NTNU
Norwegian University of
Science and Technology

A Big Data Approach to Generate Training Data for Automatic Ship Detection

An Integration of AIS and Sentinel-2 MSI

Lars Henrik Berg-Jensen
Adrian Tofting

Master of Science in Engineering and ICT
Submission date: June 2018
Supervisor: Hossein Nahavandchi, IBM

Norwegian University of Science and Technology
Department of Civil and Environmental Engineering

Abstract

A fast and scalable approach to combine global satellite images with ship navigational messages

In this project, we have developed a data processing pipeline that combines multiple sources of data to automate the process of ship detection in satellite images. The central concept is based on integrating *Sentinel-2 Optical Multi-Spectral Imagery* with ship navigational messages provided by the *Automatic Identification System* (AIS). The successful integration made it possible to automatically generate a training dataset consisting of images labelled with ship positions. A future goal is for this training dataset to be used in supervised machine learning to train a neural network to recognise ship features in the images automatically. Our vision is for this to accompany AIS in applications ensuring safety in the marine sector.

Our data processing pipeline includes all aspects of data analytics: collection, preprocessing, cleansing, storing, filtering, combining, analysing, as well as visualising the data. We have designed the system to be modular and highly scalable, such that it can further be developed into supporting real-time analysis of any aerial imagery.

The solution we have developed can be divided into three main parts:

Image Selection Optimisation is a proposed approach to select satellite images globally, with a high probability of containing ships. This is done by processing global AIS data within an arbitrary time interval. When performing a density analysis on 400 million global ship navigational messages, we experienced a total execution time below 3 minutes. The optimisation enabled by such analyses will further save an immense amount of time in the generation of the training dataset.

Ship Position Estimation computes the coordinates for every ship within the satellite image. This includes complex data integrations, both spatially and temporally. It relies on having access to a complete dataset from AIS around the time the image was sensed. It also performs significant corrections, as the provided timestamp was found to be highly inaccurate. The execution time for an arbitrary image is below two seconds.

Training Dataset Generation extracts regions from the image around each estimated ship position, resulting in smaller training images. This will further be advanced into augmenting the training dataset by varying the region extraction using appropriate transformations.

Sammendrag

En effektiv og skalerbar metode for å kombinere globale satellittbilder med navigasjonsmeldinger fra skip

I dette prosjektet har vi utviklet en metode for å kombinere flere datakilder, for å automatisere skipsdeteksjon i satellittbilder. Hovedkonseptet er basert på å integrere Optiske Multispektrale bilder fra satellittene Sentinel-2, med navigasjonsmeldinger fra Automatisk Identifikasjon System (AIS). Dette gjorde det mulig å automatisere genereringen av treningsdata bestående av bilder markert med skipsposisjoner. Et fremtidsmål er å benytte dette treningsdatasettet til å trene et nevralt nettverk til å automatisk gjenkjenne skip i bildene. Vår visjon er at dette skal kunne støtte AIS i sikring av den marine sektoren.

Vår metode for å prosessere dataen inkluderer alle aspekter av dataanalyse: innhenting, preprossesering, vasking, lagring, filtrering, kombinering, analysering, i tillegg til visualisering. Vi har designet systemet for å være modulært og skalerbart, slik at det kan bli videreutviklet til å benyttes i sanntidsanalyse av satellitt- og flyfoto.

Løsningen vi har laget er naturlig å dele i tre hoveddeler:

Optimering av Bildevalg er en foreslått metode for å velge satellittbilder globalt med høy sannsynlighet inneholder skip. Dette gjør vi ved å analysere globale AIS-data innenfor ønsket tidsintervall. Da vi kjørte en tetthetsanalyse på 400 millioner globale skipsmeldinger opplevde vi en total kjøretid på under 3 minutter. Optimaliseringen som ble muliggjort av resultatene fra silke analyser vil videre spare enorme mengder tid under genereringen av treningsdatasettet.

Estimering av Skipsposisjoner gjøres for alle skip innenfor et ønsket satellittbilde. Dette inkluderer kompleks dataintegrasjon, både romlig og temporalt. Løsningen baserer seg på tilgangen til et komplett dataset fra AIS rundt tidspunktet bildet ble tatt. Den utfører også store korreksjoner for bildets tidsstempel, da det medfølgende tidsstempelet er høyst unøyaktig. Kjøretiden for et vilkårlig bilde er under to sekunder.

Generering av Treningdatasett henter ut utsnitt fra bildet rundt hvert skips estimerte posisjon, og resulterer i mindre treningsbilder. Denne skal videre bli utviklet til å kunne hente ut forskjellige utsnitt ved å benytte passende transformasjoner for å augmentere datasettet.

Preface

The study presented represents our master project completing the civil engineering degree in *Geomatics*, spring of 2018. The project is mandatory and constitutes 30 credits in the tenth semester of the study programme *Engineering & ICT* at the *Norwegian University of Science and Technology*. The project was carried out as a tight collaboration between the two students. The topic for the project came as a result of the students' interest in both space technology and machine learning. The internal supervisor is Hossein Nahavandchi.

The project is a highly interdisciplinary study, combining methods from Data Sciences, Geomatics, Machine Learning and Remote Sensing. As the span is both broad and in-depth, not every base concept is explained in detail. It is recommended that the reader have a basic understanding in most of these fields. To fully appreciate the sections covering data management, it is further assumed some experience with managing spatiotemporal data.

We would like to express our appreciation to the people supporting the idea, motivating us to keep thinking big and to keep looking up. Norwegian Space Centre, Norwegian Coastal Administration, Science & Technology Corporation S[&]T, Kjell Eikland of Eikland Energy, André Lima from the University of Maryland, and our supervisor Hossein Nahavandchi.

We also want to thank the open-source community for making this project at all possible. The availability of high performance, well-documented and free to use programming tools is the most critical enabler for innovation. We hope to get the opportunity to give back in the years to come.

Lastly, we want to give each other a pat on the back. The teamwork has been impeccable, and we are pleased with what we have learned and accomplished, being a small team on a big blue planet. By this, we want to round off this project, as well as our five years of studying, by providing our favourite quote. We think it describes well what is illustrated by the cover photo of this thesis, but also embodies the magic that can come from great teamwork.

“ *It's not the size of the ship, it's the motion of the ocean* ”

Trondheim, 2018-06-11

[illegible]

Lars Henrik Berg-Jensen

Trondheim, 2018-06-11

[illegible]

Adrian Tofting

Contents

List of Figures

List of Tables

List of Algorithms

Code Listings

Acronyms

AIS Automatic Identification System

API Application Programming Interface

AWS Amazon Web Services

BOA Bottom-of-Atmosphere

CNN Convolutional Neural Network

CTE Common Table Expressions

DGPS Differential Global Positioning System

ECEF Earth Centred, Earth Fixed

ESA European Space Agency

FTP File Transfer Protocol

GIS Geographical Information System

GML Geography Markup Language

GNSS Global Navigation Satellite System

GSD Ground Sampling Distance

IOU Intersection Over Union

JSON JavaScript Object Notation

KML Keyhole Markup Language

MGRS Military Grid Reference System

MMSI Maritime Mobile Service Identity

MSI Multi-Spectral Instrument

NMEA National Marine Electronics Association

ORM Object Relation Mapping

RGB Red-Green-Blue

SOTDMA Self-Organized Time Division Multiple Access

SQL Structured Query Language

SRID Spatial Reference System Identifier

SSD Solid-State Drive

SWIR Short-Wave-Infrared

TCI True Colour Imagery

TOA Top-of-Atmosphere

UTC Coordinated Universal Time

UTM Universal Transverse Mercator

VNIR Visible and Near-Infrared

WGS84 World Geodetic System 1984

XML Extensible Markup Language

Chapter 1

Introduction

In this first chapter, we will introduce the problem and the motivation behind the project. The primary objectives will be presented, as well as elaborative objectives acting as their foundation. Further, we will illustrate our main approach for the technical solution, before outlining the succeeding chapters.

1.1 Background

During our studies at the university, we were introduced to the Earth Observation Programme Copernicus, coordinated and managed by the European Commission. Since then we searched for interesting problems to solve for our master thesis using this free an openly accessible data. Also, we developed an interest in computer vision and artificial intelligence. As a result, we came up with the idea of training a neural network to detect and monitor marine traffic in satellite images.

The Norwegian Space Centre and The Norwegian Coastal Administration among other agencies use satellite data for an extended range of purposes in the marine sector. Some examples are fisheries surveillance, environmental crime like oil pollution, traffic surveillance in the Arctic, helping anti-pirate operations off the coast of Africa, and monitoring ship activity in waters of interest to Norway.

Another use case of the satellite data is detecting and tracking refugees risking their way overseas without the necessary preparations and equipment. The ability to monitor these boats vastly increases the security of human lives on the oceans and makes a huge difference in search and rescue missions.

Today most of these applications are done based on the Automatic Ship Identification system (AIS), which offers global real-time ship tracking information. As regulations impose ship owners the use of AIS, the dataset is invaluable for these applications. Not all categories of ships fall under these regulations. Another shortcoming of basing analyses solely on AIS is that ship owners who are not interested in being monitored, for whatever reason, have the possibility to turn off the AIS transceivers. They are then impossible to track, using AIS only. This is highly relevant in the current political climate, as the recent news shows us. Multiple articles published December 2017, revealed Russian oil tankers that turned off their AIS transceivers to slip under the radar while delivering oil to North Korea.

To reduce this loophole, we propose an approach combining AIS with another dataset; Optical Multi-Spectral Imagery captured by the Sentinel-2 satellites, as part of ESA's Copernicus programme. As with AIS, the images have global coverage, but is naturally not dependent on the user equipment. To get extract useful information from the images, they need to be labelled or segmented with categories for what they contain. Extra motivation for using images having multiple spectral bands is that they naturally contain more information than other remote sensing images. We suspect that the 13 spectrally different bands act like "fingerprints" that allow for the recognition and identification of vessels.

To fully exploit the enormous amount of data generated by the Sentinel satellites, the image processing should be generalised and automated. The current state-of-the-art technologies are based on training Deep Neural Networks to recognise features from example data labelled with the ground truth. We have with this project created an approach for automating the generation of this training dataset. The approach facilitates the machine learning by offering various examples, thereby enable generalised learning.

Our solution, as-is, also offers an extra dimension to monitoring based on AIS. With seamless integration between the AIS data and image regions of interest, the monitoring can now be controlled from an independent source. The next step will naturally be training a neural network on the data generated by our system. This will further augment the monitoring by recognising ship outside the scope of AIS.

1.2 Objectives

First, we introduce the *main objectives* that we wanted to be present throughout every step of the project. These three objectives are what we refer to in the concluding chapter of our thesis.

1. Develop a method to generate a training dataset for ship detection automatically.
2. Investigate how our training dataset can achieve qualities that positively affect the results of supervised machine learning.
3. Optimise this method to be highly scalable and facilitate further development into a real-time data processing solution.

Objectives that should be accomplished for us to have the required basis for the further research and solution development.

4. Achieve in-depth knowledge of the relevant aspects of the two data sources.
5. Decide what tools that are best fitted for management and analysis of Big spatiotemporal Data.
6. Analyse our prior research results to identify suspected challenges concerning the integration of the two datasets, as well as the data management.

Objectives for our research and solutions development that should be met to ensure the chosen approaches leads to the accomplishment of the main goals.

7. Develop a database architecture that is highly scalable.
8. Construct SQL-queries that are executing fast and scalable.
9. Achieve a better estimate than what is provided by ESA, for the image timestamp.
10. Estimate ship positions within the image at the image timestamp.
11. Explore how to optimise the selection of relevant images.
12. Visualise the global AIS data to support the analysis.
13. Identify how conditions can affect the accuracy of the image labelling.
14. Investigate how to avoid inaccurate labelling of the images.
15. Develop an efficient approach to extract smaller extractions from the images.
16. Discuss different approaches for labelling the images, as well as how to augment the resulting dataset further.

1.3 Approach

To explain the central concept, we illustrate our approach with some technical simplifications. The full workflow is presented and further elaborated in Section A.2.

Sentinel-2 images are downloaded from the Copernicus Hub [1] using a developed Python framework based on a remote API. AIS data is downloaded from an external client and inserted efficiently into our local spatial database by using a developed script.

The approach for integrating the two datasets, resulting in the labelled images, is illustrated in Figure 1.1. Sentinel-2 images within timestamps of interest are downloaded. The timestamp and coordinates of the image are used to query AIS data from the spatial database, which outputs relevant ship coordinates. The ship coordinates are then combined with the image to create the labelled training image.

To ensure that the resulting dataset includes various examples of ship features we perform Big Data Analyses on the AIS data in order to target specific ship features, such as size, heading and shape of the trailing wake which is varying with the velocity of the ship.

We also focus on designing the system such that it is fast and highly scalable. This is done by in-depth analysis of the underlying database execution plans to optimise the queries. The database architecture is also customised to the nature of the data, to support the scalability to the fullest.

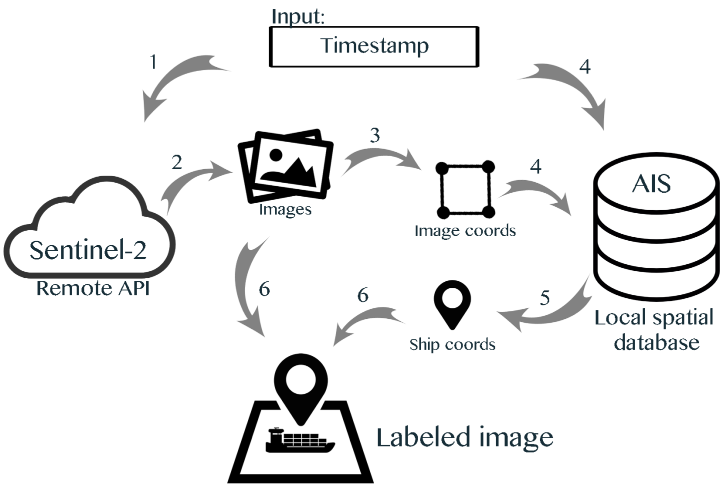


Figure 1.1: An illustration of our approach used to generate the training dataset.

1.4 Outline

The remaining parts of this thesis are organised into the following chapters: *Theoretical Background, Basis and New Discoveries, Analysis and Technical Solutions* and *Conclusions*. In the end, we have the *Appendices*

In **Chapter 2**, we present the theoretical knowledge this project is based upon. Relevant theory revolving supervised machine learning is presented, with the primary focus on what is essential qualities of a training dataset. This is followed by a thorough introduction to AIS and the Sentinel-2 satellites and products.

In **Chapter 3**, we first present the prerequisites for the datasets that either should already be met or that we should be able to develop methods to meet. This is followed by in-depth research of the two datasets and how they can be integrated to create a labelled training dataset. The last section describes some surprising and critical challenges that were uncovered during our research, and that turned out to be significant threats for the success of the project.

In **Chapter 4**, we present our in-depth analysis and the technical solutions engineered during this project. First, the solutions that had to be implemented to compensate for the critical challenges uncovered in Chapter 3 are presented. This includes the estimation of Sentinel-2 image timestamps and solutions to the Big Data management and scaling. Following this, a procedure of optimising Sentinel-2 image selection, without the need of downloading the actual images, is presented. Then we present and discuss methods and choices for the generation of the training dataset, which includes the technical solutions for estimating ship positions within images and the extraction of image regions of interest.

In **Appendix A** we present the software tools used in the project as well as the whole project workflow. This includes the data flow by following the raw data as it is fed into the system until it is outputted as image region extractions labelled with ship positions.

In **Appendix B** we present the *SQL* code used in the algorithms implemented for this project.

Chapter 2

Theoretical Background

This chapter covers the theoretical background of the project, which is the basis of the subsequent analyses. The reader will get a basic understanding of what is required by a training dataset for supervised machine learning. Following this, the reader will get a more in-depth theory about the two datasets used in this study to create the training dataset.

2.1 Training Dataset for Supervised Machine Learning

Object detection using supervised machine learning has shown very promising results the recent years. The progress and innovation involving deep learning, and especially convolutional neural networks (CNN), have been outstanding, and have made CNN state-of-the-art in computer vision problems. Figure 2.1 illustrates a basics workflow of a CNN. In the field of remote sensing, deep learning has only recently started to be applied for analysing satellite imagery. Explaining neural networks in detail is outside the scope of this study, but we will cover it briefly as it is important for understanding the qualities that make up a suitable training dataset. We have also chosen to leave out the use cases for unsupervised learning related to our problem.

An artificial neural network consists of layers of connected nodes. Each layer of nodes gets input values through the connections from the preceding layer. By applying functions on the biased, weighted sum of the input, it calculates output values that it feeds forward to the next layer. The output of the last layer in the network is compared to the ground truth provided by the training dataset by using a cost, or error function, which calculates “how wrong” the network predicts. The goal is to adjust the weights and biases throughout the network to minimise this error, as an error of zero means that the output coincides perfectly with the ground truth.

The process of learning consists of feeding training data through the network, to gradually update these weights and biases. If the training dataset is of a certain quality and volume, the network will take steps towards being able to predict new, unseen data, and thus learn to generalise. In the following two paragraphs, the qualities needed to obtain this are explained in further detail.

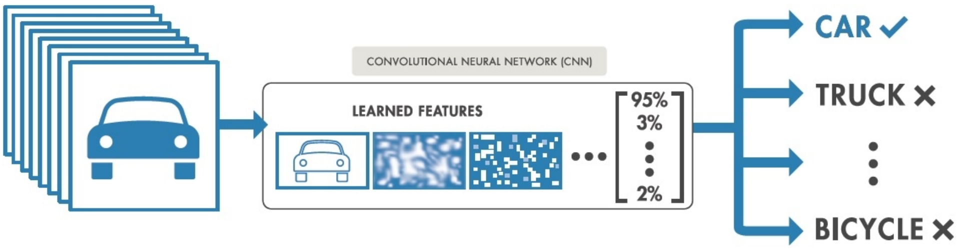


Figure 2.1: Simple illustration of a typical Convolutional Neural Network workflow [2].

Image Size
There are many attributes of importance in the making of a good quality training dataset for object detection in images. One of them is the physical size or the number of pixels in the images. As the network will have to process the whole input image for features, it is more effective to train with small images than large images. So, for instance, if one wants to train on high-resolution imagery, and the features of interest only cover a small portion of the image, it can be beneficial to crop out the parts containing these features and use this as the training data. Then it is important that the whole feature is included in the cropped picture. Moreover, if the cropped high-resolution image still is large, one might have to scale down the resolution to get an effective training rate. These questions depend on the available hardware, CPU and GPU, in addition to the network architecture one chooses. The images that are passed through the same network also need to have the same size and resolution, so one might have to scale them up or down. For instance, for multi-spectral imagery, the spectral bands might have different resolutions.

Pixel Depth
The example of multi-spectral imagery introduces another element, which is the pixel depth of the image. Standard RGB images have a depth of three, as they consist of

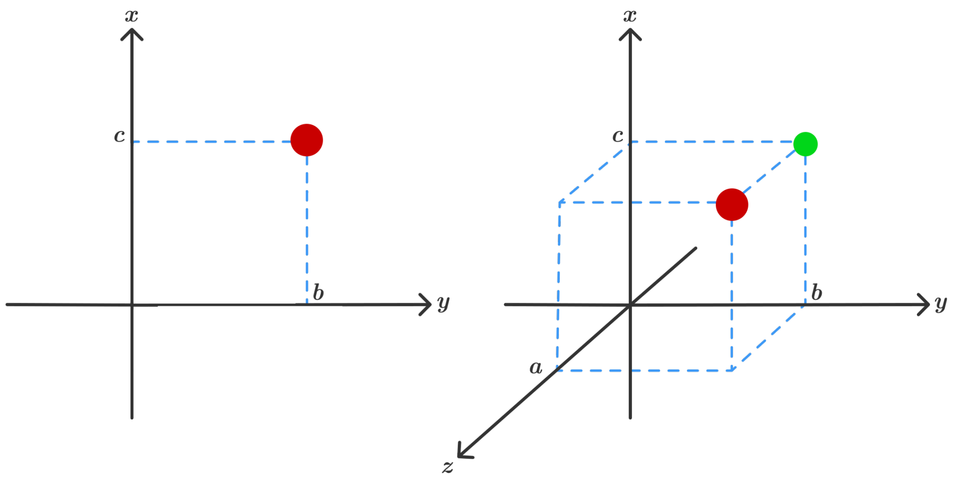


Figure 2.2: Example of the power of higher dimension features. The two points are indistinguishable in two dimensions. Adding the third dimension (i.e. one extra attribute) makes it possible to separate them.

three spectral bands: red, green and blue. Every pixel in the images has a pixel value for each band. Multi-spectral (and hyperspectral) images consist of even more spectral bands spread across the electromagnetic spectrum. Hence, the images have *many* more dimensions, as there for every pixel exists many more values. The great advantage of this in object detection is that more dimensions and values can separate features and pixels from each other. Depending on the spectral bands and the reflectance from the scene, features that are impossible to distinguish using the combination of certain bands might be stand out by the inclusion of another. We have illustrated this in Figure 2.2.

2.1.1 Accuracy

Arguably, the most important is the parameter that tells the network what the features of the data are. This is the label holding the ground truth and is what distinguishes a training data from an ordinary dataset. The ground truth is used by the network to understand what kind of abstractions and features to be extracted from the data. Hence, it is critical that the vast majority of the labels are correct and precise, for the network to be able to achieve high accuracy in predicting.

Precise labelling
In machine learning, one can divide labels into for groups: true positives, true negatives, false positives and false negatives. For object detection, true positives happen when the image is correctly labelled with the existence of the object. True negatives have correctly labelled the absence of the object. These are the kind of labels one tries to achieve. The network needs both true positives and true negatives. The difference between the images labelled as positives and negatives make the network able to distinguish the features of the object in question from the rest of the image. In addition, true negatives are important for testing the classification performance of the network.

On the other hand, if an image does not contain the object, but is still labelled with it, it is considered false positive. Similarly, if the image actually does contain the object, but the label says the opposite, it is a false negative. False positives and false negatives should be avoided in the training data, as it will disturb the learning process and decrease the accuracy of predictions. The amount of these unwanted labels varies with the type of data, and the labelling process used to generate it. In remote sensing, for instance, false positives can be a big problem due to clouds obscuring objects on the ground.

Volume
Fortunately, incorrect labels can be compensated by having a great volume of training data. In a large training dataset, individual labels have less influence on the whole outcome of the learning, and thus the network will be more robust to individual errors. When training a neural network, one is tuning its parameters such that it can map a particular input, for instance, an image, to some output that match the label. To tune all the parameters with good performance, a sufficiently large amount of training examples have to be fed into the network. A typical state-of-the-art neural network has parameters to tune in the order of millions, such that it is natural that enormous volumes of training data are required.

A large training dataset is also important as it is normal to split the training data into three parts:

- A training set used to train the network.
- A validation set used in between training to evaluate the performance of the training so far.
- A test set used to evaluate the final learning outcome of the training.

The data in the three sets are disjunct, such that there is no overlap. This means that the validation and test score are numbers of how well the network has learned to generalise or classify new data that it has never seen before.

2.1.2 Generalisation

As the target of training a neural network generally is to make it able to generalise to new data and make accurate predictions, some of the techniques widely used to obtain this are presented here. A few of them are so fundamental that they are already mentioned in the previous section, such as the importance of having a dataset of great size. The minimum required size will vary with the complexity of the problem but could be in the tenths, or hundreds of thousands.

Variety
In addition to having enough data, it is essential that the data is diverse, as the model one train only can capture what it has seen. For example, if the network is supposed to recognise dogs in images, there should be images of different breeds, in different postures and in different locations of the image. Moreover, there should be images where the dog partially covered in varying degree, such that the network can be able to classify the object as a dog, even if only parts of it is visible.

Augmentation
However, the amount of variation needed in the data is problem-specific. A technique for adding further variation and robustness in the data, as well as increasing the size of the dataset, is *data augmentation*. Data augmentation is applying transformations such as cropping, scaling, flipping, translation and rotation on the existing training dataset. The neural network will think of these as different examples of features. We have illustrated this in Figure 2.3 by augmenting a satellite image of a ship.

Not all techniques apply to every problem. An example could be the classification of road signs in images from a camera placed on a car. The classification for this example should be scale-invariant, as the program should be able to detect and classify signs from far away as well as close. Hence, the training data should contain signs in both large and small scales.

The images could also be translated or horizontally flipped, such that the object of interest is found in different parts of the images. This teaches the network to look for features everywhere in the image. However, in the case of road signs, it would not make

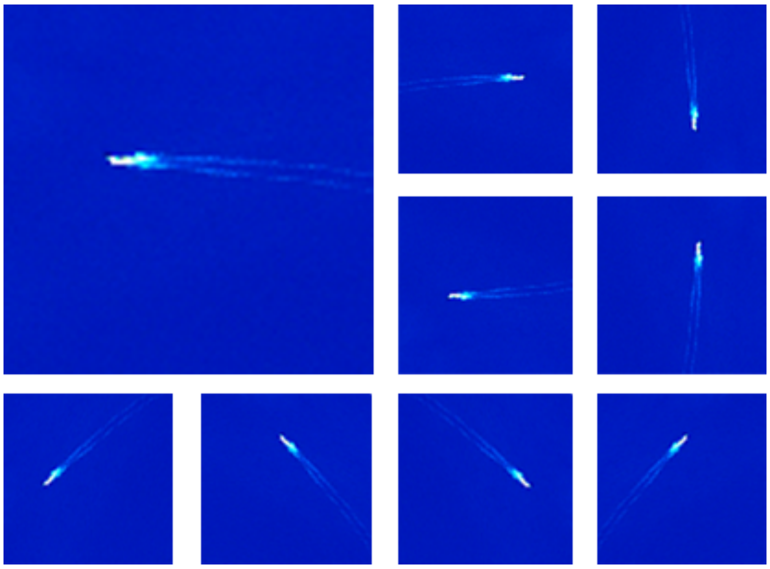


Figure 2.3: En example of how to perform data augmentation of a training data set. With the original image as a basis one can artificially create “more images” by applying transformations to it.

sense to add rotation. Upside-down signs might not make sense, or worse, they could be confused with signs of other meanings. In remote sensing applications, this is not the case, and adding rotation can be beneficial. For instance in ship detection, it is useful to have images of ships with different heading angles, producing different trailing wakes.

Another example to emphasise the importance of augmentation is if one has a dataset consisting of two brands of cars, and one wants a neural network that classify the brand. Let us assume all the cars of brand A are always facing the right direction in the images, and all cars of brand B are facing left. After training, the network has 95 % accuracy on the dataset, but when one tests it with an image of a brand A car facing left, the network outputs that it is a brand B car. This is because the network learns what feature that distinguishes one class from another most, and in this case, this feature was the direction. By adding horizontal flipping on the images, one would remove this irrelevant pattern, and the network would search for other features to distinguish the two brands.

Most datasets containing images are subject of unintended noise of different types and in varying degree. In remote sensing for instance, where the images are taken from cameras orbiting the Earth, there is a chance of atmospheric and ionospheric noise. The consequences of this can be, as with the false positives, confusing the training. One

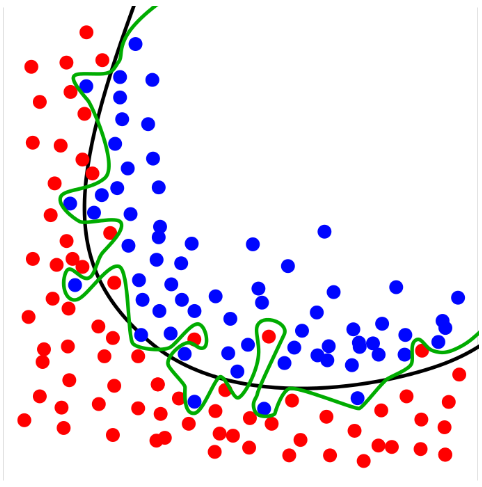


Figure 2.4: The green curve represents an overfitted model, while the black represent an optimal one. Even though the green line distinguishes the training data here the best, it is too dependent on this particular data, and would most likely perform worse than the black line when following new unseen data [3].

technique to reduce this is purposely applying noise to the images so that the network can learn to be invariant to it.

Overfitting
The phenomenon many of these techniques try to avoid, and which is analogous to bad generalisation, is called overfitting. This happens when the network outputs great accuracy on the training set but fails to generalise on the features, such that it achieves lower accuracy on new, unseen data. One can think of it as the network having memorised the training data, instead of having learned from it. Usually, it is a consequence of having a too small training set, as well as overtraining. One usually try to prevent this by checking the validation set performance in-between training and stopping the training when the validation accuracy peaks. Further training after this will lead to overtraining, and thus overfitting. The concept of overfitting can be observed in Figure 2.4.

2.2 Automatic Identification System (AIS)

The Automatic Identification System (AIS) was introduced by the UN's international maritime organisation (IMO) to increase maritime security, and for enhancing monitoring and regulation of maritime traffic.

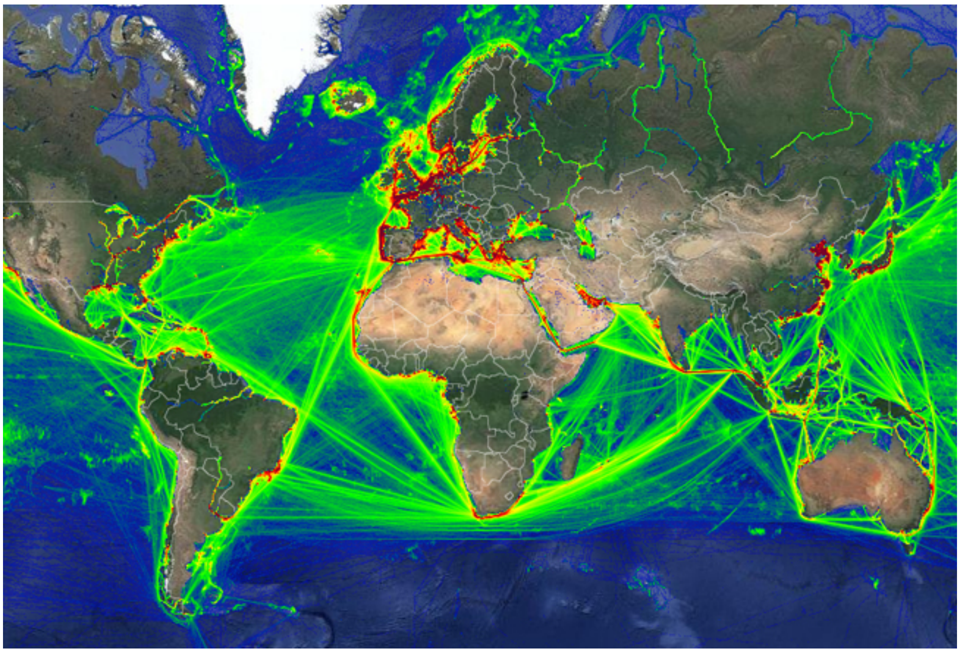


Figure 2.5: Global density map of all vessels tracked with AIS [5].

The regulation requires an AIS transceiver to be fitted aboard all ships of 300 gross tonnages and upwards engaged on international voyages, cargo ships of 500 gross tonnages and upwards not engaged on international voyages and all passenger ships irrespective of size [4]. The onboard AIS shall, along with the identity of the ship, provide information about position, course, speed, and navigational status among other safety-related information. This information is automatically transmitted to receivers on shore stations, satellites, and other ships and aircraft. Figure 2.5 shows a density map of all vessels tracked by AIS for one year (year not known).

