



BACHELOR'S THESIS

GROUP 8

Data recording of Excavator CAN bus for Hydrogen Operation

SUBMITTED BY

<i>Name</i>	<i>Student No</i>
Abdirahman Ali Abdulle	233831
Petter Wang Tveit	233813
Mahram Hussein Safdari	222988
Salem Rezaie	235114
Markus Gevelt Danielsen	223635
Fouad Ayman Foad Irkayek	212329

May, 2022

Abstract

This thesis will address the development of a system based on a description delivered by clients IoT-labs and Applied Hydrogen. The system implemented is a data collection unit fitted with multiple components for data collection, as well as a webcam for surveillance purposes. The data collection unit will in due course be mounted in an excavator to collect data in order to provide helpful information to any other parties involved.

i) Contents

Abstract	1
i) Contents	2
ii) List of Figures	5
iii) List of Tables	7
iiii) List of Acronyms	7
1 Introduction	8
1.1 Background	8
1.2 Changes to Project	8
1.3 Project Assignment	9
1.4 Scope of Work	9
2 Project Management and Development Process	10
2.1 Project Model	10
2.1.1 Jira	11
2.1.2 User Stories	11
2.1.3 Sprints	12
2.1.4 Product Backlog	13
2.1.5 Sprint Planning	14
2.1.6 Daily Stand-up Meeting	15
2.1.7 Sprint Review	15
2.1.8 Task Board	16
2.2 Project Tools	17
2.2.1 Communication Tools	17
2.2.2 Work Tools	17
2.3 Requirement Specification	18
2.4 Stakeholder Analysis	18
2.5 Test Plan	20
2.6 Risk Assessment	21

2.6.1	Risk Management	21
2.6.2	Identifying Risks	22
2.6.3	Risk Analysis	23
3	Proposed System	24
3.1	Understanding the System	24
3.2	Hardware	26
3.2.1	Inertial Measurement Unit	26
3.2.2	GPS Module	28
3.2.3	Raspberry Pi	30
3.2.4	Arduino-based CAN bus card	30
3.3	Software	31
3.3.1	Use Case	31
3.3.2	Sequence Diagrams	32
3.3.2.1	Storing Data Sequence Diagram	33
3.3.2.2	Display Data Sequence Diagram	33
3.3.3	Software Architectural Model	34
4	Implementation	35
4.1	Physical Design	35
4.1.1	Prototyping	35
4.1.1.1	Version 1	35
4.1.1.2	Version 1.1	37
4.1.1.3	Version 1.2	38
4.1.1.4	Version 1.3	38
4.1.2	Final Design	38
4.1.3	Production Method	40
4.2	Software Design	42
4.2.1	Choosing Programming Language	42
4.2.2	CAN-messages	42
4.2.2.1	Arduino Communication Problem	42
4.2.2.2	Physical Encoding of CAN duty and Troubleshooting	43
4.2.2.3	Collecting CAN-messages	43
4.2.2.4	CANopen	44

4.2.2.5	Data Frames	44
4.2.2.6	CanSniffer code	44
4.2.2.7	Serial Communication	45
4.2.3	Sensor Data Collection	45
4.2.3.1	IMU	45
4.2.3.2	GPS	46
4.2.4	Synchronizing Data	46
4.2.4.1	Run on Boot	46
4.2.4.2	Real-time Clock	47
4.2.5	Local Storage	48
4.2.6	Cloud Storage	49
4.2.6.1	Streaming Various Data	49
4.2.6.2	Real-time Data Processing	49
4.2.7	Google Cloud Platform	50
4.2.7.1	Google Cloud Console	50
4.2.7.2	APIs	51
4.2.7.3	Cloud Iot Core Service	52
4.2.7.4	GCP Iot Core and Raspberry Pi	53
4.2.8	Remote Desktop Connection	54
4.2.9	Surveillance	55
5	Review and Future Work	56
5.1	Hardware	56
5.2	Software	57
6	Conclusion	59
	References	65
	Appendices	66

ii) List of Figures

1	Scrum model	10
2	User story mapping	12
3	Sprint overview	13
4	Product backlog created in Jira	14
5	Task board	16
6	Stakeholders top-level	19
7	Stakeholders low-level	20
8	Risk process model	21
9	Risk matrix	23
10	System context	24
11	Overview of the proposed system	25
12	SEN0373 Inertial Measurement Unit	28
13	Adafruit Ultimate GPS	30
14	Arduino-based CAN bus card from Copperhill Technologies	31
15	Use case diagram	32
16	Data storing sequence diagram	33
17	Displaying data sequence diagram	34
18	Software architectural model	35
19	Small junction box	36
20	Floor layout version 1	36
21	External box for IMU	37
22	Big junction box	39
23	Floor layout for final system	39
24	3D-model of final system	41
25	Laser cut plate	41
26	CAN bus resistors	42
27	I2C communication	45
28	UART communication	46
29	Boot flow	47
30	RTC-battery compatible GPS	48
31	Class diagram - GPS data to text file	48
32	Class diagram - IMU data to text file	49

33	Raspberry Pi data handling architecture	50
34	Cloud storage architecture	51
35	Enabled application program interfaces	52
36	GCP IoT Core and Raspberry Pi setup	53
37	Cloud dataflow job template	54
38	Remote desktop connection	55
39	UI sketch	58
40	System architecture	68
41	Flowchart: Serial communication	69
42	Flowchart: IMU	70
43	Flowchart: GPS	71
44	Protocols and procedures of the OSI model. The CAN protocol defines the OSI model's bottom two levels.	106
45	Typical CAN message	106
46	Flowchart of an Arduino UNO to an Arduino Due	108
47	Schematic of Arduino Uno to Arduino Due wiring	109
48	Schematic of an Arduino UNO1 to an Arduino UNO2	109
49	Output of virtual sender and receiver	110
50	Flowchart of an Arduino UNO1 to an Arduino UNO2	111
51	Schematic of two Arduino UNOs and an Arduino Due	112
52	Flowchart of an Arduino UNO1 and an Arduino UNO2 to an Arduino Due .	113
53	Flowchart: CanSniffer	114
54	Schematic of potmeter and SD-card connection	115
55	Flow chart of storing IMU data to SD-card using Arduino Uno	116
56	IMU and SD card schematic	117
57	CanSniffer application	119
58	Decoded messages	120
59	Wiring diagram	121
60	IMU axis	122
61	Power consumption	124
62	SD-card size calculation	125
63	Requirements	133
64	Comparison of IMU [42, 43, 44, 45, 46, 47, 48, 49, 50, 51]	134
65	Comparison of GPS [52, 53, 54, 55, 56, 57]	135

66	Gantt diagram	136
67	System design	137
68	Alternative mounting of system with sliders for easier access to components	139

iii) List of Tables

1	SMART methods	18
2	PUGH matrix - IMU	27
3	PUGH matrix - GPS	29

iiii) List of Acronyms

- CAN** Controller Area Network
- IMU** Inertial Measurement Unit
- SAE** Society of Automotive Engineers
- JSON** Java Script Object Notation
- MQTT** Message Queuing Telemetry Transport
- IoT** Internet of Things
- SSH** Secure Shell
- VNC** Virtual Network Computing
- API** Application Program Interface
- RTC** Real-time clock
- GCP** Google Cloud Platform
- UI** User Interface

1 Introduction

Data recording of **Excavator CAN bus for Hydrogen Operation (D.E.C.H.O)** is a collaborative project between IoT-labs, Applied Hydrogen and the University of South-Eastern Norway (USN). The project was carried out by a team of undergraduate students as part of their bachelor's thesis.

1.1 Background

Construction is today responsible for 23% of all air pollution, and heavy machinery is responsible for a significant part of that [1]. Applied Hydrogen (AH) has taken aim at wanting to convert construction vehicles to hydrogen operation as part of the green shift within the construction industry. They are currently carrying out this project by implementing this conversion on a Volvo EC300DL excavator, donated by Tveito Maskin AS. To carry out such a conversion the plan is to replace the diesel engine of the excavator with a hydrogen fuel cell combined with a battery in such a way that no further upgrades or replacements are required. Once the conversion is completed, it is in AH's best interest to monitor the excavator's behavior through some sort of data collection system. The data should be collected while the excavator is being operated as normal, and would be used for troubleshooting purposes in the future. The data would also help potential partners with their own endeavours.

1.2 Changes to Project

Throughout the project life cycle there has been a change in the team's task at hand. The original task was concerning the unknown messages collected from CAN bus. It was the team's job to decode these unknown messages to gain information about the contents of the messages, as well as the location they were sent from, and their destination. **CAN bus** is defined as a serial protocol, commonly used within the automotive industry [2]. For more information regarding CAN bus, see Appendix H.

The expired task also included making a simulator that would simulate the excavator's functions, as well as an emulator that would emulate the excavator's engine.

On the 14th of February the team was given a new modified project description. Through meetings between Applied Hydrogen and Volvo, the clients Applied Hydrogen and IoT-labs determined the current project was not feasible. In short, there seems to be essential

information on the J1708 bus of the excavator the team was working on. Since the 1708 bus does not exist in newer excavators, the original plan of decoding was deemed a waste of both time and energy.

Instead of decoding CAN-messages, the modified project description ordered the team to collect them. The new project revolved around designing and implementing a system that can collect both CAN bus and sensor data.

1.3 Project Assignment

The following project assignment is based on information provided by the client IoT-labs through written descriptions of the task at hand.

The team has been given a smaller part of the full-scale project and the team's contribution will be concerning the collection of data from the excavator. The architecture presented in Appendix C is what the integration of a hydrogen extension will ultimately look like. The Integration Control Unit (ICU) as seen in the architecture is a vital part of the project. This unit shall be designed and implemented to work as a data recorder. The recorder will be an integral part of the system and shall include multiple functions explained later in the report. Sensors for data collection will also be implemented in the final system.

1.4 Scope of Work

The aim of this thesis is to develop a CAN bus application system that will allow for both data collection and surveillance of the excavator. The CAN bus application will also generate data to cloud storage. Furthermore, data from the application system can be consumed for troubleshooting purposes. The software will be developed in such a way that it can simply be upgraded and reused for other implementations as well as various types of construction machines.

2 Project Management and Development Process

This section addresses the team's choice of project model and the principles of project management. It also introduces elements of the team's development process in terms of requirements, tests and risks.

2.1 Project Model

After some discussion, the team decided to adopt an agile project management model based on Scrum. The Scrum model is easy to follow, and allows for each member to work independently (as seen in Fig.1). Since it is difficult to figure out how far the team will advance with the project, an agile approach was agreed upon, which allows for both adding and removing requirements if necessary.

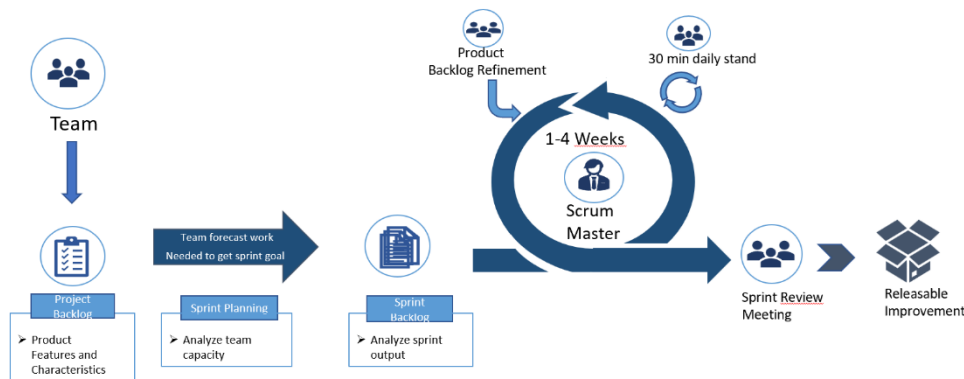


Figure 1: Scrum model

Agile is a development process that emphasizes iterative and incremental development of products and testing that takes into account system requirements and potential solutions throughout the project's life cycle [3]. Agile is most commonly associated with Scrum, and for good reason. The sprints are short, manageable cycles that divide the project into small parts. Shorter cycles make processes more adaptable to changes and allow them to respond more quickly, which results in a greater solution.

In order to maximize the efficiency of Scrum and Agile, the team started using "monday.com" for project management at first, however, it was too expensive and limited in its use in terms of the student package. Therefore, it was replaced by Jira.

2.1.1 Jira

Jira is an advanced project management tool designed for Agile teams to perform essential project planning. As an issue-tracking tool for all types of testing, it is widely used in all facets of software development, but one can also create project management workflows that work for different types of businesses. A great aspect of Jira is that it is free for small teams of up to ten users [4].

Jira has the advantage of explicitly supporting the management of the aforementioned artifacts, such as a backlog, sprints, and roadmap that will be utilized with the current project model. Jira will ultimately give the team the ability to develop an agile process that can be broken down into stages such as functionality, planning, implementation, testing, and review [4].

2.1.2 User Stories

User stories are critical Scrum artifacts that are used to define high-level functional requirements. They are brief, straightforward descriptions from the perspective of the consumer, and in the current instance, the team is following the prevalent practice of: "As a [persona], I [want to], [so that]". The purpose of user stories is to encourage team discussion and offer the team a clearer picture of what the user wants the solution to accomplish[3].

The team utilized "Easy Agile" on Jira to add user stories. The application did not only provide an intuitive, collaborative and visual mode of understanding and defining the user's journey with the product, but it also helped maintain an overview of the whole product development process without it being separated from the rest of the project's activities (as seen in Fig.2).

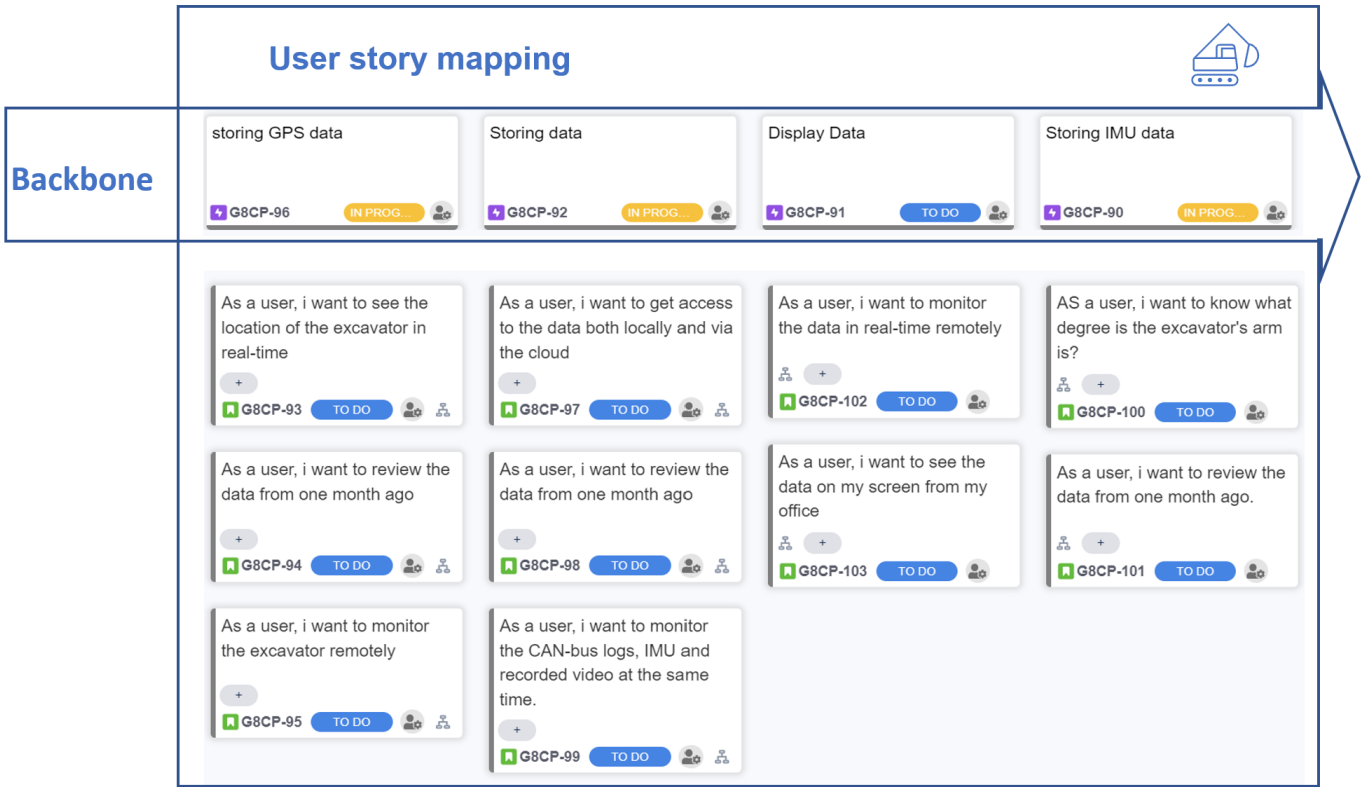


Figure 2: User story mapping

2.1.3 Sprints

When creating a new project, utilizing the Scrum template gives one the ability to collaborate using sprints in order to break down large, complex projects into manageable chunks [5]. A sprint is a condensed period of time during which a Scrum team works to complete a specific amount of work. Scrum and Agile approaches are built around sprints, and getting sprints right can help our team ship a better product with less issues [5]. Fig.3 shows the duration of each sprint, as well as which tasks are completed, which are in progress, and what tests and reviews are waiting to be completed.

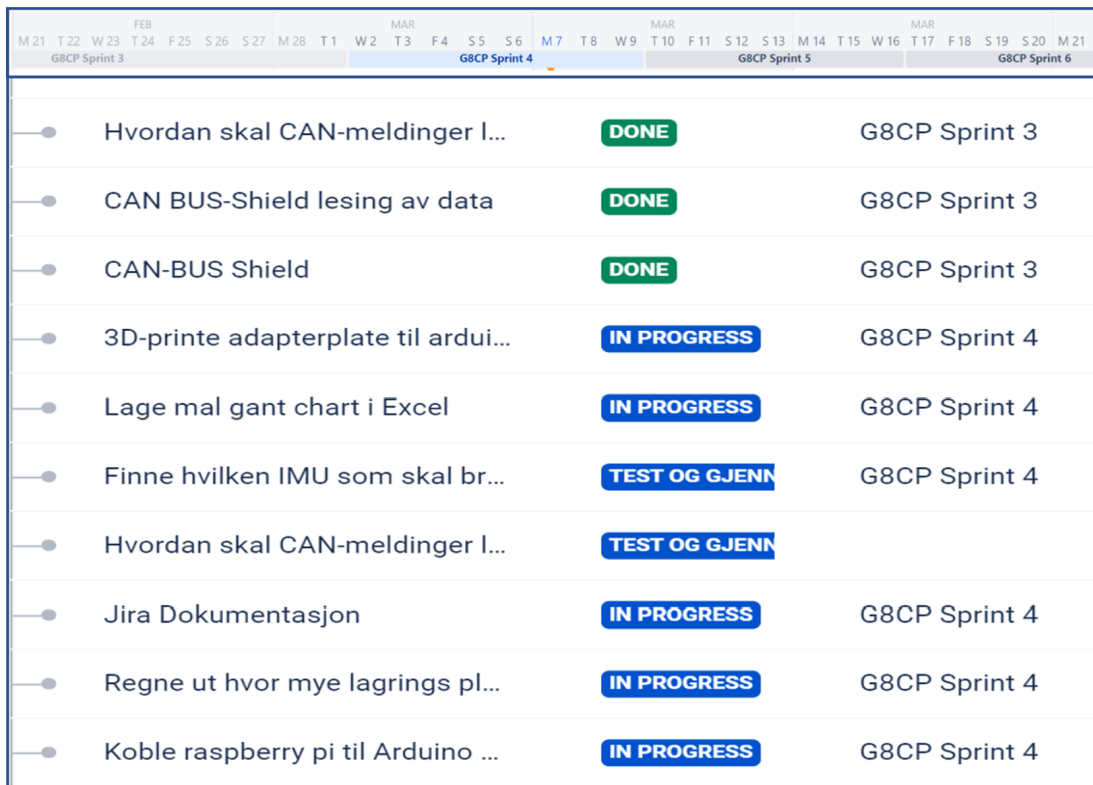


Figure 3: Sprint overview

2.1.4 Product Backlog

The road map and associated requirements are used to create a product backlog, which is a prioritized list of work for the development team. The most critical items are positioned at the top of the product backlog, allowing the team to prioritize what should be delivered first. Fig. 4 shows the product backlog that was created in Jira.

The screenshot shows a Jira Product Backlog for a sprint titled "G8CP Sprint 3" running from February 16 to March 1, with 12 issues. The interface includes a search bar, filters for "SR", "A", "PT", "MB", and "+2", and a "Type" dropdown. The issues are listed with their IDs, titles, and current statuses. The statuses are color-coded: blue for "IN PROGRESS", green for "DONE", and orange for "TEST OG GJENNOMGANG".

Issue ID	Issue Title	Status
G8CP-31	Finne hvilken IMU som skal brukes	IN PROGRESS
G8CP-58	Koble raspberry pi til Arduino Due	IN PROGRESS
G8CP-37	3D-printe adapterplate til arduino	TEST OG GJENNOMGANG
G8CP-33	Programmerings alternativer. Hvilket språk?	DONE
G8CP-49	CAN-BUS Shield	DONE
G8CP-50	Bli master i Jira	DONE
G8CP-55	CAN BUS-Shield lesing av data	DONE
G8CP-32	Hva slags hardware skal brukes til datalagring?	DONE
G8CP-34	Ny oppgavebeskrivelse med forklaring	DONE
G8CP-35	Hvordan skal CAN-meldinger lagres?	DONE
G8CP-59	Finn ut hvordan IP kanal fungerer.	DONE
G8CP-57	Regne ut hvor mye lagrings plass trenger vi for en uke?	IN PROGRESS

Figure 4: Product backlog created in Jira

2.1.5 Sprint Planning

Sprint planning is a Scrum event that starts the sprint. The aim of sprint planning is to figure out what can be achieved in a given sprint and how it will be accomplished. Since the team don't have a product owner, the Scrum team will collaborate on sprint planning. The team will need to agree on the length of the sprint, the sprint goals, and what should be prioritized. The sprint planning session sets the agenda and objectives for the sprint. During sprint planning the Scrum team has to think about:

What - The sprint's aim (or goal) is described by the team, along with which backlog items contribute to that goal. The Scrum team determines what can be accomplished in the course of the sprint and what they will do throughout the sprint to accomplish it.

How - It is the team's responsibility to develop a strategy for achieving the sprint goal. In addition, the sprint plan represents a value and effort negotiation between members in order to come to an agreement on the prioritization of each item.

Who - As previously stated, there is no product owner, thus the team decides on the duration and planning of the sprints as a collective. A sprint can be planned if all or more than four members of the team are present at the same time. For example, the sprint cannot be planned solely by the group leader or Scrum Master. During meetings the team will also decide who will be responsible for each task.

2.1.6 Daily Stand-up Meeting

Every morning the team has a short daily stand-up meeting to give a swift update on what everyone is currently working on. It is not really a detailed status meeting in the traditional sense. The Scrum Master is responsible for keeping meetings on track and making sure that everyone answers these questions briefly and concisely:

- What did I accomplish yesterday?
- What am I going to work on today?
- Is there anything that is preventing progress?

When a member reports what they did yesterday in front of the team, they imply accountability. It will help the team in working harder because no one wants to be the member who does the same thing repeatedly and never sees progress.

2.1.7 Sprint Review

At the end of each sprint, the team has a sprint review meeting where the team's work is demonstrated. During the meeting, the team members gather around a table and discuss the work that has been accomplished in the current iteration. It is a great opportunity for members to ask questions, give feedback, and try out new features. During the meeting the team goes through what went wrong, what went right, and what can be improved upon for the following sprint.

2.1.8 Task Board

Task boards represent the tasks that should be completed during the current sprint. It provides a great overview where one can see how many tasks each group member has, and how long each of these tasks will take.

There are four phases to the road as shown in Fig.5. In the first table starting from the left, one can see the number of tasks that need to be completed. The second table addresses tasks that are currently in progress. The third table presents tasks that are currently in the testing and reviewing phase. If the results of these tests are satisfactory, they will be moved to the fourth and final table, which represents completed tasks.

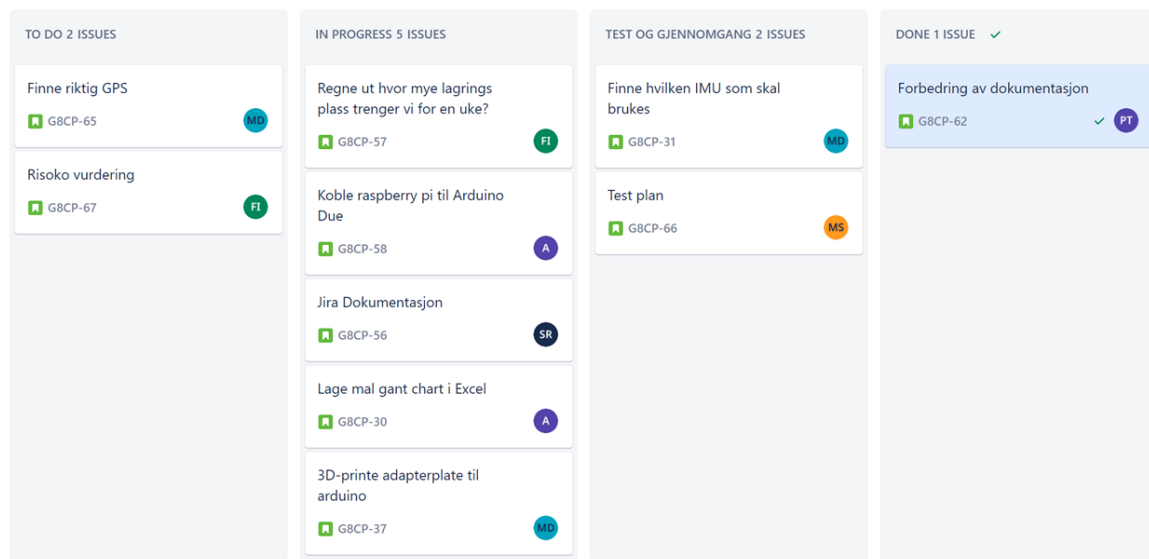


Figure 5: Task board

2.2 Project Tools

2.2.1 Communication Tools

Facebook Messenger

Facebook Messenger was used as the primary tool for communication within the team. It was used to convey information quickly as most team members were checking the application often.

Zoom

Zoom was used mainly for arranging video calls with clients at the start of the project when stricter COVID-19 restrictions were ongoing.

2.2.2 Work Tools

Microsoft Teams

Microsoft Teams was used mainly for storing various documents, powerpoints and Visio-drawings. It was also used occasionally for video calls within the team environment.

Jira

Jira was an essential administrative tool that was used for project management, as well as user story mapping.

Overleaf

The team utilized LaTeX for documenting throughout the project. Overleaf was a cheap option that allowed the team to work simultaneously on the same documents.

Solidworks

Solidworks was used primarily for 3D-modeling throughout the project.

Visio

Visio is a diagramming software that was mainly used to model UML-diagrams displaying the structure of the software, as well as other diagrams to provide more context to the system.

Github

Github was used for software development, in terms of providing backups for the team's code, as well as being used to easily transfer code from one device to another.

2.3 Requirement Specification

To better understand what the product or service should do, it is important to set requirements. The requirements represent what the customer wants and helps develop the product according to the customer's demands. When finding requirements, it is important to keep in mind that these will have to be tested and documented. One way to help do this, is by making the requirements **SMART**, meaning **S**pecific, **M**easurable, **A**chievable, **R**elevant, **T**ime bounded, as shown in Table 1.

Table 1: SMART methods

S	Specific	E.g., state what we need to do.
M	Measurable	E.g., provide a way to evaluate.
A	Achievable	E.g., it must be within our scope.
R	Relevant	E.g., make sense within our job function/similar.
T	Timebound	E.g., state when we will get it done.

Because of the big changes to the project, new requirements were necessary (See Appendix T). These requirements were more specific and attainable, as the project was more centered around creating a finalized product and not a development system as earlier. This would also make testing of the requirements easier, which in turn made the project more manageable, and made it easier reaching the desired goal.

2.4 Stakeholder Analysis

To create a good product or service, it is important to know who has interest in what is being developed. One way of doing this is performing a stakeholder analysis. This can help map who has interest in the project, and what role they play. This bachelor project is a small part of a bigger main project owned and managed by Applied Hydrogen. This means that there are several levels in terms of project management and roles, which can make it difficult to identify the stakeholders, and what part they play. To get a better overview of this, the stakeholders have been divided into two main levels, top-level and low-level.

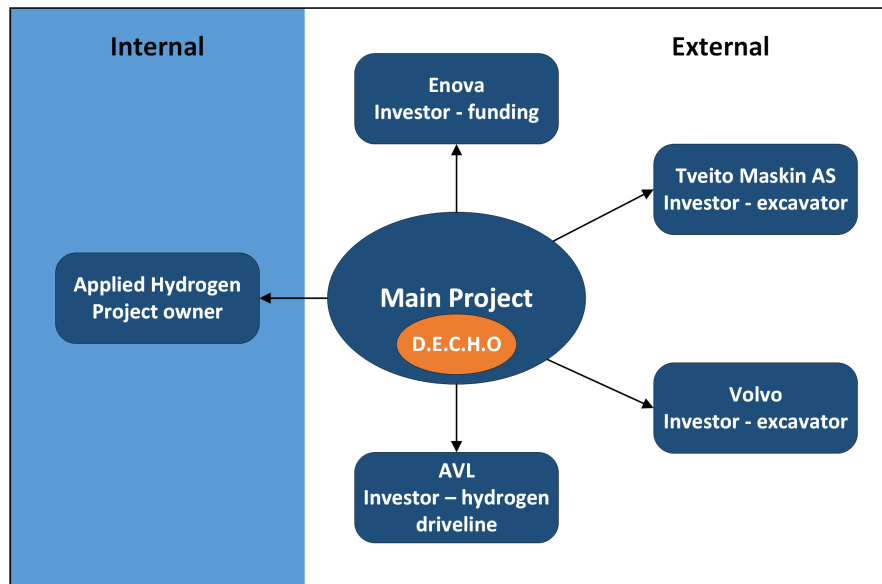


Figure 6: Stakeholders top-level

Top-level (as seen in Fig.6) is in this case, the bigger main project, owned and managed by Applied Hydrogen. They are themselves internal stakeholders, as they are the owners of the project. Externally some of the stakeholders are investors like Enova that helps with funding, and Volvo and Tveito Maskin AS, as they are the ones providing the excavator used. AVL is also an important stakeholder, as they are the ones providing the hydrogen driveline used for the project.

Low-level (as seen in Fig.7) is the bachelor project, as it only makes up a small part of the main project. Here the internal stakeholders are IoT-Labs and D.E.C.H.O. Since IoT-Labs are the ones outsourcing the project, they are the project owners. The bachelor group is categorized as employees and maintainers, as the group are the ones performing the work tasks, and will be the ones maintaining the systems developed. On this level, the external stakeholder is Applied Hydrogen as customer and user. This is because they are the ones the system is mainly created for, and the ones that will make use of the information found and systems created.

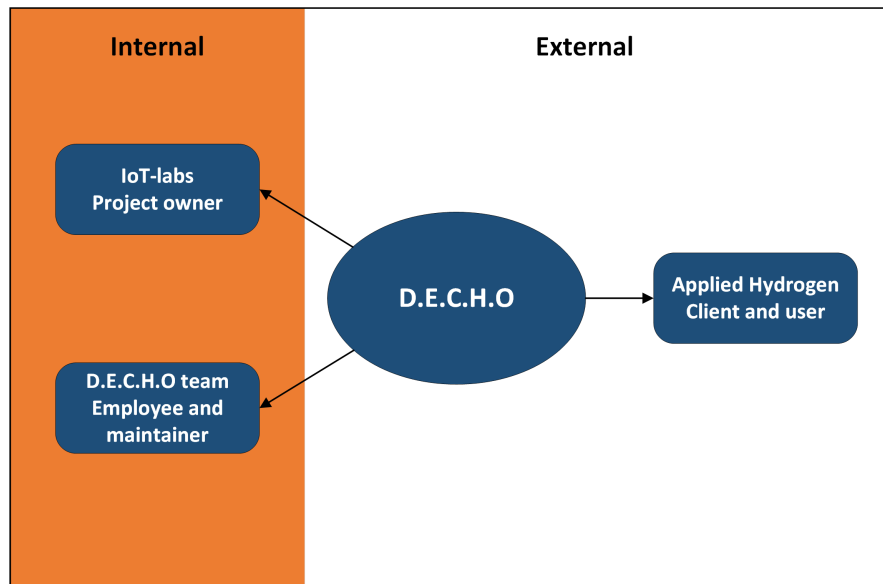


Figure 7: Stakeholders low-level

2.5 Test Plan

During the project life cycle multiple tests were carried out. It was important for the team to perform tests consistently throughout the project to identify and remove potential errors and mitigate flaws in the components or system. These tests would also follow the SMART method as shown in Table 1.

