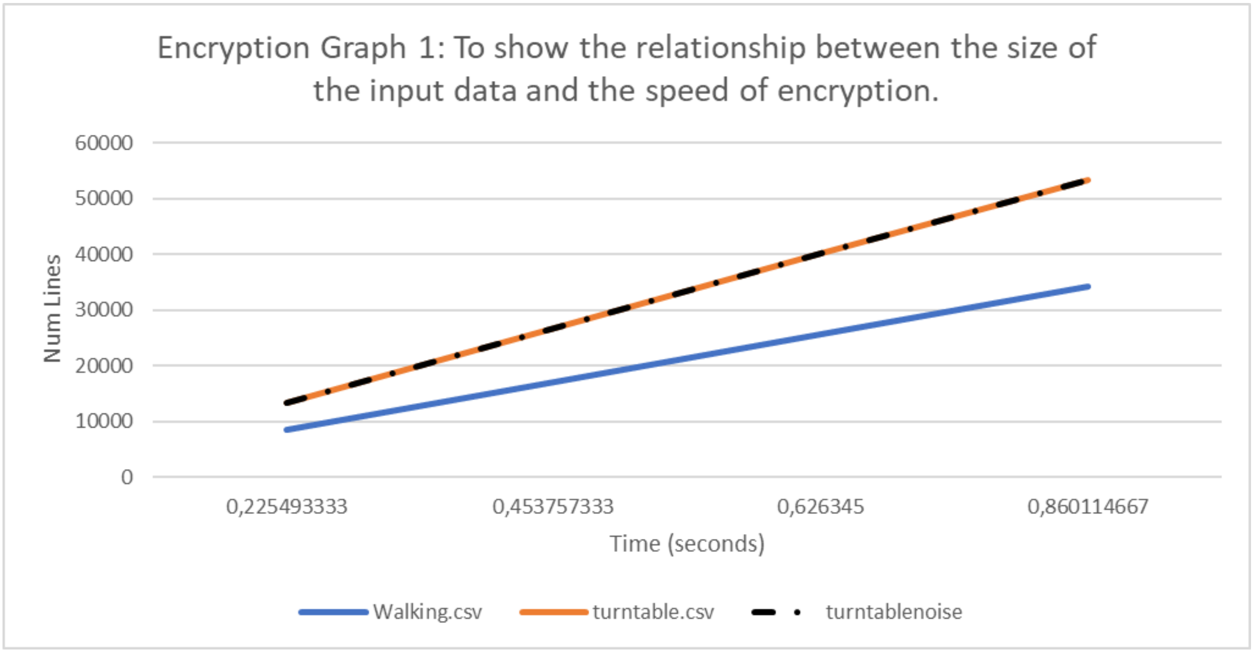
Percentage of walking.csv ,(Number of lines)	Time (seconds)			
	Run 1	Run 2	Run 3	Average
25% (13349)	0.227067	0.224817	0.211308	0.221064
50% (26698)	0.460527	0.414439	0.443077	0.439348
75% (40047)	0.611050	0.638939	0.625952	0.625314
100% (53397)	0.845476	0.820981	0.811927	0.826128

4.3.3 Encrypt and Decrypt turntablenoise.csv

Encryption Table 3: To show the relationship between the size of the data and the speed of the encryption process for the turntablenoise.csv file.

Percentage of walking.csv ,(Number of lines)	Time (seconds)			
	Run 1	Run 2	Run 3	Average
25% (13349)	0.223233	0.231559	0.221688	0.225493
50% (26698)	0.446866	0.448183	0.466223	0.453757
75% (40047)	0.634092	0.619440	0.625503	0.626345
100% (53397)	0.836545	0.861253	0.882546	0.860115

4.3.4 Encrypt and Decrypt Results



We can clearly see from the graph that as the size of the input data increases the time taken to encrypt increases linearly. This is expected as the encryption algorithm is a stream cipher and so works bit by bit with each bit taking the same amount of time. In addition to this we can see that the gaussian noise that was added had very little effect on the running of the encryption.

4.3.5 Encryption Integrity Test

To test the integrity of the encryption algorithm I created a small tester.csv file which I used to ensure none of the data was lost or compromised.