2.2.1 The Development of AIS

The development of AIS started in the 1990s as a short-range identification and tracking network and has since then evolved towards offering global real-time coverage.

The AIS transceivers are limited to the VHF range due to the curvature of the earth. Depending on the installation elevation of the coastal receiver stations, ships could be detected only up to 74 kilometres from shore. This is sufficient for coastal tracking, or ship to ship monitoring, but means that the traffic on the open waters is a big blind spot. It was thus of high interest for various entities to experiment with detecting AIS

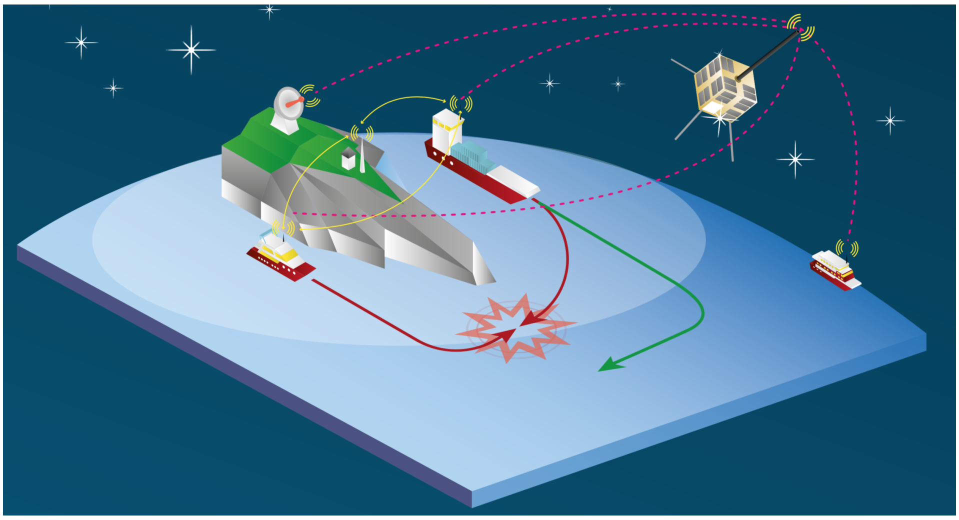


Figure 2.6: llustration of the AIS data flow. Shore based receivers and AIS-satellites are working seamlessly. Here two ships use the messages for collision avoidance [6].

transmissions from space, using receivers mounted on satellites. Figure 2.6 illustrates this principle. In November 2009 the NASA Space Shuttle mission STS-129 to the International Space Station (ISS) attached an AIS antenna to the Columbus module of the ISS [7]. This was the first step towards a satellite-based AIS-monitoring service. Since then many satellites from different nations have been launched into orbit with the purpose of improving surveillance of the global maritime traffic, in which four of them are Norwegian-made. The first Norwegian AIS-satellite is depicted in Figure 2.7.

According to one of the most comprehensive AIS databases, MarineTraffic, there are now more than 650,000 vessels with an operating AIS transceiver [9], and the number is growing rapidly. The data is transmitted via a tracking system using Self Organized Time Division Multiple Access (SOTDMA) datalink [10]. It creates 4500 available time slots each minute for separate messages to be transmitted. This seems like quite a lot and is a good architecture for the terrestrial stations, which have a restricted amount of distinct ships to handle at the same time. The satellites equipped with an AIS transceiver, on the other hand, are relatively few compared to their large reception footprint. Different companies such as exactEarth are developing new technologies that are better suited for large amounts of data per transceiver [11]. Until a new standard has been developed, the satellite segment of AIS will augment rather than replace the terrestrial segment.

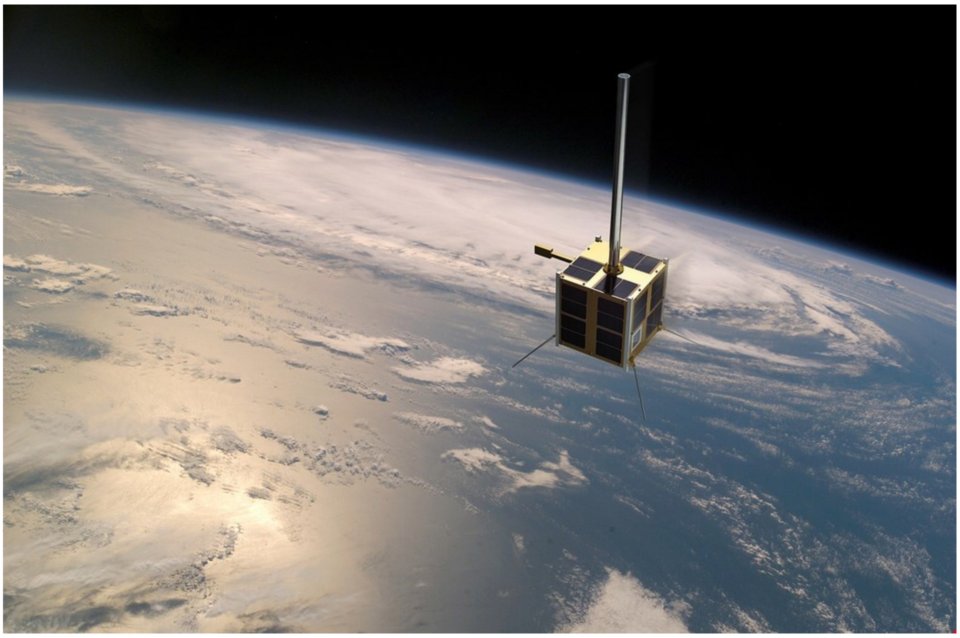


Figure 2.7: Norwegian made satellite AISSat-1 [8].

2.2.2 Use Cases

AIS is continuously evolving, and new applications are still appearing. The current use cases for AIS include, but are not limited to:

- **Collision avoidance**
- **Fishing fleet monitoring and control**
- **Vessel traffic services**
- **Maritime security**
- **Aids to navigation**
- **Search and rescue**
- **Accident investigation**
- **Ocean currents estimates**
- **Infrastructure protection**
- **Fleet and cargo tracking**

2.2.3 How AIS Works

Sensors on board the ship, typically a GNSS receiver and a gyrocompass, is the source of the time-dependent positional information transmitted by AIS. Other, more static information is preprogrammed upon installation. All data is automatically broadcast regularly via a VHF transmitter. The data is standardised for two different transmitter classes, A and B. They are further divided into 27 different types of messages targeted at different use cases. The messages of main interest to the present project, transmitted by marine vessels, are described below.

Class A Transceivers
This class is targeted at large commercial vessels, as it requires more from the equipment installed. The default transmit rate is every few seconds.

Class B Transceivers
This class is targeted at lighter commercial and leisure vessels and is a cheaper alternative to Class A. The default transmit rate is every 30 seconds.

Position Report
For Class A the position report messages are numbered 1, 2 and 3 [12]. For Class B they are numbered 18 and 19 [13]. These are broadcast every 2 to 10 seconds while underway, and every 3 minutes while at anchor. The main data transmitted include the following:

- **MMSI:** The vessel's Maritime Mobile Service Identity
- **Navigation status:** the vessels current status, e.g. "at anchor"
- **ROT:** Rate of turn, right or left, from 0 to 708 degrees per minute
- **SOG:** Speed over ground, from 0 to 102 knots (189 km/h)
- **Positional accuracy:**
  - **Longitude:** to 0.0001 minutes
  - **Latitude:** to 0.0001 minutes
- **COG:** Course over ground, relative to true north
- **True heading:** 0 to 359 degrees (for example from a gyro compass)
- **Time stamp:** The seconds field of the UTC time when these data were generated.

Static Data Report
For Class A the message has number 5, and number 24 for Class B. The static data report is broadcast every 6 minutes.

- **IMO:** International Maritime Organization number
- **Radio call sign:** international radio call sign
- **Name:** the name of the vessel
- **Type:** type of ship/cargo
- **Dimensions:** dimensions of ship to nearest meter
- **Destination:** port name
- **ETA:** estimated time of arrival at destination

2.3 Sentinel-2

The Sentinel-2 mission of ESA's Copernicus programme consists of two optical satellites designed to deliver multi-spectral satellite imagery for remote sensing. The data it produces is free and open to the public, and often available within hours of production. ESA launched the first Sentinel-2A satellite on June 23, 2015, and the second Sentinel-2B was launched March 7, 2017.

2.3.1 Orbit and Geographical Coverage

Both satellites are operating in the same sun-synchronous orbit, phased 180 degrees apart. Being sun-synchronous, the sunlight angle of any given point of the Earth's surface is consistent over time, which is important for sunlight dependent sensors in remote sensing. The orbital inclination is 98.62% and they orbit under a mean altitude of 786 km. They chose the Mean Local Solar Time (MLST) at the descending node to be 10:30, a value that gives a suitable level of solar illumination while also minimising potential cloud cover to some degree [14].

They both orbit with a 10-days repeat cycle, which in total gives a temporal resolution of five days at the equator. Note that the orbit being near-polar, the revisit frequency is higher with increased latitude. Hence, for the most of Europe, it could be as often as two or three days. During the cycle along a band of latitude extending from 56° South to 83° North, the satellites will acquire the following data over land and coastal areas:

- islands greater than 100 km^2 in area
- islands in the European Union
- all other islands within 20 km of a coastline
- the Mediterranean Sea
- all inland water bodies
- all closed seas

2.3.2 Optical Multi-Spectral Instrument

Both satellites carry a Multi-Spectral Instrument (MSI) that produces imagery in 13 spectral bands ranging from Visible and Near-Infrared (VNIR) wavelengths to Short-Wave Infrared (SWIR). To capture images, the MSI uses a *push-broom* concept where the camera sensor collects rows of image data across the orbital swath. As the satellites move forward along the path of the orbit, new rows are acquired. This continuous acquisition of an image is called a *datatake*. The different wavelengths of light are collected by a three-mirror telescope and focused, via a beam-splitter, to two focal planes: one for the VNIR wavelengths and one for the SWIR wavelengths [14].

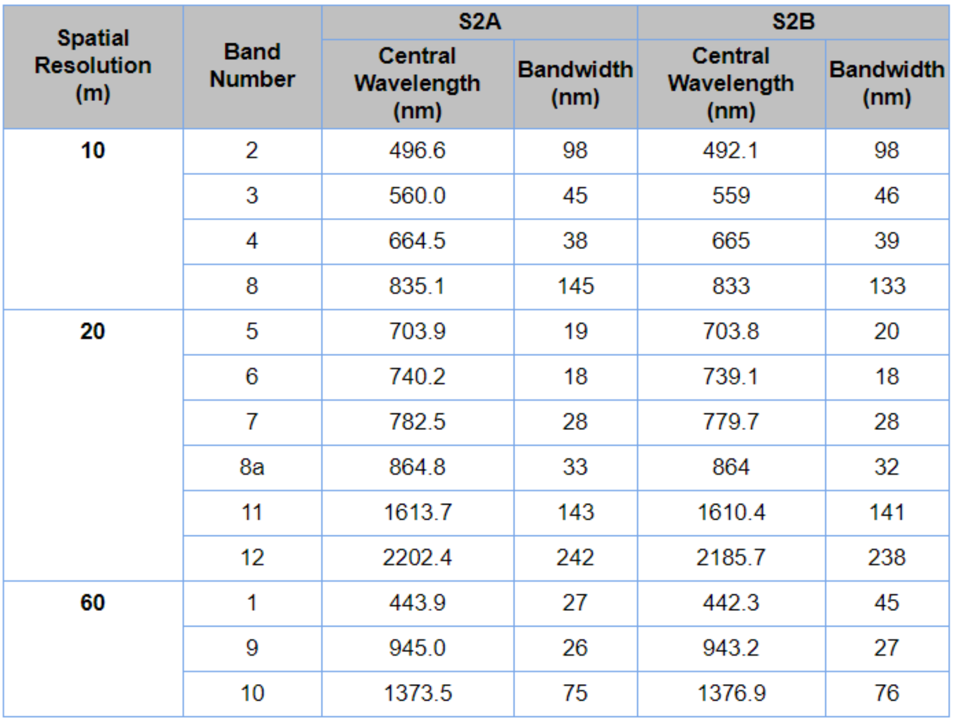
The spectral bands have different spatial resolution of 10 m, 20 m and 60 m, depending on the particular band, as illustrated in Figure 2.8 and 2.9.

2.3.3 Data Products

The data produced by the satellites is free and open to all users. There are various ways to download data. ESA has a user client where one can download data through a graphical user interface, SciHub. However, they also have an API which one can use for scripting and programming [1]. Currently (June 2018), they offer two different data products to users: the Level-1C product and the Level-2A product.

Level-1C

The Level-1C product is generated by the ground segment and is the result of having processed the Sentinel-2 raw data several times. The first two processing levels are Level-0 and Level-1A, which are compressed and uncompressed raw data, respectively. Level-1B data is radiometrically corrected radiance data. The Level-1B processing also does a refinement of the physical geometric model, using available ground control



Spatial Resolution (m)	Band Number	S2A		S2B	
		Central Wavelength (nm)	Bandwidth (nm)	Central Wavelength (nm)	Bandwidth (nm)
10	2	496.6	98	492.1	98
	3	560.0	45	559	46
	4	664.5	38	665	39
	8	835.1	145	833	133
20	5	703.9	19	703.8	20
	6	740.2	18	739.1	18
	7	782.5	28	779.7	28
	8a	864.8	33	864	32
	11	1613.7	143	1610.4	141
	12	2202.4	242	2185.7	238
60	1	443.9	27	442.3	45
	9	945.0	26	943.2	27
	10	1373.5	75	1376.9	76

Figure 2.8: The spectral bands of the two Sentinel-2 satellites, S2A and S2B [15].

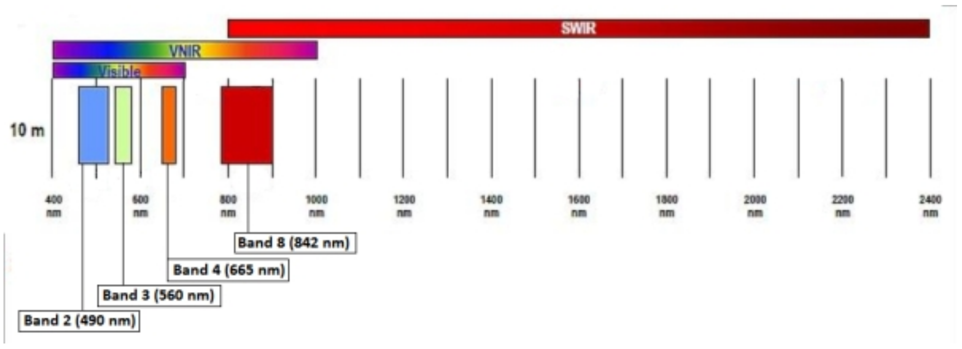


Figure 2.9: The spectral channels of 10 m spatial resolution [15].

points, which is needed to generate the Level-1C data. Finally, Level-1C does a resampling of the Level-1B image to achieve orthorectified tiles with top-of-atmosphere (TOA) reflectances. The product is composed of 100 km x 100 km tiles in the UTM/WGS84 projection. It also computes cloud and land/water masks for the tiles in the geometry. The processing can be broken down into the following steps:

1. Selection of pre-defined tiles in UTM/WGS84 intersecting the user-specified footprint that is the geographical area of interest, of the image.
2. Computations of the resampling grids linking the points of the initial image to the target (ortho-) image.
3. Resampling of each spectral band in the target image using the computed resampling grids and interpolation.
4. Calculations of TOA reflectances in the target image.
5. Computations of cloud and land/water mask for each tile.
6. Compression of the imagery using the JPEG2000 algorithm and a GML geographic imagery-encoded header.

The algorithm can be studied in more detail at ESA's webpages [15].

In later chapters and in the analysis, when referring to a Sentinel-2 image, it is one of these level-1C 100 km x 100 km image tiles that is referred to. As mentioned briefly, they come in the UTM projection using the WGS84 ellipsoid, and they are formatted as JPEG2000.

Level-2A

The Level-2A is a prototype product that performs atmospheric correction to the Level-1C products, providing bottom-of-atmosphere (BOA) corrected reflectances, as seen in Figure 2.10. It also provides enhanced scene classifications that provide pixel maps that classify clouds, cloud shadows, water, and vegetation.

APIs and Metadata

As mentioned above there is an official Sentinel API one can use for developing and creating own applications [1]. Actually, there are two APIs: the Open Data Protocol (OData) and the Open Search. They are complementary to each other, and together they let the user make product requests based on parameters like sensing time and the image footprint on Earth's surface. However, there also exists other third-party APIs that are built on a combination of these, like Sentinelsat [17]. Sentinelsat is written in Python and is

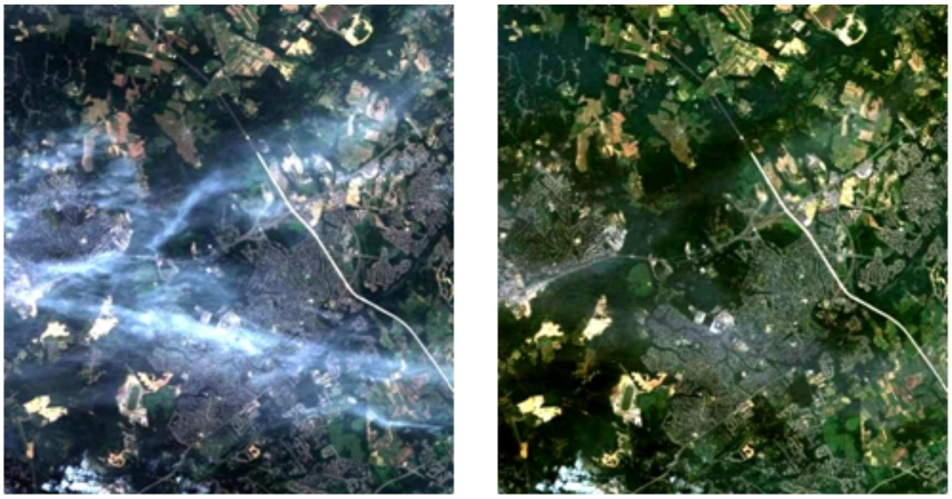


Figure 2.10: Level-1C TOA product (left) and corresponding Level-2A BOA product (right) generated by the Sentinel-2 Toolbox [16].

very well documented, making searching, downloading and retrieving metadata from the Open Access Hub much easier. Also, Amazon Web Services (AWS) stores most Sentinel products in their databases and has an API that lets you download the data directly from them [18]. An advantage of using this service is that they let one specify what parts of the products one want to download. As an example, if one is only interested in the B2-bands, one can choose to download only this, in contrast to the interface and APIs provided by ESA, where one will get the whole product folder as described in the following paragraph.

The products have a lot of metadata one can use as query parameters to obtain suitable images. The size of an image tile is about 500 MB. Thus it is useful to be able to filter out unsuitable images before downloading them. Whether an image is unsuitable or not, depends on the particular use case. In the case of the present project, unsuitable images are, for instance, images over land areas or images entirely obscured by clouds. The available metadata range from being instrument and satellite specific to being more image specific, presenting the user with parameters such as geographical footprint, sensing time, datatake id and cloud cover percentage.

Data Formats

The products follow a specific SENTINEL-SAFE format, which includes the actual image data, image quality indicators, auxiliary data, and metadata. If one chooses to use the graphical user interface or the API provided by ESA to download products, one re-

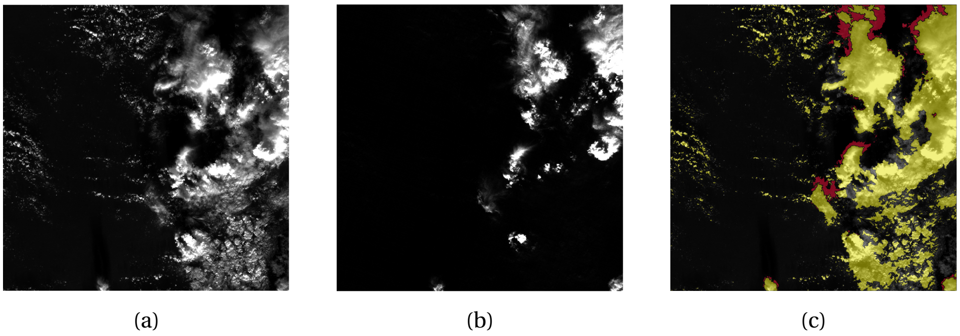


Figure 2.11: Clouds in satellite imagery over the North Sea near Stavanger, Norway. Register how the bands detect different types of clouds. (a) B2 have high reflectance in low atmosphere clouds, while (b) B10 shows high reflectance in high atmosphere. (c) B2 band with Cloud Mask where yellow and red color shows dense and cirrus clouds, respectively. Images were downloaded through the Copernicus Open Access Hub [1] and visualized in QGIS [19].

ceives a zip-compressed folder that contains a collection of information sorted in various sub-folders. One of the sub-folder contains the actual image data, containing all the 13 spectral bands, along with metadata and quality indicators (e.g. cloud mask). The images and quality indicators are in GML-JPEG2000 format. As mentioned in the previous paragraph, there exists other APIs, like the AWS API, which let one bypass this folder structure and downloads the individual data directly. If one focuses on, for instance, only some of the spectral bands, this might be the way to go.

Cloud Mask

It is useful to identify the number of clouds and their extent in the images. The Level-1C processing utilises different features from the spectral band reflectances to generate cloud masks that enable the cloudy and cloud-free pixels to be determined. There are two types of clouds it distinguishes: dense clouds and cirrus clouds.

The dense clouds are characterised by having high reflectance in the blue (B2) spectral channel. However, since snow also has high reflectance in B2, they additionally use the SWIR reflectances in B11 and B12, where it is possible to distinguish snow and clouds. Cloud reflectance in SWIR is high, whereas snow has a low reflectance.

The cirrus clouds are thin clouds forming at high altitudes and can be transparent or semi-transparent. Being semi-transparent and high-altitude, they usually have a low reflectance in the B2 blue band and high reflectance in the high atmospheric absorption band B10, respectively. There can be exceptions to this, however, so to limit false detection they apply a filter technique based on morphology operations on both dense and cirrus masks. After the filtering, if a pixel is both dense and cirrus, it will be labelled as only a dense cloud.

An illustration of the difference in reflectances in the bands is shown in Figure 2.11, together with the generated cloud mask for the image.

The result is a cloud mask with three possible values; 0, 1 and 2, where 0 is a cloud-free pixel, 1 is a dense pixel, and 2 is a cirrus cloud pixel. Due to the usage of the SWIR bands, the spatial resolution of the cloud mask is 60 m. Therefore, they do a radiometric interpolation to resample it into resolutions of 10 m and 20 m corresponding to the other spectral bands.

The Level-1C product uses the cloud mask to determine the cloud cover percentage in the image. This percentage is, among other parameters, stored in the metadata of each image, and can be useful information when determining what images to choose for potential analysis or other use cases [15].

Chapter 3

Basis and New Discoveries

In the present project, we want to explore a Big Data approach to automatically generate a training dataset consisting of satellite images labelled with ship positions. The future goal is for this dataset to be used in supervised machine learning to train a neural network to detect marine vessels in satellite images. The approach is based on an integration of the two datasets; linking ship visual features in the multi-spectral imagery provided by the Sentinel-2 satellites, to the ships exact positions provided by AIS.

In the following sections, we exploit the nature of how a neural network learns, as described in Section 2.1. Using this knowledge we establish some target qualities that the resulting training data should acquire to facilitate the machine learning. We kept these target qualities in mind throughout the whole project by defining prerequisites to both the source data itself, and to our own ability to utilise it. These prerequisites thus underlie every step of the way; from our approach to manage and analyse the data, to the choice of the right tools to do the job. Near the end of this chapter, we present critical discoveries challenging our prerequisites. These are further investigated in Chapter 4.

3.1 Prerequisites

Throughout this whole project, we have strived to keep in mind the important qualities that make up a good training dataset. As previously explained in general in Section 2.1, there are especially in two ways the training data can facilitate the machine learning. These are specifically:

1. Influence the accuracy of the predictions made
2. Allowing the Neural Network to generalise well

For the training data to obtain this, certain prerequisites need to be fulfilled. We have divided these into two groups: prerequisites for the datasets and prerequisites for us as data scientists.

3.1.1 Prerequisites for the Datasets

The following paragraphs will inform about what is required by the AIS and Sentinel-2 datasets.

Compatibility and parameters
For the network to be able to predict with high accuracy, the training data has to be precisely labelled. In our case, this means that the AIS dataset and the Sentinel-2 dataset has to be compatible. For them to be compatible, it must be possible to combine them in four dimensions: latitude, longitude, height (over the ellipsoid) and time. In other words, based on the position and timestamp of an AIS message, it must be possible to find a ship in the Sentinel-2 imagery of the corresponding position and time. Hence, these parameters must be available for both datasets. Furthermore, these parameters have to be of high quality and accuracy. Otherwise, the data labelling will be poor and the network will struggle to learn relevant features.

In addition to time and position, there should be some parameters in the AIS data such as ship speed and size. This will help in the design of training data by getting examples of ships of varying size and wakes.

Volume
For the network to generalise well, there have to be large amounts of data, as pointed out in Section 2.1. This holds for both datasets. As the target for the network is to detect ships globally, we need global AIS and Sentinel-2 data to cover all Sun zenith angle. Moreover, we need historical data from all seasons of the year. This will bring some variety in the training dataset, and the chance of correct predictions will increase.

Specifically for the AIS data, the volume of the data is important since we do not know in advance which combinations of areas and time periods are covered by the satellites. Having enough data from AIS, one can make use of arbitrary satellite imagery and be quite sure AIS covers the time and location. Additionally, as both the images and AIS messages have discrete timestamps, there is no guarantee that there exists, for a ship in a given image, a corresponding ship message that matches the timestamp perfectly. This means that the having a larger AIS dataset will increase the chance of finding a ship message that is close in time with respect to the image timestamp.

As for the Sentinel-2 data, a lot of the images will be subject to clouds, primarily, but also noise. These might cover ships with AIS messages, which will lead to false negatives in the training dataset. The amount of these images should be reduced. Thus, to get a satisfactory amount of images without too many clouds or too much noise, the original dataset has to be even more massive.

3.1.2 Prerequisites for the Data Management

As the two datasets are going to be combined, we need to get the information from both sets to be in formats that are compatible. The timestamps need to be formatted the same and should be in the same time zone. The position coordinates need to be transformed to the same projection. We must also have the availability of appropriate tools for managing prerequisites, as well as being able to learn to use them fully. The tools chosen are presented in Appendix A.

As we require a massive amount of data, it demands to be able to handle the challenges of Big Data management. Big Data is defined as datasets that are too large, varied and complex to be processed and managed by conventional data-processing applications. The amount of data generated each day globally is unbelievably vast, 2.3 trillion gigabytes, according to IBM [20]. This growth of data is mainly due to the creation of social networks and other online services, as well as data acquired by sensors related to the Internet of Things (IoT). A great example of this is remote sensing. Big Data is usually characterised by the concepts constituting the four V's of Big Data: volume, variety, velocity and veracity.

Volume
The main characteristic that makes the data "big" is the volume of the data. The total amount of information collected by sensors, social media accounts and other sources grows exponentially every year. As an example for remote sensing; the Sentinel satellites of the Copernicus programme alone generate approximately 12 terabytes of remote sensing data every day [21].

This means that we have to be able to handle the volume of the data in terms of efficiency in storage, structuring, filtering and analysing it.

Velocity
The frequency of incoming data that needs to be processed. Since there are an unlimited amount of sources of data, the velocity of the data streams that need to be analysed is exceptionally high.

We wanted to keep this in mind in the chosen methods, to facilitate future managing of real-time incoming satellite images. The ingestion of the AIS data into our database will also need to be able to keep up with the 30 million new messages collected globally by the AIS.

Variety
Traditionally the variety of data has not been particularly broad. Generally, there have been *structured data* that revolves around transactions related to finance, insurance, health and trades that fit in a relational database. This data is well defined in a set of rules. Nowadays, however, a lot of data has developed into *unstructured data*, such as video uploads, Twitter feeds, audio files, images, diverse sensor data and location-based data. Different forms of data that does not follow a set of rules and that has no metadata that neatly defines it. *One of the greatest Big Data challenges is to use technology to combine and make sense of this data.*

The variety of data that we need to manage is, of course, that of the images, and the AIS messages. Luckily, the images are provided with metadata. We need, however, to be able to rely on this data, to successfully make meaningful assumptions about the data. The AIS data also consists of messages of different types, which we need to merge into common structures. Also, the ships sending the messages have varying nature. As an example, due to the laws of physics, a small fishing boat may behave more unexpectedly than a 500-ton oil tanker. This need to be accounted for before making assumptions about the data. Regarding the images, they will contain an enormous amount of different features, and the ship features being only a small subset of it.

Veracity
Veracity refers to the uncertainty of the data. A lot of data are of bad quality or are irrelevant to the analysis. It is essential to implement the right technology to perform quality checks and filter out the unreliable data. In our case, we need to be able to detect fake or inaccurate ship messages. Other factors are different weather conditions in the images, as well as the occurrence of image noise.

3.2 Basis of Data Management

In this section, we present the management and research of the AIS dataset and the Sentinel-2 dataset. This includes the collection, storing and processing of each dataset to facilitate the integration of them.

3.2.1 Management of AIS data

Global AIS data collected from satellite and terrestrial stations with internet connection are made available on the internet through many service providers. This makes near real-time position data from anywhere in the world accessible to anyone. While much of this data is available free of charge, it is considered a high threshold for the user to be able to process the data streams. Third party services are making online tools where viewing the real-time data is free of charge, but special services such as aggregating, and doing filtered searches on their archives, as well as viewing satellite data, are usually supplied at a cost. These are often based on monthly subscription, and the cost increases with the geographical area, and time slot of interest.

In the case of the present project, access to as much AIS data as possible was highly required. This was necessary to be able to generate a training dataset with qualities that further facilitate the machine learning, as discussed in Section 3.1. The Sentinel-2 images geographical coverage is time-dependent, and AIS, on the other hand, provides *continuous global coverage*[1]. It was thus decided that the most flexible solution would be to have access to a complete dataset of AIS, and further filter it based on the footprint and timestamp of arbitrary Sentinel-2 images.

The Norwegian Coastal Administration administers AIS Norway and is responsible for distributing AIS data to other governmental agencies. AIS Norway has stored global AIS data for several years. The current policy is that non-governmental actors can be granted access under certain circumstances based on legitimate needs [23]. By contacting them directly, we were informed that they were willing to supply all AIS data that would be needed for the project, at no cost. Norway has also launched satellites to be used in AIS. Figure 3.1 shows Norsat-2, a Norwegian made satellite equipped with an AIS transceiver, that was launched in 2017.

[1] This statement is a simplification. The coverage of AIS is not continuous, nor has it time independent global coverage, as stated in Section 2.2. Compared to the limited coverage of Sentinel-2, this simplification is yet reasonable.

Figure 3.1: The Norwegian made satellite NorSat-2 launched July 2017. Among other equipment is an AIS transceiver [22].

Collection

The Norwegian Coastal Administration supplied a valid username and password for us to access their online web-based FTP-client. Global AIS-data dating back to January 1st, 2016 was available to download, both data received from terrestrial stations and satellites.

