521479S Software project (2025) - Hack the dataset from X200GO - 3D modelling sensor

Hack the dataset from X200GO -3D modelling sensor

Final Report

1.1 Version

Name	Student number
Sami Häkkilä	2504436
Syed Amin Al Mahim	2510954
Muhammad Huzaifa	2509384

1. Executive Abstract

This work addresses the challenge of extracting structured information from undocumented binary file formats originating from a Simultaneous Localization and Mapping (SLAM) system. These files encapsulate sensor data, including inertial measurements, raster imagery, global navigation satellite system (GNSS) positioning, and LiDAR-based spatial information (point clouds). In SLAM systems, data from such sensors are fused to produce an accurate and consistent three-dimensional representation of the environment. GPS provides a global position reference and long-term stability, but its accuracy and availability are limited, especially indoors. IMU sensors deliver high-frequency motion and orientation data, enabling short-term pose estimation, although they suffer from cumulative drift over time. LiDAR directly measures environmental geometry through range measurements, allowing the construction of dense and accurate point clouds.

If raw sensor data from SLAM devices can be accessed, users gain the ability to implement custom SLAM and mapping algorithms independent of vendor-specific processing. This significantly expands the device's applicability for research purposes. The lack of manufacturer-provided specifications for these proprietary file formats necessitates a systematic methodology for reverse engineering. This project explores established techniques for analyzing unknown binary structures, drawing on insights from scientific literature, and prior research in data parsing and sensor fusion [7]. In addition, it presents our practical strategies developed during the project, detailing the tools, technologies, and custom software implementations employed to interpret and extract meaningful content from the data. By providing a framework for understanding and processing these data sources, the work aims to facilitate innovation in SLAM-related applications and support the development of more robust and adaptive algorithms.

Contents

2. Introduction

Advancements in mobile mapping and autonomous sensing technologies have led to the widespread use of multi sensor platforms that combine LiDAR, Global Navigation Satellite Systems (GNSS), Inertial Measurement Units (IMU) and camera sensors to generate accurate three dimensional (3D) representations of environments [3]. One such system is the Stonex X200 GO *(Fig: 1)*, a portable SLAM based 3D mapping sensor designed for rapid data acquisition in both indoor and outdoor environments [8].

The X200 GO system integrates multiple sensors, including a rotating LiDAR scanner, cameras, IMU and GNSS. During operation, the sensor records large volumes of raw data that are later processed using the manufacturer's proprietary software, *Slam Go Post*, to produce point clouds, trajectories and raster images [8]. However, the raw sensor outputs are stored in proprietary and undocumented binary file formats, which prevents direct access to the underlying data without relying on the vendor's closed source software.

Figure 1: Stonex X200 GO

The primary objective of this project is to reverse engineer the binary data formats generated by the X200 GO sensor [8] and to develop software tools capable of parsing and converting these files into human readable and standardized formats. Achieving this would enable independent data analysis, visualization and integration with open source tools, while also supporting further research and algorithm development beyond the constraints of the proprietary system.

Unlike conventional software development projects with predefined specifications, this work is inherently research-oriented and exploratory in nature. No official documentation or reference implementation was available for decoding the X200 GO binary files, and there was no clear or straightforward method to interpret the stored sensor data. As a result, the project required a brute-force and hypothesis-driven approach, where multiple decoding strategies were formulated, tested, and iteratively refined based on observed patterns and validation results.

The X200 GO generates multiple file types, each corresponding to different sensor modalities, including LiDAR point cloud data, IMU measurements, GNSS navigation data, and raster image data. These files typically consist of a binary header followed by structured data records, often combining raw binary payloads with embedded ASCII metadata. Understanding the internal structure of these files therefore required systematic experimentation rather than direct implementation.

To address this challenge, the project adopts an experimental, data-driven reverse engineering methodology, combining low level binary inspection, pattern recognition, exhaustive brute force decoding and visualization based validation. Python was selected as the primary development language due to its strong support for binary data handling, numerical analysis,

and data visualization. Interactive Jupyter and Google Colab notebooks were used for rapid experimentation and hypothesis testing, while standalone Python scripts were developed to ensure reproducibility and structured analysis.

While complete decoding of all sensor modalities is a complex and high risk task, the project demonstrates meaningful progress in identifying file structures, frame boundaries and plausible data representations, particularly for GNSS related navigation data [3]. The outcomes of this research-oriented effort provide a solid foundation for future work aimed at fully open and vendor independent parsing of X200 GO sensor data.

3. Customer

The customer for this project is Rajesh Raveendran *(Rajesh.Raveendran@oulu.fi),* affiliated with the University of Oulu. The customer's primary requirement is the ability to access and interpret raw sensor data produced by the Stonex X200 GO SLAM system without relying on the manufacturer's proprietary processing software (*Slam Go Post*). This includes extracting and understanding data from multiple sensor modalities such as LiDAR, GNSS, IMU, and camera systems, and converting the decoded outputs into human-readable or standard formats suitable for further analysis, visualization, and research.

4. Acronyms and Definitions

Acronym/Term	Definition
ASCII	American Standard Code for Information Interchange
GNGGA	NMEA Standard, contains time, date, position, fix quality, number of satellites and horizontal dilution of precision.
GNRMC	NMEA Standard, contains time, date, position, positioning mode, course over ground, and speed.
GNSS	Global Navigation Satellite System
HEX	Hexadecimal, a base-16 numbering system.
IMU	Inertial Measurement Unit
NMEA	National Marine Electronics Association
SLAM	Simultaneous Localization and Mapping
LIDAR	Light Detection and Ranging

5. Project Description

This project focuses on the reverse engineering of proprietary binary data formats generated by the Stonex X200 GO SLAM sensor. Due to the absence of official documentation and the closed nature of the manufacturer's software, the project required research oriented and exploratory approach rather than a conventional implementation driven workflow.

The core task of the project was to analyze raw binary files produced by the X200 GO system and to develop software tools capable of extracting meaningful sensor data without relying on

the proprietary Slam Go Post application. The work involved systematic experimentation, pattern recognition, brute force decoding, and validation through visualization.

The project addressed multiple sensor data types, including IMU, GNSS, LiDAR, and raster image data, each of which posed different technical challenges due to variations in structure, encoding, and data representation.

5.1 Approaches

Given the lack of a predefined decoding strategy, multiple complementary approaches were adopted [7].

5.1.1 Binary Inspection

The initial inspection of the IMU file was performed using Microsoft's Hex Editor extension in Visual Studio Code. Figure *(Fig: 2)*, below displays the first rows of the binary IMU file in hexadecimal format. On the left, we see the hex values, with each row containing 32 addresses, while on the right, the corresponding ASCII decoded text per each address value is shown. At first glance, we can identify human readable strings such as "feimarobotics-slam imu" "feima-1200-imu," "fm-i2000," as well as a datetime string that likely represents the exact start time of the SLAM project. Search engines we used did not provide any reliable or useful data sources when we tried to investigate these strings found in the first rows of the binary file.

