Master Project IPSA

Introducing BOATMAN: A Satellite-Based System for Monitoring and Analyzing Maritime Traffic

Figure 1: Sentinel-1 is a radar imaging satellite operated by the European Space Agency (ESA). Source: [18]

Supervisor: Romain Artigue
Students: Vincent Boulenger and Lucas Chollet

Defense: 26 January 2023

Contents

List of Figures

List of Tables

Introduction

Satellite tracking has become an increasingly important tool for monitoring and analyzing the movements of boats around the world. With the proliferation of satellite technology, it is now possible to track the location and movements of boats in real time, providing valuable insights for a wide range of stakeholders, including shipping companies, national governments, and even private individuals.

One of the primary uses of satellite tracking for boats is to enhance maritime safety and security. By continuously monitoring the location and movements of boats, it is possible to identify potential safety hazards and take action to prevent accidents or incidents. For example, satellite tracking can be used to monitor the movements of boats in restricted or dangerous areas, such as near shipping lanes or in areas prone to piracy. It can also be used to track the movements of boats during emergencies, such as storms or oil spills, to ensure that appropriate rescue and response measures are taken.

In addition to its safety and security applications, satellite tracking of boats can also be used for a variety of other purposes. Shipping companies can use satellite tracking to optimize their routes and reduce fuel consumption, while governments can use it to monitor the movement of boats within their territorial waters and enforce regulations. Private individuals, such as boat owners or enthusiasts, can also use satellite tracking to monitor the movements of their own boats or to keep track of the movements of other boats of interest.

This paper presents BOATMAN for BOAT Monitoring and ANalysis. It is a system for monitoring and analyzing the movements of boats around the Earth. Using Synthetic Aperture Radar (SAR) data from the European Space Agency's Sentinel-1 satellite mission, BOATMAN is able to track the location and movements of boats in real time. The system is composed of a server that downloads, preprocesses, and processes the SAR data to write boat positions in a database, as well as a client-side that displays the boat positions on a web-based Geographic Information System (GIS). In this report, we will delve into the technical details of BOATMAN, including the data sources, processing methods, and visualization tools used to create this maritime tracking system. The code is fully available on GitHub: BOATMAN

1 Data Selection for BOATMAN

1.1 Building a GIS System for Boat Monitoring using Satellite Imagery: Characteristics, Key Requirements, and Data Sources

To create a GIS system for monitoring boats utilizing satellite imagery, we need to rely on data that provides information about the location, speed, direction, size, and type of boats. While the position is essential, having additional information about these characteristics can provide a greater understanding of boat movement and behavior. To this end, having a good source of data is primordial to the project.

1.1.1 Key Requirements

As an introduction, we can list several essential requirements that satellite imagery must meet to be able to monitor boats:

- ⋆ High resolution: The satellite imagery must be precise enough to identify and track boats.
- ⋆ Frequent coverage: The satellite imagery must be updated frequently enough to provide real-time or near real-time monitoring of boat movements.
- ⋆ Wide coverage: The satellite imagery must cover the areas where the boats are operating to track their movements.
- ⋆ Multi-spectral: The satellite imagery must allow the detection of ships in different lightning and sea conditions.
- ⋆ Access to the data: Accessing the data for free is essential for a student project without funding.

1.1.2 Possible Data Sources

There are actually several known and functional imagery data sources that respect most of the requirements mentioned above. All of them are at least a partial fit for our project, but we had to focus on only one to start experimenting.

The Copernicus Program The European Union's Copernicus program provides free and open access to satellite imagery for a broad spectrum of applications. It also has the advantage of providing different types of high-resolution data using the Sentinel constellation.

Landsat Landsat is a long-term satellite imagery dataset that is freely available for use. It is the first spatial observation program on Earth for civil applications developed by NASA during the 1960s.

Commercial providers There are also many commercial providers of satellite imagery, such as DigitalGlobe, Planet, and Airbus. They all offer high-resolution imagery for purchase.