The data was structured as text-based DAT files, one file per day of the year. Each file contained AIS-messages encoded in NMEA format [24]. These are not human readable and are meant to be easy to transfer and store. An example of these encoded messages is given below.

```
Message 1:
    \s:ASM//Port=669//MMSI=,c:1483228800*77\!    ...
    BSVDM,1,1,,A,15OJj:9001kjc0aom;saK8N60@1V,0*62
Message 2:
    \s:ASM//Port=669//MMSI=,c:1483228800*77\!    ...
    BSVDM,1,1,,A,16<ug<101u<RklJ1M`6P00460<00,0*38
```

Preprocessing

Looking at the raw nature of the NMEA-messages, it is clear that it is not possible to do filtered searches on the data. The messages would need to be decoded, and structured in a system that fits the purpose.

For the decoding the NMEA-data, a Python-library was used [25]. It outputs each message into human readable JSON-format [2]. Below is a sample of the attributes returned when decoding the first NMEA-message given above.

```
  timestamp: '2017-0101T00:00.000000Z',
   accuracy: True,
      speed: 0.10000000149011612,
       mmsi: 368489000,
       type: 1,
        lat: -14.276296615600586,
        lon: -170.68370056152344,
       turn: 2.1873741149902344,
     course: 241.1999969482422,
    heading: 511,
status_text: 'Reserved for HSC'
```

This is nicely formatted data, with coordinates in WGS84, the timestamp in ISO 8601 format [26] at time zone UTC, speed in knots, and angles are given in degrees. However, what may seem a little off-putting is the heading angle of 511 degrees. The NMEA-protocol has not defined a particular value for when data is not available. It is instead separate predefined default values for each data field. For the heading, the default value is 511 degrees, which means that no data was available. For the longitude, this value is 181 degrees, and so on for the other data fields [13]. We also get a Boolean value for accuracy, where the *True* indicates a DGPS-quality fix of the position with an accuracy below 10 meters. *False* indicates an unaugmented GNSS fix with accuracy higher than 10m. These particular values indicating *data unavailable* need to be accounted for before using the values blindly in analyses.

Looking up the MMSI on MarineTraffic [9] we can identify the vessel and view their presentation of the real-time data. We also get to see a close-up photograph of the ship, as shown in Figure 3.2.

[2]JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of key-value pairs and array data types.

Figure 3.2: Photograph of the American Fishing Vessel *Cape May* [27].

Storing

To ensure the amount of data is sufficient with respect to the prerequisites described in Section 3.1, we wanted to decode and store as much data as possible. This would optimally mean at least one year worth of AIS messages. In that way, we would be able to get samples of images from all seasons of the year, for any given geographical area.

Studying the files containing messages received by satellite, we observed that the 24-hour files were of size around 200MB. Compared to the files sizes from terrestrial receivers, of around 2.6GB, this was quite modest. Over a phone call with The Norwegian Coastal Administration, we were told that they did collect data received by the four Norwegian-made satellites. They could inform that as of mid-August 2017 they had started collecting data from 18 international satellites equipped with an AIS transceiver, making the global coverage much denser. Checking the file sizes from after August we found them to all be above 600MB, showing the effect of the higher number of satellites. Still, this is far from the amount of data collected by the terrestrial stations, showing that the satellites are not yet ready to replace the terrestrial system.

For the content of these files, we were in for a surprise. While the file sizes are not too ominous, a typical terrestrial file of size 2.6GB contains nothing less than 27 million messages. That is 27 million messages per day, making it justifiable to have them stored as encoded strings to reduce space. The method for inserting these amounts of data into a database will need to be contemplated thoroughly in order to retain efficiency.

Database

As previously stated, we needed the data structured in a database for easy filtering. Based on earlier experience with the open source geospatial database [28], it was considered the superior choice. It supports a wide range of geospatial functions (e.g. finding all ships positioned within an images bounding coordinates).

To set up the database in a pragmatic way it was necessary to understand the message standards and choose proper parameters for the columns. These include using the column type *TIMESTAMP* for the DateTime values, and the geospatial type *POINT* with WGS84 defined as the coordinate reference system.

Spatial indexing of the entries was applied [29]. This means that all entries are structured in an R-tree, which groups spatially nearby points together for fast look-up. This will vastly increase the speed of spatial queries when, in our case, searching for all ships located within an images geospatial footprint. This kind of indexing was also applied to the timestamp column for faster temporal filtering.

We decided only to create columns for the AIS message attributes we consider the most relevant to our use case. The rest of the 30 attributes are stored in a JSONB-column [30]. This means that the data is stored as JavaScript Object Notation in a decomposed binary format. It makes the insertion slightly slower due to added conversion overhead, but in return leaves it significantly faster to process. JSONB also supports indexing, which can be a significant advantage when filtering within its contents.

A ships message is transmitted as two separate messages, at different times. These are:

1. The static data report message, describing unchanged characteristics of the ship
2. The position report message, containing information that is changing over time such as the ships current position

To avoid redundancy, it was natural to separate these into two tables, which are related to each ships distinct id, the MMSI.

Static messages

From the messages documentations [13], we found that that the static messages, which each ship is transmitting every 6 minutes, can often be transmitted as two separate parts. For simplicity, we wanted the table to contain the full static message. This meant that, for each new decoded static message, we would have to check two cases:

1. Does our database contain static information about the given ship

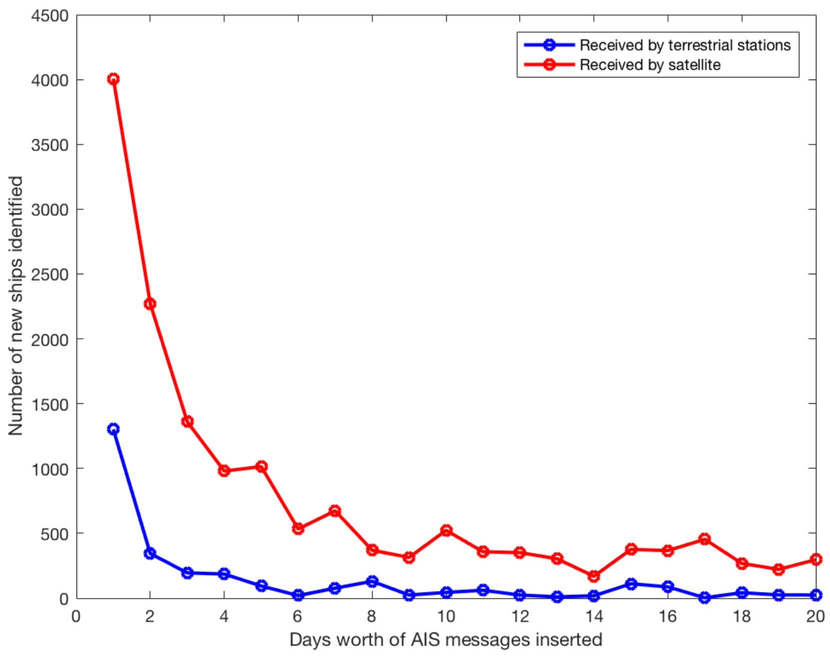


Figure 3.3: Number of new ships found per one-days-worth of AIS messages covering a 20 day period. Most new ship is found in messages received by satellite. This due to inserting all messages received by satellites before those received by terrestrial stations.

2. Does the message contain new or changed information about the given ship

This was achieved by using the ship MMSI as the table primary key and then performing an *UPSERT* [27]. This means to detect existing MMSI in the table, and doing an update on the columns where new information is present. Else, if not existing already, insert it as a new row. Hence, over time the database would accumulate new static messages from all ship, resulting in an updated list of all distinct ships broadcasting to AIS. As seen in Figure 3.3 the number of static messages received from ships not yet inserted is rapidly decreasing for each file inserted to the database. After inserting 20 days worth of AIS messages, the total number of distinct ships in our database slowly worked its way up to 21 thousand ships. Table 3.1 describes the attributes of the AIS static messages in the database.

Positional messages

Each ship transmits positional messages about every two minutes. Other ships may

Table 3.1: Description of the attributes in the database table *ais_static*, containing AIS static messages.

Column	**Type**	**Description**
mmsi	integer	Maritime Mobile Security Identity
shipname	character varying	The ships name
type	integer	AIS report message type
to_bow	double precision	Distance in meters from AIS-transmitter to bow
to_stern	double precision	Distance in meters from AIS-transmitter to stern
to_port	double precision	Distance in meters from AIS-transmitter to port
to_starboard	double precision	Distance in meters from AIS-transmitter to starboard
other	jsonb	All other message data

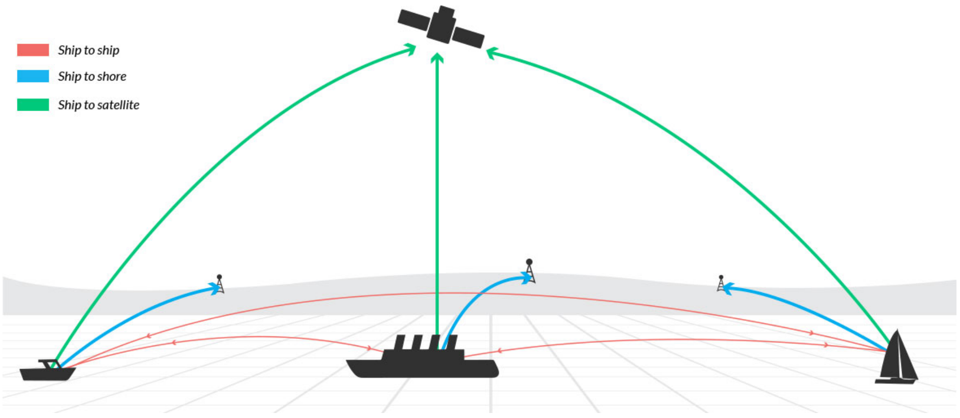


Figure 3.4: The AIS messages can be received by both terrestrial stations, AIS-endabled satellites, as well as other ships. This may result in duplicate messages being archived [31].

then receive these messages and re-transmit them to increase the likelihood of a satellite or terrestrial station receiving it. This is illustrated in Figure 3.4. As a side effect of this, the message can be received multiple times by the same receiver, or even by multiple receivers. This will, in turn, produce a lot of duplicate messages. We wanted to detect these duplicates to minimise the database storage load. With this in mind, we designed the database table for the positional messages to have a combined primary key of the ship MMSI and the message timestamp. Then, when inserting a new message, the database would ignore new messages where an existing entry contained the same values for the MMSI and timestamp. Table 3.2 describes the attributes of the AIS positional messages in the database. Figure 3.5 shows an example of AIS positional messages received from a boat docking Trondheim.

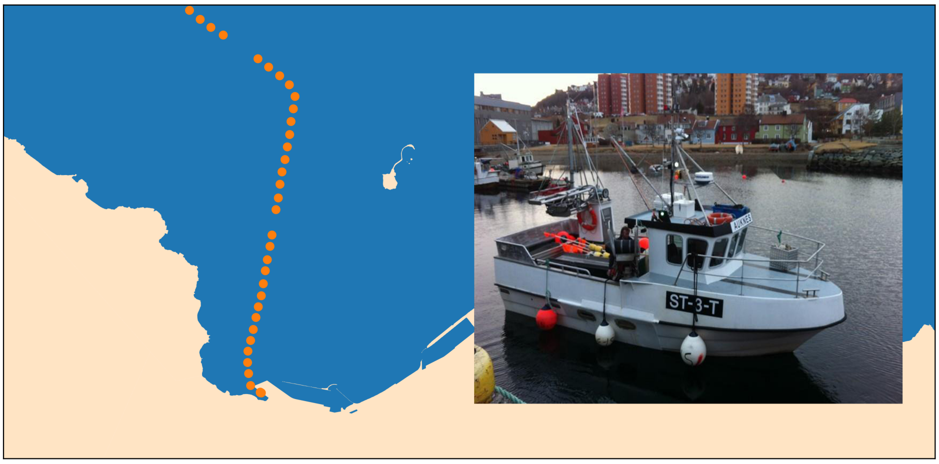


Figure 3.5: Example of discrete ship positions received from a single boat. The data is is transmitted from the Norwegian shipping boat Gulliver while docking Trondheim [32].

Table 3.2: Description of the attributes in the database table *ais_position*, containing AIS temporal messages.

Column	**Type**	**Description**
mmsi	integer	Maritime Mobile Security Identity
timestamp	timestamp	UTC time when the report was generated by the transmitter
geom	geometry(Point,4326)	Point position in WGS84
accuracy	boolean	Position has better accuracy than 10 meters when set to True
course	double precision	Course over ground in degrees
heading	integer	The ships true heading in degrees
turn	double precision	Rate of turn in degrees per minutes Positive/Negative, Starboard/Port
speed	double precision	Speed in knots
status	character varying	Navigational status message
type	integer	AIS report message type
shiptype	character varying	Description of the ship
other	jsonb	All other message data

Result of filtration

Testing these customised inserts on a file containing *27 million* messages resulted in 5 million messages getting filtered out as duplicates. Another 4 million was filtered out for having message type not relevant to our use case, bringing the total number of inserted messages down from *27- to 17 million*. This was invaluable optimisation for being able to store as much data as we considered optimal for the generation of a useful training dataset.

3.2.2 Management of Sentinel Data

Sentinel data, along with the other data gathered by the Copernicus programme, is available free of charge to all users who register an account at ESAs Copernicus Open Access Hub[1]. As the satellites continuously orbit the Earth capturing multi-spectral imagery, there are 4TB of data coming in every day, from Sentinel-2 alone [33]. In combination with historical data, there is an enormous volume of imagery available. This data has variable quality and relevance depending on the particular use case.

In this analysis, we address the challenge of detecting ships. The imagery of non-maritime areas is of no interest for our purposes. Moreover, an eventual neural network should generalise well enough to be able to recognise ships in oceans and waters globally in all seasons. Hence, ideally, the data collected should be from areas all over the world and spread in time from all seasons of the year. Naturally, most importantly is the presence of ships in the images. Therefore areas known to have a lot of ship traffic is preferred, such as coastal areas close to harbours. In Section 4.3, we describe this more in-depth and how we have narrowed down the image search regions with a significant amount to optimise the image selection process.

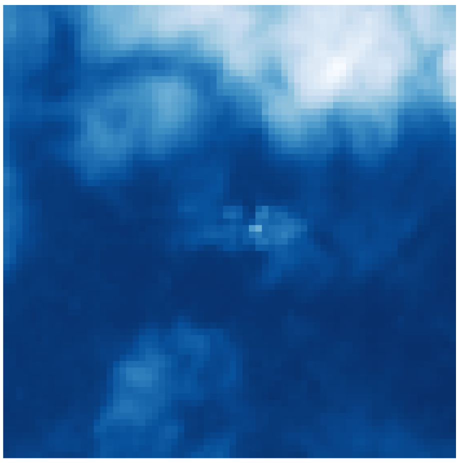


Figure 3.6: Extracted image region, of a ship visible through a thin cloud. The image is presented in band B04.

However, AIS will have an important role in selecting these areas. As a neural network needs both positives and negatives to learn features, ships are not wanted in all images.

Table 3.3: Confusion Matrix

		Label	
		P	**N**
Ship Visible	**p**	True Positive	False Negative
	n	False Positive	True Negative

Some images have to be void of ships for the network to distinguish between ships and clear ocean. Other issues are the possible occurrence of clouds, and whether or not they are transparent enough for ships to be visible through them, and the fact that there may be ships in the images that are not tracked by AIS. In Figure 3.6, we see an example of a ship in a Sentinel-2 image that is semi-transparent through clouds. There are a number of reasons for this to happen. For instance, the ship might have turned off the system or is not covered by the regulations of AIS usage. Ships like this in the images can lead to false negatives in the dataset, which will disrupt the effect of training in a deep learning algorithm. On a similar note, dense clouds covering ships tracked by AIS can lead to false positives, because the image is labelled with a ship that is not visible. The confusion matrix represented in Table 3.3 illustrates this concept.

Collection

As mentioned in Section 2.3.3, one can either access data through a graphical user interface, or by using an API. As mentioned in Appendix A, in the present project the Python API, Sentinelsat, was used to access image metadata. The API offers ease of filtering the results using query parameters that let us handle the aforementioned requirements.

To filter out images of high cloud density, we can query on the cloud cover parameter. Based on information from ESA [15], we assume the cloud cover percentage is calculated from the number of both dense and cirrus cloud pixels in the image. Cirrus clouds can be transparent or semi-transparent, and thus a high cloud cover percentage does not necessarily mean no transparency. However, as a safety measure, by setting a low threshold for the cloud cover percentage one can be quite sure potential ships are visible in the image.

By using Sentinelsat, one can download product metadata based on the aforementioned parameters and download the actual image files. This allows fast download of the metadata alone, such that it can be analysed for relevance before downloading the huge images. The image products in its whole will come structured as described in Section 2.3.3.

Examination

Early in the project, we had not yet developed a good approach for finding examples of a ship in the images. As this was an objective for later research, as described in Section 4.3, we had to inspect satellite images for ships manually. By filtering with a low threshold for cloud cover percentage, and experimenting with different time intervals at areas of interest, we were able to find examples of ship objects of high quality [3] in the images, as illustrated in Figure 3.7.

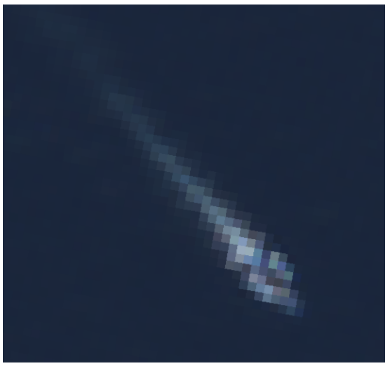


Figure 3.7: Ship found in Sentinel-2 TCI. Based on the pixel resolution of 10 m, we can estimate the length of the ship to be approximately 80 meters. This is only an approximation as it is hard to distinguish ship and wake only by observation.

The same ship is shown in the bands B2 and B8 in Figure 3.8. It is worth mentioning that all three images are cropped at the same zoom level. Notice how the ship's wake stands out in B2 and B8 compared to the true colour image (TCI). When analysing the pixel reflectance of the ship in the images, it is clear that features which are hard to distinguish in the visible spectrum, are far easier to separate using the other spectral bands.

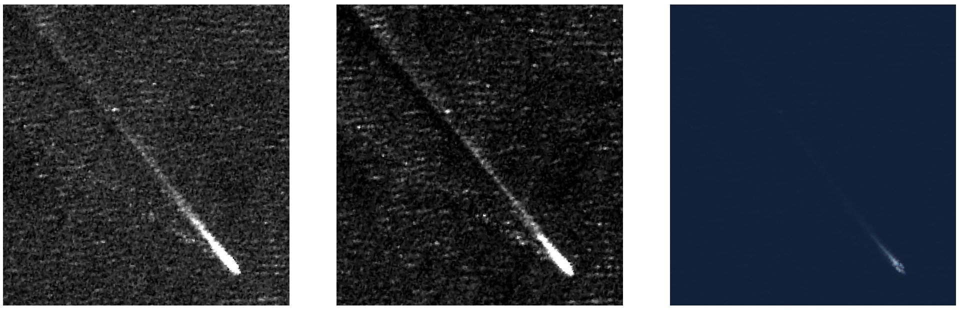

(a) (b) (c)

Figure 3.8: Ship shown in the multi-spectral bands B2 (a) and B8 (b) and TCI (c). The images are cropped at the same zoom level. The wake stands out more in B2 and B8, compared to TCI.

[3]The quality is described relative to the image resolution.

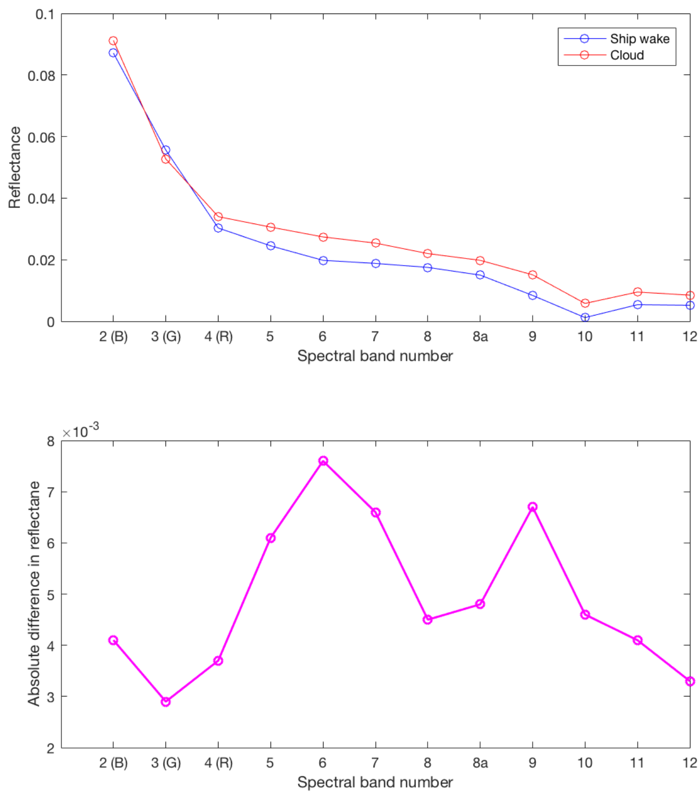


Figure 3.9: Difference in reflectance per spectral band between a pixel found in the wake of a ship and a pixel found in a cloud.

Another example is distinguishing the wake of a ship from the surrounding clouds in the image. The result of this examination is illustrated in figure 3.9. Wakes and clouds can be hard to distinguish using only the three visible RGB channels consisting of band B2, B3 and B4. This is indicated in the second graph where the absolute difference in reflectance is quite low for these three channels. On the other hand, the near-infrared bands B5, B6, B7 and B9 shows a significant difference in reflectance, making the features more distinguishable.

Storing

Each Sentinel-2 tile image product takes up around 500 MB of disk space. Additionally, there is only a small portion of the total data products that are relevant for this project. To determine which products that are relevant, we filter them by querying the metadata, which results in using a lot less storage space and faster to download. In the present project, no metadata was stored on disk due to the fast downloading time. Instead, when doing queries in combination with the AIS data, the metadata was stored only temporarily in memory until having found the suitable products.

3.3 Testing Data Integration

As mentioned earlier we want to use AIS as ground truth for the position of ships in the images. To test the compatibleness of the two datasets, we did a simple integration analysis, to see how well the AIS messages match with ship objects in the images. To do this, we needed an image containing ships, which can be found by querying the database containing the AIS messages. However, we will not go into detail about the method chosen to do this, as the method will be further elaborated in Chapter 4.

Nevertheless, the result of testing the data integration, it became quickly evident that there was, at least to some degree, compatibleness between the datasets. Observed ship objects in the images were relatively close to the AIS positions, as we can see in Figure 3.10.

However, the results of integrating the AIS messages and Sentinel-2 images also made it clear that some temporal synchronisation issue is present between the two datasets. In the figure, we have plotted two ship positions provided by AIS onto a satellite image. These are the two AIS-messages prior to and past of the timestamp associated with the image. Hence one intuitively would expect the three positions to be in strictly increasing order along the trajectory of the ship, or mathematically:

$$T_1^{AIS} < t^{image} < T_2^{AIS} \tag{3.1}$$

By inspection we found that this is not the case. Both AIS-position's timestamps are past of the image timestamp, that is:

$$t^{image} < T_1^{AIS} < T_2^{AIS} \tag{3.2}$$

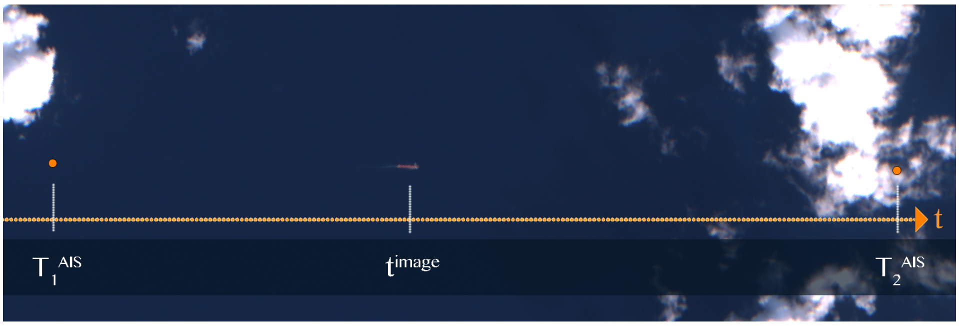


Figure 3.10: Two AIS messages plotted on a Sentinel-2 image. Due to a temporal synchronization issue, the ordering of the three timestamps, for the two AIS messages and the image, does not correspond to the ordering of the actual ship positions.

This observed time offset or temporal synchronisation issue is somewhat discouraging, as it is essential for the labelling of the training images that we can integrate the Sentinel-2 images and AIS accurately. In the next section, we go deeper into this issue, to understand where it is coming from, and what we can do to fix it.

3.4 Uncovering Challenges

In this section, we will highlight some critical challenges that we uncovered during our research of the data, and in the development of methods for managing it. Our solutions to these challenges are presented in Chapter 4, along with discussions about further improvements. The challenges turned out to be critical for the success of the project, as they were significant setbacks that challenged the core concept of the solution. They are directly in demur with the prerequisites defined in Section 3.1, such that, by in large, it became some of the main focuses for the project.

3.4.1 Incomplete Sentinel-2 Metadata

To understand the cause of the observed time offset between the datasets, as described in Section 3.3, the different time parameters in the metadata of the Sentinel products were examined thoroughly. As previously mentioned in Section 2.3, two types of product metadata are available from the APIs provided by ESA: the OpenSearch metadata and the OData metadata. The OData API provide the more complete data of the two, but in essence, most of the parameters are the same for both APIs. An example of relevant parameters from OpenSearch and OData, along with their descriptions, are listed

Table 3.4: Table showing timestamp relevant metadata for a product from both Sentinel APIs

Parameter OData	**Parameter OpenSearch**	**Value**
Filename	filename	S2B_MSIL1C_20170921T130239 _N0205_R095_T23KPQ _20170921T130809
Mission datatake id	s2datatakeid	GS2B_20170921T130239_002838 _N02.05
Datatake sensing start	datatakesensingstart	2017-09-21 13:02:39.027000
Sensing start	beginposition	2017-09-21 13:02:39.027000
Sensing stop	endposition	2017-09-21 13:02:39.027000
Ingestion Date	ingestiondate	2017-09-21 19:57:18.488000
Generation Time	N/A	2017-09-21 13:08:09
Date	N/A	2017-09-21 13:02:39.027000
date	N/A	2017-09-21 13:02:39.027000

in Table 3.4 and Table 3.5.

As the description in the tables informs, the *Sensing start* and *Sensing stop* parameters (corresponding to *beginposition* and *endposition* in OpenSearch) should intuitively be the time when the satellites senses the first and last line of the image, respectively [34, 35]. In our previous work, these parameters were used as timestamps to combine the Sentinel images with the AIS ship messages. This is where we experienced a time offset between the two systems.

Interestingly, when looking at the values in Table 3.4, one can see that both of these timestamps are exactly equal to the timestamp of the *datatake sensing start* parameter. As described in Section 2.3, the satellite imagery is captured as *datatakes*, where the satellite is in sensing-mode. It flies over an area scanning the Earth's surface, continuously capturing the imagery that the image-products consist of. The length of a datatake can be up to 15 000 km in a period up to around 30 minutes. This means that a datatake potentially can consist of hundreds of images. When using a product's datatake ID as filter criteria in a new query, we obtained metadata for all products that are part of that datatake. We discovered that all products within the same datatake have exactly the same values for Sensing start and Sensing stop time, and this value is always equal to the datatake sensing time. In other words, these two parameters are not the correct timestamps for when each image was sensed, in contrast to what is described at ESAs web pages [34, 35].

To be sure that these timestamps are not hidden elsewhere in the metadata, the other time parameters were investigated as well. Ingestion date is the date when the product

Table 3.5: Metadata parameters from both Sentinel APIs, with description found at the Copernicus Open Access Hub pages [34, 35].

Parameter OData	**Parameter OpenSearch**	**Description**
Title	title	Product name
Mission datatake id	s2datatakeid	Id for the datatake the product is part of
Datatake sensing start	datatakesensingstart	Sensing start time for the datatake the product is part of
Sensing start	beginposition	The time of the satellite on-board acquisition of the first line of the image in the product
Sensing stop	endposition	The time of the satellite on-board acquisition of the last line of the image in the product
Ingestion Date	ingestiondate	Date when the product was inserted into ESA's databases
Generation Time	N/A	Date and time of the product processed at the ground segment
Date	N/A	Information not found, but values correspond to datatake sensing start time.
date	N/A	Information not found, but values correspond to datatake sensing start time.

was inserted into ESA's databases, and generation time is the timestamp for when the product was processed at the ground segment. In the OData, there are also two other "Date" parameters, but the values of these also correspond to the datatake sensing start time.

In addition to this, one can find two timestamps in the file name of each product:

S2B_MSIL1C_**20170921T130239**_N0205_R095_T23KPQ_**20170921T130809**

which follows this naming convention:

MMM_MSIL1C_**YYYYMMDDHHMMSS**_Nxxyy_ROOO_Txxxxx_**<Discriminator>**

The first date is also equal to the datatake sensing time. The second date is the Product Discriminator, which is, according to ESA, used to distinguish between different end-user products from within the same datatake, and can be earlier or slightly later than the datatake sensing time [36]. By comparing the values, it looks like this second date is equal to the generation time parameter. In other words, neither of these are the parameters we search for. Consequently, there exists no precise information about when each individual image was sensed. The Sensing start parameter might be wrong by up to 30 minutes, depending on the length from the start of the datatake to the image.

As this timestamp is one of the fundamental prerequisites for accurate labelling of the ship positions in the training dataset, it is crucial for us to find a method of estimating it more precisely.

3.4.2 Big Data Challenges

Up to this point, the system had only been tested on a very limited amount of data. The initial pragmatic strategy had been to *fire a tracer bullet* [37] throughout the system, testing the data flow from beginning to end. This way we controlled the feasibility of the core concept, namely the integration of the two datasets. Hence, we early uncovered the missing puzzle pieces of the Sentinel-2 metadata (Section 3.4.1), before spending all our time optimising the data management.

Parallel to investigating solutions to the Sentinel-2 metadata challenge, we started analysing the performance of our system, specifically the database management of the AIS messages. In this section we will highlight our main challenges considering the insertion, storing, and filtering of it with respect to the Sentinel-2 metadata. How these challenges were handled is explained in Section 4.2.

Insertion

The first version of the program we wrote for decoding and inserting AIS messages was very basic. First decoding one message, then inserting it into the correct table using an Object Relation Mapping (ORM) [38] between Python objects and the database tables. Using an ORM makes database migrations easy to perform, and the code becomes very tidy and easy to understand. It quickly became clear, however, that the ORM was not suitable for the multitude of database insertions we would perform. While it works just fine for low scale inserts on a small project, it is just not made to perform on a bigger scale. The execution time of decoding and inserting 7 million messages was 9 hours. Extrapolating, it would mean that the execution time for each file of AIS-messages (30 million) would be a staggering 40 hours. This is due to the high number of server round trips when inserting the messages one by one.