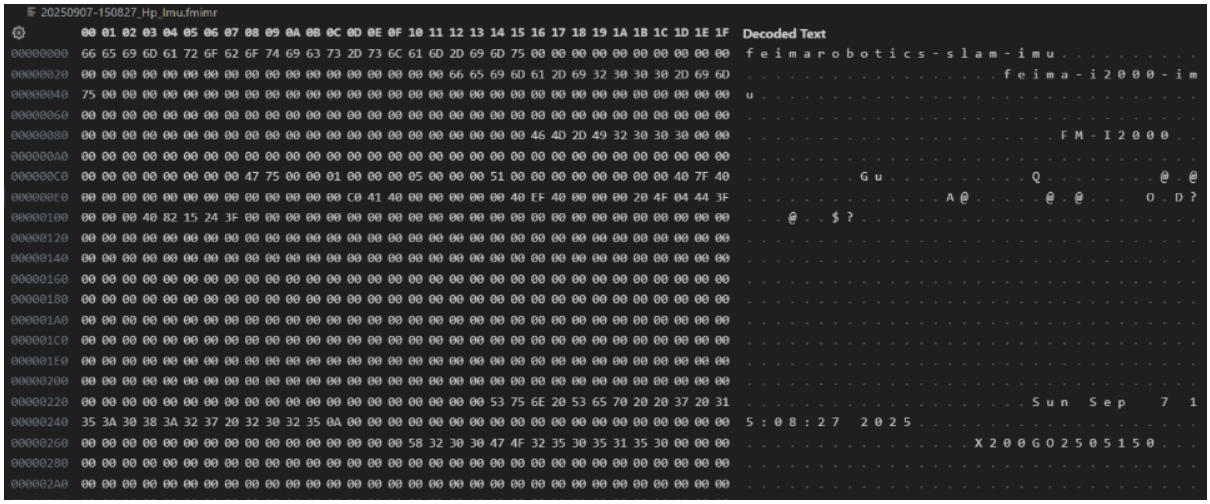

Figure 2: IMU data in hexadecimal

We continued examining the data file and as we scrolled further, some visible patterns began to emerge. As shown in Figure x, a distinct pattern in the ASCII representation starts at address ’0x400’, which corresponds to ’1024’ in decimal. Based on this observation, we can reasonably assume that the header of the binary file is 1024 bytes long (1 kilobyte) and that the actual data begins immediately after this header. This also applied to files other than the IMU file.

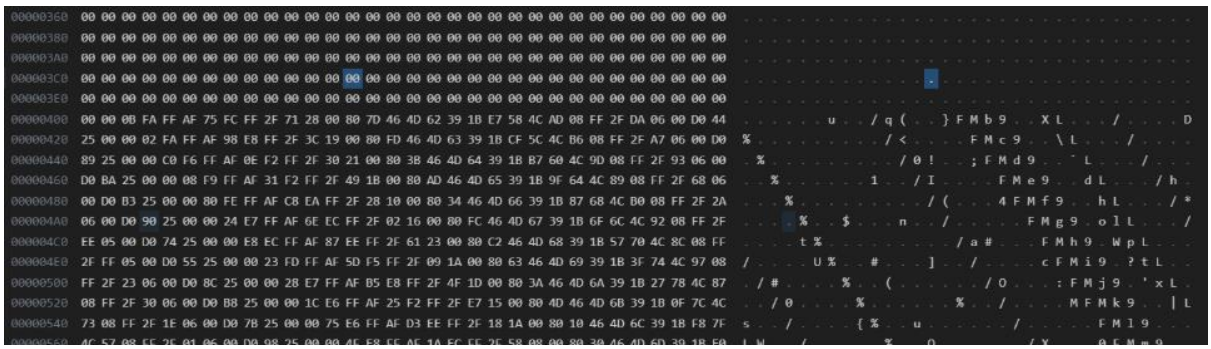

Figure 3

A closer look at ASCII decoded part (*Fig: 3)* reveals that certain characters appear at regular intervals. For example, the ‘%’ symbol occurs consistently throughout the data. In this figure, each row is 32 bytes long, which is the maximum display width for the hex editor used in the visualization. Observing the placement of the ‘%’ characters suggests that each record in the data might be 33 bytes in length. Based on this assumption, we decided to reorganize all data following the header into chunks of 33 bytes. For this purpose, we created a Python program that reads the data file, skips 1024 bytes first and then puts each 33-byte record into Python array [8].