Other satellites We can cite the MODIS instrument on the Terra and Aqua satellites from NASA and TerraSAR-X and TanDEM-X satellites from ESA.

1.1.3 Data Source Selection for the GIS System

When choosing our data source, we ruled out commercial providers as it was not feasible for our project. We then compared the resolution, revisit time, and coverage of other available data sources. The TerraSAR-X and TanDEM-X satellites were the best option for our application as they provide the highest resolution, down to 2 meters [21]. However, the data from these satellites is not directly accessible, and we would have needed to submit our project and be accepted by the European Space Agency (ESA), which was not possible due to time constraints at the start of our project. The Moderate Resolution Imaging Spectroradiometer (MODIS) instrument did not have a high enough resolution for our needs [13]. Ultimately, we decided to use data from the Copernicus program, as it provided the best resolution and revisit times [10, 1].

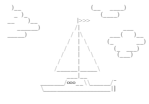
1.2 Overview of the Copernicus Program

Copernicus is the European Union Earth Observation Program, a joint initiative between the European Commission and the European Space Agency (ESA). The mission aims to provide accurate and reliable data about the Earth's environment and climate, including information about the atmosphere, oceans, land surface, and cryosphere. The mission uses satellites and in-situ sensors to monitor and measure various parameters, including temperature, humidity, precipitation, ocean currents, ice cover, and vegetation. Vast amounts of global data from satellites and ground-based, airborne, and seaborne measurement systems provide information to help service providers, public authorities, and other international organizations improve European citizens' quality of life and beyond. The information services provided are free and openly accessible to users.

Data from the mission is used by interested parties, including policymakers, scientists, and the public, to support some applications, such as weather forecasting, climate modeling, natural disaster management, and environmental monitoring. The mission is ongoing, with new sensors and capabilities being added over time to expand the scope and accuracy of the data collected [1].

In our case, we will use data from the sentinel satellites. The Copernicus program specifically develops this fleet of satellites for its needs. There are currently six Sentinel satellites in operation, with more planned in the future. The Sentinel-1 satellites' instruments are Synthetic Aperture Radar (SAR) instruments that can observe the Earth's surface in all weather conditions, day or night. The Sentinel-2 satellites carry multi-spectral optical instruments that capture high-resolution images of the Earth's surface, including details about land cover, vegetation, and urban areas. The Sentinel-3 satellites carry a suite of instruments to monitor the oceans, including sea surface temperature, sea level, and wave heights. The Sentinel-4 and Sentinel-5 satellites are designed for atmospheric monitoring, carrying instruments that measure trace gases, aerosols, and other parameters relevant to air quality and climate research [17].

In our project, we considered using data from both Sentinel-1 and Sentinel-2 satellites. While both missions have advantages and disadvantages, our focus was primarily on using Synthetic Aperture Radar (SAR) imagery from Sentinel-1. We choose this instrument because SAR imagery can provide information about the topography, surface roughness, and the presence of water, regardless of weather or lighting conditions. SAR imagery is notably valuable for monitoring boats in coastal and offshore areas, and for detecting vessels in rough sea conditions and heavy clouds. On the other hand, Sentinel-2 imagery provides detailed images of the Earth's surface using visible and near-infrared light sensors. Those sensors are practical for identifying features such as boats, harbors, and shipping lanes but are limited by cloud cover and weather conditions. Ultimately, we decided to prioritize the SAR imagery because it allows



for the detection of boats at all times, which is the most significant feature of our project.

1.3 Exploring the Capabilities of the Sentinel-1 Mission for BOATMAN

The Sentinel-1 mission comprises a constellation of two polar-orbiting satellites. Each satellite is equipped with a Synthetic Aperture Radar (SAR) instrument to observe Earth's surface. This mission has many benefits that justify its choice as our data source for boat detection [19].