An effort was made to make the program stream and decode an optimised sized batch of messages, then use special functions to insert the whole batch at the same time [39]. It was a little tricky to combine this batch insertion functions with the *UPSERT*-criteria

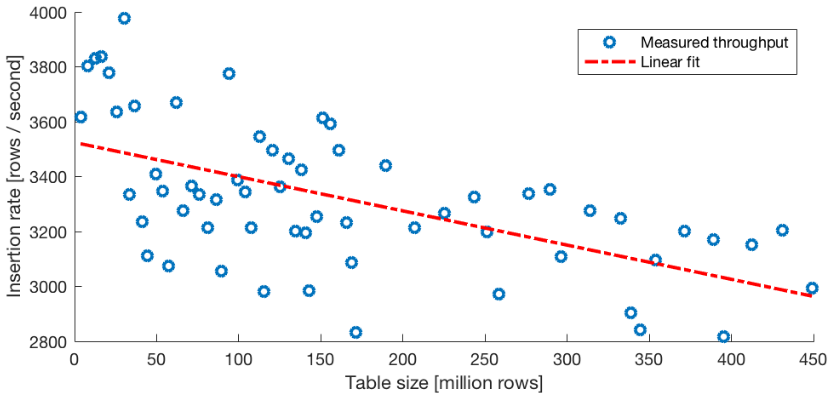


Figure 3.11: Throughput as a function of table size. The timing of insertions also includes decoding the AIS-messages from raw NMEA-format, adding additional overhead to the execution time. This overhead is constant, and does not influence the change in insertion rate.

to avoid duplicates (Section 3.2.1), but this was eventually sorted out. The execution time for the new program, processing 30 million messages, was now down to 1.2 hours. Compared to the estimated execution time of using the ORM method, 40 hours, this change made a huge difference in performance.

However, when processing 30 million messages, we are not talking about Big Data yet. 30 million messages correspond to only *ONE* day of ship messages. Even after filtering out 28% [4] of the messages as being duplicates, the number of messages spanning one whole year will become close to *6,5 billion* temporal messages. Having all this data in just one single database table is bound to cause trouble.

Figure 3.11 shows some of the consequences in practice. The plot shows how the insertion rate changes as the table size increases. 450 million rows correspond to 25 days of AIS messages. Even though the variance of the insert rate is high, it is clear that the rate drops with the increase in table size. This is something that needs to be accounted for if we want to store 15 times as much data.

Querying

If insertion speed was the only concern from having hundreds of million rows in a single table, it would not be too vital for the project. We could just let it run, and it inevit-

[4] 28% was estimated to be the average amount of messages filtered out during insertion. These are either duplicates or having a message type not relevant to our purpose. Even though we do not have an actual number of the ratio between the two, there is no doubt of the majority being duplicates.

ably would finish the insertions. However, as the table size increase, so does the query speeds. Let have a look at a typical query we are performing:

Code Listing 3.1: Find all ships within image

```sql
WITH image AS (
  SELECT ST_GeomFromText('
    POLYGON((
      3.5635641398 60.3252941106,
      5.5477274289 60.3917009786,
      5.6484839974 59.4075003635,
      3.7216859503 59.3436536606,
      3.5635641398 60.3252941106
    ))
  ',4326) AS footprint --Specifying WGS84
)
SELECT *
FROM ais_position
,    image
WHERE ST_Within(ais_position.geom, image.footprint);
```

Examining the four last lines of the query shown in Code Listing 3.1, it is understandable that it returns all AIS position messages (2.2.3) within a given image footprint[5]. When executing it on a small table consisting of 500 thousand rows, the execution time shows to be 0.142 milliseconds. That is not too bad considering the AIS messages are scattered across the whole globe, and we are trying to find only the messages located within a relatively tiny area on the earth surface. Certainly the GIST-index described in Section 4.2 is doing its job.

Anyhow, when scaling up the amount of data in the database, from 500 thousand to 400 million rows, the behaviour is not as expected. Without indexing, the spatial query 3.1 takes 3.2 minutes to execute. After applying the GIST-index the execution time increase to 4 minutes!

In Section 4.2 we investigate a solution to the challenges entrained by having a massive volume of data stored in one single database table.

[5]The footprint is defined by the four corner coordinates of the image.

Chapter 4

Analysis and Technical Solutions

In this chapter, we present the in-depth analysis and the technical solutions that are engineering in this project. First, the solutions that had to be implemented to compensate for the critical challenges uncovered in Chapter 3, are presented. This includes the implementation to estimate image timestamp as well as the big data management challenge. Following this, we analyse optimisation for selecting Sentinel-2 image tiles based on global ship traffic and Sentinel-2 coverage. Then we present the technical solutions for estimating ship positions within images and the extraction of image regions of interest, for the generation of training data. In the end, we discuss possibilities and issues relevant to the design of a quality training dataset, for accurate labelling and generalisation.

4.1 Approaches for Estimating Sentinel Image Timestamps

As the process of labelling training data needed to be automatic, the solution to the problem of estimating the true timestamp to an arbitrary image had to be generalised for all cases, as we could not monitor each case. With this in mind, three different approaches were researched:

1. **Extrapolation:** As suggested by ESA [40]. To estimate the sensing time of the image, find the coordinates of the first image to be captured in the datatake. For this image, the datatake sensing start time will be correct. Then calculate the spatial distance from the first image to the centroid of the image of interest. By using this distance, the datatake sensing start time, and the velocity of the satellite, on can get an estimate of the timestamp for the image.

2. **Interpolation:** Use information from Sentinel-2 Acquisition Plans to interpolate the timestamp of an arbitrary image in the datatake. The acquisition plans contain a-priori approximations for the geographical footprint covered by the datatake, along with timestamps for both the start and end of the acquisition.

3. **Map timestamps from orbit model:** Use satellite ephemerides to generate a high precision 4-dimensional model of the satellite orbit within a specific time period. That is a model containing both spatial and temporal information about the satellite's orbit. Use this in combination with the images' footprint coordinates, and relative orbit name, to map timestamps to the images.

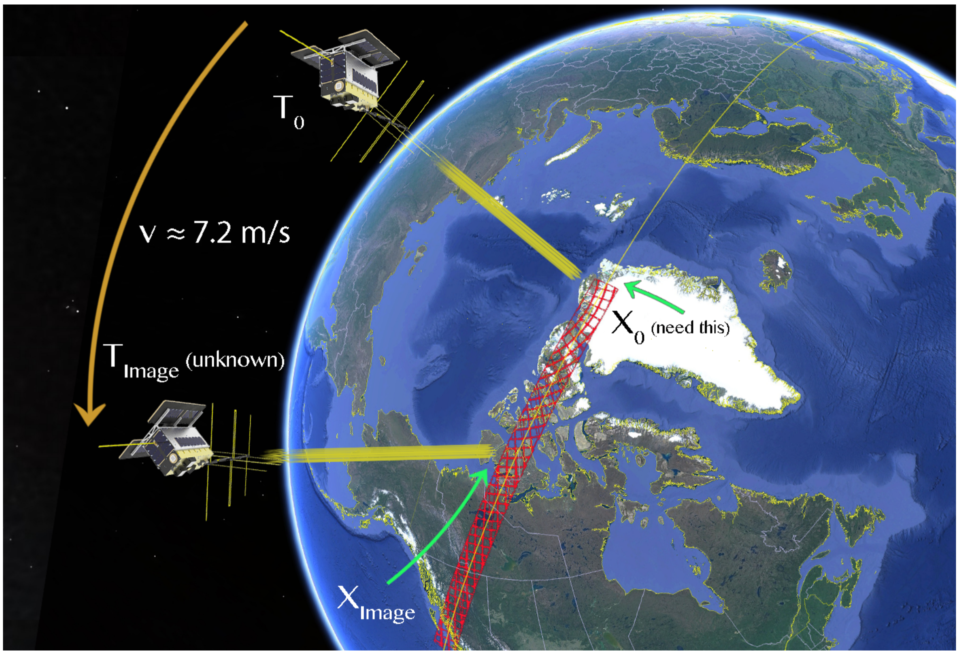


Figure 4.1: Illustration of using extrapolation as a method to estimate timestamp at the image sensing time. This requires knowing the coordinates for the temporally first image sensed within the datatake. The red grid in the figure shows the geographical footprint of every image in the datatake.

4.1.1 Extrapolation

This method was considered under the assumption of not having any information about the products except for what is provided in the metadata. We wanted to exploit the fact that we were provided with the datatake sensing start time, which would be the correct timestamp for the first image to be sensed in the datatake. Computing the spatial distance from the first image to the image of interest, then dividing by the average speed over ground for the satellite, one will obtain an estimate of the timestamp of the image of interest. The method is illustrated in Figure 4.1.

The metadata for each image also includes an attribute *s2datatakeid*, which is the identifier of the datatake in which the image was sensed (see Section 2.3.2 for technical information in this regard). Having this, one can perform a new query, filtering on this datatake id. The resulting dataset is metadata for all the products, i.e. images, within the datatake.

The number of products within one datatake can be in the hundreds, and there is no in-

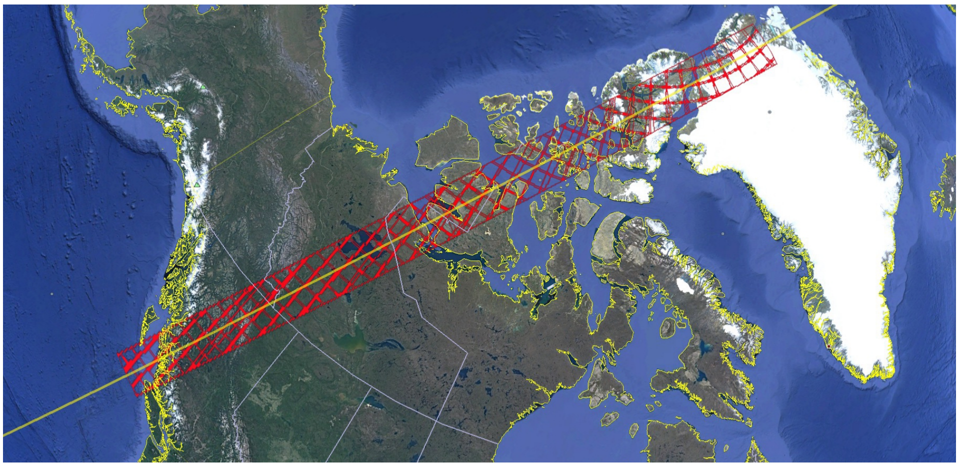


Figure 4.2: Illustration of the geographical footprint provided for all images within a datatake.

formation regarding the actual order in which the products were sensed. When downloaded, the products are ordered by ingestion time, which is the timestamp for when the product was archived. One cannot rely on this being the same order as the order of sensing. This means that there is no direct way of finding the temporally first image. This is needed to extract the geographical starting point of the datatake, which is required for the extrapolation.

However, every product has a geographical footprint parameter, represented as a polygon containing latitude and longitude point-coordinates in the WGS84 projection. Figure 4.2 illustrates the geographical footprint for images within a datatake. We began to research the possibility of using these coordinates, in combination with what we knew about the satellite orbits, to deduce a spatial ordering of the images.

Ordering Products by Latitudinal Coordinates

As the satellites have a near-polar orbit, a natural first step was looking at the latitudinal coordinates of the images. The starting point would intuitively be the northernmost or southernmost point, depending on the orbit direction of the satellite during the datatake. In other words, when the satellite is in ascending mode, which means it orbits towards the north pole, the starting point would be at the image having the most southern latitude. Similarly, for the opposite direction. When the satellite is descending, the starting point would be at the image of highest latitude. Figure 4.3 illustrates the two orbits directions.

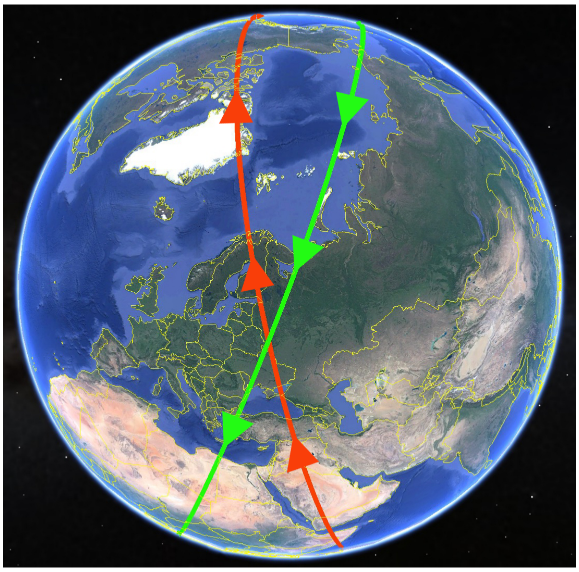


Figure 4.3: Examples of two Sentinel-2 orbits. Ascending orbit (red), and descending orbit (green).

Whether or not the satellite is in ascending or descending mode during the sensing is included as a parameter in the metadata for each product. It would make sense to sort all products of a datatake by footprint latitude, and then, according to the orbit direction, pick the first or last product. This would work if all the products of a datatake were tagged either as ascending or descending. However, problems arise when the datatake crosses over one of the poles, as illustrated in Figure 4.4. In this case, some of the products will be marked with ascending orbit direction, and others with descending.

We created an algorithm to account for these cases as well. To summarise this approach, we find whether the satellite started sensing in ascending or descending orbit direction. Then we spatially sort all product marked as having this orbit direction. However, when taking a closer look at the metadata, we made a discovery that rendered this approach useless.

While we assumed the switching point between ascending and descending to be at the point closest to the poles, this was not the case. Instead, it switches at a point at some distance to the poles. As seen in Figure 4.5 the switch takes place north of Canada, more than 2000 km away from the northernmost point, just north of Greenland. This makes the approach of calculating the starting point from the latitudinal coordinate practically impossible, without finding a general rule for the location of the switching point.

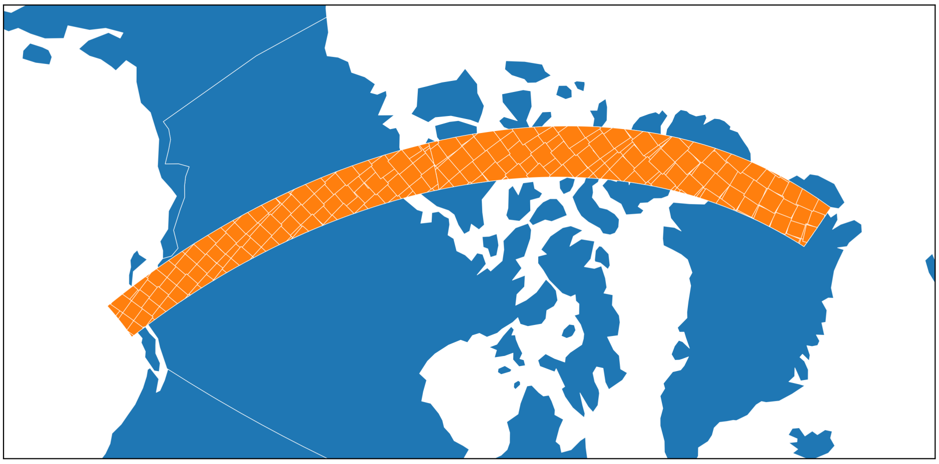


Figure 4.4: Example of a datatake which is crossing over the North Pole. In this case the starting point of the datatake cannot be deduced from the latitudinal coordinate. To accentuate the effect, the data is projected in the *August* map projection.

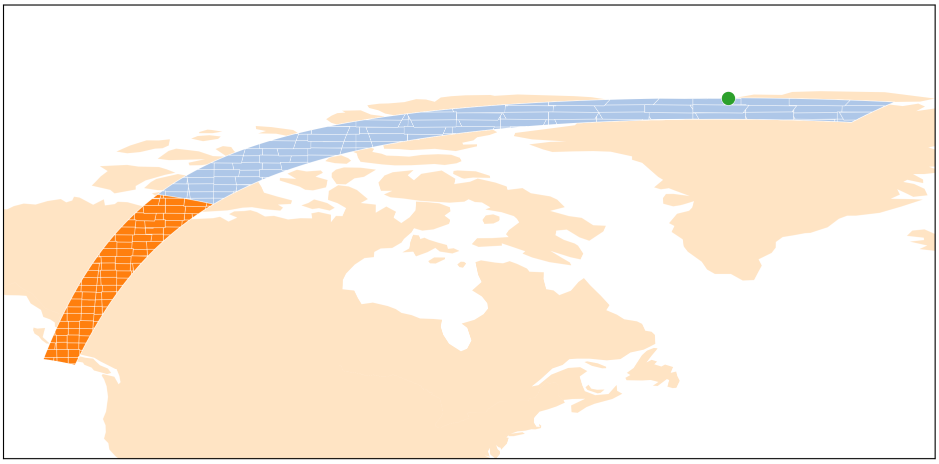


Figure 4.5: Example of a datatake containing both products sensed during ascension (blue) and products sensed during descension (orange). Curiously enough, the switching point between the two is not at the northernmost point (green dot).

We have speculated of why the switching point is not the nearest point to the poles. One article we found mentions a point that can explain this behaviour [41]. They observed the Sentinel-2 satellites view zenith angle to deviate from *nadir* (orthogonal to the earth's surface) at up to 11.93°. While this angle is not large enough to cause the shift of 2000km, this could be one of several reasons for the anomaly. This was way outside the scope of this project and was not investigated any further.

Ordering Products by Longitudinal Coordinates

Since the approach of sorting on the latitudinal coordinates did not lead all the way, we had to look at the problem from a different perspective. Figure 4.6 does just that. It shows the Earth from the point of view directly above the north pole. The Sentinel-2's relative orbits are projected onto it. From this point of view, the Earth is rotating anti-clockwise. As an outcome, the satellite orbits will always move clockwise. This observation yields the following takeaway:

☞ *The first image that was sensed in a datatake will be the image located furthest in the anti-clockwise direction along the datatake.*

Generally, this will be the point which has the largest longitudinal value. However, there is an anomaly when the datatake crosses the 180th meridian (or anti-meridian), as illustrated in Figure 4.7. This is where longitude values change from +180° to -180°. This has to be accounted for as the datatakes can cross this meridian. In these cases, the starting point will be on the negative side and thus have the smallest longitudinal value.

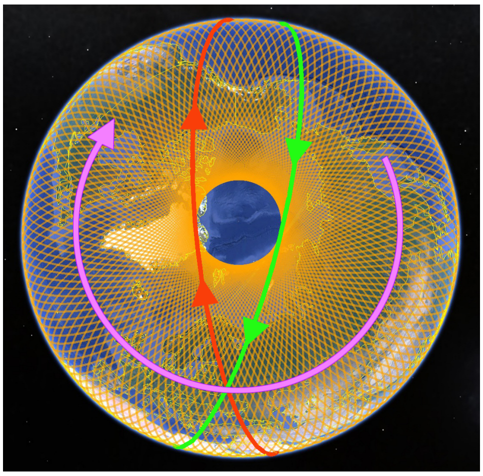


Figure 4.6: Illustration of the East-to-West direction of the Sentinel-2 orbits. The orbits are constantly clockwise (violet arrow) when looking at the earth from above the north pole.

To solve this anomaly we exploited the fact that the length of the datatakes will be up to, but no more than, 15.000 km [42]. This is great news to us because it means that the datatake will never be longer than half the circumference of the

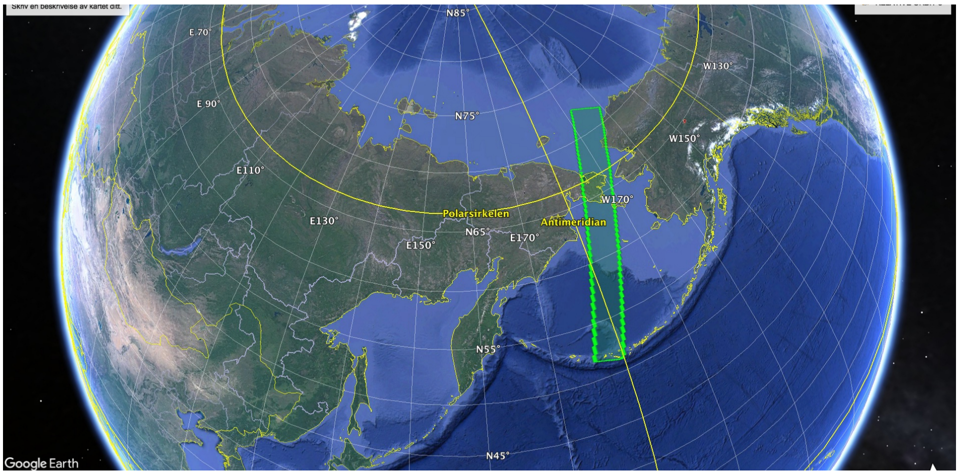


Figure 4.7: Example of a datatake crossing the 180th meridian. In this scenario the image that was sensed first will have the lowest longitudinal value. In all other cases the first image has the largest longitudinal value.

earth[1]. Transforming this observation into spherical geometry, this tells us that the angle between the start and the end is never larger than 180°. This yields the second important takeaway:

☞ *A datatake will always follow the shortest geodesic between the endpoints.*

Combining the two takeaways found, we can define the algorithm:

1. Order the image footprints by the longitudinal value, *X*, then pick out the two products having the largest and the smallest values for *X*. Keep in mind that at this point we do not know which of these points that are the start and end of the datatake.
2. Calculate the difference in *X* value between the two, *a_normal*. This is the longitudinal angle between them.
3. Take the smallest value for *X* and add 360° to it. Now find the difference between this adjusted *X* and the originally largest X, *a_adjusted*.
4. Compare the two resulting angles.
    - If *a_normal* is the shortest, it means the datatake is not crossing the 180th meridian. **The image having the *largest* value for *X* is the starting point.**

[1] The circumference of the earth is 40.075 km at the equator.

- If *a_adjusted* is the shortest, it means that the datatake is crossing the 180th meridian. **The image having the *originally smallest* value for *X* is the starting point.**

One may look at the adjustment made by adding 360° to the smallest longitudinal value as a way of avoiding the "jump" from 180° to -180°. This correction normalises the values so that our criteria for choosing the image with the largest longitudinal value holds true.

We implemented this algorithm as a single-query SQL. This way we could utilise built-in geospatial functions from PostGIS, and at the same time perform filtering on the AIS data.

However, the solution is a vast simplification of the real world and yields only a rough estimation of the real image timestamp. There are various reasons why this is not an optimal solution, which we will not discuss further. Nevertheless, this process gave us significant takeaways and techniques that were used in the approach we, in the end, decided to use.

4.1.2 Interpolation

As mentioned, the approach of extrapolation was researched under the assumption of not having anything besides the what is provided by the Sentinel-2 metadata, for the estimation of the image timestamp. However, it turns out that ESA has published predetermined acquisition plans for the pair of satellites [43].

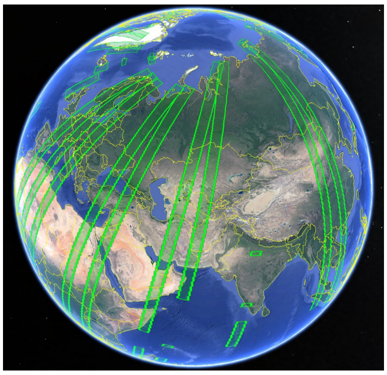


Figure 4.8: Visualisation of a priori acquisition plans for datatakes captured by the two Sentinel-2 satellites. The plans are for a time interval of 24 hours.

As seen from Figure 4.8, these contains some additional parameters, not present in the image metadata. These are:

1. Timestamp for the datatake sensing *stop*, as well as the start.
2. A polygon defining the total spatial extent of the datatake.

Having both the start and stop timestamps, as well as the spatial extent, we have the possibility of estimating the image timestamp using inter-

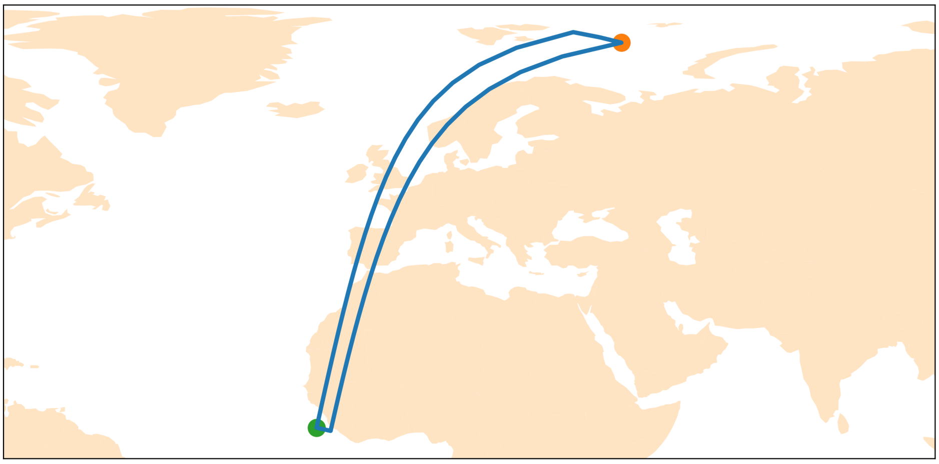


Figure 4.9: Example of the two endpoints calculated using the method of ordering by longitudinal coordinate. As the figure illustrates, in some cases these points will represent the diagonal of the acquisition. Using these points when interpolating will be inaccurate.

polation, as illustrated in Figure 4.10. This requires that we can extract the correct start and stop coordinates from the polygon. The polygon consists of a varying number of (latitude, longitude)-pair points, depending on the length of the datatake.

For this extraction, we could reuse the algorithm stated in the previous section. This would find the two points of the acquisitions that are at the extrema along the longitudinal direction. As illustrated in Figure 4.9, the two endpoints found using this method will in some cases represent the diagonal of the acquisition. Using these points when interpolating will be inaccurate. We need to make sure to extract points that are parallel to the acquisition, and more importantly, the direction of the satellite. This way we can exploit the ratio between the spatial distance and the temporal difference at the two endpoints.

It turns out that ESA has made things a little bit easier for us. We were informed by Kjell Eikland of Eikland Energy [44], that ESA is consistent when sorting the points that make up the acquisition polygons. This makes it possible to extract the appropriate points directly. The points are ordered in clock-wise direction circling the polygon, starting at the midpoint at the sensing-start end of the acquisitions, as illustrated in Figure 4.11.

An algorithm to interpolate an estimate of the image timestamp using the midpoints as the start and end points for the datatake was created. Pseudo-code for the algorithm is

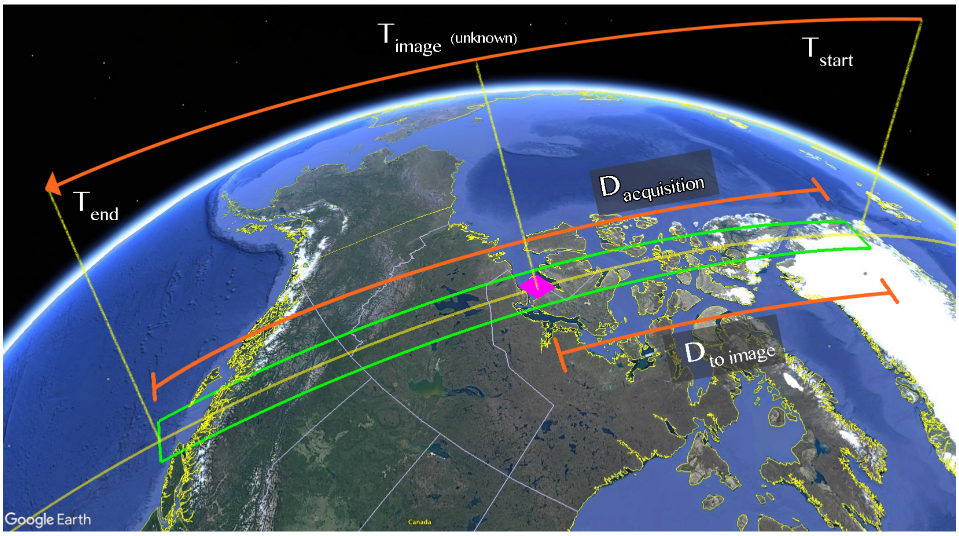


Figure 4.10: Illustration of using interpolation as a method to estimate timestamp at the image sensing time. The Sentinel-2 Acquisition Plans provides a-priori estimates of the datatakes. This includes geographical footprint and timestamp for the beginning and end of the acquisition. Having these, together with the known coordinate for the image, one can interpolate the image timestamp.

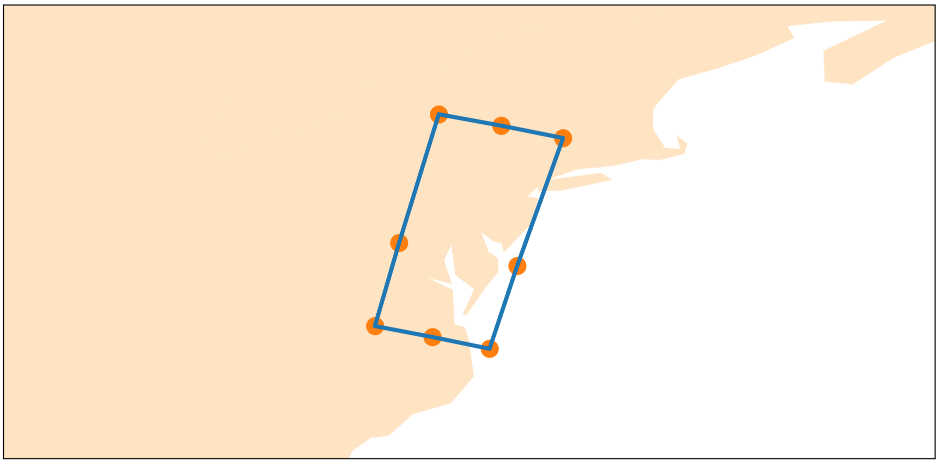


Figure 4.11: Illustration of the points defining an acquisition footprint. The points are consistently ordered in clockwise direction from the sensing-start centre point of the acquisition.

shown in Algorithms 1 and 2. The technical implementation of the approach is further discussed in Section 4.1.4.

The Acquisition plans are stored as KML files where each file covers a period of 10-15 days. As the attributes are *a priori*, they are not entirely equal to the attributes from the actual datatakes. From an email correspondence with `EOSupport@copernicus.esa.int`, asking about the expected accuracy of the acquisition plans compared to the actual datatakes, we got the following response: *The timestamps in the acquisition plans are very accurate compared to the actual datatakes; the mean difference, by daily products (ORB_PRE) and reference orbit file (REFORB), is of about 1/2 secs. for both satellites.* We assume this translates to that the timestamps for the acquisition plans have a mean difference of about half a second, compared to the actual timestamps. However, we have observed acquisitions with timestamps that differ with up to 15 seconds. Still, in most cases, we find this uncertainty to be generally low, compared to other sources of uncertainty in the system. This will also have a relatively small impact due to the limited spatial resolutions. Moreover, using these a priori attributes allows real-time estimation of the timestamps for eventual real-time processing of new images.

