Second Progress Report: Milestone 3

Yashil Luckan* and Pooja Jugnarayan*
Department of Electrical and Computer Engineering
University of Cape Town

October 3rd, 2022

* Email addresses lckyas002@myuct.ac.za, and jgnpoo001@myuct.ac.za.

Table of Contents

1. IMU Module

Two IMU modules were received, one from WaveShare and one from Sparkfun. The following information is based of the Sparkfun IMU.

1.1. Salient features

The ICM-20948 will be used for testing received from sparkfun will be used for testing as it has more similar properties to the ICM-20649 which will be used on the actual buoy.
The ICM-20948 is the lowest power 9-Axis Device at 2.5mW.
The salient features of the ICM-20948 include a 3-Axis Gyroscope with Programmable FSR of ±250 dps, ±500 dps, ±1000 dps, and ±2000 dps which differs from the ICM-20649 which also has a 3-Axis Gyroscope but with programmable FSR of ±500 dps, ±100 dps, ±2000 dps, and ±4000 dps. Both ICM's have 3-Axis Accelerometers with Programmable FSR of ±4g, ±8g and ±16g but the ICM-20948 also supports ±2g whilst the ICM-20649 also supports ±30g. The difference in programable FSR with regards to the accelerometer snd gyroscope reading should not impact testing as results of similar lengths will still be obtained on both IMU's. Both ICM's have on-chip 16-bit ADCs and programmable filters, a 7 MHz SPI or a 400 kHz Fast Mode I²C, a digital-output temperature sensor, a VDD operating range of 1.71V to 3.6V, MEMS structure hermetically sealed and bonded at wafer level and are RoHS and Green compliant.

1.2. Steps to ensure accuracy between multiple IMU's

The first step when performing tests is to ensure that the IMU that we have is (if possible) configured to the same settings as the IMU used on the buoy. The tests in section 2.3 are used to validate the readings from the IMU. Where possible, the tests were conducted in the same environment with as little environmental noise as possible. We also tried to simulate the environment that the IMU would be in, in our application. All of the above was done so that the data collected by the ICM-20948 can be extrapolated to the ICM-20649.

1.3. IMU validation tests

The following tests were performed to ensure the IMU is behaving appropriately.

- Test 1: To ensure the gyroscope is behaving as expected the IMU was rotated upon each axis, namely the x, y, and z-axis by a premeasured 10 degrees and the data displayed to the PuTTY terminal was confirmed to reflect this.
- Test 2: To ensure the accelerometer is behaving as expected the IMU placed in motor vehicle accelerating from 0-30km/h in 2 seconds. This acceleration numerically evaluates to 0.4249g. This was repeated for all three axes and was achieved by just rotating the IMU. The IMU readings were compared against the calculated value to confirm the accelerometer is in working order.
- Test 3: For completeness' sake, the magnetometer was also briefly tested by increasing and decreasing the distance of a magnet with respect to the IMU. This test was done to ensure all three sensors were in working order regardless of the fact that we don't require the magnetometer for our application.

2. Experiment setup

2.1. Simulations and experiments to determine overall functionality

The following experiments were performed to determine overall functionality:

- Experiment 1: To test the reading and writing to the STM, we wrote code to cause an LED on the STM to flash.
- Experiment 2: In order to determine the overall functionality of the program, a data log file was encrypted then compressed and thereafter decrypted and decompressed. A test script was written in the C programming language to check the integrity of the data log file after this process. It was compared to the original data log file file.
- Experiment 3: To test if data can be captured from the IMU via the STM32F0, the data is from the IMU is sent to the STM via an SPI connection and will be displayed through PuTTY on a computer. This experiment consisted of the three tests mentioned in section two of this report.
- Experiment 4: To test the time taken to read from the IMU a single shot of data will captured and timed via a C timing mechanism.

2.2. Compression experiments

The following experiments were performed on a log file produced by the PuTTY output when data is captured from the IMU to test if the GZIP algorithm works:

- Experiment 1: To test if the file was successfully being read by the algorithm, GZIP was set to run on the data log file, if the file was successfully read then an output was produced in the same directory e.g., PuTTY_log_2022.10.01.txt.gz. If the file was unsuccessful the output would be a response in terminal describing the error e.g., not in gzip format.
- Experiment 2: The timing mechanism built into Unix based systems was used to time the algorithm (time) when the gzip algorithm was called from terminal. E.g., time gzip PuTTY_log_2022.10.01.txt.
- Experiment 3: A test script was written in the C programming language to check the integrity of the data log file after extraction. It was compared to the data log file before compression.