Figure *(Fig: 4)* illustrates printing first 25 elements of the array in the Python program. Based on the assumption of 33 bytes long records, we started implementing the first hacking approach which is documented in the next chapter.

```
00 00 0b fa ff af 75 fc ff 2f 71 28 00 80 7d 46 4d 62 39 1b e7 58 4c ad 08 ff 2f da 06 00 d0 44 25
00 00 02 fa ff af 98 e8 ff 2f 3c 19 00 80 fd 46 4d 63 39 1b cf 5c 4c b6 08 ff 2f a7 06 00 d0 89 25
00 00 c0 f6 ff af 0e f2 ff 2f 30 21 00 80 3b 46 4d 64 39 1b b7 60 4c 9d 08 ff 2f 93 06 00 d0 ba 25
00 00 08 f9 ff af 31 f2 ff 2f 49 1b 00 80 ad 46 4d 65 39 1b 9f 64 4c 89 08 ff 2f 68 06 00 d0 b3 25
00 00 80 fe ff af c8 ea ff 2f 28 10 00 80 34 46 4d 66 39 1b 87 68 4c b0 08 ff 2f 2a 06 00 d0 90 25
00 00 24 e7 ff af 6e ec ff 2f 02 16 00 80 fc 46 4d 67 39 1b 6f 6c 4c 92 08 ff 2f ee 05 00 d0 74 25
00 00 e8 ec ff af 87 ee ff 2f 61 23 00 80 c2 46 4d 68 39 1b 57 70 4c 8c 08 ff 2f ff 05 00 d0 55 25
00 00 23 fd ff af 5d f5 ff 2f 09 1a 00 80 63 46 4d 69 39 1b 3f 74 4c 97 08 ff 2f 23 06 00 d0 8c 25
00 00 28 e7 ff af b5 e8 ff 2f 4f 1d 00 80 3a 46 4d 6a 39 1b 27 78 4c 87 08 ff 2f 30 06 00 d0 b8 25
00 00 1c e6 ff af 25 f2 ff 2f e7 15 00 80 4d 46 4d 6b 39 1b 0f 7c 4c 73 08 ff 2f 1e 06 00 d0 7b 25
00 00 75 e6 ff af d3 ee ff 2f 18 1a 00 80 10 46 4d 6c 39 1b f8 7f 4c 57 08 ff 2f 01 06 00 d0 98 25
00 00 4f e8 ff af 1a ec ff 2f 58 08 00 80 30 46 4d 6d 39 1b e0 83 4c 3f 08 ff 2f f9 05 00 d0 df 25
00 00 dc f3 ff af 22 f2 ff 2f 25 09 00 80 b7 46 4d 6e 39 1b c8 87 4c 41 08 ff 2f d2 05 00 d0 a1 25
00 00 f6 f0 ff af 3e ea ff 2f 0b 0e 00 80 57 46 4d 6f 39 1b b0 8b 4c 68 08 ff 2f d0 05 00 d0 60 25
00 00 60 f2 ff af 6d e6 ff 2f d9 f0 ff 8f 7d 46 4d 70 39 1b 98 8f 4c 44 08 ff 2f 67 05 00 d0 4e 25
00 00 88 e6 ff af 28 ee ff 2f 1c 13 00 80 02 46 4d 71 39 1b 80 93 4c 63 08 ff 2f f6 04 00 d0 50 25
00 00 90 ee ff af 84 e5 ff 2f 76 f4 ff 8f 4a 46 4d 72 39 1b 68 97 4c 9d 08 ff 2f cd 04 00 d0 67 25
00 00 b1 eb ff af e9 e7 ff 2f 0d fc ff 8f 83 46 4d 73 39 1b 51 9b 4c 5f 08 ff 2f e5 04 00 d0 89 25
00 00 8e fb ff af 10 ef ff 2f 4e 01 00 80 c1 46 4d 74 39 1b 39 9f 4c 6f 08 ff 2f fb 04 00 d0 ae 25
00 00 ee fe ff af 80 ed ff 2f a7 e7 ff 8f 17 46 4d 75 39 1b 20 a3 4c 27 08 ff 2f 0c 05 00 d0 90 25
00 00 6d 0a 00 a0 b5 e4 ff 2f d5 ef ff 8f 8e 46 4d 76 39 1b 08 a7 4c 2b 08 ff 2f e7 04 00 d0 d6 25
00 00 f1 ff ff af 05 f5 ff 2f f7 d5 ff 8f 8f 46 4d 77 39 1b f0 aa 4c 6d 08 ff 2f c0 04 00 d0 a8 25
00 00 e1 fb ff af 00 f9 ff 2f f0 dd ff 8f 54 46 4d 78 39 1b d9 ae 4c 9b 08 ff 2f b8 04 00 d0 71 25
00 00 23 08 00 a0 38 f1 ff 2f 2e cc ff 8f cf 46 4d 79 39 1b c1 b2 4c 9f 08 ff 2f b7 04 00 d0 51 25
00 00 08 17 00 a0 4b f2 ff 2f c6 dd ff 8f 50 46 4d 7a 39 1b a9 b6 4c 86 08 ff 2f 9c 04 00 d0 8d 25
```

Figure 4: First 25 rows

5.1.2 Frame Based Testing

Once the IMU data had been segmented into fixed size records, statistical analysis of the individual records became possible. As an initial approach, a visualization-driven method was adopted, where decoded values were plotted to identify plausible sensors like behavior. This was implemented using Python, which was a natural choice due to its extensive data processing libraries and convenient visualization tools such as Matplotlib and Seaborn.

Based on earlier observations, the file was assumed to contain a 1024-byte header, followed by data records of 33 bytes each. Within each record, byte sequences of two to four bytes of length were interpreted using multiple candidate data types to evaluate potential representations of sensor measurements. Five possible datatypes are:

- Signed 16-bit integers (int16)
- Unsigned 16-bit integers (uint16)
- Signed 32-bit integers (int32)
- Unsigned 32-bit integers (uint32)
- 32-bit floating point (float32)

For each data type, both little endian and big-endian formats were tested in the Python program [8]. Additionally, we vary the starting offset within the 33 byte block: for each data type, we attempt unpacking from every valid starting byte. This results in a total of:

$$5 \text{ data types} \times 2 \text{ endianness} \times 32 \text{ offsets} = 320 \text{ combinations}$$

For every possible combination, first 500 values are plotted. For example, figure *(Fig: 5)* shows the first 500 rows with the 20th and 21st bytes interpreted as a 16 bit signed integer in little-endian format. Figure *(Fig: 6)* represents 27th, 28th, 29th and 30th bytes as unsigned 32-bit integer in big-endian format. Figure *(Fig: 8)* shows data in Excel format where each row corresponds expected 33 bytes long record of the data and each column corresponds an offset to the first byte of the record.

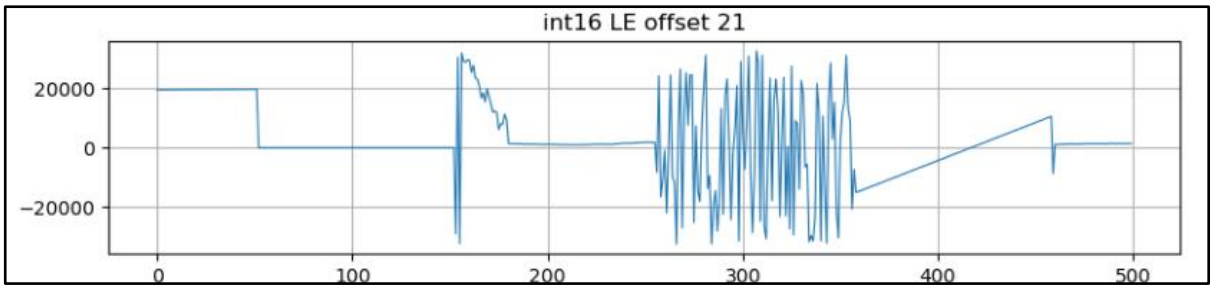

Figure 5: 20th & 21st byte interpreted for first 500 rows (16-bit signed int & little-endian order)

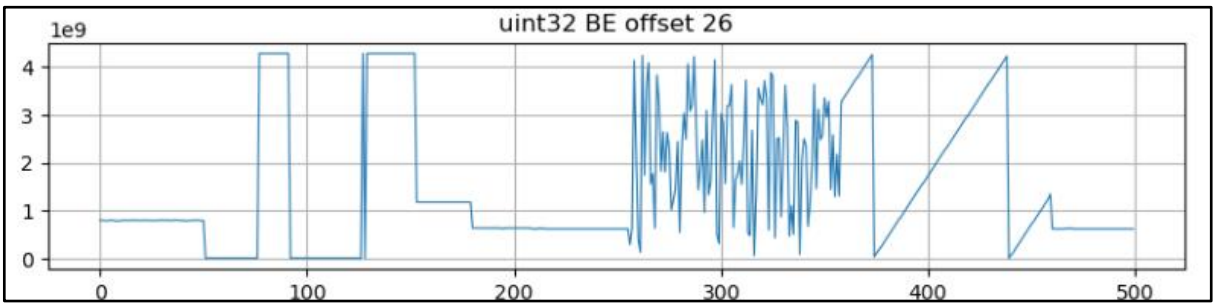

Figure 6: 27th to 30th byte interpreted for first 500 rows (32-bit unsigned int & big-endian order)