When performing a test the test results would be placed in individual test tables. These tables would include:

- The objective of the test.
- The method used to test.
- The expected result of the test.
- The actual result of the test.
- Whether the test was a success or not.
- The date of the test.

The tests performed were sometimes needed to be revisited. As new changes were made to the system, some tests would eventually become irrelevant and new tests would have to take place in order to maintain the working functions of the system.

2.6 Risk Assessment

2.6.1 Risk Management

Risk management is a tool for any project used to achieve its goals. With good risk management, one can anticipate any risks that will have a negative impact on the project. A structured use of risk management enables the team to see risks, measure the impact if the risks occur, prioritize them and remove unwanted risks. In this project, for each identified risk, the risk management process model was followed. Risk management includes activities to identify and assess the risks, describing them, analyzing their features and implementing a planned response. The team processed risk management as followed: Identified new risks, defined their statement, defined their probability and severity, laid strategy, and documented steps taken, (as seen in Fig.8). An assessment was executed at the start of each sprint. Iterating this process made the team able to assess and attend risks that were amendable at the given stage of the project.



Figure 8: Risk process model

2.6.2 Identifying Risks

Risk identification is an iterative process where the objective is to identify and record all possible risks that may affect the outcome of the project negatively. Once a risk is identified, an analysis must be carried out in order to develop a strategy to reduce its impact [6].

The team sorted risks into three different categories.

Project Risk: The ones associated with the success of the project.

Product Risk: Those associated with how the product works.

Technical Risk: Directly associated with hardware or software.

There were multiple methods that could be used to find and identify the risks that may have had an impact on the project. Thinking aloud was one of the methods the team utilized. It helped with putting together additional ideas and identifying factors that were critical throughout the project period.

The thinking aloud method revolves around spontaneously reporting everything that goes through ones minds whenever performing a task, not attempting to analyze anything [7].

2.6.3 Risk Analysis

All risks identified must be analyzed and evaluated to get a better understanding of their possible impact on the project. By doing this, one could prioritize those that needed immediate attention and those who didn't. Risk analysis identifies the possible causes of a risk occurring, and its potential impact on the project. Risk evaluation helps with defining the severity of the consequence. This is determined qualitatively through a risk matrix, as seen in Fig.9. The severity is calculated by giving the likelihood and the consequence a rank and then multiply them together [8].

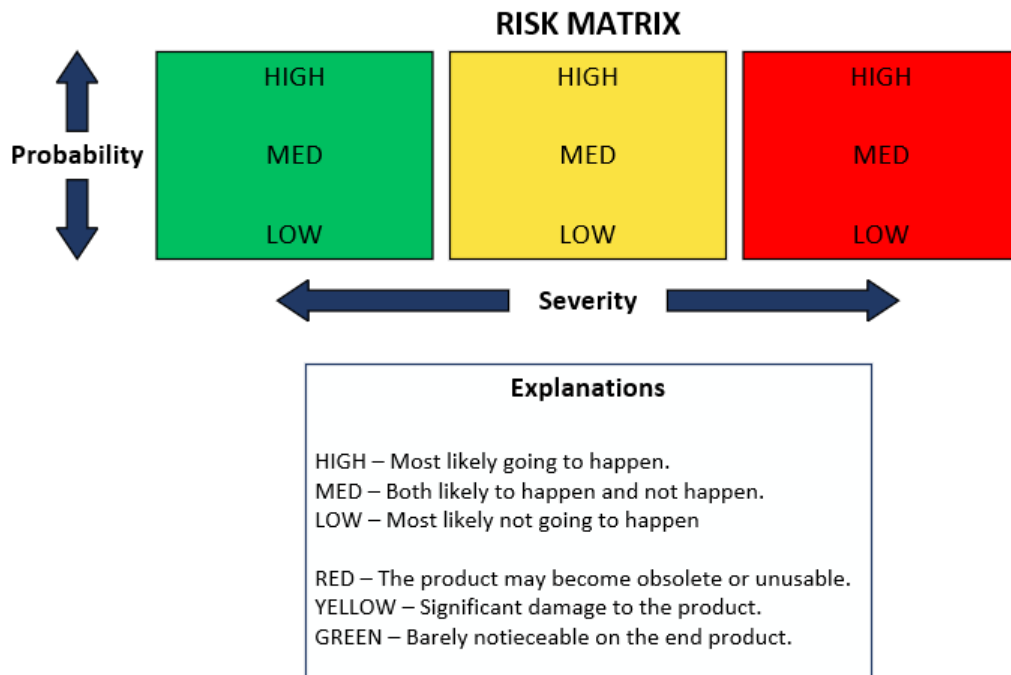


Figure 9: Risk matrix

3 Proposed System

This section presents the team's proposal of a solution, as well as introducing both the hardware and software options that were opted for.

3.1 Understanding the System

To gain greater understanding of the system the team decided on creating a system context diagram to visualize the internal and external factors that must be considered for the system. A system context diagram will often help work out the relationships without going into detail [9]. The proposed system that was used in later implementation is presented in Fig.10.

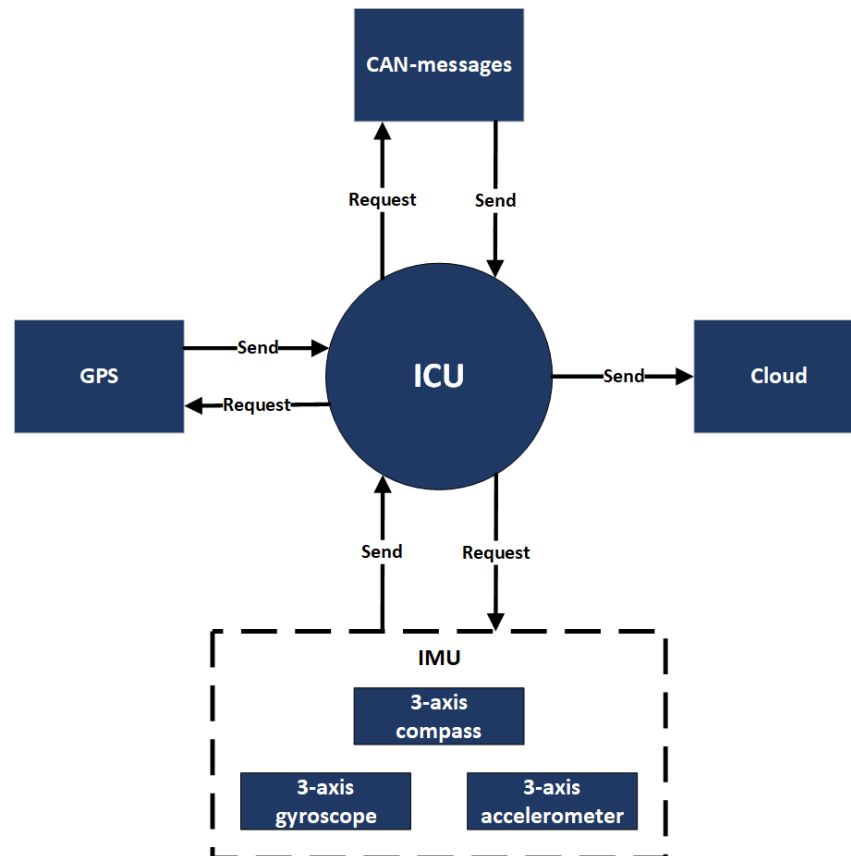


Figure 10: System context

The system consists of the Integral Control Unit (ICU) that will be mounted inside an enclosure, directly behind the cabin of the excavator, accompanying the Engine Electronic Control Unit (E-ECU). Based on the placement of the ICU, it is able to simultaneously collect CAN bus, Inertial Measurement Unit (IMU) and GPS data. The choice of hardware (as seen in Fig.11) and the reasoning behind it will be discussed in the following section.

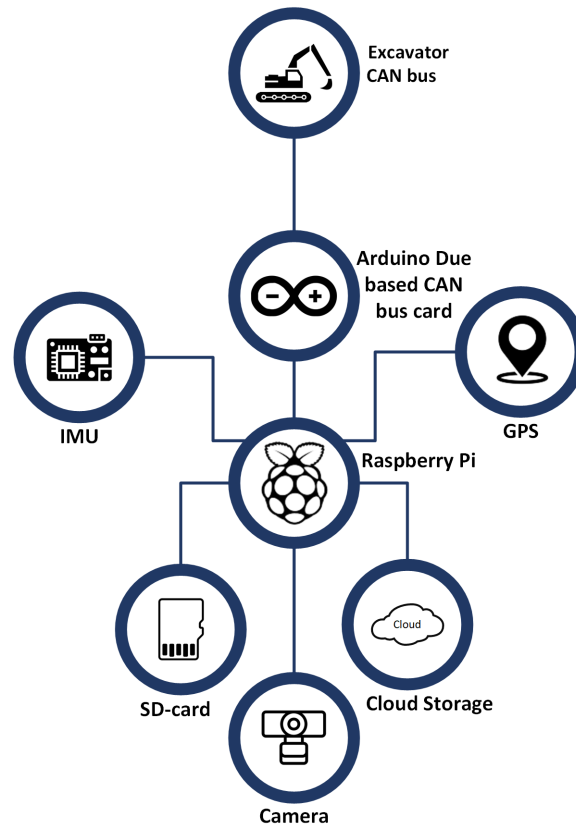


Figure 11: Overview of the proposed system

3.2 Hardware

3.2.1 Inertial Measurement Unit

An Inertial Measurement Unit (IMU) is an electronic device designed to measure the forces, angular rate, and orientation of a given object. This is achieved by combining data from an accelerometer, gyroscope, and magnetometer. Each of these often measure in all axes to ensure all movements in the 3D space are detected [10].

To find a suitable IMU for the project several IMUs were compared based on performance. These devices range in price from less than 200 NOK for commercial use, to well over 200 000 NOK for application in Aerospace. The IMUs looked at for this project were all on the cheaper end, below 700 NOK. These sensors are less accurate and have a smaller operational range but are sufficient enough for the project. For easier comparison of the IMUs, selected technical specifications relevant to the project were found for each device and noted in an Excel sheet (See Appendix U).

Based on this information the IMUs were ranked in a Pugh matrix (as seen in Table. 2). Scores were given based on performance compared to each other. As there were eight IMUs being compared, the best for each category was rewarded eight points. The remaining IMUs were further given points after order of performance. First place was awarded eight points, second place seven points and so on. If specification for an IMU was not found, the IMU was awarded zero points in that category.

Table 2: PUGH matrix - IMU

Name	Adafruit BNO055	Adafruit ICM-20948	Adafruit LSM9DS1	Adafruit BNO085	HiLetgo MPU9250	GY-85 BMP085	HWT901B-TTL MPU9250	SEN0373
Voltage	1	7	2	6	6	6	8	3
Bitrate Accelerometer	3	7	7	2	7		8	7
Bitrate Gyroscop	8	8	8	8	8			8
Bitrate Magnetometer	5	8	8		8			
Function	8	8	8	8	8	1	2	8
Range Accelerometer	8	8	5	3	8		2	5
Range Gyroscope	8	5	6	4	4		4	8
Range Magnetometer	5	8			6			4
Sensitivity Accelerometer	8	7	7		5			4
Sensitivity Gyroscope	8	7			5			7
Sensitivity Magnetometer	6	7			5			
Lowpassfilter Bandwith Accelerometer	7	8		6			5	4
Lowpassfilter Bandwith Gyroscope	7	8						6
Lowpassfilter Bandwith Magnetometer		8						
Bandwith	8							
Dimensions	6	7		5	8		4	
Price	5	6	3	5	7		2	8
Score	101	117	54	47	85	7	35	72

Although the “Adafruit ICM-20948» received the highest score in the Pugh matrix, the “Adafruit BNO055 was chosen as it has an onboard “Cortex M0” processor. This allows for faster processing and handling of data, which in turn offers more functionalities like different forms of data output.

As there seemed to be a high demand for IMUs the “Adafruit BNO055” selected for the

project was only available on a single webpage but went out of stock before Applied Hydrogen managed to buy it. Therefore, the “Adafruit ICM-20948” was chosen, as it was the second best IMU from the comparison.

Because of a test directly after the choice of IMU was made, yet another IMU had to be ordered. The "SEN0373" (as seen in Fig.12) was the IMU selected for the test and has been utilized ever since. It ranked as number four in the original comparison, but was chosen due to its availability. This guaranteed that the IMUs could be acquired in time so that the first tests could take place.

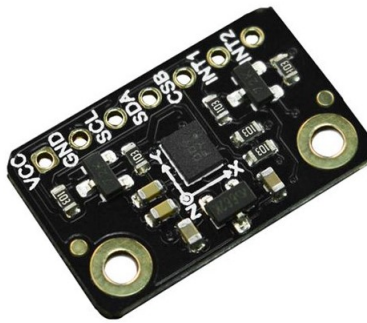


Figure 12: SEN0373 Inertial Measurement Unit

3.2.2 GPS Module

Alongside the IMU, Applied Hydrogen wanted a GPS placed in the excavator. All newer models of Volvo excavators have GPS tracking as a feature, but Applied Hydrogen do not have access to this data. Therefore a separate GPS module would be necessary.

To find a suitable GPS for the project, several different GPSs were compared based on performance. As these devices can range in price from less than 200 NOK for commercial use, to well over 20 000 NOK for application in Aerospace, the GPS looked at for this project were all on the cheaper end, below 600 NOK. In this price range the devices are less accurate and have a smaller operational range, but were enough for the project. For easier comparison, selected technical specifications relevant to the project were found for each device and noted in an Excel sheet (See Appendix V).

Based on this information the GPSs were ranked in a Pugh matrix, as seen in Table 3. Here scores were given based on performance compared to each other. As there were six GPSs

being compared, the best for each category was rewarded six points. The remaining GPSs were further given points after order of performance. First place was awarded six points, second place five points and so on. If specification for a GPS was not found, the GPS was awarded zero points in that category. In the Pugh matrix the “Adafruit Ultimate GPS” (as seen in Fig.13) received the highest score.

Table 3: PUGH matrix - GPS

Name	Seeed Studio GPS	Adafruit Ultimate GPS	MikroElektronika GPS	Digilent Pmod GPS	SparkFun GPS	Adafruit GPS
Voltage	4	5	3	1	3	6
Power Consumption Tracking	4	6			5	3
Power Consumption Acquisition	5	6				4
Tracking Channels	6	6			4	3
Acquisition Channels	6	6			4	3
PRN Channels		6				6
Operational Limits Altitude	5	5			6	5
Operational Limits Velocity	6	6			3	6
Operational Limits Acceleration	3	6			6	6
Sensitivity	3	6	2	6		6
Position Accuracy	3	5			6	5
Velocity Accuracy	5	5			6	5
Timing Accuracy	6	5			4	3
Acceleration Accuracy						6
Update rate	1	5	2	3	6	5
Dimensions	6	5				4
Price	6	5	1	2	3	4
Score	69	88	8	12	56	80

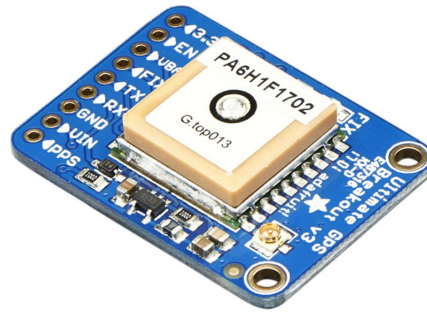


Figure 13: Adafruit Ultimate GPS

3.2.3 Raspberry Pi

To be able to collect data one needs an applicable system that can receive, filter and store data in a sustainable way. Based on the requirements of storing data both locally and on the cloud, the team decided to utilize a Single-Board Computer to fulfill the requirement. The choice of Single-Board Computer (SBC) eventually came down to the Raspberry Pi and the Orange Pi. Ultimately, the choice was relatively simple.

Compared to the Orange Pi, the Raspberry Pi comes with multiple advantages such as an established community, active forum discussions, as well as ongoing updates to both its Linux operating system and applications. Although the Orange Pi might have slightly better computing power, the team valued the ease of use of the Raspberry Pi more [11]. The team also has previous experiences working with the Raspberry Pi and Raspbian operating system.

3.2.4 Arduino-based CAN bus card

The system consists of two CAN buses. One CAN bus is part of the current excavator, while one is a part of the hydrogen driveline extension. Because there are two CAN buses the system will need hardware that is capable of collecting data from both. Using the Arduino Due with integrated dual CAN bus connection from Copperhill Tech (as seen in Fig.14), the system would be able to collect data from both CAN buses simultaneously before sending it to be stored in the Raspberry Pi. The CAN bus card incorporates dual CAN transceivers and is fully compatible with Arduino IDE, making it effortless to write C code and upload it to the card [12].

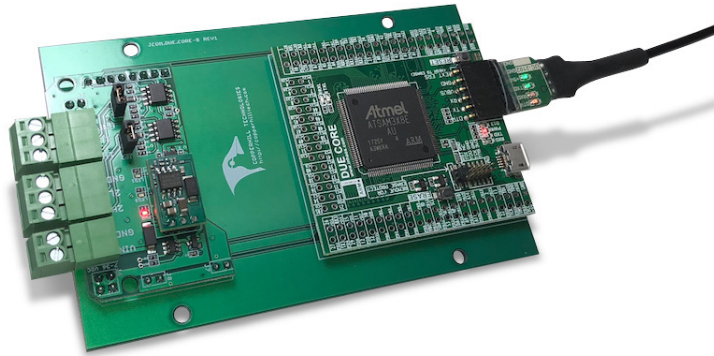


Figure 14: Arduino-based CAN bus card from Copperhill Technologies

3.3 Software

3.3.1 Use Case

When designing a system architecture, use cases are a good place to begin. They can serve as a simple way of describing what's happening within the system. They also help in the derivation of a technical grasp of how the system works.

An actor is represented in the diagram as the person who interacts with the product, such as a user. Each bubble represents a possible action an actor could take in order to interact with the system. The word "**include**" on the arrow indicates that the pointed subsystem the user is acting on contains a component of the system from which the arrow is pointing [3]. When some additional behavior needs to be added, "**extend**" will be used.

As shown in Fig.15, the team created three separate bubbles. The base use case is "**Storing data**", where "**Collecting data**" is included. The included use case addresses collection of data from the CAN bus, Inertial measurement unit, GPS, and camera continuously. "**Displaying data**" will only be called once the user wants to display the collected information on a screen.

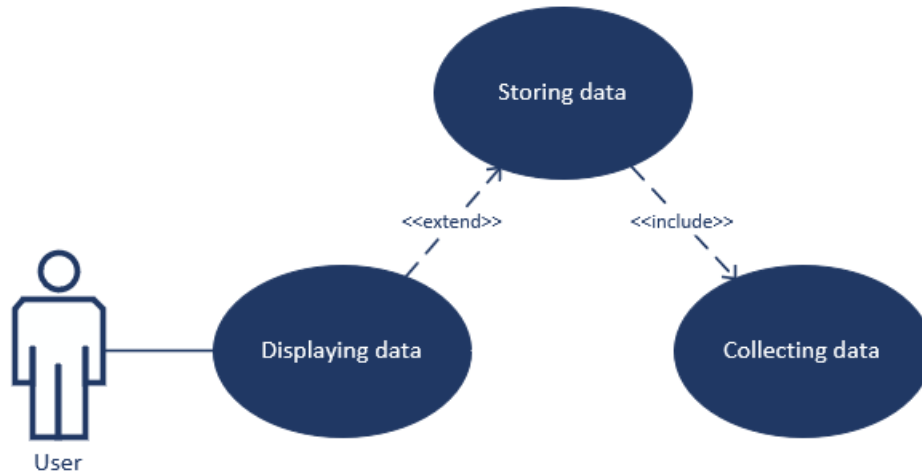


Figure 15: Use case diagram

3.3.2 Sequence Diagrams

The use case diagrams demonstrate the actor's and the use case's static relationships, but they aren't useful in designing an object-oriented system. One will need to turn to the sequence diagrams for that purpose. The diagrams show how classes communicate by sending messages to each other. Each class is used for a particular interaction and relate to a specific use case. They don't provide a comprehensive picture of how classes interact, but they do provide a general understanding [3].

3.3.2.1 Storing Data Sequence Diagram

The sequence diagram shown in Fig.16, presented by the use case, will show how the different components relate to each other. The data from the camera, GPS and IMU will be sent directly to the Raspberry Pi except the CAN-messages from the excavator that is connected to the CAN bus card which is then forwarded to the Raspberry Pi. Once the information is collected, it will be stored both locally on a SD-card, with enough memory for a month (see Appendix R), and on the cloud.

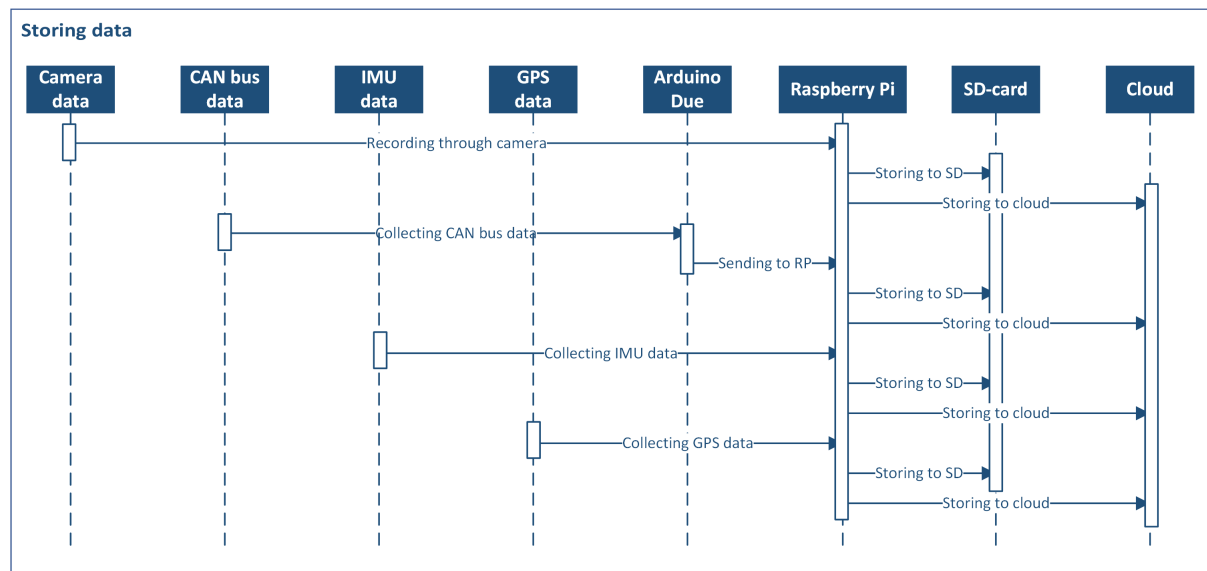


Figure 16: Data storing sequence diagram

3.3.2.2 Display Data Sequence Diagram

In the sequence diagram shown in Fig.17, it shows the relationship between the user and display controller. Here, the user can choose whether they wish to access the camera or other data such as CAN bus logs, IMU, or GPS data. The data will then be displayed through a user interface.

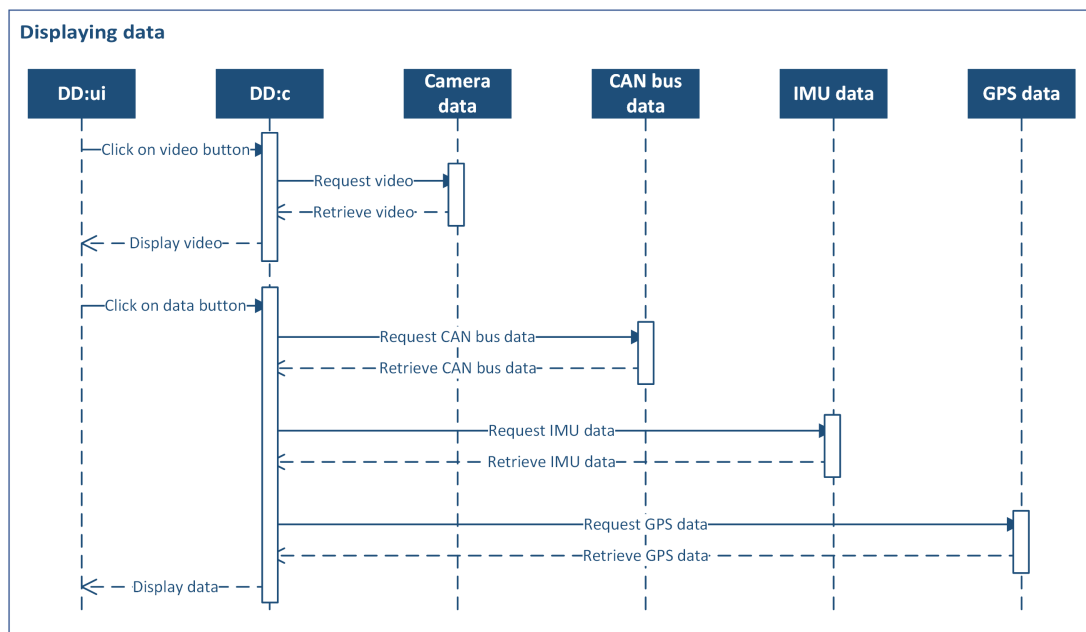


Figure 17: Displaying data sequence diagram

3.3.3 Software Architectural Model

An attractive addition to the other diagrams is the Software Architectural Model. It is deliberately simple, with the purpose of putting all things into perspective. It will show the display of data elements, data storing and the relationships between them as shown in Fig.18.

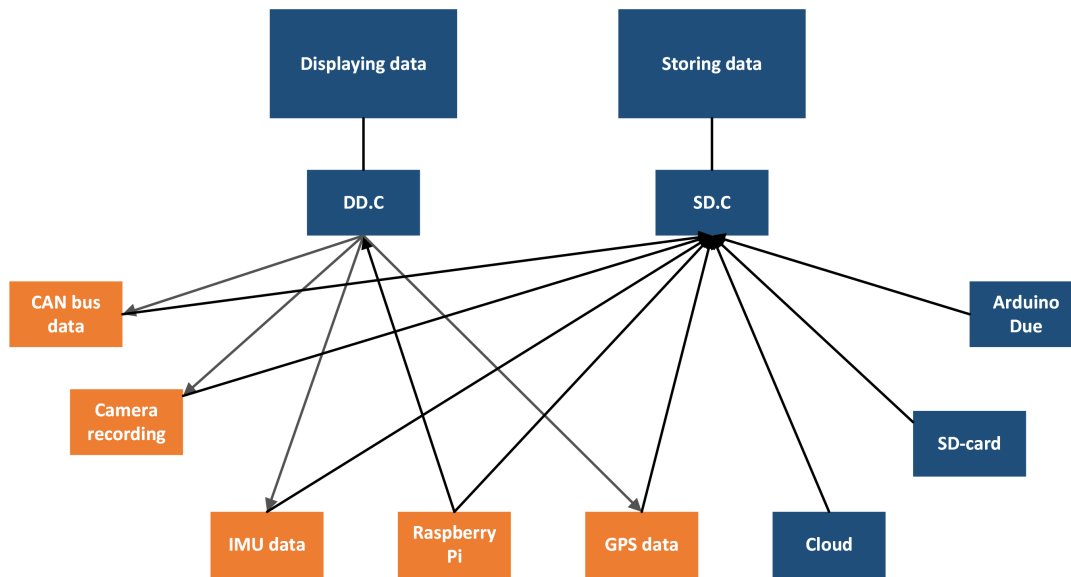


Figure 18: Software architectural model

4 Implementation

This chapter addresses the implementation concerning the proposed system in the previous section. It will discuss both the implementation of the physical design as well as the software design.

4.1 Physical Design

4.1.1 Prototyping

4.1.1.1 Version 1

For the first version of the system, a junction box was bought from Clas Ohlson to house the components as seen in Fig.19. This was done because the box had internal mounting points, and was water and dust proof, which made it suitable for housing electric components in the exposed environment the excavator is working in.

To fit all the required components in the box, the components had to be separated into two different floors as seen in Fig.20. When positioning the components in the box, some holes in the sides of the box had to be made to fit cards like the Arduino Due and Arduino Mega.



Figure 19: Small junction box

This was because the cards with cables attached were too long for the box to contain. A hole for access to power and cable for display was also made. These holes meant that the box was no longer water and dust proof but could still be used for the prototype. However, a solution for the lack of space had to be found and implemented in later versions of the system.

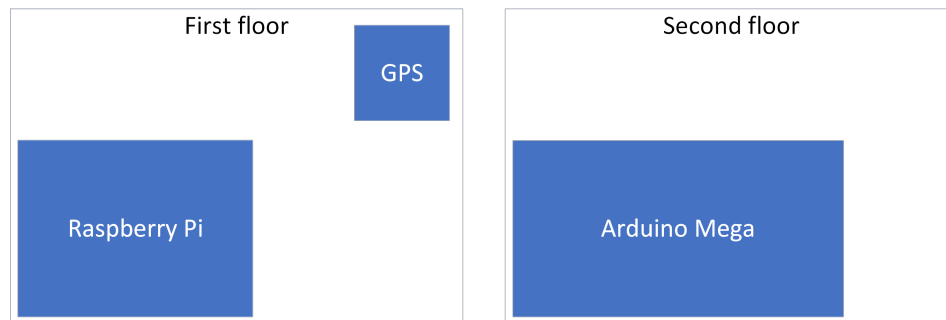


Figure 20: Floor layout version 1

On the first floor, the Raspberry Pi and GPS were located. This was because access to important ports such as power and display were necessary. For these ports to be accessible from the outside, they had to be in line with the holes made in the box. This meant that the Raspberry Pi had to be located on the first floor, as the second floor was higher up than the holes. The GPS was also located on the first floor as it was to be connected to the Raspberry Pi by cables.

On the second floor, the Arduino Mega was located. Because of the small size of the box used, the Raspberry Pi and Arduino Mega could not fit on the same floor. Therefore, the Arduino Mega was placed on the second floor. For the most accurate measurements the IMU was chosen to be placed in its own box as seen in Fig. 21. This box was to be 3D-printed with a mounting bracket included as a part of the box, for the most accurate transfer of movement and vibrations.

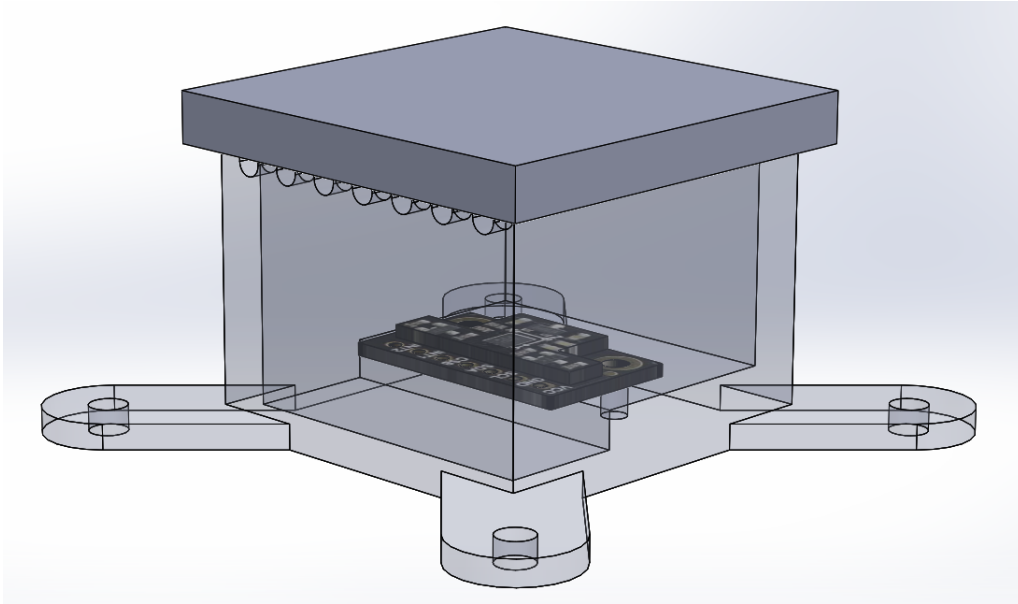


Figure 21: External box for IMU

The Arduino Due was not mounted in the box for the first versions of the system. This was because its features were not yet ready to be implemented in the system, which required the card to be easily accessible for testing. Due to the card having to be tested by itself and not part of the system, it was chosen not to be mounted in the box until code for recording CAN-data was completed and data could be transferred from the card to the Raspberry Pi

4.1.1.2 Version 1.1

For this version the Arduino Mega was replaced by an Arduino Uno. This was because the Arduino contained all the necessary functions, but only took up about half the space of the Arduino Mega. The Arduino Uno, with cables attached, was small enough for the box no

longer needing holes. This meant that yet again the system could be water and dust proof, if a new box was bought.

