Joakim Aleksandersen, Sara Savanovic Djordjevic,
Hoa Ben The Nguyen

# IceMap

## An ice safety mapping system

Bachelor's thesis in Programming
Supervisor: Pål Anders Floor
Co-supervisor: Marius Pedersen
May 2024

Bachelor's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Joakim Aleksandersen, Sara Savanovic Djordjevic,
Hoa Ben The Nguyen

# IceMap

## An ice safety mapping system

Bachelor's thesis in Programming
Supervisor: Pål Anders Floor
Co-supervisor: Marius Pedersen
May 2024

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

# Foreword

The contents of this Bachelor thesis were produced by Joakim Aleksandersen, Sara Savanovic Djordjevic, and Hoa Ben The Nguyen at the Institute of Computer Science, NTNU, Gjøvik. We thank the municipality for providing us with an engaging and meaningful task. To contribute to a service designed for the benefit of the public has been a great experience, and we anticipate further development, deployment, and use of the IceMap system. We especially want to thank our academic advisors, Pål Anders Floor and Marius Petersen, for their enthusiasm, support, and invaluable advice.

# Abstract

In Norway, a country known for its appreciation of nature, people refuse to let cold weather stop them from enjoying outdoor activities. Some winter activities are riskier than others, such as those that involve traversal of frozen lakes. The IceMap system was developed to provide an additional safety tool for exactly those activities as part of a Bachelor thesis in programming. IceMap, which was commissioned by Gjøvik municipality, consists of a Python server and a Dart/Flutter-based mobile application. The server processes weather, satellite and sensor data to assess the condition of the lake ice. The assessments are displayed in the mobile application as a color coded map. A few similar products are already available on the market, but these depend on user input from ordinary people who may have limited knowledge of ice safety. Otherwise, current methods include manual measurements which are reliable, but unsafe for the pepole conducting them. To remove the necessity for user input and manual measurements, IceMap introduces an automated measurement system with a LIDAR-mounted drone and satellite imagery. Since the necessary sensors could not be acquired during the project period, the physical components have been replaced with theoretical solutions. Instead of using data from an actual sensor, test data from various online sources was used and manipulated to fit the projects requirements. Research was carried out to identify the most suitable sensor types, and all necessary setup to implement actual sensors is included in the result. What remains is to acquire and test the system with physical sensors.

# Sammendrag

I Norge, et land kjent for sin verdsettelse av naturen nekter folk å la kaldt vinterværhindre dem fra å nyte utendørsaktiviteter. Noen vinteraktiviteter er mer risikable enn andre, for eksempel de som involverer å gå ut på frosne innsjøer. IceMap systemet ble utviklet fir å tilby et ekstra sikkerhetsverktøy til de som krysser frossne innjsøer. Systemet ble utviklet som en del av en bacheloroppgave i programmering. IceMap systemet ble bestilt av Gjøvik kommune, og består av en Python-server og en Dart/Flutter-basert mobilapplikasjon. Serveren behandler vær-, satellitt- og sensordata for å vurdere tilstanden til isen på innsjøer. Vurderingene vises i mobilapplikasjonen i form av et fargekodet kart. Noen få lignende produkter finnes fra før av, men disse avhenger av brukerinspill fra vanlige folk som kan ha begrenset med kunnskap om istrygghet. Ellers er det i dag vanlig å gjøre målinger manuelt. Selv om manuelle målinger er pålitelige, er de utrygge for de som må utføre dem. For å fjerne behovet for brukerinspill og manuelle målinger introduserer IceMap et autmatisert målesystem med en LIDAR-montert drone og satellittbilder. Siden de nødvendige sensorene ikke kunne anskaffes i løpet av prosjektperioden er de fysiske komponentene erstattet med teoretiske løsninger. I stedet for å bruke data fra en faktisk sensor ble testdata fra ulike nettkilder manipulert for møte prosjektets formål. Det ble utført forskning for å identifisere de best egnede sensortypene, og alt nødvendig oppsett for å implementere faktiske sensorer er inkludert i resultatet. Det eneste som gjenstår er å anskaffe fysiske sensorer og teste sytemet med sensorene.

# Contents

# Figures

# Tables

# Code Listings

# Acronyms

**ADB** - Android Debug Bridge

**AI** - Artificial Intelligence

**API** - Application Programming Interface

**BBox** - Bounding box

**CSV** - Comma Separated Values

**DBMS** - Database Management System

**DJI** - Da-Jiang Innovations

**DOT** - The Department of Transportation of the Northwestern Territories

**EMI** - Electromagnetic Induction

**ER diagram** - Entity Relationship diagram

**GPS** - Global Positioning System

**I** - Intensity

**IDE** - Integrated Development Environment

**ID** - Identification

**ILI** - Icy Lake Index

**JSON** - JavaScript Object Notation

**LAS** - Log ASCII Standard

**LAZ** - LASzip

**LIDAR** - Light Detection And Ranging

**MVP** - Minimal Viable Product

**NDMI** - Normalized Difference Moisture Index

**NDWI** - Normalized Difference Water Index

**NIR** - Near Infra Red

**NDVI** - Normalized Difference Vegetation Index

**NOK** - Noregian currency

**NVE** - Noregs Vassdrags- og Energidirektorat (The Norwegian Energy Regulatory Authority)

**OSM** - OpenStreetMap

**POI** - Point Of Interest

**QGC** - QGroundControl

**RTH** - Return-To-Home

**SAR** - Synthetic Aperture Radar

**SDK** - Software Development Kit

**SVM** - Support Vector Machine

**SWIR** - Short Wave Infra Red

**SWOT** - Strengths, Weaknesses, Opportunities, and Threats analysis (analysis framework)

**UI** - User Interface

**USB** - Universal Serial Bus

**UTM** - Universal Transverse Mercator coordinate

# Chapter 1

# Introduction

As home to Norway's largest lake, Mjøsa, the citizens of the municipality of Gjøvik are naturally fond of utilizing the lake for recreational purposes. The lake is a particularly popular spot during summertime, but many continue to visit the lake throughout winter. For some, the height of winter is the perfect time to pull out ice skates, skies, and fishing gear. This is especially true on particularly cold winters, such as the recent winter of 2023/2024. In order to provide the citizens of Gjøvik with reliable information regarding the safety of the lake ice, the municipality of Gjøvik has commissioned a system for automatic measurement of the ice, as well as an application to convey the ice safety to the general population. To fulfill this task, we developed IceMap: an automated ice thickness measurement system with its very own mobile application.

## 1.1 Background

Traversing a frozen lake can be dangerous. Although it is difficult to find concrete statistics, people occasionally fall through the ice on Mjøsa. A quick search on Firefox shows multiple articles from recent years describing incidents where people have fallen through the ice. Some may think that the natural solution to this problem is to ban traversal of the ice, but in Norway such an attempt would likely be unfruitful. In a country that has long winters and a culture that greatly values nature, it is difficult to deter the general population from such activities, even if they are dangerous. Instead, local governments and the police tend to encourage the population to take precautions and make safety evaluations.

Until 2024, the municipality of Gjøvik has mitigated such risks by manually measuring the ice thickness. This approach provides reliable measurements, but also risks the safety of the person conducting the measurements. Not only are manual measurements risky, but they are also limited by multiple factors. Firstly, the person conducting the measurement must be experienced, and such individuals are few. There are limits to how large areas can be covered by foot or ski, and only areas that are already assumed to be safe can be measured. To share the information gathered by the sensors, the municipality has also requested an application to notify people of the measurements conducted by the system.

### 1.1.1 Problem area and scope

The project focused on implementing existing sensor technologies, processing their data, creating a map solution, and implementing the map in a mobile application. The group did not delve into engineering of existing sensor technologies, or the creation of new ones, even if the existing technologies were not optimal for their intended purpose. Due to the width of the project scope, usability and user testing was not prioritized.

### 1.1.2 Task description

In addition to a short description of the tasks background, the municipality proposed the following approaches to solving the task:

> 1. What kind of sensor technology is best suited for measuring ice thickness?
> 2. Design and create an app for the municipality's residents for ice thickness status and notification of changes.
> 3. Develop a prototype ice measurement service, complete with sensor, IT solution and citizen app.

A combination of all three approaches was chosen. The request was completely open beyond these approaches, meaning that the exact design, requirements, and tools were left for the group to chose. To further define the task, the group added their own requirements, which will be covered in chapter 2. The entire Norwegian task description provided by the municipality can be found in appendix A Original task description.

## 1.2 Goals

### 1.2.1 Result goals

This project aimed to develop a system that could both automate the measuring, and an application that could convey the data. The system would consist of a server, a mobile application, and sensors. The server would deal with data input and processing, while the mobile application would deal display the processed information. To keep the application user friendly, the resulting mobile application was expected to be relatively simple and only consist of a couple pages.

The server should utilize a couple of different sensor technologies and third party APIs to determine the ice thickness. Since sensors could not be deployed and tested within the project time frame, demonstrating the systems functionalities on test data was deemed sufficient. The system must at the very least demonstrate its functionalities on Mjøsa, although adding more lakes must be possible.

### 1.2.2 Effect goals and learning goals

The desired effect of this project is to reduce the need for manual measurements and user input, and inform people of where safe ice is. The primary learning goals were to learn to code in Python and Dart, improve knowledge of data processing and sensors, learn concepts unique to mobile programming, and to build and design more extensive systems.

## 1.3 Prior knowledge and skills

### 1.3.1 Programming

All group members had some prior experience developing in Python, but this experience was very limited. Dart, Flutter, and mobile programming on the other hand were almost entirely new to the group. The group was also not familiar with any of the utilized third party APIs and software's. These had to be learned completely from scratch.

### 1.3.2 Ice theory and data interpretation

To effectively select appropriate data sources and understand how to process the data, the group had to spend a lot of time researching topics related to ice, LIDAR, and satellites. To utilize satellite imagery, the group had to gain a baseline understanding of various satellite lenses and theory related to wavelengths. Understanding how the different lenses work, their purpose, and how to convert the satellite data required some proficiency in physics.

## 1.4 Existing software and technologies

The most similar existing product to IceMap is Iskart.no[1] published by NVE. IceMap is similar in its intended usage, but does not rely on user input. In researching Iskart.no and other existing products, the group discovered that the website utilizes an automatic calculation model, which the group came to implement in their system as well.

Several existing sensor solutions were considered for IceMap's specific requirements, but only a few were suitable. The most promising sensor technologies were experimental and not commercially available, while others were impractical for

[1] https://www.iskart.no/

use in water and outdoor environments. In researching additional data sources, satellite technology appeared as a viable option.

## 1.5 Limitations

Because the group was unable to acquire the necessary hardware within the project timeline, physical testing of sensor implementations was not possible. These implementations were instead tested on sensor output data from various online resources. This data was often not ideal for its intended purpose. LiDAR data and satellite imagery are rarely created specifically for the measuring and mapping ice thickness. Consequently, the group had to make use of data sources which were relevant, but not exact matches.

Usage of various APIs and relevant tools was limited by a budget of 0 NOK and the absence of data for Norway. Consequently, the group sometimes had to rely on test data from geographic regions with climates similar to that of Norway. Additionally, some sensor types lacked adequate documentation of the data format which they outputted, which made if difficult to develop data processing solutions.The difficulty of this task was greatly increased by the many limitations of the available data and technologies.

## 1.6 Contributors

In total, 3 parties were involved in this project: the groups three developers, the municipality of Gjøvik, and two academic advisors. Three meetings were held with the municipality during the project period. In these three meetings, the group gained all the necessary information from the municipality. Afterwards, the group focused on refining the task further and implementing the system as they deemed most appropriate. Since the municipality specifically expressed that the group could solve the task which way they pleased, both parties felt that there was little need in holding more meetings.

The two academic advisors, Pål Anders Floor and Marius Pedersen, were actively involved in the process. They both aided in ensuring a smooth progression and provided plenty of valuable ideas and feedback. Choices regarding sensors, application features, UI (User Interface) design, and the report were all made with the help of their contributions. Their expertise and guidance played a crucial role in the success of this project.

## 1.7 Report organization

Chapter 2 describes the self defined system requirements and design. Chapter 4 and Chapter 3 contain background theory. Chapter 5 describes the implementation of the mobile application and the design decisions from Chapter 2. Chapter 5 can be interesting to read even for those lacking understanding of programming as it delves into usability and design, but does also contain a lot of code and technical terms.

Chapter 6 is closely tied to Chapter 5 as it describes the process of creating the maps displayed in the application. Chapter 7 describe the concrete implementations of the ice and sensor theory described in Chapter 4. Chapter 5 describes the end use of all the data processing from Chapter 7. All of the code examples can be found in Chapter 5, Chapter 6, and Chapter 7.

# Chapter 2

# System requirements and design

This chapter covers decisions regarding UI design, some individual system components, and the overarching system design. The majority of these decisions were finalized during the planning and system modeling phases. For more details about how the project timeline was structured, refer to chapter 4 of appendix B Pre-project plan.

## 2.1 Task division

The task was divided into three main areas, each of which was assigned to a *one* group member. The first area was hardware, which was assigned to Hoa Ben The Nguyen. This included the research, selection, and implementations of appropriate sensors and drones. The second are was third party APIs and software, which was assigned to Joakim Aleksandersen. This area dealt with implementing third party APIs and software for non-sensor data sources, as well as research on the properties of how ice freezes. The third area was mobile development, which was assigned to Sara Savanovic Djordjevic. This area dealt with designing and developing the mobile application, and developing a custom solution for the color coded map. Many of the tasks conducted during the planning and system design phase were done collectively. Such tasks included developing the system requirements, domain model, and the database design.

## 2.2 System requirements

During the planning phase, the group focused on preliminary research and project requirements. The requirements where then iterated over as the group gained more knowledge of mobile programming and background theory. By the end of the system modeling phase, the requirements were finalized. Given that these requirements were established early in the project, they were designed to be flexible enough to accommodate new knowledge and ensure their feasibility. The requirements served as a framework rather than a rigid specification. Their flexibility was necessary to accommodate a "learn as you go" approach, as it would be difficult to learn *all* the necessary knowledge and skills before the implementation phase began. The following list shows the final iteration of the system requirements.

Sensors: At least one type of stationary sensors must be implemented, in addition to a moving solutions that can cover a larger area. The stationary sensor must be able to measure ice thickness under snow cover.

Unit tests: The server must include unit tests with a overall test coverage of 60% or more.

Mobile application: Must utilize some form of color-coded map to visualize ice thickness, as well as convey the uncertainty of the used data. The color coding must be based on data gathered by sensors as well as data from third-party APIs (Application Programming Interfaces). Such data may include temperature, precipitation, humidity, water depth, etc.

Response time: The map must be fully loaded within 5 seconds of opening the application.

Database: The system must be connected to a database that stores all sensor measurements with the coordinates and time of measurement.

Data exportation: Any user can export the data used in the calculation of the ice thickness. This includes measurements from sensors, supplementary data such as weather conditions sourced from third-party APIs, and satellite imagery.

The decision to aim for 60% unit test coverage was a compromise between achieving a high degree of test coverage and a consideration of the project's scope and time constraints. For a longer project period, larger team, or more experienced team, the required test coverage could be set higher. The mobile application was specifically requested by the municipality, while the decisions to include a color coded map and multiple data sources were decided by the group. The selection of a 5-second response time was somewhat arbitrary, based on the group's subjective assessment of what constitutes an acceptable loading time. The database was intended to server mostly as an archive for the collected data, and the data export functionality was intended to provide transparency for the end user.

## 2.3 Use-case diagram

The use case diagram on Figure 2.1 includes the central features of the application. Of these, three were chosen as strict requirements. The strictly required features included viewing a color coded map, viewing more detailed data in the form of graphs, and the ability to export the data used in the application. These functionalities were expected to be part of the MVP, but were not expected to be the only features. In addition to these, three additional features were added: search for other lakes, select areas on the map, and view more layers. Theses functionalities were added on as goals for the prototype, after the MVP was developed. Since the system does not rely on user input, verification of users was deemed unnecessary. Therefore, the use-case diagram does not differentiate between different user types.

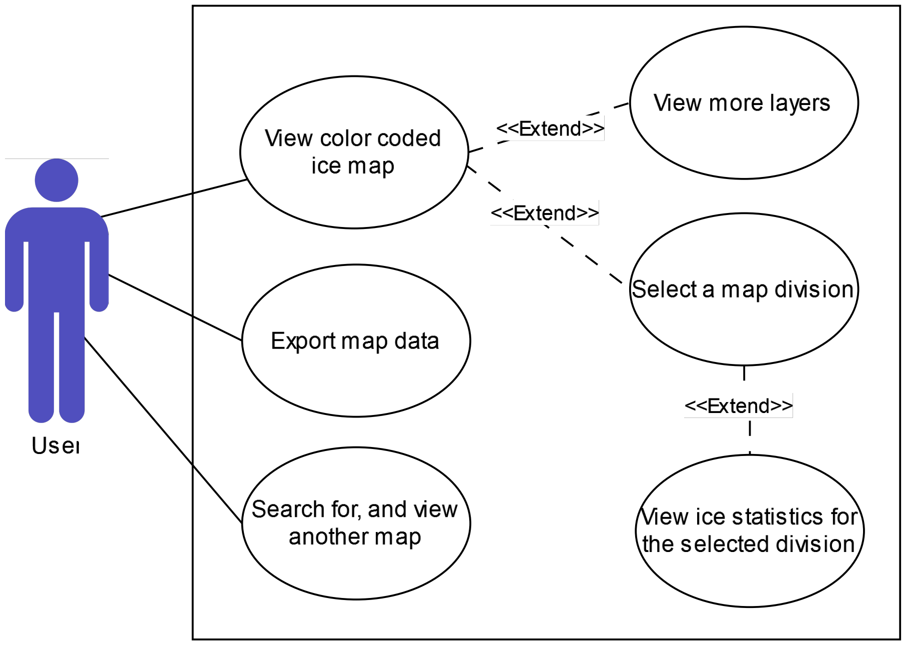



**Figure 2.1:** Use-case diagram with the six central features of the mobile application

## 2.4 Wireframes

The wireframes (see Figure 2.2) were made to map out the general layout for the main application page. The main components of the wireframe included the color-coded map, a widget containing various graphs and statistics, and a smaller overlay graph/box for quick access to key map data. The graph and statistics widget aimed to display detailed information about the state of the ice, weather conditions, measurement timestamps, and measurement methods. The wireframe served as an active design reference throughout the entire development phase.

## 2.5 Design reference

To set a goal for the complexity of both the design and the features of the IceMap application, the group looked at the My Aurora Forecast application by jRustonApps B.V. [1] for inspiration. The design of this application was simple and neat, and provided a good reference for some features like searching, sharing, and map views. The design was not intended to be replicated, but rather acted as an anchor to prevent the group from undertaking implementations that were far beyond their skill level. This approach aimed to ensure that the application would have a robust implementation with a simple, but aesthetic design.

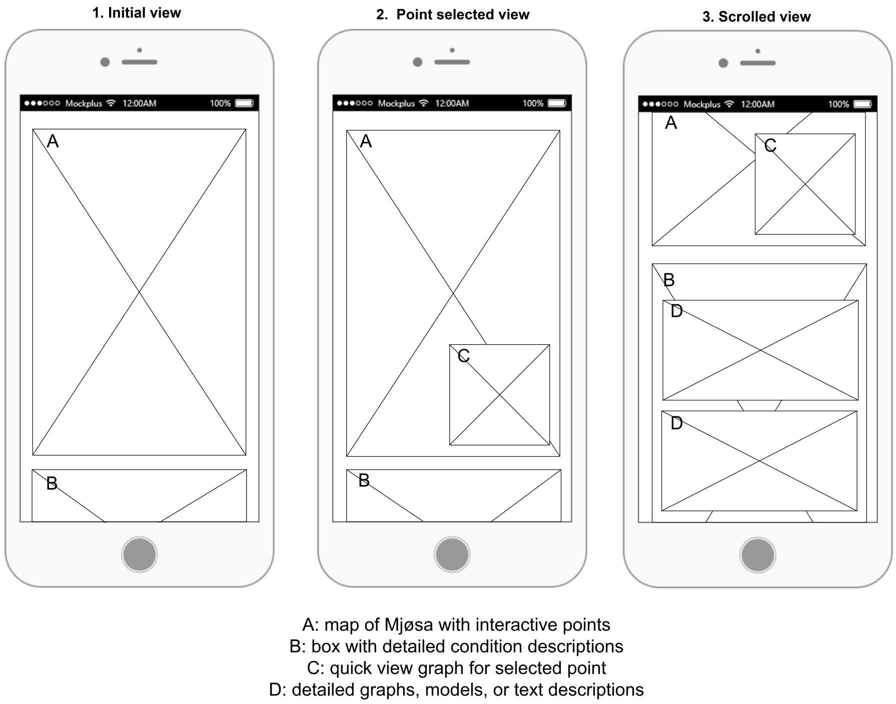



**Figure 2.2:** Wireframes

## 2.6 Database design

A SQLite database was put in place to archive the processed data. The database consisted of four entities. Each `Measurement` entity is uniquely identified by an unique Identification (ID) and a timestamp indicating when it was taken. Sensor entities are identified by a unique ID number, and are connected to a single body of water. Each `Measurement` has an affiliated `Sensor`, `BodyOfWater`, as well as multiple `SubDivision` entities. While the `Measurement` entity contains general information about the entire measurement, `SubDivision` entities contain data for a specific point within the affiliated `Measurement` area. Figure 2.3 shows each entity, their affiliated attributes, and the relations between all the entities.

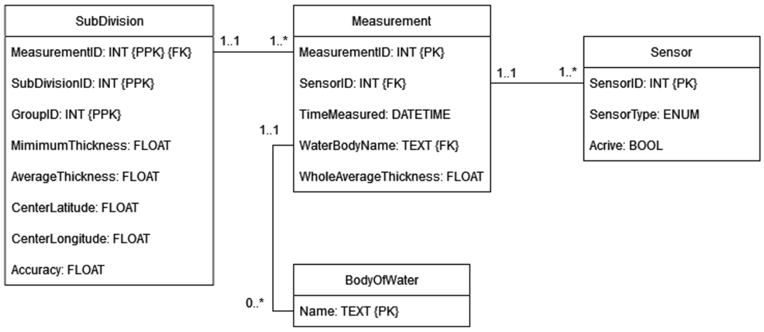



**Figure 2.3:** Entity relation diagram for archival of processed data

The choice of SQLite for the DBMS was primarily intended for the production stage, but while the system only includes a few lakes and sensors, SQLite may be sufficient. If the system were to expand beyond the reach of Gjøvik, it would probably require a more comprehensive DBMS like MySQL. As the database primarily served an archival purpose, and the system was intended only for Gjøvik, there were few performance requirements. Hence, the choice of DBMS was not considered thoroughly.

## 2.7 Domain model

The domain model on Figure 2.4 was made during the earlier stages of the system modeling phase. The model shows how the group initially envisioned the system to function. Four data source were expected, each providing different types of data required for the accurate assessment of the ice thickness. The standardization model was intended to process all the input data to a standard format. The standardized data would then be passed to a calculation model, which would output a safety rating calculated based on all the input data from the standardization model. The inclusion of the standardization model was a bit redundant, as the standardization of data could be done in each data input point or in the calculation model. The domain model was purely intended to aid the group in designing the system prior to implementation, and was not utilized as an active design reference. Therefore, the implementation does not match the domain model perfectly.

## 2.8 System architecture

A shown on Figure 2.5, the system was divided into three parts: an app, server and various data sources. The data sources include all the system sensors and two third party APIs. Most of the interesting application code is located in the Lib directory at the root level of the App directory. Lib includes the entry point,

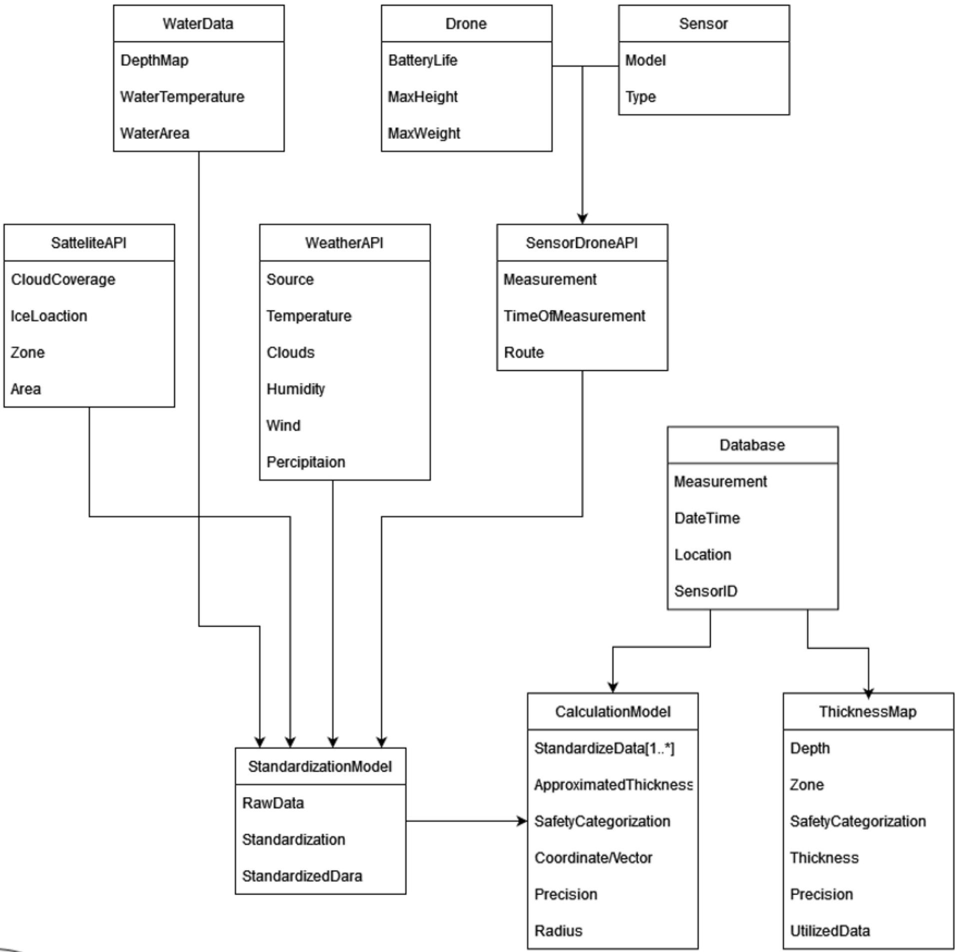



**Figure 2.4:** Domain model

a file containing constants, and a file containing self defined data structures. The constants file contains paths to folders and assets, as well as variables which are read (not written to) globally. The pages, widgets, code handling server requests are separated in dedicated folders. Most of the other code outside of Lib is auto generated boiler plate code for various operating systems and platforms.

The entry point for the server is at the root level, together with a constants file. The constants file mostly contains paths to various files which are used across multiple folders. The database folder contains the SQLite database contents and schema. The DataProcessing folder contains code for processing LIDAR data. The MapHanlder contains code for creating lake maps and updating data for the maps. The NVE model contains a third-party software from NVE for generating ice statistics. This model is described in more detail in section 3.4. The certificates folder contains SSL certificates and key files for the server-app communication.