1.3.1 Orbit and Geographical Coverage

Sentinel-1 is in a near-polar sun-synchronous orbit with a 12-day repeat cycle and 175 repeat orbits per cycle for each satellite. The two-satellite constellation can map the entire world every six days. The highest frequency of repeat acquisitions will be at the equator, where it will be possible to acquire images every three days (ascending and descending orbits).

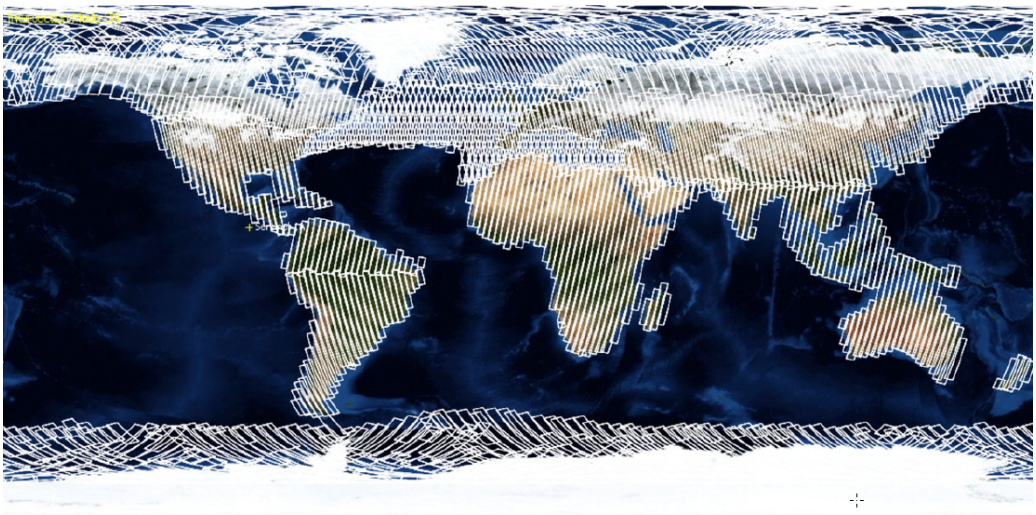


Figure 2: Coverage of the Sentinel-1 mission. Source: ESA [19]

The mission should provide coverage on the main shipping routes in one to three days. The coverage and revisit time of the Sentinel-1 mission make it a good choice for our project, as it will allow us to monitor boats in near real-time.

1.3.2 The C-SAR Instrument

Sentinel-1 satellites carry a single C-band synthetic aperture radar instrument operating at a center frequency of 5.405 GHz.

SAR is a type of active data collection where a sensor mounted on a moving platform transmits microwaves and then records the back-scattered signals and their time delays after interacting with the Earth. The strength and time delays of the signals both depend on the physical properties of the observed surface (such as its roughness and electrical conducting properties). Those signals are then processed to build the image. SAR specifically is a technique for producing high-resolution images from systems that are physically limited in resolution.