4.1.1.3 Version 1.2

For the test planned 07.04.22, 3D-printing was unavailable. Because of this, the separate box for the IMU, as seen in Fig.21, was not possible to produce and the IMU was therefore mounted in the box with the other components. The IMU was mounted on the second floor as it was to be connected to the Arduino Uno.

In this version the GPS was also moved to the second floor. This was due to the need for good signals. Having the GPS on the second floor meant that there was nothing above blocking signals, except the lid.

To help create space between the two floors, walls were created to prevent the second floor from laying directly on the components mounted on the first floor. This mainly helps prevent the cables from being bent and potentially broken, as well as creating space for the air to cool the Raspberry Pi.

4.1.1.4 Version 1.3

For this version the Arduino Uno was removed completely from the system. It had previously been used to collect IMU data and transfer it to the Raspberry Pi. As this was made possible on the Raspberry Pi, the need for the Arduino Uno was no longer there, and it could therefore be removed. This change freed up a lot of space in the box, and two floors were no longer necessary to mount all the components.

4.1.2 Final Design

As mentioned previously, there were problems fitting the bigger components with cables attached, such as the Arduino Due. To overcome this issue, a bigger junction box was bought from Clas Ohlson to house all the components for the final system as seen in Fig.22.

In this version the Arduino Due is included, so the box has to contain two floors, as seen in Fig.23. Although all the components would have fit on one floor, it was opted against as this would leave no room to access ports like power for the Raspberry Pi.

On the first floor, the Arduino Due and IMU are placed. The Arduino Due is placed there



Figure 22: Big junction box

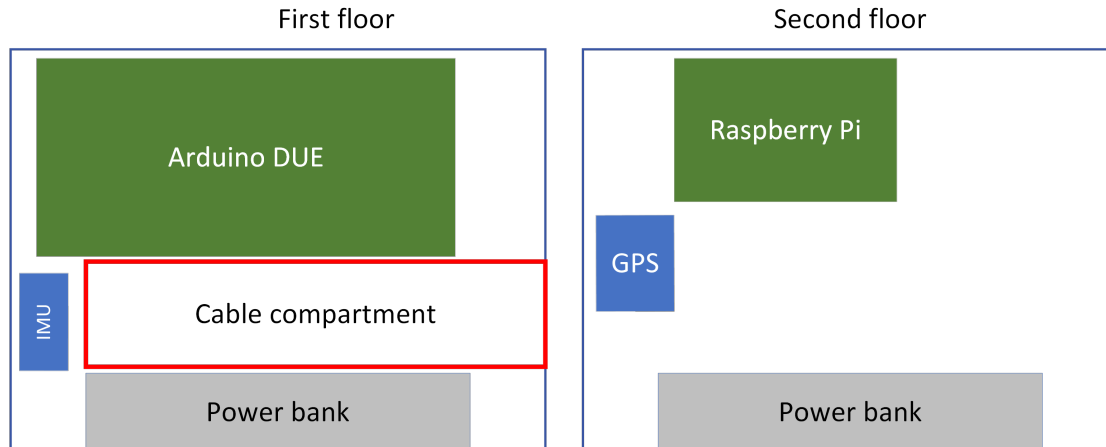


Figure 23: Floor layout for final system

because access to this component is not as important as with some of the others, and when the system is assembled the second floor is blocking access to the first floor. The Arduino Due still has to be connected by cable to both the excavator and the Raspberry Pi. These cables are prerouted during the assembly for easier access when the second floor is mounted.

Although the IMU is connected to the Raspberry Pi by cable, it is placed on the first floor as this places the IMU closer to the mounting point of the box. This lets the movement and vibration of the excavator travel a shorter distance, and therefore give more accurate readings.

On the second floor the Raspberry Pi and GPS are located. The Raspberry Pi is the most important component as it is the centre point of all data in the system. Because of this it is the component most in need of access and is therefore placed on the second floor in the box. This allows for access to all ports and pins on the card, without the need of disassembling the system if changes or troubleshooting is needed.

The GPS, like in version 1.2, is placed on the second floor. This leaves only the lid covering the antenna, minimizing the disruption of signals. A real-time clock function was also added to the GPS by soldering on a mount for a clock battery. This allows the system to keep track of time and keep its clock updated even when the system is powered off.

To power the system and not having to rely on electricity from an external source, a power bank with a capacity of 10000 mAh is implemented in the box. This power bank is connected to the Raspberry Pi delivering power to the other components. It is placed alongside one of the long sidewalls of the box, in a cut-out made in both floors of the box. To better organize the excess cables, both from the power bank and Arduino Due, two walls were added to form a compartment. These are located on the first floor between the Arduino Due and the power bank.

4.1.3 Production Method

To make creating and changing the system easier, a 3D-model of the system and the components used was created using Solid Works (as seen in Fig.24). This allowed for experimenting with designs and ideas without the need of physical parts. Creating 3D-models of the components also made it easier during the production of other components, as measurements for holes and clearances could be taken from the 3D-model, eliminating the need to measure components several times.

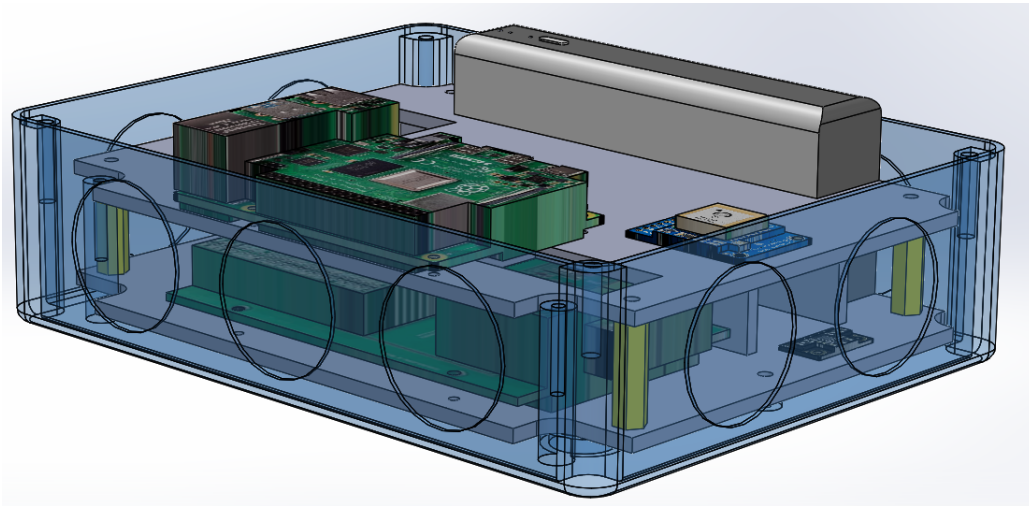


Figure 24: 3D-model of final system

The mounting plates used as floors in the system were laser cut from sheets of wood and not 3D-printed with plastic as seen in Fig.25. This method of production was chosen as it allowed for faster production time and eliminated the need for help from a teacher to operate the 3D-printer. This allowed for fast correction of the product if mistakes were made in the 3D-model, as laser cutting a new plate takes a fraction of the time it takes to 3D-print. As the plates were to hold small and light electrical components, they did not need significant strength, and the sheets of wood would be strong enough to support the weight.



Figure 25: Laser cut plate

4.2 Software Design

4.2.1 Choosing Programming Language

Initially when choosing what programming language to use for the system, C/C++ looked like an appropriate option. The Arduino-based CAN bus card would follow this preliminary assessment, but throughout the stage of implementation Python was utilized in a larger portion than initially planned. Because the Inertial Measurement Unit had limited options regarding C language libraries and the existing code base for Python was larger, it became the standard language used for the Raspberry Pi-side of the system.

4.2.2 CAN-messages

4.2.2.1 Arduino Communication Problem

In order to transmit a virtual message to the CAN bus card, an Arduino Uno and MCP2515 CAN controller was used to send messages (read more about MCP2515 in appendix II). At the start there was a communication problem between the Arduino Uno and CAN bus card. After some days of troubleshooting, we discovered that the CAN bus card needs terminating resistors at the end of each bus.

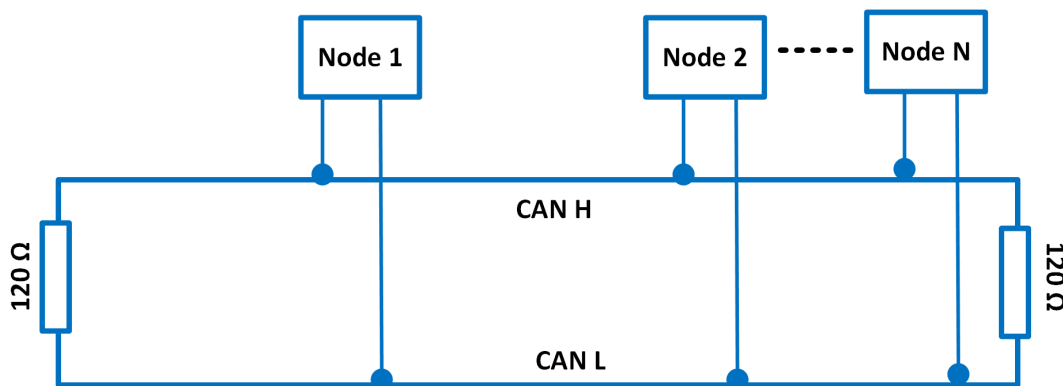


Figure 26: CAN bus resistors

As shown in Fig.26 it is necessary that both nodes are terminated with a 120 Ω resistance resulting in a 60 Ω difference between CAN High and CAN Low[13, p.12-14]. Without the applied resistors there will be no communication between them. To solve this problem the jumper on the CAN bus board was activated in order to work as a termination resistor (as shown in Fig.51).

4.2.2.2 Physical Encoding of CAN duty and Troubleshooting

As mentioned earlier, a bus consists of two wires, CAN High and CAN Low. There are two logical states defined by the CAN, the differential voltage between CAN_H and CAN_L corresponds to one of two logic states: dominant (logic 0) or recessive (logic 1). Logic states are recessive if the differential voltage is less than 0.5 volts on a reception condition and less than 1.5 volts on a transmission condition. If the differential voltage is higher, the logic state is dominant [13, p.12].

- Dominant (Logic 0): CAN_H = 3.5V, CAN_L = 1.5V
- Recessive (Logic 1): CAN_H = 2.5V, CAN_L = 2.5V [14, p.10-12]

In order to find the problem, one had to check if the voltage was outside of the ranges above and measure the resistance between CAN_H and CAN_L.

- If there are 60 Ω terminators between CAN_H and CAN_L, both are working correctly.
- If there is 120 Ω between CAN_H and CAN_L, one CAN bus terminator is not working properly.
- If there is 0 Ω between CAN_H and CAN_L, one or both CAN bus terminators are not working properly.

Since voltage changes quickly, a voltmeter could not display a constant voltage or an exact voltage for CAN High or CAN Low. It was necessary to use an oscilloscope to observe the changes occurring on the CAN bus. As mentioned earlier, a jumper was used which is utilized on the CAN bus board as a resistor to solve the issue.

4.2.2.3 Collecting CAN-messages

As mentioned in Sec. 3.2.4 in order to receive data both from the excavator and hydrogen driveline extension a CAN bus board with dual CAN bus transceivers is used. The system will use both J1939 and CANopen.

J1939 is a standard maintained by the Society of Automotive Engineers. The standards describe how information can be transferred across a network in order to allow ECUs

(computers) to exchange information with one another (e.g. vehicle speed information)[14, p.81-89].

4.2.2.4 CANopen

CANopen is a standard containing several protocols and recommendations, ranging from the physical layer all the way up to the application layer. It is also similar to J1939, since it specifies not only connectors, cables and synchronization functions, but also transport-level protocols, network management features, and standard services, including time and synchronization, and especially a set of standards for application-level objects grouped into application profiles and device-specific profiles [14, p.190].

4.2.2.5 Data Frames

The use of data frames provides a means of transferring information between a node and one or more recipients. There is no explicit addressing used to identify the message recipients when using data frames. An example of this is when a receiver node indicates what messages it will receive based on the data that they contain. This information is encoded as part of the identifier field in the frame. The CAN protocol uses two types of message identifiers and the formats for CAN messages will depend on the type of message identifiers used.

The standard frame is defined as a frame whose identifier is an 11-bit value. With the 2.0 version of the protocol, extended frames have been made available, which is embodied by a frame with a 29-bit identifier field. There is no restriction to which node can transmit standard or extended frames on the same bus. Using the arbitration part of the protocol, messages with 29-bit and 11-bit identifiers can be transmitted on the same network regardless of identifier version [14, p.14].

4.2.2.6 CanSniffer code

The code from appendix E1 has the functionality to read the incoming CAN-messages from the excavator. Both the `can_due.h`, and `DueLayer.h` [15] libraries were used as they contain tons of functions.

In order to have a faster transferring speed over the communication channel, the baud rate 115200 is used for the Arduino Due which is faster than the standard 9600.

The `printHex(a,b)` function will store the data temporarily in an array and convert the data to hexadecimal.

In the `loop()` function, a while loop was used to receive the CAN messages both in extended and standard frame. The data will come as bytes and will always be somewhere between 0 and 8 bytes in length. Because `Serial.print()` will not be accepted by Arduino Due when transmitting data to the Raspberry Pi, `SerialUSB.print()` was used instead. The flowchart in Fig.53 gives an overview of how the code works.

4.2.2.7 Serial Communication

Aiming to send CAN-messages received by the Arduino-based CAN bus card to the Raspberry Pi, the team managed to establish a serial communication between both devices. Appendix D1 shows the flow of the serial communication script.

4.2.3 Sensor Data Collection

4.2.3.1 IMU

In order to communicate with the Inertial Measurement Unit, the device can be connected directly to the Raspberry Pi through an Inter-Integrated Circuit (I2C), as seen in Fig.27. I2C is a fairly common communication protocol that allows for data transfer between on-board peripherals [16].

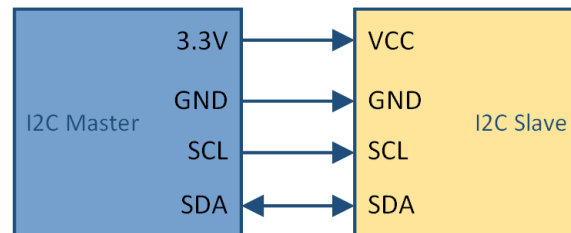


Figure 27: I2C communication

Using the BMX160 library from DFRobot [17] one can control the magnetometer, accelerometer and gyroscope through I2C communication. Using the `bm.begin()` function the script (as seen in Appendix E14) initializes the i2c communication and returns the status whether the initialization succeeded or not. The `bm.get_all_data()` function is used to get and return the data from the different sensors.

4.2.3.2 GPS

The GPS uses UART communication to communicate with the Raspberry Pi (as seen in Fig.28). UART stands for Universal Asynchronous Receiver and Transmitter and is a transmission protocol for serial data used to communicate with a device serially [18]. Using `gpsd`, an interface daemon for GPS receivers, the Raspberry Pi is able to extract

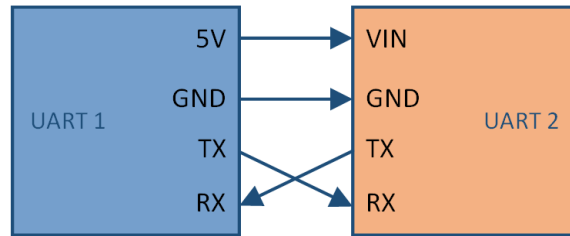


Figure 28: UART communication

and display the GPS data [19]. One can also specify the GPS device to `gpsd` at startup to start monitoring it on boot. To grab the information from `gpsd` in python the team utilizes a buffer (as seen in Appendix E13) that loops and grabs each set of `gpsd` data using `GPSPoller()`. It will continue to grab sets of data with a one second interval until the buffer is fully clear.

4.2.4 Synchronizing Data

4.2.4.1 Run on Boot

In an effort to synchronize the different scripts there must be boot scripts (as seen in Appendix E11) available that can run whenever the device is booted up. Using desktop files and making them executable is a seamless way of doing it. It only requires to provide a complete path to the directory where the scripts are located in order to run them, as seen in Fig.29.

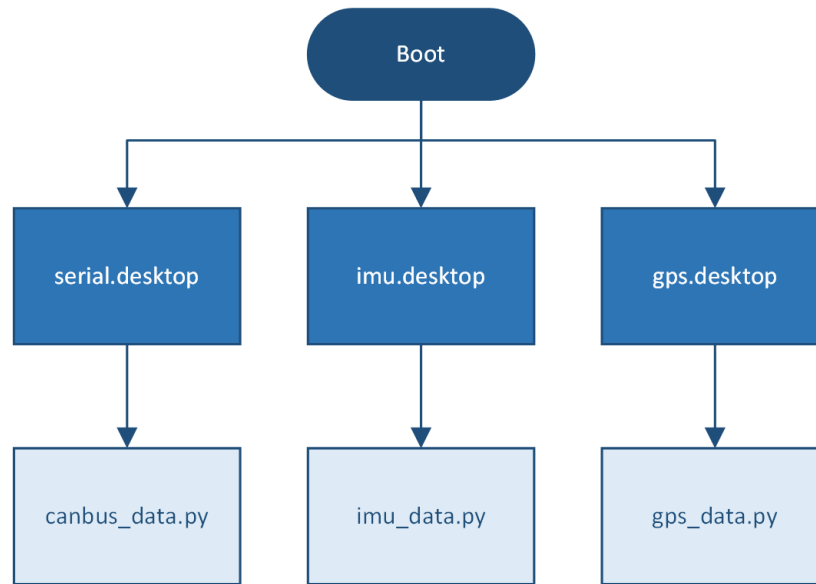


Figure 29: Boot flow

4.2.4.2 Real-time Clock

Synchronizing data includes adding timestamps to the incoming data in order to ensure different types of data is collected simultaneously if it were to be reviewed in non-real time. Out of the box the Raspberry Pi does not include a real-time clock and needs an external real-time clock to achieve said functionality [20].

Instead of using a real-time clock module the team decided to use the GPS's real-time capability in order to fulfill this necessity (as seen in Fig.30). Using the chronyd package, a package for system clock synchronization [21], one can set the system clock equal to the GPS's built-in clock. More information regarding RTC implementation can be found in Appendix Y.

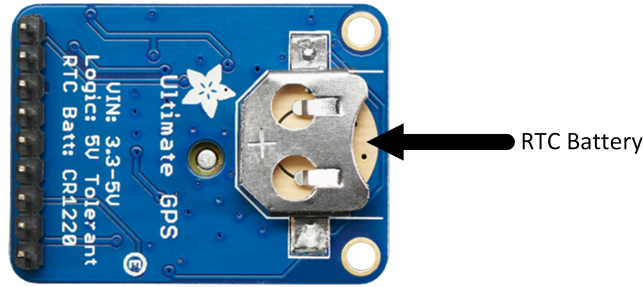


Figure 30: RTC-battery compatible GPS

4.2.5 Local Storage

When it came to local storage, the team decided to utilize the SD-card on the Raspberry Pi to store data in text files. The reason as to why the team decided on text files over comma separated files (CSV) was that occasionally the data would contain commas which would collide with the CSV syntax.

Utilizing a real-time clock meant one could assign a timestamp to each text file, creating a file for every session. All data would eventually be converted to string to satisfy the requirements of a text file (as seen in , Fig.31, Fig.32).

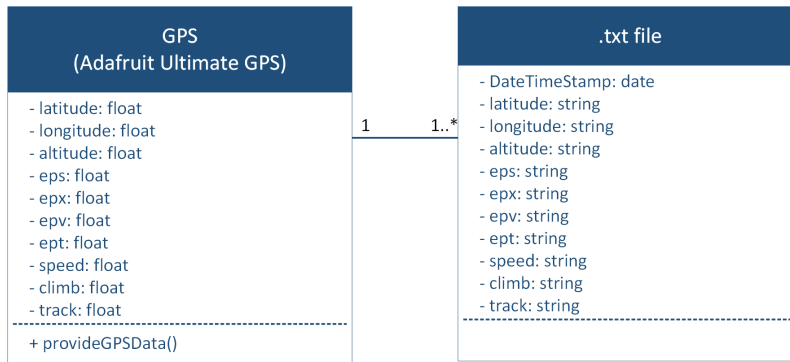


Figure 31: Class diagram - GPS data to text file

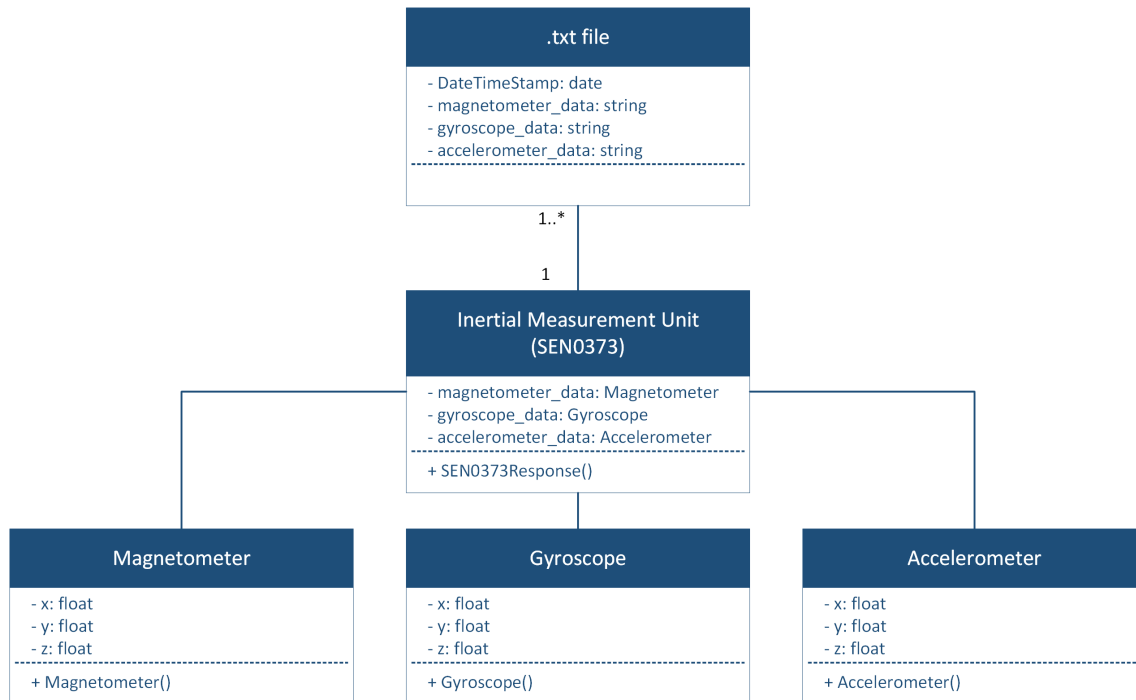


Figure 32: Class diagram - IMU data to text file

4.2.6 Cloud Storage

4.2.6.1 Streaming Various Data

Sending data allocated by the Raspberry Pi to cloud storage is among the most important requirements in this project. Data can be shared in a variety of cloud services, such as Amazon Web Services, Microsoft Azure, and Google Cloud. They all follow the same path, but they have some differences.

4.2.6.2 Real-time Data Processing

Google Cloud Platform (GCP) was utilized for the system as it houses one of the largest databases, BigQuery. Because some databases cannot handle as many messages at once and others have limited storage space, it is critical to pick a suitable one.

More cloud storage is required in order to store large amounts of data sent by the Raspberry Pi. It is capable of deploying the world's largest IoT-platform. Internet of Things (IoT) is a

network of devices that communicate with one another. Through some form of connectivity, all of the devices and sensors can receive and transmit data to a cloud service [22]. Google Cloud delivers information directly to BigQuery, a massive parallel query engine able of collecting a million rows of metrics per seconds, as well as running queries on terabytes of data in less than ten seconds [23].

The data collection unit can collect data from various devices and sensors from any location. There is no need to build a large distributed front end, load balancers, app servers and so on. BigQuery provides all of these services in Google’s largest data center.

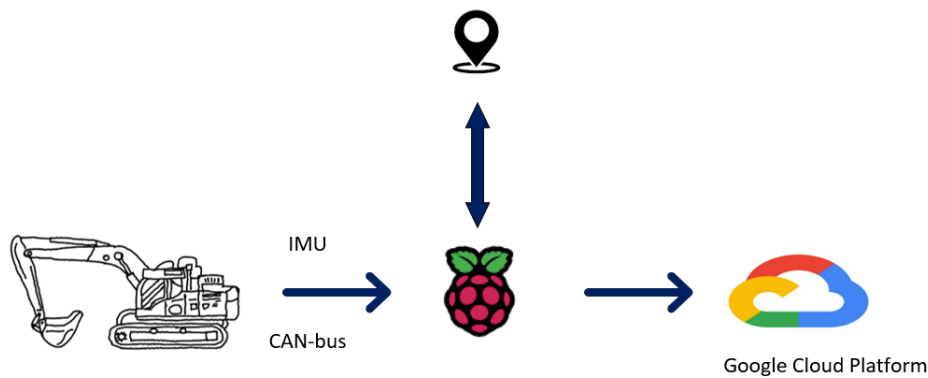


Figure 33: Raspberry Pi data handling architecture

4.2.7 Google Cloud Platform

4.2.7.1 Google Cloud Console

IoT Core is a Google cloud service. It allows users to easily interact with connected devices, in this case a Raspberry Pi. It is scalable, as most cloud services, because it can handle billions of messages [24]. The Google Cloud Platform IoT message server’s implementation is based on MQTT. MQTT stands for Message Queuing Telemetry Transport, and is a lightweight messaging protocol that is utilized in IoT Core to transmit and receive information [25]. As seen in Fig.34 the cloud storage architecture consists of the Raspberry Pi and several cloud services, including Cloud IoT Core, Cloud Dataflow, Cloud Pub/Sub, and BigQuery.

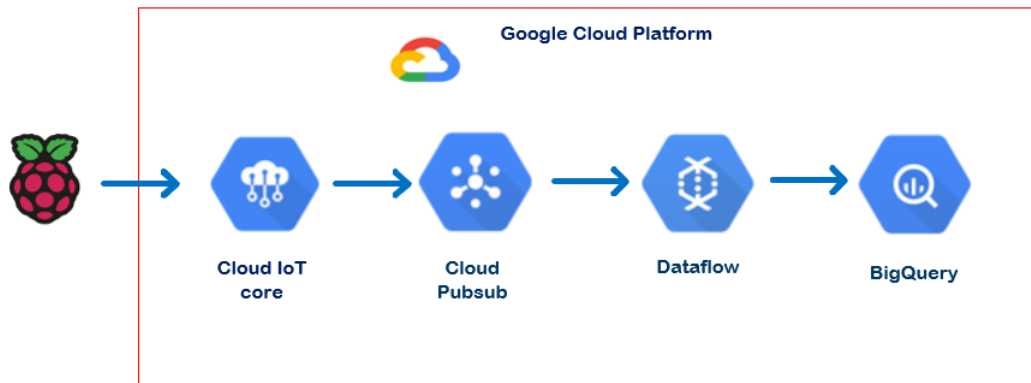


Figure 34: Cloud storage architecture

4.2.7.2 APIs

Google Cloud offers a wide range of APIs in various categories [26]. However, as shown in Fig.35, the necessary APIs have been enabled. Cloud IoT Core is used to manage and ingest data from devices. It essentially acts as the authentication service for the Raspberry Pi. When sending data, Cloud Pub/Sub is enabled, allowing to subscribe to a specific topic. Cloud Pub/Sub is used as a queue engine to collect data from MQTT or publisher.

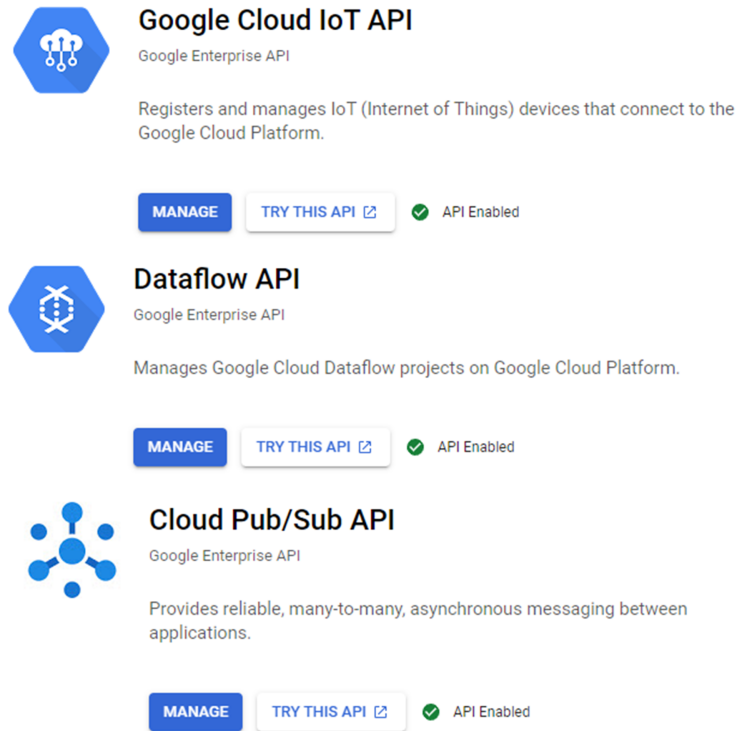


Figure 35: Enabled application program interfaces

4.2.7.3 Cloud Iot Core Service

Fig.36 outlines the steps taken in order to establish cloud connection. Using a configured service account to generate new certificates and attaching policies to them, allowed access from the Cloud IoT Core platform with appropriate certificates. Using filezilla, the credential JSON file certificate was transferred over to the Raspberry Pi.

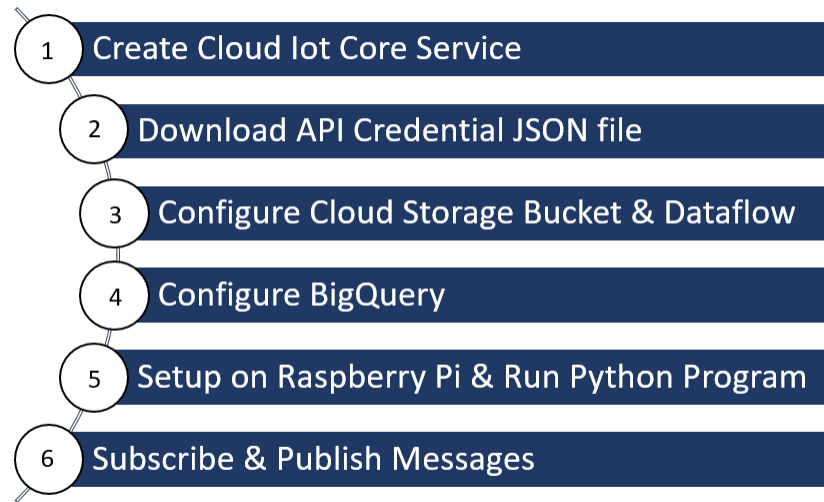


Figure 36: GCP IoT Core and Raspberry Pi setup

4.2.7.4 GCP Iot Core and Raspberry Pi

The Python script (as seen in Appendix E) on the Raspberry Pi sends requests to the Google Cloud service, where they are authenticated using the Cloud IoT Core. A Pub/Sub pulls the request, which is then pushed to dataflow, which then writes data directly to the cloud database.