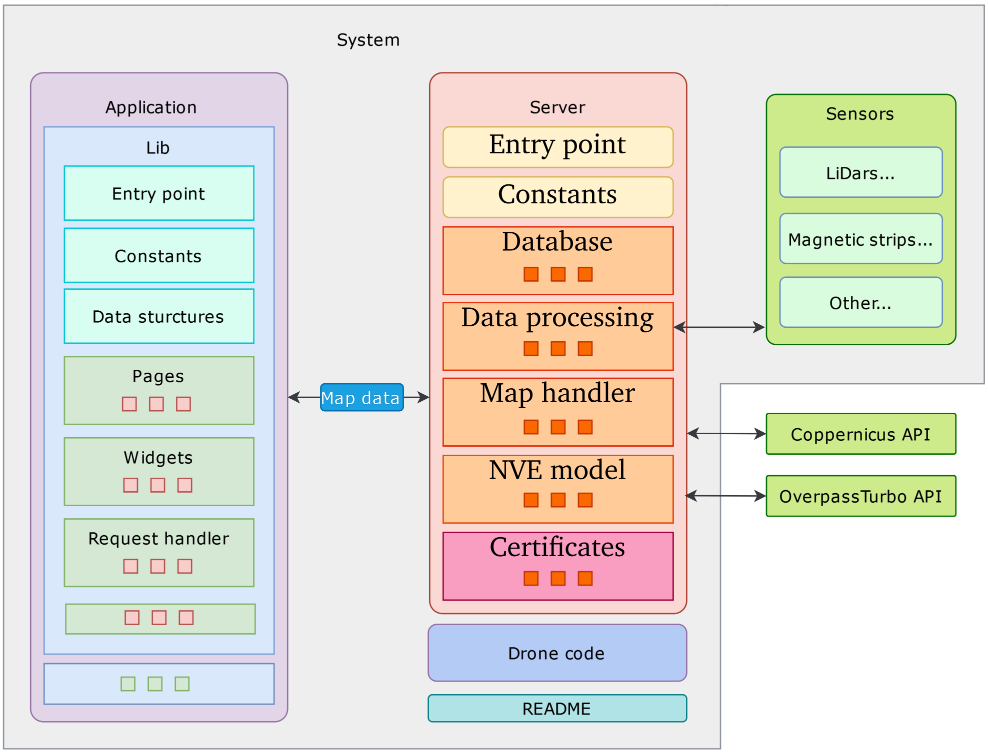



**Figure 2.5:** This system architecture diagram illustrates the general folder structure of the system.

## 2.9 Selected programming tools

The following programming languages, frameworks, and development environments were deemed most fitting for implementing the requirements:

### 2.9.1 Python and Pycharm

There are several reasons that made Python the ideal choice for the server. The language has simple syntax, a long list of libraries, is documented extensively, and has a very active user community. The user community provides a large collection of tutorials and code examples. Python also has many libraries for data visualization and manipulation of geometrical data, such as Shapely, GeoPandas, and Matplotlib. Pycharm was selected as the most suitable IDE (Integrated Development Environment), mainly due to the groups prior experience with IntelliJ's other IDEs.

### 2.9.2 Dart and Flutter

The two main contestants for the mobile application were React and Dart/Flutter. Eventually, React Native was ruled out due to having a JavaScript framework, something the group was not particularly fond of. Since the group also had some introductory experience with Kotlin, Dart seemed easier to learn. Eventually, the combination of Dart and Flutter was chosen. The advantages and disadvantages of Dart and React Native were discussed in the appendix B Pre-project plan.

### 2.9.3 Android Studio

Android Studio was selected as the development platform for the mobile application. As with Pycharm, it was chosen mainly due to the groups prior experience with the IDE. Android Studio comes pre-equipped with multiple free Android emulators, and has a built in interface for ADB (Android Debug Bridge) connection. This allows physical devices to quickly be connected to a PC and to transfer the application wirelessly.

# Chapter 3

# Ice theory and selected APIs

Given the project's focus on ice safety and measurement technologies, a grasp of relevant ice-related theory is essential for understanding many of the decisions made by the group. The following theory will aid in the comprehension of choices of sensors, other hardware, APIs, and mathematical formulas implemented in the server.

## 3.1 Ice formation theory

Knowing when ice initially manages to establish itself on lakes is what is most important when it comes to estimating ice thickness. This is due to when ice forms on the surface of a lake, it grows thicker as heat moves from the ice's bottom to the air above. The bottom of the ice is always at the freezing point. If no extra heat comes from the water below from residue heat, all the heat loss helps the ice to grow thicker, from the bottom and down. The formation itself might not happen before the average daily temperature falls below the freezing point as the temperature needs time to cool down the residue heat inside the lake [2].

When ice sheets grows, they grow from the bottom of the sheet and downwards. This growth is caused by the air temperature being sufficiently low to reduce the water temperature below the freezing point. As ice layers and snow establish themselves on the water, they effectively act as a thermal insulator between the air and the water [3]. This insulating effect can prevent the ice from thickening, even when temperatures are below 0 °C. The underlying waters warmth can also contribute to the ice formation plateauing.

Under normal conditions, water will freeze at 0 °C and have maximum density at 4 °C. This causes water at greater depths to converge towards 4 °C. This effect allows water at deeper levels to remain above freezing temperatures, even when the surface water has frozen.

The density of water decreases when it cools, causing colder water to rise and replace warmer surface water. This creates a layer closer to the surface where the water is colder than 4 °C and a layer where the temperature is at 4 °C. These layers will slowly try to balance each others temperature out. Since water has a maximum density at 4 °C, some may believe that the water temperature will stay at 4 °C at specific depths, but this is only true in certain cases [2]. Strong winds can cause surface water and deeper water to mix, which can prevent the surface water from reaching the freezing point and blending the heat causing the temperature to be almost equally same throughout. Lake Erie in North America is one such example. Here, wind effects are so great that stable ice rarely forms over the entire body, despite the water being nearly 0 °C throughout the entirety of winter.

## 3.2 Icetypes

Ice is mainly categorized into three types: black ice, slush ice and spring ice. The ice types have different properties and appear at different times.

#### Black ice

The first ice type that establishes itself on an lake is black ice. Black ice is characteristic for its transparent and seemingly black color. It is the strongest ice type and can almost carry an average adult at only 5cm thickness [4].

#### Slush ice

Slush ice is formed when a layer of wet snow freezes. When the snow weighs down the underlying ice, the ice can crack and let lake water seep through. This is usually how the snow becomes wet. The layer of slush freezes from top to bottom, which can result in a layer of wet snow sandwiched between a layer of black ice at the bottom, and a layer of frozen slush ice at the top. The color of the ice is an indicator of its strength. Slush ice is characteristically white or grayish, which is caused by its high content of air. The higher the air content, the whiter the slush ice. A lower air content and a higher water content will causes a darker, grayish color. Less air and more water usually creates stronger ice, but this is only true if the slush is already frozen. Wet slush also has a gray color, but does not have any carrying capability.

If the slush ice has not bonded with the layer below when new snow falls, new layers of ice can form on top of the existing ones. This can result in a stack of multiple slush layers which are mostly composed of surface water. Such an ice formation is not particularly strong despite being thick. In extreme cases, there can be meters of slush ice layered on top of a base of black ice. Slush ice can grow incredibly fast if there are no insulating layers on top of it.

**Spring ice/candle ice**

During spring when the temperature rises, the ice will slowly become hollow. As the daily air temperature rises above the freezing point, the ice will start to melt. While the ice becomes thinner, the ice crystals will start to break down at their edges and form into vertical rod-like structures. These long vertical crystals have weak connections between each other. As the crystals gradually become hollow, they fill with water. Such formations greatly diminish the carrying capacity of the ice, even when the ice is thick. As this ice type usually forms in spring, it is called "spring ice", but it is also known as "candle ice" due to the rod-like crystal [2] [5].

### 3.2.1 Determining effective ice thickness

As different types of ice form on the water, they are not equal in their carrying capacity. If there are multiple layers on the ice, the strength of the ice is only as strong as the strongest layer and not the total amount of ice thickness. This leads to significant doubt about slush ice and candle ice, as it's uncertain whether the layers have merged and how strong the bindings are [6]. Because of this, a choice have been made to make the effective ice thickness always be defaulted to the black ice as it is the strongest and most predictable ice. Despite this choice, slush ice and candle ice still does have an carrying capability, but this should be primarily be assessed by personnel with prior experience with determining ice thickness through manual labor [7].

## 3.3 DOT guidelines for ice safety

The Department of Transportation of the Government of the Northwest Territories (DOT) has guidelines for determining safe ice bearing capacity in a document for safe ice construction [8]. The document is mainly meant for maintenance and construction of crossings over ice with vehicles, and will thus be mainly used for ice at thicker values (effective ice thickness of >10 cm).

The DOT have three operation conditions. These are routine, enhanced and acute operation. The routine is the most common and simplest to conduct, but also the most conservative estimate. Acute is the least common, but also the level requiring the most effort to commence. The operating conditions are applied in various ways over three phases of operation. These are divided into Pre-construction, construction and operation phase.

In the scope of this project, the phases and conditions that will be considered will be restricted to the routine conditions for all phases. This is because it is the most conservative, thus ensuring higher levels of safety despite the potential unreliable/sparse data that are available. This said, it is important to remark that it is not possible to achieve a routine operation level with only use of already existing sensors technology as one of the requirements for routine operation level require manual labor and supervisors.

The Routine conditions that are applicable in the scope of the project are as follows. In all states of operation, Gold's formula (Eq Equation 3.1) should be calculated with an $A$ value of 4. For calculating the Gold's formula, the newest data should be used. The spatial distance between every measured area should be within 12 meters of each other. There should be 10-14 days of ice thickness measurements available.

### 3.3.1 Gold's formula

The first step in routine and enhanced operation levels is to utilize Gold's formula (Equation 3.1). Golds formula is used for estimating the bearing capacity of an ice sheet [8]

$$P = Ah^2 \tag{3.1}$$

$P$ is the estimated bearing capability of the ice in kilograms. $A$ is an value between 3.5 and 6 which is chosen to set how conservative the model is. 3.5 is the most conservative and 6 is the least. $h$ is the thickness of effective ice at its lowest point.

Gold's formula can be a good estimator for general bearing efficiency for vehicles. Canada has historically used this formula for estimating the safety of ice for work environments. The model originally published in 1981 is not recommended to be used on current day heavy equipment load configurations, as the model made by Gold does not resemble the same general load configurations of today's standard [9]. Thus Gold's model should not be used in contemporary organizations when determining required ice thickness for such equipment [9], but can still be for smaller general use case.

To achieve routine safety by the DOT, the Gold's formula should be done with an $A$ of 4 in all phases of operation. Hence when using the Gold's formula, the $A$ value will always be set to 4. As 5 cm of effective ice is the estimated thickness necessary for "navigable ice" [7], Gold's formula can calculate the estimate carrying capability. Calculating a maximum capacity of 100 kg for this thickness.

## 3.4 NVE calculation model

In order to get an estimated ice thickness for any given coordinate in Norway, a model and software provided by the NVE was utilized. This same model is used by a NVE in their "iskart" application[10]. The model uses air temperature and weather data from SeNorge's API to conduct an ice prognosis. SeNorge is an gridded dataset of daily aggregated temperatures of the Norwegian mainland from 1957 to present day[11].

When using the model, it creates a prognosis for the entire season worth of data. This is done by requesting data from the closest available point from seNorge to get the weather and temperature information. The model iterates through the entire selected timeframe by calculating the evolution of the ice. The model determines if the ice is freezing or melting by the temperature and snow conditions for each time step. Specifically, the model checks if the air temperature is below freezing to simulate ice formation and above freezing for ice melting scenarios.

This day-by-day iterative process involves calculating the necessary heat transfer to determine the changes in ice thickness. The model adds new snow layers and updates the slush levels if applicable. During freezing conditions, it calculates the amount of ice formed by considering the heat flux through the ice column using the thermal conductivity of the layers. If melting occurs, it uses a degree-day model to estimate the reduction in ice thickness, as shown in Equation 3.2.

$$\Delta h = \text{melting coefficient} \times \text{Degree Days} \tag{3.2}$$

The equation uses the unique melting coefficients of each type of snow and ice respectively. It multiplies this with the time step and the temperature above freezing. This simplifies the complexity of melting. These detailed calculations make the model relatively time-consuming if run directly in the front end.

The freezing process involves updating the total ice by adding new ice layers based on the calculated heat flux and thermal conductivity from the top to the bottom of the total. During melting, the model removes or reduces the thickness of the top layers using different melting coefficients for snow, slush, and black ice, depending on the temperature and time step.

As ice thickens and unforeseen variables affect the physical ice which the model cannot accord for, thus the model becomes more and more deviant over time. This is something that can be counteracted by including ice run dates, which are dates where the water is known to have a layer of ice. These dates are manually given in order to correct the model. Supplying ice run dates also allows for the most accurate estimation as the dates for the initial ice layer are crucial for having a most accurate estimation.

When utilizing this model, it is imperative to consider the assumptions taken and uncertainties underlying in the model. The model itself allows the use for average water temperature and ice run dates, but these have to be manually provided to the model. To further improve the model an attempt to utilize the SENTINEL-2 in order to provide the ice run dates have been used as NVE themselves already use SENTINEL-2 to create iskart.no, but apparently not in the act of creating ice run dates[10]. More information about SENTINEL can be found in section autorefSent-2-sat Water temperature is something that in the future can be improved by the use of magnetized strip sensors as discussed in this section section 4.6.

The NVE model is supposed to act like a baseline or fallback strategy in the IceMap system. By default, if a subdivisions includes sensor data, the sensor data will be utilized to determine the ice thickness of the subdivision. If a subdivisions has not been measured by a sensor, it will utilize the data produced by the NVE model. chapter 5 describes the end usage of the different data sources in more detail. The selected sensors, which are described in chapter 4, should gather data with a higher accuracy, but also requires more effort to deploy. Since smaller and more remote lakes may not have the capability to deploy a drone or even have stationary sensors, the NVE model provides a baseline for such cases. To mitigate the lower accuracy of the model output, subdivisions with only NVE data are marked in the IceMap application as being less accurate (see chapter 5). The model also works as a fallback strategy when the drone fails or goes out of commission.

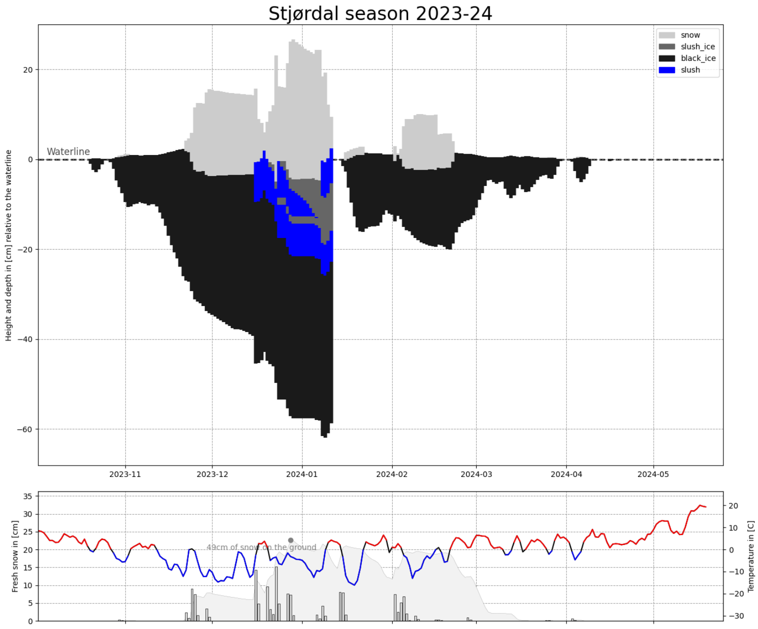
## 3.5 SENTINEL-2 satellites

The problem of having to set ice run dates manually can be solved by integrating the SentinelHub API. The SENTINEL-2 is part of the Copernicus Programme, which is coordinated by the European Union and specifically tailored to provide comprehensive and sustainable Earth observation capability [12]. SENTINEL-2 is a satellite mission with two identical satellites, SENTINEL-2A and SENTINEL-2B, which were launched in June 2015 and March 2017 respectively. These satellites are equipped with high-resolution multi spectral cameras designed to capture visible, near-infrared (NIR), and short-wave infrared (SWIR) images. [12]

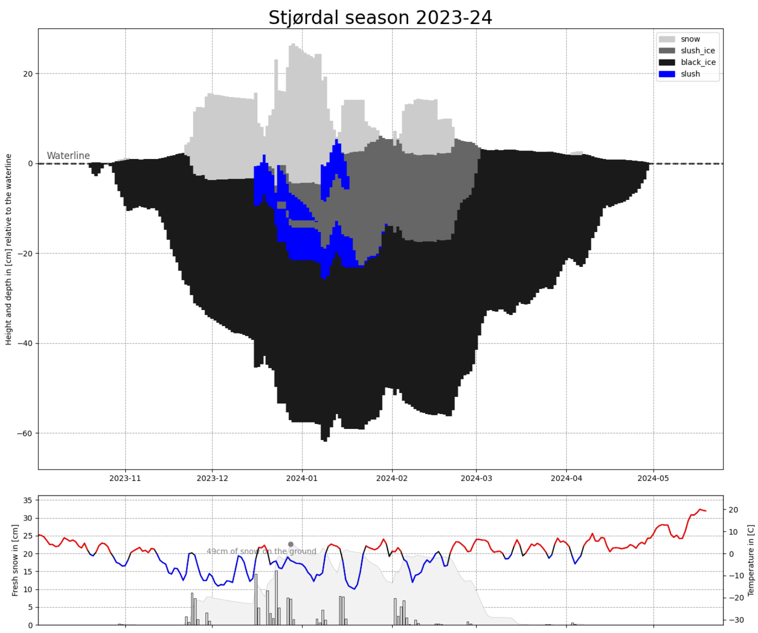
The objective of these satellites is to monitor surface condition and variability. They achieve this through their high revisit frequency. Each satellite revisits every 10 days, effectively every 5 days when combined. Additionally, with a swath width of 290km, they can effectively revisit areas at mid-latitudes every 2-3 days [12]

SENTINEL-2 is equipped with 13 spatial bands: four at 10m, six at 20m and three at 60m spatial resolution [12]. This causes the images captured by SENTINEL-2 to be highly versatile for various observational purposes. The different spatial resolutions capture detailed observations of vegetation, soil, and urban areas, and most importantly bodies of water. This allows the SentinelHub API to be utilised for observing the ice state on lakes.

SENTINEL-1 and SENTINEL-2 have previously been utilized for monitoring and forecasting river ice conditions to help manage flood risks. SENTINEL-1, with its Synthetic Aperture Radar (SAR), was used to detect and differentiate types of ice formations along the river. SENTINEL-2, was utilized to view the flooding




**(a)** Output **with** icerun dates




**(b)** Output **without** icerun dates

**Figure 3.1:** Examples of NVE model outputs with and witout icerun dates

and the status of the ice by providing high-resolution imagery in various spectral bands [13].

The SENTINEL satellites provide valuable insights into ice conditions, demonstrating that visual appearances alone are not enough to understand the full story of ice duration. Appendix D SENTINEL hub, Mjøsa over time features multiple images of Lake Mjøsa taken over a 30-day period, captured using true color lens as well as the use of a modified multi temporal evalscript [14]. The evalscript view the Normalized Difference Vegetation Index (NDVI) and the Normalized Difference Water Index (NDWI) in order to highlight the difference between snow and water or vegitation and water respectivly. NDVI and NDWI are variations of the *normalized difference index or ratio formula* equation (Equation 3.3).

$$\Delta = (a - b)/(a + b) \tag{3.3}$$

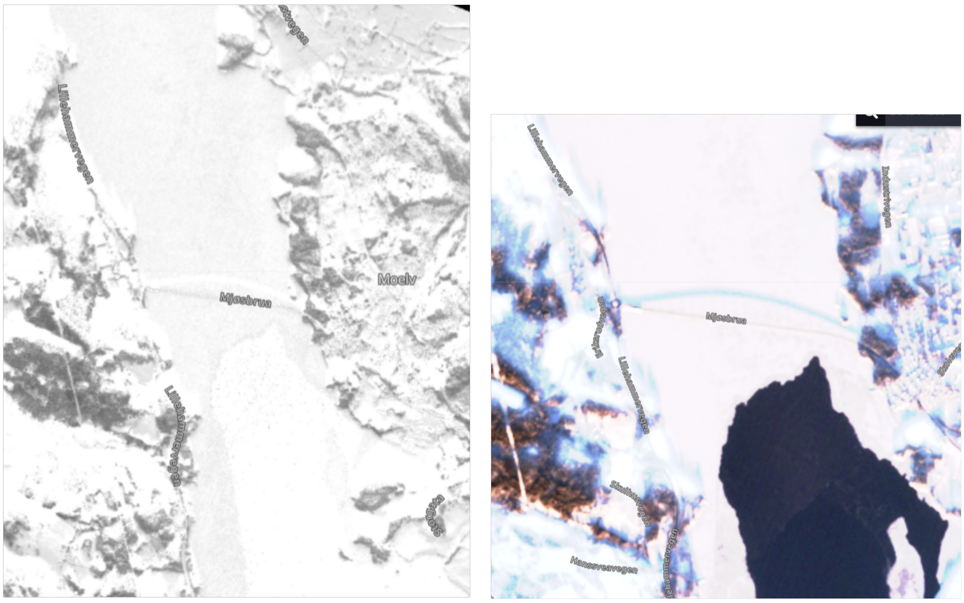
NDVI uses $a$ for NIR and $b$ for red, while NDWI employs $a$ as green and $b$ as NIR. This help to illustrate the duration of ice presence on the lake. Despite the ice appearing similar in the images, significant differences in its longevity are evident. This matches section chapter 4 about how the lake's conditions affect the ice. Mjøsa has greater water depths and a river running, see appendix E Depth map, Mjøsa, causing currents in the area where the ice spent more time to pass. With this in mind the proposed usage of the SENTINEL framework in this project would thus be to get the ice runs for given areas on the ice in order to better suit the model given by NVE specific areas on lakes. Specific evalscripts are found in appendix F Evalscript using 30 days NDMI, NDWI and NDVI and appendix G Evalscript using 30 days NDMI.

The standard deviation between pixels on the satellite imagery can be used to differentiate between areas of water from areas of ice and snow. It is proposed to use the Icy Lake Index (ILI) equation (Equation 3.4), which uses a variation of the normalized difference index or ratio (Equation 3.3) with satellite bands as:

$$\text{ILI} = (\text{Red} + \text{SWIR2})/(\text{NIR} + \text{SWIR}) \tag{3.4}$$

The ILI as described in this study [15], it was attempted to classify land, snow, ice and water. They were able to achieve 94.5% precision for ice with a Support Vector Machine(SVM). Though it is a higher complexity project that also includes way more variables, this is what is important for lake ice classification. Figure 3.2a is a picture taken with the use of the ILI, the same picture but as true color is shown in Figure 3.2b.

By using the statistical API from SENTINEL hub, it's possible to run an evalscript on specific areas over a designated time interval, eliminating the need to download all the images. By specifying a Bounding Box(BBOX) for the interior of




**(a)** Picture using Ice Lake Index **(b)** Picture using true color

**Figure 3.2:** SENTINEL-2 L2A pictures of Mjøsa using SENTINEL hub EO Browser

a lake would thus have different standard deviation based on if there are ice or not, ultimately thus attaining when the ice is on and off the water.

Figure 3.4b shows a plot of Mjøsa utilizing data from the statistical API. The plot was generated from multiple months of data where the cloud coverage was below 40%. As the precise numbers used in the previously mentioned article [15] were not public at the time of writing this report, the specific evalscript is written based on the ILI formula and might not be the same as the one used by Jugier. Using the pure ILI in an evalscript yielded lacking results, which can be seen on Figure 3.4b where the standard deviation never goes beyond 1. This said, it can easily be mitigated by multiplying the formula by a single factor to increase the contrast. This can be easily visualized in Figure 3.3, which shows that there is a clear difference between water and ice.

In the graph, it is clear that the standard deviation is far lower when ice is present. The 23rd of December 2023 is the first date where the cloud coverage is low enough to get a clear picture of Mjøsa with the presence of ice. In this instance, the standard deviation is 2.58, in contrast to the deviation of 12.65 recorded on 2023-12-03. For more detailed data, see appendix H CSV with Labeled Data.

This difference in deviation makes it possible to distinguish between days with ice ('ice on') when deviation is low, and days without ice ('ice off') when deviation is high. However, this method does have some limitations. Clouds can affect the

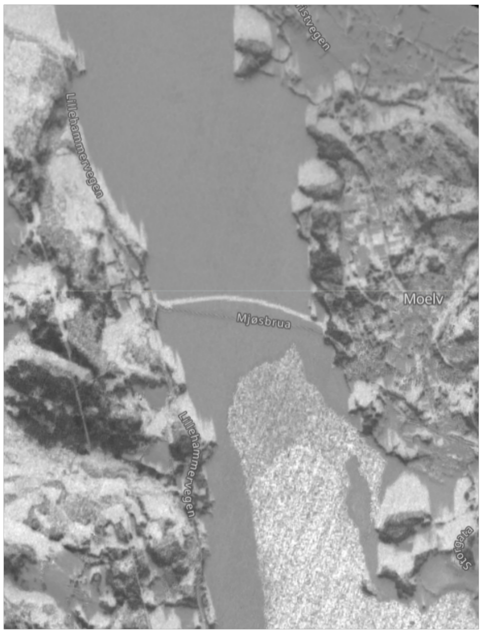
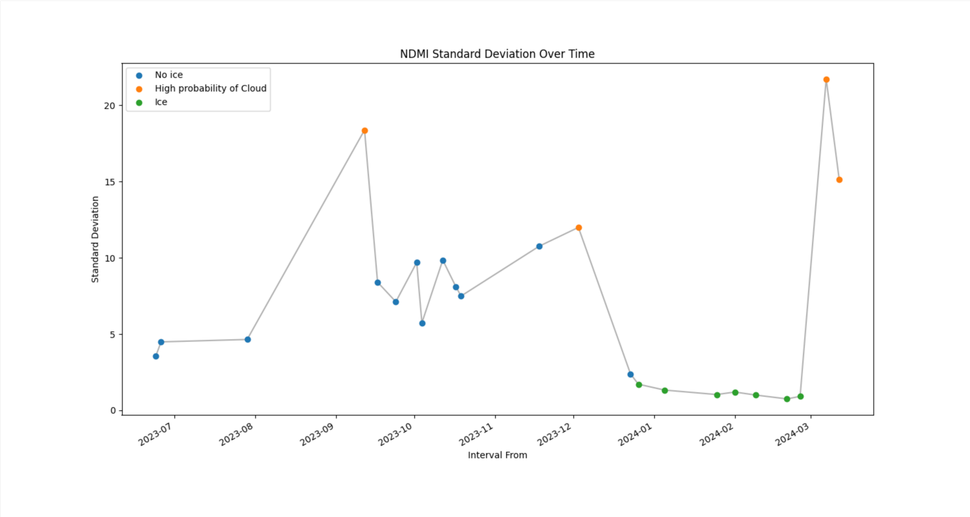


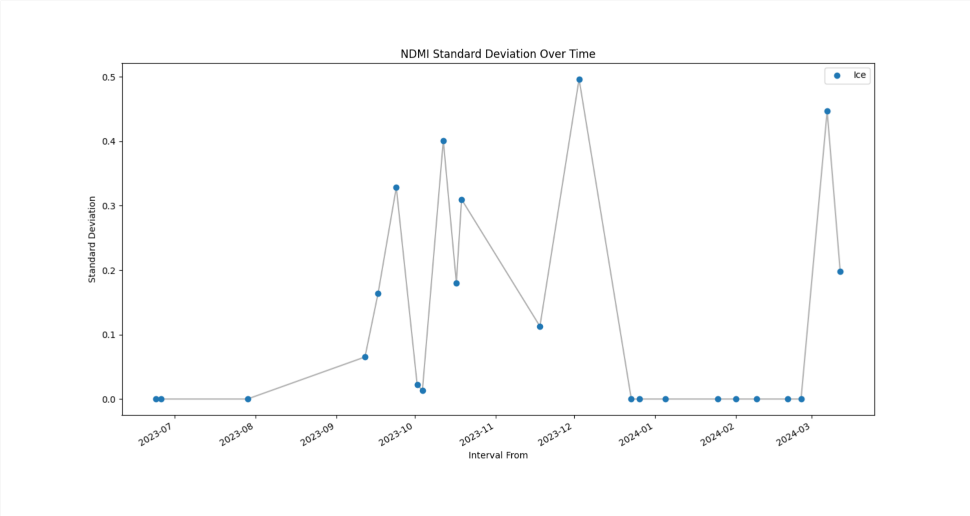

**Figure 3.3:** Picture taken by SENTINEL-2 l2A 2023-12-26 using ILI with a contrast factor

accuracy of the satellite images. It is possible to filter images from Sentinel Hub by percentage of cloud coverage, but it is not perfect. Since higher cloud coverage leads to less reliable data, a more strict filtering for cloud coverage could increase the reliability. However, this could lead to large time periods with insufficient data. Figure 3.4a and Figure 3.4b show an example where the API did not output any data between August and October due to a strict cloud coverage filtering.