Byte_26	Byte_27	Byte_28	Byte_29	Byte_30	Byte_31	Byte_32
255	47	218	6	0	208	68
255	47	167	6	0	208	137
255	47	147	6	0	208	186
255	47	104	6	0	208	179
255	47	42	6	0	208	144

Byte_20	Byte_21	Byte_22	Byte_23	Byte_24	Byte_25
27	231	88	76	173	8
27	207	92	76	182	8
27	183	96	76	157	8
27	159	100	76	137	8
27	135	104	76	176	8
27	111	108	76	146	8

Figure 7

As a conclusion or result of the visualization approach, none of the interpreted byte combinations produced a plot that could be clearly interpreted as data from an accelerometer or gyroscope. After this, we tried using a pair of consecutive zero bytes (00 00) as a delimiter for the data records instead of assuming that each record was 33 bytes long. Upon closer analysis, we observed that this pair of zero bytes appears throughout the entire dataset, from start to finish. As shown in Figure 4 *(Fig: 4)*, in the first rows of the data, a pair of consecutive zero bytes appears at intervals of 33 bytes.

When we visualized the two bytes following the 00 00 sequence as an uint16 data type, the resulting plot started to look significantly clearer and possibly resembled accelerometer or gyroscope data. However, the visualization still contained ambiguities, and it was evident that this was not yet the final solution. Furthermore, when we split the records based on the 00 00 sequence, the number of different lengths of records turned out to be in the dozens. Nevertheless, the histogram showed that approximately 96% of the records (assuming each record starts with 00 00) were exactly 33 bytes long. This gave us confidence that we were on the right track.

Table X presents randomly selected segments of the dataset, split based on the occurrence of two consecutive zero bytes (00 00). The table reveals clear patterns of recurring values, such as the bytes at offsets 16–17 and 19–20. An exception is the row highlighted in yellow, which is significantly shorter than the others. However, this row still contains the byte sequences 464d and 391b. From this observation, we inferred that the positions of different measurement values and information are not fixed relative to the 00 00 delimiter but rather associated with specific byte sequences, which we will refer to as flags in the following discussion

Offset	0-1	2-3	4-6	7-8	9-10	11-12	13-14	15	16-17	18	19-20	21-23	23-24	25-26	27-28	29-30	31-32
	0000	1820	00a0	320a	0020	2733	0080	0a	464d	52	391b	bd8a63	a808	ff2f	ee05	00d0	fe25
	0000	7b21	00a0	d009	0020	2733	0080	f9	464d	58	391b	2ca263	9e08	ff2f	ac05	00d0	c525
	0000	80ab	464d	ec	391b	46	fc61	b108	ff2f	4b06	00d0	cd25					
	0000	870c	00a0	fafe	ff2f	fae4	ff8f	44	464d	b9	391b	1a1d65	6808	ff2f	8004	00d0	2926
	0000	a057	ffaf	b8f5	ff2f	76dc	ff8f	4a	464d	a8	391b	bfc268	b208	ff2f	b406	00d0	e825
	0000	399f	ffaf	1eca	ff2f	583a	0080	97	464d	1b	391b	52389b	fb09	ff2f	ea07	00e0	5126
	0000	30ec	fbaf	e523	0120	e002	fe9f	38	464d	d3	391b	456136	12c3	fe2f	fc09	00f0	791c
	0000	b050	feaf	0dfc	fd2f	f59e	0090	e3	464d	da	391b	e7a6d6	c200	ff2f	dd03	00f0	992a
	0000	e8dd	00a0	e2c8	0020	4ec6	0590	64	464d	4b	391b	f307bd	b902	ff2f	f502	00f0	9f34
	0000	7b3a	00a0	a5f4	ff2f	3ab5	ff8f	5f	464d	85	391b	a1e14c	7608	ff2f	e504	00d0	c925

Table 1: Repeated $GNGGA and $GNRMC pattern

Next, we investigated whether each data segment could be marked by a specific flag within the dataset. This idea originated from the constant byte sequences '464d' and '391b', which always appear together but may occur at different offsets relative to the first byte. When we extracted the byte following the '464d' flag from each record and combined it with the three bytes following the '391b' flag, we obtained a clear and uniform sawtooth pattern that extended precisely across the entire dataset. This pattern represents a counter that increments up to the maximum uint32 value (i.e., 2^{32}) and then resets to zero. Figure X illustrates the visualization of this counter throughout the full length of the IMU file.

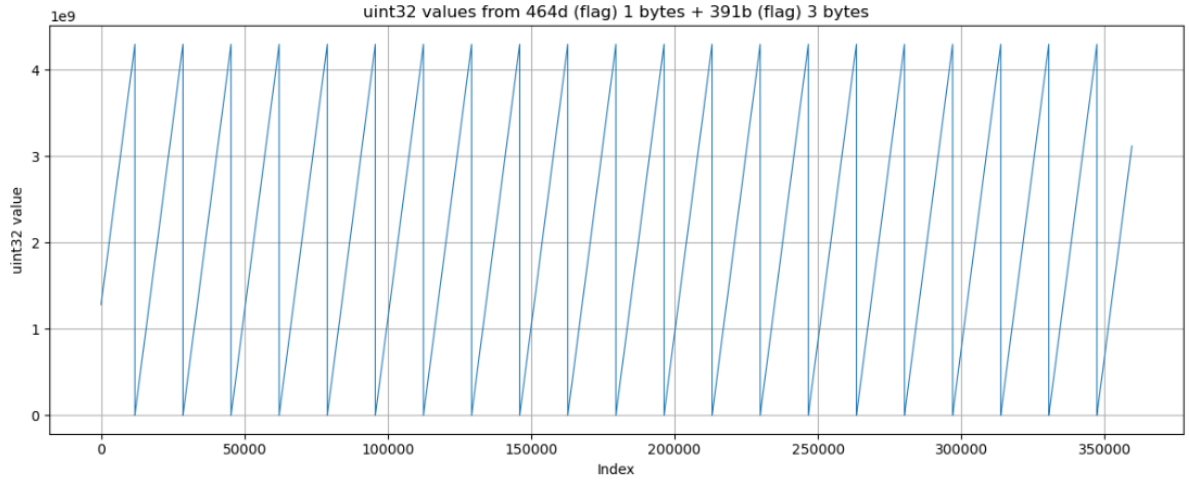

Figure 8: 32-bit ticker/timer value plotted throughout the data.