A Google Cloud token for device authentication is run on the terminal in order to connect the Raspberry Pi to the Cloud IoT Core, enabling the Raspberry Pi to be used for logging data to the cloud [27].

BigQuery and a cloud storage bucket are also employed. Cloud storage buckets are simple data containers. Everything stored in Cloud Storage is kept in a bucket. It is primarily used to organize data and control data access [28]. BigQuery generates a dataset as well as tables, which allows data to be pushed to BigQuery via dataflow. The data is then published to the Pub/Sub topic. The system is also running a cloud dataflow job-template as shown in Fig.37, where data is being pulled into BigQuery tables.

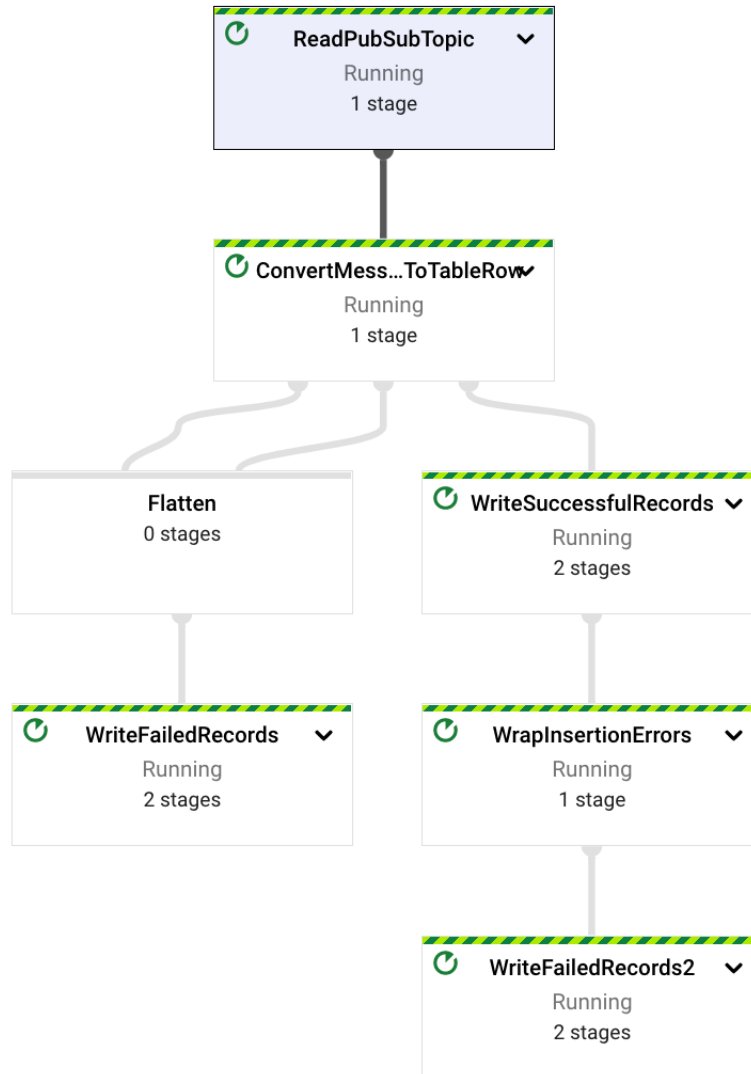


Figure 37: Cloud dataflow job template

4.2.8 Remote Desktop Connection

Remote access is essential in order to view the data from the Raspberry Pi without connecting it to a monitor. It is necessary in this project to be able to view data on a laptop

in order to control it from there if needed. Aiming to connect to the Raspberry Pi using a laptop, Secure Shell (SSH) and Virtual Network Computing (VNC) was used by establishing communication through a local IP address. An IP address is assigned to any device connected to a Local Area Network. If both the laptop and Raspberry Pi is connected using the same local IP address it allows for using SSH and VNC to utilize the laptop as a Raspberry Pi display (as seen in Fig.38) [29].

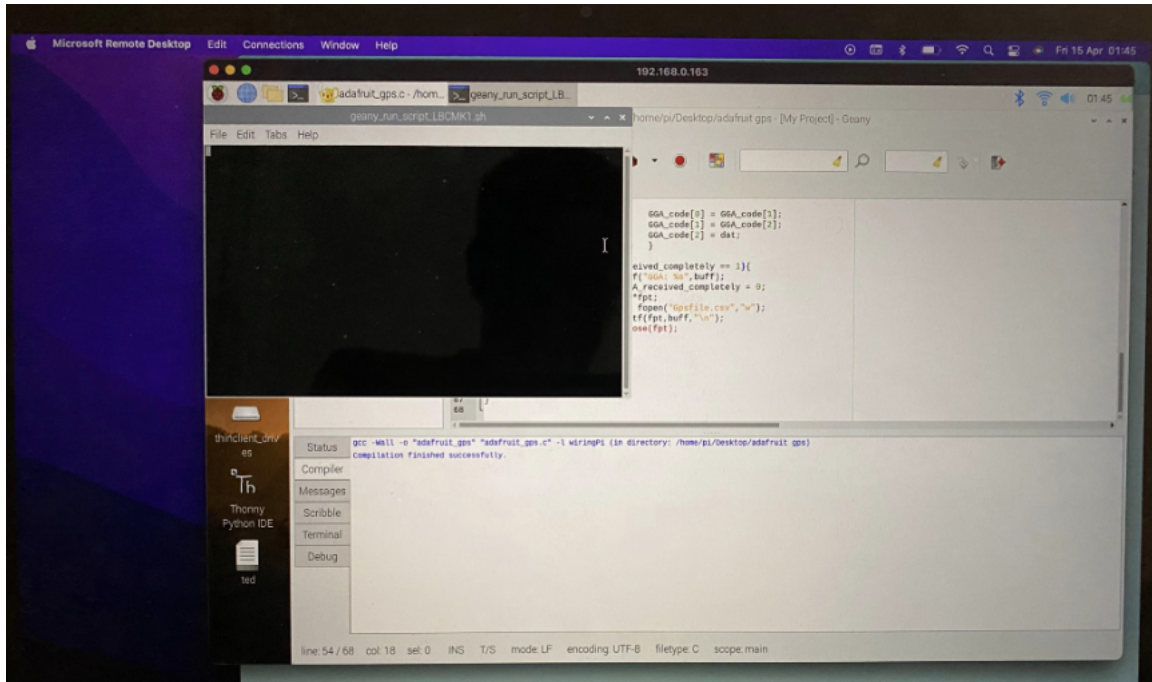


Figure 38: Remote desktop connection

4.2.9 Surveillance

Another addition to the system is a surveillance system using a web camera. The camera was connected to the Raspberry Pi directly through a USB connection. Using `fswebcam`, a simple webcam application for Linux, the Raspberry Pi is able to capture images through the webcam [30]. Once the application was implemented a simple bash script was written in order to take pictures every five minutes. This script also stored the pictures with timestamps as JPG/JPEG files in a predetermined directory. Since JPG/JPEG files contain less data than PNG files [31], it seemed like the better option considering storage space.

5 Review and Future Work

This section will reflect upon the results that have been achieved, what could have been done different, as well as what could be implemented through future development. Looking back at what has been achieved, one cannot help but look for potential improvements to the system.

5.1 Hardware

The system as of now is a closed box with no holes or gaps for air to get through. This means that all circulation of air will be inside the box. As the components produce heat, especially the Raspberry Pi, the air in the box will heat up with no way of cooling. This may compromise performance of the components if temperatures become too high. A potential solution to this could be to circulate the enclosed air inside the box. This will give components with specific hot spots like the Raspberry Pi's processor some airflow which will help cool the components. This could be achieved by mounting a fan inside the box. Holes could also be added to add cooling to the box by helping in circulating the air. This would however make the system no longer protected from water and dust, which could lead to damaged or broken components.

Mounting the box also creates a challenge when it comes to keeping the system secure from water and dust. This is because the box as of now has no way of being mounted without having to create holes for screws. A possible solution would be to create a custom box with a bracket or external mounting points as part of the box. This would make it so that no screw holes through the walls or floor of the box would be necessary and therefore would let it maintain its water and dust proof capabilities.

For finishing details on the box the current mounting plates could be 3D-printed instead of laser cut for a more finished look to the product. This would not increase the strength of the plates by any significant means, but this is not necessary either as they are only supporting small electronic components with no significant weight. It would however increase the production time of the plates as 3D-printing is a much slower process. Another alternative is to still use laser cutting as the production method, but use acrylic sheets instead of wood. This would also give the system a more finished look while keeping the short production time that comes with laser cutting.

The cables currently used in the system are also a point of interest, as they are all premade, meaning many are longer than necessary. This adds unnecessary clutter to the box as the excess cables are in the way. The current solution is having a dedicated compartment inside the box to tuck the excess cables into. This works, but takes up space within the box that could be used for future upgrades or additions to the system. A solution to this could be to reduce the length of the cables by creating custom cables to fit the box. This would free up space in the box as the current compartment could be removed. On the downside shorter cables would leave less room for flexibility when it comes to placement of the components if they ever were to be moved.

As of now the system is powered by a 10000mAh power bank. This was chosen as a temporary solution to power the system while testing. This means that a cable to a power outlet is not needed, allowing for free movement of the system as it is totally wireless. As the system is to be mounted in an excavator, a more permanent solution would be to remove the power bank, and power the system with the battery in the excavator. This would remove the need for recharging and would let the system turn on automatically when the excavator is turned on. To achieve this a converter would need to be used, as the excavators systems run on 25 volts, while the data recorder uses 5 volts.

5.2 Software

In regards to software there are several improvements to be made. There is currently no way of easily streaming data to a display system for potential local demos. Some research has been put into coming up with a solution to this, including finding software used for development of said system. Qt, an application and UI development framework was on the forefront of being utilized for this purpose because of its cross-platform capabilities [32]. As seen in Fig.39 a sketch was made in order to provide a frame of reference to what the user interface could look like.

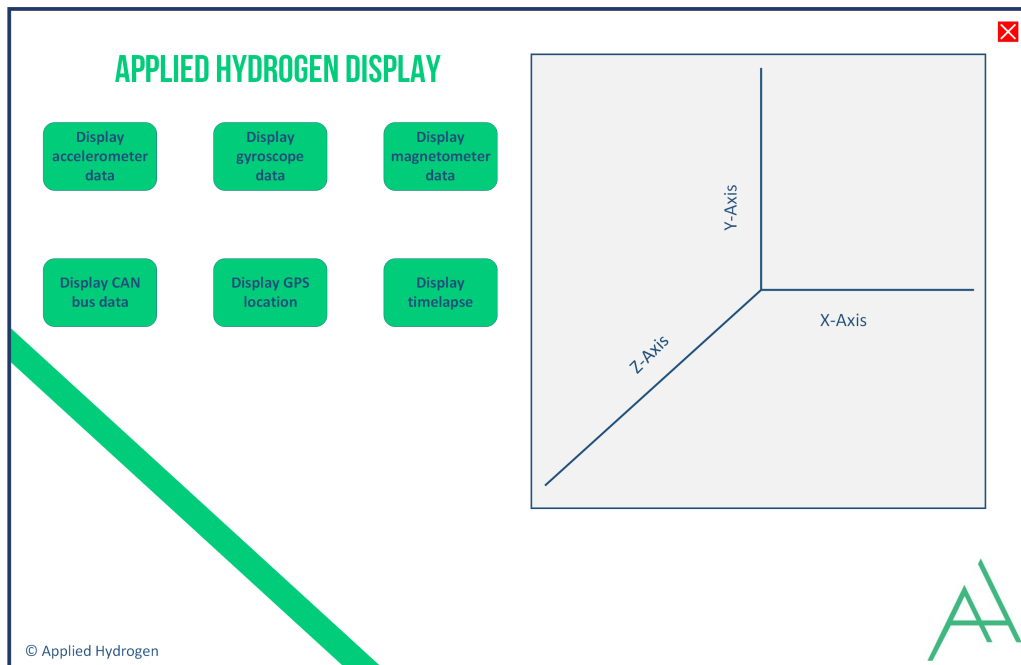


Figure 39: UI sketch

Cloud storage was one of the requirements that wasn't fully developed at the end of the project. While the cloud connection was established, code for storing data to the cloud was not developed. Due to time constraints and inexperience working on a project of this size, the code in general is not as good as it could be. Given more time this could also be improved upon during future development.

6 Conclusion

The goal of this thesis was to implement a system based on a description delivered by clients IoT-labs and Applied Hydrogen. Ultimately, a data collection unit was developed for an excavator based on requirements provided by the clients. The final system is capable of collecting data from multiple devices in real-time. It includes an Inertial Measurement Unit for determining the orientation of the excavator, a GPS module and webcam for tracking and surveillance, as well as a CAN bus card for collecting crucial CAN-data. All of this combined with local storage alongside advancements with cloud storage has resulted in an up and running data collection unit. While a lot of work has been done, there are still areas that require further improvements before the system is ready for deployment in a wide scale industrial setting.

References

- [1] *Construction Emissions*. URL: <https://fuelcellsworks.com/news/china-developing-hydrogen-fuel-cell-construction-vehicles/>.
- [2] Thirumavalavasethurayar P and Ravi T. *Implementation of Replay Attack in Controller Area Network Bus using Universal Verification Methodology*. Department of Electronics, Communication Sathyabama Institute of Science, and Technology Chennai, India, 2021.
- [3] Alina S. et al. *Extended Reality Inspection (XRI)*. Bachelor Thesis. University of South-Eastern Norway, June 2020.
- [4] Kelly Drozd. *Jira confluence scrum*. Accessed Mar. 14, 2022. URL: <https://www.atlassian.com/agile/scrum/jira-confluence-scrum>.
- [5] Max Rehkopf. *Sprint*. Accessed Mar. 14, 2022. URL: <https://www.atlassian.com/agile/scrum/sprints>.
- [6] *The Standard for Risk Management in Portfolios, Programs, and Projects*. Project Management Institute, 2019.
- [7] C. Dominik Güss. *What Is Going Through Your Mind? Thinking Aloud as a Method in Cross-Cultural Psychology*. Aug. 2018. URL: <https://www.frontiersin.org/articles/10.3389/fpsyg.2018.01292/full>.
- [8] K. Heldman. *Project Management JumpStart*. John Wiley and Sons Inc, 2018.
- [9] *What is a system context diagram?* URL: <https://www.gleek.io/blog/system-context-diagram.html>.
- [10] *What is an internal measurement unit?* Accessed Mar. 28, 2022. URL: <https://www.vectornav.com/resources/inertial-navigation-articles/what-is-an-inertial-measurement-unit-imu>.

- [11] Tyler S. Love, Joel Tomlinson, and Derrek Dunn. *the orange pi: Integrating programming through electronic technology*. International Technology Education Association, Oct. 2016.
- [12] *Arduino Due with dual CAN bus port*. Accessed Mar. 21, 2022. URL: <https://copperhilltech.com/arduino-due-based-ecu-development-board-with-two-can-bus-ports/>.
- [13] Yu Zhu. *CAN and FPGA Communication Engineering: Implementation of a CAN Bus based Measurement System on an FPGA Development Kit*. Herstellung: Diplomica® Verlag GmbH, Hamburg, 2010.
- [14] Marco Di Natale et al. *Understanding and Using the Controller Area Network Communication Protocol: Theory and Practice*. Springer New York Dordrecht Heidelberg London, 2012.
- [15] *SBS_CAN*. Accessed Apr. 5, 2022. URL: https://github.com/fhackenberger/SBS_CAN.
- [16] Cang Liu et al. *A Flexible Hardware Architecture for Slave Device of I2C Bus*. 2019 International Conference on Electronic Engineering and Informatics (EEI), Nanjing, China. Nov. 2019.
- [17] *BMX160 Library*. Accessed. URL: https://github.com/DFRobot/DFRobot_BMX160.
- [18] Ashok Kumar Gupta et al. *Design and Implementation of High-Speed Universal Asynchronous Receiver and Transmitter (UART)*. 2020 7th International Conference on Signal Processing and Integrated Networks (SPIN), Noida, India. Feb. 2020.
- [19] *gpsd*. Accessed May. 14, 2022. URL: <http://manpages.ubuntu.com/manpages/trusty/man8/gpsd.8.html>.
- [20] Brendan Horan. *Practical Raspberry Pi*. Apress Berkeley, CA, 2013.

- [21] *chronyd*. Accessed May. 15, 2022. URL: <https://manpages.ubuntu.com/manpages/bionic/en/man8/chronyd.8.html>.
- [22] Iot Core Service. *Iot Core*. Accessed May. 08, 2022. URL: <https://cloud.google.com/iot-core>.
- [23] Google Cloud bigQuery. *What is BigQuery*. Accessed May. 11, 2022. URL: <https://cloud.google.com/bigquery/docs/introduction>.
- [24] Google Cloud Service. *Google Cloud overview*. Accessed May. 11, 2022. URL: <https://cloud.google.com/docs/overview>.
- [25] Iot Core Service. *Iot Core*. Accessed May. 08, 2022. URL: <https://cloud.google.com/iot-core>.
- [26] Google Cloud Service. *Data Analytics APIs*. Accessed May. 05, 2022. URL: <https://cloud.google.com/apis#section-5>.
- [27] Google Cloud Service. *Creating public/private key pairs*. Accessed May. 9, 2022. URL: <https://cloud.google.com/iot/docs/how-tos/credentials/keys>.
- [28] Google Cloud Storage Bucket. *cloud storage bucket key terms*. Accessed May. 11, 2022. URL: <https://cloud.google.com/storage/docs/key-terms>.
- [29] Raspberry Pi. *Remote Access*. Accessed Apr. 7, 2022. URL: <https://www.raspberrypi.com/documentation/computers/remote-access.html>.
- [30] *fswebcam*. Accessed May. 15, 2022. URL: <http://manpages.ubuntu.com/manpages/bionic/man1/fswebcam.1.html>.
- [31] *JPEG vs PNG*. Accessed May. 15, 2022. URL: <https://www.adobe.com/creativecloud/file-types/image/comparison/jpeg-vs-png.html>.

- [32] Zeng Gui-gen et al. *The implementation of oSIP stack in developing Qt software in Embedded Linux*. 2010 First International Conference on Networking and Distributed Computing, Hangzhou, China, Oct. 2010.
- [33] Mabrouka Gmiden, Mohamed Hedi Gmiden, and Hafedh Trabelsi. *Vehicle CAN bus: An intrusion detection method for securing in-Vehicle CAN Bus*. Institute of Electrical and Electronics Engineers, Dec. 2016.
- [34] Guilherme M. Zago and Edison P. de Freitas. *CAN FD: A Quantitative Performance Study on CAN and CAN FD Vehicular Networks*. Institute of Electrical and Electronics Engineers, May 2018.
- [35] Copperhilltech. *can bus protocol*. Accessed Mars. 03, 2022. URL: <https://copperhilltech.com/a-brief-introduction-to-the-sae-j1939-protocol/>.
- [36] B.V.P. Prasad, Jing-Jou Tang, and Sheng-Jhu Luo. *SAE J1939 Standard Document: Design and Implementation of SAE J1939 Vehicle Diagnostics System*. Institute of Electrical and Electronics Engineers, Oct. 2019.
- [37] Camil Jichici et al. *Vehicle SAE-J1939 CAN bus: Effective Intrusion Detection and Prevention for the Commercial Vehicle SAE J1939 CAN Bus*. Institute of Electrical and Electronics Engineers, Feb. 2022.
- [38] Microchip Technology Inc. *Stand-Alone CAN Controller with SPI Interface*. Aug. 2018.
- [39] *BMX160 library*. Accessed Apr. 1, 2022. URL: https://github.com/DFRobot/DFRobot_BMX160/blob/master/examples/readAllData/readAllData.ino.
- [40] *CAN reader application*. Accessed: Feb. 11, 2022. URL: <https://github.com/adamtheone/canDrive>.
- [41] *What is an SD-Card?* Accessed May. 16, 2022. URL: <https://www.businessinsider.com/what-is-an-sd-card?r=US&IR=T>.

- [42] Bosch Sensortec. *BNO055 Intelligent 9-axis absolute orientation sensor, Revision 1.4*. June 2016.
- [43] TDK InvenSense. *World's Lowest Power 9-Axis MEMS MotionTracking™ Device, Revision 1.3*. Feb. 2017.
- [44] STMicroelectronics NV. *iNEMO inertial module: 3D accelerometer, 3D gyroscope, 3D magnetometer, Revision 3*. Mar. 2015.
- [45] CEVA Inc. *BNO080/85/86 Data Sheet, Revision 1.11*. June 2021.
- [46] InvenSense Inc. *MPU-9250 Product Specification, Revision 1.1*. June 2016.
- [47] Inc. Analog Devices. *ADXL345, Revision B*. Nov. 2010.
- [48] Inc. InvenSense. *ITG-3205 Product Specification, Revision 1.0*. Aug. 2010.
- [49] Honeywell. *3-Axis Digital Compass IC HMC5883L*. Dec. 2010.
- [50] Ltd WitMotion Shenzhen Co. *The Robust Acceleration, Angular velocity, Angle , Magneticfield & Air Pressure Detector, Revision 20-0707*. July 2020.
- [51] Bosch Sensortec. *BMX160 Small, low power 9-axis sensor, Revision 1.4*. Nov. 2021.
- [52] Allen Wang. *Ultra High Sensitivity and Low Power GPS Receiver*. Seeed Studio, Mar. 2012.
- [53] CD Technology. *CD-PA1616S GPS patch antenna module, Revision 0.4*. June 2019.
- [54] u-blox. *LEA-6, Revision R13*. Feb. 2017.
- [55] MediaTek. *MT3329 Data Sheet, Revision 0.5*. Feb. 2008.
- [56] SparkFun Electronics. *GPS Breakout - Chip Antenna, SAM-M8Q (Qwiic)*. Apr. 2019.

- [57] GlobalTop Technology Inc. *FGPMMOPA6B GPS Module Data sheet, Revision A07*. July 2011.
- [58] Lady Ada. *Adafruit Ultimate GPS*. Adafruit Industries, Mar. 2022.

Appendices

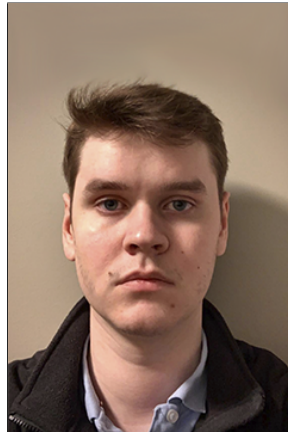
Appendix A - Team Members



Abdirahman Ali

Project lead

Software architect



Petter Tveit

Documentation lead

Software architect



Mahram Safdari

Test lead

Software architect



Saleem Rezaie

Scrum master

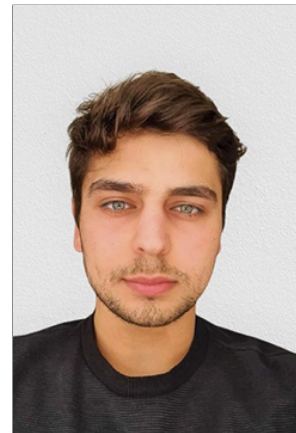
Software architect



Markus Danielsen

Design lead

Requirements lead



Fouad Irkayek

Risk lead

Appendix B - Roles

This section contains brief descriptions of the different project roles.

The project lead is responsible for facilitating team collaboration and being a point of contact for both USN and IoT-labs. They are also responsible for sending and submitting the documentation.

The documentation lead is responsible for ensuring the quality of the documentation, overseeing and approving revisions as well as ensuring the consistency across the thesis. They are not responsible for sending and submitting the documentation.

The test lead is responsible for planning, deploying and managing the tests. They are responsible for applying appropriate testing measurements as well as documenting the tests in separate test tables.

The scrum master is responsible for leading the scrum team in agile methodology and scrum practices. They are also responsible for managing daily stand-up meetings, creating and managing tasks in Jira, and resolving conflicts and issues that may occur.

The design lead is responsible for planning and leading the design processes. They are also responsible for approving concepts and managing production.

The risk lead is responsible for monitoring risks throughout the project cycle. They are also responsible for identifying and adding new risks, in addition to pairing them up with the right tests and requirements.

Software architects are responsible for developing, designing and implementing the software. They are also responsible for troubleshooting and resolving issues within the code, as well as updating the software if required.

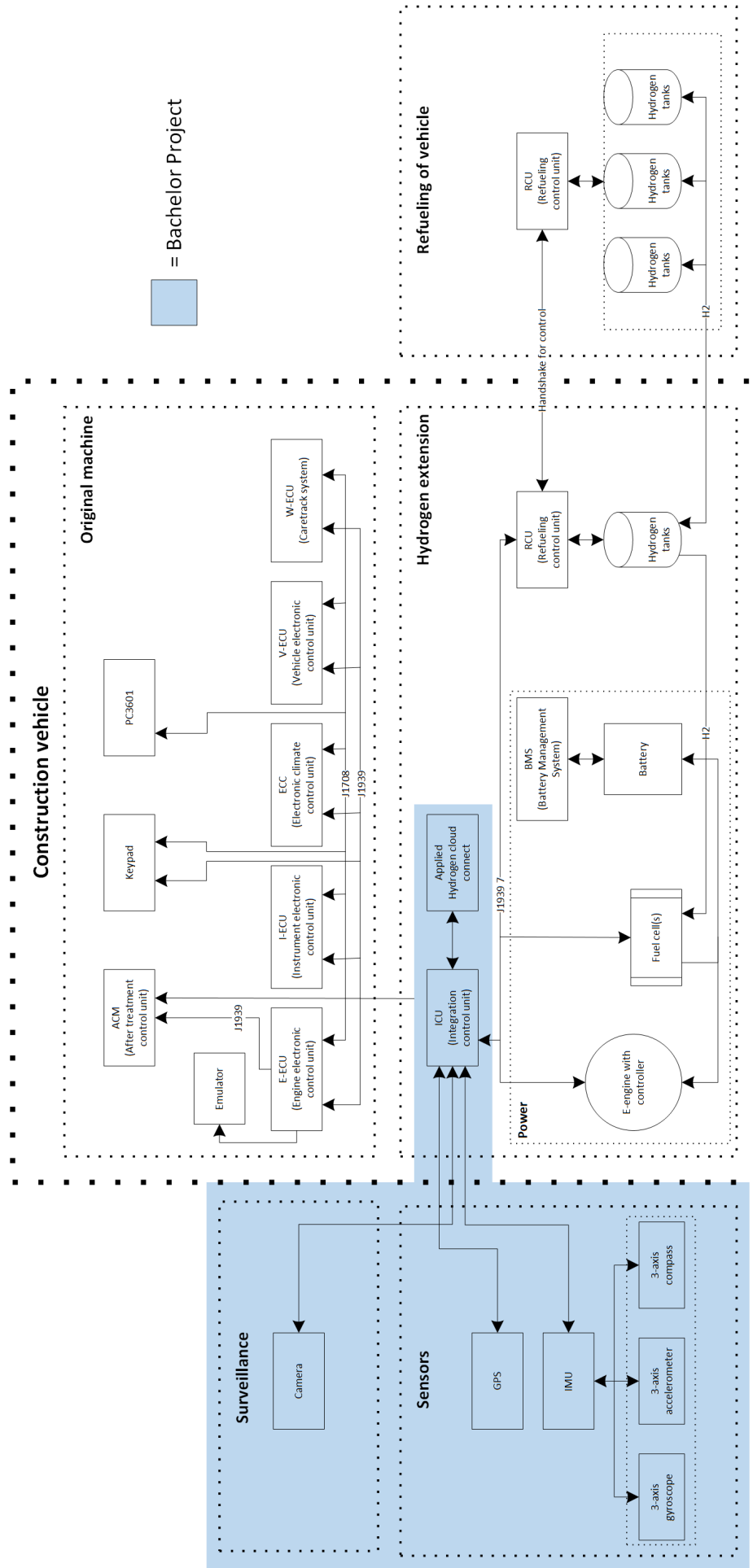


Figure 40: System architecture

Appendix D - Flowcharts

D1 - Flowchart: Serial Communication

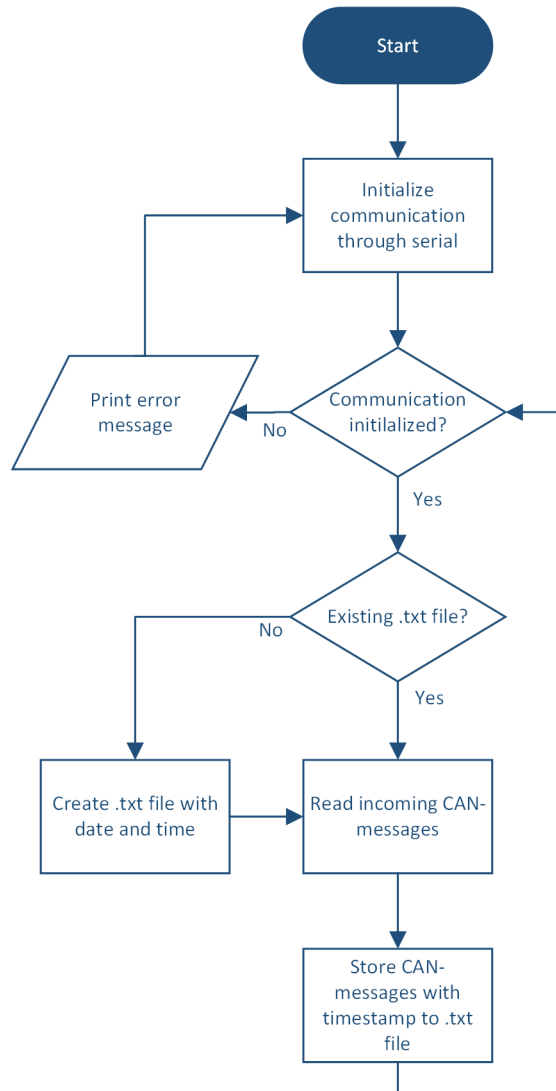


Figure 41: Flowchart: Serial communication

D2 - Flowchart: Inertial Measurement Unit

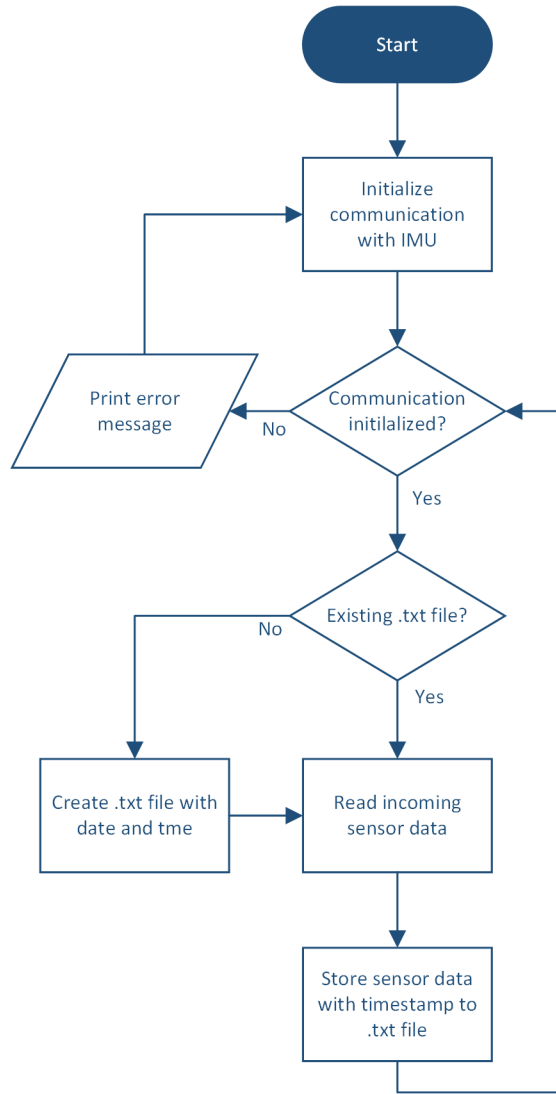


Figure 42: Flowchart: IMU

D3 - Flowchart: GPS

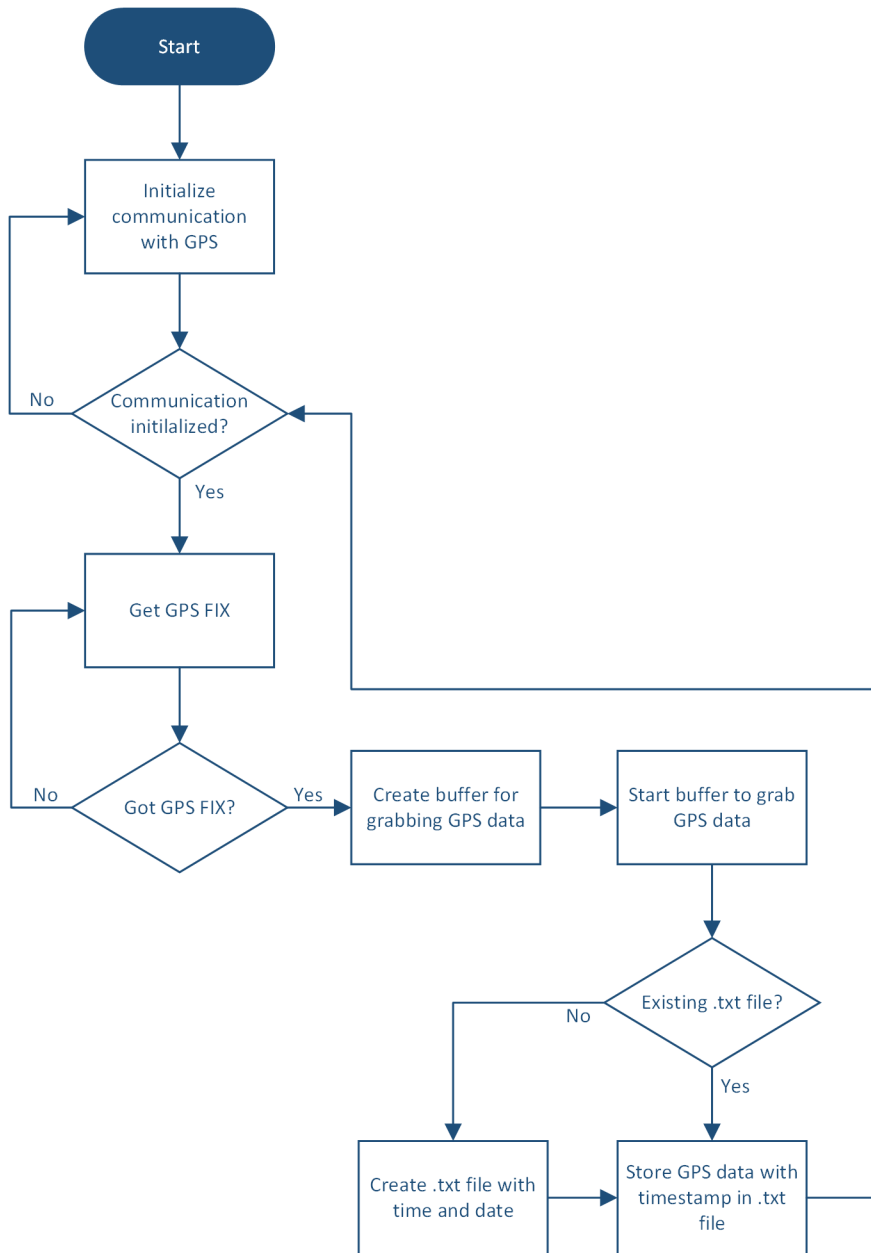


Figure 43: Flowchart: GPS

Appendix E - Codes

This appendix includes the names of the code files that are used in the final system along with code files that have expired and are no longer in use. These files can be found in the attached folder called "Codes".