In the present study, the method using interpolation was chosen as the approach for estimating the true image timestamps. The technical implementation of it is described in detail in Section 4.1.4.

```
Algorithm 1: EstimateImageTimestamp(image, acquisition_plans)
Input: Image, with attributes datatakesensingstart and footprint
       Acquisition plans of datatakes, with attributes start_time, end_time and
       footprint
Result: Estimated image timestamp
acquisition $\longleftarrow \emptyset$
foreach plan $\in$ acquisition_plans do
    if plan.start_time is near image.datatakesensingstart then
        acquisition $\longleftarrow$ plan
    end
end
start_point, end_point $\longleftarrow$ extract central endpoints from acquisition.footprint
centerline $\longleftarrow$ EllipsoidalArcLine(start_point, end_point)
crosspoint $\longleftarrow$ point along centerline closest to Centroid(image.footprint)
estimated_image_timestamp $\longleftarrow$ InterpolateTimestamp(start_point, end_point,
                crosspoint, acquisition.start_time, acquisition.end_time)
return estimated_image_timestamp
```

```
Algorithm 2: InterpolateTimestamp(start_point, end_point, point, start_time,
             end_time )
Result: Timestamp
distance_endpoints $\longleftarrow$ EllipsoidalArcDistance(start_point, end_point)
distance_to_point $\longleftarrow$ EllipsoidalArcDistance(start_point, point)
endpoints_time_difference $\longleftarrow$ TimeDifference(start_time, end_time)
distance_ratio $\longleftarrow$ distance_to_point / distance_endpoints
timestamp $\longleftarrow$ start_time + endpoints_time_difference $*$ distance_ratio
return timestamp
```

4.1.3 Orbit Model

When investigating the timestamp issue, we got in contact with André Lima, a Post-Doctoral Researcher at University of Maryland [45]. We stumbled over him on a forum, seemingly also exploring solutions for the same problem. He had started to implement a solution using a more advanced approach. Using a physics modelling software fed with satellite ephemerides, he simulates the Sentinel-2 orbits in four dimensions (x, y, z, time) with a temporal resolution of one second. He then maps the simulated orbits to the non-temporal full cycle relative orbits provided by ESA [46]. The image metadata provides the name of the relative orbit the satellite had while sensing, which in turn maps to his simulated orbit. The orbit simulations provide the timestamp at the time the satellite was directly above it the image of interest.

However, the Sentinel-2 orbit simulation needs to be modelled within a specific time interval of interest. For this approach to be working in this project, we would have to simulate an extensive time range, optimally up to one year. It would require much time to learn the simulation software, which we do not have at our hands at the moment. Furthermore, as it depends on satellite ephemerides, it will not work with real-time data image. This should be considered for future work, as it looks very promising concerning accurate estimation.

4.1.4 Technical Implementation of Timestamp Correction

In this section, we first present the necessary steps for facilitating and preparing the acquisition plans in which the interpolation approach is based on, before presenting the technical implementation of our algorithm.

Preparing the Data

As mentioned in Section 3.1, we need to bear in mind that we have huge volumes of data, so when implementing solutions, we need to focus on efficiency and runtime for the algorithms, in addition to precision. To facilitate this, we wanted to store most of the data in our PostGIS database to efficiently be able to perform spatial queries, as well as utilising the wide range of spatial functions PostGIS provides. As mentioned, the Acquisition Plans are published by ESA in KML format. Hence, they need to be preprocessed and converted to SQL for insertion into the database.

To process and convert the KML files, the GDAL/OGR command line tool *ogr2ogr* was used. The tool lets one convert spatial features data between multiple file formats, and perform various operations during the process such as re-projecting, and attribute selections and reformatting.

The whole process of converting and inserting was broken down into several steps. The different steps required extensive data cleansing to remove dirty data, and duplicates that occurred during conversions.

Using ogr2ogr, the KML files were first converted to shape-files. Each file resulted in several shape-files. However, only one of these resulting shape-files contained the data of interest. This is the *NOMINAL* layer, which represents the image products available

Table 4.1: Table showing the relevant parameters in the acquisition database table

Attribute	**Description**
name	Acquisition name
start_time	Sensing start time of acquisition
end_time	Sensing stop time of acquisition
duration	Duration of acquisition, in milliseconds
absolute_orbit	Absolute orbit of acquisition
relative_orbit	Relative orbit of acquisition
wkb_geometry	WKB formatted geometric footprint of acquisition
satellite	Satellite, either 2A or 2B, of acquisition

for public use. The next step was thus to convert these NOMINAL shape-files to SQL and insert them into the acquisition table in the database.

In the conversion from KML to Shape, some of the attribute names were altered, and also, a lot of empty attributes were added to each file. To compensate for this, when further converting to SQL, ogr2ogr was used with an optional SQL statement. This statement let us select the attributes that are relevant to us, as well as rename the affected attribute names to something more describing. We also used it to cast the different attribute types, such as the timestamps of correct DateTime format. Also, as we wanted a single acquisition table in the database, but have data from two satellites, 2A and 2B which acquisitions overlap in time, we added an attribute that distinguish the two satellites acquisitions from each other. For some reason, probably as a result of the conversion from KML to shape-file, the resulting acquisition table contained a lot of duplicated data. As the last step of data cleansing, an SQL statement was created to remove the duplicates in the table.

The result is an acquisition table in the database containing the attributes shown in Table 4.1.

The image metadata can efficiently be queried and downloaded from ESA using different APIs, and thus we decided against storing it in the database. In our solution, we query the metadata and store only in memory. We structure it in Geopandas dataframes and extract the appropriate attributes to be used as input in the SQL-queries described below.

Implementing in PostGIS

Since we store the acquisition table in PostGIS, the algorithm for interpolating timestamp is implemented in SQL using spatial functions. The actual SQL-code can be found in Appendix B.

As shown in Algorithm 1, in the previous section, the algorithm starts by querying the acquisition table for the product acquisition datatake and then selecting the relevant attributes from the table. The right acquisition is found by querying the table for rows where the image datatake sensing start time is between the start time and end time of the acquisition, as well as filtering the table on the correct satellite. For the time query, a margin of 15 seconds is added as we remember that we have observed acquisition timestamps with differences up to 15 seconds with respect to the actual datatake. In addition to the relevant attributes, we add a new attribute for the number of points in the footprint polygon, using the PostGIS function *ST_NPoints*. Note that we subtract this parameter with one, as the first point of the polygon is counted twice.

The next step is extracting the middle start and end points, taking advantage of the standardised point order. The start point is the first point in the polygon, which we extract using the function *ST_PointN*. The end middle point is a little less trivial, having the index equal to the number of points divided by two and rounded negatively to the nearest integer, before adding a value of 1 (we need to add this since *ST_PointN* uses 1-based indexing). This is obtained by doing a right arithmetic bit shift of 1, and then adding the value of 1 due to the indexing. Thus, we have the correct start and end points of the acquisition, and this way of extracting is general for all datatake situations.

This is followed by calculating the, as we call it, crosspoint. The crosspoint is the point, on the ellipsoidal arc line between the middle start and end point, that is closest to the centroid of the image in question. Why we are interested in this point can be explained by how the satellite acquires imagery. As mentioned in Section 2.3.2, the MSI uses a push-broom concept, capturing line by line along the width of the datatake. We remember that one can find the shortest path from a point to a line by measuring the length of the line from the point that is orthogonal to the given line. By our definition, this is how we calculate the crosspoint, and thus it is captured at the same time instant as the image centroid.

Using the crosspoint instead of the image centroid, we will, as one will see in the next step, calculate the length of two arc lines that are parallel to each other. This will result in a more exact ratio when interpolating, than calculating the distance to the image centroid.

The calculation of the crosspoint is done using multiple PostGIS functions. First *ST_MakeLine* creates the ellipsoidal arc line between the start and end point. Furthermore, the centroid of the image is calculated using *ST_Centroid* and is given the SRID 4326, as all geometry is defined in WGS84. Instead of using WGS84, we could have used other, more locally accurate ellipsoid or geoid models. However, WGS84 has qualities such as being general on a worldwide basis, which is important for our solution to be general. In addition, the surfaces of interest in this project are the oceans, which are flat, making the ellipsoidal reference surface provided by WGS84 a good approximation. The result of these two calculations is further used in *ST_ClosestPoint*, to find the crosspoint.

Now that the crosspoint is defined, the actual lengths to be used in the interpolation can be calculated. This is done using *ST_Distance* with the appropriate points cast from geometry type to geography type. Using geography objects, *ST_Distance* returns the shortest geodesic distance between the two points, and in meters as well, which is precisely what we need.

The last step is the actual interpolation. First, the distance from start point to crosspoint is divided by the length of the acquisition, to achieve the ratio between them. The new timestamp is then interpolated by adding this ratio to the acquisition start time, along with the sensing duration for the acquisition.

The result is the estimated timestamp for the image centroid is returned as a date time object.

4.2 Big Data Management

From the prerequisites defined in Section 3.1.1, we need to be conscious of the four V's of Big Data Management. Since we are looking at historical data, some of these V's are, although present, not too prominent. The *Velocity* of the data will only be relevant when looking at real-time data, which will be a challenge if one, for instance, would extend this research to detecting ships in real-time images. We have to deal with both *Variety* in the data, as we are integrating data from difference completely different sources and formats. Also, there is much variation within the images which, if not considered, will alter the machine learnings performance. A critical element here is the presence of clouds in the images. The *Veracity* of the data is also a significant concern, be it noisy images, or even fake AIS-positions.

In the present project, there is primarily the *Volume* aspect that is the most prominent. Luckily, the enormous amounts of Sentinel-2 images are managed by both ESA and AWS, such that we can quickly find what we are after. On the other hand, the *~6.5 billion* AIS-messages are not. It was our job to develop a good strategy for managing the enormous amounts of data.

4.2.1 Scalability

There are two separate ways to think about scaling: scaling up so that a single machine can store more data, and scaling out so that data can be stored across multiple machines.

For this project, it was only relevant to consider the single machine scalability, as it was desired to exploit the local machine provided to us by the faculty. Relevant specifications include 4.20GHz CPU with four cores, a 500GB solid-state drive (SSD), and 32GB System Memory.

A common problem with scaling database performance on a single machine is that, eventually, our entire dataset will not fit in memory, which is why we will need to write our data and indexes to disk. This is a common problem for relational databases. Under most relational databases, a table is stored as a collection of fixed-size pages of data (for PostgreSQL 8KB pages), on which the system builds data structures (such as R-trees, for spatial data) on top of to index the data. With an index, a query can quickly find a row with a specified value without scanning the entire table.

Now, if the working set of data and indexes is small, we can keep it all in memory. However, as our AIS table is so large that we cannot fit all pages of our R-tree in memory, then updating a random part of the tree can involve significant disk I/O as we read pages from disk into memory, modify in memory, and then write back out to disk. Also, a relational database like PostgreSQL keeps a B-tree, or other data structure, for each table index, for values in that index to be found efficiently. Hence, the problem compounds as one indexes more columns. To put this in context, our AIS tables spatial R-tree index was at 20 GB when having 230GB of data in the table. To reach our goal of having a full year worth of AIS data in the table, this means the spatial index will grow up to 700GB. This, of course, won't fit into memory all at once.

The increase in the cost of swapping in and out of memory can be seen in the performance graph in Figure 4.12, where we observe how insertion throughput decrease with table size. The variance present is depending on whether requests hit in memory or require, potentially multiple, fetches from disk.

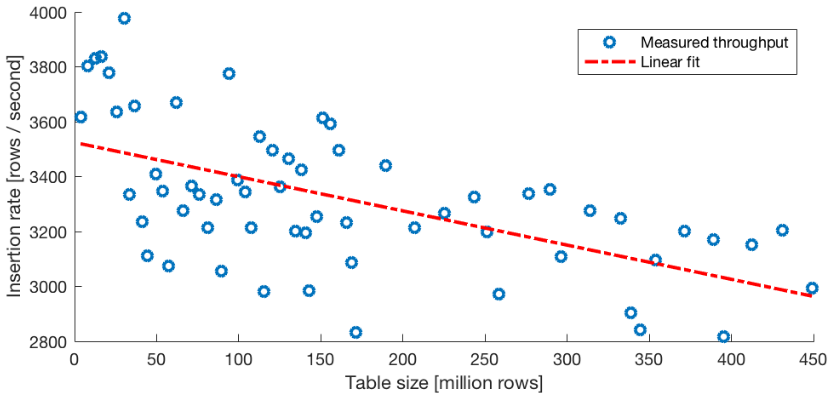


Figure 4.12: Insertion throughput as a function of our database table size. The insertion rate also includes decoding the AIS-messages from raw NMEA-format, adding additional overhead to the execution time. This overhead is constant and does not influence the change in insertion rate. The variance is depending on whether requests hit in memory or require, potentially multiple, fetches from disk.

As described in Section 3.2.1, we are doing *UPSERTS* when inserting new AIS messages, in a manner to remove duplicates[2]. This means that for each new message, the database scans both the time-index and the identifying vessel MMSI-index to detect pairs that are identical to the new message to be inserted. As a result, the database is required to perform additional I/O from disk, which makes scaling up very difficult.

As it turns out, the answer to this challenge lies in the nature of the data itself.

4.2.2 Scaling Up Time Series Data

As opposed to, say, bank transactions where database updates are applied to accounts randomly distributed in the system, the AIS messages are (partially) identified by a timestamp. Also, as a bonus, we can feed the data into the database in somewhat chronological order with respect to this timestamp. This is due to the fact that the raw data provided to us contains messages ordered by the timestamp they were received, which will moderately coincide with the timestamp of transmission.

This means, if data is sorted by time, we are always writing towards the end of the database table. Organizing the table by time will also keep the actual working set of database pages small, and maintain them in memory. So in the case of doing UPSERTs, we

[2] The UPSERT also guarantees an atomic INSERT or UPDATE outcome. This makes for more robustness when running the inserts on multiple threads.

need to compare the new message only with inserted messages that are still cached in memory, as we know these are the only ones that have similar timestamps.

While it seems like this is possible to obtain only by indexing the table on time, this is usually not enough, as in our case. We also need to index the table on the identifying vessel MMSI, which makes the database perform random inserts into a B-tree for that index.

Doing some research on the subject, we came upon an extension to PostgreSQL called TimescaleDB [47]. It solves this challenge by introducing what they call a *hypertable*. The hypertable acts exactly like a traditional PostgreSQL table, but under the hood consist of many smaller tables. These tables, or chunks, each contains data within a specific time interval. Non-temporal indexes are further only applied locally per time-chunk. This restricts indexes to only grow with the size of each chunk, not the entire hypertable. As our inserts are mainly to the more recent time interval, the time chunks containing that interval remains in memory, avoiding expensive swaps to the disk, as Figure 4.13 illustrates.

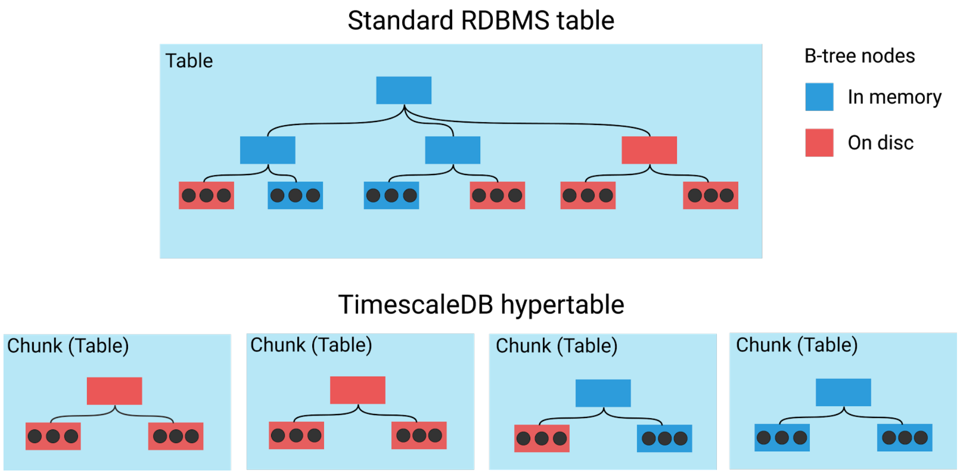


Figure 4.13: Illustration of how TimescaleDB stores each time-chunk in an internal database table, so other, non-temporal indexes only grow with the size of each chunk, not the entire hypertable. This allows keeping the most recent data and indexes in memory, making moderately ordered UPSERTs more efficient [48].

In the case of the AIS messages, this time-chunking allows us to split the data according to two dimensions; the time interval and the primary key (vessel MMSI). The query planner can then immediately tell which chunk to which the query applies to. So if we size these chunks correctly, we can fit the latest tables (chunks) and all their indexes entirely in memory.

TimescaleDB greatly supports optimising the temporal chunking of the data by dividing the chunks into adaptive intervals. This lets us define a fixed time interval, for example, one day, which is further adapted based on the changes in data volumes.

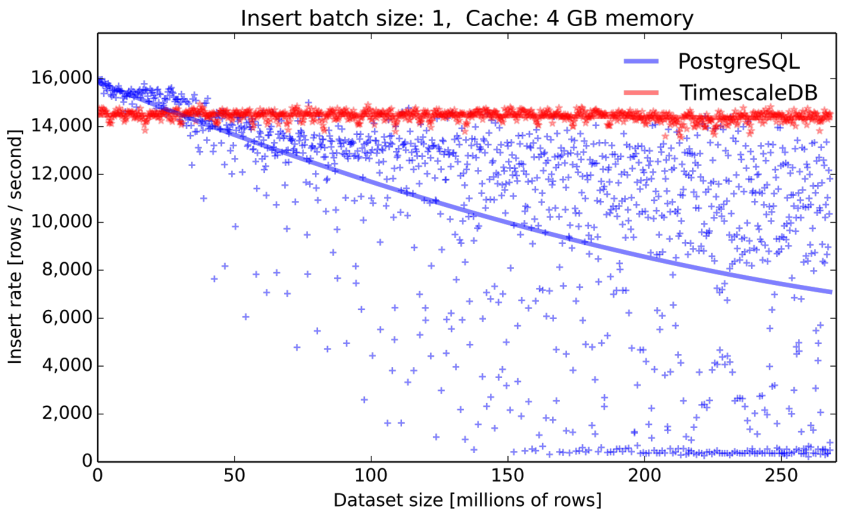


Figure 4.14: Insertion throughput of TimescaleDB vs. standard PostgreSQL. TimescaleDB maintains a constant insert rate, and a very low variance. Contrary to standard PostgreSQL it is independent of the dataset size [48].

Since we had already inserted all the data we had the capacity to store on the available 500GB SSD, we did not perform any additional insertions into the hypertable. As previously stated, the ultimate goal is to make a training dataset based upon one year worth of AIS data, i.e. 18 times more data than what was used for this analysis. Hence, the implementation of TimescaleDB will greatly cut down on insertion time, as it will keep the throughput constant as the dataset size increases. Figure 4.14 shows this behaviour in practice.

4.2.3 Enhancing Queries

Now let us look at the most important query of this project. That is: *Find all ship positions within an image at the image timestamp.*

Code Listing 4.1: Find all ships within image and 20 seconds time interval

```sql
WITH image AS (
  SELECT ST_GeomFromText([WKT String]) AS footprint
  ,       '2017-09-29 11:16:20.0' AS timestamp_utc
)
SELECT *
FROM ais_position, image
WHERE ST_Within(ais_position.geom, image.footprint)
AND ais_position.timestamp_utc
  BETWEEN  image.timestamp_utc - interval '10 seconds'
      AND  image.timestamp_utc + interval '10 seconds'
;
```

Code Listing 4.1 shows the most natural way of structuring the query for the given filter criteria. The above query shows an execution time of 8442 milliseconds. That is much better than the query filtering only spatially, executing at 4 minutes. Nevertheless, we sought an even faster execution time. Let us have a look at what is going on under the hood. Using PostgreSQL's *explain analyze*-function [49], the database will devise a query plan, execute the query, and then return the algorithm explained step by step. It is a little extensive to those unfamiliar with it. Hence we have cleaned it up a bit to highlight the essential. We substitute the criteria for *timestamp between timestamps* as [*timestamp_between*].

Code Listing 4.2: Query plan after execution

```
1  Nested Loop  (rows=281)
2    CTE image
3      ->  Result (actual rows=1)
4    ->  CTE Scan on image (actual rows=1)
5    ->  Bitmap Heap Scan on ais (actual rows=281)
6      Recheck Cond: ((image.footprint ~ geom)
7        AND ([timestamp_between]))
8      Filter: _st_contains(image.footprint, geom)
9      Rows Removed by Filter: 21
10     Heap Blocks: exact=204
11     ->  BitmapAnd (actual rows=0)
12       ->  Bitmap Index Scan on idx_ais_geom (
13               actual time=6995.158 rows=14948509)
14             Index Cond: (image.footprint ~ geom)
15       ->  Bitmap Index Scan on ix_ais_timestamp (
16               actual time=0.909 rows=4889)
17             Index Cond: ([timestamp_between])
18 Planning time: 0.152 ms
19 Execution time: 8441.691 ms
```

Although it may seem overwhelming, one thing stands out in the query plan in Code Listing 4.2, and that is the number of rows scanned during the *Bitmap Index Scan* of the spatial filtering condition (line 12-13). This reads to be ~15 million rows. This is quite a lot of rows to run the spatial condition on, which in this case shows to require 7000 milliseconds to execute. On top of it all, it even turns out that this number will strictly increase as more data is inserted into the database. By utilising our knowledge about the nature of the data itself, we can find out why, and perform some crucial optimisations.

Consider the following task: We want to retrieve all ship messages of interest from our AIS dataset. We have two criteria for the result:

1. The messages timestamp has to be within a specified *time interval*, in this task one hour
2. The messages position has to be within a *spatial boundary*, in this task the port of Trondheim, Norway

We are solving this task based on two scenarios.

Scenario 1

Our database contains one hour worth of global AIS-messages.

Total messages globally: 882.153

Messages within spatial boundary: 3.555

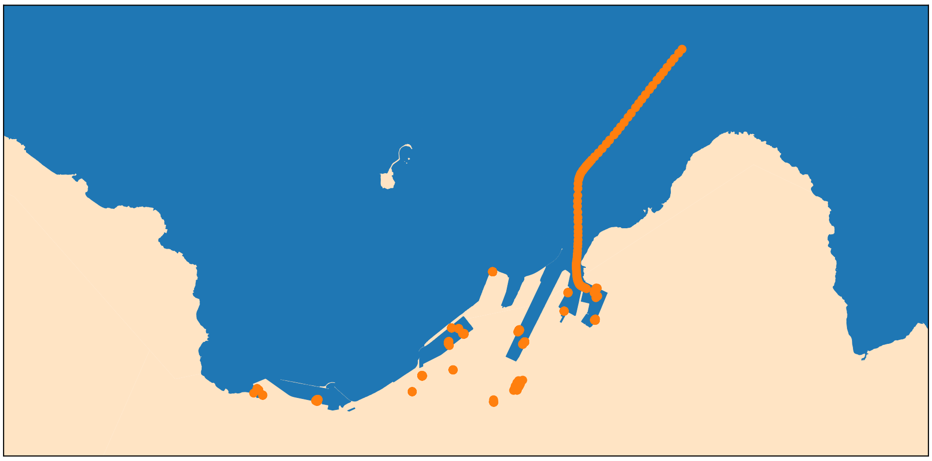


Figure 4.15: Visualisation of accumulated ship positions (orange dots) for one hour, outside the port of Trondheim, Norway. 3.555 AIS messages from 24 unique vessels.

In this scenario, it makes sense to filter in the following order

1. perform spatial filtering for messages within Trondheim (a scan of 3.555 rows)
2. perform temporal filtering for messages within the one hour (a scan of the resulting 3.555 rows)

With this approach, we leave out ship messages from the whole globe, except for the small portion of them being within our specified coordinates. The resulting accumulated ship positions are illustrated in Figure 4.15. This way we filter out 99,6% of all the messages already in the first step! Looking at Figure 4.16 it is intuitive to the human brain to "zoom in" on the area of interest.

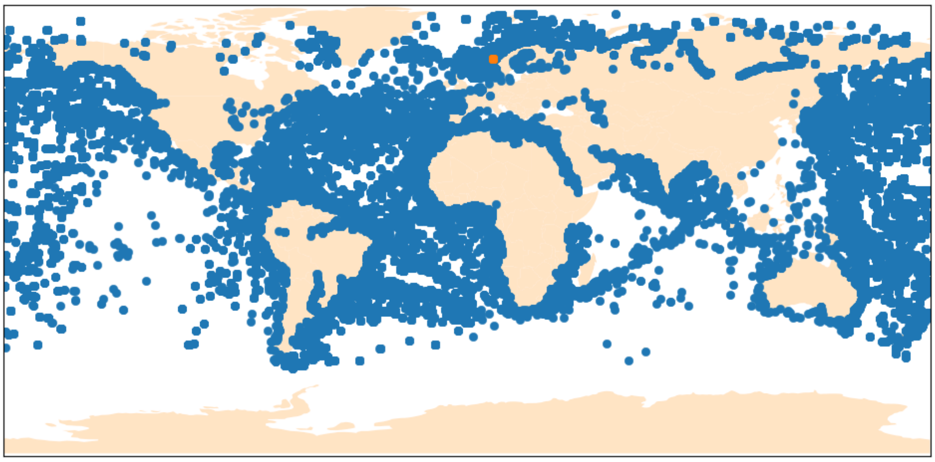


Figure 4.16: Visualisation of accumulated ship positions (blue dots) for one hour globally (882 thousand AIS messages from 22 thousand unique vessels). The orange dot represents the messages of interest, at the port of Trondheim.

If we did it the other way around the database would first scan rows that match the criteria of being within the hour (which in reality is the whole dataset of 882 thousand rows). Then it would further scan the 3.5 thousand rows that match the spatial criteria as well. This makes the query run more the 1000 times slower! This surely must be the recipe for success.

Scenario 2

Our database contains 20 days worth of global AIS-messages.

Total messages globally:	400.000.000
Messages within spatial boundary:	1.567.095

By filtering spatially first, following the same strategy as considered optimal in the previous scenario, the database would end up having to scan ~1.6 million rows that match the spatial criteria. This result in execution time of 2 minutes. One could settle with the thought of this being the nature of having a big dataset, and accept that the execution time would have to increase. This was not an option for us, as we eventually wanted to work with a *twenty times* larger dataset. We took a step back and tried to free our minds from the intuitive solution, by considering flipping the order of the filtering.

If we do the temporal filtering first, the database would only need to scan 882 thousand rows. This is the same number as the total global count in scenario 1, which sounded huge in that context. However, it turns out that regardless of how much more data we put into the database, this number never increase due to the fact that the timestamp of the new data is strictly increasing. This means a given time interval will not accumulate more data when more data is loaded into the database. On the other hand, the new data will make the whole dataset strictly denser, spatially, across the whole globe. This means that the port of Trondheim will gradually fill up more and more as we put more data into the database, as illustrated in Figure 4.17. However, the one hour we are interested in has already capped at 882 thousand, and will henceforth always be the best first choice for filtering.

In the scenario where the total number of ship messages within Trondheim is higher than the global count for one hour, the result of choosing the temporal filter first provides a 180 times speed-up in terms of execution time.

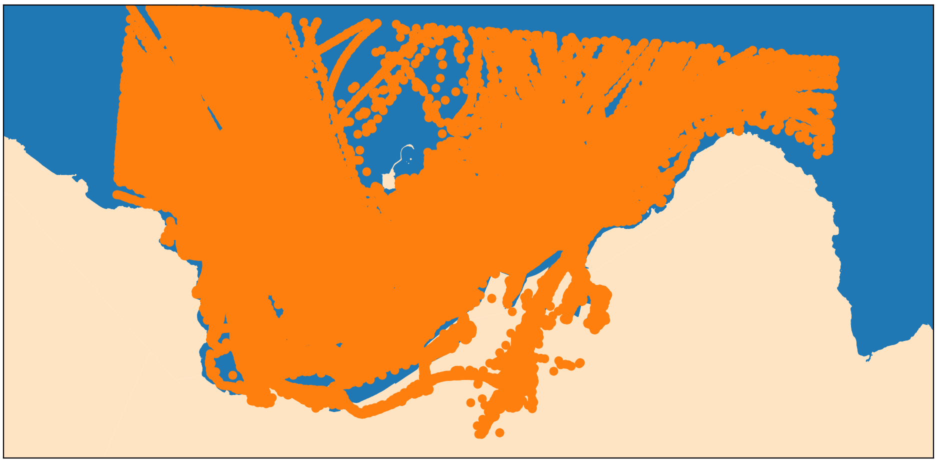


Figure 4.17: Visualisation of accumulated ship positions (orange dots) for 20 days, outside the port of Trondheim, Norway. 1.7 million AIS messages from 158 unique vessels, which is more than twice as many as the global count showed in Figure 4.16! A further note is that the restricted extent of the ship messages is due to the maximum receiving distance of the AIS shore receiver, not a result of the spatial or temporal filtering.

In reality, running Query 4.1 on the port of Trondheim, having 20 days worth of data in the database, it performs a Bitmap Heap Scan on both criteria at once, bringing the execution time down to 0.6 milliseconds. This is a staggering 200.000 times quicker than the aforementioned worst approach of first filtering spatially.

Sadly, when filtering on a bigger spatial area, in our case a $10,000 km^2$ satellite image, letting the database do a scan on both criteria simultaneously results in poor performance. This is due to the vast imbalance between the rows scanned by the two criteria. The spatial index on the AIS coordinates is simply impractically large.

By forcing the temporal criteria to be run first, the simple query executes 400 times faster. In Code Listing 4.3 we show the approach used to force the database to execute the temporal filtering before the spatial filtering. It is done by utilising the PostgreSQL function *WITH* that provides a way to write *Common Table Expressions* (CTE), that roughly put defines a temporary table that exists only locally within the query.

Furthermore, by keeping this in mind on the more complicated queries, described in Section 4.4.1, we brought the execution time down from 4 minutes to 4 milliseconds. To put this in perspective; we want to execute this query on at least 20.000 satellite images. That means *we have brought the total execution time down from 60 days to 80 minutes.* In Code Listing 4.3 we show the approach used to force the database to execute the temporal filtering before the spatial filtering.

Code Listing 4.3: Optimisation of the query for ship within image and 20 sec time interval

```
WITH image AS (
  SELECT ST_GeomFromText([WKT String]) AS footprint
  ,      '2017-09-29 11:16:20.0' AS timestamp_utc
), ships_within_time_slot AS (
  SELECT *
  FROM ais_pos, image
  WHERE ais_pos.timestamp_utc
  BETWEEN image.timestamp_utc - interval '10 seconds'
      AND image.timestamp_utc + interval '10 seconds'
)
SELECT count(*)
FROM ships_within_time_slot, tile
WHERE ST_Within(
        ships_within_time_slot.geom
      , image.footprint)
;
```

4.3 Image Selection Optimisation

The Sentinel-2 satellites generate 4 Terabytes of data each day [33]. Processing large amounts of image data is a computationally demanding task, especially when computational resources are limited. However, for most applications, most images in the archive are redundant as it does not contain relevant information. Identifying common characteristics for the redundant images, one can reduce the number of images to be processed, or even downloaded, substantially. At the same time, we are interested in finding images scattered around the globe, for the resulting training dataset to allow the Neural Network to generalise well.

For our use case, the obviously redundant images are those lacking oceans. While this narrows the search down substantially, it is not sufficient, as the earth's surface consists of 70% water. On the other hand, the Sentinel-2 satellites, while having 'global coverage', does not capture every square meter of the globe, actually far from it. Combining these two characteristics with the fact that not all marine area contains sufficient ship traffic, we managed to narrow the search down to below 0.6% of the earth surface[3] In this section we present the general approach we used to select areas of interest based on analysis of the two datasets.

4.3.1 Grid System for Image Footprints

The metadata supplied with the Sentinel-2 Images includes the geographical footprint of the images, which we wanted to use for selecting regions of interest around the globe. ESA made this task a lot easier by locking these footprints to an *Earth Centred, Earth Fixed* (ECEF) reference system. The specific system is called *Military Grid Reference System* (MGRS) and consists of 56.686 $100km * 100km$ cells locked on the earth's surface [50]. It is derived from the Universal Transverse Mercator (UTM) grid system, together with the Universal Polar Stereographic (UPS) grid system. The structure of the grid is shown in Figure 4.18.