Another problem with using standard deviation to tell ice, water, and clouds apart is when the ice starts breaking, like in Figure 3.5. This approach drastically increases the standard deviation, incorrectly marking it as clouds. This mistake commonly occurs during spring and off-seasons. Mistaking ice during off-seasons isn't a particularly big problem because the model does not predict ice growth in warm weather.




**(a)** Graph of standard deviation with contrast factor




**(b)** Graph without adjusted contrast factor

**Figure 3.4:** Graphs generated with SentinelHub data with cloud coverage filtering

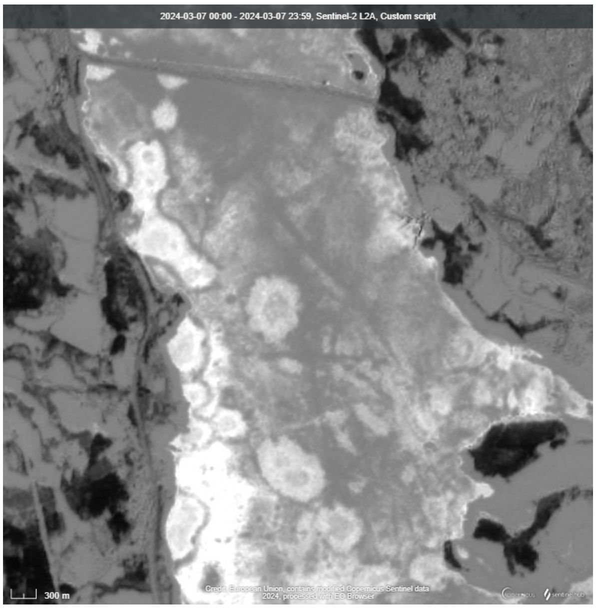



**Figure 3.5:** Picture taken from SENTINEL-2 l2A 2024-03-07 using ILI with an contrast factor

As a final note, when using the ILI, it is important to consider the size of the area being analyzed, especially when using standard deviation to classify ice. The evalscript (section 7.6) uses data from bands B04 (Red) and B08 (NIR) at 10m resolution, and bands B11 (SWIR1) and B12 (SWIR2) at 20m resolution. This means each pixel represents 10 square meters, but B11 and B12 overlap by 2 by 2 pixels. So, to get a meaningful standard deviation, it needs to capture quite large areas of the picture [16] as only every $20m^2$ are effectively unique pixels. Because of this, the BBox'es should be at least $km^2$ in order to have an adequate amount of pixels to do produce an effective evaluation.

# Chapter 4

# Selected sensors and background theory

The group chose to utilize a drone to operate a LIDAR (Light Detection and Ranging) system. This solution addressed the requirement of a mobile sensor. Integrating a laser altimeter onto the drone enables it to conduct area scans for thickness calculations, thereby evaluating the safety of traversing the ice surface. The installation of the laser altimeter on the drone is crucial to facilitate its scanning capabilities and ensure accurate assessment of ice thickness. To meet the requirement of a stationary sensor, the group chose to utilize magnetic strip sensors.

## 4.1 Laser altimeter theory

A laser altimeter is a variant of LIDAR that is primarily deployed for determining vertical distances from the altimeter to a target object. The technology utilizes laser to measure distances, generate detailed maps, and create intricate three-dimensional models of the Earth's surface.

Laser altimeters work by emitting a pulse of scattered light particles. The distance to the target object is determined by measuring the time it takes for the emitted light to return to the altimeter. This concept leverages the constant speed of light [18] and the second *equation of motion* formula, as outlined in Equation (4.1):

$$d = v_0 t + 12 \cdot a t^2 \tag{4.1}$$

In this formula, $d$ denotes the distance covered, $v_0$ represents the initial velocity at the commencement of the calculated distance, $v$ signifies the velocity at the conclusion of the distance calculation, $t$ denotes the time taken to travel the distance, and $a$ denotes the acceleration from start to finish.

While the concept relies on the constant speed of light, variations in the drone's speed and environmental elements introduces additional complexities to be con-

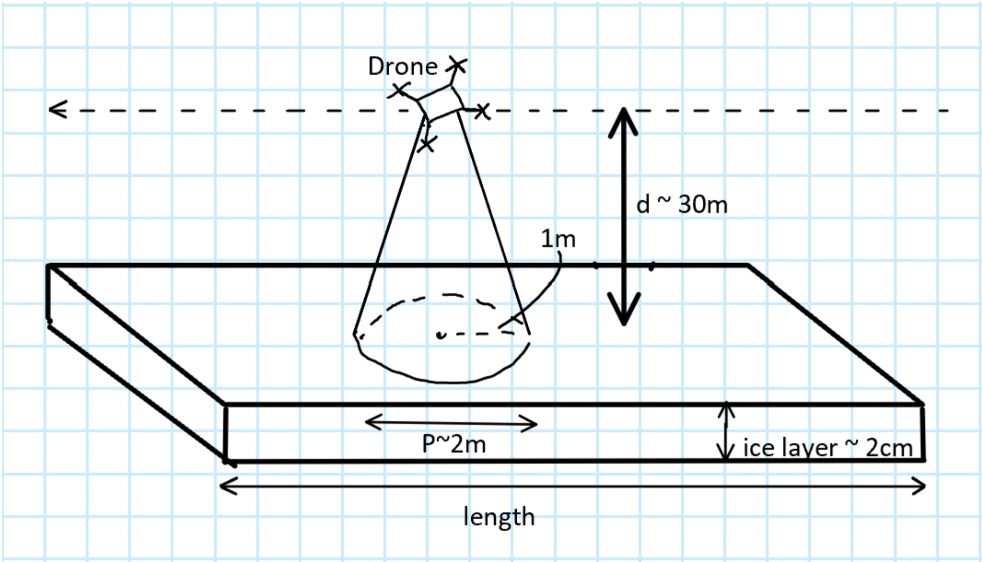



**Figure 4.1:** A typical setup of proposed ice thickness measurement: An airborne LIDAR illuminates the surface with tightly focused laser beam and measures the light returning. Reproduced from [17].

sidered. Despite these potential challenges, the position of the target object relative to the altimeter is determined based on the direction and distance of the returning light. This data can be mapped onto a 3D coordinate system, with supplementary data depending on the type and sophistication of the LIDAR system.

The thickness of ice can be determine by leveraging the penetration capability of the emitted laser. This is done by ascertaining the distance between the reflection points on the top and bottom surface of an ice layer. Various studies have confirmed this method, including those by Sergey et al.cite[19], Fons et al.[20], and A. Gold[21]. These studies primarily involve space-borne laser altimeters for analyzing sea ice thickness and estimation of mass. However, it is important to acknowledge the inherent uncertainties associated with this data, which are influenced by variables such as humidity, snow, precipitation, and precision of the sensor apparatus.

The standard output of a LIDAR consist of multiple sets of XYZ coordinates, as shown in Table 4.1. Each set of XYZ coordinates represents a single point reflected from the ice surface. By utilizing the X and Y coordinate, points that are vertically aligned can be grouped together. By examining the Z coordinates and intensity (I) of these grouped points, the thickness of the ice layer can be discerned.

To examine the utility of LIDAR data, an open-source LIDAR file of the Arctic was obtained from opentopography.org [23]. Since no LIDAR data for Norway was publicly available, this data from the Arctic was thought to be adequate for

| X (m) | Y (m) | Z (m) | I |
|---|---|---|---|
| -25.6245 | 16.3245 | 8.2345 | 0.7435 |
| -25.5456 | 16.2655 | 8.3532 | 0.1535 |
| -25.5463 | 14.2542 | 6.1435 | 0.1453 |
| -25.5413 | 14.4235 | 6.3246 | 0.8874 |
| -25.4352 | 16.3452 | 8.4235 | 0.2546 |
| ... | ... | ... | ... |

**Table 4.1:** Example raw point cloud data from LIDAR scan.
Reproduced from: [22]

testing and demonstrating the LIDAR data processing. By converting and analyzing the LIDAR scan results from this source using CloudCompare, the graphical representation in Figure 4.2 of the terrain was generated.
The limitations of a LIDAR lies in its necessity for an unobstructed view of the ice.

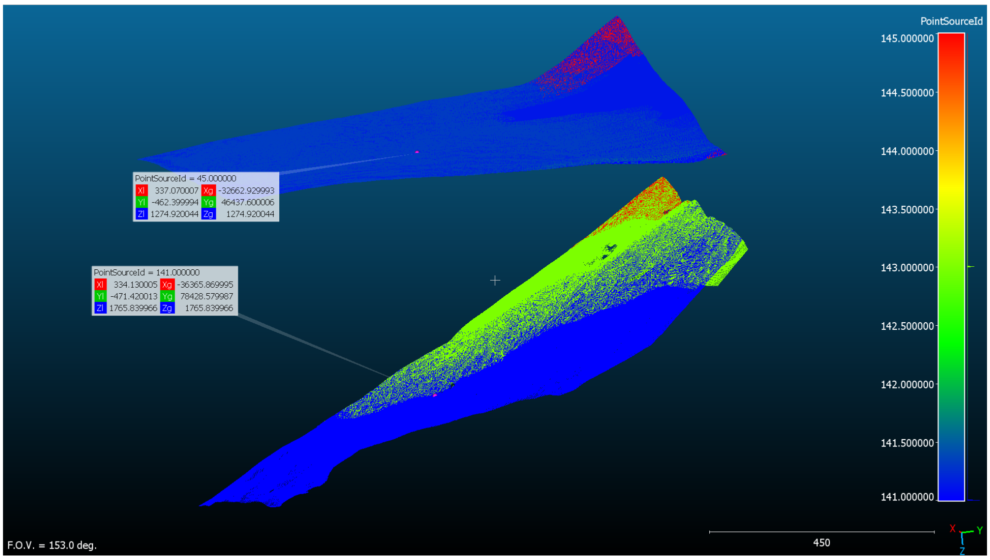



**Figure 4.2:** LIDAR data from Opentopography [23] plotted on a 3D map, showing how the points are set up using cloudCompare.

In cases where the ice is obstructed by for example snowfall or rain, the measurements will be inaccurate. Determining ice thickness requires evaluating physical characteristics inherent to both sea ice and freshwater ice, thereby presenting challenges to monitoring endeavors[24].

## 4.2 Selected laser altimeter

The specific laser altimeter model chosen for the IceMap application is the SEN-14032 from Garmin[1]. Multiple other models were considered, but the SEN-14032 ultimately met the groups requirements and limitations best. While many commercial altimeters offer extended range capabilities, they come with a high cost. Moreover, the majority of available sensors possess limited operating temperatures, rendering them unsuitable for colder environments. It was only upon discovering the SEN-14032 model that the group found a sensor capable of measuring distances of up to 40 meters, operating in temperatures as low as -20 °C, maintain a compact dimensions of 20 x 48 x 40 mm and deliver a precision of +/-2.5 cm. Despite the modest accuracy, it adequately fulfills the requirements for determining the minimum ice thickness necessary to support multiple individuals, as detailed in section 3.3.

## 4.3 Selected drone

The drone planned to be used for this conceptual process was initially the DJI Matrice 100, a developer drone made by Da-Jiang Innovations (DJI). This drone provides several customization options allowing the addition of third party hardware and software. The DJI Matrice also includes its own compatible hardware and software applications for smooth operation of the drone. The drones manual[2] describes functionalities like waypoints navigation for automated flights, point of interest (POI) for operating on specific coordinates, and some other functionalities which are relevant for the IceMap system.

The DJI Matrice model seemed to be a great fit upon initials research, until discovering that its minimum operational temperature was -10°C. Considering that Norwegian winters commonly reaches temperatures below -20°C, this posed a significant limitation. A newer model of that same drone, DJI Matrice 300, was later discovered to be a better fit. This updated model offers comparable or superior specifications, the same software and hardware benefits, and is capable of operating in temperatures as low as -20°C. Hence, the DJI Matrice 300 was selected to be utilized in the conceptual proof of the system[3].

A SWOT analysis was conducted to evaluate the multiple viable options for the drone. The analysis highlights strengths, weaknesses, opportunities, and threats associated with the different options. These factor are crucial for making an informed decision that align with the project's objectives and requirements. Table 2 in appendix shows how the SWOT analysis: DJI Metrice 300 compares to other

[1]https://www.garmin.com/en-US/p/557294specs
[2]https://dl.djicdn.com/downloads/m100/M100_User_Manual_EN.pdf
[3]https://dl.djicdn.com/downloads/matrice-300/20200507/M300_RTK_User_Manual_EN.pdf

drone options which were considered.

The SWOT analysis reveals several important insights regarding the drone selection process. While certain drones offer strengths like cost-effectiveness and high payload capacity, they may also present limitations such as limited flight time and the need for additional hardware. Opportunities exist for leveraging third-party software and hardware integration, but potential threats such as technical issues and software bugs must also be considered. It is essential to weigh these factors carefully and consider potential strategies to mitigate weaknesses and capitalize on opportunities to select the most suitable drone.

## 4.4 Integration challenges

The group assumed that integration of the selected drone and sensor would be possible, but could not guarantee its feasibility due to lacking expertise in both drones and sensors. However DJI offers a payload SDK for integrating tools like LIDAR into their drones [4]. If this option proves unfeasible, it is suggested to consider a LIDAR provided by the drone manufacturer. DJI provides multiple options for LIDAR implementations, however, these LIDARs come at a much greater cost than the ones proposed by the group, and may not meet the operational temperature requirements. Nonetheless, the other specifications outlined in section 4.2 generally meet or exceed the groups requirements for these sensors.

## 4.5 Alternative approaches

The original plan involved utilizing any inexpensive drone with specifications that met the minimum requirements. In order to accommodate an automated flight, an Arduino board was proposed to be mounted on the drone together with the sensor. Later in the project period, the group discovered the DJI Matrice 300 which allowed for a much simpler implementation of flight path automation. The code intended for implementing the Arduino remains available in the GitLab repository for those interested in a budget-friendly setup. Additionally, code and links to hardware resources are provided. The necessary components include an Arduino MKR Zero and NEO 6m (GPS device), along with equipment for mounting the component on the drone. Wiring instructions for the GPS system and LIDAR can be found in the tutorials for NEO-6m[5] LIDAR[6]. Although, it is not cost-efficient given the need of a professional-grade drone capable of operating at a minimum temperature of -20 °C, which typically includes their own integrated hardware and software solutions.

[4] https://developer.dji.com/payload-sdk
[5] https://randomnerdtutorials.com/guide-to-neo-6m-gps-module-with-arduino/
[6] https://learn.sparkfun.com/tutorials/lidar-lite-v3-hookup-guide/all

### 4.5.1 Air planes

Air planes were also considered as an alternative to using drones. Multiple air planes commonly fly over Gjøvik and Mjøsa. These could theoretically be used to mount a LIDAR, but would presented multiple challenges. Specifically, air planes lack the necessary agility to navigate smaller bodies of water, meaning this solution would only work for Mjøsa. As the municipality probably does not have the funding to buy and operate an air plane, the group considered utilizing already deployed air planes. The local hospital and weather station do fly smaller air planes over Mjøsa, however, the lack of a consistent route would complicate the measurement process. Hence, a drone was deemed the most feasible approach.

## 4.6 Magnetic strip sensor

Magnetic strip sensors serve as a versatile tool for both saltwater and freshwater environments [25]. This technology provides both a portable and efficient solution for the measurement of ice thickness on bodies of water. Magnetic strip sensors operate on the basis of electromagnetic induction principles (EMI). The sensor can detect variations in the electrical conductivity of the ice, snow and water layer, resulting in induced changes in the surrounding magnetic field [25]. The thickness of the ice can be determined by measuring the changes in the magnetic field. This technology is used in lakes, rives and other bodies of water, often for climate research and environmental monitoring. The accuracy of magnetic strips can be affected by heat, which changes the conductivity in freshwater [26].

## 4.7 Stationary sensors

The magnetic strip sensor serves as a stationary sensor. In the IceMap system, it was intended to be utilized for verifying the accuracy of data acquired from the LIDAR. The sensor was supposed to provide an additional data source in conjunction with the NVE model. Magnetic strip sensors are primarily used for experimental purposes, and are not commercially available. This can make the sensors difficult to obtain. Some research articles offer instructions for building a magnetic strip sensor[27], however, this was outside of the groups expertise, and thus not pursued. Instead, the group focused on enabling the integration and processing of data from this sensor type. Magnetic strip sensors were deemed the most fitting sensor type due to their relatively low cost and small size. Most alternative technologies were either far too costly, or not suitable for use in water.

### 4.7.1 Proposed placements

While the specific positioning of the altimeter ultimately falls under Gjøvik municipality decision, the group has provided a proposal for the flight path of the drone on Mjøsa. The group proposes that the drone flies along the shoreline, keeping a distance of approximately 70m from the shore. The flight path begins in Gjøvik Sentrum and the LIDAR conducts one scan every 500 meters.

Multiple considerations went in to selection of these two distances. Firstly, it allows for the creation of a safety zone for individuals to walk on the ice, while also enabling regular assessment of ice thickness along the shoreline. Additionally, considering that Gjøvik Sentrum's shoreline spans about 5000 meters, placing the scans 500m apart provides broader coverage of variations in ice thickness within the water. This broader coverage ensures the effectiveness of the data collection, but frequent enough to account for changes in current which may impact the ice stability. Since a single drone cannot scan the entire lake, the group has proposed to focus on the shoreline, where most recreational active happens.

The group recommends to position the magnetized strips sensors along the shoreline to align with the LIDAR scans to compliment its data. These strips will be uniformly spaced with a minimum distance of 500 meters, and approximately 70meters from the shoreline. For large lakes such as Mjøsa, the group recommends deploying a minimum of four sensors, while smaller bodies of water may only require one. This placement is a compromise between area coverage and cost.

# Chapter 5

# Mobile application

The mobile application was developed using Dart, the Flutter SDK, and Android Studio. Three mobile emulators and a physical device were used to test the application, these being Pixel 7, Pixel 7 Pro, and Medium Phone API 31. These emulators are included in the installation of Android Studio. The physical devices was a Motorola g(9) plus. All testing was preformed on Android devices as the group lacked access to iOS devices. Ideally, the application would be tested on an iOS device too, but since Flutter provides full support for the newest version of iOS [28] the group assumes that IOs can be accommodated with minimal effort.

## 5.1 Server communication

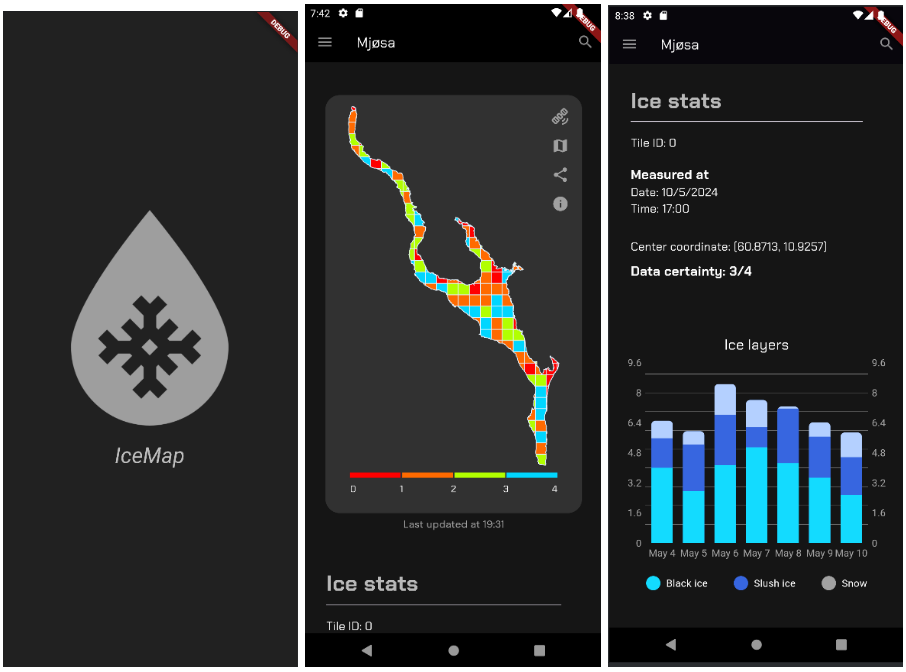
The server and app communicate via HTTPS. Every time the application requests map data from the server, the lake name is passed as a URL parameter. This URL is an example of how the application requests data from the server:

```
https://127.0.0.1:8443/update_map?lake=Mjøsa
```

Upon launching, the application sends a few get requests to the server in order to retrieve all the necessary map data.

## 5.2 Main components

The app consists of two pages: the default page, and the loading page. The loading page is only visible while the application is initializing. Once the application is done loading, the user is taken to the default page. The default page consists of a widget with an interactive color coded map, and a widget with a graph and various ice related data. The app bar contains the name of the currently selected lake, and a search icon, and a menu icon. As seen on Figure 5.1, the application was designed with a predominantly dark color scheme. Beyond providing a modern aesthetic, the color scheme also consumes less power.




**(a)** Loading screen **(b)** Map widget **(c)** Statistics widget

**Figure 5.1:** Loading page and two main widgets

## 5.3 Map widget

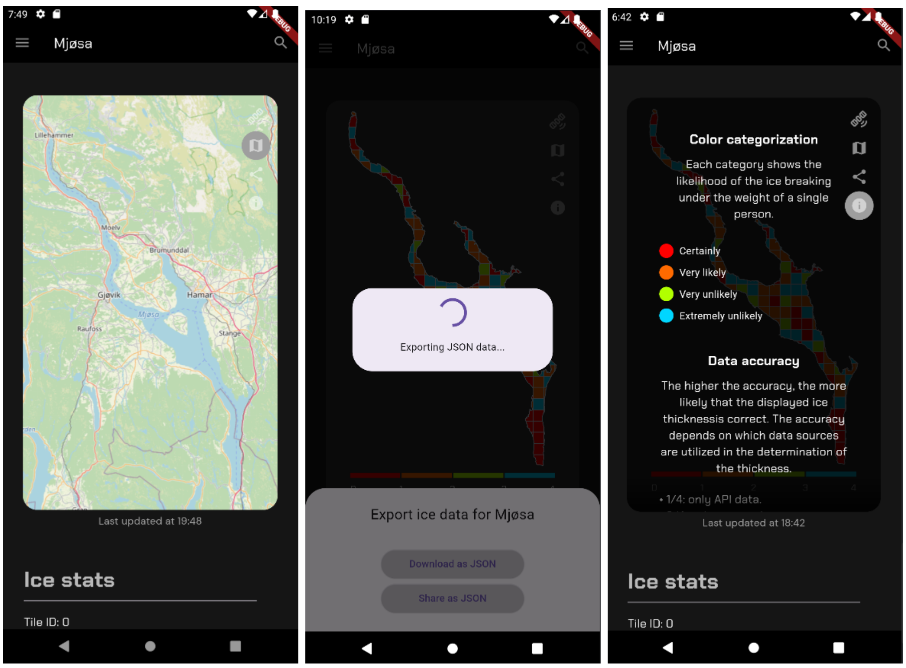
The map is rendered by a custom class called ChoroplethMap (see Code listing 5.1). The map consists of subdivisions, each of which can be selected by tapping on it. When a subdivision is selected, the information in the statistics widget changes to display information for the selected division. Four buttons are layered over the map widget in its upper right corner. The top button was intended to overlay a satellite image from the Copernicus API, but this feature was not implemented. The second button displays a map from OpenStreetMap (OSM). The third button allows the user to download or share the ice data as JSON. The fourth button displays a layer with information about the color categorization. These features can be seen on Figure 5.2. The legend just below the map categorizes the colors from a safety of 0 to 4. Below the map widget, the text "Last update at ..." is displayed. The text shows the last time that the application received data from the server. If the last update was done within the last day, it will only display the time of the last update. Otherwise, it will only display the date.

### 5.3.1 Color categorization

Each subdivision is either red, orange, green, or blue. These four colors were chosen to increase the readability of the map. More than four colors could make it difficult to discern safe areas from unsafe areas. Red and orange communicate that the ice is dangerous, but to different degrees. Green and blue communicate that the ice is safe, again to different degrees. Yellow was chosen to be kept out to ensure a clear and unambiguous transition from safe to unsafe sections. The information layer as seen on Figure 5.2c, includes some text and a legend to explain the color coding in more detail.

**Code listing 5.1:** Custom choropleth map class

```
/// ChoroplethMap is a stateful widget that contains a choropleth map.
/// The map is created using the Syncfusion Flutter Maps library and
/// coordinates fetched from the server.
class ChoroplethMap extends StatefulWidget {
  const ChoroplethMap({
    Key? key,
    required this.relation,
    required this.measurements,
    required this.subdivisions,
    required this.onSelectionChanged,
  }) : super(key: key);
```




**(a)** OpenStreetMap layer **(b)** Export feature **(c)** Info layer

**Figure 5.2:** Map buttons

### 5.3.2 Map rendering

ChoroplethMap requires relation data, a list of measurements, a list of subdivisions, and the callback function onSelectionChanged as parameters. onSelectionChanged returns the index of the selected subdivision to the parent class MapContainerWidget. The parent class indexes the subdivision list to extract and display the appropriate data in the statistics widget. The relation data is GeoJSON data which is used to render the map. The map is rendered with the Syncfusion library. This library allows a map to be rendered directly from GeoJSON data and has very simple implementations for color coding and shape selection.

The relation data contains coordinates that define each subdivision shape. Each subdivision has a unique ID. The IDs from the relation data are matched up with the corresponding IDs from the subdivision list. The subdivision list contains information about how each subdivision is supposed to be colored and the statistics for the subdivision. The coloring is decided in the server. Code listing 5.2 shows how the map data source is defined and how the colors are mapped to each subdivision. The code which decides the coloring and generates the map source data is described in chapter 6, while the generation of the measurement data is described in chapter 7.