E1 - CAN message receiver

- APH_CAN_Sniffer.ino
- due_can.h
- due_can.cpp
- DueCanLayer.h
- DueCanLayer.cpp

E2 - GPC connection

sendmessagetobq.py

E3 - IMU and GPS data

bmx160imu.py
adafruitgps.c
adafruitgps.py
sensorInputdata.c

E4 - Virtual sender

SendV3.ino

E5 - Virtual receiver

receiveV33.ino

E6 - Virtual CAN message transmitter

Sender.ino

E7 - CAN receiver Arduino uno

Receiver.ino

E8 - Storing IMU data to SD-card

Storing-IMU-data-to-SD.ino

E9 - Storing potmeter values to SD-card

Potmeter_value.ino

E10 - CanSniffer application

Folder 1: 01_canSniffer_arduino

Folder 2: 02_canSniffer_GUI

E11 - Boot Scripts

Folder "Boot scripts":

- canstart.desktop
- gpsstart.desktop
- imustart.desktop

E12 - Cloud Connection

Folder "GCP_connection": sendmessage.py

E13 - GPS data collection V1 and V2

Folder "GPS_data":

- gps_data.py (V1)
- gpsd_data.py (V2)

E14 - IMU data collection

Folder "IMU_data": imu_data.py

E15 - Serial data collection

Folder "Serial_data": canbus_data.py

E16 - Webcam timelapse

timelapse.sh

E17 - MATLAB 3D simulation

Folder "MATLAB: "
matlab3D-simulation.txt
inertial-measurement.m

Appendix F - Test Tables

Test: T1
Objective: Establish communication between Raspberry Pi and Arduino Due.
Method: Send messages from Arduino and print the messages on the Raspberry Pi.
Expected result: Managing to send messages from Arduino to Raspberry Pi.
Actual result: Managed to send messages from Arduino to Raspberry Pi.
Pass/Fail: Pass
Date: 03.03.2022

Test: T2
Objective: Establish CAN bus connection through MCP2515.
Method: Send CAN-messages from one Arduino Uno to another through MCP2515.
Expected result: Managing to send and receive CAN-messages.
Actual result: Managed to send and receive CAN-messages.
Pass/Fail: Pass
Date: 06.03.2022

Test: T3
Objective: Find suitable library for IMU.
Method: Test library examples to check if it works together with the IMU.
Expected result: The IMU generates data.
Actual result: The IMU generated data.
Pass/Fail: Pass
Date: 31.03.2022

Test: T4
Objective: Write IMU data to Raspberry Pi file.
Method: Send IMU data from Arduino to Raspberry Pi and write to a .csv/.txt file.
Expected result: Managing to store data in file.
Actual result: Managed to store data in file.
Pass/Fail: Pass
Date: 03.04.2022

Test: T5
Objective: Read the Adafruit Ultimate GPS position data.
Method: Connecting GPS to Arduino Mega and using Arduino IDE to run the code.
Expected result: Being able to read the GPS position data.
Actual result: Managed to read the GPS position data.
Pass/Fail: Pass
Date: 05.04.2022

Test: T6
Objective: Storing pot-meter data to SD-card while we wait for the Inertial Measurement Unit.
Method: Connected three pot-meters to Arduino and read the value from them and stored values onto SD-card.
Expected result: Read pot-meter and store on SD-card
Actual result: Was able to store pot-meter data to SD-card
Pass/Fail: Pass
Date: 28.03.2022

Test: T7
<p>Objective: Establish Virtual CAN communication between two Arduino Unos.</p> <p>Method: Connect two MCP2515 (CAN-Controllers) with two Arduino Unos and generate fake/virtual CAN-messages.</p>
<p>Expected result: Managing to send virtual CAN-messages from one Arduino to the other through MCP2515 and read through serial monitor.</p> <p>Actual result: Managed to send the CAN-messages via MCP2515 and read them through serial monitor.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 18.04.2022</p>

Test: T8
<p>Objective: Establish Virtual CAN communication between Arduino Uno and CAN bus card.</p> <p>Method: Connect MCP2515 to Arduino UNO and use CAN bus card as a receiver Generate CAN-messages on Arduino UNO and send to CAN bus card.</p>
<p>Expected result: Managing to read the messages sent from Arduino Uno on the CAN bus card.</p> <p>Actual result: Sent CAN-messages were received on the CAN bus card. .</p>
<p>Pass/Fail: Pass</p>
<p>Date: 22.04.2022</p>

Test: T9
<p>Objective: Read CAN-messages from excavator CAN bus using CAN bus card.</p> <p>Method: Connect the CAN bus card to the excavator CAN bus and turn on the engine to produce CAN-messages</p>
<p>Expected result: Managing to read the CAN-messages from the excavator using CAN bus card and print it to serial monitor.</p> <p>Actual result: Managed to read the CAN-messages from the excavator and print it to serial monitor.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 25.04.2022</p>

Test: T10
<p>Objective: Send both virtual standard and extended CAN-messages from via MCP2515 to CAN bus card.</p> <p>Method: Connect two Arduino UNOs (transmitters) to two MCP2515 and connect the two CAN-controllers to the CAN bus card (receiver).</p>
<p>Expected result: Managing to send virtual standard and extended CAN-messages from CAN0 and CAN1 and read through serial monitor on CAN bus card.</p> <p>Actual result: Managed to read both standard and extended CAN-messages on the CAN bus card.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 28.04.2022</p>

Test: T11
<p>Objective: Read standard CAN-messages with CAN bus card and store it on SD-card.</p> <p>Method: Connect CAN bus card to Raspberry Pi serially and store CAN-messages locally on SD-card.</p>
<p>Expected result: Managing to store CAN-messages to Raspberry Pi SD-card.</p> <p>Actual result: Managed to store CAN-messages to Raspberry Pi SD-card.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 02.05.2022</p>

Test: T12
<p>Objective: Read the extended CAN-messages from the excavator CAN bus with CAN bus card and store it on SD-card.</p> <p>Method: Connect CAN bus card to the excavator CAN bus and turn on the engine to produce CAN-messages. Store onto Raspberry Pi which is connected serially.</p>
<p>Expected result: Managing to read the extended CAN-messages from the excavator and storing locally on the Raspberry Pi.</p> <p>Actual result: Managed to store the CAN-messages locally on the Raspberry Pi.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 02.05.2022</p>

Test: T13
<p>Objective: Get GPS, IMU, and CAN bus data while the excavator is being operated.</p> <p>Method: Place the data collection unit inside the excavator and drive the excavator in order to receive data.</p>
<p>Expected result: Managing to collect and store CAN-messages, GPS and IMU data.</p> <p>Actual result: Managed to collect and store said data onto SD-card.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 02.05.2022</p>

Test: T14
<p>Objective: Find suitable library for MCP2515 (CAN-controller) which works with both Arduino Unos and Arduino Due.</p> <p>Method: Connect MCP2515 to Arduino Unos or Arduino Due and include the library on Arduino IDE to see if library works.</p>
<p>Expected result: Managing to send CAN-messages though MCP2515 with Arduino Unos without getting library related errors.</p> <p>Actual result: Managed to send CAN-messages as expected without getting library related errors.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 18.04.2022-02.05.2022</p>

Test: T15
Objective: Find suitable Dual-CAN library for Arduino Due.
Method: Connect two Arduino unos to Arduino DUE using MCP2515 and include the library on Arduino IDE for the Arduino Due side.
Expected result: Managing to receive virtual CAN-messages through MCP2515 with on the Arduino Due without getting library related errors.
Actual result: managed to receive said CAN-messages without library related errors occurring.
Pass/Fail: Pass
Date: 18.04.2022-02.05.2022

Test: T16
<p>Objective: Determine the frequencies of Arduino Due and MCP2515 (CAN controller).</p> <p>Method: Use Analog Discovery DIGILENT (ADD) and Waveforms application (Digital oscilloscope). Connect ADD to a PC and run the application. Follow the ADD manual description for pin connections.</p>
<p>Expected result: Managing to see the power range, frequencies, resistance range, baud-rate (speed) of CAN controllers and Arduino Due on digital oscilloscope.</p> <p>Actual result: The only correct results were received when the test was running on the ADD pin connected to itself. The test based on the expected result was not successful.</p>
<p>Pass/Fail: Failed</p>
<p>Date: 18.04.2022-02.05.2022</p>

Test: T17
<p>Objective: Test the fit of the 3D-printed plate for Arduino DUE</p> <p>Method: Physically mount the Arduino DUE to the plate using screws</p>
<p>Expected result: The holes in the plate align with the holes in the DUE, and are the right size for the screws used</p> <p>Actual result: All holes aligned and were the correct size, allowing the DUE to be mounted as planned</p>
<p>Pass/Fail: Pass</p>
<p>Date: 25.02.2022</p>

Test: T18
<p>Objective: Test the fit of the laser cut plate for Arduino Mega</p> <p>Method: Physically mount the Arduino Mega to the plate using screws</p>
<p>Expected result: The holes in the plate align with the holes in the Mega, and are the right size for the screws used</p> <p>Actual result: All holes aligned and were the correct size, but the screws available were too big for the Mega to be mounted as planned</p>
<p>Pass/Fail: Holes and plate: Pass Screws: Fail</p>
<p>Date: 04.04.2022</p>

Test: T19
<p>Objective: Test the fit of the laser cut plate for GPS and Raspberry Pi</p> <p>Method: Physically mount the GPS and Raspberry Pi to the plate using screws</p>
<p>Expected result: The holes in the plate align with the holes in the GPS and Raspberry Pi, and are the right size for the screws used</p> <p>Actual result: All holes aligned and where the correct size, but the screws available were either too big or small for the Mega to be mounted as planned</p>
<p>Pass/Fail: Holes and plate: Pass Screws for GPS: Pass Screws for Raspberry Pi: Fail</p>
<p>Date: 04.04.2022</p>

Test: T20
<p>Objective: Test the fit of the laser cut plate for IMU, GPS, and Arduino Uno</p> <p>Method: Physically mount the IMU, GPS, and Arduino Uno to the plate using screws</p>
<p>Expected result: The holes in the plate align with the holes in the IMU, GPS, and Uno, and are the right size for the screws used</p> <p>Actual result: All holes aligned and where the correct size, but the screws available were too big for the IMU and either too big or small for the Uno to be mounted as planned. For the IMU a secondary mounting was created that worked with the screws available</p>
<p>Pass/Fail: Holes and plate: Pass Screws for IMU: Pass Screws for GPS: Pass Screws for Uno: Fail</p>
<p>Date: 06.04.2022</p>

Test: T21
<p>Objective: Test the fit of the laser cut plate for Raspberry Pi</p> <p>Method: Physically mount the Raspberry Pi to the plate using screws</p>
<p>Expected result: The holes in the plate align with the holes in the Raspberry Pi and are the right size for the screws used</p> <p>Actual result: All holes aligned and where the correct size, but the screws available were either too big or small for the Raspberry Pi to be mounted as planned.</p>
<p>Pass/Fail: Holes and plate: Pass Screws for Raspberry Pi: Fail</p>
<p>Date: 06.04.2022</p>

Test: T22
<p>Objective: Test the fit of the laser cut spacing walls on the laser cut plates</p> <p>Method: Physically mount the spacing walls to the plates</p>
<p>Expected result: The notches cut in the spacing walls fit the holes cut in the plates, creating a tight fit holding the walls upright</p> <p>Actual result: The notches cut in the spacing walls fit the holes cut in the plate, but were cut a bit big, making the walls a bit loose, but still kept upright</p>
<p>Pass/Fail: Pass</p>
<p>Date: 06.04.2022</p>

Test: T23
Objective:
Test the fit of the laser cut plate for Raspberry Pi, GPS, and IMU
Method:
Physically mount the Raspberry Pi, GPS, and IMU to the plate using screws
Expected result:
The holes in the plate align with the holes in the Raspberry Pi, GPS, and IMU, and are the right size for the screws used.
Actual result:
All holes aligned and where the correct size, allowing the Raspberry Pi, GPS, and IMU to be mounted as planned
Pass/Fail:
Pass
Date:
22.04.2022

Test: T24
<p>Objective: Test the fit of the laser cut plate for Arduino DUE, IMU, and power bank</p>
<p>Method: Physically mount the Arduino DUE and IMU to the plate using screws, and fit the power bank in the designated cutout in the plate</p>
<p>Expected result: The holes in the plate align with the holes in the Arduino DUE and IMU and are the right size for the screws used. The power bank fit in the designated cutout in the plate</p>
<p>Actual result: All holes and cutouts aligned and were the correct size, allowing the Arduino DUE, IMU, and power bank to be mounted as planned</p>
<p>Pass/Fail: Pass</p>
<p>Date: 29.04.2022</p>

Test: T25
<p>Objective: Test the fit of the laser cut plate for Raspberry Pi, GPS, RTC, and power bank</p> <p>Method: Physically mount the Raspberry Pi, GPS, and RTC to the plate using screws, and fit the power bank in the designated cutout in the plate</p>
<p>Expected result: The holes in the plate align with the holes in the Raspberry Pi, GPS, and RTC, and are the right size for the screws used. The power bank fit in the designated cutout in the plate</p> <p>Actual result: All holes and cutouts aligned and were the correct size, except for the ones for the RTC. This meant that the 3d-model of the plate needed an update and a new plate had to be laser cut with the correct holes and cutouts</p>
<p>Pass/Fail: Holes and cutouts for Raspberry Pi, GPS, and power bank: Pass Holes and cutouts for RTC: Fail</p>
<p>Date: 29.04.2022</p>

Test: T26
Objective: Test the fit of the updated laser cut plate for Raspberry Pi, GPS, RTC, and power bank
Method: Physically mount the RTC to the plate using screws
Expected result: The holes in the plate align with the holes in the RTC, and are the right size for the screws used
Actual result: All holes and cutouts aligned and were the correct size
Pass/Fail: Pass
Date: 29.04.2022

Test: T27
Objective: Test the fit of the updated laser cut plate for Raspberry Pi, GPS, RTC, and power bank
Method: Physically mount the GPS to the plate using screws
Expected result: The new cutout made in the plate align with the antenna on the GPS, allowing the GPS to be mounted as planned
Actual result: The cutout in the plate aligned with the antenna on the GPS, allowing the GPS to be mounted as planned
Pass/Fail: Pass
Date: 02.05.2022

Test: T28
Objective: Test the fit of the new laser cut plate for Arduino DUE, IMU, and power bank
Method: Physically mount the Arduino DUE and the IMU to the plate using screws, and fit the power bank in the designated cutout in the plate
Expected result: The holes in the plate align with the holes in the Arduino DUE and IMU and are the right size for the screws used. The power bank fit in the designated cutout in the plate
Actual result: All holes and cutouts aligned and were the correct size, allowing the Arduino DUE, IMU, and power bank to be mounted as planned
Pass/Fail: Pass
Date: 06.05.2022

Test: T29
<p>Objective: Test the fit of the new laser cut plate for Raspberry Pi, GPS, and power bank</p>
<p>Method: Physically mount the Raspberry Pi and the GPS to the plate using screws, and fit the power bank in the designated cutout in the plate</p>
<p>Expected result: The holes in the plate align with the holes in the Raspberry Pi and GPS and are the right size for the screws used. The power bank fit in the designated cutout in the plate</p> <p>Actual result: All holes and cutouts aligned and were the correct size, allowing the Raspberry Pi, GPS, and power bank to be mounted as planned</p>
<p>Pass/Fail: Pass</p>
<p>Date: 06.05.2022</p>

Test: T30
<p>Objective: Test the fit of the laser cut walls for the cable compartment on the laser cut plates</p>
<p>Method: Physically mount the walls to the plates</p>
<p>Expected result: The notches cut in the walls fit the holes cut in the plates, creating a tight fit holding the walls upright</p> <p>Actual result: The notches cut in the walls fit the holes cut in the plate, creating a tight fit keeping the walls upright</p>
<p>Pass/Fail: Pass</p>
<p>Date: 06.05.2022</p>

Test: T31
<p>Objective: Establish communication between Google Cloud Platform and Raspberry Pi.</p> <p>Method: Register Raspberry Pi as a Iot device on GCP and store test data.</p>
<p>Expected result: Being able to send a message to GCP from Raspberry Pi and store to BigQuery table.</p> <p>Actual result: Managed to store test data to GCP.</p>
<p>Pass/Fail: Pass</p>
<p>Date: 13.05.2022</p>

Test: T32
<p>Objective: Selection of a programming language for IMU and GPS</p> <p>Method: GPS and IMU sensor interfacing with Raspberry Pi via serial communication</p>
<p>Expected result: Able to use either one of them python or C/C++</p> <p>Actual result: wrote GPS sensor into C on Raspberry Pi, but didn't manage with the IMU sensor</p>
<p>Pass/Fail: GPS Pass IMU Fail</p>
<p>Date: 19.04.2022</p>

Test: T33
Objective: Establish GPS communication with Raspberry Pi through serial.
Method: Collect GPS data with Arduino Uno and send to Raspberry Pi via serial.
Expected result: Receive GPS data from Arduino Uno on the Raspberry Pi using Python
Actual result: Received GPS data on Raspberry Pi
Pass/Fail: Pass
Date: 05.04.2022

Test: T34
Objective: Collect IMU data on Raspberry Pi through serial communication
Method: Collect IMU data on Arduino Uno and send to Raspberry Pi via USB-connection
Expected result: Send IMU data from Arduino Uno to Raspberry Pi using Python
Actual result: Received IMU data on Raspberry Pi
Pass/Fail: Pass
Date: 04.04.2022

Test: T35
Objective: Generate virtual CAN-messages
Method: Generate virtual CAN-messages using a Raspberry Pi, MCP2515 and Arduino Nano
Expected result: Being able to generate virtual CAN-messages through serial monitor
Actual result: CAN 0 was initialized and some data was received on serial monitor
Pass/Fail: Pass
Date: 24.02.2022

Test: T36
Objective: Store CAN bus data using Arduino Uno and SD-card shield.
Method: Connect Arduino Uno to SD-card shield and send test sample to SD-card.
Expected result: Being able to store test sample to SD-card.
Actual result: Communication between components was initiated.
Pass/Fail: Communication: Pass Sending virtual CAN data: Fail
Date: 08.03.2022

Test: T37
<p>Objective: Connect Raspberry Pi to a car through On Board Diagnostics.</p> <p>Method: Connect OBD-2 to Raspberry Pi via USB cable and connect to car.</p>
<p>Expected result: Establish communication and receive CAN data on screen</p> <p>Actual result: Communication was established, but didn't receive CAN messages</p>
<p>Pass/Fail: Fail</p>
<p>Date: 02.03.2022</p>

Test: T38
<p>Objective: Write GPS and IMU data to CSV or text file with time and date</p> <p>Method: GPS sensor and BMX160 sensors interfacing with Raspberry Pi and storing data to SD-card</p>
<p>Expected result: Managing to store GPS and IMU data to .txt file</p> <p>Actual result: Received GPS data. Did not receive IMU data.</p>
<p>Pass/Fail: GPS: Pass IMU: Fail</p>
<p>Date: 15.04.2022</p>

Test: T39
Objective: Use Raspberry Pi through remote desktop connection.
Method: Assigning PC and Raspberry Pi to same local network and connecting using SSH.
Expected result: Remote control Raspberry Pi from laptop.
Actual result: Remote control was established and was able to control Raspberry Pi from laptop.
Pass/Fail: Pass
Date: 23.04.2022

Test: T40
Objective: Run log file on busmaster
Method: Display log file using busmaster client
Expected result: Being able to run log files on busmaster and sort CAN data
Actual result: Didn't work.
Pass/Fail: Fail
Date: 27.01.2022

Test: T41
Objective: Visualize IMU data on Matlab
Method: Connect Arduino Uno and IMU to MATLAB through serial communication and visualize collected IMU data in Matlab.
Expected result: Visualize IMU data in 3D-format on Matlab
Actual result: Initialized serial communication with Arduino, but failed to detect IMU.
Pass/Fail: Fail
Date: 05.05.2022

Test: T42
Objective: Establishing GPS communication with Raspberry Pi through UART
Method: Connect GPS module to TX/RX pins, GND and 5V on the Raspberry Pi and check if communication is established.
Expected result: Managing to establish GPS communication.
Actual result: Managed to establish GPS communication and monitor GPS info.
Pass/Fail: Pass
Date: 01.04.2022

Test: T43
Objective: Establish IMU communication with Raspberry Pi through I2C.
Method: Connect IMU to SCL/SDA pins, GND and 3.3V on the Raspberry Pi and check if communication is established.
Expected result: Managing to establish IMU communication.
Actual result: Managed to establish IMU communication and monitor IMU data.
Pass/Fail: Pass
Date: 03.04.2022

Test: T44
Objective: Store GPS, IMU and CAN bus data to text files with timestamps.
Method: Write scripts and test if the data is stored in said text files.
Expected result: Managing to store data to text files in a structured manner.
Actual result: Managed to store data to text files with no problems.
Pass/Fail: Pass
Date: 24.04.2022

Test: T45
Objective: Have GPS, IMU and CAN bus scripts run at Raspberry Pi boot.
Method: Create desktop files redirecting to scripts and running them.
Expected result: Scripts should run on boot.
Actual result: Scripts run on boot.
Pass/Fail: Pass
Date: 25.04.2022

Test: T46
Objective: Syncing Raspberry Pi clock using RTC module
Method: Connect RTC module to Raspberry Pi through I2C and synchronize time.
Expected result: RTC module is able to synchronize Raspberry Pi clock to equal local time.
Actual result: RTC did manage to synchronize the Raspberry Pi clock successfully, but IMU stopped worked due to problems regarding I2C.
Pass/Fail: Fail
Date: 30.04.2022

Test: T47
Objective: Syncing Raspberry Pi clock using GPS's real-time capability
Method: Connect GPS to Raspberry Pi through Raspberry Pi and synchronize time.
Expected result: GPS is able to synchronize Raspberry Pi clock to equal local time.
Actual result: GPS works for synchronizing time.
Pass/Fail: Pass
Date: 02.05.2022

Test: T48
Objective: Use webcam to take pictures with a predetermined interval.
Method: Connecting webcam to Raspberry Pi through serial and create bash script in order to continuously take pictures with a set interval
Expected result: Managing to utilize webcam as a surveillance unit
Actual result: Webcam does take pictures with a set interval as expected.
Pass/Fail: Pass
Date: 05.05.2022

Appendix G - Risk Management Framework

Project risks:		Risks associated with the success of our project that is not covered in the other groups.				
ID	EVENT	Source	Asset	Consequences	Risk product	Strategy
R1	Risk event	What is the source of the risk	What asset is affected	Consequences of the event		
R1.1	Lack of communication	Group members, faculty staff, client.	Product, project.	Tasks are not finished in time. Decreased productivity.	LOW	Pre Risk: Use of facebook Messenger as the official communication channel for the team. messy channel doesn't divide topics into different channels. Post Risk: Be strict and frequent in sharing communication issues.
R1.2	Unfinished tasks	Group members not prioritizing the project. Unrealistic tasks, spending to much time on writing tests.	Product , grade.	Tasks are not finished in time. Decreased productivity.	HIGH	Pre Risk: Delegate less urgent tasks to the members. Post Risk: Converse why tasks where not finished.
R1.3	Failure to follow project methodology.		Product, grade.	Progression might suffer and we wont get to test the methodology fully.	LOW	Pre Risk: Have a clear definition of our project methodology and communicate uncertainties Post Risk: Set a meeting where everyone share their understanding of the methodology.
R1.4	Team members forgetting their parts while presenting	stress/anxiety, lack of preparation.	Grade.		LOW	Pre risk: Always have 2 people knowing each part of the presentation, start rehearsals in adequate time before presentations.
R1.5	Team members cant work as expected because of illness.	Quarantine, illness, loss of close relations	Product	Decreased productivity, higher workload on team member	MED	Pre risk: Set of alternative routines. Post risk: Use tools to cooperate online and communicate issues and solutions.
R1.6	Laser cutter failing.	Either user error or system error	Fixtures	Fixures not ready for presentation.	LOW	Pre risk: Cutting effect is not good or cut through, change old mechanical components. Post Risk: Outsource printing to Richard
R1.7	Equipment unavailable	Late delivery, unavailable key people.	Product	Some tasks might not be solvable.	HIGH	Pre risk: Acquire all necessary equipment for the project as soon as possible. Post risk: Work on other tasks/features, consider changing the product.
R1.8	Overheating components	To much load	Product	Product stops working until it has cooled down	HIGH	Pre risk: Acquire all necessary equipment to get turned off often. Post risk: Use a cooling fan to cool down the components.
R1.9	Wrong reading of data	To much vibrations	Product	construction site	HIGH	Pre risk: Acquire all necessary equipment to be fastened properly. Post risk: The equipment needs some kind of noise-reducing foam.

Product risks: Risks associated with how the product works						
ID	EVENT	Source	Asset	Consequences	Risk product	Strategy
R2	Risk event	What is the source of the risk	What asset is affected	Consequences of the event		
R2.1	Product does not meet the client's needs	Insufficient understanding of needs and the process to fulfill the clients needs	product, project	Unsatisfied customer	MED	Pre Risk: Frequently discuss customer needs with customer. Start testing early. Post Risk: Consider mitigation options
R2.2	Requirements not met	Bad project methodology or methodology not used correctly.	product, project	Product might be useless to our customer	LOW	Pre Risk: Revisit requirements frequently. Use good test methods. Start testing at an early stage.
R2.3	Low performing application	Our teams' technical skills	Product	Product might be useless, operators find it inconvenient to use.	LOW	Investigate issues, restructure and optimize.
R2.4	Insufficient photo documentation	Short time left to integrate the functionality and test alternative solutions.	Product	Not all measurements will have good photos	MED	Solution: Make pictures available to the operator as he measures.

Technical: Risks directly associated with hardware or software						
ID	EVENT	Source	Asset	Consequences	Risk product	Strategy
R3	Risk event	What is the source of the risk	What asset is affected	Consequences of the event		
R3.1	Our implementation requires more storage than available on the SD-Card	Using some quantity of high quality images for object.	Our implementation and its functionality	Less precise photoage	LOW	Pre Risk: We get approx 26 Gb of photoage in a month. Make an assessment on how much we need for overtime work.
R3.2	Application crash regularly	Lack of knowledge with Raspberry pi, not applying proven methods, insufficient testing.	Software	Little experience, inefficient work process.	MED	Pre Risk: We are using unit testing. Post Risk: Investigate and debug system.
R3.3	Bad choices of external libraries/ technology	Lacking knowledge and experience with the tools and technology	Our implementation and product.	Suboptimal solution.	LOW	Using the available resources and local knowledge.

R3.4	Hardware components malfunctioning	Production faults, user errors, wear.	Project, product	Project progression might suffer.	MED	Pre risk: order duplicate components. Post Risk: Order new components
R3.5	Electrical system	Wrong calculation	hardware	Little experience, inefficient work process.	MED	Pre risk: Acquire all necessary Information to a success project. Post risk: Work on Common knowledge for other tasks/features, consider changing the product.
R3.6	Prototype is not yet finished on time.	lack of knowledge	Technical	Project progression might suffer.	MED	Pre risk: Trying to solve the problem without help. Post Risk: Order new components
R3.7	Device will not power on before operation	Non function product	Technical	Device drains power when not in use. Battery is empty Error i Circuit board	HIGH	Mode power off of battery in detachment Physical
R3.7	Corrupted storage	Preconfigured settings will be lost	Mechanical	Faulty/Defect Storage Component from sources such as heat, water, dust, sand, cold.	LOW	Casing is sealed with gaskets at interfaces and with araldite on edges and corners
R3.8	Unsynchronized devices	Uneven received data	Product	Unable to Connect to server/wifi	HIGH	Pre risk: Acquire all necessary equipment for the project as soon as possible. Post risk: Work on other tasks/features, consider changing the product.

Appendix H - A Brief Introduction to Controller Area Network

Introductory concepts

Vehicle data collecting within a structure for smart solutions is not a new or unusual concept. The usage of microelectronics in construction machines dates back a few decades. As time passed, more manufacturers added electrical capabilities to construction machinery. Several control systems have been implemented to make it more user friendly, as well as better interfaces of full machine operating parameters that are manufacturer navigable. Because more electronic applications were being introduced to construction machinery, an efficient communication system was necessary [33].

The Controller Area Network (CAN) was created by Automotive Bosch in the 1980s to facilitate inter-ECU (Electronic Control Unit) communication. The CAN bus, which is linked to the ECUs, can broadcast messages to all nodes on the CAN bus to gather information about the sensors attached to the automobile, such as vehicle speed and engine coolant temperature [33].

The main limitation of today's CAN buses is that they can only transfer data at a rate of one megabit per second. Vehicles will, however, require a new version of the CAN bus known as CAN FD in the future, which will boost the transmission rate to 8 Mbit/s and the data packet size from 8 bytes to 64 bytes [34].

CAN bus in construction machinery

Computer-controlled heavy machinery requires a large number of signals [35]. Vehicle subsystems conduct a massive set of measures per second and require information about numerous parameters that these systems must share with one another. Communication is essential for coordinating, sequencing, and cohering operations for each control unit.