[3] The 0.6% is an approximation derived from the ratio between the MGRS-tiles found to be of relevance and the total number of MGRS-tiles globally. Due to overlap between the MGRS-tiles, the percentage does not correctly reflect geographical area, which will be significantly less.

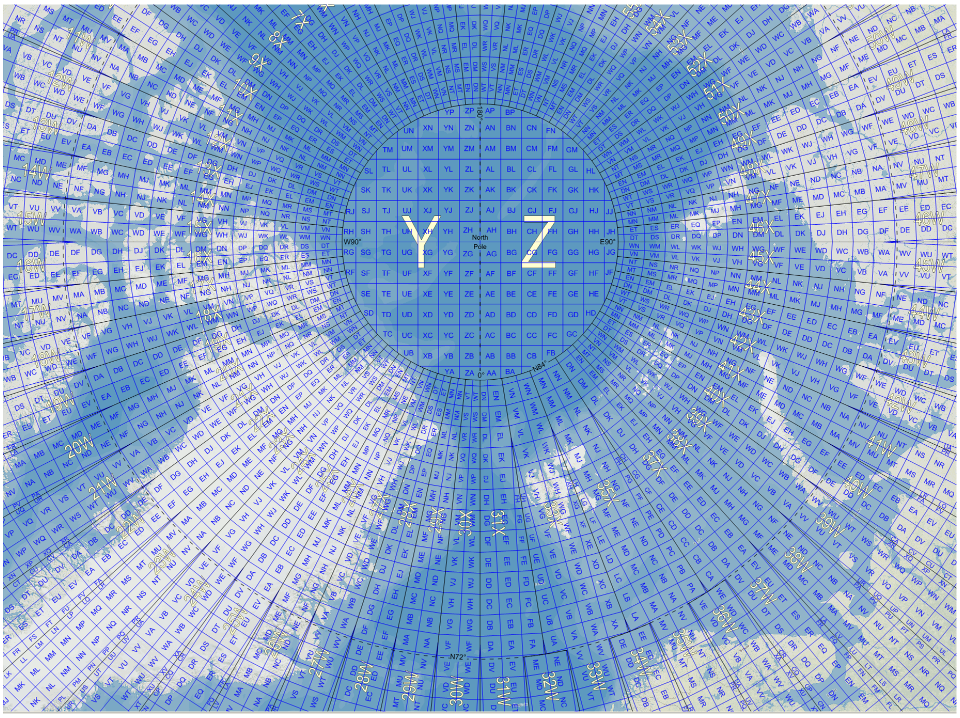


Figure 4.18: Illustration of the Military Grid Reference System, the geocoordinate standard used by NATO militaries for locating points on the earth. The 100km grid cells coincide with the Sentinel-2 images geographical footprints [51].

The MGRS-tiles are provided by ESA as a KML-file. This was further converted into a shapefile before inserted into PostGIS table. All of this was accomplished using ogr2ogr in a similar matter as the process of data cleansing of the acquisition plans, as explained in Section 4.1.4.

What this essentially offers is a way for us to analyse the image metadata before even downloading them. Furthermore, it allows us to select regions of interest based upon analysis in combination with the other datasets in our database. However, as it turns out, doing global spatial analysis is a very computational demanding task, especially when dealing with highly dense data, such as AIS messages.

4.3.2 Big Data Challenges in Image Selection

The main analysis that was crucial to perform was combining the AIS messages with the MGRS-tiles. Specifically, counting the density of ship messages within each tile. Having both datasets in a PostGIS database, the first thing that comes to mind is the following SQL query:

Code Listing 4.4: Calculate ship density within each MGRS tile

```sql
SELECT COUNT(*)
FROM ais_position
,    mgrs
WHERE ST_Within(ais_position.geom, mgrs.geom)
GROUP BY mgrs.name;
```

Query 4.4 can be interpreted the following way: *For each MGRS tile, count the number of AIS messages that is spatially within it.* This feels very natural and simple and gives us exactly the result that we want. For the small tests-table of AIS messages, consisting of 500 thousand rows, this works like a charm, finishing in under 1500 milliseconds. Not bad for a global analysis. Unfortunately, we would not be content until we had analysed the whole dataset of 500 million rows.

From our previous encounters with challenges regarding Big Data analysis, we suspected this to be a computationally heavy operation that would take a long time to execute. However, after it had run for one week straight, we terminated it and went back to the drawing board.

We *suspect* the reason for this running indefinitely is because of the *WHERE*-clause having to execute on every pair of MGRS-tiles and AIS messages. Not until this is finished can the *GROUP BY*-clause execute, and the counting begin. This means that the database has to keep the intermediate results in memory until the very end. As every AIS message is within an MGRS-tile, the database will need to store the whole AIS table in memory. However, we did not find any theory to support this, but apparently, we needed a different approach.

Taking a step back, we contemplated different approaches to the challenge. A fair amount of time went into the matter, and we eventually landed on what we still consider the best approach. Other approaches, like *Materialized Views*, were considered, but will not be further discussed.

4.3.3 Our Solution to the Big Data Challenges

The basis of the solution lies in avoiding to perform the spatial WITHIN-operation at query time. Since the resulting MGRS tile for each AIS message is constant, we needed only to perform the query once and store the result for later usage. We decided to perform it already at insertion time.

To make it harder for us, due to overlap between the MGRS-tiles, each AIS message can be contained by up to 7 MGRS-tiles[4].

We considered adding a column in the AIS table with type ARRAY [52]. Upon insertion, the database would find the names of all MGRS-tiles containing each AIS messages and store them in the array. While this accomplishes the goal, it feels wrong to add data to the table that is not directly related to it. And for the sake of later performing analysis on this data, the ARRAYs does not (yet) support element-wise indexing. [5]

What we ended up doing was creating a relation table between the two tables containing AIS messages, and MGRS. This contains foreign keys to the name-column in the MGRS table, and a composite foreign key to the (mmsi, timestamp)-column pair (i.e. the primary key of the AIS table). Then, upon insertion of the AIS messages, the database finds the MGRS-tile(s) containing each AIS message, and stores many-to-many relations between the two tables.

Quite some effort went into making the insertion work with the previously implemented batch-insertion function, as well as the UPSERT criteria. The result, a rather beautiful SQL code for performing this insertion, can be observed in Code Listing 4.5.

[4]The average number of MGRS-tiles that one AIS message is within, due to overlap, is 1.4 tiles

[5] The solution for storing MGRS names in an ARRAY-column for each AIS message was thoroughly tested. It worked out very well, allowing us to perform global analysis on the entire AIS table (400 million rows) in less than 7 minutes. However, the ARRAY-columns can only be indexed as a whole, which makes it non-optimal for targeted queries.

Code Listing 4.5: Insert AIS Messages and populate relation table

```sql
WITH inserted AS (
   INSERT INTO ais_position (
       timestamp_utc, mmsi, geom, accuracy,
       rate_of_turn, course, speed, ... )
   VALUES %s -- injected by batch-insert function
   ON CONFLICT (mmsi, timestamp)
   DO NOTHING
   RETURNING timestamp, mmsi, geom
)
INSERT INTO ais_mgrs_relation
   (timestamp_utc, mmsi, mgrs_name)
   SELECT inserted.timestamp_utc
   ,      inserted.mmsi
   ,      mgrs.name
   FROM inserted
   ,    mgrs
   WHERE ST_Within(inserted.geom, mgrs.geom)
```

Having the single pair relations in separate columns now allowed us to apply indexes on the timestamp, mmsi *and* the MGRS-name, making it very efficient to perform global analyses on the full dataset. For scalability, the relation table could even be inserted into a TimescaleDB hypertable, as mentioned in Section 4.2.2.

Now, counting the ship density for each MGRS-tile is done as simple as shown in Query 4.6, and the execution time is only a few minutes. This can further be filtered on time intervals as well, right from the relation table. By joining with the AIS table, one can even filter it on ship sizes, speed and so on.

Code Listing 4.6: Optimised calculation of ship density within each MGRS tile

```sql
SELECT count(*)
FROM ais_mgrs_relation
GROUP BY mgrs_name;
```

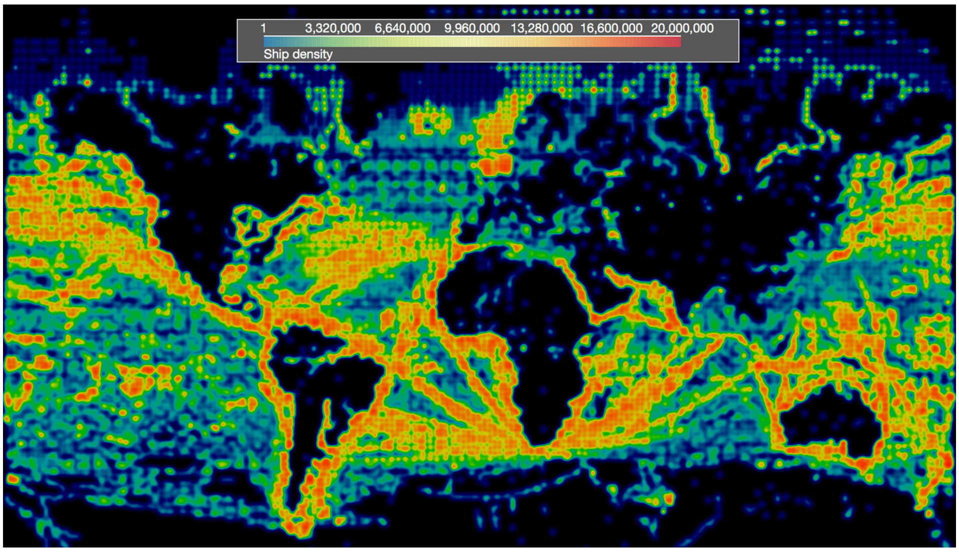


Figure 4.19: Heat map showing global ship density, created using our solution, for a period of 20 days (400 million AIS messages). We also created an interactive map, that can be studied in detail at `https://shipdensityheatmap.herokuapp.com`.

4.3.4 Global Density Analysis Utilising Our Solution

Having solved the challenges preventing us from performing global spatial analysis, it was now time to find a good approach to find MGRS-tiles of interest. As a reminder, *the MGRS-tiles are analogous with the Sentinel-2 image footprints.* As we were now able to achieve global results at an instant, we looked into ways of visualising the data.

In Figure 4.19 we have plotted the ship density of every MGRS-tile based on all 400 million AIS messages. The colour reflects the density of ships within each tile. The density varies from 0 (black) to 20 million (red) ship messages within the area.

As previously explained, the Sentinel-2 satellites do not capture images of the whole globe. As its primary application is at land, the marine areas it covers are primarily near the coastal boundary.

Having our database full of Sentinel-2 Acquisition Plans, the task of finding MGRS-tiles covered by satellite images were only an *ST_Intersect* away [53]. The resulting set of MGRS-tiles was converted back to KML-format for easy visualising using Google Earth, as seen in Figure 4.20.

Figure 4.21 shows the resulting heat map of joining the relevant MGRS-tiles with the result of the ship density analysis.

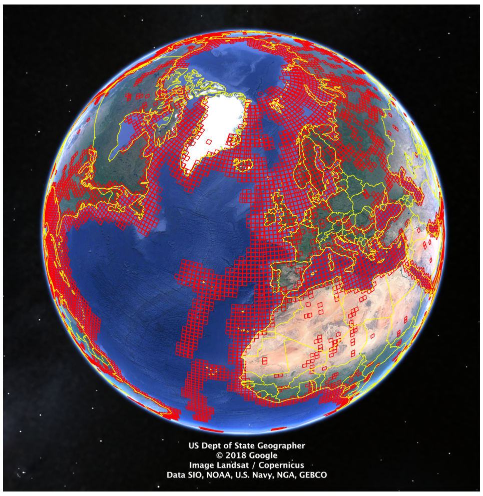

US Dept of State Geographer
© 2018 Google
Image Landsat / Copernicus
Data SIO, NOAA, U.S. Navy, NGA, GEBCO

Figure 4.20: Illustration of MGRS-tiles covered by Sentinel-2 Acquisition Plans, created by us as a KML file and plotted in Google Earth.

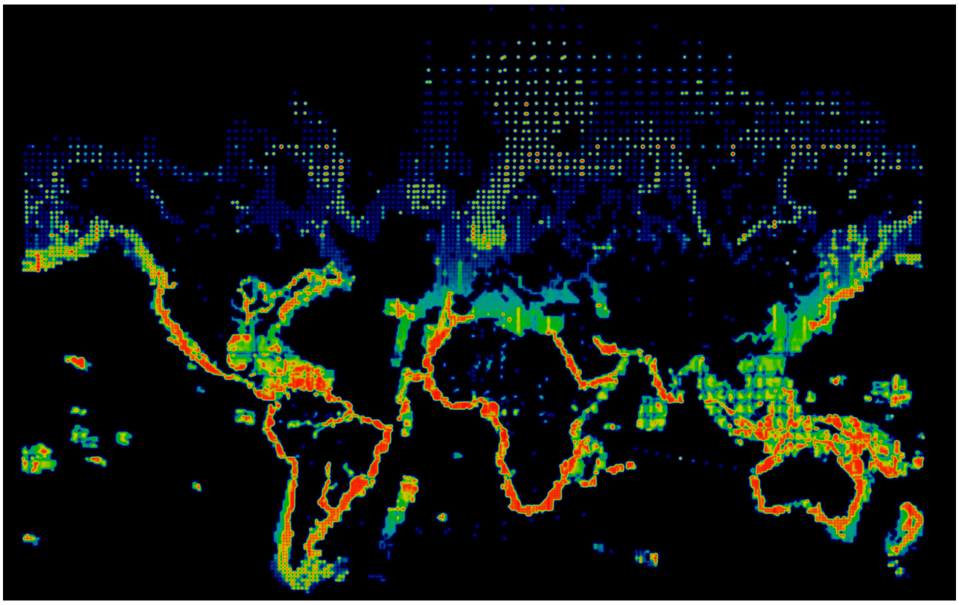


Figure 4.21: Heat map showing global ship density, created using our solution, for a period of 20 days, showing only areas covered by the swath of Sentinel-2. The covered areas correspond to the MGRS-tiles illustrated in Figure 4.20.

The heat maps are great for visualising the total data, but the distinct ship count of each

MGRS-tile quickly disappears in the crowd. This is due to the tool used for the visualisation will add together spatially nearby densities. Due to the rectification of the Mercator Projection, the maps will deceive us to some extent. The points near the equator will look very dense, and gradually become more sparse closer to the poles.

Other methods of visualisation can provide more valid information in this regard. We accomplished this by separating density due to spatial overlap, and single-tile ship density. The latter is indicated by circle radius and varying colour. To provide an example of different analysis, in Figure 4.22 we have filtered the data to only show AIS messages received by AIS compatible satellites.

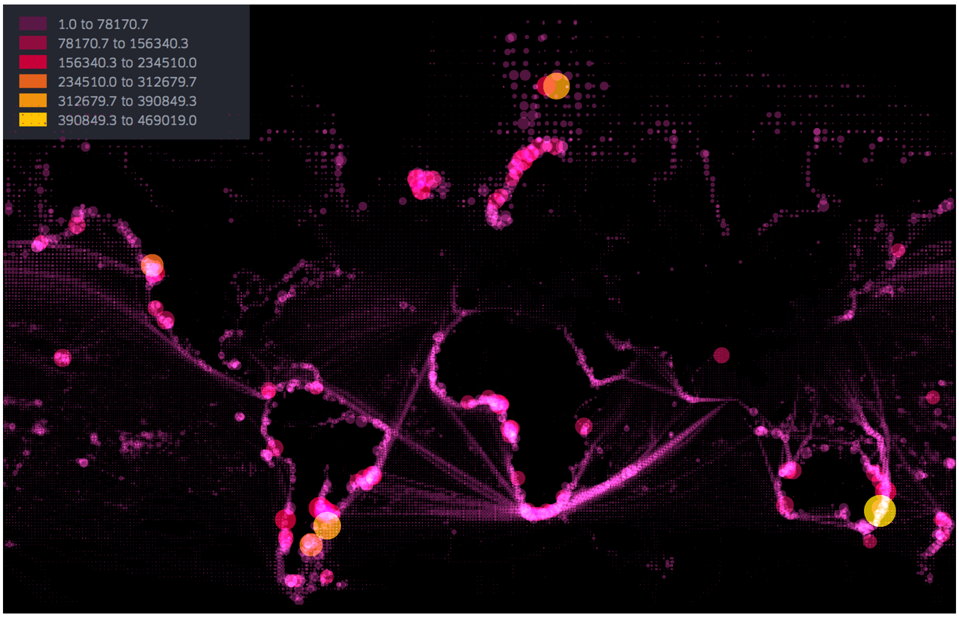


Figure 4.22: Heat map showing global ship density, created using our solution, within a period of 20 days and only using data collected by AIS compatible satellites. Cumulative ship density due to spatial overlap is indicated by the tone of pink. Single-tile ship density is indicated with circle radius and colour.

We wanted this analysis to end in regions of interest having a high likelihood of containing ship. We also wanted to maximise our chances to find images within that region that is cloud free. To do this, we combined the ship count with the revisit frequency of the satellite for each tile. To find the revisit frequency, we queried the database to find how many acquisition plans overlapped with each MGRS-tile.

By normalising and combining the two counts, we now had a tile-attribute that was

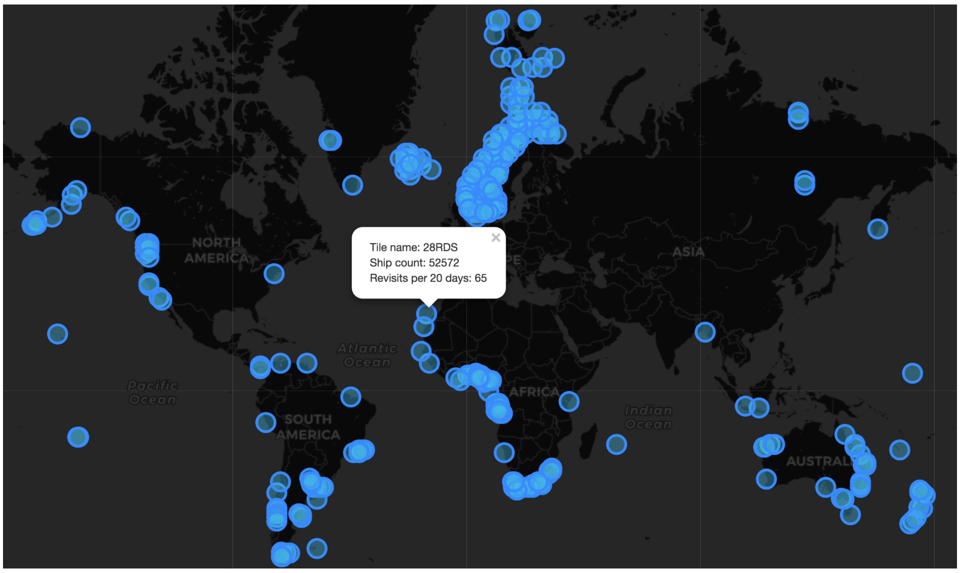


Figure 4.23: An interactive map we made, showing the set of MGRS-tiles having the highest likelihood of containing ship. They also have a high revisit frequency. In addition to ensuring more training data, this makes it easier to find images without cloud coverage.

natural to call the *ship likelihood constant* of the tiles. We now had a subset of tiles that could be ordered on having both high ship density, as well as high revisit frequency. This set consisted of 13.232 MGRS-tiles scattered globally, meaning we now had filtered out 76% of the total MGRS-tiles. To further decrease the number of tiles, we chose only to keep the tiles having a ship likelihood constant above the total mean. This produced 322 tiles, being 0.6% of the total 56.686 global MGRS-tiles. Inspecting the result, we consider the resulting set to be sufficiently spread globally, which means we will find examples of ship features from all seasons of the year, as well as at varied sun zenith angles. We have plotted these regions of interest in Figure 4.23. For convenience, the map is interactive with clickable markers, informing about MGRS-tile name, ship count, and revisit frequency.

On top of this, our approach easily lets us aggregate on the image tiles based on desired AIS attributes. This means, if needed, we can find images tiles having a high likelihood of containing ships having a specific speed, size, or heading. Also, as a reminder, all of this is done *without the need of downloading a single image*, saving us an enormous amount of time.

4.4 Dataset Generation

In the previous Section 4.1.4, we presented our solution for estimating the image timestamps as well as database query optimisation for handling the Big Data challenges from Section 3.4.2. A method for selecting the image footprints having the highest probability of containing a ship was also presented. Combining these results, we were able to start the actual ship position estimation within the images and use the resulting ship coordinates for labelling the images. Bearing in mind the desired dataset qualities, presented in 3.1, the following section will cover the steps to achieve this training dataset. This dataset should facilitate the future supervised machine learning such that it generalises well, and achieves accurate predictions.

4.4.1 Ship Position Estimation

Due to the AIS messages being *discrete* along the temporal axis, it is not likely that the image timestamp will match perfectly the timestamps of the AIS messages. Without correcting this, the difference in time between the two datasets leads to a displacement of the ship positions in the image, which in turn results in inaccurate labels. Therefore, more accurate ship positions were estimated by spatial interpolation between AIS positions prior to and past of the image timestamp. More accurately, for *all distinct* ships within an image, we found its two messages, transmitted right before and after the image was sensed. An example of two AIS positions prior to and past of the image timestamp is illustrated in Figure 4.24.

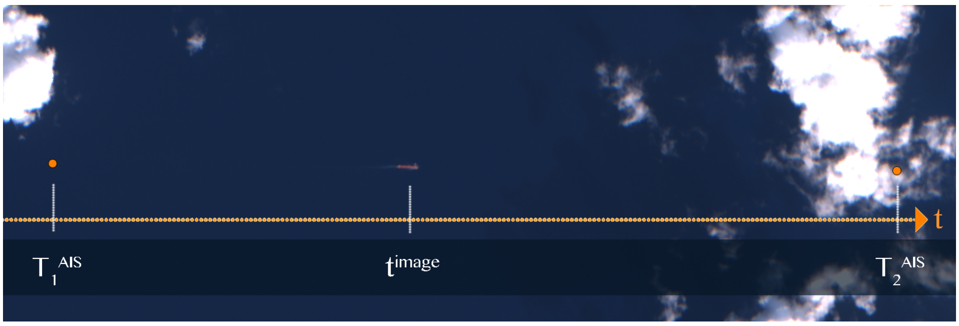


Figure 4.24: Visualisation of AIS positions, prior to and past of image timestamp, plotted on a Sentinel-2 TCI image.

AIS Position Interpolation

We define the known position vector $X(T_1)$ at timestamp T_1 prior the image timestamp t. The ships true position $X(t)$ in the image can be defined based on the known position

$$X(t) = X(t_1) + \Delta X \tag{4.1}$$

The change in position, ΔX is, for a given constant velocity v, defined as:

$$\Delta X = v \cdot \Delta t = v \cdot (t - T_1) \tag{4.2}$$

The speed may be provided by the ships AIS message, but not in all messages. In this case, we compute it using the other known position $X(T_2)$, after the image timestamp. As all three points are arranged on a straight line, the two known points will provide the direction of the speed, needed to compute the unknown position. This resulting velocity can hence be estimated by:

$$v = \frac{\Delta X_{1-2}}{\Delta T_{1-2}} = \frac{X_2 - X_1}{T_2 - T_1} \tag{4.3}$$

Inserting equations 4.3 and 4.2 into equation 4.1 yields the ships true position at the image timestamp, defined using only known values:

$$X(t) = X(T_1) + \frac{X_2 - X_1}{T_2 - T_1} \cdot (t - T_1) \tag{4.4}$$

Equation 4.4 describes how we estimate the ship positions in the further implementation.

However, before we start this, we have to go through the different scenarios of AIS messages that we have to take into account when interpolating the ship positions. There are three main scenarios, illustrated in Figure 4.25. The first and most common scenario is when both the prior and past pair of a ships AIS messages are within the image footprint. The second scenario takes place when one of the messages are positioned outside of the bounds of the image, while the other one is within. The third scenario occurs when both messages are outside of the image, but the straight line between them intersects the image footprint. All three scenarios are of interest, as long as the resulting ship position after the interpolation is located within the image bounds. This is always

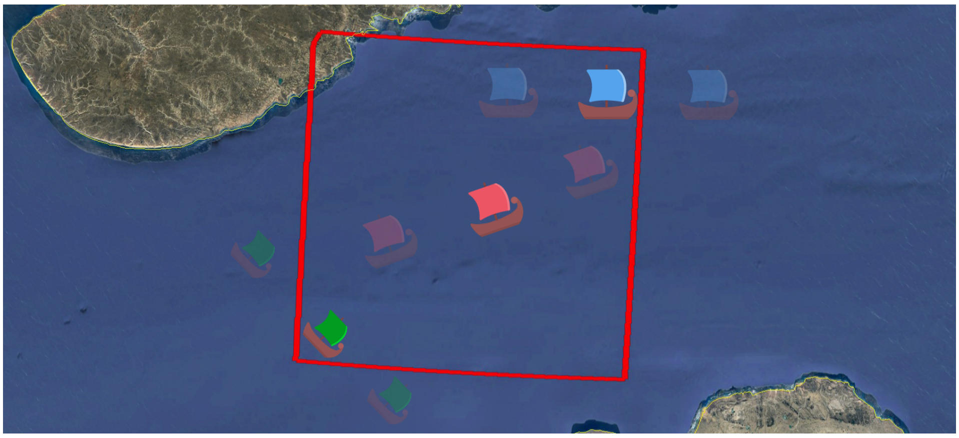


Figure 4.25: Illustration of different scenarios for interpolation of ship positions at the image timestamp. The transparent ships represent position each ship had when transmitting the AIS messages. The intermediate ships represent the ships position at the timestamp the image was sensed.

true for the first scenario, given that our estimation of the image timestamp is correct. The second and third scenario are a bit different, as one or both messages are positioned outside of the image bounds. We therefore had to include a spatial tolerance for the messages outside of the image bounds, as well as the ones within. However, as this might result in some ship position estimations outside of the image as well, one has to double check the resulting position is within the original image bounds.

To get a better overview of the algorithms for estimating ship positions in the images, we have written pseudo-code for the steps taken. They can be found in Algorithm 3 and 4.

Technical Implementation

The implemented algorithms are based on the presented pseudo-code, but are, as with the timestamp estimation, written in SQL and thereby differs a little. The code can be found in the Appendix B.

It begins by defining the input attributes: the image footprint, the estimated image timestamp and a time margin. The next step is to use this time margin to create a time interval around the image timestamp, and extract the AIS attributes from the ais_position table within this time interval. In this step we also have the option of filtering on ship speed, as well as the accuracy flag of the messages. In the next section, we will discuss the pros and cons of including these filters.

Algorithm 3: InterpolateShipPositionsWithinImage(
image, ais_position, time_margin, buffer_distance)

```
Input: image, with attributes footprint and timestamp
       ais_position, containing the AIS temporal messages
       time_margin, time difference tolerance
       buffer_distance, spatial distance tolerance
Result: Estimated ship positions
1  time_interval ⟵ TimeInterval(image.timestamp, ± time_margin)
2  ships_in_time_interval ⟵ messages from ais_position within time_interval
3  footprint_with_buffer ⟵ SpatialBuffer(image.footprint, buffer_distance)
4  ship_messages ⟵ SpatiallyWithin(ships_in_time_interval, footprint_with_buffer)
5  ship_positions[] ⟵ ∅
6  foreach Distinct(ship) ∈ ship_messages do
7      prior_msg ⟵ FirstMessagePriorTo(image.timestamp)
8      past_msg ⟵ FirstMessagePastOf(image.timestamp)
9      estimated_ship_position ⟵ InterpolatePosition(prior_msg, past_msg,
                                      image.timestamp)
10     if Within(estimated_ship_position, image.footprint, time_margin) then
11         ship_positions[] ⟵ estimated_ship_position
12     end
13 end
14 return ship_positions
```

Algorithm 4: InterpolatePosition(*prior_msg, past_msg, image_timestamp*)

```
Input: Prior and past AIS message, with attributes position and timestamp
       Image timestamp
Result: Interpolated position
1  position_vector_endpoints ⟵ EllipsoidalArcDistance(prior_msg, past_msg)
2  endpoints_time_difference ⟵ TimeDifference(prior_msg, past_msg)
3  image_time_difference ⟵ TimeDifference(prior_msg, image_timestamp)
4  mean_ship_velocity ⟵ position_vector_endpoints / endpoints_time_difference
5  ship_position ⟵ start_position + mean_ship_velocity ∗ image_time_difference
6  return ship_position
```

After filtering on time, we have a table of AIS messages that are temporally close to the image. However, the messages also have to be spatially filtered, which is obtained in the next step by using the PostGIS function *ST_DWithin*. Some of the relevant AIS messages might be positioned outside of the image's bounds. With *ST_DWithin* we collect these, as well as the ones within, by including all messages that are within a specified a distance to the image. For our purposes, we set this distance to 50 000 km, which is half of the image width, regarding the messages further away to be unsuitable for accurate interpolation.

The following two steps finds the nearest message prior to and past of the image timestamp for each distinct ship. These are obtained by first grouping the table by the ship identifying *MMSI*. Then each group are split into subgroups of messages having timestamps prior to, and past of the image timestamp. Ordering these subgroups descending and ascending, respectively, before extracting the first element of each, we obtain the two messages spanning the image timestamp, per distinct ship.

Knowing the prior and past message of each ship, we can calculate the parameters needed for the actual interpolation of the ship positions. The geodetic distance between the prior and past position is calculated using the function *ST_Distance*, as in the time interpolation algorithm. The two time differences are calculated by subtracting the timestamps from each other and represented as epoch values (that is, number of seconds since January 1st, 1970), giving us the differences in seconds. Dividing the geodetic distance with the time difference between prior and past message gives us the mean velocity of the ship. Multiplying this with the time difference between prior message and the image, we have the estimated distance to the ship. In addition, the azimuth angle between the prior and past position, that is the heading of the ship, is calculated.

For the interpolation, we use the PostGIS function *ST_Project*, which returns a point projected from a start point using a distance in meters and azimuth in radians. Using the position of the prior message, the calculated distance from this position to the ship, and the azimuth angle as inputs, we get at, last the, estimated ship position.

Lastly, we check if the newly estimated ship position is inside the image bounds. If not, there is no use for the estimated ship position. An example of estimated ship positions in an image is illustrated in Figure 4.26.