After extracting clear information from the binary file—specifically the 32-bit counter—we proceeded to investigate other potential data, such as accelerometer and gyroscope readings. We did not identify a specific flag for these values; instead, they appeared to reside at fixed offsets within the 33-byte records. Figure *(Fig: 8)* likely represents accelerometer data collected throughout the entire SLAM measurement. The SLAM project folder contained a video of the recording session, which lasted approximately five minutes. By comparing the turns and movements visible in the video with the presumed accelerometer data, we concluded with high confidence that the values indeed correspond to accelerometer measurements.

Figure *(Fig: 9)* shows a sensor value parsed from the dataset using a parser implemented in Python [8]. For plotting purposes, the data point indices on the x-axis were converted to seconds, corresponding to the duration of the SLAM recording when the X200GO device was mounted on a drone during flight. The average value of the data is approximately 9810, and dividing this by 100 yields 9.81, which is close to the standard gravitational acceleration on Earth. This strongly suggests that the data represents measurements from one axis of the accelerometer in the IMU sensor.

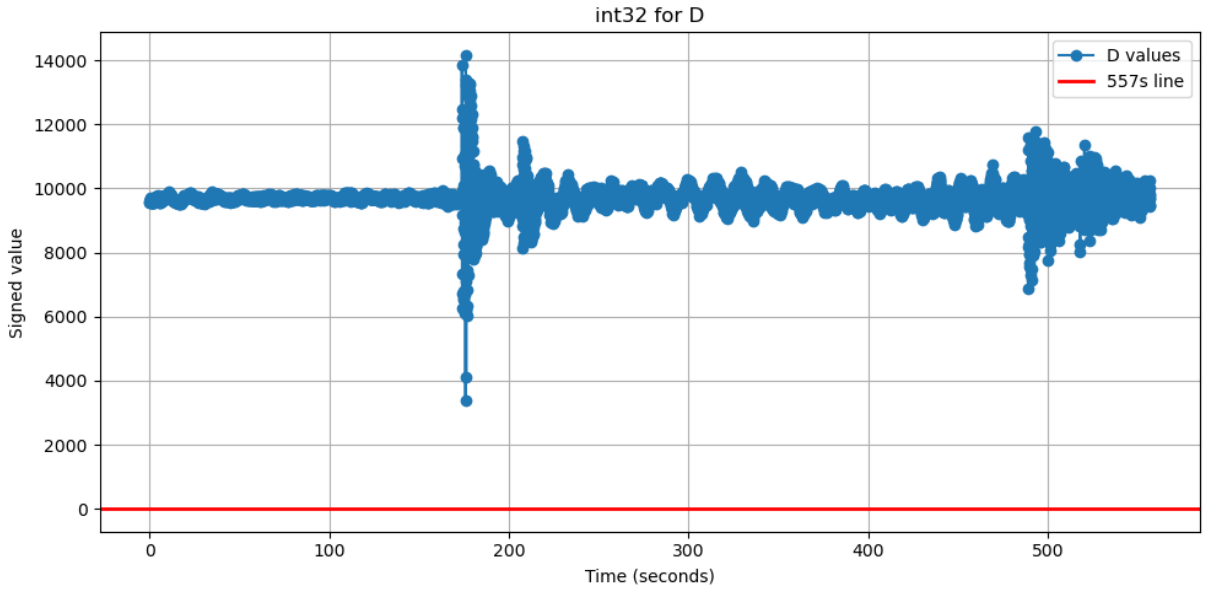

Figure 9: IMU accelerometer data

5.1.3 GNSS & NMEA Pattern Recognition

In this approach, we were able to identify embedded GNSS data within the proprietary binary files generated by the X200 GO sensor. Specifically, recognizable NMEA sentence patterns, such as $GNGGA and $GNRMC [4], were found within the binary stream. These standardized ASCII based GNSS messages confirmed that the files contain interleaved textual navigation data alongside proprietary binary payloads. The figure below shows the repeating patterns.

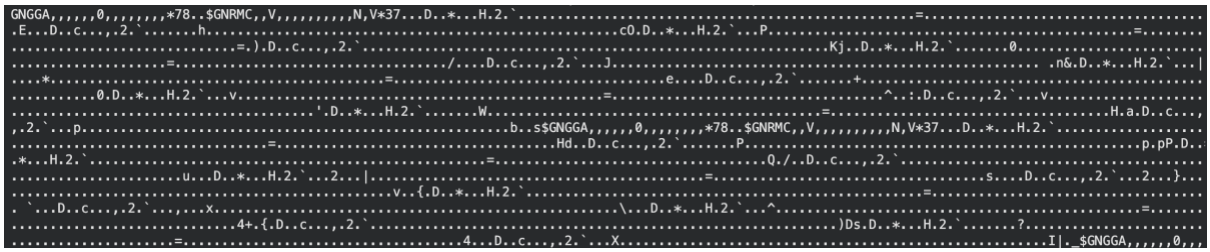

Figure 10: Repeated $GNGGA and $GNRMC pattern

After identifying these NMEA patterns, the GNSS messages were extracted and decoded to obtain timing, positioning, and fix quality information. The decoded values were then plotted and. This visualization confirmed that the GNSS data was recorded at a consistent sampling frequency and exhibited realistic behavior.

	timestamp	lat_i	lon_i	alt_i	v1	v2	v3	quality
0	228492260264288	55065	1179649	1	0.0	0.0	0.0	0
1	228492260264288	55066	1179648	1	0.0	0.0	0.0	0
2	228707008629088	55115	1179649	1	0.0	0.0	0.0	0
3	228707008629088	55116	1179649	1	0.0	0.0	0.0	0
4	228921756993888	55165	1179650	1	0.0	0.0	0.0	0
5	228921756993888	55166	1179649	1	0.0	0.0	0.0	0
6	229136505358688	55215	1179650	1	0.0	0.0	0.0	0
7	229136505358688	55216	1179649	1	0.0	0.0	0.0	0
8	229351253723488	55265	1179650	1	0.0	0.0	0.0	0
9	229351253723488	55266	1179649	1	0.0	0.0	0.0	0

	timestamp	lat_i	lon_i	alt_i	v1	v2	v3	quality	lat_deg	lon_deg	alt_m
0	228492260264288	55065	1179649	1	0.0	0.0	0.0	0	0.005507	0.117965	1.0
1	228492260264288	55066	1179648	1	0.0	0.0	0.0	0	0.005507	0.117965	1.0
2	228707008629088	55115	1179649	1	0.0	0.0	0.0	0	0.005511	0.117965	1.0
3	228707008629088	55116	1179649	1	0.0	0.0	0.0	0	0.005512	0.117965	1.0
4	228921756993888	55165	1179650	1	0.0	0.0	0.0	0	0.005516	0.117965	1.0
5	228921756993888	55166	1179649	1	0.0	0.0	0.0	0	0.005517	0.117965	1.0
6	229136505358688	55215	1179650	1	0.0	0.0	0.0	0	0.005522	0.117965	1.0
7	229136505358688	55216	1179649	1	0.0	0.0	0.0	0	0.005522	0.117965	1.0
8	229351253723488	55265	1179650	1	0.0	0.0	0.0	0	0.005527	0.117965	1.0
9	229351253723488	55266	1179649	1	0.0	0.0	0.0	0	0.005527	0.117965	1.0