SAE J1939

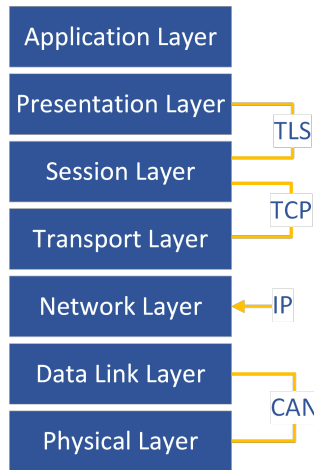


Figure 44: Protocols and procedures of the OSI model. The CAN protocol defines the OSI model’s bottom two levels.

In the early 1990s, the Standards of Automotive Engineering began work on a higher layer CAN protocol draft. The upper layer protocol is based on the Open Systems Interconnection (OSI) model’s seven layers (Fig.44). The CAN 2.0B architecture is used by SAE J1939 to transmit a 29-bit message identification. SAE J1939 has a pre-defined communication structure to allow many manufacturers to create systems that are identical. Parameter Group Number is used to create the SAE J1939 message (PGN) (SAE J1939 Standard Document). SAE J1939 communications are broadcast in hexadecimal format with specific bit timing and byte size to signal the message’s priority, the message identity, and the data contained inside it [36].

15:17:52:4238 Rx 1 0xCF00400 x 8 F3 87 87 26 19 00 F3 FF

Figure 45: Typical CAN message

Fig.45 depicts a typical CAN message (PGN F004) being sent across the bus. The message identification (F004) appears at the start of the message to let other ECUs on the bus know where the message is coming from and what data is contained within it. For example, F004 contains messages for Actual Percent Engine Torque and Engine Speed. A Suspect Parameter Number (SPN) is allocated to certain parameters within each parameter group (for example, SPN 185 inside PGN F004)[37].

Appendix I - CAN bus and Arduinos

This appendix includes the reasoning behind methods used for CAN-communication and sending of virtual CAN-messages.

Since the team did not have access to the excavator until Apr. 25, 2022, tests and preparation of technical hardware and software required different approaches. The team had to create virtual CAN messages to make sure that the Arduino Due was capable of receiving them, because one imagined the excavator would provide messages in similar ways.

I1 - MCP2515

MCP2515 is a stand-alone Controller Area Network (CAN) controller that implements the CAN specification. The controller helps with filtering packages and removing anything unnecessary traveling between CAN bus and the Arduino Due [38]. MCP2515 was used in order to send virtual CAN-messages to the Arduino Due.

I2 - Arduino UNO to Arduino Due

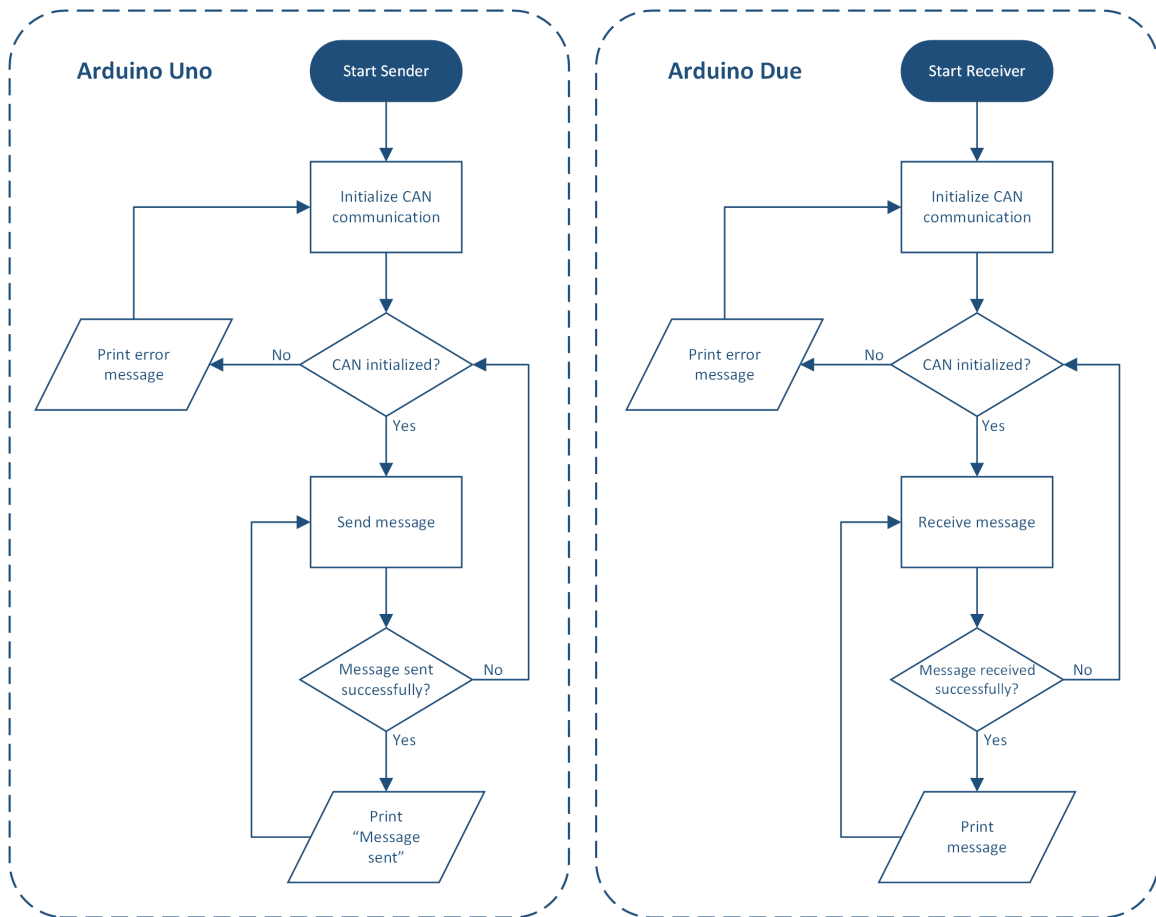


Figure 46: Flowchart of an Arduino UNO to an Arduino Due

First, the team tried reading virtual CAN-messages using an Arduino Uno as a sender. For the virtual sender code, see Appendix E4. To get an idea of how it should work, a flowchart was drawn. Figure 46 shows a flowchart of CAN communication between the Arduino Uno (sender) and Arduino Due (receiver).

The Arduino Uno was capable of sending virtual CAN-messages, while the Arduino Due would neither receive the virtual CAN-messages nor initialize the CAN bus communication. The team checked multiple times if the hardware was wired correctly or not.

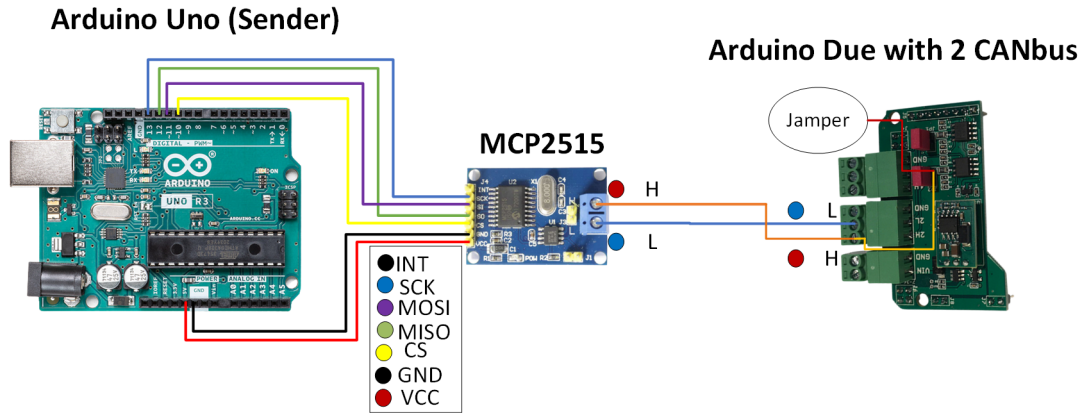


Figure 47: Schematic of Arduino Uno to Arduino Due wiring

As shown in Fig. 47, the Arduino Uno is connected to the Arduino Due using a MCP2515 (CAN controller) in order to establish CAN communication between the two devices.

In order to try solve this issue, another method had to be used. The Arduino Due was reokaced by a second Arduino Uno.

I3 - Arduino Uno1 to Arduino Uno2

Fig. 48 shows the connection between the two Arduino Unos, using a MCP2515 to establish communication.

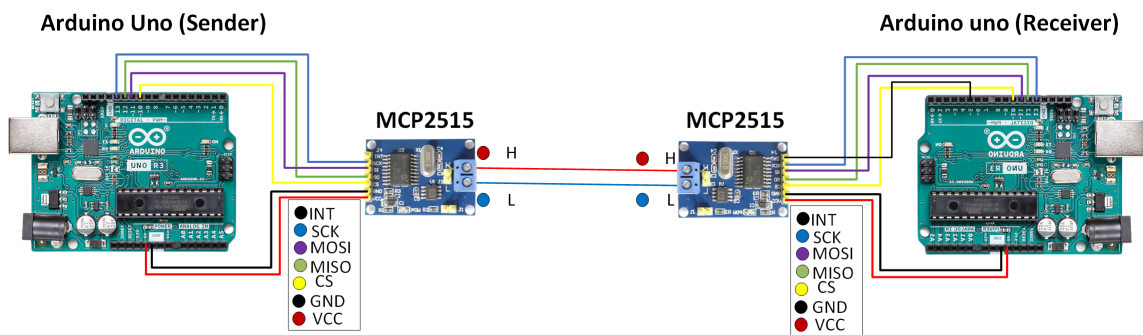


Figure 48: Schematic of an Arduino UNO1 to an Arduino UNO2

After replacing the Arduino Due with an Arduino Uno, the sender code from the Appendix E4 was used for one of the Arduino Unos, while the receiver code from Appendix E5 was used for the second Arduino Uno. This resulted the Arduino Uno receiver being able to initialize communication to the sender and receive CAN-messages. The team got outputs on both the sender and receiver side as seen in Fig. 49, top side showing the sender output, while bottom side is showing the receiver output.

```

COM7
02:03:38.651 -> In loop
02:03:39.656 -> In loop
02:03:40.670 -> In loop
02:03:41.665 -> In loop
02:03:42.652 -> In loop
Sender output

COM7
02:01:03.562 -> -----
02:01:03.562 -> Data from ID: 0x43
02:01:03.562 -> 1      1      2      3      4      5      6      7
02:01:04.594 -> -----
02:01:04.594 -> Data from ID: 0x43
02:01:04.594 -> 1      1      2      3      4      5      6      7
02:01:05.599 -> -----
02:01:05.599 -> Data from ID: 0x43
02:01:05.599 -> 1      1      2      3      4      5      6      7
    
```

Figure 49: Output of virtual sender and receiver

There was lots of confusion as to why the CAN-communication worked when replacing the Arduino Due with an Arduino Uno. After some troubleshooting, the team found out that the CAN bus interface on the Arduino Due had to use a 120-ohm resistance or jumper as called in the schematics. For more information about the troubleshooting, see Sec. 4.2.2.2. One had to be aware to not use a 120-ohm resistance when testing with the excavator running. As a result of solving the issue, new tests were performed with the Arduino Due sending two types of CAN-messages, standard and extended.

There are two types of CAN message identifiers, standard format is an 11-bit identifier which allows a total of $2^{11} = 2048$ different messages. An extended format on the other hand is 29-bit identifier which allows for $2^{29} = 536$ +million messages [14, p.14, p.27, p.184].

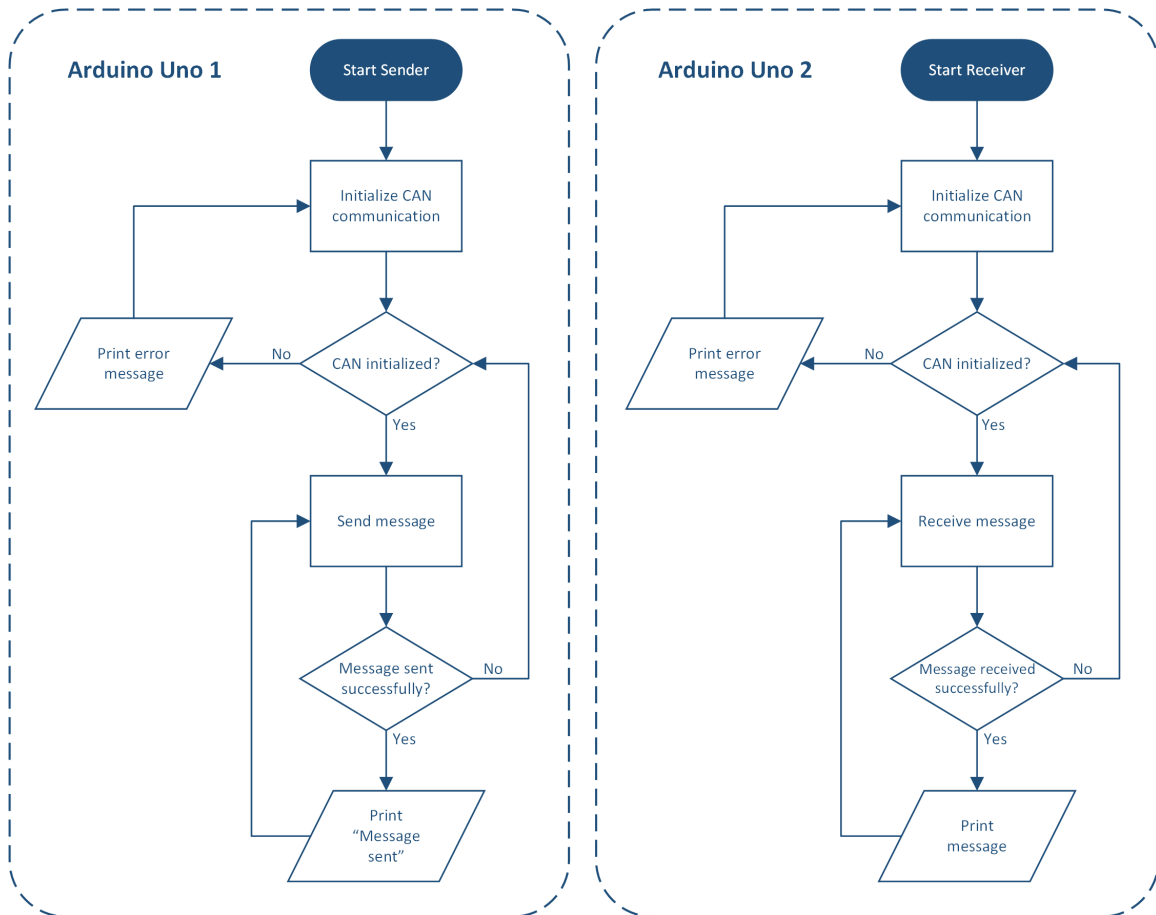


Figure 50: Flowchart of an Arduino UNO1 to an Arduino UNO2

Fig. 50 showcases the CAN-communication between the two Arduino Unos, one being the sender and one being the receiver.

To send both CAN-message formats, the team had to utilize two Arduino Unos, two MCP2515 (CAN controllers) and an Arduino Due with CAN bus interface.

I4 - Arduino UNO1 and Arduino UNO2 to Arduino Due

Fig. 51 shows how the two Arduino Unos are wired to two MCP2515s which are connected to the Arduino Due.

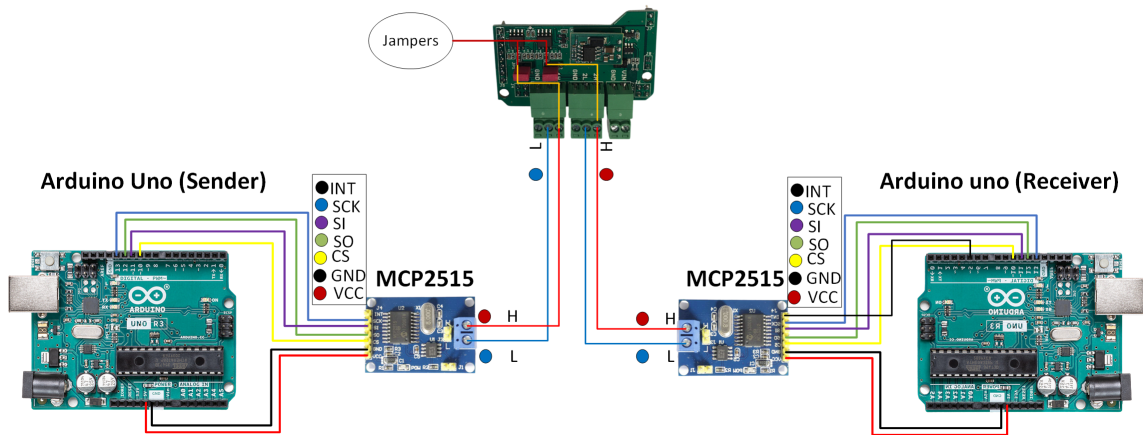


Figure 51: Schematic of two Arduino UNOs and an Arduino Due

To get an idea how the code should work, a flowchart was drawn. The flowchart displayed in Fig. 52 shows the CAN communication between the two Arduino Unos used as senders to an Arduino Due used as a receiver. The two CAN controllers were used to send standard and extended CAN messages. One can find more information in Sec. 4.2.2 about these and about CAN sniffer used to read the CAN messages.

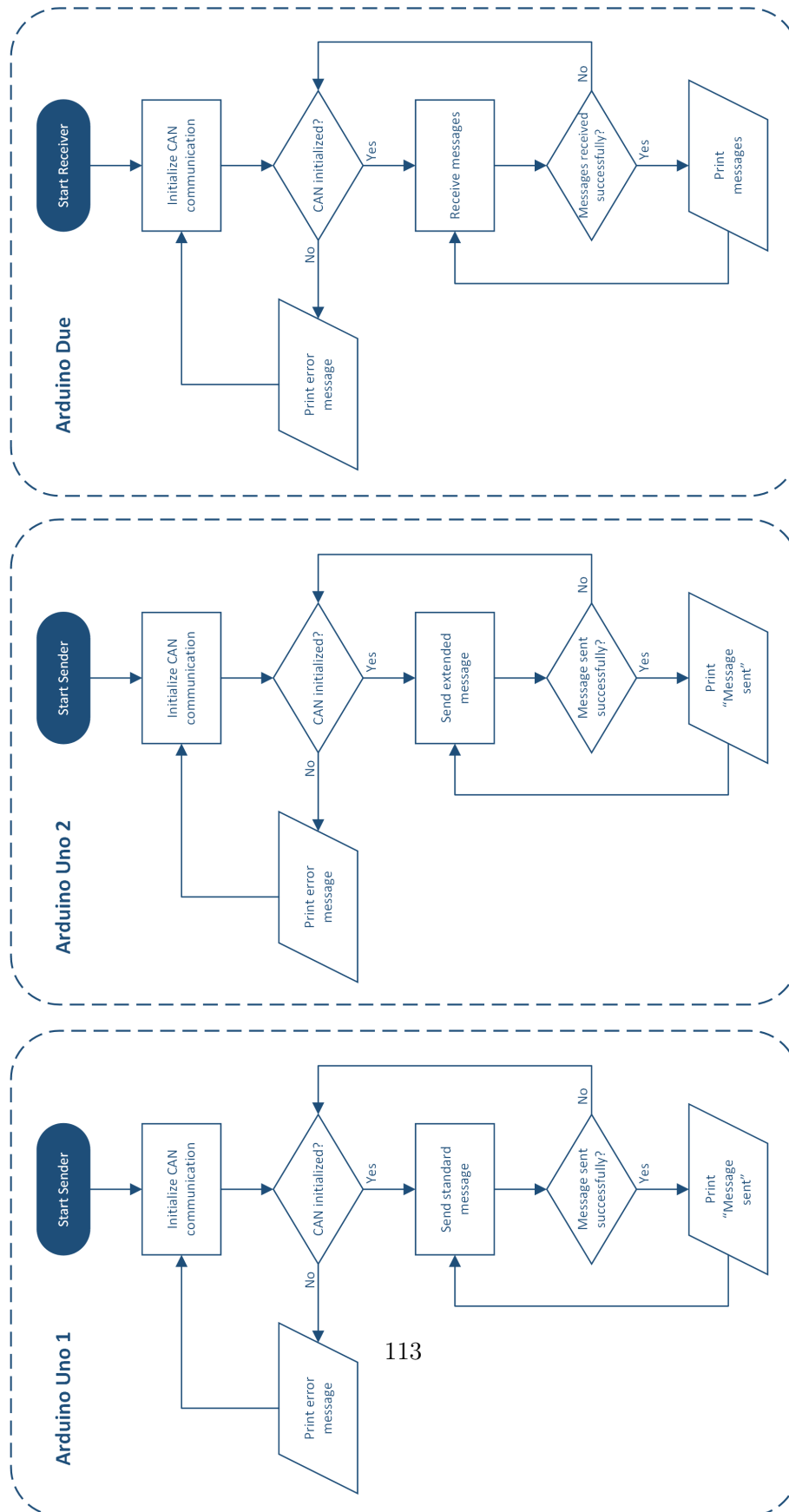


Figure 52: Flowchart of an Arduino UNO1 and an Arduino UNO2 to an Arduino Due

Appendix J - Flow Chart for CanSniffer code

The flow chart in Fig.53 gives a simple overview of how the code from Appendix E1 works.

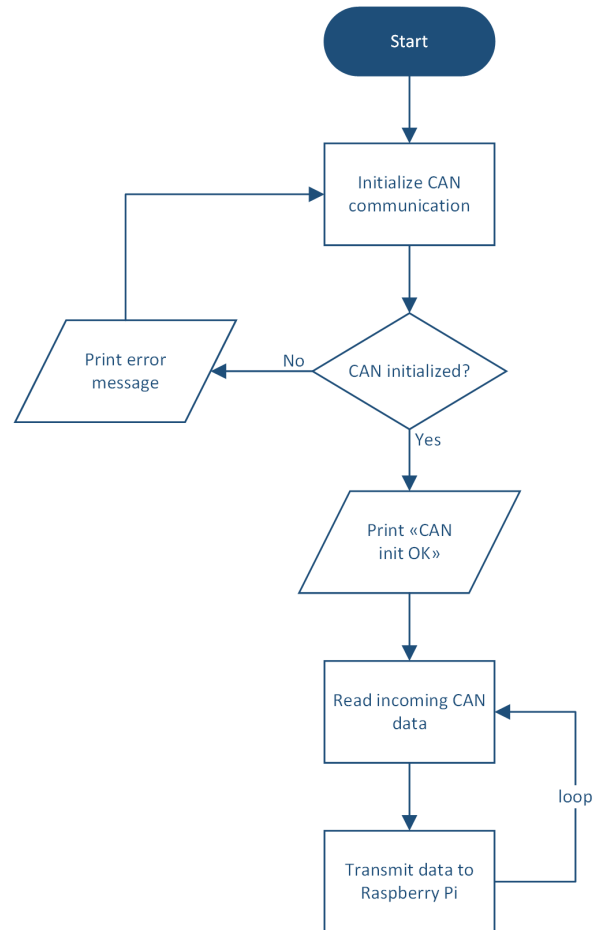


Figure 53: Flowchart: CanSniffer

Appendix K - Store Potmeter Values to SD-card

Appendix E9 is the code that was used before the Inertial Measurement Unit arrived to read values from a potmeter and store the data onto a SD-card. It was just a test to obtain a better understanding of how to read and store data before the team received the IMU.

K1 - PotMeter schematic

The figure below shows how potmeter and SD card are wired to an Arduino Uno. The color codes demonstrate which wire is connected to which pin.

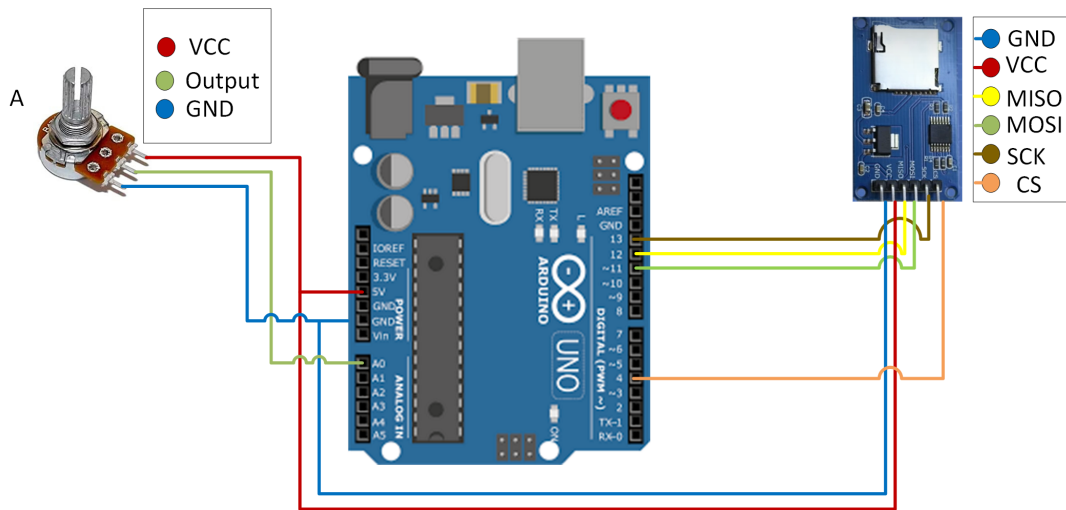


Figure 54: Schematic of potmeter and SD-card connection

Appendix L - IMU data storing to SD-card using Arduino

In the beginning, the plan was to collect the IMU data using an Arduino Uno and transfer the information to the Raspberry Pi and store the data onto a SD card. The code in Appendix E8 was used to read the IMU data and store it to a SD card by using an Arduino Uno [39].

L1 - IMU Flowchart

The following flowchart as seen in Fig.55 demonstrates how the code works.

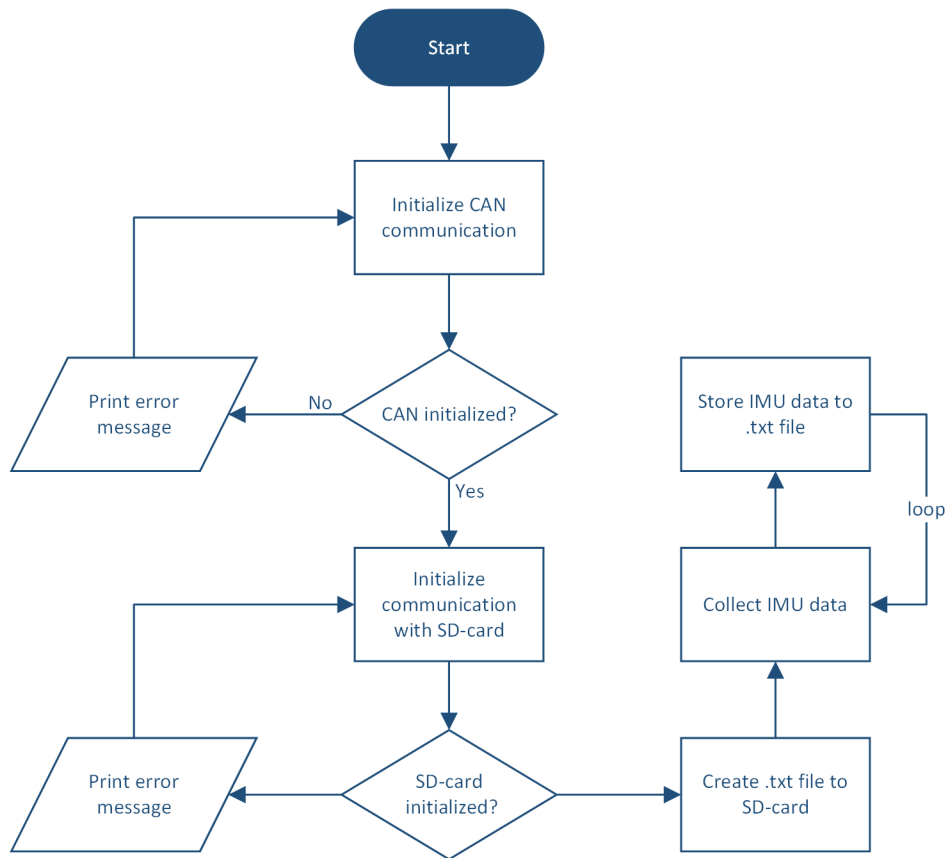


Figure 55: Flow chart of storing IMU data to SD-card using Arduino Uno

L2 - IMU Schematic

Fig. 56 demonstrates how the IMU and SD-card are wired to the Arduino Uno. The color codes demonstrates which wire is connected to which pin.

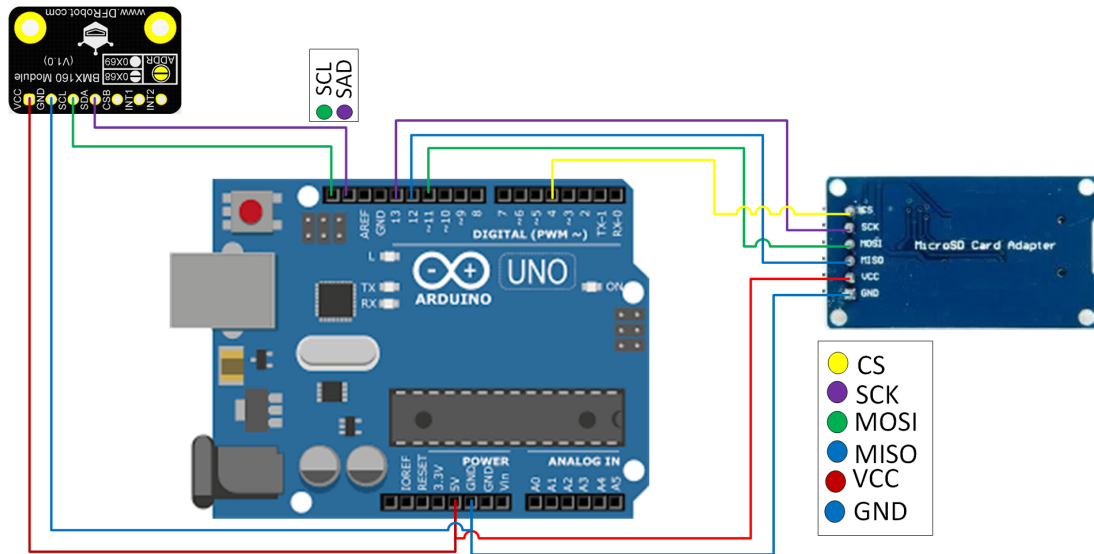


Figure 56: IMU and SD card schematic

Appendix M - CanSniffer Application

CanSniffer GUI and its features

The team decided to develop an application that can read and filter data packets in real-time in order to get an overview of incoming traffic and decode unknown messages. After some research on the internet, we came across a GitHub link that did exactly what we needed [40]. When we downloaded the code, I faced a library problem, which I resolved, and used the time to learn about the application and how it functions.

CanSniffer is a program that reads CAN-messages in real-time. The purpose of this application is to simplify the decoding of CAN-messages by providing an overview of which packets contain new data. The most important part of the app is the filter section. One has the option to group the packets based on their IDs, as well as highlight the changes in the payload for each packet identifier. The old packets can be either shown or hidden if wanted as well as specific packet identifiers.

Since the team had minimal access to an excavator for tests, the code in Appendix E10 was used to generate random virtual CAN-packets and sending them to the CanSniffer-application for testing purposes. To start sniffing with the Arduino the preloaded code should be connected, once the Arduino is in random mode the data packet will be sent by the Arduino. Fig.57 demonstrates how the application looks like. It displays how many different packets are available, how many messages are contained in each packet, and which packet received new data. Purple indicates which packet received new data.