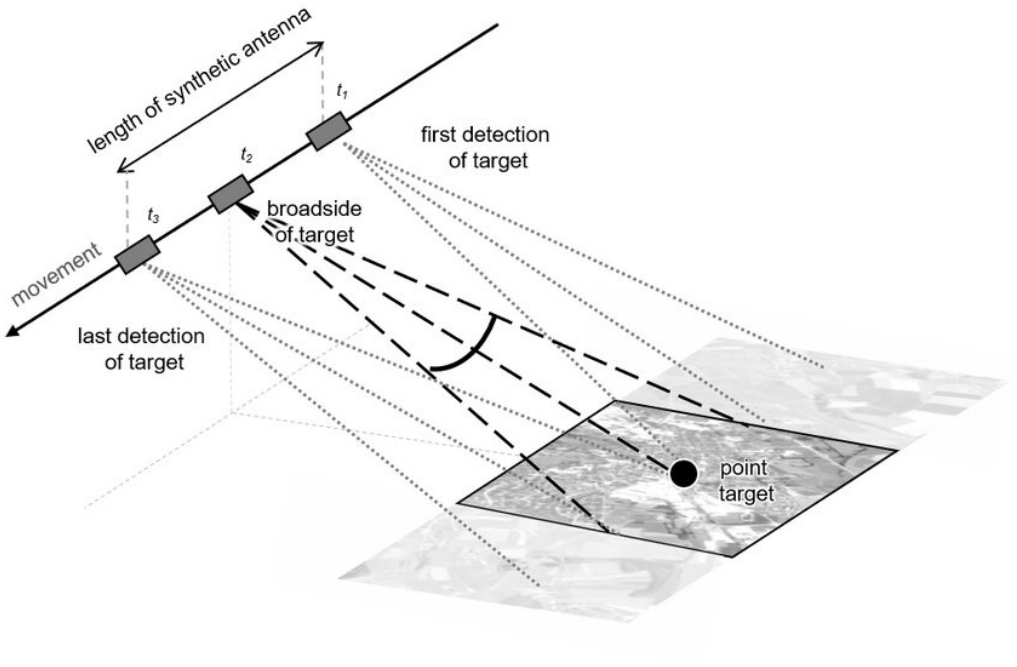


Figure 3: Principle of the synthetic aperture radar (SAR). Source: J.Braun, 2019 [4]

Remote sensing radars use microwaves, typically wavelengths ranging from a few to tens of centimeters, to observe the Earth's surface. However, as the radar signal travels through the atmosphere, it loses energy, resulting in a beam width (wavelength divided by antenna size) that becomes too large for most applications. Synthetic Aperture Radar (SAR) is a solution to this problem, as it can significantly improve the resolution of the resulting images without increasing the length of the physical antenna.

The forward motion of the platform enables the transmission and reception of signals to the same object from successive sensor positions. The distance between the radar and a specific point on the ground is constantly changing. The phase of the returned signal pulse reflects this variation. By compensating for the phase history of each pulse recorded for a given point, it is possible to focus the signal through computer processing, effectively creating a "synthetic aperture" that is much longer than the physical antenna. The resulting image resolution can be improved to a theoretical one-half of the antenna's diameter, significantly improving the resolution of the images.

One of the main advantages of SAR is that it is effective at wavelengths hindered by cloud cover or a lack of illumination. It means that data is acquired regardless of the weather conditions or time of the day [14, 15].

1.3.3 Polarization

Polarization refers to the direction in which the tip of an electromagnetic wave's vector travels. The orientation of the wave's electric field determines this direction, which can be vertical, horizontal, or circular. Radar systems can be designed to transmit and receive electromagnetic waves with a specific polarization. By using Synthetic Aperture Radar (SAR) to transmit signals with varying polarizations and receive multiple polarized images from the same series of pulses, it is possible to gather detailed information about the polarimetric properties of the observed surface.