2.3. Encryption experiments

The following experiments were performed if the AES algorithm works both on a computer and the STM using data captured from the IMU:

- Experiment 1: To test if the file was successfully being read by the algorithm, a test was written to display an error if the PuTTY log file containing just the IMU data was NULL in C and to display the plaintext that will be encrypted in terminal to further prove that the file is being read
- Experiment 2: The timing mechanism built into Unix based systems was used to time the algorithm (time) when the AES algorithm was called from terminal. E.g., time gcc -o aes aes.c main.c
- Experiment 3: A test script was written in the C programming language to check the integrity of the PuTTY log file after decryption. It was compared to the PuTTY log file before encryption.
- Experiment 4: The same test script as experiment 3 will be run on the encrypted and original file to show that there are differences in the file and it is sufficiently encrypted.
- Experiment 5: With both the encryption and decryption algorithms loaded onto the STM, output will be produced in PuTTY to show that the decrypted text is the same as the data captured by the IMU. This can be observed as the data is encrypted line by line.
- Experiment 6: Multiple lines of encrypted text produced on the STM will be displayed on PuTTY and the log file will be saved as a textfile.

2.4. Data types

When data collection from the IMU is initiated in CubeIDE, data is sent as raw bits to the STM via a SPI connection. The STM then forwards these raw bits of data to the computer via a UART interface. This data is encrypted and displayed via PuTTY and saved as a .txt file. Once the data is in the .txt file, and then compressed into a .txt.gz file.

3. Results

3.1. Overall functionality

- The first experiment showed the correct LED flashing on the STM board. This shows that data and instructions can be successfully written to the STM.
- The results from the second test yielded that there were no errors or mismatches between the files before encryption and compression and after extraction and decryption. The terminal output displayed: “Errors: 0”. This means that the compression algorithm is confirmed to be lossless and that no data was lost during the encryption and decryption process.
- The third experiment showed that the data was outputted to PuTTY successfully. Not only was the data being displayed but the data was also accurate with the exception of experimental uncertainty in the three tests from section two and environmental effects such as Gaussian White Noise.
- The timing test yielded a time of 3.01 seconds.

3.2. Compression functionality

- The results from test one shows the data log file was able to be successfully compressed as no error messages such as “not in gzip format” were thrown to the terminal. Instead, the compressed file was delivered to the same directory as the original file and had the following name: PuTTY_log_2022.10.01.txt.gz. The .gz extension signifies that the file was successfully compressed. The input file size was: 479 bytes. The output file size was: 285 bytes which is 1.7 times smaller than the original file.
- The results from the second test yield that GZIP compression on the PuTTY data log took a real-world clock time of 0.008 seconds. Extraction process took 0.005 seconds. These values are an average after repeating the second experiment 2 times.
- The results from the third test yielded that there were no errors or mismatches between the files before compression and after extraction. The terminal output displayed: “Errors: 0”. This means that the compression algorithm is lossless. Errors were intentionally introduced by deleting values in order to ensure the IntegrityChecker.c file works correctly and the following output was produced after removing the headings of each value in the CSV file:

  Total Errors: 160

Line Number: 1	Error Position: 1
Line Number: 1	Error Position: 2
Line Number: 1	Error Position: 3
.	.
.	.
.	.
Line Number: 1	Error Position: 160

3.3. Encryption functionality

- There was no error message displayed in terminal when the "PuTTY.log.txt" was run through the testing algorithm. The plain text displayed in terminal matched the text in the CSV file which showed that the file was being read accurately.
- The timing algorithm was run on for only encryption of the "PuTTY.log.txt" file. This experiment was repeated three times and the real time average was 0m0.082 which is sufficient. The timing algorithm was run another three time for both encryption and decryption of the file which took an average 0m0.103s in real time. This shows that the algorithm runs in a reasonable amount of time and that encryption alone is quicker than both encryption and decryption.
- The results from the third test yielded that there were no errors or mismatches between the files before encryption and after decryption. The terminal output displayed: "Errors: 0". This shows that no data was lost during the encryption and decryption process. Errors were intentionally introduced by deleting a row in the data log file in order to ensure the IntegrityChecker.c file works correctly and the following output was produced after removing the headings of each value in the data log file:

  Line Number : 1 Error Position : 1
  Line Number : 1 Error Position : 2
  .
  .
  .
  Line Number : 1 Error Position : 451
  Total Errors : 450

- The results from the final test showed a 360504 Errors which indicates that little to no values in the encrypted and original file matched. This proves the validity of the encryption algorithm.
- When the encryption and decryption algorithms are run on the STM, the output captured from the IMU matches exactly the decrypted data and is completely different from the encrypted data which shows that the data was correctly encrypted and decrypted on the STM.
- Multiple lines of encrypted text produced on the STM were displayed on PuTTY and the log file will be saved as a textfile which was then compressed.

3.4. Changes in performance

We noticed some small differences when changing the sampling rate of reading data from the IMU. The only differences include the log file gets larger in less time when sampling at a higher rate. At higher sampling rate, more data is being collected by the IMU per unit time, thus making the IMU more sensitive to changes in the environment. For example, if the IMU was set to sample at 1Hz, and the IMU was moved many times in one second, this movement would not be logged. While higher sampling rates led to more sensitivity, this isn't always a good thing as it also becomes more susceptible to undesirable factors such as Gaussian white noise. This is due to the fact that if more data is sampled from a noisy or undesirable environment, more noise is present in the results. These factors, however, are independent of the compression and encryption processes.

4. ATP

4.1. Acceptance performance definition

Previous ATPs:	
A1	The GZIP algorithms has built in file checking functionality. If the file contents are read successfully before the compression algorithm is run then then an output file of format: .csv.gz is outputted into the same directory. If the file was unsuccessfully read or any other reason to that led to the file not being compressed, then an output to the system out will say “File not in gzip format”.
A2	A timing mechanism in Unix will be used to calculate the read and write speeds. The mechanism will encapsulate the reading from or the writing to STM block of code. The Read/write speed will be calculated by taking file size and dividing by the time taken to read/write. A pass will be indicated by a speed greater 100B/s then and a fail will be indicated as a speed less than 100B/s
A3	An if-statement will be used to test if data was read in from the file by testing if the file was empty within a testing algorithm written in C. If the file contents are read before the encryption algorithm is run successfully then then a “success” message will be displayed which will qualify as a pass. Else, a “Error” message will appear if the file was unsuccessfully read which will qualify as a fail.
A4	A test script will be written and will be run to check if the output file after extraction and decryption is the same as the raw data file. The test script will be written in C and will include and algorithm that compares every character of the output and input file and verifies that it is the same.
A5	A timing mechanism in Unix will be used to calculate the runtime of the compression algorithm. The mechanism will encapsulate the entire algorithm. A pass will be indicated by a time less than 1.1s and a fail will be indicated as a time greater than 1.1s.
A6	A timing mechanism in C will be used to calculate the runtime of the encryption algorithm. The mechanism will encapsulate the entire algorithm. A pass will be indicated by a time less than 1s and a fail will be indicated as a time greater than 1s.
A7	Once the file is written onto the STM, it will be sent back to the computer to ensure that the file sent to the STM is indeed correct. A test script will be written and will be run to check if the input file is the same as the raw data file sent to the STM. The test script will be written in C and will include and algorithm that compares every character of the two files and verifies that it is the same.