Figure 11: Decoded timestamps, latitude and longitude

As part of this analysis, timestamps, latitude, and longitude values were extracted from the decoded GNSS data. Multiple timestamp formats were considered during interpretation, including Unix time and other commonly used GNSS related time representations. However, the decoded timestamp values were found to be unrealistic when compared to the known data acquisition period, as the original dataset was recorded in 2025, while the interpreted timestamps did not correspond to this timeframe. This suggests that the timestamp field may use a different reference epoch, scaling factor, or encoding scheme than initially assumed.

Similarly, decoded latitude and longitude values were plotted to assess spatial plausibility. Although the values exhibited continuity and structural characteristics consistent with positional data, their absolute locations were significantly offset from the known data collection site. The dataset was recorded in Oulu, Finland, yet the plotted coordinates appeared in regions corresponding to southern Africa *(Fig: 12)*.

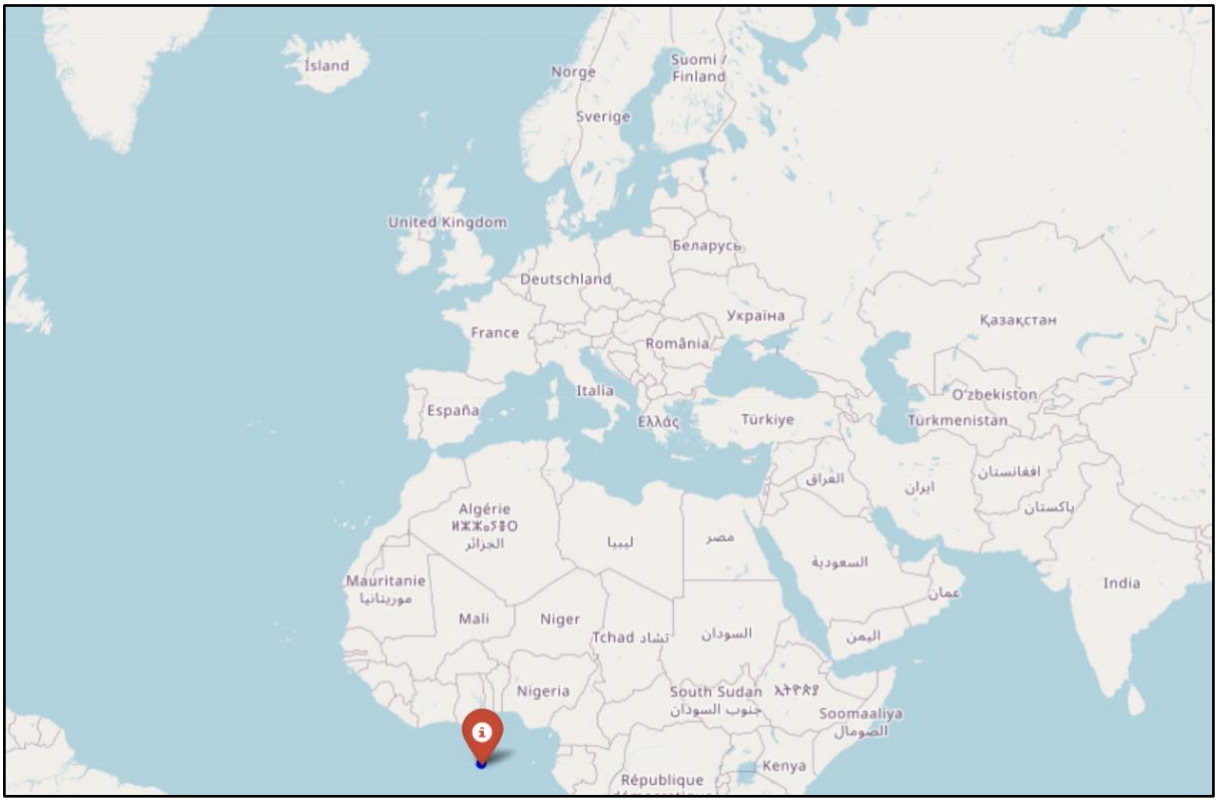

Figure 12: Decoded data when mapped.

The GNSS and NMEA pattern recognition approach confirmed that the X200 GO navigation files contain embedded standardized GNSS messages, specifically $GNGGA and $GNRMC. While timestamps, latitude, and longitude fields were identifiable and structurally consistent, their decoded values could not be directly mapped to real world time or geographic location. The presence of standardized NMEA messages provides a reliable reference for synchronizing and validating binary navigation records, although full geodetic interpretation requires further investigation.

5.1.4 Visualisation Based Validation

During the initial stages of the project, an exploratory decoding approach was implemented based on direct visualization of decoded binary values. In this approach, the binary data was segmented into fixed size records and interpreted using various candidate data types and byte offsets. The resulting values were plotted as time series signals to identify patterns that might resemble sensor outputs, such as accelerometers or gyroscope measurements.

At this stage, it was assumed that meaningful sensor behavior could be detected through visual inspection of individual value streams, including characteristics such as periodic motion, noise patterns, or saturation effects. This approach was supported by visualization scripts that generated plots for multiple decoding configurations, enabling rapid comparison across different offsets and data type interpretations.

Although this method produced several visually structured plots, the results were not conclusive. Many decoded signals exhibited continuity and variation but could not be reliably associated with physical sensor quantities. Subsequent discussion with the customer revealed that this approach was inherently limited, as the X200 GO sensor operates within a three-dimensional spatial context, where meaningful interpretation depends on multi-axis correlation, coordinate frame alignment, and sensor fusion rather than isolated one-dimensional signals.

Consequently, interpreting individual decoded values without accounting for their spatial relationships and orientation in 3D space proved insufficient. The approach was therefore deemed unsuitable for further decoding of IMU and motion-related data, and the project's focus shifted toward structure-driven decoding, GNSS anchoring, and frame-level analysis.

5.1.5 Reverse Engineering

5.1.5.1 Tupni

One approach for automatic input format inference is Tupni, which uses dynamic analysis and symbolic execution to reconstruct file structures from program behavior [2]. It uses techniques such as symbolic execution, dynamic analysis, and differential fuzzing to infer the structure, constraints, and semantics of the underlying format. The result is a formal specification that can be used to build parsers or generate valid test cases. Tupni learns how the program works that handles the binary files and doesn’t take the actual binary file as input.