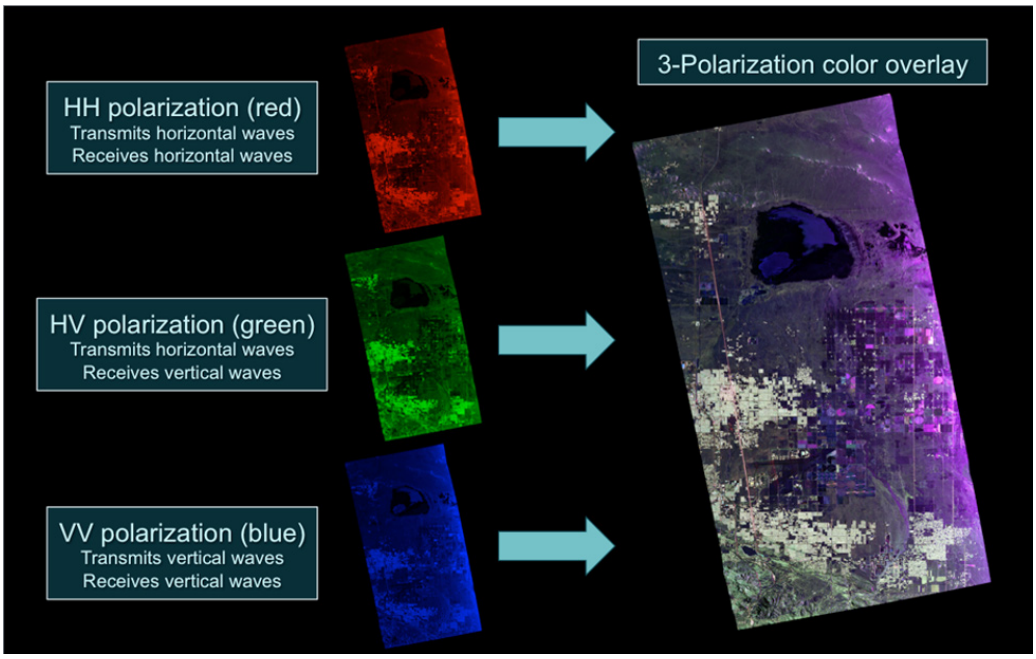


Figure 4: An example of polarimetry imagery from the airborne UAVSAR instrument obtained over Rosamond, California. Source: NASA/JPL-Caltech

This information can reveal the structure, orientation, and environmental conditions of the surface features. Different combinations of polarizations and wavelengths can provide complementary information about the surface. The C-SAR instrument from Sentinel-1 provides dual-polarization capability, which means it can transmit in one polarization and receive in two, resulting in either HH and HV or VH and VV imagery. Dual polarization provides additional detail about surface features through the different and complementary echoes. The dual polarization can be helpful for differentiation between different types of boats or other objects in the water [16].

1.3.4 Resolution and Swath

The high resolutions available (from 80 meters and down to 5 meters) and its extensive coverage (from 80 kilometers to 400 kilometers) [22] allow for the detection of relatively small objects (such as boats) on large areas in a single pass.

1.3.5 Data Products and Access

Radar data from the Copernicus program is made available within one hour of its acquisition, which is crucial for building our system. This one-hour turnaround time is considered acceptable for near real-time applications. Furthermore, access to the data is freely available to all users, making it a cost-effective option for boat detection and tracking. It is a significant factor, particularly for student projects and prototypes.

2 Methodology and Data Processing

Now that we have identified the source of our data, we need to plan for how we will acquire, process, and store it.

2.1 Automating Data Retrieval and Processing with Copernicus API and sentinelsat package.

As mentioned in the previous section, we use the Copernicus mission to obtain our data. To be authorized to access their services, we need to create an account. Once that is done, we can make our way through the Copernicus API - OpenHub [2] and download products on our computer.

However, as we want to automatize this task, we need to use the Application Programmable Interface (API). The ESA provides multiple APIs [3] to access its data, but it won't impact us too much as we will interact with it through abstraction.

Indeed, the Python package sentinelsat can accomplish this task conveniently. Instead of interacting with a verbose REST API, sentinelsat allows us to fulfill a query and download products in a few lines of code. In the annex B, you can find a simplified version of what we implemented for BOATMAN that will download the latest product corresponding to a region.

These products contain all the information we need to process them: the data - the actual radar image and the metadata about the provenance. Both information will be processed and used, as described in the following parts.

As a note, each product has a size of approximately one gigabyte. The download will easily take up to one minute or more, depending on the server's internet connection. This time cannot be shortened and is a critical part of the total response time.

2.2 Processing data using the Sentinel Applications Platform

To ensure the scientific quality of this project, we wanted to base our work on some existing results. Our work is based on Marzuki et al. (2021) [12], and a Reasearch and User Support (RUS) Copernicus training guide [20]. In their paper, Marzuki et al. use ESA's Sentinel Applications Platform (SNAP) to process the data. This project is a toolbox for satellite data processing, and it contains all features we needed to go ahead.