Plaintext: b" \xef\xbb\xbfName Height Gender['Reid', ' 175', ' male']['Blake', ' 174', ' male']['Kelly', ' 168', ' female']"

Key: b'C\xd8\x8d^\xf7 \xe1\xf4P\r{\x81\xe8B\xca\xb8a\xb9\x9f\x96\x06\x82#\xff\xa8!\xa0R.Wp>'

CipherText:
b'\xf7\x13\xe9bM\xce\xc4\x06$\xf1\xd0\xa2\x00\xea\x930\xa3\x025\xc8\x98q>\x15E\x9b51<I\xe9\xa2i\xa3\x9e\xc3\xd6\xdf\xfa\xbd\x0em\x99\x17\x13\x13\x7f\x8f\xc2\x90\x91\xb3\xef#=\xec\xf8\x8cR=\x00\xc6\\I\x08\xd3\xf5"\x8ah*g\x0c\xb5\xc1e\x1d/\xb6\xfb?\xd4\x83I\xe1?3\xd0\xd1g;\xfex\xfbS\xf7\x03\x86[\x91P'

Decrypted Plaintext: b" \xef\xbb\xbfName Height Gender['Reid', ' 175', ' male']['Blake', ' 174', ' male']['Kelly', ' 168', ' female']"

We can clearly see through this test that the encryption method performs as expected and is 100% lossless. The cipher text is also completely illegible meaning the encryption was performed successfully.

5. Acceptance Test Procedures

U ID	F ID	S ID	ATP ID	Explanation	Met?
U1	U1.F1	U1.F1.S1	U1.F1.S1.A1	Data recovery correctness tests will be conducted on data sets with random ASCII characters. They should match 100%	Yes
			U1.F1.S1.A2	Data recovery correctness	Yes

				tests will be conducted on data sets with random formats (to simulate different IMU outputs). They should match 100%	
U2	U2.F1	U2.F1.S1	U2.F1.S1.A1	Various data sets will be compressed and their file sizes will be compared to the original files'. There should be no compression ratios greater than 0.7	Yes
		U2.F1.S2	U2.F1.S2.A1	On-board timers will be used to record runtimes. Compression operations should not take longer than 5 seconds.	Yes
	U2.F2	U2.F2.S1	U2.F2.S1.A1	Encryption tests will be executed on sets of data that vary in size and consist of ASCII characters. The outputs will be inspected to ensure they are illegible.	Yes
		U2.F2.S2	U2.F2.S2.A1	Decrypted data will be compared to the same data before being encrypted. There should be no difference.	Yes
U3	U3.F1	U3.F1.S1	U3.F1.S1.A1	Data recovery correctness tests will be conducted on various data sets, ensuring that the recovered data contains at least the lower 25% of the Fourier coefficients of the original.	Yes. Zlib and chacha20 are lossless algorithms. 100% of the Fourier coefficients of the original data are present in the recovered data.

U4	U4.F1	U4.F1.S1	U4.F1.S1.A1	On-board timers will be used to record runtimes. All operations should complete within 10 seconds.	Yes

6. Bibliography

[1]
“Example IMU Data,” Accessed: Sep. 11, 2022. [Online]. Available: https://vula.uct.ac.za/access/content/attachment/4542c49f-47ed-40a0-8b3c-efed3756d216/Announcements/ccc3ad16-ff16-46a6-878d-146edaba26ee/EEE3097S%202022%20Walking%20Around%20Example%20Data.csv

[2]
“Data Logger,” Accessed: Sep. 11, 2022. [Online]. Available: https://vula.uct.ac.za/access/content/group/4542c49f-47ed-40a0-8b3c-efed3756d216/03.%20Lessons/Week%202/Logger%20Data.zip

[3]
“ChaCha20 and XChaCha20 — PyCryptodome 3.15.0 documentation,” *pycryptodome.readthedocs.io*. https://pycryptodome.readthedocs.io/en/latest/src/cipher/chacha20.html (accessed Sep. 13, 2022).

[4]
“Crypto.Cipher.ChaCha20,” *legrandin.github.io*. https://legrandin.github.io/pycryptodome/Doc/3.2/Crypto.Cipher.ChaCha20-module.html (accessed Sep. 13, 2022).