5.1.5.2 Binwalk

Binwalk is a popular open source tool for analyzing and reverse engineering binary files [1]. It is primarily used to identify embedded file signatures, compressed data, and firmware structures within binary blobs. Binwalk can automatically scan a file for known patterns, extract data sections, and provide insights into the file’s internal structure. It is commonly

used in embedded systems of security research, firmware analysis, and binary forensics. Binwalk attempted to analyze the IMU binary data to detect any recognizable structures, headers, or embedded sections. However, in this case, the Python environment could not import binwalk.core, preventing direct programmatic use of Binwalk. As a result, alternative approaches were pursued to parse and visualize the binary IMU frames manually.

5.2 Discovery of White Label Application

During the research phase of the project, an important additional finding emerged outside the original project scope. While surveying existing tools and software related to SLAM systems, the team identified an alternative application capable of decoding and visualizing data generated by the X200 GO sensor [5].

Further investigation revealed that this application is a white label variant of an existing proprietary solution, introduced under a different name and supplemented with additional features and interface modifications. Although the core functionality appeared to be derived from the same underlying technology as the manufacturer's official software, the existence of this white label version was not known to the customer prior to this project.

From a research perspective, the white label application [5] served as a reference point and feasibility indicator, reinforcing the assumption that vendor independent access to the data is achievable [5]. The discovery added tangible value for the customer by increasing awareness of alternative solutions and highlighting the broader ecosystem surrounding the X200 GO platform.

It is important to note that the existence of a white-label solution does not diminish the need for independent and open parsers. On the contrary, it emphasizes the importance of transparent, well documented decoding tools that enable reproducibility, long term maintainability, and research oriented access to sensor data.

5.3 Results

Despite the complexity and proprietary nature of the data formats, the project achieved several meaningful results:

- Identification of consistent binary header structures and metadata regions.
- Confirmation of fixed length frame structures for multiple file types.
- Successful partial decoding of GNSS data, including:
    - Timestamps
    - Position related fields
    - Velocity components
    - Quality indicators
- Validation of GNSS update rates and data continuity through visualization.
- Successful parsing and visualization of raster/image data using standard geospatial libraries.

While full decoding of IMU and LiDAR payloads was not achieved, several candidate byte regions and structural patterns were identified, providing a clear direction for future work.

5.4 Deployment

The development was primarily in Python, leveraging its strong support for binary data handling, numerical computation, and visualization. The project utilized two main execution environments:

- Google Colab / Jupyter Notebooks: Used for exploratory analysis, rapid hypothesis testing, and visualization-based validation.
- Standalone Python Scripts: Used for structured decoding, reproducibility and batch processing of binary files.

The modular design of the scripts allows future extensions to additional file types and integration into larger processing pipelines. No external deployment environment was required beyond a standard Python runtime.

5.5 Challenges

Several significant challenges were encountered during the project:

5.5.1 Lack of Documentation

No official documentation or reference implementations were available for the X200 GO binary formats, making the decoding process inherently uncertain and experimental.

5.5.2 Proprietary and Potentially Obfuscated Data

Some data fields appeared intentionally encoded in non-obvious ways, complicating direct interpretation and increasing the risk of false assumptions.

5.5.3 Validation Without Ground Truth

Without access to raw reference values or internal sensor specifications, validation relied heavily on indirect methods such as visualization and numerical plausibility.

5.5.4 High Computational Complexity

Brute force decoding across multiple data types; offsets and endianness combinations required careful management to remain computationally feasible.

6. Realised requirements

The requirements of this project were defined with a clear distinction between primary research objectives and secondary implementation objectives. The primary objective of the project was the reverse engineering and decoding of proprietary binary data generated by the X200 GO SLAM sensor. Development of a user interface for data visualization and interaction was defined as a secondary objective, intended to be pursued only if sufficient progress was achieved in decoding the underlying data formats.

6.1 Binary Data Decoding

6.1.1 *Requirement*

Develop binary file parsers capable of decoding and interpreting proprietary data generated by the X200 GO SLAM sensor.

6.1.2 Realisation

This requirement was partially but meaningfully realized. The project successfully:

- Analyze binary file structures and headers.
- Identify frame boundaries and recurring patterns.
- Implemented brute force decoding strategies.
- Achieved partial decoding of GNSS/navigation data.

Although full decoding of all sensor modalities was not completed, the achieved results demonstrate the feasibility of vendor independent access to X200 GO data and fulfill the primary research objective of the project.

6.2 Development of Experimental User Interface

6.2.1 *Requirement*

Develop a user interface to visualize decoded data and provide basic interaction with the parsed sensor outputs.

6.2.2 Realisation

This requirement was not realized during the project.

The development of a user interface was defined as a secondary objective, dependent on sufficient progress being achieved in the primary objective of decoding the proprietary binary data formats. Due to the research oriented and exploratory nature of the project, a significant portion of the available effort was required for reverse engineering tasks, including binary inspection, brute force decoding, and validation of sensor data.

As the decoding process involved high uncertainty and required iterative experimentation, prioritizing user interface development would have reduced the effectiveness of the primary research objectives. Consequently, the project team intentionally focused on understanding data structures and validating decoding feasibility rather than implementing a standalone graphical interface.

6.3 Documentation & Knowledge Transfer

6.3.1 *Requirement*

Document all approaches, findings, tools, and limitations to ensure reproducibility and future development.

6.3.2 Realisation

This requirement was fully realized *[6]*White Label Application. Comprehensive documentation was produced covering:

- Reverse engineering methodology.
- Experimental approaches.
- Achieved results.
- Discovered limitations and open challenges.

7. Used Software Development Method

The software development method used in this project can be best described as a research oriented, iterative, and exploratory development approach, rather than a traditional linear software engineering model. This choice was driven by the nature of the problem domain; reverse engineering undocumented and proprietary binary file formats, for which no predefined specifications, reference implementations or guaranteed solution paths were available.

At the outset of the project, it was not possible to define detailed functional requirements or a fixed implementation plan. Instead, development progressed through iterative experimentation, where hypotheses about data structures and encoding schemes were continuously formulated, tested, and evaluated. This approach aligns closely with experimental software engineering and research based prototyping practices.

Python was selected as the primary development language due to its strong ecosystem for binary data handling, numerical computation, and visualization. Jupyter and Google Colab notebooks were used extensively during the exploratory phases to enable rapid experimentation and interactive analysis, while standalone Python scripts were used to consolidate stable findings and improve reproducibility.

While elements of traditional methodologies such as Agile can be identified particularly in terms of incremental progress and continuous feedback, the overall approach is more accurately characterized as research driven iterative development, where learning and discovery were the primary drivers of progress rather than feature completion.