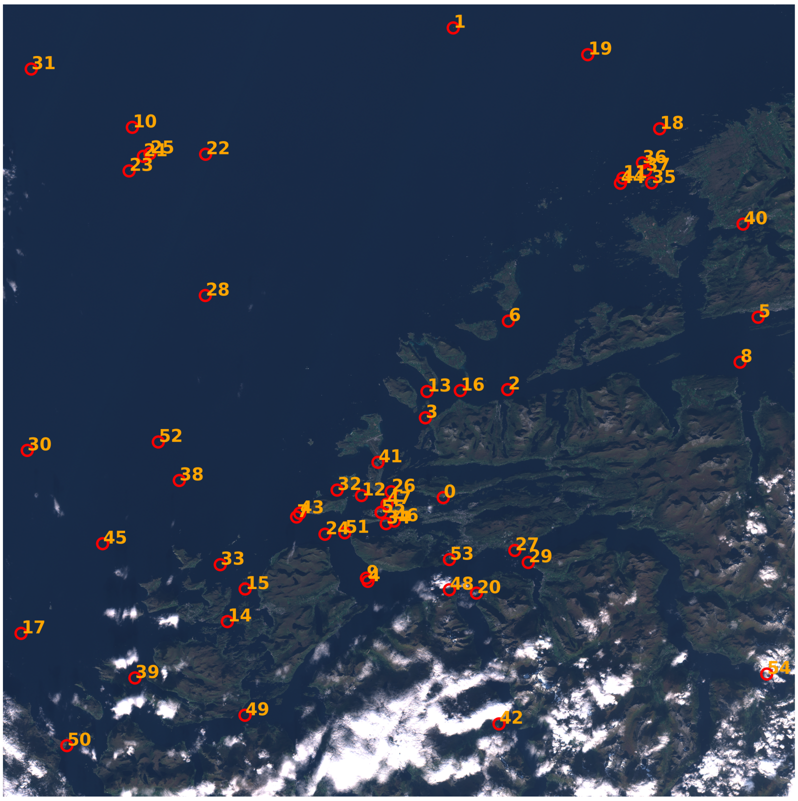


Figure 4.26: Estimated ship positions outside of Ålesund (MGRS tile 32VLQ), 24.09.2017, with numbering corresponding to images in Figure 4.27. The image size is 10 000 x 10 000 pixels and stretch over an area of 100 x 100 km, making the ships small and hard to spot. The image is of true colour TCI format, consisting of bands B02, B03 and B04.

4.4.2 Region Extraction and Labelling

In this section, we will present the ship region extraction procedure, as well as discussing methods and possibilities for the labelling of images. The ship region extraction facilitates dataset generation and image labelling. However, since the actual labelling itself can be further customised to the neural network architecture, it is a case for further work.

As the images are of size 100 x 100 km, and the ship features only covers a relatively small portion of it, they are not themselves well suited as training images. Indeed, the sheer size of them would make a neural network learn very slowly. Instead, we extract small regions of the areas around the estimated ship positions and use these as training images. In Figure 4.26, we see the full size image spanning Ålesund, Norway, with our estimated ship positions. One can barely see the ships, as they are so small relative to the image size, and hence, minimal parts of the image are of interest for the machine learning algorithm for detecting ships. In Figure 4.27 we have extracted the nine ships of the highest mean velocity around Ålesund, based on the estimated ship positions. The images are numbered, and correspond to the numbering in Figure 4.26.

To extract the regions of interest, we process the images using the Python GDAL API. As the regions cover the estimated ship positions, the first thing we need to do is to transform the ship coordinates to pixel coordinates. The ship coordinates are given in the WGS84 geographic coordinate reference system, and thus need to be transformed into the images' different UTM projections. They can further be transformed into pixel coordinates. [6]

Having the pixel coordinates, suitable regions of a defined size around the pixel are created. The regions are created in such a way that they are centred around the ship unless this causes a conflict with the full-size image border (e.g. the region crosses the image border where there is no data). If there is a conflict, the region will be moved within the image borders, while preserving the region size, as this is important for the training dataset. Having defined a region, we extract it the pixel values within the region and structure them in an array. The arrays can then be used to create the training images, such as the images in Figure 4.27.

An attribute one has to decide when creating the training images are the size of the regions. This is also related to how the images are labelled. In theory, we want the image size to be as small as possible, while still keeping all the relevant features within

[6]This is done using various GDAL functions including getting the affine transformation coefficients for the images. The pixel coordinates are calculated by subtracting the ship coordinates with the coordinates of the images' origin, the top left corner, divided by the pixel resolution.

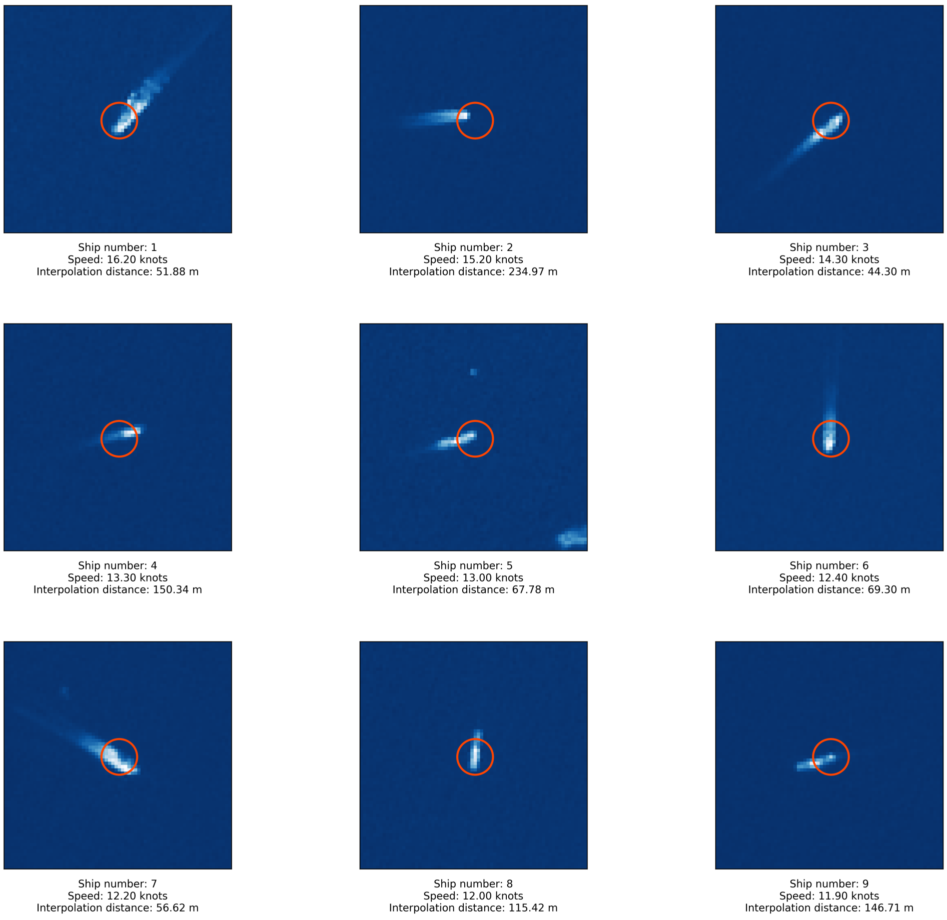


Figure 4.27: Extracted 70 x 70 pixel regions of the estimated ship positions in Figure 4.26, with corresponding numbers, presented in the B04 (Blue) band. As one can see, the estimated positions (orange circle) fits very good with the actual ship positions, even with varying interpolation distances.

the image. Keeping the images small make a neural network learn more effectively, as it has to process fewer pixels per image. A smaller area also reduces the chances of having unintentional other ships in the images, which, depending on how the images are labelled, could lead to false negatives. The issues of false positives and negatives are discussed in more detail in the next section. However, since the trailing wakes of the ships are important features for detection, the region has to be large enough for these to be included as well. The regions also need to compensate for eventual errors in the position estimation, to make sure the ship is within the extracted regions.

As for the labelling, there are several possibilities. When choosing labels, it is important to have a plan for how the labels are going to be used by the cost function in the neural network. In other words, how to measure the error between the ground truth in the labels and the output of the network. In this way, the choice of labelling decides what kind of output that will come from the network.

Perhaps the easiest method is to use binary labelling for image-level classification; either there is a ship in the image, or there is not. This is simpler because it is less reliant on exact positional coordinates. The output of the network would be a percentage of the likelihood for the regions to contain a ship. The errors could be calculated using the Cross-Entropy cost function between the percentages and the binary truths.

Another approach is to try to fit the ships around bounding boxes, and use these as the labels, for pixel-subset-based classification. The bounding boxes have to be automatically generated, which can be challenging due to the variety of parameters such as ship sizes, velocities, rates of turn, heading angles and trailing wakes. However, many of these attributes are present in the AIS messages or can be calculated from them. These can then be combined to optimise a bounding box for the given ship. The output of the network would be predictions of bounding boxes, and the cost could be calculated by the *Jaccard Index*, also known as *Intersection over Union* (IoU), between the prediction and ground truth. It is defined as the area of the intersection, divided by the area of the union between the boxes. An IoU close to 1 is considered a perfect match, and is what the network would try to optimise towards.

A third approach is to label the images with exact pixel coordinates for pixel-level classification. For a good result using this depends on accurate ground truth labelling of the ship positions. The advantage of this approach is that the network will learn to output exact ship coordinates in the images. For computing the error, one could use the L2-norm cost function, also known as the least squares error. Here, the difference between the predicted coordinate and the ground truth is the Euclidean distance, as proposed by Mišeikis et al. [54].

As previously mentioned, the work done in the present study facilitate for the labelling of images. However, the in-depth research for the optimal way of labelling should be performed in parallel to the choosing or designing, of the neural network architecture.

Acknowledging Incorrect Labelling

As introduced in the theory section, the accuracy of the ship position labelling can be measured by the number of false positives and false negatives. These scenarios occur when the labels do not correspond with what is observed at the coordinates in the image. As mentioned, false positives happen when the labels tell the program that there is a ship at coordinate (x, y) in the image, but when in fact there is *seemingly* not. This is quite typical for our case, where optical satellite imagery is involved, due to clouds and noise in the images that obscure the ship features. For these scenarios we introduce the term *disguised* positives, which we consider a more precise term for the ships that really are there, but not visible due to clouds or noise. We keep the term false positive for the scenarios where, contrary to the labels, there are in fact are no ships. False negatives, on the other hand, happen when there is a ship in the image, but the labels say otherwise.

The frequency of these positives and negatives depends largely on how well we select, filter and process both the Sentinel-2 images and the ship messages from AIS. However, how we choose to select and filter also affect the variety and volume of the training data.

In Section 4.3, a method for selecting appropriate Sentinel-2 images based on ship density was presented. Furthermore, the cloud cover percentage in the images' metadata allow us to filter out the images having high cloud density, which will reduce the number of disguised positives. Moreover, the cloud masks provided with the images allows us to filter out disguised positives by excluding the estimated ship positions overlapping with the masks. However, the cloud masks are of varying quality, so clouds can still be an issue. The noise in the images is much less frequent, but can still be an issue as it is harder to filter out. The noise is usually a result of an instrument anomaly, either onboard the satellite or in the processing done by the receiving ground station.

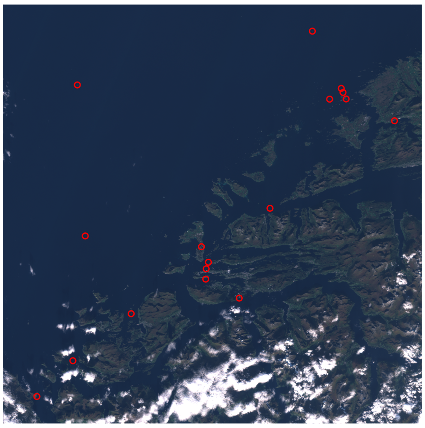
As there are 13 spectral bands, and the noise usually only appears in a limited amount of the bands at the same time, the other bands not affected by noise can often compensate for it. If an image is labelled as containing a ship at a given coordinate, but the ship is not visible in one of the bands due to noise, it could still be visible in the remaining bands.

The major contributor to false negatives is lack of AIS messages for ships. Smaller ships, especially, are not under regulations, and will thus not be labelled in our images. Larger ships might also lack messages due to reasons such as for some reason not wanting to be found, transporting refugees, or simply that the message itself has not been received by AIS.

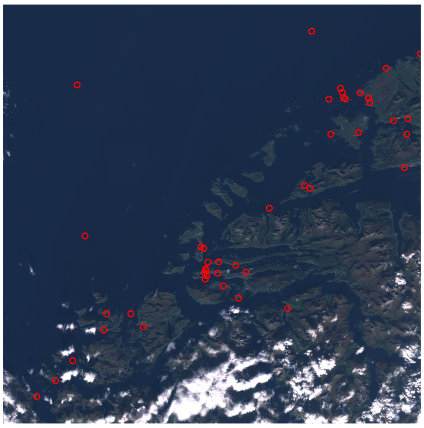
Another shortcoming of AIS is security threats such as ship spoofing and AIS hijacking [55], involving creating fake messages for non-existent ships, as well as altering information about real ships. The results of this, with respect to the accuracy of our labelling, are increased numbers of false positives and/or negatives. Non-existent ships lead to false positives, whereas information altering, such as changing locations of existing ships, can lead to both false positives and negatives.

In the previous section presenting the technical implementation of the ship estimation algorithm, we mentioned the options of filtering the AIS messages on attributes such as the accuracy flag and the speed. Including these two filters when filtering ship messages have both pros and cons with respect to the training dataset. The accuracy flag set to true allows that the positional uncertainty of the ship be less than 10 m, so by using it, we ensure precise AIS positions. However, as one can see in Figure 4.28a and 4.28c, where it is set to true and false, respectively, including it results in a lot less ship positions. This will again result in more ships in the images that are not labelled, thus more false negatives. Moreover, the 10 m accuracy might have little significance, as we have multiple other sources of uncertainty for the ship position estimations.

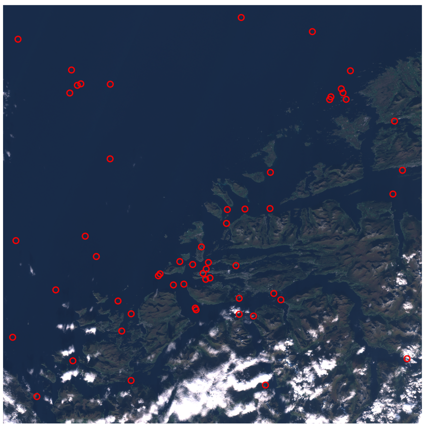
As one can see in Figure 4.28a and 4.28b, including the speed attribute when filtering will also reduce the number of ships greatly. For the results in the figures, we have set the speed filter to zero, meaning we filter out the ships not moving. Hence, most of the ships filtered out are at port. These are harder to distinguish than ships in the open ocean, due to the non-uniform background and lack of trailing wake. This can be seen in Figure 4.29, showing examples of ships in Ålesund area standing still. Notice also how the clouds in the upper right image obscure the ship, making it a false positive.



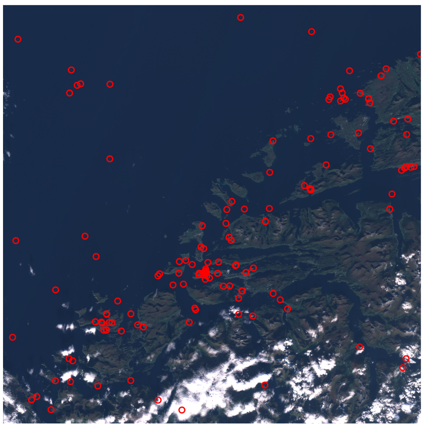
(a) Accuracy and Speed - Ships: 17



(b) Accuracy - Ships: 46



(c) Speed - Ships: 56



(d) Neither - Ships: 149

Figure 4.28: Estimated ship positions around Ålesund (MGRS tile 32VLQ), 24.09.2017, with different combinations of filtering on the AIS attributes Accuracy Flag and Speed. Filtering on speed means speed > 0, e.g. filter out ships not moving.

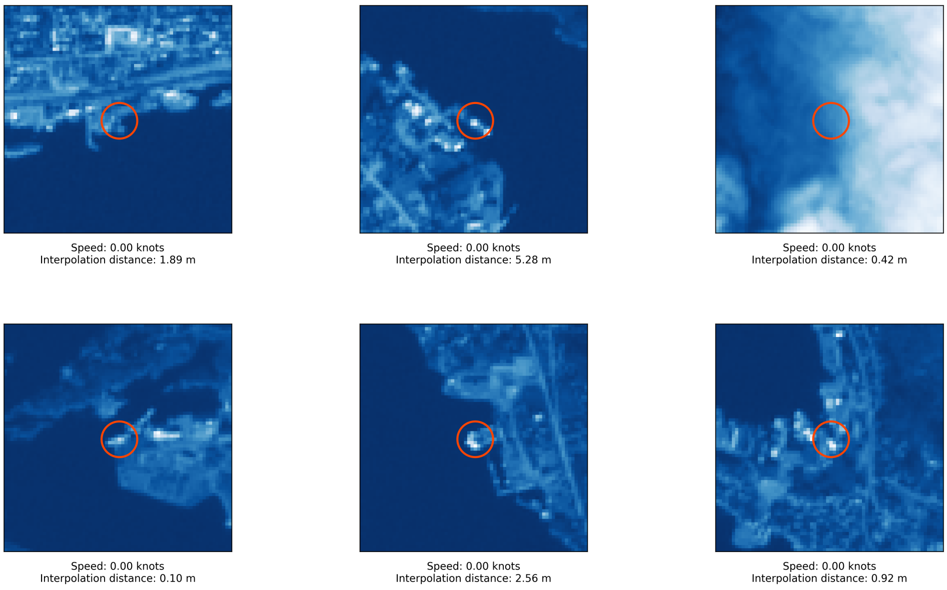

Speed: 0.00 knots
Interpolation distance: 1.89 m
Speed: 0.00 knots
Interpolation distance: 5.28 m
Speed: 0.00 knots
Interpolation distance: 0.42 m
Speed: 0.00 knots
Interpolation distance: 0.10 m
Speed: 0.00 knots
Interpolation distance: 2.56 m
Speed: 0.00 knots
Interpolation distance: 0.92 m

Figure 4.29: Extracted 70 x 70 pixel regions of some estimated ship positions for ships standing still, presented in the B04 (Blue) band. As one can see, it is harder to distinguish the ships as they blend in with the harbour and nearby buildings.

Ship wakes and waves can hopefully, in further work, be used to help in determining ship parameters such as ship size, velocity and maybe even tonnage. Hence, as we are also mainly interested in these kinds of ships, and think it is a good idea to filter the ship messages on speed, even though it reduces the number of training data per image and *can* lead to some false negatives. For our purposes, we believe it will increase the prediction accuracy for moving ships.[7]

Consequently, we decided to filter on speed, but not accuracy. For the image of Ålesund, this will result in the ship messages presented in Figure 4.28c.

[7]However, as we have multi-spectral imagery, the network should be able to distinguish ship and harbour more easily than the human eye. Also, one could train a network with a classifier for moving ships and a classifier for ships in harbours. Another solution could be to train two networks; one for detecting moving ships based on wakes, and one for detecting ships standing still. Then we would be sure the two types do not interfere with each other's learning. This is, however, research for further work.

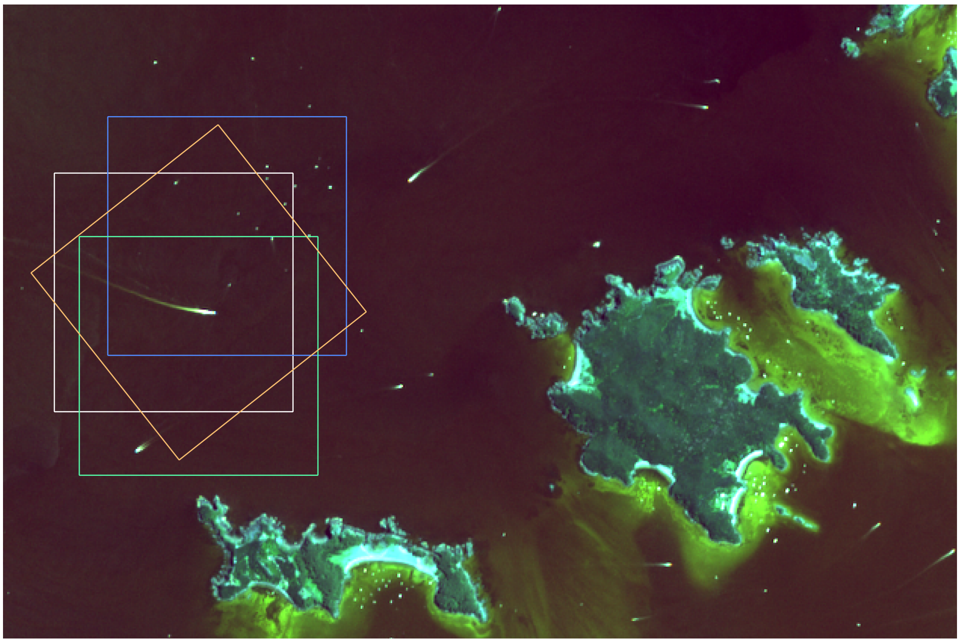


Figure 4.30: Dataset augmentation through multiple image region extractions.

4.4.3 Volume, Variety and Augmentation

In the Section 3.1.1, we also mentioned the importance of having great volume and variety in the training set. Fortunately, the volume is of no issue in this project, as we have an enormous amount of both Sentinel-2 images and AIS messages to base the generation of training data on. Moreover, a single Sentinel-2 image can contain many ships, and result in numerous training images. Generating a large set of training images, we reduce the effect of disguised and false positives, and false negatives, as long as the majority of the images are correctly labelled.

Large volumes of data also bring natural variety into the training dataset, since a more diverse range of ship scenarios will be observed. Such as ships of different velocities, directions, sizes and perhaps varying degrees of obscurity due to clouds. This variety is essential for the network's ability to generalise new, unseen data. As mentioned in Section 3.1, this implies that the images we select are of areas spread globally, to account for differences in sun zenith angles and other possible unforeseen, location-dependent variables. The Image Selection Optimisation has largely solved this in Section 4.3.

To add additional variety in the training data, as well as enlarging the volume, we can perform data augmentation. There are various augmentation operations relevant for remote sensing training images. X. Yu et al. [56] perform horizontal mirroring, translation and rotation on remote sensing data, obtaining great effect for the diversity of the training dataset. These three operations enhance the diversity while preserving the scene topologies in the imagery, which in general is important for consistent scene classification. In our case, the ship wake should always be in the region immediately after the moving ship. As we can extract an what every region we want from the images, we have a great advantage. Examples of region extraction in a multi-spectral image is shown in 4.30.

For our purposes, the rotation would add additional variety to the heading angles of a ship, which would help the network to generalise to ships moving in all directions. Translation and mirroring would add variety in the locations to recognise ships in the image. By not having this variety in the training dataset, the network might make an incorrect correlation between the labelled position in the image, and ship features.

Additional augmentation operations that are relevant include scaling and noise insertion. As AIS does not cover ships of smaller sizes, there is no ground truth for these ships in the images. By scaling down larger ships, while keeping the topology, one could generate training data for the network to detect smaller ships as well.

Chapter 5

Conclusions

In the introduction of this thesis we stated the following primary objectives:

1. Develop a method to generate a training dataset for ship detection automatically.
2. Investigate how our training dataset can achieve qualities that positively affect the results of supervised machine learning.
3. Optimise this method to be highly scalable and facilitate further development into a real-time data processing solution.

These fundamental objectives were broken into multiple partial objectives. During the research and analysis, they proved to be more or less challenging and time-consuming than expected. This was primarily due to the unforeseen issues and challenges described in Section 3.4, that threatened the central concept. In this way, they affected the whole course of the project. However, the challenges, although time-consuming, were confronted head-on and, solved in such a way that we could proceed within the planned direction. In the next paragraphs, we summarise the work before we discuss our findings and how the different objectives were met. In the end, we present the recommendations for further work.

In Chapter 2, we presented important qualities the resulting training dataset should hold for the machine learning generalise well and predict with high accuracy. Throughout the thesis, we have strived to keep a focus on how to achieve these qualities in our training dataset for ship detection. In the remaining of the chapter, we made a thorough introduction of both systems used in the approach, the Automatic Identification System (AIS) and the Sentinel-2 satellites.

Having a basis for how the two systems work, and what is required from a training dataset to facilitate machine learning, we defined in Chapter 3 prerequisites for the two datasets. First and foremost, this includes the compatibleness of them. To combine the two datasets with respect to four dimensions, both datasets had to have attributes that describe geolocation and time. And for accuracy in the labelling, these attributes had to be of high quality. Secondly, it included that we are able to generate a dataset of sufficient volume, such that a neural network can learn to generalise. The rest of the chapter went into in-depth research of the two datasets and how they could be integrated.

Upon this research, we were met with the aforementioned, critical issues that changed the scope of the project. The first issue was the incorrect image timestamps of the Sentinel-2 metadata. This left us with only being provided with the timestamp of the beginning of the datatake, not a precise timestamp that we needed to match the images with the AIS messages. Much effort went into researching this issue more in-depth, as it seemed improbable that such an important attribute was non-existent in ESAs metadata.

The second issue was the performance of our system, especially the insertion, sorting and querying of the overwhelming amount of AIS data. The system that had performed well so-far on a limited amount of AIS data had big trouble handling the now hundreds of millions of AIS messages, that we needed to use in the forthcoming training dataset generation.

Needing to solve these issues before going further with ship position estimation and dataset generation, Chapter 4 presented different approaches to estimate a correct timestamp for the Sentinel-2 images. As a generalised method for the estimation, applicable to all images taken globally, was required. Appropriate metadata attributes for the ordering of images in a datatake were lacking, which made this task all other but trivial. Luckily, we came upon the Sentinel-2 acquisition plans, that made us able to interpolate the image timestamp from the start and end points of the *planned* datatake, spanning the image. Parallel to this, in-depth analysis and performance optimisation of the database queries were carried out. Since the required amount of AIS messages in the database was enormously high, the filtering on these, with respect to image footprints and timestamps, had to be significantly optimised. Interesting findings include the performance advantage of utilising *Common Table Expressions* in the queries to force the database to filter in an optimal order. This resulted in an execution time improvement in the magnitude of thousands.

This further led us to the task of optimising Sentinel-2 image selection. As processing large amounts of image data is a computationally heavy task, it was important to narrow down the search for relevant images. Furthermore, it was essential to create a method for calculating the probability for an image to relevant, without having to download the actual image. In other words, finding a way of estimating the a priori likelihood for an image to contain ships, based on the historical ship traffic in the area the image covers.

To solve this, we counted the number of AIS messages for each MGRS tile that are covered by the swath of the Sentinel-2 satellites. This resulted in a heat map showing global ship density in the relevant tiles, giving an indication of the probability to contain ships. Furthermore, by combining the ships count per tile with the satellite revisit frequency for this tile, we ended up with the image tiles having the highest likelihood of containing ships.

This was followed by the method for estimating ship positions in an image. As mentioned, due to the AIS messages being discrete along the temporal axis, it is not likely that the image timestamp will match perfectly with the timestamps of the AIS messages. This will lead to an inevitable displacement of the ship position in the image, due to the movement of the ship. To compensate for this, an algorithm involving spatial interpolation between the two AIS positions spanning the image timestamp was implemented.

Having the estimated positions of the ships in the image, we had, in fact, the information needed to start the generation of training images. Due to the images being extremely large relative to the ship features, these full-size images are not very suited for training. The network would need to process a lot of uninteresting pixels. Instead, we implemented a method that extracts the regions, of predefined size, around every ship position estimated in the image, and use these regions as training images. Extracting the regions and plotting them, we also observed that the estimated ship positions meet the actual ship positions quite well and relatively consistent for all ships in the image. Further in this section, we discussed the various possibilities at hand to decrease the probability of incorrect labelling, with respect to false positives and negatives. In addition, we presented methods of augmentation to increase the volume and variety of the training dataset, to achieve generalisation in the machine learning.

These analyses and implementations were structured programmatically so that the whole process is automatic, from selecting an image to the extraction of regions around ships. In this way, we have created a data processing pipeline for the labelling of training images. The different parts that make up the pipeline are also designed to be modular so that the modules can be interchanged and used flexibly to fit future needs. The pipeline consists of the following three main modules: *Image Selection Optimisation*, *Ship Posi-*

tion Estimation and *Training Dataset Generation*.

In the *Image Selection Optimisation*, we are able to perform a density analysis on 400 million ship messages, with an execution time of less than 3 minutes. In the *Ship Position Estimation*, we correct the timestamp of an arbitrary satellite image and do complex spatial and temporal data integrations to compute the coordinates of every ship in the image, with an execution time of less than two seconds. In the *Training Dataset Generation*, we extract regions from the image around each estimated ship position, resulting in multiple training images of more suitable size.

This leads us back to the first primary objective; ***Develop a method to automatically generate a training dataset for ship detection***, which we certainly feel we have met. The data processing pipeline takes care of this by effectively integrating Sentinel-2 MSI with AIS. In addition, the current data pipeline also has value as-is, without the further aspect of machine learning. The images can be used to control the AIS, detecting, for instance, fake AIS messages, or find suspicious ship behaviour, with respect to, for instance, illegal fishery. The solution for estimating image timestamp can also be used in applications not involving ships, such as an application monitoring woodlands for warning about forest fires.

As for the second objective; ***Investigate how our training dataset can achieve qualities that positively affect the results of supervised machine learning***, we have throughout the whole thesis strived to underline and base the choices we have made on how they positively affect supervised machine learning. By bearing this in mind from the start, we have managed to achieve good estimations for the ship positions, which affect the accuracy of the labelling. We have also acknowledged the possibility of incorrect labelling when it comes to false positives and negatives due to the factors such as the presence of clouds, or fake AIS messages. However, as we also discussed, there are also ways of reducing the probability of the occurrences of these, such as utilising cloud masks to filter out the ships obscured by clouds. Furthermore, we have the choice of applying different filters in search of ships, such as the accuracy flag and ship speed, to limit the search to more reliable and suitable ship objects, but these come with a price of reduced volume and variety in the dataset. Regarding this, we ended up filtering out ships not moving, as they are harder to observe due to the non-uniform background close to harbours as well as the absence of waves and wakes. The other qualities we have strived to underline the importance of, are the volume and variety of the dataset, to achieve generalised machine learning. Concerning this, we can do data augmentation operations such as rotation and translation of the training images, to get more variety into the training dataset, and an increase in volume as a bonus. However, this is a case for further work.

The third objective was ***Optimise this method to be highly scalable, and facilitate further development into a real-time data processing solution***. Referring to the execution times of the modules in the data processing pipeline, there is undoubtedly room for scaling up the system substantially. The fast execution times are a result of the significant optimisation of the database queries, where we facilitated for further work requiring scaling of the database with respect to AIS messages. As for the facilitation of an eventual future real-time solution, this is also something that we have kept in mind throughout the work and is also substantiated by the execution times. For instance, the estimation of image timestamp using *a priori* acquisition plans was favourable, as one by this can estimate the individual image timestamps *before* the image is taken, which is preferable for a real-time solution.

Summing up, we feel that the objectives have been met to a satisfactory extent. The implemented data processing pipeline generates a training dataset for ship detection using supervised machine learning. It facilitates for achieving the qualities needed to obtain generalisation and predictions of high accuracy, as well as the opportunity to scale the system greatly and develop the system into a real-time data processing solution in the future.

5.1 Recommendations for Further Work

Since the target of this thesis has been to generate a training dataset for supervised learning, this is an obvious extension of our work. However, there are numerous other topics and improvements that we have not covered, which we recommend for further work.

Label Method Optimisation

As discussed in Section 4.4.2, there are several ways of labelling the training images. In the results of the present project, exact pixel point coordinates of the estimated ship position were used as the label. This might also be the way to go in future work, but there also exists labelling methods such as including bounding boxes using IOU as the cost function, or binary labelling where exact coordinates are not used. The different methods have both pros and cons and are dependent on the wanted output of a machine learning algorithm. This also includes defining cost functions as well as training image size. A recommendation for further work is to do an in-depth analysis of the labelling options and optimise the labelling method, with respect to ship detection.

Data Augmentation

Also discussed in Section 4.4.2, is to use data augmentation to enhance the variety in the training dataset, as well as increasing the volume. This includes applying operations such as rotation, translation and horizontal flipping to vary the positions of ships in the image as well as the directions it moves. Scaling can be used to get labelled examples of small ships in the images. Semi-transparent cloud insertion onto images can be used to train the network to look through clouds.