**Code listing 5.2:** Map data source and color mapping

```
...
// Initialise data source and color mappers
      dataSource = MapShapeSource.memory(
        widget.relation,
        shapeDataField: 'sub_div_id',
        dataCount: widget.subdivisions.length,
        primaryValueMapper: (int index) => widget.subdivisions[index].sub_div_id,
        shapeColorValueMapper: (int index) => widget.subdivisions[index].color,
        shapeColorMappers: const [
          MapColorMapper(
              from: 0,
              to: 1,
              color: Color(0xffff0000),
              text: '{0},{1}'),
          MapColorMapper(
              from: 1,
              to: 2,
              color: Color(0xffff6a00),
              text: '2'),
          MapColorMapper(
              from: 2,
              to: 3,
              color: Color(0xFFb1ff00),
              text: '3'),
          MapColorMapper(
              from: 3,
              to: 4,
              color: Color(0xFF00d6ff),
```

```
            text: '4'),
        ],
    ...
```

## 5.4 Statistics widget

The statistics widget is right below the map widget, and containes more detailed information about the state of the lake ice (see Figure 5.1c). The topmost information is text that displays the unique ID of the selected subdivision, the time and date of the measurement, the coordinates of the measurement point, and the accuracy of the data. Below the text, a bar chart displays the various ice layers and their thicknesses. The data which was used to generate this chart is from the NVE model. The chart displays the ice layer conditions for today, the last three days, and the coming three days.

### 5.4.1 Data accuracy

The text "Data accuracy" field displays the accuracy of the data for the selected subdivision. A subdivision that is only based on statistics from the NVE model has a rating of 1, a subdivision only based on LIDAR data has a rating of 2, and a subdivision with both data sources has a rating of 3. If both data sources are included *and* the discrepancy between them is less than 1.0cm, the rating is 4. More factors could be included into this classification system, like the sensors accuracy and the present ice layers, but the group felt that these topics were not understood well enough to implement. Code listing 5.3 shows part of the code for classifying the accuracy.

**Code listing 5.3:** Accuracy clasification

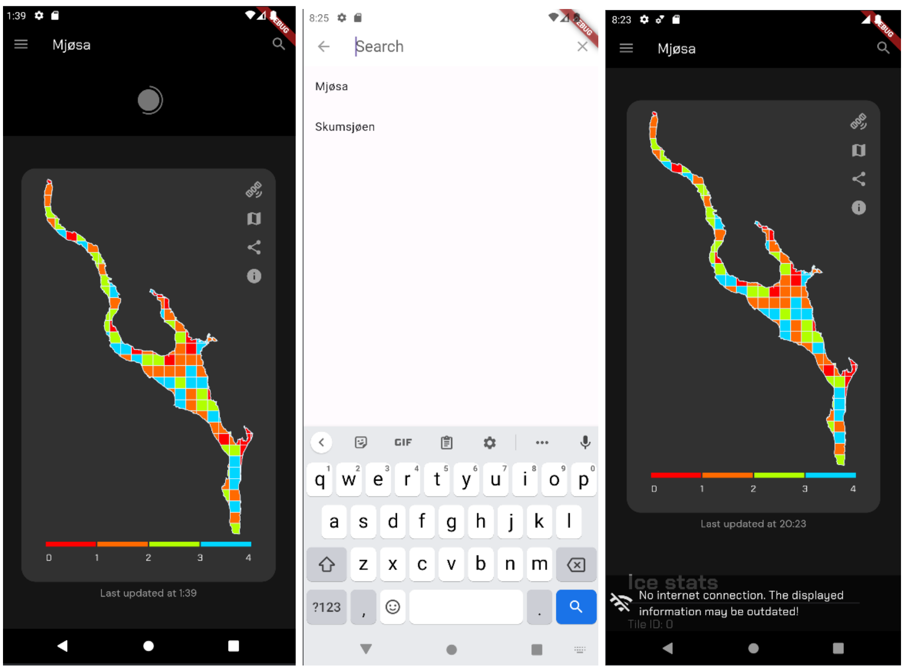
```
# Retrieve ice statistics for current subdivision
ice_stats = get_raw_dates(ice_prognosis_raw_data(sub_div_id=subdiv_id,
                                            x=center_lat, y=center_lng))

# Ice statistics were retrieved successfully
if len(ice_stats) > 0 and len(ice_stats[0]) > 0:
    accuracy = 3

    # Set accuracy to 4 if LIDAR data and NVE data have a minimal discrepancy
    if abs(avg_thickness - ice_stats[0]['Total␣ice␣(m)']) < 1.0:
        accuracy = 4

else:  # Failed to retrieve ice statistics, initialise empty ice stats object
...
    accuracy = 2
```

## 5.5 Lake search

The search feature allows users to change the displayed lake. Tapping the magnifying glass on the app bar will open a new view. This view contains a search bar and a list of all the system lakes (see Figure 5.3). Before anything is entered in the search bar, all the system lakes are displayed. When the user enters some text, the results will be narrowed. The narrowing and matching of the search string are done with the help of the Dart package Fuzzy. Fuzzy provides customization for threshold of the input matching, which is set to 0.4. After selecting a search results, the application sends new requests to the server, but with the newly selected lake as the URL parameter.

**Code listing 5.4:** Fuzzy search match

```
List<String> searchResults = [];
final options = FuzzyOptions(threshold: 0.4, findAllMatches: true);
final matcher = Fuzzy(lakeSearchOptions, options: options);
final results = matcher.search(query);
searchResults = results.map((result) => result.item as String).toList();
\end{minted}
```




**(a)** Reload animation **(b)** Search bar **(c)** No internet connection

**Figure 5.3:** Search feature and no internet behavior

## 5.6 Initialization and persistence

The application attempts to establish an internet connection when it launches. If an internet connection is established, the app makes a few requests to the server. If the requests receive a response, the server will save the responses to a set of three files, and update the persistent variable `lastUpdate` and `lastLake`. If the variable `lastLake` contains a value, this lake will be included as the URL parameter. Otherwise, the value defaults "Mjøsa". Code listing 5.5 shows how a persistent variable is set with the Shared Preferences plugin.

**Code listing 5.5:** Persistent variable lastUpdate

```
import 'package:shared_preferences/shared_preferences.dart';
...

final prefs = await SharedPreferences.getInstance();
await prefs.setString('lastUpdate', '${DateTime.now()}');
```

If any of the server requests do not receive a response, the application will try to read the map data from files. These files contain all the necessary map data which was saved from the last successful server request. On every successful request, the data in these files is overwritten, effectively always storing the last viewed lake. This mechanism allows the application to function even when the server does not respond, or when the application fails to establish an internet connection. If the application fails to establish an internet connection, a SnackBar displays a warning at the bottom of the page as seen on Figure 5.3c. The SnackBar remains until an internet connection is established. The application does not only check the connectivity upon launching, but actively keeps checking. This means that the SnackBar will also appear if the application is initialized with a connection, but loses it at a later point.

The loading page is displayed while the application is waiting for initialization to complete. The function `initialiseSate()` starts the initialization. This function is called on every launch, but can also be triggered by reloading the default page. Reloading can be done by pulling down on the page form above the map widget. Doing so will trigger a short loading animation which runs until `initilaiseState()` completes the re-initialization of the state (see Figure 5.3a). The same function is called when a new lake is selected trough the search feature. The icon on the loading page was created by Freepik and posted on Flaction[29], and is free to use with attribution.

## 5.7 Remaining work

Usability testing was not included as a requirement. Nevertheless, the group invested time and effort into making the application user friendly. Such efforts included formulating text in a clear and concise manner, selecting suitable fonts and font sizes, meaningful color choices, intuitive icons, and including adequate information without creating clutter. It would have been beneficial to conduct user testing on the following aspects: the expected functionalities of the map widget buttons, the clarity and relevance of the text within the statistics widget, and the overall comprehension and usability of the color-coded map.

The current color scheme looks fine, but is not very accessible. Since the map colors are very similar in saturation and shade, individuals with visual impairments may have trouble distinguishing the colors. The general color scheme of the application could also cause difficulty due to its low contrast. The application should have provided settings for both increasing the contrast of the map colors, and the colors of the rest of the application. Again, such considerations could not be implemented due to time restrictions. Given more time, the group would have utilized the currently decorative menu icon to set up a couple buttons for toggling between themes. These themes would consist of two sets of default and high-contrast themes, one set for the general application and one set for the map coloring.

Proper HTTPS certificates were not generated, so the app was configured to ignore certificate validation. Before the system is deployed, valid certificates must be generated and the application must be configured to validate the certificates. Three files in the application must be updated, all of which are located in the folder called "server_requests". Code listing 5.6 shows how the app clients are currently configured. The valid certificates must be placed in the server folder called "certificates". These are the only two steps required to set up proper communication between the app and server. All other required setup and configuration was implemented.

**Code listing 5.6:** HTTPS client configuration

```
// Custom HTTP client
HttpClient client = HttpClient()
  ..badCertificateCallback = // NB: temporary disabled SSL certificate
      validation
      (X509Certificate cert, String host, int port) => true;
```

# Chapter 6

# Custom map creation

The application requires a map shape file in order to render the color coded map. Although many solutions for rendering the maps exist, no readily available tools for creating the necessary map data were found. Therefore, the group developed a custom method for creating the required map files. This method involves using lake shape files from the OverpassTurbo API[1], and dividing the shapes into a uniform grid.

## 6.1 Adding a lake relation to the system

The first step in adding a lake to the system is manual. OverpassTurbo does not support fetching relations programmatically and is the only API that allows exporting of GeoJSON files for lakes. Therefore, a system administrator must extract the relation files from OverpassTurbo, export them as GeoJSON files, and then add them to the system manually. The administrator needs to navigate to the OverpassTurbo API and retrieve the polygon data for the desired lake. This is done with a Overpass query. Code listing 6.1 shows the query that must be entered into the Overpass API. This query will fetch the shape data for the requests lake. The output will consist of coordinates that make up the outline of the entire lake.

**Code listing 6.1:** Overpass API query

```
[out:json];
    (
      way["natural"="water"]["name"="lakeName"];
      relation["natural"="water"]["name"="lakeName"];
    );
    (._;>;);
out body;
```

The output from the query must be exported to an appropriately named GeoJSON file and added to the server. Eeach step of this process is described in detail in the

[1]https://overpass-turbo.eu/

projects README file in the GitLab repository[2]. After the GeoJSON file is added to the server, the system administrator must initialize the division of the map shape. This can be done by making a request to `https://127.0.0.1:8443/add_new_lake?lake=lakeName&cell_size=1`. The endpoint requires that the name of the newly added lake is provided as a URL parameter. The `cell_size` parameter determines the size of the subdivisions in kilometers, and is optional. If `cell_size` is omitted, the division will use a default size of 0.5km. After the shape file is added to the server, and the request is made, the manual part of the process is over. All remaining steps happen pragmatically and require no intervention.

The endpoint can be requested multiple times on the same lake. Every time the function is called, the new output will overwrite any files generated in prior requests for the specified lake. Re-running the division may be done if the cell size of prior requests was undesirable. The endpoint writes the resulting GeoJSON file to the response, and plots the resulting map. The resulting file and the plot should be inspected to ensure a desirable result. `cut_map()` is the function that is called by this endpoint, and can take multiple minutes to complete. The larger the lake and the smaller the divisions, the longer the function takes to complete. Plotting is the most time consuming part of the process, but it is highly discouraged from disabling. It is difficult to ensure that the result is desirable wihtout plotting it.

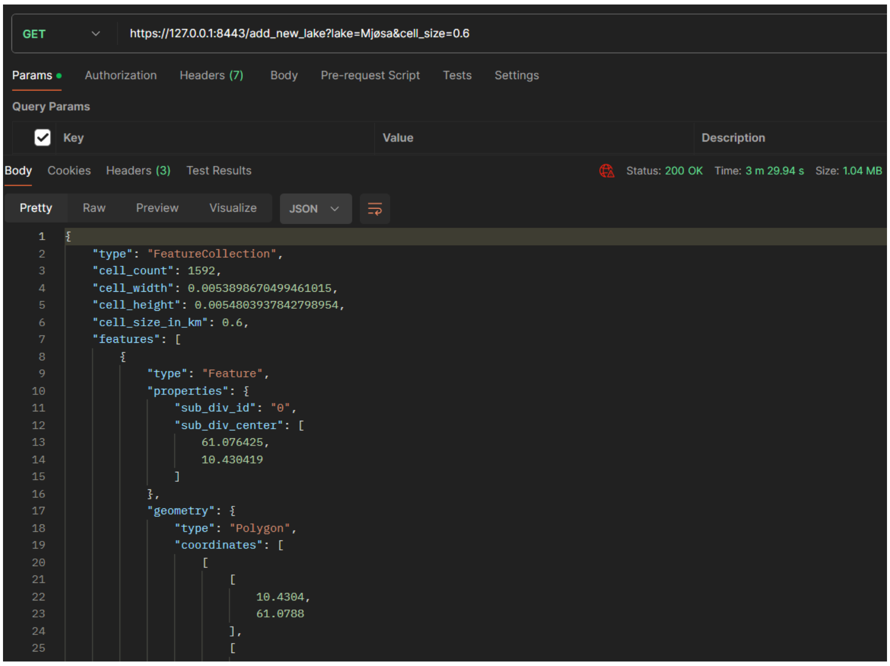



**Figure 6.1:** Example of a time consuming request

[2]https://gitlab.stud.idi.ntnu.no/sarasdj/prog2900

Figure 6.1 shows an example of running the request in Postman, and Figure 6.2 shows the resulting plot. As seen in the upper right corner of Figure 6.1, the example took about 3 minutes and 30 seconds to run, with a cell size of 600m. This example demonstrates the required time for an extreme case. When running the same code with a cell size of 2.5km, which is the size shown in all figures in chapter 5, the response is received in under 30 seconds. For all other Norwegian lakes, the required time should be far shorter as both the lake size and cell size can be smaller.

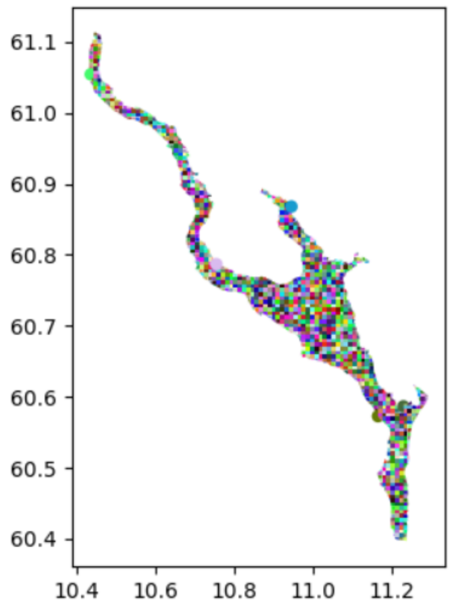



**Figure 6.2:** Pyplot graph from add_new_map endpoint

## 6.2 Dividing the map polygon

`cut_map()` creates the maps by generating a uniform grid and combining it with the map shape. The spacing between the grid lines is either determined by the size specified in the endpoint request, or set to the default value of 0.5km. The cell size is then converted to a height and width in degrees of latitude and longitude. Converting the height and width separately is necessary to ensure that the divisions become square. If this conversion was omitted, lakes further away from the equator would get elongated subdivision. In Norway, the height of the cells would become far longer than they were wide. To prevent this from happening, the cell width and height are calculated using the code from Code listing 6.2. This code was developed by modifying formulas posted by Numan Karaaslan on StackOverflow [30]. Code listing 6.5 shows how the grid lines are generated.

**Code listing 6.2:** Cell size calculation

```
\begin{figure}[h]
    \begin{minted}{Python}
        # Select an arbitrary x and y value from within the polygon
        bounds = polygons[0].bounds
        start_x, start_y, _, _ = bounds

        # Convert the cell size to lat and lng
        cell_width = cell_size_in_km / 111.3200
        cell_height = cell_width / cos(start_x * 0.01745)
```

**Code listing 6.3:** Generating the grid

```
    # Retrieve the polygon bounds
    bounds = poly.bounds
    min_x, min_y, max_x, max_y = bounds

    # List to store all created lines
    grid_lines = []

    # Create new horizontal lines while within bounds
    y = min_y
    while y <= max_y:
        line = LineString([(min_x, y), (max_x, y)])
        grid_lines.append(line)
        y += cell_height

    # Create new vertical lines while within bounds
    x = min_x
    while x <= max_x:
        line = LineString([(x, min_y), (x, max_y)])
        grid_lines.append(line)
        x += cell_width

    return grid_lines
```

Once the grid lines are generated, they are combined into a single polygon using unary operations. Then, a for loop iterates over all the lines and checks which parts of the map polygon intersect the lines. The parts of the polygon that intersects the lines are appended onto a list, which will eventually contain all each grid division. Code listing 6.4 shows how this is done. Upon creating every division, it is formatted as a GeoJSON feature object. Each object contains a unique subdivision ID, the shapes center coordinates, and the coordinates which make up its geometry. The list of subdivisions is then written to a file called "*lakeName_div.json".

**Code listing 6.4:** Combining the map polygon with the grid

```
# Generate a grid based on the calculated cell size
lines = create_grid(polygon, cell_width*2, cell_height)
lines.append(polygon.boundary)

# Merge the grid lines into a single grid object
lines = unary_union(lines)
lines = linemerge(lines)
lines = list(polygonize(lines))
```

```
# Combine the polygon and the grid to form the subdivisions
for line in lines:
    if line.intersects(polygon):
        divided_map.append(line.intersection(polygon))
```

**Code listing 6.5:** Subdivisions feature objects

```
sub_div_id = 0
...
for tile in divided_map:
    tile_feature = {
        'type': 'Feature',
        'properties': {
            'sub_div_id': str(sub_div_id),
            'sub_div_center': center
        },
        'geometry': geometry
    }

    # Append new feature object to list, and increment sub_div_id for next
        iteration
    features.append(tile_feature)
    sub_div_id += 1
```

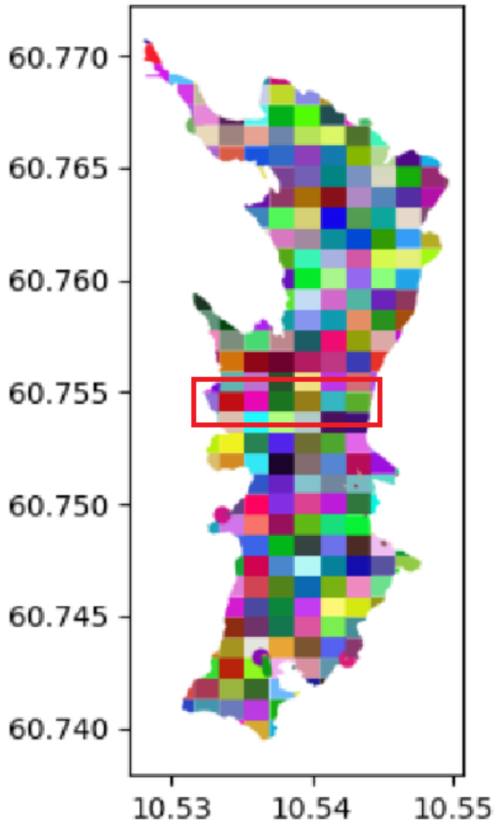
## 6.3 Verifying the output

The output of these calculations was verified by comparing the plot created by `cut_map()` with distances measured on Google maps. Skumsjøen was used for simplicity. Firstly, the distance between two points parallel to the x-axis (see Figure 6.3a) were measured on GoogleMaps. Then, the lake was added to the system and processed. The cell size was set to 100m for ease of calculation. The distance from Google maps measured approximately 636m, and the number of divisions along the measured line counted 6 whole divisions and one partial divisions. With divisions measuring $100m \times 100m$, this confirmed the expected output. The same verification was conducted on the height of the lake. The height measured 3.42km and the divided map measured just about 34 cells tall. Keep in mind that the aspect ratio of the plot on Figure 6.3b is s little squished, which makes the subdivisions appear slightly rectangular when they actually are square.

## 6.4 Determining subdivision colors

`calculate_color()` is called for every subdivision. The function determines the coloring of each subdivision. As mentioned in chapter 3, the coloring is determined by sensor data if it is available for a subdivisions. Otherwise, the color is determined by the thickness of the black ice layer form the NVE model. Code listing 6.6 shows how the `calculate_color()` is defined and called for subdivisions


(a) Width from Google Maps




**(b)** Map divided into $100m \times 100m$ cells

**Figure 6.3:** Verification of cell size formulas

without sensor data. `calculate_color()` assigns colors to subdivisions based on ice thickness as follows: red for 0cm to 4 cm, orange for 4cm to 8cm, green for 8cm to 12cm, and blue for thicknesses greater than 12 cm. For invalid thickness values, a default color of grey is used.

**Code listing 6.6:** Determining subdivision coloring

```
 # Create new subdivision object
sub_division = {
    'SubdivID': sub_div_id,
    'MinThickness': black_ice_thickness,
    'AvgThickness': black_ice_thickness,
    'CenLatitude': center_lat,
    'CenLongitude': center_lng,
    'Accuracy': accuracy,
    'Color': calculateColor(black_ice_thickness),
    'IceStats': ice_stats,
}

def calculate_color(thickness: float):
    if 0 < thickness <= 4:
```

```
        return 1  # Red
    elif 4 < thickness <= 8:
        return 2  # Orange
    elif 8 < thickness <= 10:
        return 3  # Green
    elif thickness > 12:
        return 4  # Blue
    else:
        return 0  # Default: grey
```

## 6.5 Exposing the processed data

All the data processed by the server has to be compiled and exposed on endpoints. This process is a bit complicated, as data from various sources and with differing formats have to be matched up with every single subdivision. First, all the most recently added measurements and their data is read from a file. This file contains coordinates, dates and times, and the thickness measured by the sensors. Once all the data is extracted and added to a list of dictionaries, the ice statistics from the NVE model are requested for every subdivision. While this data is compiled, the accuracy level and color of the each subdivision is determined as described in subsection 5.4.1. Since all of this code is very long, those who wish to look at the concrete implementation can view the file "update_measurements.py" in "server/map_map_handler" in the GitLab repository[3].

### 6.5.1 Updating the map data

The class `UpdateScheduler` updates and compiles the map data for each lake. The class runs indefinitely in a dedicated thread, and calls the function `update_all_measurements(update_bbox: bool)` regularly. The function fetches the newest API and sensor data for all lakes. This data is compiled and stored in "*_measurements.py" files for each lake. Once a day, `update_all_measurements(update_bbox: bool)` is called with the argument `False`. This means that the function only updates the data from NVE. Since seNorge updates their data once a day, updating it any more frequently is not necessary. If the argument `True` is passed, all SentinelHub bboxes will be updated in addition to updating the NVE data. Since the Sentinel satellites have a revisit frequency of 2-5 days, 3 days seemed an appropriate frequency to update the bboxes. Code listing 6.7 shows some of the code for updating the map data.

**Code listing 6.7:** Class for updating the map data

```
class UpdateScheduler:
    def __init__(self):
        self.day_counter = 1

    def start(self):
```

[3]https://gitlab.stud.idi.ntnu.no/sarasdj/prog2900/-/tree/main/server/map_handler?ref_type=heads

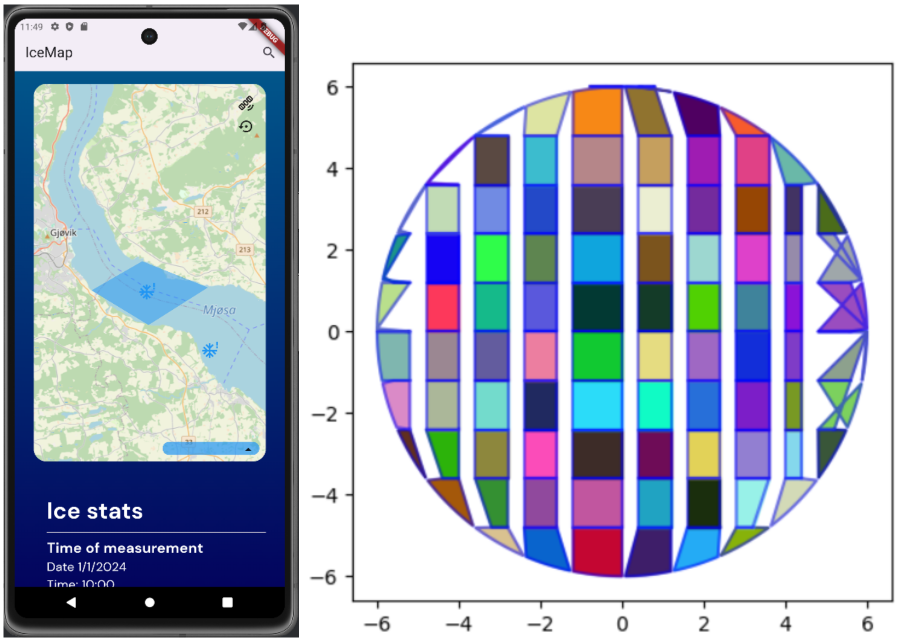
```
        """Schedules the updating of all maps every three days"""
        try:
            print("Updating␣all␣lake␣data....")

            # Run update_all_measurements on startup
            update_all_measurements(True)

            # Schedule updates to occur daily
            schedule.every(1).days.do(self.daily_update)

            # Keep scheduler running indefinitely
            while True:
                schedule.run_pending()
                time.sleep(1)
        except Exception as e:
            print(f"Failed␣to␣schedule␣updates:␣{e}")

    def daily_update(self):
        if self.day_counter < 3:  # Day 1 or 2, no bbox update
            update_all_measurements(False)
            self.day_counter += 1  # Increment counter
        else:  # Day three, update all data including bbox
            update_all_measurements(True)
            self.day_counter = 1  # Reset counter
```

## 6.6 Alternative map creation methods

The entirety of the method described so far was the last of three attempted methods. The first method used an OSM layer as a base map and rendered polygons over this map. The problem with this approach was related to how the polygons would be created. Defining a simple polygon like a square or a pentagon was easy, but generating a shape that matched up with the complex geometry of the shoreline was difficult. Figure 6.4a demonstrates this problem and the applications appearance in its earlier stages. The second method involved continuously splitting the map shape in two from top to bottom, then left to right, until the grid was formed. This implementation required unexpectedly complicated code and could at best produce a partially correct grid with noticeable gaps. Figure 6.4b shows the best achieved result from this method before the approach was abandoned. Although it can be improved in multiple areas, the final attempt was the simplest and only working solution.

## 6.7 Areas of improvement

It is unfortunate that there are no options to retrieve relation data pragmatically. If such an option was available, it would be much preferred to the current partially manual implementation. As the map creation stands now, the sizing of the grid cells is also manual. Although manual sizing is fine, it would be better to have a default size that would be proportional to the size of the lake. For example, the




**(a)** First method **(b)** Second method

**Figure 6.4:** Alternative map creation methods

default size of the grid could be calculated based on the height or width, and ensure that the amount of cells were within a meaningful boundary.

As readers familiar with Mjøsa have probably noticed, the island Helgøya is missing from some of the figures. This island is located in the middle of the widest part of Mjøsa, as can be seen on Figure 5.2a. The reason that the island is not included is that only the exterior coordinates of the maps are included in the processing. This means that any shapes completely enclosed in a lake, like an island, is not included. When the interior shapes were included, the server would throw errors. Some attempts were made at addressing the errors, but a viable solution was not achieved by the deadline. This issue is not detrimental, but a user that wishes to inspect the ice conditions surrounding the island may have problems locating the exact subdivisions which border the island.