8. Software Acceptance

In this project, software acceptance was judged by how well the research goals were met, rather than by traditional user acceptance testing. The main aim was to see whether undocumented binary data formats from the Stonex X200 GO SLAM sensor could be reverse engineered, so the “acceptance tests” focused on analytical correctness, consistency of decoded data, and how reliably the same results could be reproduced.

Each decoding idea was checked against what could actually be seen in the data. The binary parsing logic was tested by confirming that file headers, frame boundaries, and recurring patterns were detected correctly across different datasets. For GNSS-related information, the decoded values were compared against known properties of NMEA messages, including their structure, field order, update rate, and continuity over time.

The implementation itself was evaluated through repeated experimentation and visual inspection. Decoded values were plotted to see if they behaved in a physically reasonable way, such as changing smoothly over time, showing realistic sampling rates, and maintaining consistent relationships between fields. Whenever results looked implausible or inconsistent, the underlying assumptions were refined or rejected so that accepted outputs were always grounded in evidence rather than chance.

Several tools supported this acceptance process [6]. Python served as the main language, offering robust libraries for binary data handling, numerical work, and visualization. Jupyter and Google Colab notebooks were used for rapid, interactive exploration, while standalone Python scripts confirmed that stable decoding approaches were reproducible. Visualization libraries helped validate GNSS trajectories and data continuity, and hex viewers were used to verify low-level interpretations of the binary files.

Even though a polished end-user application was not produced, the acceptance criteria were met by showing clear progress toward vendor-independent decoding, achieving partial but consistent GNSS data extraction, and delivering a documented methodology that others can reproduce and build on

9. Project Timeline

The project was planned and executed over the period from September to December 2025. Due to the exploratory nature of reverse engineering undocumented binary data formats, the project planning emphasized iterative progress, continuous validation and regular communication with the customer rather than strict feature based milestones.

The timeline was guided by an initial Gantt chart, which divided the project into research, development, documentation, and testing activities. These phases were not strictly sequential; instead, several activities overlapped to allow findings from experimentation to directly inform subsequent analysis and documentation.

Milestone description	Category	Assigned to	Start	Days
Study & Research				
Project Understanding	On Track	All	9/25/2025	10
Feasibility Study	Milestone	All	10/5/2025	15
Research	Low Risk	All	10/20/2025	10
Development				
Parsing	High Risk	All	10/25/2025	25
Validation	Med Risk	Sami & Mahim	11/20/2025	5
Interface Develoment	Low Risk	Huzaifa	11/15/2025	5
Integration	Milestone	Huzaifa	11/20/2025	5
Documentation				
Research	On Track		10/20/2025	10
System Diagrams	On Track	Huzaifa	11/12/2025	3
Testing				
Integration Testing	Low Risk		11/25/2025	2
Usability Testing	Low Risk		11/28/2025	3
Issues/Bugs Fixes	Low Risk		11/25/2025	5

Figure 13: Gantt chart

The project began with familiarization phase, during which the team studied the X200 GO sensor, its role in SLAM applications and the overall decoding challenge. This was followed by a study and research phase involving a literature review on binary reverse engineering, investigation of similar SLAM systems and assessment of tools for low-level binary analysis. Initial inspection of the X200 GO data files was performed to identify headers, metadata and recurring structural patterns. The main phase of the project consisted of development and experimentation, focusing on exploratory binary analysis and brute force decoding. Multiple hypotheses regarding frame structure, data types and byte alignment were tested, with visualization-based validation.

Documentation activities were conducted in parallel with development throughout the project. Experimental findings, assumptions, and limitations were continuously recorded to ensure reproducibility and clarity, even when certain decoding attempts did not yield conclusive results. This parallel documentation approach was essential in a research driven project where learning outcomes are as important as final implementations.

A testing and consolidation phase was planned toward the end of the project timeline. Given the absence of a finalized user interface and the experimental nature of the decoding tools, testing focused primarily on validation of decoding logic and consistency checks rather than

traditional user facing testing.

9.1 Communication & Progress Monitoring

Project planning also included regular communication with customers. Weekly meetings were scheduled as the primary mechanism for progress reporting, discussion of findings, and alignment of expectations. During these meetings, the team presented ongoing results, demonstrated experimental tools and visualizations, and discussed challenges encountered during the decoding process. It should be noted that due to scheduling constraints, not all weeks included a formal meeting.

10. Feedback

This course offered a rare chance to tackle a problem that felt very close to real research and engineering work. One of its biggest strengths was the freedom to explore an open-ended task without strict, predefined specifications. That freedom encouraged independent thinking, experimentation, and critical questioning of assumptions, which are key skills in research-focused and advanced technical roles.

The project deepened our understanding of reverse engineering methods, binary data analysis, and sensor-based systems like SLAM. It showed how to make progress when documentation is missing, how to validate results without a clear ground truth, and how to structure exploratory work so that it remains systematic and reproducible. Being required to document failed attempts and limitations was especially helpful, because it highlighted learning and reasoning rather than only the final outcome.

Interaction with the customer was another major positive. Regular discussions helped clarify expectations, confirm intermediate results, and keep the work focused despite the uncertainty of the problem. This mirrored real-world research collaboration and was an important part of the learning experience.

One area for improvement would be to offer slightly clearer guidance on how research-oriented projects are evaluated, especially when a full technical implementation is not realistic. Providing a few examples of past exploratory or reverse-engineering projects could also help students better judge the expected scope and depth of their work at the start.

Overall, the course was demanding but very rewarding. It built a strong bridge between theory and practical research skills and fostered a mindset cantered on exploration, evidence-based reasoning, and technical curiosity.

11. References

[1] D. Loss, “Binary Parsing Tools,” [Online]. Available: https://github.com/dloss/binary-parsing.

[2] F. AG, “Fixposition Documentation.,” [Online]. Available: https://docs.fixposition.com/.

[3] “. S. L. S. S. T. Stonex. [Online]. Available: https://stonex.it/product/x200go-slam-laser-scanner.

[4] N. M. E. Association, “NMEA 0183 Standard.,” [Online]. Available: https://www.nmea.org.

[5] H. Y. Z. L. a. D. S. J. Caballero, “Polyglot: Automatic Extraction of Protocol Message Format Using Dynamic Binary Analysis,” *Proc. 14th ACM Conf. Computer and Communications Security,* 2009.

[6] Binwalk, “”Firmware Analysis Tool.” [Online].,” [Online]. Available: https://github.com/ReFirmLabs/binwalk.

[7] “White Label SLAM Application.,” [Online]. Available: http://www.slam100.com/slam-user-guide/ .

[8] S. A. A. M. a. M. H. S. Häkkilä, “Project Source Code Repository,” [Online]. Available: https://github.com/huzaiifaaaaa/x200-go