Improved Ship Position Estimation

Once again in Section 4.4.2, we mentioned some possibilities for improving the ship position estimation. The current solution does not compensate for ships making a turn or accelerating between the two AIS messages. This can lead to false positives (and false negatives). By including the AIS attributes turn rate and speed in the interpolation of the ship position, this could be compensated for.

Utilising Cloud Masks

In Section 4.4.2 we also mentioned the possibility to utilise the Sentinel-2 Cloud Masks to filter out ship positions obscured by clouds. This could be an additional step in the data processing pipeline, following the ship position estimations, where one merely filter out the ship coordinates that are within the cloud mask that ship with the product.

Quantitative Analysis of Timestamp Interpolation

In Section 4.1.2, we estimated the image timestamp by interpolation using the acquisition plans. To ensure the acquisition plans were of use, we did a quick spatial and temporal comparison with their respective, actual datatakes. Moreover, we filter out the acquisitions that are too far off temporally. However, it could be useful to do a quantitative analysis of, especially, the spatial, but also, the temporal uncertainty of the acquisitions, and the timestamp interpolation in general. As an example, how well the timestamp interpolation performs with varying datatakes lengths as well as the location of the image in the datatake. Linking up with the ship position estimation, one could see if there are any correlation between the accuracy of the ship position estimation, and the length of the distances used in the timestamp interpolation.

Estimate Image Timestamp By Orbit Model

As introduced in Section 4.1.3, it is possible to satellite ephemerides to simulate the Sentinel-2 orbit in four dimensions, with a temporal resolution of one second. These timestamps can be, in turn, be mapped to an image based on the image footprint coordinates and the relative orbit the image was captured. The approach was not investigated further, as it requires an orbit simulation of the whole time range one has images that need timestamp estimation, and this would require a lot of time to learn the simulation software. However, it looks very promising in terms of estimation accuracy and could be tested in further work.

Deep Learning

The goal of this thesis was to develop a method to automatically generate a training dataset for ship detection in machine learning. The motivation for this is to use this training dataset to train a Convolutional Neural Network that automatically detects ships in Sentinel-2 images. A trained network would detect ships without having to rely on AIS data, and could additionally detect ships not using AIS. This is advantageous in many fields, such as detecting illegal fishery, piracy, refugees and other ships not able or not wanting to be found by AIS.

Classifying clouds
The neural network could also learn to classify clouds, and in this way reduce the number of *disguised* positives due to obscured ships. To train the network to classify clouds, the ESA Cloud Masks could be used as training data.

Image Timestamp
A trained neural network would detect ships in images without having to rely on AIS messages. Thus it would not need to integrate the data sets, and because of this, the incorrect timestamp in the image metadata would not be an issue.

Spectral Band Combinations
By testing different combinations of spectral bands as inputs to the network, one can see if some combinations of multispectral reflectances are better suited than others to distinguish ships from other similar objects, such as ship wakes and harbours. This would further enhance the prediction accuracy of the network.

Ship Recognition and Determining Ship Parameters

By comparing the ship objects of the Sentinel data with the ground truth in AIS, one can evaluate the feasibility of determining ship parameters, such as length, breadth, speed, ship depth and heading, by doing calculations on the pixel reflectances. Heiselberg et al. introduce a method for calculating ship parameters based on satellite imagery [57]. By taking advantages of the different pixel reflectances in the multispectral bands, he was able to compute ship parameters with reasonable accuracy, and additionally do analysis on the wakes and Kelvin waves of the ships. The amount of wakes and waves a ship make is correlated to ship parameters such as length and breadth, and speed. This lets him not only detect ships in images but also recognise and identify them. This could further be tested for a neural network for automatic identifying of ships. It could be made more robust by comparing predicted ship parameters, of an observed ship in the image, with parameters from ship messages of spatially and temporally close ships. If the network detects a ship not covered by AIS, and the estimated parameters is a good match with the parameters found in temporally and spatially close AIS message, one could calculate a probability of it being this ship having turned off its AIS system.

Targeted Ship Detection

Similar to the last paragraph, but not involving determining ship parameters or machine learning. With the basis of having the data processing pipeline including a central database containing global AIS messages for a long time period, one could implement methods for targeted ship detection and tracking. This would involve using sequential AIS messages for the same ship in a defined time period, and use the attributes, such as position, speed and direction, for instance in a Kalman filter, to predict the ship movement trajectory. Combining this with acquisition plans, one could predict where and when the ship would be sensed in an image next.

Real-Time Solution

One of the primary objectives of the thesis is to facilitate a future real-time solution. Further work could include implementing a real-time solution both for the data processing pipeline developed in this project, but also for real-time ship detection by training a neural network. A real-time solution for the data pipeline would require real-time data of both Sentinel-2 imagery and AIS. A real-time solution for the neural network would only require real-time Sentinel-2 data, although controlling with additional AIS would make the solution more robust. The information resulting from real-time solutions could be of great value to different companies and actors in the market, as well as for applications such as search and rescue missions and disaster management due to for instance ship collisions.

Bibliography

[1] European Space Agency. *Open Access Hub*. 2017. URL: https://scihub.copernicus.eu/ (visited on 08/12/2017).

[2] MathWorks. *Convolutional Neural Network - MATLAB & Simulink*. 2018. URL: https://se.mathworks.com/solutions/deep-learning/convolutional-neural-network.html (visited on 11/06/2018).

[3] Chabacano. *Overfitting*. 2008. URL: https://commons.wikimedia.org/wiki/File:Overfitting.svg (visited on 11/06/2018).

[4] Marine Safety Comittee. *Automatic Identification Systems (AIS)*. 2015. URL: http://www.imo.org/en/OurWork/safety/navigation/pages/ais.aspx (visited on 08/12/2017).

[5] MarineTraffic. *Satellite AIS - Global AIS Coverage | AIS Marine Traffic*. 2017. URL: https://www.marinetraffic.com/en/p/satellite-ais (visited on 10/12/2017).

[6] Norsk Romsenter. 'Norge I Rommet'. In: 2 (2015).

[7] European Space Agency. *Atlantis leaves Columbus with a radio eye on Earth's sea traffic*. 2009. URL: http://www.esa.int/esaMI/Operations/SEMIHX49J2G{_}0.html (visited on 09/12/2017).

[8] Bjørnar Kleppe. *Kystverket*. 2015. URL: http://www.kystverket.no/Maritime-tjenester/Meldings--og-informasjonstjenester/AIS/AISSat-1-og-AISSat-2/ (visited on 10/12/2017).

[9] MarineTraffic. *Ships List - Vessel Search | AIS Marine Traffic*. 2017. URL: https://www.marinetraffic.com/en/ais/index/ships/all (visited on 09/12/2017).

[10] All About AIS. 'AIS TDMA Access schemes - Technical summary'. In: (2012), pp. 1–11. URL: http://www.allaboutais.com/jdownloads/Accessschemestechnicaldownloads/ais{_}tdma{_}access{_}schemes.pdf.

[11] ExactEarth. 'Real-Time Advanced Ship Tracking Solution'. In: (2015). URL: https://cdn2.hubspot.net/hubfs/183611/Landing{_}Page{_}Documents/exactView{_}RT{_}Slick{_}Sheet.pdf.

[12] US Department of Homeland Security. *Class A AIS position report*. 2013. URL: http://www.navcen.uscg.gov/?pageName=AISMessagesA (visited on 10/12/2017).

[13] US Department of Homeland Security. *AIS Standard Class B Equipment Position Report (Message 18)*. 2013. URL: https://www.navcen.uscg.gov/?pageName=AISMessagesB (visited on 10/12/2017).

[14] European Space Agency. 'Sentinel-2 User Handbook'. In: (2015). URL: https://sentinel.esa.int/documents/247904/685211/Sentinel-2{_}User{_}Handbook.

[15] European Space Agency. *Sentinel-2 - Missions - Sentinel Online*. 2017. URL: https://sentinel.esa.int/web/sentinel/missions/sentinel-2 (visited on 08/12/2017).

[16] European Space Agency. *Sentinel-2 Toolbox*. 2017. URL: https://sentinel.esa.int/web/sentinel/toolboxes/sentinel-2 (visited on 17/12/2017).

[17] Sentinelsat. *Sentinelsat — Sentinelsat 0.12.1 documentation*. URL: http://sentinelsat.readthedocs.io/en/stable/index.html (visited on 11/12/2017).

[18] Amazon Web Services. *Sentinel-2 on AWS*. 2017. URL: https://aws.amazon.com/public-datasets/sentinel-2/ (visited on 19/12/2017).

[19] QGIS. *QGIS website*. URL: https://www.qgis.org/en/site/ (visited on 17/12/2017).

[20] Hairunnizam Wahid et al. 'Prestasi kecekapan pengurusan kewangan dan agihan zakat: perbandingan antara majlis agama islam negeri di Malaysia'. In: *Jurnal Ekonomi Malaysia* 51.2 (2017), pp. 39–54. ISSN: 01261962. DOI: 10.1017/CBO9781107415324.004. arXiv: arXiv:1011.1669v3. URL: http://www.ibmbigdatahub.com/infographic/four-vs-big-datahttp://www.ibmbigdatahub.com/sites/default/files/infographic{_}file/4-Vs-of-big-data.jpg.

[21] Copernicus. *Develop your Copernicus-based business with the Copernicus Incubation Programme | Copernicus*. 2018. URL: http://copernicus.eu/news/develop-copernicus-based-business-incubation-programme (visited on 06/06/2018).

[22] Norwegian Space Centre. *NorSat-1 and NorSat-2 launched!* 2017. URL: https://www.romsenter.no/no/News/News/NorSat-1-and-NorSat-2-launched (visited on 05/06/2018).

[23] Norwegian Coastal Administration. *Kystverket*. 2016. URL: http://www.kystverket.no/en/EN{_}Maritime-Services/Reporting-and-Information-Services/Automatic-Identification-System-AIS/AIS-User-Access/ (visited on 12/12/2017).

[24] National Marine Electronics Association. *NMEA*. 2008. URL: https://www.nmea.org/content/nmea{_}standards/nmea{_}0183{_}v{_}410.asp (visited on 12/12/2017).

[25] Goatbar. *libais 0.16 : Python Package Index*. 2015. URL: https://pypi.python.org/pypi/libais (visited on 13/12/2017).

[26] International Standards Organization. *Date and time format - ISO 8601*. 2017. URL: https://www.iso.org/iso-8601-date-and-time-format.html (visited on 08/05/2018).

[27] MarineTraffic. *Vessel details for: CAPE MAY (Fishing Vessel) - IMO 8103028, MMSI 368489000, Call Sign WDE2195 Registered in USA | AIS Marine Traffic*. 2018. URL: https://www.marinetraffic.com/en/ais/details/ships/shipid:454779/mmsi:368489000/vessel:368489000 (visited on 08/05/2018).

[28] PostQGIS Development Team. *Spatial and Geographic objects for PostgreSQL*. 2015. URL: http://postgis.net/.

[29] Mark Leslie, LISAsoft and OpenGeo. *Introduction to PostGIS Section 8: Spatial Indexing*. 2009. URL: http://revenant.ca/www/postgis/workshop/indexing.html (visited on 16/12/2017).

[30] The PostgreSQL Global Development Group. *PostgreSQL: Documentation: 9.4: JSON Types*. 2018. URL: https://www.postgresql.org/docs/9.4/static/datatype-json.html (visited on 07/05/2018).

[31] James Graffenberg. 'What is AIS?' In: *Stretching The World* (2008). URL: http://www.vtexplorer.com/what-is-ais/.

[32] Marine Traffic. *Vessel details for: GULLIVER (Fishing) - MMSI 257085340, Call Sign LK7260 Registered in Norway | AIS Marine Traffic*. 2017. URL: https://www.marinetraffic.com/en/ais/details/ships/shipid:1684817/mmsi:257085340/vessel:257085340 (visited on 05/06/2018).

[33] European Space Agency. *Sentinel-2 images the globe every 5 days - Sentinel-2 - News - Sentinel Handbook*. URL: https://sentinel.esa.int/web/sentinel/missions/sentinel-2/news/-/article/sentinel-2-images-the-globe-every-5-days (visited on 03/06/2018).

[34] ESA. *Open Access Hub Advanced Search*. 2018. URL: https://scihub.copernicus.eu/userguide/4AdvancedSearch (visited on 11/05/2018).

[35] ESA. *Open Access Hub Full Text Search*. 2018. URL: https://scihub.copernicus.eu/userguide/3FullTextSearch (visited on 11/05/2018).

[36] ESA. *Naming Convention*. 2018. URL: https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/naming-convention (visited on 11/05/2018).

[37] Andrew Hunt and David Thomas. 'Tracer Bullets'. In: *The Pragmatic Programmer : From Journeyman to Master; Amazon Kindle Version*. 1999, pp. 1057–1144. URL: http://don.fed.wiki.org/view/tracer-bullets.

[38] SQLAlchemy authors and contributors. *SQLAlchemy ORM — SQLAlchemy 1.2 Documentation*. 2017. URL: http://docs.sqlalchemy.org/en/latest/orm/ (visited on 16/12/2017).

[39] Federico Di Gregorio and Daniele Varrazzo. *Psycopg 2.7.4.dev1 documentation*. 2017. URL: http://initd.org/psycopg/docs/extras.html{\#}psycopg2.extras.execute{_}values (visited on 17/12/2017).

[40] Andre Lima. *Sensing time data from Sentinel 2 Tile/granule (L-1C) - s2tbx - STEP Forum*. 2017. URL: http://forum.step.esa.int/t/sensing-time-data-from-sentinel-2-tile-granule-l-1c/4570 (visited on 07/05/2018).

[41] N. Pahlevan et al. 'Sentinel-2 MultiSpectral Instrument (MSI) data processing for aquatic science applications: Demonstrations and validations'. In: *Remote Sensing of Environment* 201.September (2017), pp. 47–56. ISSN: 00344257. DOI: 10.1016/j.rse.2017.08.033.

[42] ESA. *User Guides - Sentinel-3 OLCI - Coverage - Sentinel Online*. 2018. URL: https://earth.esa.int/web/sentinel/user-guides/sentinel-2-msi/product-typeshttps://sentinel.esa.int/web/sentinel/user-guides/sentinel-1-sar/acquisition-modes/interferometric-wide-swath (visited on 06/06/2018).

[43] European Space Agency. *Sentinel-2A Acquisition Plans - Sentinel Online*. 2018. URL: https://sentinel.esa.int/web/sentinel/missions/sentinel-2/acquisition-plans (visited on 06/06/2018).

[44] Kjell Eikland. *Overview of services - Eikland Energy AS - Energy Perspectives*. 2017. URL: https://www.energyper.com/ (visited on 06/06/2018).

[45] University of Maryland. *André de Lima | GEOG l Geographical Sciences Department l University of Maryland*. URL: https://geog.umd.edu/facultyprofile/deLima/Andr{\'{e}} (visited on 06/06/2018).

[46] ESA Earth online. *Orbit - Sentinel 2 - Mission - Sentinel Online*. 2015. URL: https://sentinel.esa.int/web/sentinel/missions/sentinel-2/satellite-description/orbithttps://earth.esa.int/web/sentinel/missions/sentinel-2/satellite-description/orbit (visited on 06/06/2018).

[47] Timescale. *Timescale | an open-source time-series SQL database optimized for fast ingest, complex queries and scale*. 2017. URL: http://www.timescale.com/https://www.timescale.com/ (visited on 02/06/2018).

[48] Mike Freedman. *Time-series data: Why (and how) to use a relational database instead of NoSQL*. 2017. URL: https://blog.timescale.com/time-series-data-why-and-how-to-use-a-relational-database-instead-of-nosql-d0cd6975e87c.

[49] The PostgreSQL Global Development Group. *PostgreSQL: Documentation: 9.4: What is PostgreSQL?* 2018. URL: https://www.postgresql.org/docs/9.6/static/using-explain.htmlhttps://www.postgresql.org/docs/9.4/static/intro-whatis.html (visited on 19/05/2018).

[50] European Space Agency. *Sentinel-2 - Data Products - Sentinel Handbook*. 2015. URL: https://sentinel.esa.int/web/sentinel/missions/sentinel-2/data-productshttps://sentinels.copernicus.eu/web/sentinel/missions/sentinel-2/data-products (visited on 03/06/2018).

[51] Mikael Rittri. *File:MGRSgridSouthPole.png - Wikipedia*. 2017. URL: https://en.wikipedia.org/wiki/File:MGRSgridSouthPole.png (visited on 02/06/2018).

[52] The PostgreSQL Global Development Group. *PostgreSQL: Documentation: 9.1: ARRAY function*. 2018. URL: https://www.postgresql.org/docs/9.1/static/arrays.html (visited on 04/06/2018).

[53] The PostgreSQL Global Development Group. *PostGIS: Spatial function ST_Intersects*. 2018. URL: https://postgis.net/docs/ST{_}Intersects.html (visited on 04/06/2018).

[54] Justinas Mišeikis et al. 'Multi-Objective Convolutional Neural Networks for Robot Localisation and 3D Position Estimation in 2D Camera Images'. In: (). URL: https://arxiv.org/pdf/1804.03005.pdf.

[55] Marco Balduzzi and Kyle Wilhoit. 'A Security Evaluation of AIS'. In: (2014). URL: https://www.trendmicro.de/cloud-content/us/pdfs/security-intelligence/white-papers/wp-a-security-evaluation-of-ais.pdf.

[56] Xingrui Yu et al. 'GIScience & Remote Sensing Deep learning in remote sensing scene classification: a data augmentation enhanced convolutional neural network framework ARTICLE Deep learning in remote sensing scene classification: a data augmentation enhanced convolution'. In: (2017). DOI: 10.1080/15481603.2017.1323377. URL: http://www.tandfonline.com/action/journalInformation?journalCode=tgrs20https://doi.org/10.1080/15481603.2017.1323377.

[57] Henning Heiselberg. 'A direct and fast methodology for ship recognition in sentinel-2 multispectral imagery'. In: *Remote Sensing* 8.12 (2016), pp. 1–11. ISSN: 20724292. DOI: 10.3390/rs8121033.

Appendix A

Tools and Workflow

A.1 Choice of Tools

In this section, we will briefly cover the different tools we have used in this study. In Section A.2, we will explain how we combine these different tools in our workflow.

Python
The primary programming language will be Python. It is easy to use and is supported by the other tools to be used in this study. There exist numerous good data analysis packages for Python, and because of this, it is very popular for this kind of work.

GDAL
GDAL, or Geospatial Data Abstraction Library, is an open source library for reading, writing, translation and processing of raster and vector geospatial data formats. It is released under the X/MIT license by the non-profit Open Source Geospatial Foundation (OSGeo). It is free, broadly supported and updated frequently, making it the go-to geospatial library for most applications. Programs such as Google Earth, QGIS, PostGIS and several others make use of GDAL's huge set of functionalities. In practice, most open-source geospatial frameworks and libraries are in some way based on GDAL. They have an API for Python, as well as for most other programming languages.

PostGIS
PostGIS is an open source spatial database extension for the PostgreSQL database man-

agement system. It adds support for geographical objects allowing spatial queries to be run in SQL. Based on GDAL, it has a wide range of spatial functions, optimised to query spatial data types stored in the database. Spatial data types refer to shapes such as point, line and polygon. It also allows spatial indexing for efficient processing of these operations.

Pandas

Pandas is an open source Python library for data manipulation and analysis. It provides high-performance data structures designed to make working with relational and labelled data easy. It is well suited for many kinds of data, including SQL tables as we will get from PostGIS queries. As it is fast, it is well suited for Big Data analysis.

Jupyter Notebook

The majority of the code, both Python and SQL, is written in a Jupyter Notebook. This is a web application that allows one to create documents that contain computer code (such as Python), programming outputs, visualisations and explanatory text. In this way, the notebooks are both human-readable containing analysis description and the results as well as executable documents which can be run to perform data analysis. It works well with Pandas and is excellent for, among other things, data processing and machine learning. One of the advantages is that one can divide the programming code into cells, and run each cell individually while keeping variables from other cells in the memory. This permits making changes in the code without having to re-run the whole script, which is excellent especially for time-consuming operations.

Sentinelsat

As mentioned shortly in 2.3.3, Sentinelsat [17] is a third-party API for Sentinel imagery. It is written in Python and is very well documented, making searching, downloading and retrieving data easy. It is especially well suited for retrieving image metadata. The API offers ease of filtering data products using query parameters, such as geographical footprint, time intervals and cloud cover percentage.

A.2 Project Workflow

In this section, we present the whole workflow, in every step of the data processing. An outcome of designing a good workflow is a corresponding orderly and tidy data flow that can be easily maintained and fitted to our needs. The resulting data processing pipeline is illustrated in Figure A.1. However, the primary focus of this section will be on the workflow of the main analysis.

To achieve the current workflow, a lot of time and work has been devoted into the system set up and configuration, for everything to work together flawlessly. We will not go in detail into every different configuration necessary to make, or all the challenges that arose. However, we want to provide a small example. Although *Jupyter Notebook* is incredibly robust and easy to use, we experienced some important challenges in configuring it for teamwork. Challenges include connecting the Python virtual environment to the notebook, as well as making it support cooperation using *Git*. Git is what was used for version control and code merging through GitHub.

The four primary sources of raw data are the AIS messages, the ESA Sentinel-2 acquisition plans, the MGRS-tiles and, of course, the Sentinel-2 images with its metadata. The AIS messages, acquisition plans and MGRS-tiles are processed and fed into a PostGIS database on our local Ubuntu machine. We have set up an SSH Tunnel to the Ubuntu to seamlessly connect to the database from remote machines. The Sentinel-2 images and metadata are not stored in the database but are queried directly from ESA on the fly. In this way, we have a data platform for the following analysis.

We have integrated and automated the major steps of the analysis in Jupyter Notebooks. In this way, the program takes two inputs: an MGRS-tile name, describing the area of interest, and a date-time timestamp. Based on these two inputs it queries and downloads the metadata of the corresponding Sentinel-2 image. It then runs all the steps sequentially, outputting, in the end, the extracted ship regions of the image, and ship positions in the image estimated from the AIS messages.

With Jupyter Notebook, we have the advantage of splitting the code into cells that can be run separately, and that store the instantiated variables in memory. As several of the steps in the analysis are computationally demanding, this is a significant advantage. We can do minor changes in the code, and then re-run only these parts without having to re-run all the heavy computations. It also displays well-structured outputs from each cell directly in the notebook, that let us quickly test small changes in the code.

To structure and represent the data in the notebook we take advantage of *Geopandas Data Frames*, an extension to the *Pandas* library. It can be seamlessly integrated with PostGIS, such that resulting rows from queries are structured in data frames directly. In a Jupyter notebook, these can be easily visualised in tables, and the spatial data can be efficiently plotted on maps. Having all this within the notebooks provides a quick and straightforward way of examining the attributes and their values. It also has necessary spatial operations, such as conversion of geometries between projections. The image metadata is also structured using data frames, letting us easily filter on the appropriate attributes to be used in the method that estimates the image timestamp. Following this estimation, the ships positions within the image are estimated within the database and structured in a data frame consisting of the point geometries as well as relevant ship attributes from AIS.

Not before any ships are found within an images' spatial bounds, the actual Sentinel-2 image are downloaded. If no ships are found, the image is of no interest, and the script will not waste time and storage space on downloading it. When finished downloading, the image is processed to convert the ship coordinates to pixel coordinates and extract the corresponding regions. To keep the data structure consistent throughout the whole script, the image regions and pixel coordinates are inserted into the data frame containing the estimated ship positions. In the end, using the Python library *matplotlib*, the full-size image marked with ship positions are plotted within the notebook, along with all the extracted ship regions.

Our workflow brought us closer to the data and feels more responsive and robust, compared to other tools we have experience with. We can recommend these tools for similar projects based on R&D by data analysis.

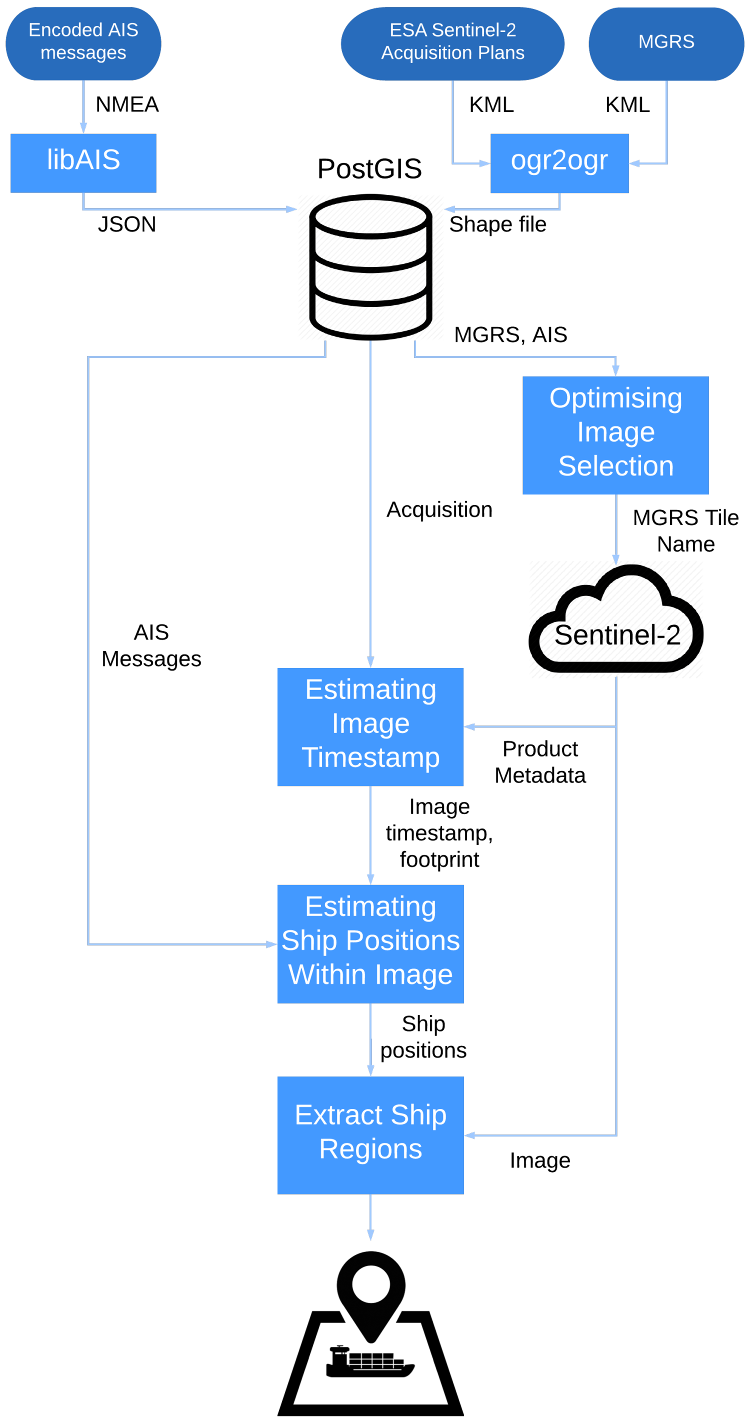


Figure A.1: Data processing pipeline for the project

Appendix B

SQL Queries

Code Listing B.1: Estimate image timestamp label

```sql
WITH -- Define input variables
input AS (
  SELECT product_footprint
  ,      datatake_sensing_start
  ,      platform_serial_identifier -- Sentinel-2A/B
), -- Find acquisition containing the product
spanning_aquisition AS (
  SELECT geom
  ,      start_time_utc
  ,      end_time_utc
  ,      satellite
  ,      ST_NPoints(geom)-1 as numPoints
  FROM acquisition, input
  WHERE datatake_sensing_start
    BETWEEN start_time - interval '15 second'
        AND end_time   + interval '15 second'
    AND satellite = platform_serial_identifier
),



[ Continued on the next page ]
   -- Find the end points of the
```

```sql
    -- acquisition center line
end_points AS (
  SELECT ST_PointN(geom, 1) as start
  ,      ST_PointN(geom, (numPoints >> 1) + 1) as end
  FROM spanning_aquisition
), -- Find the closest point on the acquisition
    -- center line to the product centroid
centerline AS (
  SELECT ST_ClosestPoint(
              ST_MakeLine(start, end)
         ,    ST_Centroid(product_footprint)
         ) as crosspoint
  FROM end_points, input
), -- Compute distances along WGS84 ellipsoid
distances AS (
  SELECT ST_Distance(start_point::geography
         ,           end_point::geography
         ) as datatake_length
  ,      ST_Distance(start_point::geography
         ,           crosspoint::geography
         ) as crosspoint_dist
  FROM end_points, crosspoint
) -- Finally estimate the image timestamp
SELECT  start_time
  + ( crosspoint_dist
    / datatake_length
    * (start_time_utc - end_time_utc)
  )  as estimated_timestamp
FROM spanning_aquisition, distances;
```

Code Listing B.2: Interpolate ship positions within image label

```
-- Define input attributes:
-- Image footprint
-- Our image timestamp estimation
-- Time margin to filter AIS messages
WITH
input AS (
  SELECT image_footprint
  ,      image_timestamp_utc
  ,      time_margin
),
-- Find all ship messages within time margin
ships_within_timerange AS (
  SELECT a.mmsi, a.timestamp_utc, a.geom
  FROM ais_position AS a
  ,    input
  WHERE  a.timestamp_utc BETWEEN
    input.timestamp_utc - time_margin AND
    input.timestamp_utc + time_margin
  -- Optional filtering
    AND accuracy = True
    AND speed > 0
),
-- Find all ship messages also within image footprint
ships_within AS (
  SELECT a.mmsi, a.timestamp_utc, a.geom
  FROM ships_within_timerange AS a
  ,    input
  WHERE ST_Within(a.geom, input.footprint)
),








[ Continued on the next page ]
```

```sql
-- For each distinct ship, find its nearest message
-- prior to the image timestamp
prior AS (
  SELECT *
  FROM (
    SELECT
      ROW_NUMBER () OVER (
          PARTITION BY mmsi
          ORDER BY a.timestamp_utc DESC
      ) AS r
    , a.*
    FROM ships_within as a
    ,    input
    WHERE input.timestamp_utc >= a.timestamp_utc
  ) x
  WHERE
    x.r <= 1
),
-- For each ship, find its nearest message
-- past of the image timestamp
past AS (
  SELECT *
  FROM (
    SELECT
      ROW_NUMBER () OVER (
          PARTITION BY mmsi
          ORDER BY a.timestamp_utc ASC
      ) AS r
    , a.*
    FROM ships_within as a
    ,    input
    WHERE input.timestamp_utc <= a.timestamp_utc
  ) x
  WHERE
    x.r <= 1
)

[ Continued on the next page ]
```

```sql
-- Interpolate each ships position
-- at the image timestamp
SELECT
  prior.mmsi
, prior.timestamp_utc as prior_t
, past.timestamp_utc as past_t
, ST_Project(
    prior.geom::geography
  , ST_Distance(prior.geom::geography
              , past.geom::geography)
      / EXTRACT(EPOCH FROM
          (past.timestamp_utc - prior.timestamp_utc)
        )
      * EXTRACT(EPOCH FROM
          (input.timestamp_utc - prior.timestamp_utc)
        )
  , ST_Azimuth(prior.geom, past.geom)
  ) as geom
FROM prior, past, input
WHERE prior.mmsi = past.mmsi;
```