Updated ATPs:	
A1	The GZIP algorithms has built in file checking functionality. If the file contents are read successfully before the compression algorithm is run then then an output file of format: .txt.gz is outputted into the same directory. If the file was unsuccessfully read or any other reason to that led to the file not being compressed, then an output to the system out will say “File not in gzip format”.
A2	A timing mechanism in STMCubeIDE will be used to calculate the initialisation, read and write time of the IMU to the STM and for the necessary files to be downloaded to the STM. A pass will be granted if the time taken to write to the STM and receive the firs reading is completed in under 20 seconds and a failure otherwise.
A3	An if-statement will be used to test if data was read in from the file by testing if the file was empty within a testing algorithm written in C. If the file contents are read before the encryption algorithm is run successfully then then a “success” message will be displayed which will qualify as a pass. Else, a “Error” message will appear if the file was unsuccessfully read which will qualify as a fail.
A4	A test script will be written and will be run to check if the output file after extraction and decryption is the same as the raw data file. The test script will be written in C and will include and algorithm that compares every character of the output and input file and verifies that it is the same.
A5	A timing mechanism in Unix will be used to calculate the runtime of the compression algorithm. The mechanism will encapsulate the entire algorithm. A pass will be indicated by a time less than 1.1s and a fail will be indicated as a time greater than 1.1s.
A6	A timing mechanism in C will be used to calculate the runtime of the encryption algorithm. The mechanism will encapsulate the entire algorithm. A pass will be indicated by a time less than 1s and a fail will be indicated as a time greater than 1s.
A7	Once the file is written onto the STM, it will be sent back to the computer to ensure that the file sent to the STM is indeed correct. A test script will be written and will be run to check if the input file is the same as the raw data file sent to the STM. The test script will be written in C and will include and algorithm that compares every character of the two files and verifies that it is the same.

4.2. ATP Performance

Updated ATPs:	
A1	When the compression algorithm was run, no errors were observed and thus this ATP is being met.
A2	The time taken to download the data to the STM and receive the first data reading from the IMU (includes initialisation and read/write times) was 3.01 seconds which is under the 20 seconds threshold and thus this ATP is a pass.
A3	When the encryption algorithm was run, no errors were observed and the original text was displayed to be the same as the text entered and thus this ATP is being met.
A4	The file before encryption and compression was exactly the same as the file after decryption and extraction and thus this ATP is met as integrity is fully preserved.
A5	The time taken for the whole compression algorithm to run was less than 1s and thus this ATP is met.
A6	The time taken for the whole encryption algorithm to run was less than 1s and thus this ATP is met.
A7	This ATP could not be fully tested as no compression or encryption algorithms were written to the STM at this stage in testing.

4.3. Traceability matrix

UR	FR	Specs	ATP	Pass/Fail
U1	U1.F1	U1.F1.S1	U1.F1.S1.A1	Pass
	U1.F2	U1.F2.S2	U1.F2.S2.A2	Pass
	U1.F3	U1.F3.S3	U1.F3.S3.A3	Pass
	U1.F4	U1.F4.S4	U1.F4.S4.A2	Pass
U2	U2.F5	U2.F5.S5	U2.F5.S5.A4	Pass
	U2.F6	U2.F6.S6	U2.F6.S6.A4	Pass
	U2.F7	U2.F7.S7	U2.F7.S7.A5	Pass
	U2.F8	U2.F8.S8	U2.F8.S8.A5	Pass
U3	U3.F9	U3.F9.S9	U3.F9.S9.A6	Pass
	U3.F10	U3.F10.S10	U3.F10.S10.A6	Pass
	U3.F11	U3.F11.S11	U3.F11.S11.A6	Pass
	U3.F12	U3.F12.S12	U3.F12.S12.A6	Pass
U4	U4.F13	U4.F13.S13	U4.F13.S13.A7	Pass
	U4.F14	U4.F14.S14	U4.F14.S14.A7	Pass
	U4.F15	U4.F15.S15	U4.F15.S15.A7	Pass
	U4.F16	U4.F16.S16	U4.F16.S16.A7	Pass

5. References

[1]Research.cs.wisc.edu, 2022. [Online]. Available: https://research.cs.wisc.edu/wpis/examples/pcca/gzip/gzip.c. [Accessed: 12- Sep- 2022].

[2]"GitHub - openluopworld/aes_128: Implementation of AES-128 in pure C. No modes are given. Only one block of encryption and decryption is given here.", GitHub, 2022. [Online]. Available: https://github.com/openluopworld/aes_128. [Accessed: 11- Sep- 2022].

[3]S. (Qwiic), W. Well, G. IMU and A. Worthless, "SparkFun 9DoF IMU Breakout - ICM-20948 (Qwiic) - SEN-15335 - SparkFun Electronics", Sparkfun.com, 2022. [Online]. Available: https://www.sparkfun.com/products/15335. [Accessed: 03- Oct- 2022].