The last issue in the map creation process is the occurrence of disproportionately small subdivisions. The map generation process guarantees that subdivisions have two or more points, but this means that any shape with at least 3 points is considered adequate. This poses a problem when the average subdivision consists of, for example, 50 points. These small subdivisions are difficult to spot and select in the application. Ideally, subdivisions any smaller than a tenth of the average subdivision size should have been merged with the nearest larger subdivisions. Implementing this approach could have mitigated the occurrence of disproportionately small subdivisions. This implementation would require logic to determine which subdivision is the closest and most meaningful to merge the smaller subdivision with.

# Chapter 7

# Sensor and API implementations

## 7.1 Drone

For the Drone, the group choose to utilize the *DJI Metrice 300*[1] leveraging its preexisting functions for automated flight, waypoint navigation, and POI management as mentioned in section 4.3. Additionally, DJI offers their own SDK[2] providing users with the option to customize their applications for greater control over drone operations.

Alternatively, for drones other than DJI Metrice 300 that lack built-in automation functions, the group recommends utilizing QGroundControll (QGC) flight navigation software along with the PX4 extension. This combination offers enhanced automation and stabilization capabilities.

## 7.2 LIDAR files

The LIDAR is configured to scan a designated area upon reaching a specified coordinate. Data from the scan is then saved and stored in its memory. If a file already exist, the LIDAR will overwrite it. The saved file is named to identify the measured area, with a folder named after the name of the water body. The file itself will start with "*measurement_id_*" followed by the ID of the measurement coordinates.

Currently, stored files are transferred manually when the drone returns to its station. Although the system has the capability for automatic transfer, this feature has not yet been implemented.

While code was developed for Arduino implementation, since the group switched hardware to the system from DJI Matrice 300. Consequently updating the LIDAR data must be done manually. This process involves naming the Log ASCII Standard

[1] https://enterprise.dji.com/matrice-300?site=brandsitefrom=recommended
[2] https://developer.dji.com/mobile-sdk-v4/

zip (LAZ) file from the LIDAR after the ID of its measurement coordinate ("*measurement_id_{ID_OF_MEASUREMENT}*"), then moving it to the folder with the lake name of the measurement "*server/lidar_data/{LAKENAME}/*". If file already exist, it must be overwritten. Here is an example of how the path would look if a scan of Mjøsa with coordinate ID 3 was added: "*server/lidar_data /mjøsa/measurement_id_3*".

## 7.3 LIDAR data processing

During the LIDAR's imagery process, it captures a designated zone around it. By utilizing known coordinates and the size of captured zone, specific ice portions observed on the map can be identified. This determination involves calculating coordinates relative to the LIDAR data's coordinates, demanding processing it as a relative estimation within the file.

### 7.3.1 First method

Our initial approach to connecting the LIDAR data with the application map involved converting the designated area on the application's map into the LIDAR coordinate system using vectors. The group determined the relative sizes by utilizing Equation (7.1)

$$\begin{aligned} A_1 &= B - C \\ A_2 &= G - F \\ P_2 &= (P_1 - C_1) \cdot \left(\frac{A_2}{A_1}\right) + (P_1 - C_2) + C_2, \end{aligned} \tag{7.1}$$

where $P_2$ is a vector in the LIDAR coordinate system, $P_1$ is a vector in the map coordinate system, $A_2$ and $A_1$ are coordinate limits of the LIDAR and map respectively, $B$ and $C$ represents the top-right and bottom-left corners of $A_1$, and $G$ and $F$ for $A_2$'s top-right and bottom-left corners, $C_1$ signifies the center of $A_1$ and $C_2$ represents the center of $A_2$. This these points are displayed in Figure 7.1 and demonstrated in pseudo code in Code listing 7.1.

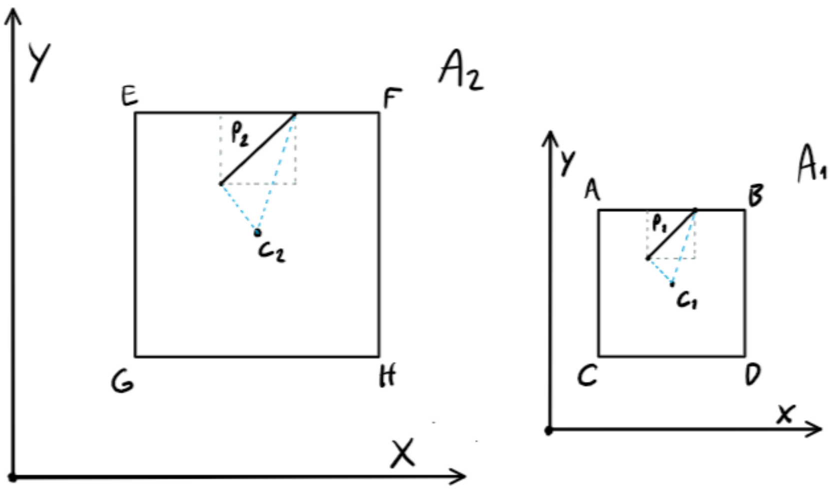



**Figure 7.1:** LIDAR trial calculation model

Despite our efforts, stress testing with real coordinates revealed inconsistencies, likely due to memory issues rather than calculation error. The data often return incorrect values when dealing with decimals. To address this, we considered rounding and converting back to integer before converting back to to floats to prevent memory leaks. However, due to the these inconsistencies, we ultimately decided to abandon the approach displayed in 7.1.

**Code listing 7.1:** Identify world coordinates relative to LIDAR coordinates

```
def position_relative_to_pointcloud(A1, P, C1, A2):
minA1, maxA1 = A1
minA2, maxA2= A2
minP1, maxP1 = P1
C2 = tuple((a+b)/2 for a, b in zip(l1,l2))

return [
    ((p1[0] - C1[0]) * ((minA1[0] - maxA1[0]) / (maxA2[0] - minA2[0])) + C2[0],
     (p1[1] - C1[1]) * ((minA1[1] - maxA1[1]) / (maxA2[1] - minA2[1])) + C2[1])
         ,
    ((p2[0] - C1[0]) * ((minA1[0] - maxA1[0]) / (maxA2[0] - minA2[0])) + C2[0],
     (p2[1] - C1[1]) * ((minA1[1] - maxA1[1]) / (maxA2[1] - minA2[1])) + C2[1])
         ,
]
```

### 7.3.2 Current method

We adopted a different approach: gridding the LIDAR data similarly to the map's grid, assuming that each division represented the same area on both the LIDAR scan and the map's subareas. The LIDAR and subareas were merged as demonstrated in Code listing 7.2, and compared with the application map to determine which scanned area corresponds to which part of the application map. To ensure the accurate merging of the two grids, it was essential that the scanned area size matched the map's subarea size. This was achieved by storing the dimensions on each subarea for each lake's data. During processing, these dimensions were used by a flexible function to allocate scanned points into their corresponding subareas, considering the area size, scan position, and the number of grids, as demonstrated in Code listing 7.3.

This approach proved more efficient than finding relative positions from map to LIDAR coordinates. Although it introduced potential issues, such as multiple or no grids corresponding to a map grid, the group addressed these by finding the closest center within the scan grids to the map grid. This was done by comparing the distance between center coordinates using the distance formula between two points (7.2)

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tag{7.2}$$

and allowing for adjustable grid sizes.

**Code listing 7.2:** Zip grid of map positions and LIDAR data

```
...
# define all the sub-areas within the area, local coordinates
grid_sub_area = define_gridareas(center[0], center[1],
                                        (cell_x, cell_y),grid_size)
# define all the sub-areas within the area, LIDAR coordinates
grid_area_lidar_heights =
    define_grid_lidardata((min_point, max_point), grid_size, ice_points)

# zip together the two list
sub_area_heights = list(zip(grid_sub_area, grid_area_lidar_heights))
...
```

However, the group faced challenges with converting distances in meters to coordinates because coordinates do not represent distance. This was resolved by using a function called *calculate_corners*, as shown in **??**, which used a formula provided by Numan Karaaslan as mentioned in chapter 6 to accurately create each grid as show on Code listing 7.3.

**Code listing 7.3:** griding of scanned area

```
# separate the zones into smaller area, into a grid
def define_gridareas(lat, lng, area_offset, grid_size):

    ...

    # find the center coordinates of each area in grid to find the corner areas
    for y in (range(grid_size)):
        relative_size_lat = y / grid_size # relative y position on grid
        for x in (range(grid_size)):
            # relative x position on grid
            relative_size_lng = x / grid_size
            lat_pos = main_area[3][0] + relative_size_lat *
                            area_size[0] + dist_to_subcenter[0]
            lng_pos = main_area[3][1] + relative_size_lng *
                            area_size[1] + dist_to_subcenter[1]

            # use the center of sub areas to find the corner of each subarea
            subarea_offset = (subarea_offset_lng, subarea_offset_lat)
            corners = calculate_corners(lat_pos, lng_pos, subarea_offset)
            grided_area.append(corners)

    return grided_area
```

### 7.3.3 Thickness calculation

To determine the thickness of the ice using LIDAR data, the group utilized the points collected from the LIDAR. This involves comparing points with identical XY coordinates and filtering out those lacking a counterpart. Subsequently, the difference between the Z coordinates of points sharing the same XY coordinates was calculated to determine the ice thickness, as demonstrated in Code listing 7.4. In zones where multiple ice thickness exist, the smallest thickness in that zone is considered to determine the safety of the area. While this method may not be the most sophisticated, the group considers it the best course of action to mitigate the uncertainties of traversing the ice.

**Code listing 7.4:** Identify and calculating ice thickness if identical coordinates

```python
def find_height(points):
    ...
    # sort the points
    sorted_coords = sorted(points, key=lambda coord: (coord[0], coord[1]))

    # group the sorted points that has the same xy- coordinates together
    groupCoords = [list(group) for key, group in
            groupby(sorted_coords, key=lambda coord: (coord[0], coord[1]))]

    # loop through the groups to find the difference in height
    for group in groupCoords:
        if len(group) < 2 or group[0] == group[1]:
            # jump over iteration if there is only one coordinate or
            # LIDAR registered the same point twice
            continue

        # find max and min Z-position of group
        min_height = min(coords[2] for coords in group)
        max_height = max(coords[2] for coords in group)

        # difference between them
        difference = max_height - min_height
        height_differences.append(difference)

    # list of thickness in an area
    return height_differences
```

### 7.3.4 Reflection

Integration of third-party hardware proved more challenging than anticipated. Despite the system's flexibility, combining multiple third-party components proved difficult. Limited resources, including documentation, hindered seamless integration. Additionally, compatibility issues stemming from differences in protocols and interfaces complicated the process, Despite our efforts, time and resource constraint prevented complete integration within this thesis. Acquiring these hardware would allow us to further develop and explore the DJI's payload SDK and mobile SDK development for a more automatic implementation rather than the current manual implementation handling files.

Moreover, our current ice thickness calculation model, while functional, is rather simple and lacks complexity. Because it primary relies on the XYZ-coordinates in a plane disregarding various factors like light diffusion upon colliding with ice, snow layers on top of the ice surface and diverse type of ice layers by utilizing the intensity data provided by the LIDAR. Addressing these complexities requires collaboration with researchers in this field, the group could refine our models. Integrated more comprehensive data processing techniques, and enhance the accuracy and reliability of our ice thickness measurement.

## 7.4 Implementing model given by NVE

To work with the computational model obtained from NVE, several smaller functions have been developed. These functions are designed to interact with the front-end interface of the system. The main objective is to create and store data files that include estimated ice thickness and types, making this information readily available for any part of the system that needs access.

Code listing 7.5 is an example of how the NVE model is applied to calculate data for different subdivisions. The code fetches subdivision IDs and their coordinates, filters data for specific dates, and then converts this information into a JSON format to be stored in a designated folder.

**Code listing 7.5:** Using the NVE model

```
sub_divs = get_subdiv_ids_n_cords(location)
filtered_data_for_dates = [(i[0], get_raw_dates\
    (ice_prognosis_raw_data(sub_div_id=i[0], x=i[1], y=i[2]), from_date, to_date))\
    for i in sub_divs ]

jsonify_data_sub_div_ids("skumsjoen", \
    filtered_data_for_dates, location = se.plot_folder)
```

This approach ensures that data generated by the ice prediction model is stored and ready to be accessed by the front-end without delay. This is particularly important because retrieving weather data from the SeNorge API, which is used in these calculations, can be time-consuming when dealing with hundreds of subdivisions. Therefore, by preparing the data in advance, thus ensures the system being able to operate more efficiently without having to wait for live API calls.

The calculation model used is a critical part of how the system predicts ice conditions. As shown in Code listing 7.6, the function requires several parameters including weather data from the SeNorge API, location data, time stamps based on time and place, and icerun dates which are supplied through the Sentinel Hub API (more about Sentinel Hub implementation here section 7.5).

**Code listing 7.6:** NVE Calculation function

```
def calculate_ice_cover_air_temp(inn_column_inn, date, temp, dh_sno, cloud_cover=
     None, time_step=60*60*24,
                                 icerun_dates=[]):
    """
    Last update: 12.02.2024 aask

    :param inn_column_inn:  [IceThickness] Initial ice column for modelling.
    :param date:            [] dates for plotting
    :param temp:
    :param dh_sno:          [] new snow over the period (day)
    :param cloud_cover:
    :param time_step:       [int] fixed time step of 24hrs given in seconds
```

```
    :param icerun_dates     [] dates when an ice run has cleaned the river for ice.
           - Ice reset to ice free
    :return:
    """
```

To make the subdivisions fit into the calculation model and using the SeNorge helper functions provided by NVE, it is simply a matter of ensuring the coordinates and time frames align properly with the model's expectations. This process involves converting geographical coordinates to Universal Transverse Mercator coordinate 33 (UTM 33) system and calculating the appropriate dates for data processing. The code snippet shown in figure Code listing 7.7 demonstrates how these adjustments are made within the system to fit the prediction model's requirements.

**Code listing 7.7:** Handeling sub_div area and time

```
def ice_prognosis_raw_data(to_date=None, sub_div_id=0, x=10.70, y=60.81,
                           altitude=0, awt=[], mf=0, icerun_dates=[]):

    current_date = dt.datetime.now()

    if current_date.month < 10:
        from_date = dt.datetime(current_date.year - 1, 10, 1)
    else:
        from_date = dt.datetime(current_date.year, 10, 1)

    if to_date is None or to_date > dt.datetime.now():
        to_date = dt.datetime.now() + dt.timedelta(days=7)

    tyear, tmonth, tday = from_date.year, from_date.month, from_date.day
    if tmonth > 7 and tday < 7:
        tyear += 1

    first_ice = ice.IceColumn(from_date, [])
    first_ice.add_metadata('LocationName', sub_div_id)  # Using sub_div_id as the
        location name for metadata
    observed_ice = [first_ice]

    # make the x and y into utm 33 from lon lat
    cords = utm.from_latlon(x, y, 33)
    x, y = int(cords[0]), int(cords[1])

    # check if utm is valid
    if validate_cords(x, y) is False:
        return None

    ...
```

In using the calculated ice cover, it is important to classify the raw data into different ice types, which are then compiled into comprehensive daily records. These records are meant for front-end use but can also provide a historical overview of ice growth. This process is shown in this Code listing 7.8 code where the `calculate_ice_cover_air_temp` function has been called and then data necessary for the front-end is extracted.

**Code listing 7.8:** Using NVE to get wanted data

```
def ice_prognosis_raw_data(to_date=None, sub_div_id=0, x=10.70, y=60.81,
                           altitude=0, awt=[], mf=0, icerun_dates=[]):

    ...
    Building request
    ...

    calculated_ice = it.calculate_ice_cover_air_temp(copy.deepcopy(first_ice), date
        , temp,                       sno, cloud_cover=cc, icerun_dates=icerun_dates
        )
    ...
    Classify ice
    ...
    data = []

    for date, slush, black, total, snow2, sno_tot2, cc2, temp2 in zip(dates,
        slush_ice, black_ice, total_ice, sno,
                                                                sno_tot, cc,
                                                                    temp):
        daily_data = {
            "Date": date.strftime("%Y-%m-%d"),
            "Slush␣ice␣(m)":    round(slush, 3),
            "Black␣ice␣(m)":    round(black, 3),
            "Total␣ice␣(m)":    round(total, 3),
            "Snow␣depth␣(m)":   round(snow2, 3),
            "Total␣snow␣(m)":   round(sno_tot2, 3),
            "Cloud␣cover":      round(cc2, 3),
            "Temperature␣(c)":  round(temp2, 3)
        }
        data.append(daily_data)

    return data
```

These functions and data handling methods help ensure that the system can effectively predict and manage ice conditions using the NVE model, maintaining efficient and timely access to critical environmental data.

Further work on the model would focus on providing the option to include average water temperature. Currently, the model uses hardcoded values from NVE for this purpose. While magnetized strip sensors would be ideal for measuring average water temperature, both the average water temperature data and the magnetized strip sensors are still experimental. Therefore, further testing and development are needed to fully integrate these features into the model.

## 7.5 Using Sentinelhub to get icerun dates

The primary function of the Sentinelhub API is to gather details about the presence of ice within specified BBox. Once the data is retrieved, it is saved to CSV files for each BBox. This information is then used to supply the NVE model with estimates of ice runs by finding the nearest BBox to any given point required by the NVE model. This section explains how this process is carried out.

Working with the Sentinelhub framework allows for making detailed requests using their statistical API to analyze every pixel within a specified frame. This capability is demonstrated with the `statistical_request_sentinel` function shown in Code listing 7.9. For such requests, a configuration and an evalscript are essential. These configurations can be accessed and set up within Sentinel Hub's own dashboard, detailed in [31].

To access the statistical API, at least the exploratory pricing plan is necessary. If usage exceeds the allocated amount for the exploratory tier, an upgrade will be required. For testing purposes, a free account can be created which remains active for 30 days. More information about accessing configurations in Sentinel Hub can be found in [31].

**Code listing 7.9:** Sending request to Sentinel Hub

```
def statistical_request_sentinel(config, evalscript, time_interval, maxcc, bbox):
    try:
        request = SentinelHubStatistical(
            aggregation=SentinelHubStatistical.aggregation(
                evalscript=evalscript,
                time_interval=time_interval,
                aggregation_interval="P1D"
            ),
            input_data=[
                SentinelHubStatistical.input_data(
                    data_collection=DataCollection.SENTINEL2_L1C,
                    maxcc=maxcc
                )
            ],
            bbox=bbox,
            config=config,
        )

        stats = request.get_data()

        dfs = [stats_to_df(polygon_stats) for polygon_stats in stats]

        return pd.concat(dfs)

    except Exception as e:
        print(f"Something␣is␣wrong␣with␣request,␣error:␣{e}")
        return None
```

This code snippet outlines how to set up and execute a statistical data request using Sentinel Hub's API. The function initializes a request with specific parameters such as the evalscript, time interval, maximum cloud coverage, and geographic bounding box. It then processes and converts the retrieved data into a format suitable for further analysis. If an error occurs, it catches the exception and notifies the user when running the function to update the iceruns.

To determine the ice runs, the necessary coordinates are provided, and then running a function called `get_closest_bbox_and_id` sequentially searches through every bounding box (BBox) listed in a JSON file. It finds the closest BBox and uses a file with information for the found BBox to retrieve the ice runs for a season. Although this process may take longer with an increased number of BBoxes in the JSON file, the function only needs to run concurrently with the NVE model. Since the NVE model does not need to operate continuously, it is feasible to run both tasks simultaneously without affecting the front-end operations.

**Code listing 7.10:** Getting ice runs for an cordinate

```
# location for an given box and returns the closest box cordinates and the id of
    the box
lon, lat = 10.66, 60.95     # mjos
closest_box, box_id = box_funcitons.get_closest_bbox_and_id(lon, lat)

data        = read_from_csv(f"bbox_sat_data/lake_{box_id}")
icerundates = get_ice_run_dates(data)
```

This Python snippet in Code listing 7.10 demonstrates how to find the closest bounding box based on given longitude and latitude coordinates and then how to read and interpret satellite data associated with that bbox to determine ice run dates. This method ensures that the NVE model receives timely and accurate updates about ice conditions, crucial for its operations.

To update of the information for each BBox and their ice runs, a function (see Code listing 7.11 is designed to manage updates. This function handles reading from a CSV file that stores satellite data per BBox. If the file does not exist, it creates a new one. If it can read an existing file, the function retrieves the date of the last recorded image. It then uses this date to make a request to Sentinelhub, fetching all new images from that date to the current day. The decision to retrieve images is based on the cloud coverage percentile. It determines if the pictures are clear enough when the satellite takes pictures of the BBoxes. As a note, it also means if there are cloudy periods, trying to update the ice runs will try and fail to receive any data causing large discrepancies in the data.

**Code listing 7.11:** Updating the iceruns

```
def update_to_most_recent(file, BBox, evalscript = evalscript.lake_ice_index, maxcc
     = 0.4):
    """ if file exists, tries to get the most recent date. uses that date to do get
         all dates until now. Appends that data to the file
    if the file does not exist. create a new one and put all data into it.  """
    try:
        data = pd.read_csv(f"{file}.csv")
        last_date = get_last_date(data)
        today = dt.datetime.now().strftime("%Y-%m-%d")

        data = statistical_request_sentinel(config= sentinel_config.config,
            evalscript=evalscript,
```

```
                                            time_interval=(last_date, today),
                                            maxcc=maxcc, bbox=BBox)

        if data is not None and data.empty is False:
            data = classify_ice(data)
            append_data_to_csv(file, data)

        return

    except FileNotFoundError:
        last_date = get_last_date(None)
        today = dt.datetime.now().strftime("%Y-%m-%d")

        data = statistical_request_sentinel(config=sentinel_config.config,
             evalscript=evalscript,
                                       time_interval=(last_date, today),
                                       maxcc=maxcc, bbox=BBox)

        if data is not None:
            data = classify_ice(data)
            save_data_to_csv(file, data)
        return
```

Improvements in fetching ice run data from Sentinel Hub could focus on refining the evalscript and the method of labeling the data. A more detailed discussion about adjusting the evalscript can be found in the next chapter, referenced here 7.6. Currently, labeling is based on specific values of variation in each image used in the statistical request. These values are meticulously chosen after reviewing three years of data for Mjøsa, assessing the standard deviations and corresponding conditions. Mislabeling typically occurs when the ice breaks as both ice breaking and clouds can have high standard deviations.

Ice runs are identified by interpolating dates when the water state of a BBox changes while filtering out cloudy days. However, filtering for clouds can lead to errors since cloudy images often show high standard deviations that don't accurately reflect changes in water state. During ice breakage, these deviations may indicate broken ice or ice patches, potentially mislabeling ice run dates. While mislabeling during ice breakage isn't a significant safety concern—it is apparent that ice with large visible breaks is unsafe—it can cause inaccuracies in the model, especially when the model resets ice runs during cloudy winter days when the ice isn't actually breaking. This does not pose a safety risk, but it could make the model less accurate than if the ice runs were used at all.

Ice run dates are calculated by interpolating changes in conditions and filtering out non-representative data (see **??**).

**Code listing 7.12:** Getting iceruns based on state of water changeing

```
def get_ice_run_dates(data):
    """ Get the dates for when the ice is on and of"""
    """ Interpolates the date between two dates where the state for the
         ice_condition changes, "does count cloudy days " """
    ice_run_dates = []
```

```
    # Iterate through the DataFrame to find condition changes
    for i in range(len(data) - 1):
        current_condition = data.iloc[i]['ice_condition']
        next_condition = data.iloc[i + 1]['ice_condition']

        # Check if there is a change in condition, skipps clouds during fall, but
            assumes clouds to be breaking ice
        # if it is after ice as the cloud labeling has a probability to be caused
            by ice breakage.
        if current_condition != next_condition and (current_condition,
            next_condition) not in \
               {('No␣ice', 'High␣probability␣of␣Cloud'), ("High␣probability␣of␣
                   Cloud", "No␣ice")}:

            # Interpolate the date between the current interval_to and the next
                interval_from
            current_to = pd.to_datetime(data.iloc[i]['interval_to'])
            next_from = pd.to_datetime(data.iloc[i + 1]['interval_from'])
            midpoint_date = current_to + (next_from - current_to) / 2
            ice_run_dates.append(midpoint_date.strftime('%Y-%m-%d'))

    return ice_run_dates
```

Code listing 7.12 demonstrates a method for detecting significant changes in ice conditions by interpolating between known intervals of change, while also accounting for cloudy conditions to avoid misleading data. This approach helps ensure the accuracy of the NVE model by recording and considering only valid ice runs, provided that the labels are recorded correctly.

## 7.6 Sentinel-hub eval script

The evalscript used in the statistical API from Sentinel Hub (see Code listing 7.13) is written in JavaScript. When using an evalscript in the Sentinel Hub framework, the first step is to declare the spectral bands used in the `setup()` function. To calculate the ILI, the group need bands `B04` (Red), `B08` (NIR), `B11` (SWIR1), and `B12` (SWIR2), as well as the `dataMask`, which helps identify each valid pixel. The ILI calculation takes place within the `evaluatePixel` function, where it is amplified using a hardcoded contrast amplification variable discussed in this chapter (see section 3.5).

**Code listing 7.13:** Evalscript for statistical request on sentinel hub

```
//VERSION=3

function setup() {
  return {
    input: ["B04", "B08", "B11", "B12", "dataMask"],
    output: [{
      id: "default",
      bands: 1,
      sampleType: 'UINT8'
```

```
    }, {
      id: "dataMask",
      bands: 1,
      sampleType: 'UINT8'
    }]
  };
}

function evaluatePixel(sample) {
  var Red = sample.B04;
  var SWIR2 = sample.B12;
  var NIR = sample.B08;
  var SWIR1 = sample.B11;

  var contrast = 120
  var result = ((Red + SWIR2) / (NIR + SWIR1)) * contrast;

  if (sample.dataMask === 1) {
    return {
      default: [result],
      dataMask: [1]
    };
  } else {
    return {
      default: [0],
      dataMask: [0]
    };
  }
}
```

This JavaScript code defines how the satellite data is processed using Sentinel Hub's capabilities. The `evaluatePixel` function is crucial; it applies the ILI to calculate the result based on the selected bands and enhances it using the specified contrast, as discussed in section 3.5. This function determines what each pixel in the satellite images represents—whether it's ice, water, or clouds—based on the standard deviation of the selected area.

# Chapter 8

# Result and discussion

A system for automated ice thickness measurement and mapping was successfully created, meeting the municipality's requirements. However, further development is needed before the system can be fully deployed. This chapter reviews the accomplished goals, evaluates their success and shortcomings, and outlines the remaining tasks.

## 8.1 Result goals

A MVP and prototype were developed for the entire system. While the resulting system addressed the municipality's requirements, practical implementation and hardware testing remains a future task. The prototype, though functional, contained some minor bugs. Despite these bugs, all the selected requirements were successfully implemented. Figure 8.1 shows some of the programming languages, libraries, third party APIs, and SDKs which were utilized to create the result. The figure is not exhaustive, but covers the more interesting technologies.

### 8.1.1 System requirements

The system requirements were comprised of six points as described in subsection 1.1.2. Of these points, five were fully implemented. The requirement which was not fulfilled was the point on unit testing. The following subsections covers the requirements which were met, while section 8.9 goes more into details of remaining work.