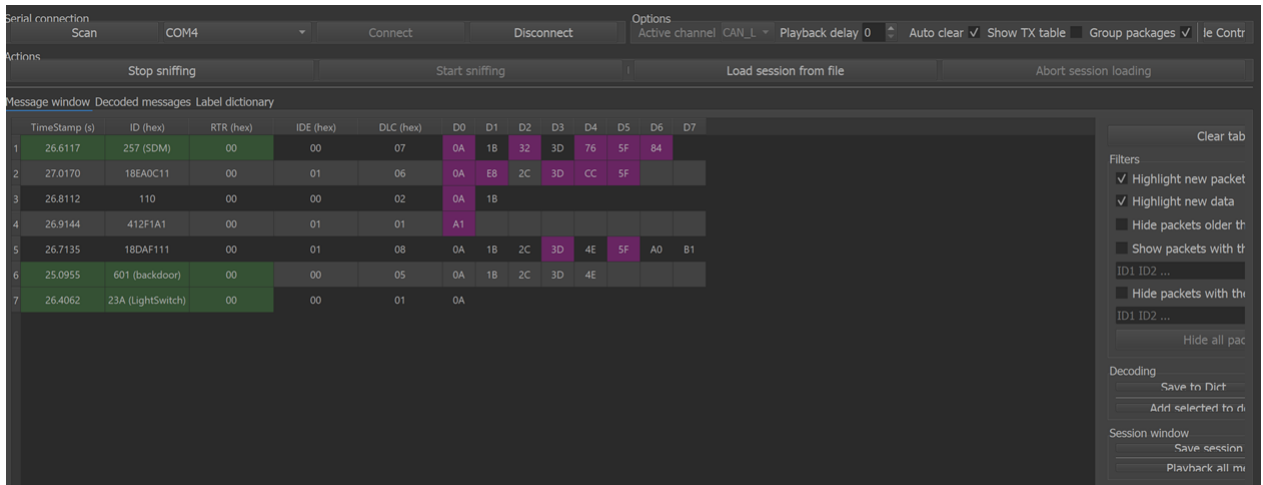


Figure 57: CanSniffer application

The application also provides the option of saving a session as a file and uploading it back later for further review. As seen in Fig.58, one can also rename the message IDs to give them more meaningful names after being decoded. These decoded messages will be stored in a separate category called “label dictionary” which provides an overview of which messages are decoded.

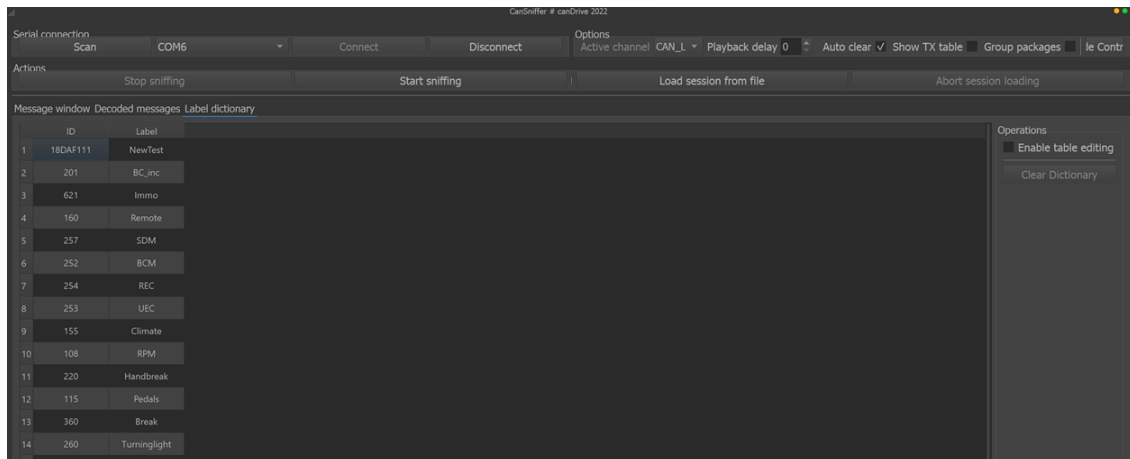


Figure 58: Decoded messages

The application was designed primarily for the decoding of CAN-messages, but the project description changed from decoding to collecting and storing data, so this application is no longer in use.

Appendix N - Electrical System

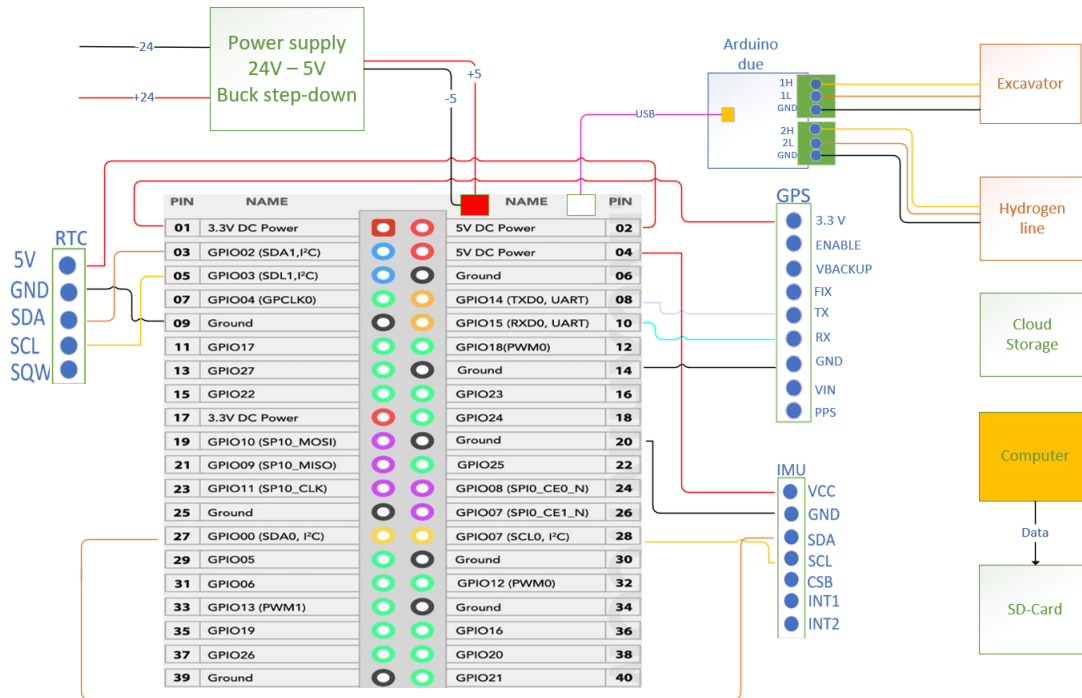


Figure 59: Wiring diagram

One can see an overview of the wiring of the hardware system in Fig.59. The hardware system consists of a Raspberry Pi as the unit which will receive data from various sensors. The Arduino Due collects data from the excavator and hydrogen driveline CAN bus and sends it to the Raspberry Pi together with IMU and GPS data. An Inertial Measurement Unit (IMU) measures the three different angles of the excavator. A voltage converter or power supply is used to lower the voltage so that the system can handle the voltage since most of the components function with 3.3 V and 5 V. The real-time clock (RTC) will provide correct time for synchronization.

Appendix O - IMU 3D Simulation

In our system we use (IMU) "Inertial Measurement Unit" to read the data on how the excavator orientation, and we want to show the different angles axis when the excavator moves, we were trying to show it by using Matlab simulation, but we faced some problem with the program and could not make it to work by using Matlab simulation,we've been doing it for over two weeks. Figure 60 shows the orientation of the IMU.

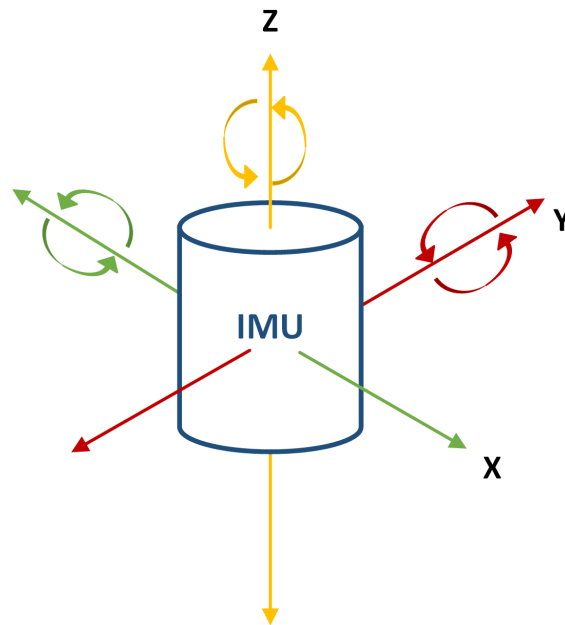
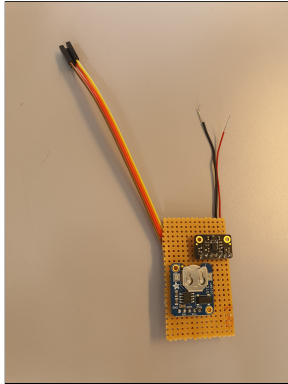


Figure 60: IMU axis

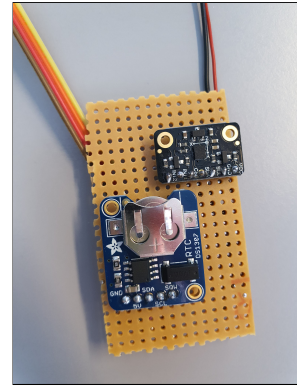
Appendix P - Soldering Hardware Components



Front side



Back side



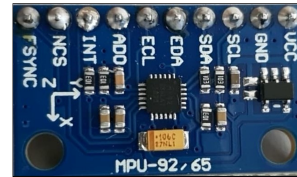
IMU and RTC soldered in veroboard



GPS front



GPS back



IMU

Appendix Q - Power Consumption

As shown in figure 61 the power consumption of the different components have been calculated. Both research and calculations were made for components in order to find the difference between the current, voltage, and power of each of the components. This is presented in figure 61.

	Raspberry Pi	Arduino Uno	Arduino Due	GPS	IMU
Current	1.5-3.0A	3.3V = 11.45mA 9V = 27.9mA	I/O = 130mA 5V = 800mA	Min. 48mA Max. 53mA	1585 μ A Full operation mode
Voltage	5V	5V	External power supply: 7-36 VDC Can also use USB power cable.	Min. 3.3V Max. 5.0V	3.3-5V
Power	Raspberry Pi 4 consumes idle 3.8-4.0W With 1: 4.5W With 2: 5.0W With 3: 5.4-5.5W With 4: 6.0W	3.3V = 0.037W 5V = 0.057W 9V = 0.10W	5V => 4W	Min. effect => 0.15W Max. Effect => 0.26W	3.3V => 5.23mW 5V => 7.92mW

Figure 61: Power consumption

Appendix R - SD-card Calculation

SD-card data storage

	FPS	Resolution	Days	Hours	SD-card size
Video	10	960p	22	10	25 GB

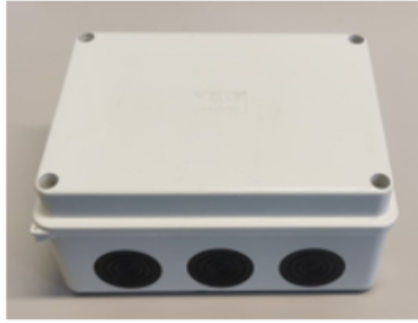
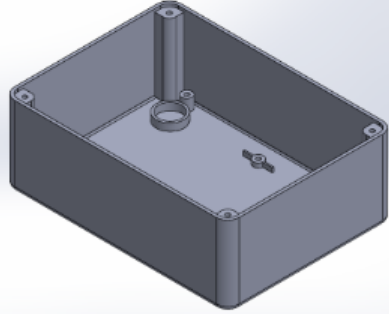
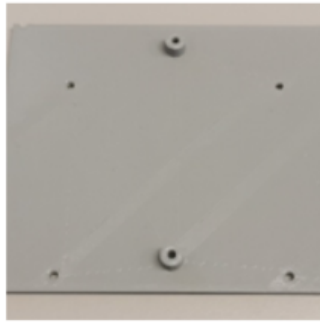
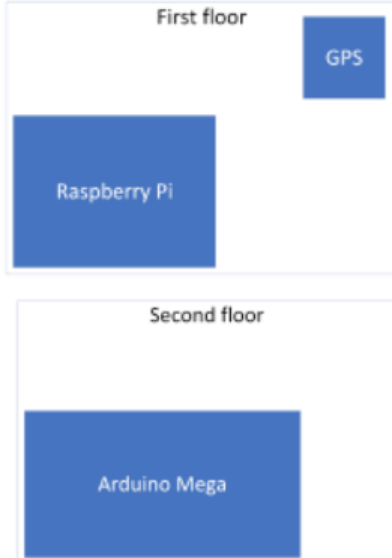
	Bits per character	Number of characters	Days	Hours	SD-card size
Data	7	xx	22	10	2 GB

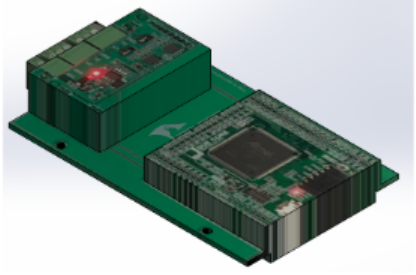
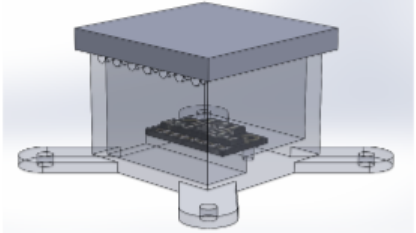


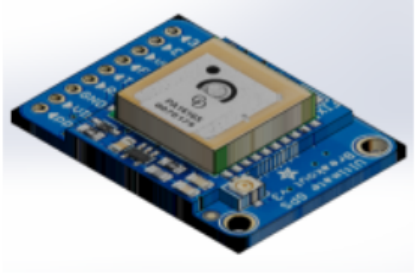
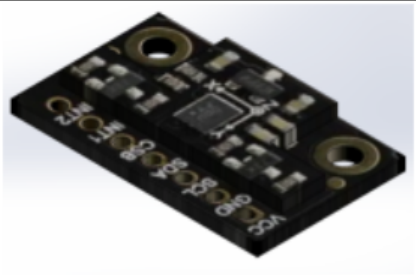
Figure 62: SD-card size calculation

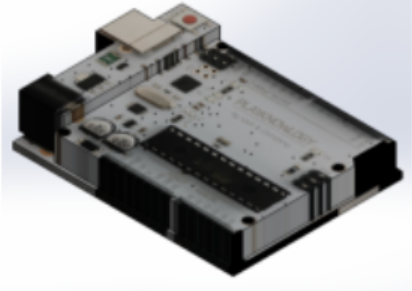
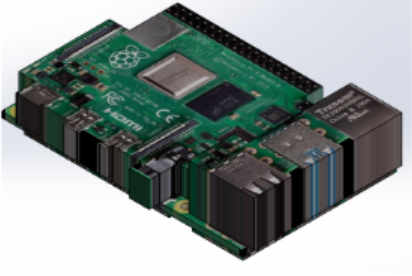

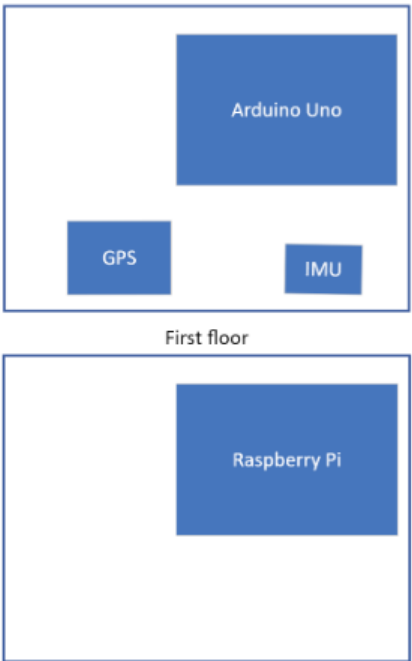
An SD card, is a movable memory card get used to read and write large amount of data in a large different areas like in many kind of electronics, and more [41]. In the system it is used to store data received from the CAN bus, IMU and GPS. The data received will be stored in text files. Each character consists of 7 bits and the amount of characters varies. The data will be saved every working day for each month, assuming a normal working day is 10 hours. After calculations 2GB seemed like enough storage for storing text files.

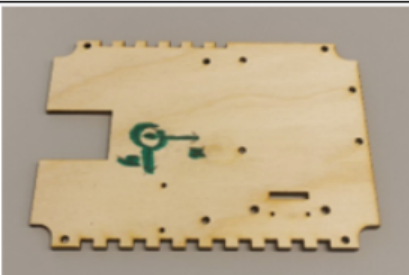

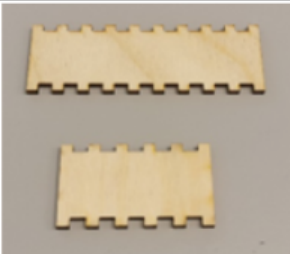

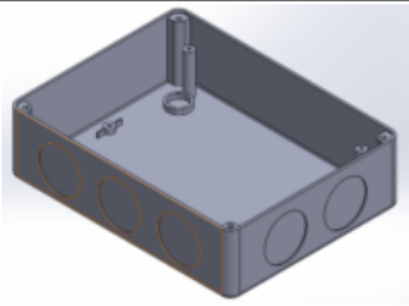
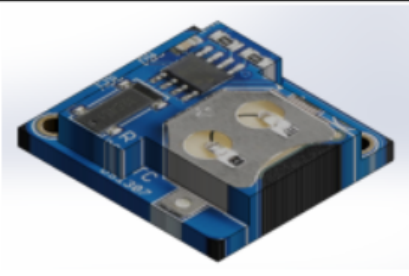
A camera is also utilized in the system. The calculations were made based on video recording 62, the video quality is 10 frame per second with 960p resolution, the same working days as the IMU and GPS which is 22 days, 10 working hours. after the calculation it ended up with 25 GB for video recording data. We end up with 2 GB for text files and 25 GB for video recording data, that end up with 27 GB, there is no 27 GB available in the market so the closest one available is 32 GB, which we will be using in our system.

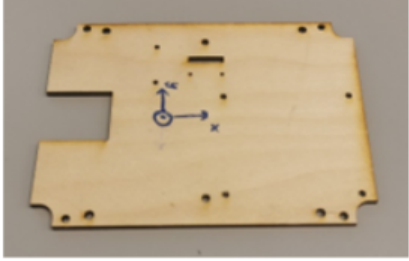
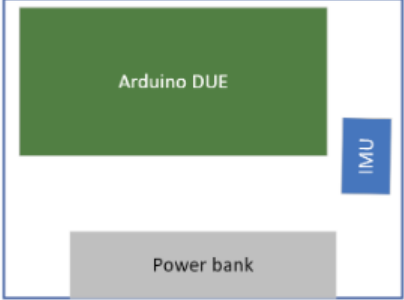
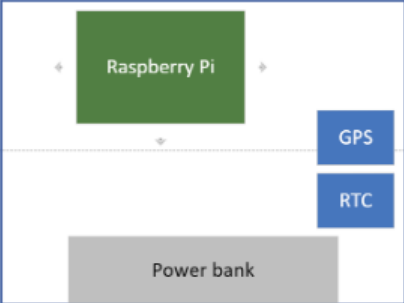
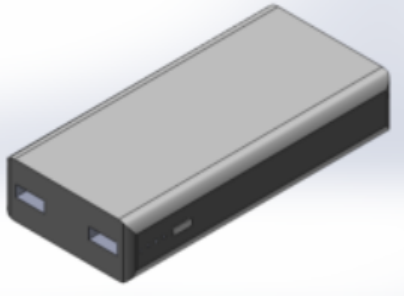
Appendix S - Changelog for Physical System


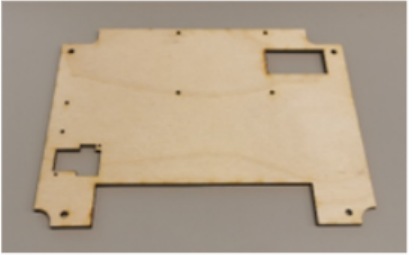
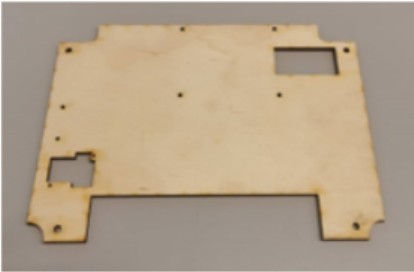

Dato:	Activity:	Version:	
09.02.22	Bought small junction box	v1	
10.02.22	Created 3D-model of small junction box	v1	
22.02.22	Created plate for Arduino DUE	Previous project	
25.02.22	3D-printed plate for Arduino DUE	Previous project	
01.03.22	Removed Arduino DUE and plate	Previous project	
04.04.22	Found placement for components 1. floor Raspberry Pi and GPS 2. floor Arduino Mega	v1	

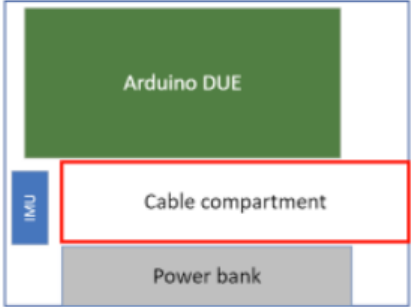
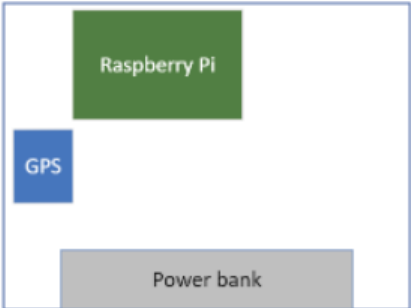
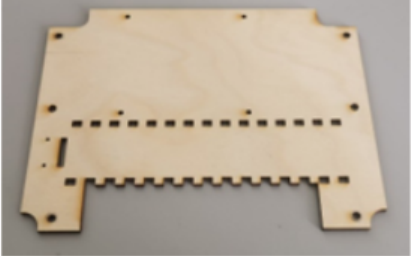


04.04.22	Created 3D-model of Arduino DUE	v1	
04.04.22	Created plate for Arduino mega	v1	
04.04.22	Created plate for GPS and Raspberry Pi	v1	
04.04.22	3D-modeled separate box for IMU	v1	
04.04.22	Laser cut plate for Arduino Mega	v1	
04.04.22	Laser cut plate for GPS and Raspberry Pi	v1	
05.04.22	Changed from Arduino Mega to Arduino Uno	v1.1	
05.04.22	Created 3D-model of Adafruit Ultimate GPS	v1.1	
05.04.22	Created 3D-model of Sen0373 IMU	v1.1	

05.04.22	Created 3D-model of Arduino Uno	v1.1	
06.04.22	Created 3D-model of Raspberry Pi	v1.1	
06.04.22	Created 3D-model of Arduino Mega	v1.1	
06.05.22	Moved GPS and IMU to 2. floor in box	v1.2	
06.04.22	Created hole for cables	v1.2	
06.04.22	Created 3D-model of walls for spacing between floors	v1.2	
06.04.22	Created notches for walls in 3D-model of plates	v1.2	
06.04.22	Updated mounting holes for Raspberry Pi and Arduino Uno	v1.2	
06.04.22	Created alternative way for mounting IMU because of lack of screws	v1.2	
06.04.22	Floor layout	v1.2	<p>Second floor</p>  <p>First floor</p>

06.04.22	Laser cut plate for Arduino Uno, GPS, and IMU	v1.2	
06.04.22	Laser cut plate for Raspberry Pi	v1.2	
06.04.22	Laser cut walls for spacing between floors	v1.2	
06.04.22	Improvised method of mounting components, using barbecue skewers and tape because of lack of screws	v1.2	
20.04.22	Removed Arduino Uno	v1.3	
21.04.22	Bought bigger junction box	v2	
21.04.22	Created 3D-model of big junction box	v2	
22.04.22	Created 3D-model of RTC	v2	
22.04.22	Moved GPS and IMU down to first floor	v1.4	
22.04.22	Removed second floor	v1.4	

22.04.22	Removed alternative way of mounting IMU	v1.4	
22.04.22	Removed notches in plate for walls and walls	v1.4	
22.04.22	Floor layout for simplified system	v1.4	
22.04.22	Laser cut plate for Raspberry Pi, GPS, and IMU	v1.4	
25.04.22	Found placement for components in new box 1. floor Arduino DUE and IMU 2. floor Raspberry Pi, GPS, and RTC Hole in both floors for power bank along the side of the box	v2	<p>First floor</p>  <p>Second floor</p> 
25.04.22	Created 3D-model of power bank	v2	
26.04.22	Found screws and metal standoff used as spacer	v2	
26.04.22	Created mounting holes for the Arduino DUE	v2	
26.04.22	Created mounting holes for Raspberry Pi	v2	
26.04.22	Created mounting for GPS	v2	
28.04.22	Created mounting for IMU	v2	
28.04.22	Created mounting for RTC	v2	
29.04.22	Created cut out in plate for access to SD-card in Raspberry Pi	v2.1	
29.04.22	Created cut out inn plates for Power bank	v2.1	
29.04.22	Created plate to create space for cables between box and lid	v2.1	

29.04.22	Laser cut plate for Arduino DUE and IMU	v2.1	
29.04.22	Laser cut plate for Raspberry Pi, GPS, and RTC	v2.1	
29.04.22	Updated mounting cut out and mounting holes for RTC	v2.2	
29.04.22	Laser cut plate with updated cut out and mounting holes for RTC	v2.2	
02.05.22	Created cut out for GPS antenna	v2.3	
02.05.22	Laser cut plate with updated cut out for GPS	v2.3	
06.05.22	Removed RTC	v2.4	
06.05.22	Removed cut out and mounting holes for RTC	v2.4	
06.05.22	Moved IMU and mounting holes to opposite side of box	v2.4	
06.05.22	Flipped power bank 180 degrees so cables extend on opposite end	v2.4	
06.05.22	Moved GPS, cut out, and mounting holes to opposite end of box	v2.4	
06.05.22	Updated mounting holes for Arduino DUE	v2.4	
06.05.22	Created walls to form a compartment for cables	v2.4	
06.05.22	Created notches for mounting walls	v2.4	
06.05.22	Created cut out in 2. floor for access to compartments for cables	v2.4	

06.05.22	Final floor layout	v2.4	<p style="text-align: center;">First floor</p>  <p style="text-align: center;">Second floor</p> 
06.05.22	Laser cut plate for Arduino DUE, IMU and cable compartment	v2.4	
06.05.22	Laser cut plate for Raspberry Pi and GPS, with cut out for access to cable compartment	v2.4	
06.05.22	Laser cut walls for cable compartment	v2.4	

Appendix T - Requirements

Rank	Requirement
A	Design and implement the ICU to work as a data recorder
	Find suitable sensors in form of an IMU (Inertial Measurement Unit), which shall include: <ul style="list-style-type: none"> • 3-axis gyroscope • 3-axis accelerometer • 3-axis compass
	The sensors shall be fully encapsulated with connections in place and a mounting bracket to ensure correct sensor data
	Mount and handle data of a GPS. <ul style="list-style-type: none"> • Find suitable sensors.
	The recorder shall record and store all CAN-messages from both the excavator and hydrogen driveline in real time.
	The recorder shall record and store all sensor data in real time
	CAN-messages shall be stored together with the sensor data to keep the data synchronized.
	The recorder shall store the data locally.
	The recorder shall be fully encapsulated with connections in place and a mounting bracket to make sure it is mounted correctly proportional to the IMU's axes.
B	The recorder should be able to transfer the data to a cloud storage.
	The recorder should also be able to record video.
C	Create features to play a recording in real time
	The recorder should be able to stream the data to a display system over Wi-Fi or be used for a local demo, preferably combined with video playback.

Figure 63: Requirements

Name	Number	Voltage	Bitrate	Function	Range	Sensitivity	Lowpassfilter Bandwith	Dimensions	Price
Adafruit 9-DOF Absolute Orientation	BNO055	2.4-3.6V	14 16 13/13/15	3-Axis Accelerometer 3-Axis Gyroscope 3-Axis Magnetometer	± 2-16 g 125-2000deg/sec ± 1300 (x,y) ± 2500 (z) µT	1 LSB/g 16.0 900 LSB/°/s rad/s 0.3 µT/LSB	8 - 1k Hz 12 - 523 Hz	20mm x 27mm x 4mm	221 nok
Adafruit TDK InvenSense ICM-20948	ICM20948	1.8- 5V	16 16 16	3-Axis Accelerometer 3-Axis Gyroscope 3-Axis Magnetometer	± 2-16 g 250-2000deg/sec ± 4900 µT	2 LSB/g 16.4 LSB/dps 0.15 µT/LSB	562-9k Hz 562-4.5k Hz 100Hz	25.7mm x 17.7mm x 4.6mm	133 nok
Adafruit LSM9DS1	LSM9DS1	1.9-3.6V	16 16 16?	3-Axis Accelerometer 3-Axis Gyroscope 3-Axis Magnetometer	± 2-16 g (ikke 6g) 245-2000deg/sec	2 LSB/g			355 nok
Adafruit 9-DOF Orientation	BNO085	3-5V	12 16	3-Axis Accelerometer 3-Axis Gyroscope	± 8 g ± 2000deg/sec		90-1kHz	25.6mm x 22.7mm x 4.6mm	221 nok
HiLetgo MPU9250 9-Axis 9 DOF	MPU9250	3-5V	16 16 16	3-Axis Accelerometer 3-Axis Gyroscope 3-Axis Magnetometer	± 2-16 g up to ± 2000 deg/sec ± 4800 µT	up to 16,384 LSB/g up to 131 LSB/deg/sec 0.6 µT/LSB		25.5mm x 15.4mm x 3mm	126.75 nok
GY-85 BMP085 Sensor	BMP085	3-5V		3-Axis Magnetometer Angle Barometer	± 4900 µT (X, Z-axis: ±180°, Y ±90°) (1 axis)			22mm x 17mm	
HWT901B-TTL MPU9250 9-axis	MPU9250	5-36V	32?	3-Axis Accelerometer 3-Axis Gyroscope	± 16 g up to ± 2000 deg/sec		0.2-200 Hz	55mm x 36.8mm x 24mm	697.6 nok
SEN0373		1.7-3.6	16 16	3-Axis Accelerometer 3-Axis Gyroscope	± 2-16 g (ikke 6g) 125-2000deg/sec	2048-16384 LSB/g 16.4-131 LSB/°/s	80-200 Hz 74.6-200 Hz	2.5mm x 3mm x 0.95mm	124 nok
			?	Geomagnetic sensor	± 1150 (x,y) ± 2500 (z) µT				

Figure 64: Comparison of IMU [42, 43, 44, 45, 46, 47, 48, 49, 50, 51]

Name	Chipset	Voltage	Power Consumption	Channels	Operational Limits	Sensitivity	Accuracy	Update rate	Dimensions	Price
Seeed Studio 113020003 GPS Module	UB-5010	2.75-3.6	Tracking Acquisition Sleep/Standby	Tracking Acquisition PRN	22 Altitude 66 Velocity Acceleration	-160dBm	Position Velocity Timing	1Hz 5m CEP without SA 0.1m/s without SA 10ns RMS	16mm x 12.2mm x 2.4mm	210nok
Adafruit Ultimate GPS Breakout	MT3339	3-4.3	Tracking Acquisition	Tracking Acquisition PRN	22 Altitude 66 Velocity 210 Acceleration	-165dBm	Position Velocity Timing	Up to 10Hz 3m CEP without SA 0.1m/s without SA ±20 ns RMS	16mm x 16mm x 5.2 mm	260nok
MikroElektronika GPS Click LEA-6S		3.3				-147dBm		5Hz		449nok
Digilent Pmod GPS	MT3329	3				-165dBm		10Hz		407nok
SparkFun GPS Breakout		3.3	Tracking		72 Altitude Velocity Acceleration		Position Velocity Timing	18Hz 2.5m 0.05m/s 30ns		358nok
Adafruit GPS BREAKOUT	MT3329	3.2-5	Tracking Acquisition	PRN	66 Altitude 210 Velocity Acceleration	-165dBm	Position Velocity Timing Acceleration	Up to 10Hz 3m 2D-RMS 0.1 m/s 100ns RMS 0.1 m/s ²		269nok

Figure 65: Comparison of GPS [52, 53, 54, 55, 56, 57]

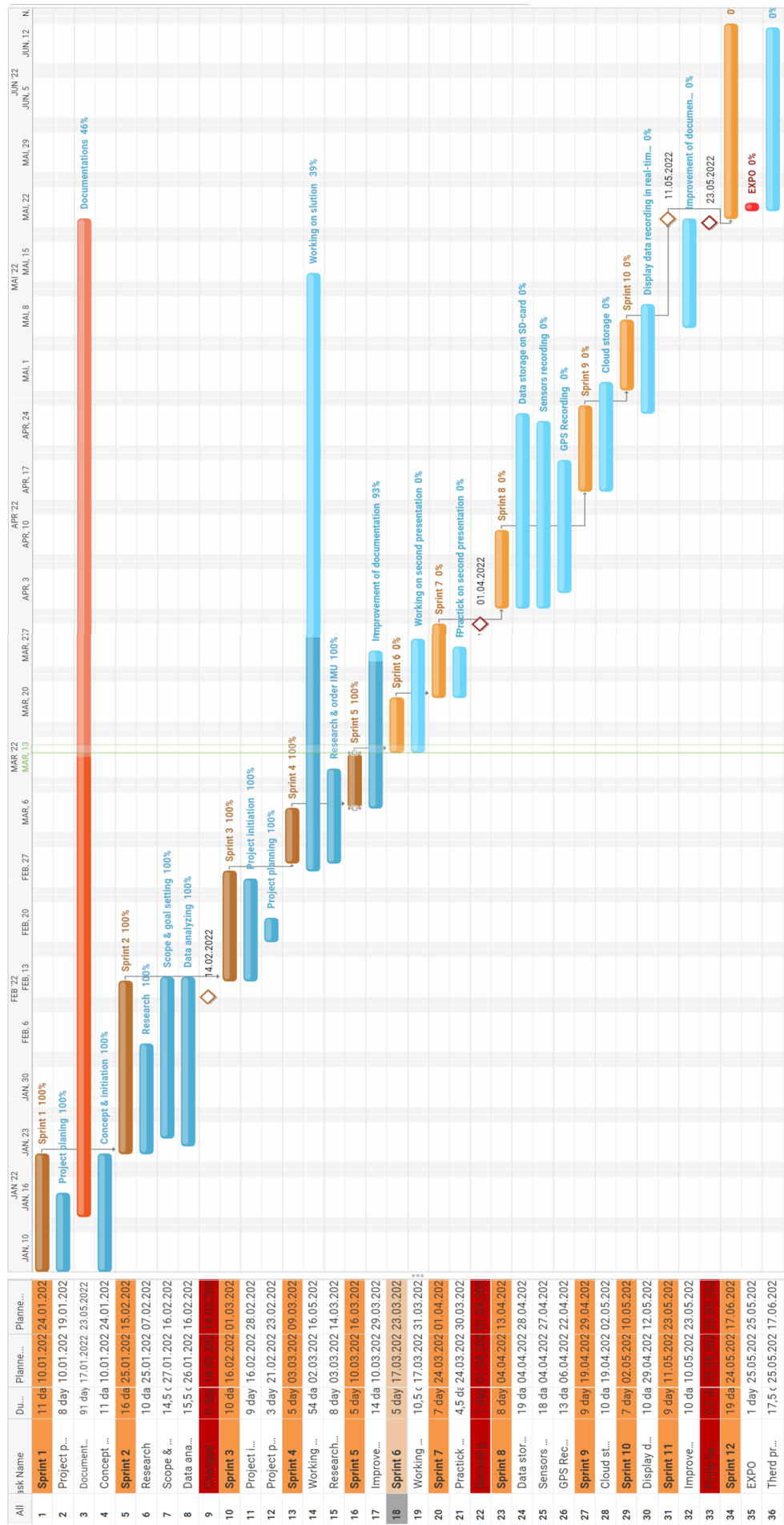


Figure 66: Gantt diagram

Appendix X - System Design

The project is about collecting data from an excavator, sensors, and GPS and storing it both locally, on the cloud, and displaying it in real-time. The diagram gives a high-level picture of the whole system.

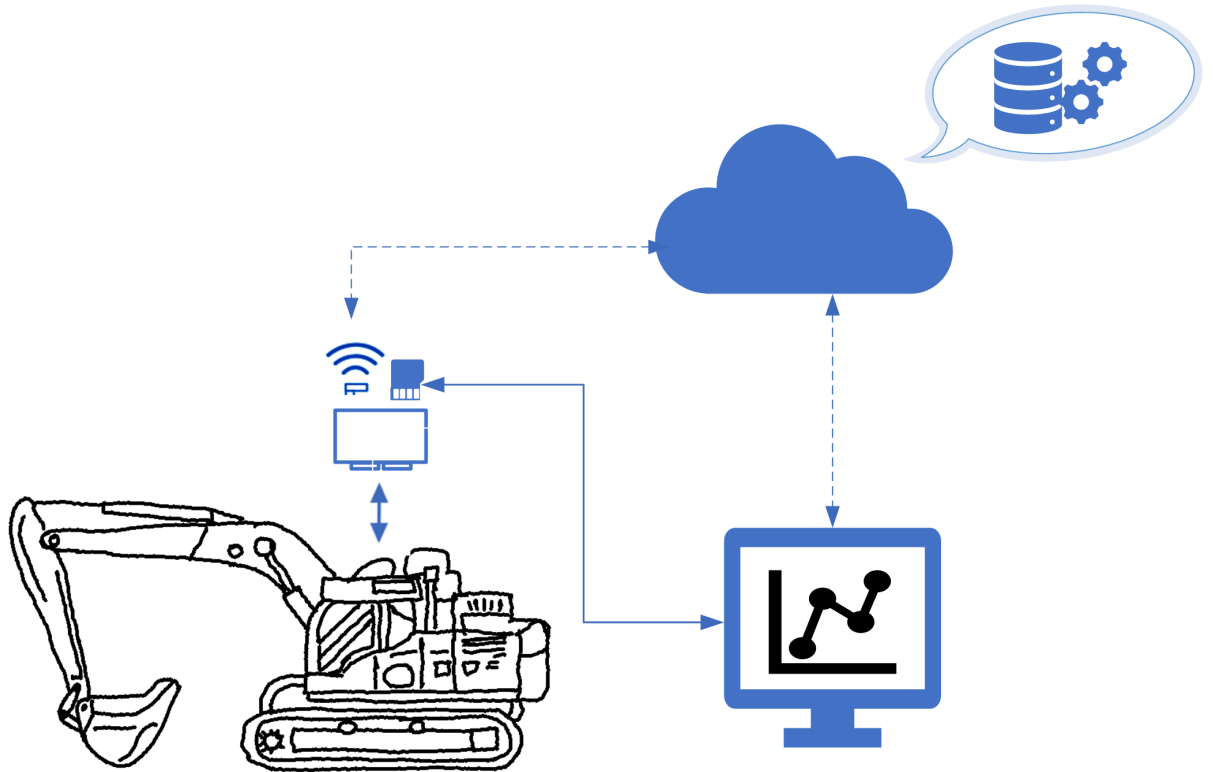


Figure 67: System design

Appendix Y - RTC Implementation

When implementing a real-time clock there were lots of back-and-forths on how to accomplish it. First, the GPS module was used in order to set the Raspberry Pi's clock to equal the GPS's real-time clock. When implementing this, the Raspberry Pi was set two hours behind local time. Reading through the datasheet it seemed like using the GPS to sync time would be impossible since the module is set to UTC time out of the box and it is not "writeable" [58, p.44].

Because of the obstacle met, a new real-time clock was acquired. This new real-time clock used I2C communication instead of UART communication like the GPS. Since the Inertial Measurement Unit was also using an I2C connection, another obstacle was encountered. The Raspberry Pi has multiple I2C pins which in theory should not cause any complications when using two I2C units, but that was not the case. Both units were recognized by the Raspberry Pi, but in order to access the real-time capability of the RTC one needs to assign the I2C address exclusively to the unit. The problem with this is that both the IMU and RTC have the same address of 68. When assigning the address to the RTC to access its real-time capability the IMU no longer had access to this address and would stop working.

Because of the obstacles mentioned above, the team had to try utilizing the GPS module once again. Thankfully there was a way of adjusting the GPS time on the Raspberry Pi itself, disregarding the locked timer of the internal RTC and the Raspberry Pi would finally be able to sync up to local time.

Appendix Z - Alternative Mounting

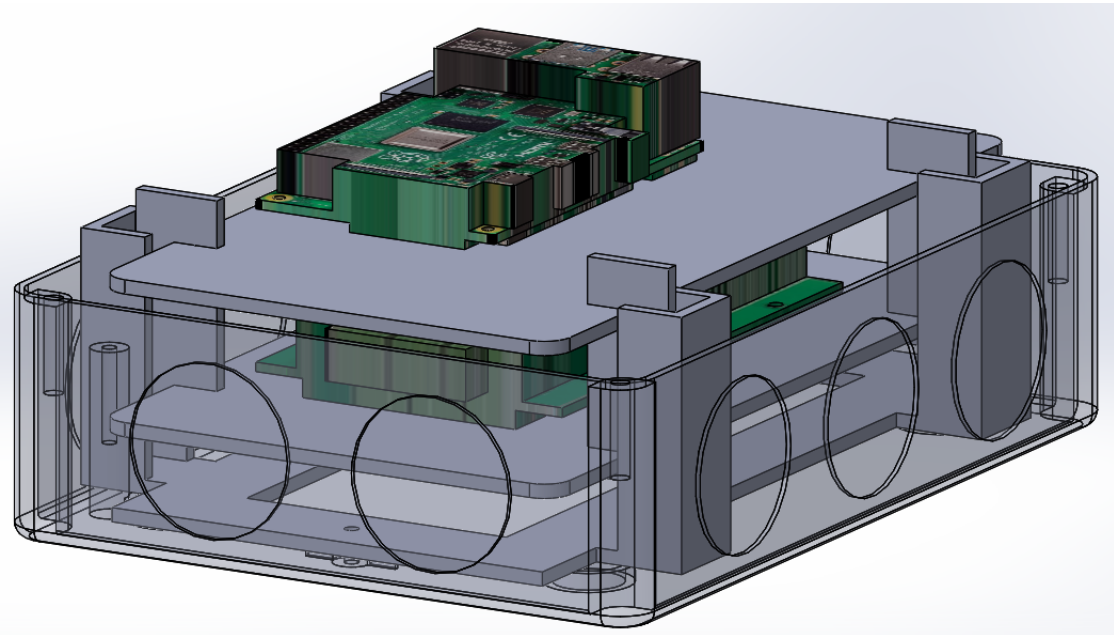


Figure 68: Alternative mounting of system with sliders for easier access to components

Abdirahman Ali

Student No. 233831

Group 8

Technical Contribution	When	Documented
<p>At the beginning of the project, I was mostly involved in planning and project management. I have spent time reading the documentation we have received from the client. Read on, among other things, j1939-71 pdf, warning system, information system, instruments.pdf.</p> <ul style="list-style-type: none">- I have made gantt chart diagram- Get started with the CAN shield for the Arduino Mega	<p>week - 1 10.01 - 18.01 week - 2 19.01 - 25.01</p>	<p>It is not documented</p>
<p>Week 3 through week 5 I continued working on what I did in the first two weeks. I worked with log files that were sent by the client, the log files were recordings of the CAN bus messages. began researching what the CAN messages are meant by and the frequency of the number of messages per second. In addition, I worked with bus master.</p>	<p>week - 3 26.01 - 01.02 week - 4 02.02 - 08.02 week - 5 09.02 - 15.02</p>	<p>In Appendix H</p>
<p>Week 6 through week 9, I worked to establish CAN communication. We got Arduino uno with CAN shield with SD card access to collect CAN data on it until we were told to switch to raspberry pi instead of Arduino. Week 7 and 8 I worked to establish communication between Raspberry Pi and Arduino DUE. Communication between them I got it connected.</p> <ul style="list-style-type: none">- I worked on sending virtual CAN messages using two MCP2515 and Arduino nano and raspberry pi. <p>Week 9 so I started working on ssh connection to raspberry pi</p>	<p>week - 6 16.02 - 22.02 week - 7 23.02 - 01.03 week - 8 02.03 - 08.03 week - 9 09.03 - 15.03</p>	<p>In section: Remote Desktop connection — page 53-54</p>
<p>Week 11 and 12 I started working with inertial measurement unit (IMU) and gps we received from the company. In early April, the company wanted to test IMU and GPS. I worked on getting them ready for the test. I worked on compiling GPS and IMU through on serial communication over on the Raspberry Pi. I wrote python code for gps and connected directly to raspberry pi. for IMU I used the arduino uno and connected it to the Raspberry Pi via USB cable.</p>	<p>week - 11 23.03 - 29.03 week - 12 30.03 - 05.04</p>	<p>In Appendix E3 IMU and GPS data</p>

<p>Week 13 and week 14 I worked with rewriting codes from python to C/C ++ for GPS and IMU. C/C ++ that we had on our plan for choosing a programming language. I worked with IMU and GPS in parallel. I got GPS written to C and then I started saving GPS data on CSV file. IMU it was a bit difficult to calculate the accelerometer, gyroscope and magnetometer together. I had random IMU-data written on the screen.</p>	<p>week - 13 06.04 - 12.04 week - 14 13.04 - 19.04</p>	<p>In Appendix E3 IMU and GPS data</p>
<p>Remote Desktop Connection: had to fix a bug that appeared during a test with the client. there was no local network, Raspberry Pi IP address could not connect to SSH, I used VNC instead</p>	<p>week - 15 20.04 - 26.04</p>	<p>In Section: Remote Desktop Connection — page 55</p>
<p>From week 16 to 18 i was working with MATLAB to visualize IMU in 3D From week 16 to Week 18 I created the project website (D.E.C.H.O) From Week 16 to week 18, I established Connection between Google cloud platform and Raspberry Pi.</p>	<p>week - 16 27.04 - 03.05 week - 17 04.05 - 10.05 week - 18 11.05 - 17.05</p>	<p>In Sections: Cloud Storage – page 50 and GCP – page 51-54 and Appendix E2 – GCP connection. Appendix E17 MATLAB – inertial-measurement.m</p>
<p>The tests shown in appendix F: I have worked with the tests from T31 to T41 in this project</p>	<p>week - 4 to week - 18</p>	<p>In Appendix F: T31 - T41</p>

Petter Wang Tveit

Student No. 213813

Group 8

Technical Contribution	When	Documented
Worked on decoding CAN-messages.	Week 5	Not documented in final report.
Picked out components used in the final system in terms of Single-Board Computer and CAN bus card.	Week 6	Sec. 3.2.3, 3.2.4
Developed other diagrams to gain greater understanding of the system. This included: <ul style="list-style-type: none">• System Context Diagram• Overview of proposed system• Complete System Architecture	Week 6	Fig.10, Fig.11, Fig.40
Established communication between Raspberry Pi and CAN bus card.	Week 8	Sec. 4.2.2.6, 4.2.2.7
Established communication between two Arduino Unos through two MCP2515 CAN controllers.	Week 8	Appendix H3
Developed UML-diagrams for software architecture. This included: <ul style="list-style-type: none">• Flow Charts• Class Diagrams I also helped with other flow charts, use cases and sequence diagrams.	Week 11, 17	Fig.29, Fig 31, Fig.32, Fig.41, Fig.42, Fig.43.
Established communication between Inertial Measurement Unit and Raspberry Pi through I2C.	Week 12	Sec. 4.2.3.1
Established communication between GPS and Raspberry Pi through UART.	Week 13	Sec. 4.2.3.2
Wrote code for collecting Inertial Measurement Unit data and writing to a text file with timestamps.	Week 12, 15	Sec. 4.2.3.1, 4.2.5
Wrote code for collecting GPS data and writing to a text file with timestamps.	Week 15	Sec. 4.2.3.2, 4.2.5
Wrote scripts that would run on boot in order to run IMU, GPS and serial code whenever the system was booted.	Week 15	Sec. 4.2.4.1
Wrote code for collecting serial data and writing to a text file with timestamps.	Week 16	Sec. 4.2.4

Implemented RTC using RTC module communicating with Raspberry Pi through I2C	Week 16	Appendix Y
Implemented RTC using the GPS's real-time capability.	Week 13, 16	Sec. 4.2.4.2
Implemented a surveillance system by adding a web camera taking pictures with a fixed interval.	Week 17	Sec. 4.2.9
Established communication between Raspberry Pi and Google Cloud.	Week 18	Sec. 4.2.7
Performed tests throughout the project	Week 8-18	Appendix F: Test T1, T2, T3, T13, T31, T42, T43, T44, T45, T46, T47, T48

Mahram H. Safdari

Student No. 222988

Group 8

Technical Contribution	When	Documented
Group work about the project plan.	Week 1: 10/Jan-18/Jan	
Group work about the requirements and finding project models. I have created a test plan where the plan should contain a test objective, test method, expected result, actual result, test condition (Pass/Fail), and date.	Week 2: 19/Jan-25/Jan	Section: 2.5 Test plan.
Group work with first presentation, documentation, and among other things I made: Test table for test plan. Prioritization of test plan with table. Prioritization of requirements plan in table. Wrote documentation for the points above.	Week 3: 26/Jan-01/Feb	Section: 2.5 Test plan, Appendix F: Test Tables.
Group work with the first presentation, improved and completed the presentation.	Week 4: 02/Feb-08/Feb	
Group work and I analyzed the "belting" log. Found the PGN numbers for the logs. Found out why there is the same PGN in several places. Managed to identify sender and recipient, but still unsure of some recipient in CAN messages. Since the project plan was changed, this is not documented in the main documentation.	Week 5: 09/Feb-15/Feb	
Group work and I made a PUGH matrix for different programming languages for to choose the right language for hardware.	Week 6: 16/Feb-22/Feb	<ul style="list-style-type: none">• Second documentation. Sec: 5.5.6• Third documentation Sec: 4.2.1
Group work, and I started to make a new test plan based on the new project plan.	Week 7: 23/Feb-01/Mar	Section: 2.5 Test plan, Appendix F: Test Tables.

<p>Group work and I produced a new test plan. Made finished template for test plan with a test example. I got a new assignment from the group to work with Markus to find the right sensor for GPS. Found two types of GPS, one with USB port power outlet, and one that gets power by connecting via standard pins. Got another task that was about helping Salem get in touch with the CAN bus screen and Arduino Mega to be able to read the SD card so that we can read the messages from the SD card via the Arduino IDE. To avoid repetition in our document, this will be document it only in one place.</p>	<p>Week 8: 02/Mar-08/Mar</p>	<p>Section: 2.5 Test plan, Appendix F: Test Tables.</p>
<p>Group work and Improved PUGH matrix for programming language selection.</p>	<p>Week 9: 09/Mar-15/Mar</p>	<p>Section: 4.2.1.</p>
<p>Group work with presentation two, and documentation.</p>	<p>Week 10: 16/Mar-22/Mar</p>	
<p>Groupwork with second documentation and second presentation preparation. I have documented new tests in the test plan. Worked with code for Adafruit Ultimate GPS in Arduino IDE to find position and time. I wrote minutes of meetings with internal and external supervisors.</p>	<p>Week 11: 23/Mar-29/Mar</p>	<p>Appendix F: Test Tables</p>
<p>Group work with second documentation and second presentation preparation. I managed to read GPS coordinates (Adafruit Ultimate GPS) by connecting to Arduino Mega, but the coordinates contain some commas. The solution for empty values is an external GPS antenna. Later the GPS has been used with Raspberry Pi and IMU by Peter and Abdirahman. To avoid repetition in our document, the GPS will be document it only in one place. I have documented new test in the test plan.</p>	<p>Week 12: 30/Mar-05/April</p>	<p>Appendix F: Test Tables</p>
<p>Group work and I documented three new tests to the test plan. With Abdirahman, I tried to get in touch with the Raspberry Pi via Remote Desktop Connection, but it was unsuccessful. But via Putty, we managed to get contact with the Raspberry Pi. I also found that the GPS code was correct anyway, but the GPS is missing an antenna. The GPS receives signals without an antenna, but it is very slow. When the GPS receives signal, the empty commas that I last mentioned are replaced with values from the satellites via the signal receiver on the GPS. I wrote the minutes of the meetings with the internal supervisor. Documentation in GPS section. Me and Salem we received a new task, read (Read J1939 (CAN bus) data on Arduino DUE by listening to CAN messages on Arduino DUE by connecting MCP2515 CAN controllers), The challenge was that we are depending on retrieving data from the excavator that we do not have access to now. We found a solution that we can test it in another way. For example, we tried to read data from a computer by connecting the Arduino DUE with two MCP2515 and an Arduino UNO to two computers.</p>	<p>Week 13: 06/April-12/April</p>	<ul style="list-style-type: none"> • Sec: 2.5 Test plan • Sec: 4.2.2.1, • Sec: 4.2.2.2 • Sec: 4.2.2.3 • Sec: 4.2.2.5 • Sec: 4.2.2.6 • Appendix F: Test Tables • Appendix H: CAN bus and Arduinos

<p>We did group work as usual. Then me and Salem, we tried to read J1939 CAN bus data on the Arduino DUE by connecting two computers to pretend it was an excavator. We created virtual CAN messages to test it. We got output values on serial monitor only via one PC, but our expectation was to get output values on both PCs at the same time when the Arduino DUE, Arduino UNO, and MCP2515 (CAN controllers) are connected. I also documented new performed test to the test plan.</p>	<p>Week 14: 13/April-19/April</p>	<ul style="list-style-type: none"> • Sec: 2.5 Test plan • Sec: CAN-messages: 4.2.2.1 - 4.2.2.6 CanSniffer • Appendix F: Test Tables • Appendix I: CAN bus and Arduinos
<p>We did group work as usual. Then me and Salem, we continued working on reading J1939 CAN bus data on the Arduino DUE by listening to CAN messages. We tried to get the "CAN-Sniffer" code to work, but it did not work as planned. We thus had to investigate how the Arduino DUE, MCP2515 (CAN controllers) and CAN bus work, and had to find out more about the J1939 documentation. On Monday we visited the workshop and tested our code on the excavator. The code worked halfway where we got some data from the excavator, but also some error messages. We needed to keep working on the Arduino DUE and fix our code. A lot of time was spent looking, trying, changing, testing, and failing. I also documented a new test to the test plan which was performed.</p>	<p>Week 15: 20/April-26/April</p>	<ul style="list-style-type: none"> • Sec: 2.5 Test plan • Sec: CAN-messages: 4.2.2.1 - 4.2.2.6 CanSniffer • Appendix F: Test Tables • Appendix I: CAN bus and Arduinos
<p>We did group work as usual. Then me and Salem, we had to continue working together to fix our issues on the last task on the CAN bus communication. We have spent a lot of time on troubleshooting on the communication between Arduino Due and Arduino Uno. We have also spent some time getting to know the oscilloscope / "Analog Discovery DIGILENT" to measure the frequency etc. for troubleshooting. We found that the MCP2515 (CAN controllers) used is 8kHz. We found that the ideal CAN "topology" is for the bus to have a connector with terminating resistors at each end. Each resistor should ideally be 120 ohms to achieve a 60-ohm difference between CANH and CANL. Due to different resistors between CANH and CANL the communication failed, but the problem was solved when we switched resistors. We have improved the code after the test we have taken on the excavator at the company 02.05.2022. In addition, I worked on tests.</p>	<p>Week 16: 27/April-03/May</p>	<ul style="list-style-type: none"> • Sec: 2.5 Test plan • Sec: CAN-messages: 4.2.2.1 - 4.2.2.6 CanSniffer • Appendix F: Test Tables • Appendix I: CAN bus and Arduinos
<p>Group work as usual. This week we have been with the company and performed tests on the excavator. I have documented 11 new tests and documented the programming part for transmitter and receiver for virtual CAN communication. I have made three "flowcharts", in addition to documented wiring diagrams for these. I wrote the minutes of the meetings with the internal supervisor.</p>	<p>Week 17: 04/May-10/May</p>	<ul style="list-style-type: none"> • Sec: 2.5 Test plan • Sec: CAN-messages: 4.2.2.1 - 4.2.2.6 CanSniffer • Appendix F: Test Tables • Appendix I: CAN bus and Arduinos

<p>Group work and worked with documentation on tests, part of programming part (CAN bus communication (Arduino UNOs with Arduino DUE and the excavator)).</p>	<p>Week 18: 11/May-17/May</p>	<ul style="list-style-type: none"> • Sec: 2.5 Test plan • Sec: CAN-messages: 4.2.2.1 - 4.2.2.6 CanSniffer • Appendix F: Test Tables • Appendix I: CAN bus and Arduinos
<p>Group work on documentation and EXPO stand preparation with group. I documented tests, and part of CAN bus communication etc.</p>	<p>Week 19: 18/May-24/May</p>	<ul style="list-style-type: none"> • Sec: 2.5 Test plan • Sec: CAN-messages: 4.2.2.1 - 4.2.2.6 CanSniffer • Appendix F: Test Tables • Appendix I: CAN bus and Arduinos
<p>Group work with EXPO stand, preparation for the final project presentation three, and performing the final presentation.</p>	<p>Week 20: 25/May-01/June</p>	

Salem Rezaie

Student No. 235114

Group 8

Technical Contribution	When	Documented
Before week five, we worked almost exclusively on project management and planning. From week five, I was tasked with developing an application that can read and filter data packets in real-time in order to get an overview of incoming traffic and decode unknown messages. I started to learn about QT and how it interacts with Python to build an application, but after some research on the internet, we came across a GitHub link that did exactly what the we needed [?]. When I downloaded the code, I faced a library problem, which I resolved, and used the time to learn about the application and how it functions. By the end of week five, there were changes in the project, and this application is not used in the new project.	Week: 5	In appendix M and E10
Focused on familiarizing myself with the new project and investigating how to implement the new requirements. In order to get a complete overview of the project, I draw system design diagrams.	Week: 6	In appendix: X
Examined how sensor data can be stored and how to synchronize CAN data and sensor data, in order to transfer the data by using parallel threads. I have also investigated how we can minimize the possible delay of data transfer.	Week: 7	
Developed UML-diagrams: use cases, sequence diagrams, and software architecture models.	Week: 10	Documented in: <ul style="list-style-type: none">• Sec: 3.3.1 Sec. 3.3.2 Sec. 3.3.2.1 Sec. 3.3.2.2 Sec. 3.3.3• figures: 15, 16, 17 and 18

<p>Wrote code for data storing on SD-card, but since we did not have any "Inertial Measurement Unit" (IMU) to test, i used potmeter to read the values and store them on SD-card by using an Arduino Uno.</p>	<p>Week: 8 and 11</p>	<p>Appendix K, K1 and E9</p>
<p>Finally, we received the IMU, wrote code to read the data from the BMX160 sensor (IMU) and stored data on SD card by using Arduino Uno.</p>	<p>Week: 12</p>	<p>Appendix: L, L1, L2 and E8</p>
<p>The list of Schematics, diagrams and flowchart i drown.</p> <ul style="list-style-type: none"> • CAN-bus resistance <p>Schamatics:</p> <ul style="list-style-type: none"> • Arduino Uno to - Arduino Uno • Arduino Uno to - Arduino Due • Two Arduino Uno to - Arduino Due • Potmeter to and SD - Arduino Uno • IMU and SD-card to- Arduino Uno <p>Flow charts:</p> <ul style="list-style-type: none"> • Flow chart of IMU using Arduino • Flow chart of canSniffer 	<p>Week: 13-17</p>	<p>List of figures:</p> <ul style="list-style-type: none"> • Fig. 26 Fig. 27 Fig. 48 Fig. 49 Fig. 52 Fig. 54 Fig. 55 Fig. 56
<p>Group work with Mahram: From week 13 until week 17, we worked constantly with the requirement to read the incoming data from the excavator. Since we did not have access to the excavator all the time to conduct tests, so we created virtual messages to send to our receiver, an Arduino Due. In order to resolve the communication problem between the Arduino Due and Arduino Uno, we spent a significant amount of time troubleshooting. However, the system is able to read data from two CAN buses at the same time in both extended and standard formats. we wrote codes including virtual CAN message transmitter and CAN receiver</p>	<p>Week: 13-17</p>	<p>documented in:</p> <ul style="list-style-type: none"> • Sec. 4.2.2.1 Sec. 4.2.2.2 Sec. 4.2.2.3 Sec. 4.2.2.4 Sec. 4.2.2.5 Sec. 4.2.2.6 Sec. 4.2.2.7 <p>In appendix:</p> <ul style="list-style-type: none"> • E1, E4, E5, E6, E7, I, I1, I2, I3, I4 and J

Markus Danielsen

Student No. 223635

Group 8

Technical Contribution	When	Documented
Bought and created 3D model of small junction box used in version 1 of project	Week 5	Chapter 4.1.1.1 Figure 19 Appendix S
Found information about IMUs and GPSs and put it in an excel document and compared them	Week 6	Chapter 3.2.1 Chapter 3.2.2 Appendix U Appendix V
3D-modeled plate for Arduino DUE	Week 6	Appendix S
3D-printed plate for Arduino DUE	Week 7	Appendix S
Chose what IMU to use for project by rating IMUs in Pugh-matrix	Week 7	Chapter 3.2.1 Table 2
Chose what GPS to use for project by using rating GPSs in Pugh-matrix	Week 8	Chapter 3.2.2 Table 3
Researched and compared more IMUs and GPSs because of test requiring IMUs and GPSs with fast delivery. Chose new IMU and GPS	Week 11	Chapter 3.2.1 Chapter 3.2.2 Appendix U Appendix V
Created system overview to help decide what components to use and how they should be connected. No longer in use	Week 11	Second Presentation Slide 14
3D-modeled components and plates	Week 12	Chapter 4.1.1.1 Appendix S
Found placement of components in box	Week 13	Chapter 4.1.1.2 Appendix S
3D-model components and plates, 3d-printing not available	Week 13	Chapter 4.1.1.3 Figure 21 Appendix S
Laser cut plates, and assemble system	Week 13	Chapter 4.1.1.3 Appendix S
Started on other concept for mounting of system inside box	Week 14	Appendix Z
Bought and 3D-modeled bigger junction box	Week 15	Chapter 4.1.2 Figure 22 Appendix S

3D-modeled and laser cut simplified system	Week 15	Chapter 4.1.1.4 Appendix S
3D-modeled complete system, except RTC and IMU as these had to be soldered together on a Vero board first	Week 15	Appendix S
Completed 3D-model for system and laser cut plates for system in bigger box	Week 16	Figure 25 Appendix S
Created list of changes made to the project and 3D-model	Week 16	Appendix S
Changed 3D-model for more optimal placement of components	Week 17	Chapter 4.1.2 Appendix S
Updated list of changes	Week 17	Appendix S
Tested fitment of all 3D-printed and laser cut plates throughout the project	Week 7 - 17	Appendix F Test T17 - T30
Created figures showing floorplan	Week 18	Chapter 4.1 Figure 20 Figure 23 Figure 24 Appendix S

Fouad Irkayek

Student No. 212329

Group 8

Technical Contribution	When	Documented
first week I used my time to understand the project that was given to us from the employer, also have spent some time reading the documentation we received from the employer. I downloaded bussmaster program to find/read the signal from the ECU, and worked with decoding of the encrypted code	week - 1 (10.01). week - 2 (25.01).	not documented
I went through the wiring system to the excavator so that can helps us to find the 5 different ECU's that inside the excavator. Have used bussmaster program to analyze, filter and sort the different data-logs we received from the employer to find the encrypted messages faster. I worked with "digging" log file, i used excel program to sort the date, the time, PGN numbers, the encrypted code, data bytes and the channel type.	week - 3 (26.01) - week - 5 (01.02).	appendix N
I have been working on finding a method to store and send data using Arduino MKR GSM 1400, with this we can receive and store it directly. I also worked with what is "IP" and how to use in the our project. I worked with finding how large storage space we need to store the collected data. I made 3 different types of risk tables. I made system context diagram to understand the system even more. and made risk process model on how we solve for risk.	week - 6 (16.02) week - 9 (15.03).	appendix R and G

<p>I continued working with Risk Table for new risks, I went through the different data sheets for the different components like IMU sensor Gps sensor, and made wiring diagram for the different sensors we used. also made a wiring diagram for the whole system on how thing would be connected. as group we discussed whether we will use arduino or raspberry pi so, i know what operative system to use when drawing the wiring diagram. also did some reading on how long cable we need when use to send data. but after a while we agreed to use wireless metod of sending data when we were near the excavator. I soldered IMU and GPS sensor. i made a table on how much current each component draws.</p>	<p>week - 10 (16.03) week - 13 (12.04).</p>	<p>appendix G, P, O</p>
<p>After kowing how much current we need to power the each component in the system, i started on what type of voltage step-down we need since there are many different type to choose from. I used visio program to draw the wiring diagram, but was not the best, so i tries to use solid-works program for designing of eletrical system, it tok a lot of time to just Barely understand how the system work. I soldered IMU and RTC sensors to a veroboard. i worked with MATLAB to show the Orientation of the imu in 3D dimension when mounted to the excavator.</p>	<p>week - 14 (13.04) week - 18 (17.05).</p>	<p>appendix Q, G, O</p>