#### Sensors

The goal of creating a solution for a mobile sensor was met. The system includes code for processing raw LIDAR data, formatting it, and calculating estimations of the ice thickness. An automation of drone flight path has been established, but

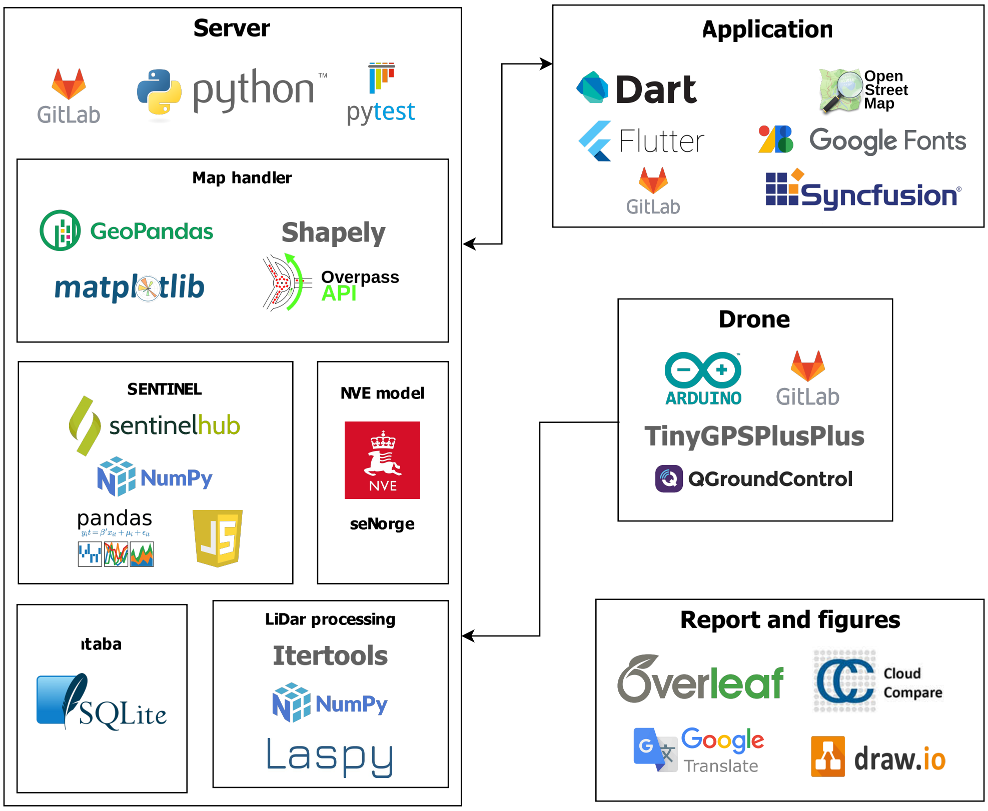



**Figure 8.1:** A non-exhaustive diagram of programming languages, libraries, and other technologies utilized to create the result.

not tested on an actual drone.

As discussed in section 4.6, the group could not find a fitting stationary sensor solution. Magnetic strips were researched and considered, but did not adequately fit the requirements of the system. Regardless, the system was made modular enough to allow for relatively simple implementation of such a technology if it were to become available.

SENTINEL-L2A satellite data and outputs from the NVE model were implemented to server as a baseline for areas where a drone cannot be deployed. This combination allows the system to gather information about when ice first establishes on lakes, and the current conditions of the ice. The data is updated every few days.

### Mobile application

The mobile application and color coded map were fully implemented. The map worked as expected, but it still does not render fully enclosed islands. The statistics widget below the map effectively displays details about the state of the lake ice, both in the form of a chart and some text. The app was developed with usability in mind, but user testing is yet to be conducted.

#### Data exportation

The application allows for exportation of the map data, but only as JSON format. Since this data was intended for regular users, it could have been better to include at least *one* second option like CSV, as the average user may not be familiar with JSON format.

#### Response time and data exportation

Since the complexity of the system and data processing was unknown, the response time served as an upper limit for the servers performance and the efficiency of the server-app communication. As the required response time was set at a generous 5 seconds, it was easily achieved. Each map data request takes up to a few hundred milliseconds, resulting in a loading time of one to two seconds for the application. This has been tested both with Postman and by launching the app on a phyiscal device.

#### Unit tests

The unit tests were implemented with he help of the pytest framework. The assertion of the exact test coverage of the system was complicated by the introduction of the NVE model. As this model is technically a third party software, its code was not intended to be covered by the unit tests. Since the group was primarily interested in testing the code they themselves wrote, the NVE model reduces the overall test coverage. Nevertheless, the total test coverage was 36%, which can be verified by running pytest –cov. All the tests should pass, and take a few minutes to complete.

#### Database

As the database was only intended for long-term archival, the design did not require a very complex implementation. It included only 4 entities with a few attributes each. While SQLite may not be ideal for large-scale systems, it adequately serves the needs of a smaller system. Given that IceMap's expected usage is restricted to municipality of Gjøvik, the existing database implementation is sufficient. The database implementation can easily be expanded or replaced. An additional database can also easily be added to store user credentials and facilitate a login system. Since the current database was intended to be an archive of the ice data, the group does not recommend that this database is also used for user credentials.

Looking back at the choice of DBMS, it is worth to question weather a relational database was the best choice. The group anticipated that the data would be more structured and require more complex connections. Considering the final implementation, it seems that a non-relational DB could have been sufficient. If the data structures remain more or less unchanged, a solution like AtlasDB or Firebase could be implemented instead of SQLite. Otherwise, a relational DB like SQLite or MySQL would remain a better choice.

## 8.2 Effect goals

If the system is deployed with the proposed sensor technologies, it can fully remove the need for manual measuring of the lake ice. This meets the desired effect goals of the group. Despite this, it is not possible to achieve to reach routine conditions outlined by DOT, but the LIDAR would be able to reduce the manual labor by supplying measurements.

## 8.3 Learning goals

In terms of programming, the group learned a lot. Initially with little understanding of mobile programming, the group gained enough knowledge to develop a decent mobile application. The mobile development process also included new insights into UI design and usability. The group has become far more familiar with Python programming, data processing, and ice related theory. With this, the group deems that the learning goals were reached.

### 8.3.1 Research of background theory

Researching the mechanics of ice formation and thickening was challenging due to limited literature on freshwater ice. Understanding of multiple concepts relating to physics was necessary to both interpret and utilize this research in a meaningful manner and a lot of time have been spent on trying to comprehend the physics behind ice. As few data sources were both freely available and relevant, the group had to to utilize data from Canada, which has the most similar climate to Norway.

Interpretation of LIDAR data require converting the raw sensor data into a suitable format. Moreover, discerning the significance of the output data was essential to repurposed the data for the measuring of ice thickness, a purpose which converged from its conventional use. These processes required the group to understand how LIDAR technologies work on a theoretical level.

## 8.4 Testing and quality assurance

Besides unit tests, a few other testing and quality assurance methods were employed. Postman was a heavily utilized tool for inspecting the JSON and GeoJSON files which resulted from the map creation process. Given the visual nature this process, Pyplot was very often used to visually inspect its results. As section 6.3 is exemplary of, the results of the map creation were not only verified for the aesthetics, but also verified to be geographically accurate. Pyplot was also used to plot the processed data derived from the NVE model.

## 8.5 Documentation and organization

Each component of the system is documented well. This includes comments, models, diagrams, a README file, and this report. The documentation not only served the group in mapping out and planning the system, but also serves as a valuable resource for future developers. The aim in creating this documentation was to allow easy and quick understanding of the system, thereby reducing the time and resources required for future developers to familiarize themselves with the system. Notably, the documentation includes an ER-diagram, domain model, system architecture diagram, and a README file.

The README file lists and explains all the endpoints. This includes the methods and paths to the endpoints, as well as explanations of usage and required parameters. The README also covers dependencies, known bugs, an open use license agreement, and required setup for the development of the system.

### 8.5.1 GitLab repository and version control

All of the projects code is contained within a single GitLab repository. To ensure smooth integration of code, it was forbidden to push directly to the main branch. This branch was dedicated to keep a fully functional and integrated code base. Each developer freely created new branches and pushed to those, but could only introduce new code to 'main' by first pulling the contents of 'main' into their own branch. Then, all merge conflicts would be resolved before pushing the branches contents into 'main'. This strategy worked well, and allowed three different sets of code to be integrated with minimal effort.

The group decided to push code to GitLab regularly and keep atomic commits. This means that most of the codes history and progress is kept within the repository's history. Anyone who may wish to look further into the progression of the development can look into the commit history of main or any of the other branches. This feature was also actively used by the developers for version control. There were multiple instances where it proved useful to roll back to an earlier commit, and atomic commits certainly made this process easier.

Proper organization of the code structure and the GitLab repository[1] was maintained throughout the entire project. The codebase was organized for easy navigation, sorted into files and folders with short and descriptive names. To aid in navigating the codebase, one can also refer to the system architecture diagram on Figure 2.5.

## 8.6 Sustainability

Developing with sustainability in mind is exceptionally important in a time of increasing dependence on software. Although this project never set any formal goals for sustainability, measures that improved the systems sustainability were taken. Aspects like computational optimization, power usage, and proper documentation were taken into consideration. These were not only considered to improve the quality of the result, but also to reduce the required resource usage. If the project period was longer, the group would certainly have spent more time in optimization and efficient power usage. The resulting computational efficiency was not ideal, and is probably one of the greater downfalls of the system.

Further research into ice behavior significantly contributes to broader climate studies. This is crucial as the understanding of dynamics of ice is essential in understanding climate change. Automated data collection through advanced technologies like satellite imaging and use of drone equipped with LIDAR not only reduces the need for physical expeditions but also enhances the efficiency and breadth of data gathering. This approach allows for continuous monitoring of remote and expansive ice-covered regions, contributing vital data for climate models and predictions. A system like ice map can aid scientists in better understanding and forecasting changes in ice. This can be useful in studying broader environmental impacts, which are key for developing, mitigating and adapting to climate change.

## 8.7 Method and process

As outlined in the pre-project plan, the group chose to use Kanban as the development method. The group used a shared Kanban board on Jira to organize tasks on a more detailed level, while a Gantt chart was created to organize the project on a larger scale. The Gantt chart divided the entire timeline into four main phases: planning, system modeling, implementation, and reporting. These phases forced the group to work systematically and ensure a reasonable design before starting the implementation. There were minor deviations from the expected start and end

[1] https://gitlab.stud.idi.ntnu.no/sarasdj/prog2900

dates, but such deviations were within anticipated bounds. The project plan was otherwise followed almost step by step.

Two to three physical meetings were held weekly. In two of the weekly meetings, the group discussed the overall advancement of the project progression. This time was also used to ensured that the group members were up to date on each other's contributions. The third weekly meeting included the two academic advisors, and discussed the groups progression, ideas for the system, and general feedback. To reiterate, the pre-project plan contains more detailed descriptions of the entire method and the project phases. It also includes a short risk analysis and more in depth discussions about the choice of programming technologies and usage of GitLab.

The division and allocation of the three main areas of the project also worked well. Each member was able to work independently within their assigned area, while maintaining active communication during the weekly meetings. The project organization worked perfectly, as it effectively managed organization, progression, administration, and related areas without any noteworthy problems.

## 8.8 AI usage

AI was primarily used in three ways in the coding process: learning, debugging, and generating test data. To allow the group to quickly learn Dart and Python, ChatGPT was consulted for explanations of unfamiliar topics and syntax. During the earlier stages of the application programming, ChatGPT was often asked to generate code, but this habit quickly waned as the group gained understanding of the Dart syntax and general workings of mobile applications. Any code that was initially generated by AI was altered, and never used as-is. Eventually, the group came to rely more on documentation, tutorials, and code examples. Tutorials and documentation like the one used to render the color coded map provided more compressive explanations.

When debugging tools proved insufficient for identifying or understanding bugs, ChatGPT was sometimes conferred. The bugs that ChatGPT was asked to fix were usually simple mistakes in syntax or logic that had evaded the developers. ChatGPT is very limited in its ability to identify bugs. Multiple instances arose where AI was the first attempt at identifying a bug, but proved to be entirely useless. For example, ChatGPT was unable to identify a bug that causes the shape selection of the map to not work. This bugs was eventually addressed by reviewing the code logic and reading documentation. Other bugs were fixed by reviewing existing code with similar logic, or completely scrapping the bug-riddled implementation.

ChatGPT was also asked to produce some test data. This use of ChatGPT was probably the most valuable. Figure 8.2 shows one example of a prompt and output where we asked ChatGPT to produce a few lines of test data for the database. In addition to coding aid, AI was used as writing aid. To improve the flow and formulation of the report, some paragraphs were passed through ChatGPT 3, but no text was fully generated by ChatGPT except for the license agreement in the README. All text that passed through the AI was of course vetted for mistakes and corrected.

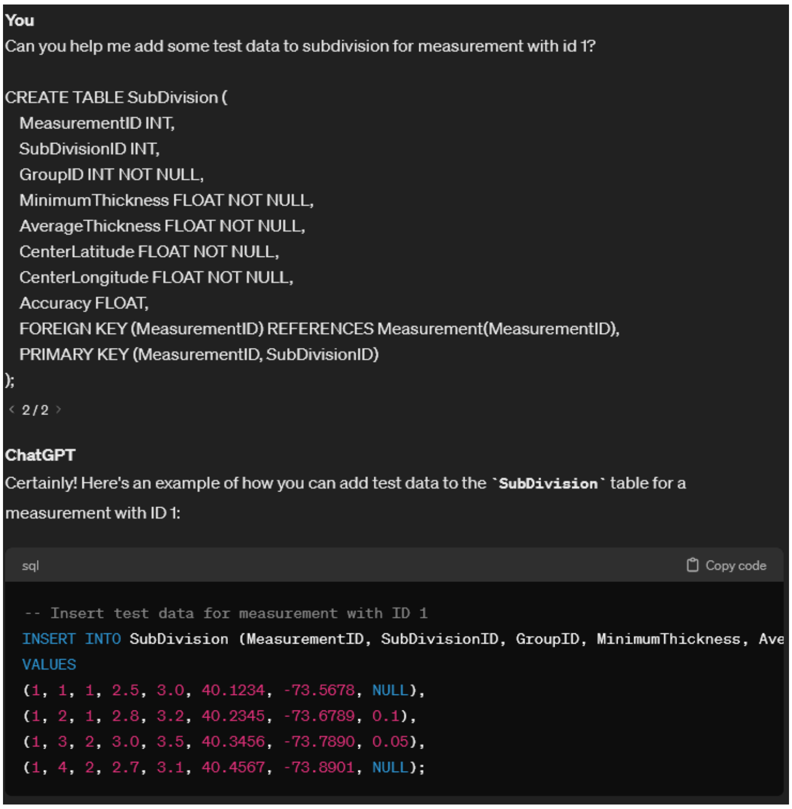



**Figure 8.2:** Example of AI generated test data

AI certainly saved the group time, mostly by identifying simpler problems, working like a quick search engine for learning Python and Dart, and generating test data. Various online tutorials and forums like StackOverFlow were often more useful tools for more comprehensive tasks.

## 8.9 Future work

Since the unit test coverage requirements was not fulfilled, the coverage should be increased by the inheritors of this project. Additionally, some of the problems which have been pointed out throughout this report should be addressed.

### 8.9.1 Sensors

The current thickness calculation method relies solely on the LIDAR's coordinated data, overlooking the potential benefits of utilizing the intensity data. Incorporating LIDAR's intensity data could significantly enhance ice thickness measurements, allowing for a more accurate assessments and differentiation of various ice layers. Additionally, intensity could aid in identifying debris within water or ice. Identifying debris has a potential to be expanded to create a rescue search system.

In terms of file management, achieving seamless integration for handling sensor data could be accomplished through automated processes rather than manual intervention. Utilizing the mobile SDK to develop a program for the drone to autonomously manage the files, aligning with our initial vision when creating the Arduino program. This approach would enhance the efficiency and is less prone to errors.

### 8.9.2 SentinelHub

Once the previously mentioned concrete numbers and training data [2] become accessible, using the same SVM to categorize ice will be an improvement. This will eliminate the need to manually fit BBoxes within a lake, resulting in more precise coordinates instead of interpolated ones. Additionally, enhancing the evalscripts for statistical analysis on ice will be possible once the SVM evalscripts are available. Moreover, utilizing SENTINEL-1 to penetrate cloud cover, despite its lower resolution and data availability, could be beneficial. Clouds often hinder satellite observations, so this approach could enhance data availability despite the lower quality.

### 8.9.3 Mobile app and map

As discussed in chapter 6, there are some shortcomings of the map generation process. These shortcomings are not detrimental, but they do negatively impact the user experience. These include the occurrence of disproportionately small subdivisions, not including fully enclosed geometries, and the partially manual process of adding new lakes. The design implementation of the application is fairly neat, but the app still requires usability testing and the inclusion of high contrast color themes. While these were omitted from the requirements due to time constraints, usability and accessibility must be ensured before deployment. Although the coloring in the map is based on the black ice thickness, it is admittedly a very basic implementation and could be improved by taking additional factors into account.

## 8.10 Alternative use cases

The application itself is not restricted to use in Norway. As long as the Overpass API or other suitable APIs can provide relation data for a given lake, it can be added to the system. All the code and documentation is written in English, which would allow non-Norwegian speakers to utilize any parts of the result. The only part of the system that pertains to Norway is the NVE model. The NVE model is included in the process of deciding the map colors, but could easily be omitted to only rely on sensor input. The model could also be swapped out with an alternative data source. Although the system was developed for usage on lakes, it may be suitable for other bodies of water too, including saltwater bodies.

The color coded map and the application could in theory be used for purposes other than measurement of ice thickness. The application could for example be modified to map flooding during summers. Any shape file could be added to the server, so it is not even restricted to lake shapes. Other geographical shapes like rivers or even countries could be mapped.

As the system logs and archives ice data, it provides a valuable resource for researching ice thickness and climate. All gathered data is saved, including measurements, timestamps, average thickness within each subdivision, and all individual measurements used in these averages. The ice run data and estimated ice thickness calculated with the NVE model are also stored in logs.

# Chapter 9

# Conclusion

Most of the goals that the group set out to achieve were met. The project resulted in a server that utilizes sensor and API data to determine ice thickness, and a mobile application that displays the processed data on a custom map. However, the practical functionality of the system still needs to be tested and deployed. Considering that the system was designed and developed entirely from scratch, and the groups initial lack of knowledge in ice, sensors, and mobile programming, we are pleased with the outcome and feel our expectations were met.

To address the remaining components of the project, we believe that enlisting experts is essential. This includes an expert in ice safety, whether a scientist or an experienced citizen, and ideally an electrical engineer with expertise in sensor technologies. Collaborating with such individuals will allow for the implementation of a more comprehensive ice assessment system that considers more factors than just the ice thickness. Nonetheless, we have provided a solid foundation for collecting, processing, and communicating data to the residents of Gjøvik. We are eager to see how the municipality continues to develop the IceMap system and the impact it will have on the citizens of Gjøvik.

# Bibliography


[1] 'My aurora forecast,' jRustonApps B.V. (2024), [Online]. Available: https://play.google.com/store/apps/details?id=com.jrustonapps.myauroraforecast&pli=1 (visited on 16/05/2024).

[2] G. D. Ashton. 'Ice in lakes and rivers - formation, thickness, melting,' Britannica. (2007), [Online]. Available: https://www.britannica.com/science/lake-ice/Ice-growth (visited on 08/05/2024).

[3] NVE. 'Stålis,' varsom.no. (n.d.), [Online]. Available: https://www.varsom.no/is/isskolen/istyper/stalis/ (visited on 02/05/2024).

[4] J. H. Halleraker, *stålis*, in *Store norske leksikon*, 15th Oct. 2023. [Online]. Available: https://snl.no/st%C3%A5lis (visited on 07/05/2024).

[5] 'Våris,' NVE. (n.d.), [Online]. Available: https://www.varsom.no/is/isskolen/istyper/varis/ (visited on 08/05/2024).

[6] 'Istykkelse og bæreevne,' NVE. (n.d), [Online]. Available: https://www.varsom.no/is/isskolen/istykkelse-og-baereevne/ (visited on 09/05/2024).

[7] K. Valtonen. 'Praktisk isbedömning, lektion 5 - sammanvägd bedömning.' (2003), [Online]. Available: https://www.thinkice.com/glaciology/sv/praktiskisbedomning/sammanvagd/index.htm (visited on 09/05/2024).

[8] Department of Transportation of the Government of the Northwest Territories, *Guidelines for safe ice construction*, Department of Transportation, 2015. [Online]. Available: https://www.inf.gov.nt.ca/sites/inf/files/resources/0016-001_norex_ice_road_constr._web.pdf (visited on 23/03/2024).

[9] A. Fitzgerald and W. Janse van Rensburg, *Limitiations of gold's formula for predicting ice thickness requirements for heavy equipment*, 2015. [Online]. Available: https://tspace.library.utoronto.ca/bitstream/1807/129630/1/cgj-2022-0464.pdf (visited on 24/03/2024).

[10] 'Om iskart.no,' NVE. (n.dm), [Online]. Available: https://www.varsom.no/is/om-is/kart-og-observasjoner/om-iskart-no/ (visited on 14/01/2024).

[11] C. Lussana, *seNorge observational gridded datasets. seNorge_2018, version 20.05*, 5th Aug. 2020. arXiv: 2008.02021[physics]. [Online]. Available: http://arxiv.org/abs/2008.02021 (visited on 09/05/2024).

[12] 'Sentinel-2 - missions - sentinel online,' Sentinel Online, European Space Agency. (2022), [Online]. Available: https://sentinel.esa.int/web/sentinel/missions/sentinel-2 (visited on 20/04/2024).

[13] 'Copernicus sentinels help classify river ice - sentinel success stories - sentinel online,' European Space Agency. (28th May 2020), [Online]. Available: https://sentinels.copernicus.eu/web/success-stories/-/copernicus-sentinels-help-classify-river-ice (visited on 20/04/2024).

[14] K. Chastko. 'Custom-scripts/sentinel-2/snow_cover_change at main · sentinel-hub/custom-scripts · github.' (2022), [Online]. Available: https://github.com/sentinel-hub/custom-scripts/tree/main/sentinel-2/snow_cover_change (visited on 11/03/2024).

[15] R. Jugier, R. Cremese, H. Fournier, N. Duran Gomez, G. Salgues and C. Thenoz, *On water and ice classification from sentinel-2 imagery using machine learning*, 19th Oct. 2022. DOI: 10.1002/essoar.10512606.1. [Online]. Available: https://essopenarchive.org/doi/full/10.1002/essoar.10512606.1 (visited on 20/04/2024).

[16] 'Sentinel-2 - missions - resolution and swath - sentinel handbook,' Sentinel Online, European Space Agency. (n.d.), [Online]. Available: https://copernicus.eu/missions/sentinel-2/instrument-payload/resolution-and-swath (visited on 30/04/2024).

[17] R. F. Cahalan and T. Várnai. 'Concept of new lidar measurements of snow and sea ice thickness,' NASA and UMBC JCET. (2007), [Online]. Available: https://earth.gsfc.nasa.gov/sites/default/files/lab_climate/i3rcimagearchive-17.pdf (visited on 07/02/2024).

[18] 'Mission - space lasers,' NASA. (n.d.), [Online]. Available: https://icesat-2.gsfc.nasa.gov/space-lasers (visited on 06/05/2024).

[19] P. Sergey M, L. Vasily N, K. Vladimir K, Y. Renat N and B. Alexey F. 'Ice thickness measurements by raman scattering,' Optics letters. (May 2014), [Online]. Available: https://doi.org/10.1364/OL.39.002573 (visited on 12/02/2024).

[20] F. Steven, K. Nathan and B. Marco. 'A decade-plus of antarctic sea ice thickness and volume estimates from cryosat-2 using a physical model and waveform fitting,' The Cryosphere. (May 2023), [Online]. Available: https://tc.copernicus.org/articles/17/2487/2023/tc-17-2487-2023.pdf (visited on 21/04/2024).

[21] 'Methods for measuring sea ice thickness in the arctic,' Deep Trekker. (2023), [Online]. Available: https://www.deeptrekker.com/resources/measuring-sea-ice-thickness (visited on 08/05/2024).

[22] K. Dai, A. Li, H. Zhang, S.-e. Chen and Y. Pan, *Surface damage quantification of postearthquake building based on terrestrial laser scan data*, ResearchGate, Jun. 2008. [Online]. Available: `https://www.researchgate.net/publication/325512595_Surface_damage_quantification_of_postearthquake_building_based_on_terrestrial_laser_scan_data` (visited on 07/02/2024), p. 10.

[23] A. G. Fountain, J. C. Fernandez-Diaz, M. Obryk, J. Levy, M. Gooseff, D. J. V. Horn, P. Morin and R. Shrestha, *High-resolution elevation mapping of the mcmurdo dry valleys, antarctica, and surrounding regions*, openTopography, 2017. [Online]. Available: `https://doi.org/10.5194/essd-9-435-2017` (visited on 12/02/2024), Earth Syst. Sci. Data, 9, 435–443, [dataset].

[24] S. Laxon, N. Peacock and D. Smith, *High interannual variability of sea ice thickness in the arctic region*, nature, 30th Oct. 2003. [Online]. Available: `https://www.nature.com/articles/nature02050` (visited on 21/04/2024).

[25] D. Yinke and C. Xiaomin. 'Metrology and measurement systems.' (2012), [Online]. Available: `https://bibliotekanauki.pl/articles/221894.pdf` (visited on 06/05/2024).

[26] W. Huang, Z. Li, X. Liu, H. Zhao, S. Guo and Q. Jia, 'Effective thermal conductivity of reservoir freshwater ice with attention to high temperature,' *Annals of Glaciology*, vol. 54, no. 62, pp. 189–195, 2013. DOI: `10.3189/2013AoG62A075`. [Online]. Available: `https://www.cambridge.org/core/journals/annals-of-glaciology/article/effective-thermal-conductivity-of-reservoir-freshwater-ice-with-attention-to-high-temperature/D1CC6510A0D4399061FE7031F741D151` (visited on 07/05/2024).

[27] L. Cui, J. Qin and X. Deng, 'Freshwater ice thickness apparatus based on differences in electrical resistance and temperature,' *Cold Regions Science and Technology*, vol. 119, pp. 37–46, 2015, ISSN: 0165-232X. DOI: `https://doi.org/10.1016/j.coldregions.2015.07.009`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0165232X15001688` (visited on 08/04/2024).

[28] *Supported deployment platforms*, Flutter, Feb. 2024. [Online]. Available: `https://docs.flutter.dev/reference/supported-platforms` (visited on 04/03/2024).

[29] Freepik. 'Cold-water icons.' (n.d), [Online]. Available: `https://www.flaticon.com/free-icons/cold-water` (visited on 20/05/2024).

[30] N. Karaaslan and volkut. 'Calculating new longitude, latitude from old + n meters.' Answer by Numan Karaaslan to StackOverflow post by Ben. (Nov. 2016), [Online]. Available: `https://stackoverflow.com/a/40471701` (visited on 22/04/2024).

[31] 'Dashboard,' Sinergise Solutions d.o.o. (n.d.), [Online]. Available: `https://www.sentinel-hub.com/develop/dashboard/` (visited on 12/05/2024).

# A Original task description

Oppgave 52 BPROG, BIDATA 2-4 stk

# Oppgavetittel: Kartlegging og varsling av istykkelse på innsjøer/vann

| | |
|---|---|
| Bedrift: | Gjøvik Kommune |
| Kontaktperson: | Ingun Revhaug |
| E-post: | Ingun.revhaug@gjovik.kommune.no |
| Telefon: | 94803195 |
| Lokasjon: | Gjøvik |

Beskrivelse av oppgaven

**Mulige problemstillinger/tilnærminger:**

1. Hva slags sensorteknologi egner seg best for måling av istykkelse?

2. Design og lag en app for kommunens innbyggere for status istykkelse og varsling ved endringer

3. Utvikle en prototype på ismålingstjeneste, komplett med sensor, IKT-løsning og innbyggerapp

**Bakgrunn**

Mange innsjøer, vann og tjern dekkes av is på vinteren i Innlandet. Islagte vann brukes til en rekke rekreasjonsaktiviteter, som isfiske, stå på skyøter og skiløyper legges ofte over islagte vann. Det kan imidlertid være vanskelig å vite når isen er tykk/trygg nok. I dag gjøres målinger av istykkelse manuelt, ved boring.

Både kommunen og innbyggere ønsker seg en løsning for automatisert måling av istykkelse, med tilhørende mulighet for varsling vårisen endrer status fra "utrygg" til "trygg" og vice versa.

https://www.ntnu.no/bridge/ajax/listing/apply/33357

# B Pre-project plan

# Ice thickness and safety mapping

## PROG2900 - Project plan

Joakim Aleksandersen
Sara Savanovic Djordjevic
Hoa Ben The Nguyen

NTNU

Department of Computer Sience
NTNU
Norway

January 2024

# Contents

# Figures



# Tables

# 1 Background and goals

## 1.1 Background

In countries like Norway that have abundant lakes and cold long winter seasons, frozen lakes can be great locations for recreational activities. This is the case at Mjøsa, Norways largest lake. Every winter Mjøsa attracts ice skaters, fishermen, skiers, and many others, but there are few who have extensive knowledge in ice safety. Recognizing this gap in knowledge, the municipality of Gjøvik wish to assure that its inhabitants can use the lake ice safely. Our objective is to increase the general public's understanding of lake ice safety and encourage the recreational use of the lake ice. To achieve this goal, we intend to develop an application centered around a color coded map of the ice thickness and safety on Mjøsa.

Morten Sangvik, an enthusiastic ice skater from Gjøvik, is one of many whom have shared personal experiences using the lake ice. The article posted in the local newsletter OA descirbes his ice-skating journey from Gjøvik to Hamar on January 13th 2024 with a friend (Børresen & Lund, 2024). Despite Sangvik and his friend having multiple years of experience on the ice, they were still met with public skepticism. The article showcases the prevailing lack of understanding among the general public as well as some reasons for this lack of knowledge. One very important reason mentioned is the prevalence of conflicting information. Merely 2 days after the release of Sangviks article, another newsletter posted an article which stated that the ice was not safe to use (Sangvik, 2024), the very opposite of what Sangvik claimed. This is one of many examples of news sources publishing information which may confuse readers and ultimately deter them from using the ice. Such instances make it clear that there is a lack of reliable and consistent sources that citizens can turn to for information on ice safety. In response, our initiative endeavors to provide a reliable resource, fostering a culture of informed decision-making and providing better indicators of safety for those venturing onto the frozen expanses of lakes like Mjøsa.

## 1.2 Goals

Currently, the municipality does not have a system to inform its inhabitants of ice thickness and safety. The closest existing mechanism is the conduction of manual measurements. A handful of private people in Gjøvik conduct manual measurements by venturing out onto frozen bodies of water and drilling through the ice. This is both time consuming and risky, especially if done by inexperienced individuals. One of the municipalities primary wishes is to reduce the need for manual measurements. Hence, they have requested an automated sensor solution alongside a user application. We have decided to focus the application around a color coded map. This map should distinguish between safe and dangerous areas of the ice. The color coding system must be determined by a set of data including at least one sensor solution, and weather/geological data from third party application program interface (API). In addition to the rendering of the map, the application must clearly communicate the accuracy of all data and include important safety precautions when venturing the ice.

### 1.2.1 Effect goals

As developers, we are able to measure whether the application effectively communicates ice safety and increases user interest in utilizing the ice safely for recreational activities. These metrics may be measured through quick user surveys. Beyond these two metrics, there are multiple desired effects that can only be measured once the application has been deployed, preferably for at least 1 year. Such metrics include whether the application increases citizens knowledge on ice safety, and weather it contributes in reducing the number of ice related accidents. If the number of accidents does decrease, it may also be interesting to research if the application also reduces the severity of accidents, but they are beyond the scope of the project.

### 1.2.2 Learning goals

As this is our first experience dedicating a minimum of 25 hours per week to a single project, a major learning objective for us should be the concentration of our efforts primarily, if not exclusively, on this one project. As this is very similar to how many of us will work in the future it is imperative for everyone's learning opportunities to work under these conditions. Another critical aspect of the project is the fact all of the data we deal with can have serious repercussions. This means we have to be especially critical when analysing data and when representing data as facts. Learning to deal with data that concerns human safety, and knowledge about what is fine to present as the ”truth” in safety is quite simply the most important part of the project.

In the solution we wish to utilize sensor-technology, new and old. There is incentive to get an deeper knowledge of how we can use software to utilize hardware to achieve our goals. This may include aspects such as computer vision and general usage of sensors technology. We will also attempt to integrate preexisting sensor technology that are being used to this day such as the Sentinel satellites that Varsom uses in their ice map (Varsom, n.d.).

- Learn a new mobile development technology, such as Flutter or React Native
- Gain experience in communicating and cooperating with a client
- Employ third-party databases and APIs with a level of complexity exceeding prior experience

- Increase knowledge related to the interaction of hardware and software
- Learn to design and structure comprehensive systems
- Increase knowledge of data accuracy and integrity

# 2 Scope

## 2.1 Problem area

This project naturally encompasses a diverse range of disciplines. Relevant disciplines may include geometry, algebra, cartography, physics, geology, meteorology, and glaciology. Despite the group's limited expertise in such areas, they are crucial for ensuring accurate mapping of ice safety. Moreover, as a group predominantly skilled in programming, the integration of hardware may pose additional challenges. As Gjøvik municipality has provided an open description, the group has a lot of freedom in how the project is executed and how the application is implemented. Hence, a substantial portion of the project time frame is allocated for research and system modelling. For more detailed insights into the planned project execution, please refer to section 5.

## 2.2 Delimitations

A substantial amount of the project period must be allocated to research and modelling, as the data we are dealing with data that may affect safety of people. The group has therefore decided to focus on making a solid proof-of-concept, rather than aiming for an off-the-shelf application. In practice this means that more time will be spent on back-end development. This decision has been made with the groups existing experience and interests in mind. Rather than making an extensive User Interface (UI), the group aims to create a bug-free prototype with only essential UI components. Further development of the UI will be left to the municipality. This solution could be advantageous for the municipality, as they can seamlessly integrate their standardized UI aesthetic rather than having to replace our UI implementation.

Further, the application will initially only map and display information of specific parts of Mjøsa. We will start by mapping out the water that goes along Mjøspromenaden, since we anticipate that this is where people are most likely to utilize the ice. While such an application may also be useful on other bodies of water, we have chosen to use Mjøsa as a starting point. Mjøsa has been deemed as the ideal starting points as the municipality's population is concentrated around the coast of Mjøsa. Despite this initial focus, the application will be designed with scalability and modularity in mind, ensuring the ease of potential future expansions.

## 2.3 Project description

The project aims to create a mobile application for visualizing lake ice thickness. The app will feature a color-coded, partially interactive map indicating ice thickness and calculation accuracy. A simple calculation model will determine data utilization and treatment, influencing the map's color coding. The UI simple with minimal features, with a greater focus on making sure that the information from the back-end is correct. Ice thickness will be calculated using satellite data, weather data and a sensor solution, preferably mounted on an automated vehicle. The primary goal will be to implement a sensor solution utilizing a drone. As a sort of security net, the application should also accommodate a set of stationary sensors. The drone solution will ensure a larger coverage of the lake, while stationary sensor will ensure measurements in the case that the drone is not operational.

### 2.3.1 Preliminary requirement specification

To give a better idea of which features and functionalities the application should include, a preliminary requirements specification has been made. This specification will be revised after more research has been conducted.

**System requirements**

- Data integrity: Satellite and weather data must come from reliable sources, such as state organizations, research institutes, or companies that work in coordination with such organizations.
- Data Accuracy: Each measurement must be adjusted for potential sensor solution discrepancies. For instance, if a sensor with a +/- 2.5cm accuracy measures the ice to be 10cm thick, the output should be 7.5cm. In other words, a measurement must always be adjusted towards the thinnest possible alternative.
- Response time: Upon opening the application, the map must be fully loaded within 10 seconds.
- Safety categorization: Ice safety will be based on the models from World Meteorology Organization (W.M.O.) egg code[1] and Ice Thickness Safety Chart[2].
- Sensors: A system that does not surpass the total cost of 100 000 NOK.
- Data storage: Each measurement and the time of the measurement must be stored in an internal database.
- Scalability: The application must be built to allow for changes of API solutions and implementation of additional APIs.

[1] https://cryo.met.no/en/understanding-ice-charts
[2] https://www.almanac.com/ice-thickness-safety-chart

- Calculation model: Calculations must be redone at least every 12th hour, or twice a day in other words.

**Functional requirements**

- Map: Color-coded will be the primary data visualization method. Predefined locations on the map will serve as measurement points for the sensor(s). The measured points will exhibit a strong color, gradually fading in a predetermined radius from each measurement point.
- Graphs: A user may click on a colored section of the map to view a more detailed description of the data that was used to calculate the color/safety level.
- Data exportation: Any user, logged in or not, can export any data that is used in the calculation of the safety level. This includes measurements from sensors and surveyors, as well as supplementary data such as weather conditions sourced from weather stations and APIs, and satellite imagery for assessment of ice coverage and changes over time.

**Other requirements**

- Unit tests: the back-end of the code must have a minimum coverage of 60% with automated unit tests.
- Service Level Agreement(SLA): a SLA must be made where the risk of any accidents resulting from the use of the application is transferred to the publisher of the application.

# 3 Proposed technologies and data sources

## 3.1 Programming languages, frameworks and libraries

### 3.1.1 Data visualization

Python serves as a powerful tool for data visualization, being widely used and sought after, meticulously documented, highly flexible, and having a large and active community (Mikke Goes Coding, 2022). Importantly, Python provides a wide range of libraries and frameworks that are suited for data visualization in various forms. Notable libraries include GeoPandas, Plotly, and Folium, each offering capabilities for generating maps and graphs.

### 3.1.2 Mobile programming

There are many frameworks and Software Development Kits (SDK) for mobile programming, all with their own benefits. When researching possible technologies, we have both considered the groups prior experience with related technologies, and aspects that make the technology suited for this specific project.
The selection has been reduced to Flutter or React Native. These are some of the best tools that utilise a single cross-platform code base. Including strong community support and hot reload feature that enables a real-time visual feedback upon changes in code, enabling a faster development process. But there are distinct differences that makes the choice difficult. The following table showcases some pros and cons of both technologies, as shown in Table 1.

Table 1: Flutter vs React Native

| **Flutter** | **React Native** |
|---|---|
| SDK | JavaScript(JS) framework |
| Flexible and expressive features | JS framework |
| Support for modern interfaces | Easy manipulate native component |
| High performance, architecture + dart, speed | Large active user community |
| Rich customization | Highly sought after in the industry |
| Organized interface | Use familiar language Java Script |
| Younger than react native | Fast, but not as fast as flutter |
| Must learn dart language | User interface is complicated |
| Large file size | Is in development |
| Limited libraries | Quality consistency |

The choice between these two technologies will be made after the completion of a more comprehensive requirement specification.

## 3.2 Sensor technologies

There are many sensor technologies that may be relevant for this project. While researching such technologies, there is a budget that has to be considered. The higher end of this budget is 100 000 NOK. This threshold represents the maximum expenditure the municipality can allocate without necessitating external investors or intervention from higher state authorities. Solutions that go beyond this threshold will therefore not be considered, unless external investors express interest. Additionally, solutions that require a human to walk/ski/skate on the ice will be excluded. These restrictions immediately exclude most Radar and Sonar solution. The following solutions are the ones which may be viable, with a estimated minimum cost:

**Drone**

- Laser altimeter mounted on drone : 2000 NOK
- Single drone equipped with camera : 500 NOK

**Stationary sensor**

- System of 5 magnetized strip sensors : 830 NOK
- R-T ice-thickness sensor : 10 000 NOK

## 3.3 Other data sources

In addition to data gathered from sensors and manual measurements, the following sources may provide information which may be relevant for creating a calculation or prediction model:

- Weather stations on Mjøsa
- Weather data from third party APIs such as Yr
- Depth and area measurements from the Norwegian Energy and Water Resources Directorate
- NaturalEarth database of rivers and lakes
- World Meteorological Organization (WMO EGG model)
- Copernicus Programme Sentinel satelite API [3]

[3] https://sentinel.esa.int/web/sentinel/copernicus

# 4 Planning, organization, and execution

## 4.1 Agile development methodologies

Amongst the many popular development methods, we are most inclined to utilise an agile methodology. Agile methodologies include Kanban, Extreme Programming (XP), and the very popular Scrum. The main benefits of Agile methodologies for our group is their iterative nature, which is preferable when a complete requirement specification is not provided by the client.

## 4.2 Chosen methodology

As the client has not expressed any preferences for development methodologies, we stand free to choose whatever suits the group most. Considering the nature of the project and previous experiences, we have chosen Kanban. The reason we have chosen Kanban over the more popular Scrum is due to previous negative experiences with Scrum. For a larger team, it may make sense to invest time into administration, project management, and formalities. For a team of only three developers the time and effort required simply outweigh the associated benefits. Spending time estimating and allocating time for each small task, planning sprints, holding sprint meetings, and daily scrum is simply unnecessary and complicates the process. Another reason for Kanban being more desirable is the Just In Time (JIT) delivery system. One of the main parts of the project is combining multiple data sources to find the answer on how thick the ice is at a given time and space, and the amount of data available being effectively limitless. A result of this is that the probability of us integrating absolutely all data is low, thus JIT can be used to further integrate more and more data until deadline for delivery.

### 4.2.1 Documentation, quality assurance and configuration management

We will utilize Jira and GitLab as the primary project management tools. To track working hours, gauge division of tasks, and porblem areas, we will use a shared Excel sheet. The sheet is divided into five categories: development, debugging, documentation, research, and other. Any documentation that is not natural to keep in the GitLab repo or OverLeaf, such as models and research notes, are to be kept in a shared OneDrive folder. As documentation is an essential part of the project, we chose to define the following guidelines to assure proper documentation:

- All tasks should be put on the Kanban board on Jira
- The work in progress column in the Kanban board is limited to a fixed boundary that must not be exceeded. This boundary is set to 3 tasks per member. If a task is too large

to be tackled by a single member, the task may be split into a series of smaller tasks after discussion with the other group members.

- Lead time (time from task creation to completion) and cycle time (time from a task initiation to completion) are metrics that will be used to identify bottlenecks and ensure that tasks are divided into appropriate sizes. These metrics are calculated by Jira.
- Commits should be atomic and have a short and informative message. This is to ensure that every team member has a base understanding of which changes have been made in the commit.
- All code must be sufficiently commented following the commenting standards of the specific language.
- Class and variable names must be informative and follow the standards of the specific languages.
- Every member is encouraged to document daily and consistently.
- Research sources should be written down and included in the report immediately.
- At the end of every milestone/internal deadline we will use up to a week to document
- If members are unsure where to put information, we have a Discord channel dedicated to communication between team members to ensure a low threshold for asking any applicable questions.

## 4.3 Roles

We have created three formal roles: Sara will serve as the project leader; Joakim will take on the role of documentation and storage leader; and Hoa will be the work log leader.

- Project leader: responsible for ensuring that individual tasks and the overall project are progressing as expected.
- File storage leader: responsible for ensuring that all project files are located at their appropriate locations; Jira, Discord, OneDrive, GitLab, or Overleaf, and that these platforms are kept neat and tidy.
- Work log leader: responsible for ensuring that all members consistently and accurately log their work hours in the dedicated Excel spread sheet.

### 4.3.1 Client

The client for the project is Gjøvik municipality. The municipality has provided a short description of their desired product. The client has given us a large degree of freedom in how we chose to design and implement the product. While they will be part of the development process, their level of active involvement may be less than what is typically expected. The product description provided by the municipality can be found in the appendix.

## 4.4 Execution

We have chosen to divide the project period into 4 main phases, these being a planning phase, system modelling phase, implementation phase, and a reporting phase. The planning phase shall take up the first 3 weeks of the project period and focuses on mapping out the foundation of the project as well as conducting preliminary research. Research should be conducted on programming languages, libraries and frameworks, existing technologies, and ice thickness calculation models. The system modelling phase follows directly after the planning phase. This phase should result in a clear plan for the system architecture, choice of technologies, requirement specification, and use case diagram. The implementation phase is where coding and testing begins. This phase should result in a MVP and a prototype that are both thoroughly tested. Testing should be conducted throughout the entire phase.

The last two weeks of the implementation phase are dedicated solely to debugging and polishing the prototype. The last phase is the reporting phase which begins simultaneously as the system modelling phase, and ends three weeks after the implementation phase is completed. We plan on following the proposed phases rigidly. If the result is not satisfactory at the end of a phase, we will try to accept the result as it is and proceed to the next phase without delays. The end of each phase is considered a milestone. Additionally, a handful of other milestones are specified within some of the phases. The progression strategy aims create a smooth project progression by prioritizing the completion of important tasks over detail work and minor bugs. Figure 1 and figure 2 on page 13 and 14 show the planned project trajectory and milestones.

## 4.5 Status meetings and decision points

Status meetings are held every other Thursday. These meetings are intended to be informal meetings to clarify the status of all tasks in the current project phase. If the project is not progressing as expected, these meetings should be utilized to take management and administration decisions to ensure that deadlines and milestones are met.

## 4.6 Inspections and testing

The primary form of testing will be a set of automated unit tests. These tests should have a relatively high coverage. The coverage goal is a minimum of 60%. If sensors are available to us in the project period, we may attempt to verify that the sensor data is consistent with manual measurements, but this is not a priority. As the development will focus on back-end, extensive user testing will not be prioritized. Upon completing the MVP, a quick user survey may be held to gauge the overall usability of the application.

Figure 1: Gantt-chart

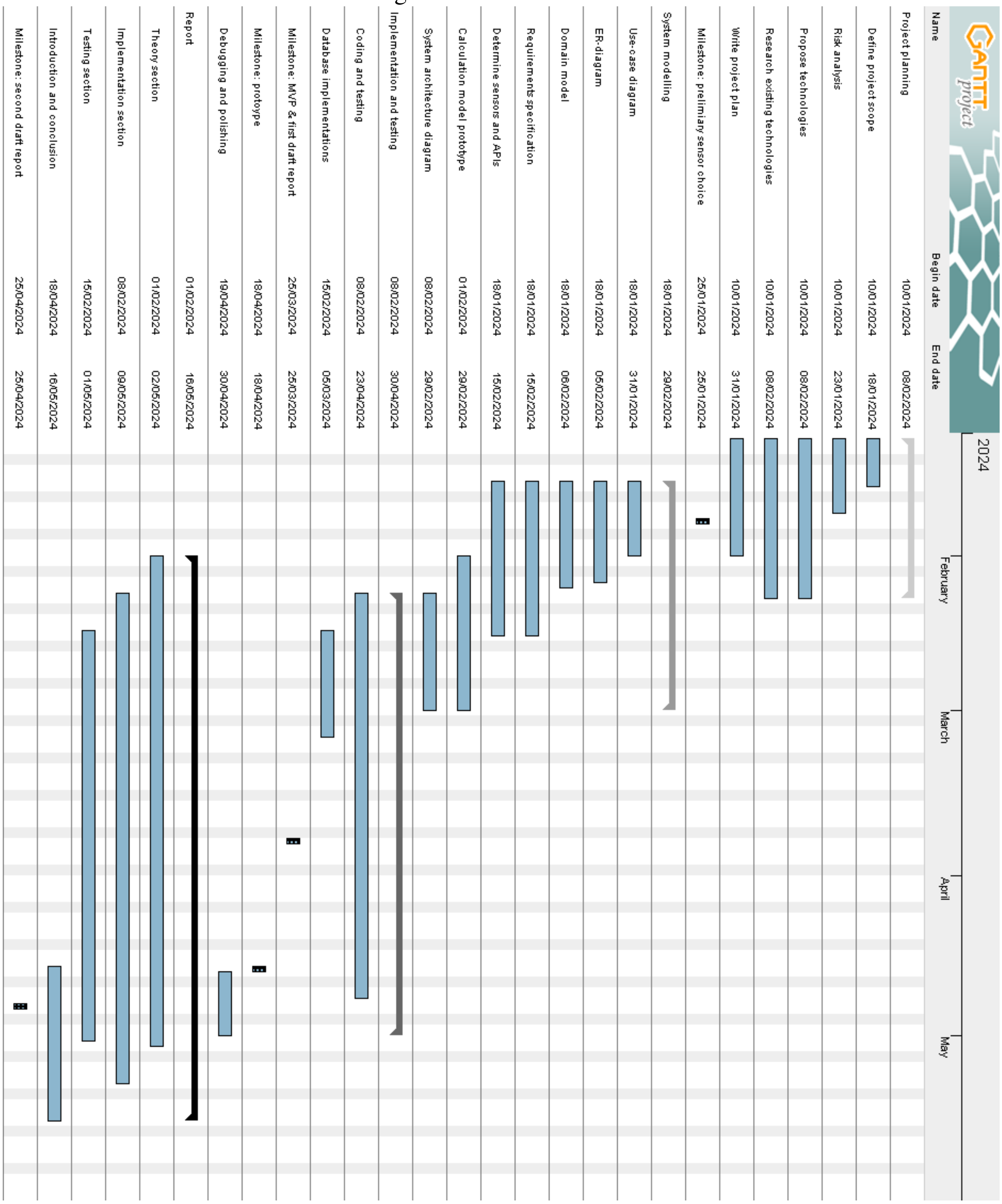


Figure 2: Tasks from Gantt-chart

## Tasks

| Name | Begin date | End date |
|---|---|---|
| Project planning | 10/01/2024 | 08/02/2024 |
| Define project scope | 10/01/2024 | 18/01/2024 |
| Risk analysis | 10/01/2024 | 23/01/2024 |
| Propose technologies | 10/01/2024 | 08/02/2024 |
| Research existing technologies | 10/01/2024 | 08/02/2024 |
| Write project plan | 10/01/2024 | 31/01/2024 |
| Milestone: prelimiary sensor choice | 25/01/2024 | 25/01/2024 |
| System modelling | 18/01/2024 | 29/02/2024 |
| Use-case diagram | 18/01/2024 | 31/01/2024 |
| ER-diagram | 18/01/2024 | 05/02/2024 |
| Domain model | 18/01/2024 | 06/02/2024 |
| Requirements specification | 18/01/2024 | 15/02/2024 |
| Determine sensors and APIs | 18/01/2024 | 15/02/2024 |
| Calculation model prototype | 01/02/2024 | 29/02/2024 |
| System architecture diagram | 08/02/2024 | 29/02/2024 |
| Implementation and testing | 08/02/2024 | 30/04/2024 |
| Coding and testing | 08/02/2024 | 23/04/2024 |
| Database implementations | 15/02/2024 | 05/03/2024 |
| Milestone: MVP & first draft report | 25/03/2024 | 25/03/2024 |
| Milestone: protoype | 18/04/2024 | 18/04/2024 |
| Debugging and polishing | 19/04/2024 | 30/04/2024 |
| Report | 01/02/2024 | 16/05/2024 |
| Theory section | 01/02/2024 | 02/05/2024 |
| Implementation section | 08/02/2024 | 09/05/2024 |
| Testing section | 15/02/2024 | 01/05/2024 |
| Introduction and conclusion | 18/04/2024 | 16/05/2024 |
| Milestone: second draft report | 25/04/2024 | 25/04/2024 |

# 5 Risk analysis

## 5.1 Identification and categorization

Table 2 on the next page shows an overview over risks that we have deemed to be likely to occur, have meaningful consequences, and require a collective effort to manage. Here we consider both the assumed probability of the risk coming to fruition as well as the consequences of the risk itself. The probabilities and the severity of the consequences are both divided into 3 categories. A risk with the consequence 'Slight' is a risk that if it occurs, can be resolved or managed within a single work day with minimal effort. A risk with the consequence 'Moderate' is one that may require up to a week to manage and require smaller parts of the implementation and/or documentation to be reworked. A risk with the consequence 'Sever' is one that may require multiple weeks to manage, remodelling of the entire system, reworking major parts of the implementation, and/or significant alterations to major parts of the documentation.

Table 2: Assumed risks

| **Num** | **Risk** | **Probability** | **Consequence** |
|---|---|---|---|
| 1 | The application still contains minor bugs at the end of the implementation phase | Possible | Slight |
| 2 | Data we receive is unreliable or false | Possible | Severe |
| 3 | A member is suddenly unable to participate over an extended period | Unlikely | Severe |
| 4 | A piece of documentation is lost | Unlikely | Moderate |
| 5 | A technology or API must be changed due to becoming unavailable or the discovery of unforeseen limitations | Possible | Moderate |
| 6 | Other course work unexpectedly increases, taking time and focus away from the project | Possible | Moderate |
| 7 | We cannot ensure reliable safety of the information published in the application | Likely | Severe |
| 8 | Milestone or internal deadline is not met | Likely | Moderate |
| 9 | Major conflict within the group | Unlikely | Severe |
| 10 | Loss of or significant damage to a major part source code or report | Unlikely | Severe |
| 11 | A crucial requirement is not met, or the application contains major bugs by the end of the implementation phase | Possible | Severe |
| 12 | Final application is not user friendly | Likely | Moderate |

## 5.2 Mitigation and management

Table 3 shows mitigation and management strategies for each of the risks listed in table 2. Some of the risks do not have mitigation strategies as their occurrence is beyond our control. Others do not have management strategies as we chose to accept the risk if it occurs. The numbers in table 3 corresponds with the number of the risk in table 2.

Table 3: Mitigation and management strategies

| Num | Mitigation | Management |
|---|---|---|
| 1 | An entire two week period at the end of the implementation phase is dedicated to debugging in order to minimise the existence of bugs | No management |
| 2 | Idealy we get access to an verified source of data or multiple sources where we can cross reference them | If we are unable to receive reliable data, we will then be forced to set the scope of the project as a proof of concept and not something that the public can use as it will be far to dangerous to release predictions based on faulty data |
| 3 | No mitigation | The project scope will be reduced, starting with the least crucial requirements |
| 4 | A shared OneDrive folder is dedicated to storing various documentation, and one member is given the role of documentation and storage leader | The cause of the loss must be unveiled and prevented from reoccurring. The documentation must be remade |
| 5 | We try to use more popular and well documented technologies with large user bases, or get explicit access to databases we wish to use | The group must collectively find and decide upon a suitable alternative. If a practical alternative cannot be found, the project must depend on a theoretical solution |
| 6 | No mitigation | The group must create a strategy to make up the lost time in the following weeks, for example by holding additional meetings or adding a couple of extra weekly work hours. |
| 7 | We will do our best to research the subject and ask professionals in order to have knowledge of safety standards and implement them to the best of our ability | If we see that the data we create is too unreliable it should not be release to the public, but rather be given to Gjøvik municipality as guideline instead of facts |

| | | |
|---|---|---|
| 8 | We use Kanban with loose dates for deadlines where the deadlines should be completed within 4 days of a set date. | If the allocated time passes, the group shall accept the result as it is and move onto the next deadline. If the result strays considerably from the requirements, reduction of the project scope will be discussed |
| 9 | The group holds at least 2 meetings weekly where members are encouraged to work together, express any grievances, and to gain insight into work done by other members | See group contract |
| 10 | Platforms and software like GitLab, OverLeaf, and OneDrive are used to store code and documentation | The group must collectively decide which parts of the lost code/documentation must be redone, and which must be abandoned |
| 11 | Consistent planning, administration and management should ensure that crucial components of the application are at an acceptable state by the end of the implementation phase | The group may extend the phase by up to 1 week and may have to work additional hours. If a requirement cannot be implemented due to practical obstacles, a theoretical replacement must be developed |
| 12 | We will conduct an short user test in order to determine if the applicataion is user friendly enough | If the application is not user friendly enough we will have to spend more time to make the UI be intuitive. |

# References


Børresen, E., & Lund, S. A. (2024). *Dette synet vekket bekymring: – Er det jeg som er redd eller er det dumdristig?* Retrieved from https://www.oa.no/dette-synet-vekket-bekymring-er-det-jeg-som-er-redd-eller-er-det-dumdristig/s/5-35-1877337?onboarding_mode=true (Published January 15, 2024. Accessed January 25, 2024)

Mikke Goes Coding. (2022). *12 Essential Advantages of Python (Why Learn Python in 2022).* Retrieved from https://mikkegoes.com/advantages-of-python/ (Published September 4, 2022. Accessed January 25, 2024)

Sangvik, M. (2024). *Det er ikke sånn at isen er farlig.* Retrieved from https://www.oa.no/dette-synet-vekket-bekymring-er-det-jeg-som-er-redd-eller-er-det-dumdristig/s/5-35-1877337?onboarding_mode=true (Published January 18, 2024. Accessed January 25, 2024)

Varsom. (n.d.). *Slik lager vi isvarselet.* Retrieved from https://www.varsom.no/is/isvarsling/slik-lager-vi-isvarsel/ (Accessed January 22, 2024)

# C Gantt Chart

# Lake Ice Mapping

31 Jan 2024

## NTNU

| | |
|---|---|
| **Project manager** | |
| **Project dates** | 10 Jan 2024 - 17 May 2024 |
| | |
| **Completion** | 0% |
| **Tasks** | 27 |
| **Resources** | 0 |

This Gantt chart describes the proposed project trajectory of the Bachelor thesis project for mapping of lake ice thickness, NTNU Spirng 2024, BPROG group 206.

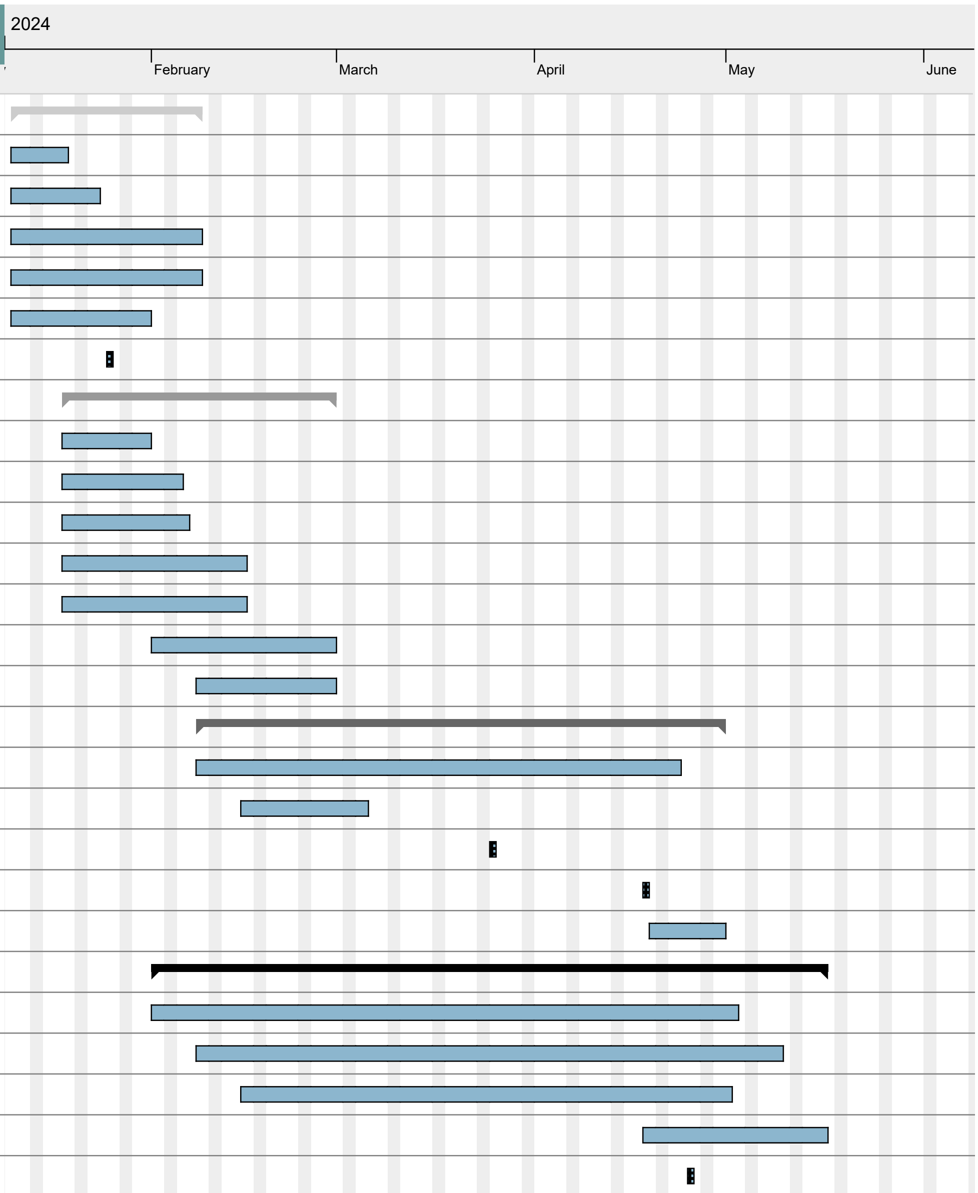
# Tasks

| Name | Begin date | End date |
|---|---|---|
| Project planning | 10/01/2024 | 08/02/2024 |
| Define project scope | 10/01/2024 | 18/01/2024 |
| Risk analysis | 10/01/2024 | 23/01/2024 |
| Propose technologies | 10/01/2024 | 08/02/2024 |
| Research existing technologies | 10/01/2024 | 08/02/2024 |
| Write project plan | 10/01/2024 | 31/01/2024 |
| Milestone: prelimiary sensor choice | 25/01/2024 | 25/01/2024 |
| System modelling | 18/01/2024 | 29/02/2024 |
| Use-case diagram | 18/01/2024 | 31/01/2024 |
| ER-diagram | 18/01/2024 | 05/02/2024 |
| Domain model | 18/01/2024 | 06/02/2024 |
| Requirements specification | 18/01/2024 | 15/02/2024 |
| Determine sensors and APIs | 18/01/2024 | 15/02/2024 |
| Calculation model prototype | 01/02/2024 | 29/02/2024 |
| System architecture diagram | 08/02/2024 | 29/02/2024 |
| Implementation and testing | 08/02/2024 | 30/04/2024 |
| Coding and testing | 08/02/2024 | 23/04/2024 |
| Database implementations | 15/02/2024 | 05/03/2024 |
| Milestone: MVP & first draft report | 25/03/2024 | 25/03/2024 |
| Milestone: protoype | 18/04/2024 | 18/04/2024 |
| Debugging and polishing | 19/04/2024 | 30/04/2024 |
| Report | 01/02/2024 | 16/05/2024 |
| Theory section | 01/02/2024 | 02/05/2024 |
| Implementation section | 08/02/2024 | 09/05/2024 |
| Testing section | 15/02/2024 | 01/05/2024 |
| Introduction and conclusion | 18/04/2024 | 16/05/2024 |
| Milestone: second draft report | 25/04/2024 | 25/04/2024 |

# Gantt Chart

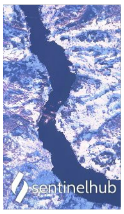


| Name | Begin date | End date |
|---|---|---|
| Project planning | 10/01/2024 | 08/02/2024 |
| Define project scope | 10/01/2024 | 18/01/2024 |
| Risk analysis | 10/01/2024 | 23/01/2024 |
| Propose technologies | 10/01/2024 | 08/02/2024 |
| Research existing technologies | 10/01/2024 | 08/02/2024 |
| Write project plan | 10/01/2024 | 31/01/2024 |
| Milestone: prelimiary sensor choice | 25/01/2024 | 25/01/2024 |
| System modelling | 18/01/2024 | 29/02/2024 |
| Use-case diagram | 18/01/2024 | 31/01/2024 |
| ER-diagram | 18/01/2024 | 05/02/2024 |
| Domain model | 18/01/2024 | 06/02/2024 |
| Requirements specification | 18/01/2024 | 15/02/2024 |
| Determine sensors and APIs | 18/01/2024 | 15/02/2024 |
| Calculation model prototype | 01/02/2024 | 29/02/2024 |
| System architecture diagram | 08/02/2024 | 29/02/2024 |
| Implementation and testing | 08/02/2024 | 30/04/2024 |
| Coding and testing | 08/02/2024 | 23/04/2024 |
| Database implementations | 15/02/2024 | 05/03/2024 |
| Milestone: MVP & first draft report | 25/03/2024 | 25/03/2024 |
| Milestone: protoype | 18/04/2024 | 18/04/2024 |
| Debugging and polishing | 19/04/2024 | 30/04/2024 |
| Report | 01/02/2024 | 16/05/2024 |
| Theory section | 01/02/2024 | 02/05/2024 |
| Implementation section | 08/02/2024 | 09/05/2024 |
| Testing section | 15/02/2024 | 01/05/2024 |
| Introduction and conclusion | 18/04/2024 | 16/05/2024 |
| Milestone: second draft report | 25/04/2024 | 25/04/2024 |

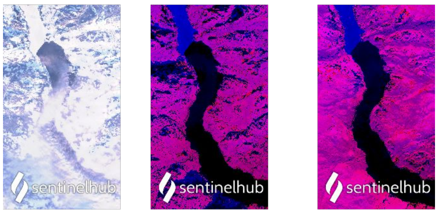


# D SENTINEL hub, Mjøsa over time

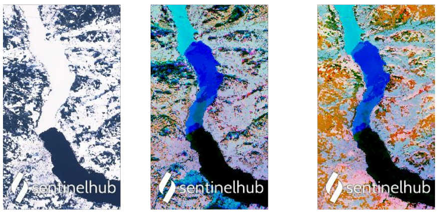
22-12-27



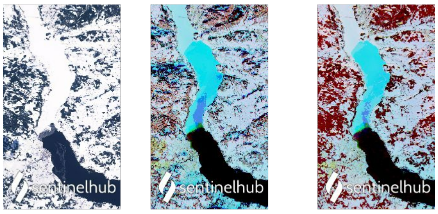
23-01-27 NDMI Snow index



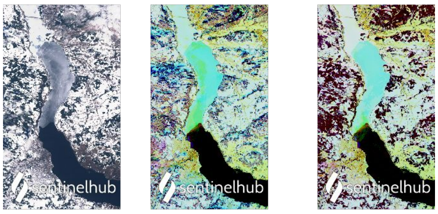
23-02-27



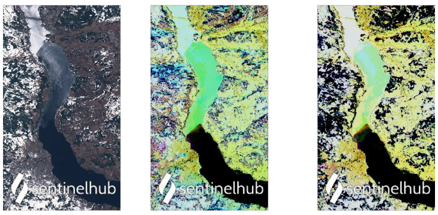
23-02-27



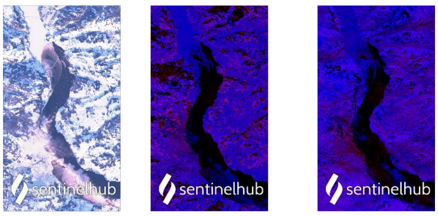
23-04-17



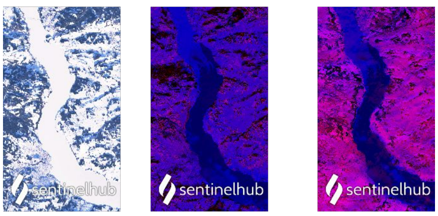
23-04-27



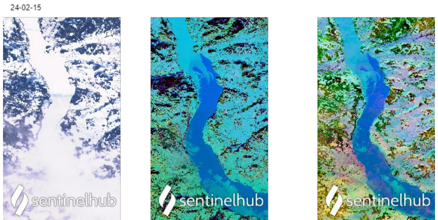
24-01-06



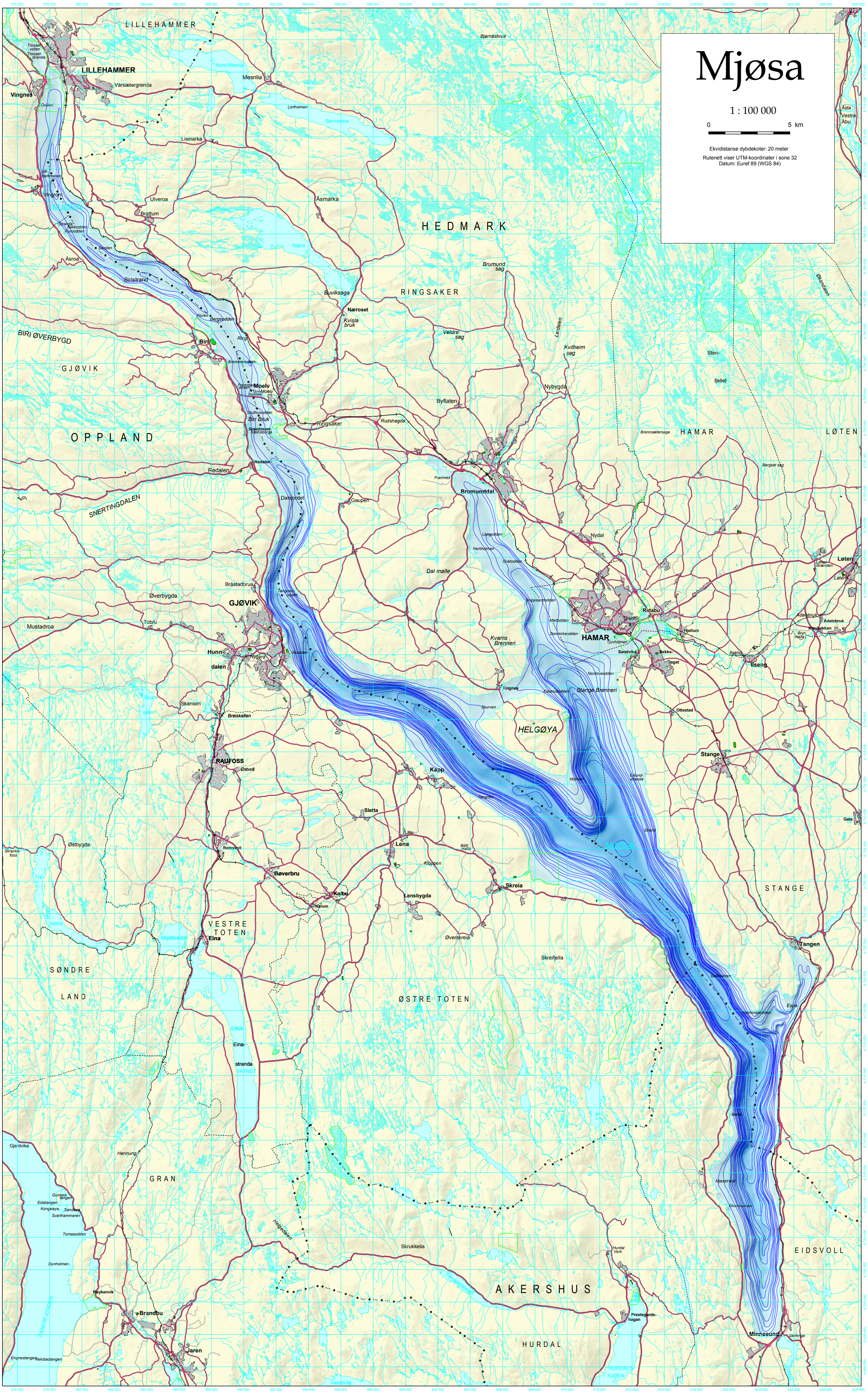
24-01-25



24-02-15



# E Depth map, Mjøsa


Mjøsa
1 : 100 000
0
5 km
Ekvidistanse dybdekoter: 20 meter
Rutenett viser UTM-koordinater i sone 32
Datum: Euref 89 (WGS 84)
LILLEHAMMER
Vingnes
Vingrom
Biri
Moelv
Ringsaker
GJØVIK
HAMAR
HELGØYA
Brumunddal
RAUFOSS
Kapp
Skreia
Lena
Bøverbru
Tangen
Minnesund
Stange
Løten
Ilseng
OPPLAND
HEDMARK
AKERSHUS
RINGSAKER
GJØVIK
STANGE
EIDSVOLL
HURDAL
GRAN
SØNDRE LAND
VESTRE TOTEN
ØSTRE TOTEN
BIRI ØVERBYGD
SNERTINGDALEN

# F Evalscript using 30 days NDMI, NDWI and NDVI

```
//VERSION=3 (auto-converted from 1)
/*
Author: Karl Chastko OG Joakim Aleksandersen
*/


function setup() {
  return {
    input: [{
      bands: [
          "B03",
          "B11",
  "B08"
      ]
    }],
    output: { bands: 3 },
    mosaicking: "ORBIT"
  }
}


// Snow index
function calcNDSI(sample) {
  ndsi = (sample.B03 - sample.B11)/ (0.01 + sample.B03 + sample.B11);
  return ((ndsi>0.2)&(sample.B03>0.15)) ? (ndsi) : 0.0
}

// Water index
function calcNDWI(sample) {
  ndwi = (sample.B03 - sample.B08)/ (0.01 + sample.B03 + sample.B08);
  return ((ndwi>0.2)&(sample.B03>0.15)) ? (ndwi) : 0.0
}
```

```
// Moisture Index
function calcNDMI(sample) {
  ndmi = (sample.B08 - sample.B11)/ (0.01 + sample.B08 + sample.B11);
  return ((ndmi>0.2)&(sample.B08>0.15)) ? (ndmi) : 0.0
}

function evaluatePixel(samples,scenes) {
  var avg1  = 0;
  var count1  = 0;
  var avg2  = 0;
  var count2 = 0;
  var avg3  = 0;
  var count3 = 0;
  var endMonth  = scenes[0].date.getMonth();

  for (var i=0;i<samples.length;i++) {
      var ndvi = calcNDSI(samples[i]);
      var ndwi = calcNDWI(samples[i]);
      var ndmi = calcNDMI(samples[i]);
      if (scenes[i].date.getMonth()==endMonth)
      {
avg3 = avg3 + ndvi + ndmi + ndwi;
        count3++;
      }
      else if (scenes[i].date.getMonth()==(endMonth-1))
      {
avg2 = avg2 + ndvi + ndmi + ndwi;
        count2++;
      }
      else
      {
avg1= avg1 + ndvi + ndmi + ndwi;
        count1++;
      }

  }
  avg1 = avg1/count1;
  avg2 = avg2/count2;
  avg3 = avg3/count3;

  return [avg1*5/3,avg2*5/3,avg3*5/3];

}
```

```
function preProcessScenes (collections) {
  collections.scenes.orbits = collections.scenes.orbits.filter(function (orbit) {
      var orbitDateFrom = new Date(orbit.dateFrom)
      return orbitDateFrom.getTime() >= (collections.to.getTime()-3*31*2*24*3600*1000);
  })
  return collections
}
```

# G Evalscript using 30 days NDMI

```
//VERSION=3 (auto-converted from 1)
/*
Author: Karl Chastko OG Joakim Aleksandersen
*/


function setup() {
  return {
    input: [{
      bands: [
          "B03",
          "B11",
  "B08"
      ]
    }],
    output: { bands: 3 },
    mosaicking: "ORBIT"
  }
}


// Snow index
function calcNDSI(sample) {
  ndsi = (sample.B03 - sample.B11)/ (0.01 + sample.B03 + sample.B11);
  return ((ndsi>0.2)&(sample.B03>0.15)) ? (ndsi) : 0.0
}

// Water index
function calcNDWI(sample) {
  ndwi = (sample.B03 - sample.B08)/ (0.01 + sample.B03 + sample.B08);
  return ((ndwi>0.2)&(sample.B03>0.15)) ? (ndwi) : 0.0
}

// Moisture Index
```

```
function calcNDMI(sample) {
  ndmi = (sample.B08 - sample.B11)/ (0.01 + sample.B08 + sample.B11);
  return ((ndmi>0.2)&(sample.B08>0.15)) ? (ndmi) : 0.0
}

function evaluatePixel(samples,scenes) {
  var avg1  = 0;
  var count1  = 0;
  var avg2  = 0;
  var count2 = 0;
  var avg3  = 0;
  var count3 = 0;
  var endMonth  = scenes[0].date.getMonth();

  for (var i=0;i<samples.length;i++) {
      var ndvi = calcNDSI(samples[i]);
      var ndwi = calcNDWI(samples[i]);
      var ndmi = calcNDMI(samples[i]);
      if (scenes[i].date.getMonth()==endMonth)
      {
avg3 = avg3 +  ndwi;
        count3++;
      }
      else if (scenes[i].date.getMonth()==(endMonth-1))
      {
avg2 = avg2 +  ndwi;
        count2++;
      }
      else
      {
avg1= avg1 +  ndwi;
        count1++;
      }

  }
  avg1 = avg1/count1;
  avg2 = avg2/count2;
  avg3 = avg3/count3;

  return [avg1*5,avg2*5,avg3*5];

}
function preProcessScenes (collections) {
  collections.scenes.orbits = collections.scenes.orbits.filter(function (orbit) {
```

```
      var orbitDateFrom = new Date(orbit.dateFrom)
      return orbitDateFrom.getTime() >= (collections.to.getTime()-3*31*2*24*3600*1000);
  })
  return collections
}
```

# H CSV with Labeled Data

| Date From | Standard Deviation | Ice Condition |
|---|---|---|
| 2023-09-12 | 17.09 | No ice |
| 2023-09-17 | 9.43 | No ice |
| 2023-09-24 | 8.26 | No ice |
| 2023-10-02 | 18.72 | No ice |
| 2023-10-04 | 5.82 | Ice |
| 2023-10-12 | 10.86 | No ice |
| 2023-10-17 | 8.66 | No ice |
| 2023-10-19 | 7.47 | No ice |
| 2023-11-18 | 11.41 | No ice |
| 2023-12-03 | 12.65 | No ice |
| 2023-12-23 | 2.58 | Ice |
| 2023-12-26 | 1.68 | Ice |
| 2024-01-05 | 1.32 | Ice |
| 2024-01-25 | 1.05 | Ice |
| 2024-02-01 | 1.26 | Ice |
| 2024-02-09 | 1.02 | Ice |
| 2024-02-21 | 0.75 | Ice |
| 2024-02-26 | 1.72 | Ice |
| 2024-03-07 | 20.65 | No ice |
| 2024-03-12 | 14.76 | No ice |
| 2024-04-01 | 14.9 | No ice |
| 2024-04-16 | 4.43 | Ice |
| 2024-04-21 | 22.89 | No ice |
| 2024-05-04 | 3.22 | Ice |

# I AI declaration

# Declaration of AI aids and -tools

Have any AI-based aids or tools been used in the creation of this report?

- ◯ No
- ◉ Yes

If *yes*: please specify the aid/tool and area of use below.

**Text**

☑ **Spell checking.** Are parts of the text checked by:
*Grammarly, Ginger, Grammarbot, LanguageTool, ProWritingAid, Sapling, Trinka.ai* or similar tools?

☑ **Text-generation.** Are parts of the text generated by:
*ChatGPT, GrammarlyGO, Copy.AI, WordAi, WriteSonic, Jasper, Simplified, Rytr* or similar tools?

☑ **Writing assistance.** Are one or more of the report's ideas or approach suggested by:
*ChatGPT, Google Bard, Bing chat, YouChat* or similar tools?

If *yes*, use of text aids/tools apply to this report - please specify usage here:

Spell checking: ChatGPT, Overleaf, and Google Translate.
Text generation: License agreement in README file (GitLab repository) was generated by ChatGPT.
Writing assistance:

- ChatGPT: rewriting and spell checking.
- Google Translate: translation and spell checking.

**Codes and algorithms**

☑ **Programming assistance.** Are parts of the codes/algorithms that i) appear directly in the report or ii) have beenused to produce results such as figures, tables or numerical values been generated by: *GitHub Copilot, CodeGPT, Google Codey/Studio Bot, Replit Ghostwriter, Amazon CodeWhisperer, GPT Engineer, ChatGPT, Google Bard* eller lignende verktøy?

If *yes*, use of programming assistance aid/tools apply to this report - please specify usage here:

ChatGPT: learning Python and Dart. Some code was generated and used as a foundation, but later modified and expanded by hand. Also used for some debugging and generation of test/example data.
Copilot: auto completion of single code lines and comments.

**Images and figures**

☐ **Image generation.** Are one or more of the reports images/figures generated by:
*Midjourney, Jasper, WriteSonic, Stability AI, Dall-E* or similar tools?

If *yes*, use of image generator aids/tools apply to this report – please specify usage here:

**Other AI aids or tools.** Have you used other types of AI aids or -tools in the creation of this report?If *yes*, please specify usage here:

☑ I am familiar with NTNU's regulations on artificial intelligence. *I declare that any use of AI aids or tools areexplicitly stated i) directly in the report or ii) in this declaration form.*

Hoa Ben The Nguyen, Joakim Aleksandersen, [illegible]

*Signatures*

16/05/24

*Date*

# J SWOT analyses tables

**Table 2:** SWOT analysis: DJI Metrice 300

| Strengths | Weakness | Opportunities | Threats |
|---|---|---|---|
| Operating temperature is -20 to 40 °C, high compared to most professional-grade drones that goes from -10 to 40 | Temperature under -20 °C will degrade the performance of the drone. Might have to let the drones rest multiple times under scans | Additional hardware can be included, that includes a battery heating system or other hardware allowing it to operate in colder weather | Its operating temperature is still low compared to winter 2023/2024 in Norway that went past -20 °C, drones under -20 °C needs to be specialized and is not commercially used. |
| High payload, capable of 0.93kg. Smaller than DJI Matrice 100, but not enough to make a big difference | Privacy concerns when flying with a camera over or past homes, turn the camera off and avoid privacy policies | Having camera can aid in search and rescue | |
| Sufficient range, 55 minutes flight time. If it travels at max speed with GPS (17m/s), can it travel as far as 76km | | Additional battery packages for longer flight, can be mounted without affecting payload too much. Making up for lost performance in cold weather | Designed to be highly Water resistant not water proof, should not be used when raining or snowing |

| | | | |
|---|---|---|---|
| Transmission range 8 km, more than enough to cover the shoreline of the city | | Base position of the transmission is not limited to the center of Gjøvik Sentrum's shoreline, but is advised to be sure | |
| Relative cheaper compared to other commercial drones with the same operating temperature | High cost drone, but you get what you pay for. approximately 4 times more expensive than DJI Matrice 100 | Provide their own mobile SDK for greater control or customized designers for software | |
| Provide their own external third party software and hardware | | compatible with most third party hardware and software, to be added to a drone | |
| Provide their own software development kit for full control of designing a it's behavior | Software produced by user is time consuming to make from scratch | Arduino are compatible and has it own mount to the drone | Software produced by user has a potential for bug that results in crashing |

NTNU

Norwegian University of
Science and Technology