To allow a user-customized utilization, SNAP provides an abstraction called a "Graph". These graphs are composed of "Operators", which represent atomic operations. Every graph starts with an "Open File" node and ends with the "Write File" node.

The user can customize the core of the graph as per their preference. We will now detail what we did for BOATMAN. The total graph is visible in Figure: 5.

2.2.1 Graph Creation with snapista

On a side note, to create the graph automatically, we are using Python's package snapista. It only enables the creation of individual operators and assembles them into a final output file. Then it calls gpt, a command-line-based tool to use SNAP, and processes the graph.

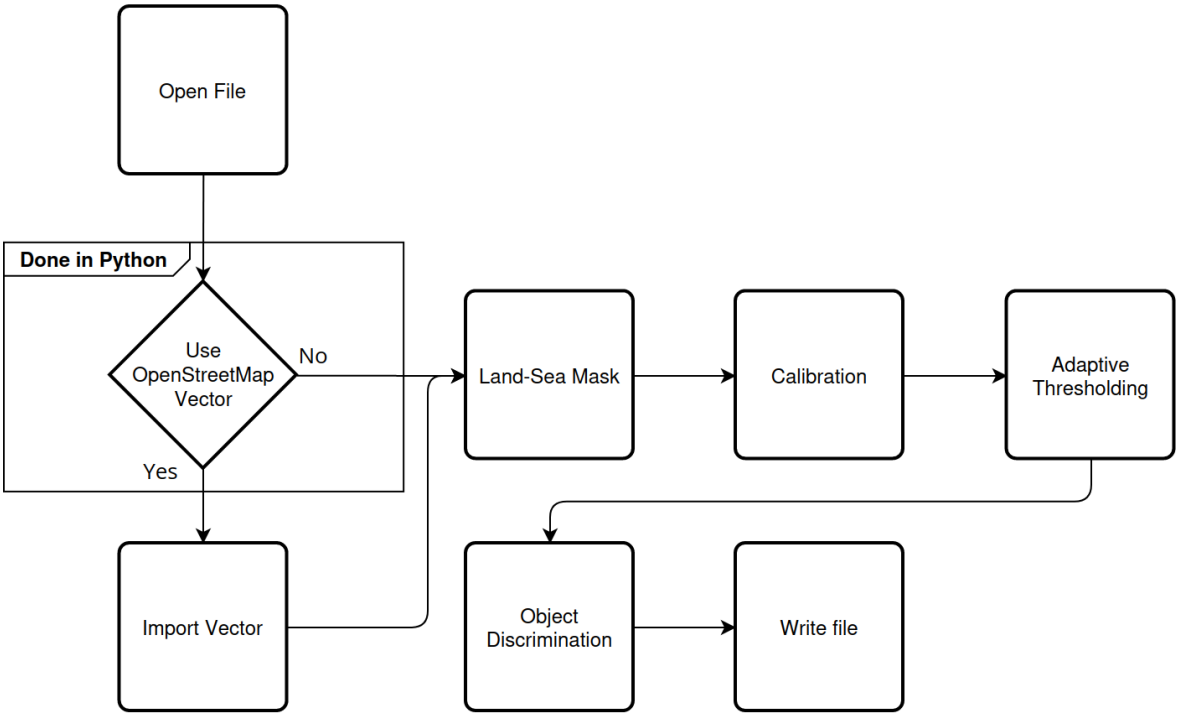


Figure 5: Flowchart of SNAP's graph used for BOATMAN.

2.2.2 Land-Sea Mask

The first computational operator is the "Land-Sea Mask". A land-sea mask is a binary dataset that separates land from water bodies. It is a raster dataset, with each cell in the grid being assigned a value of 1 for land and 0 for water. The resolution of the mask can vary, but more on that later. The aim is to hide a part of an image that we don't want to analyze, and for BOATMAN, we use it to remove the land from the image. Here is an example of applying a land-sea mask around Amsterdam on a SAR product:

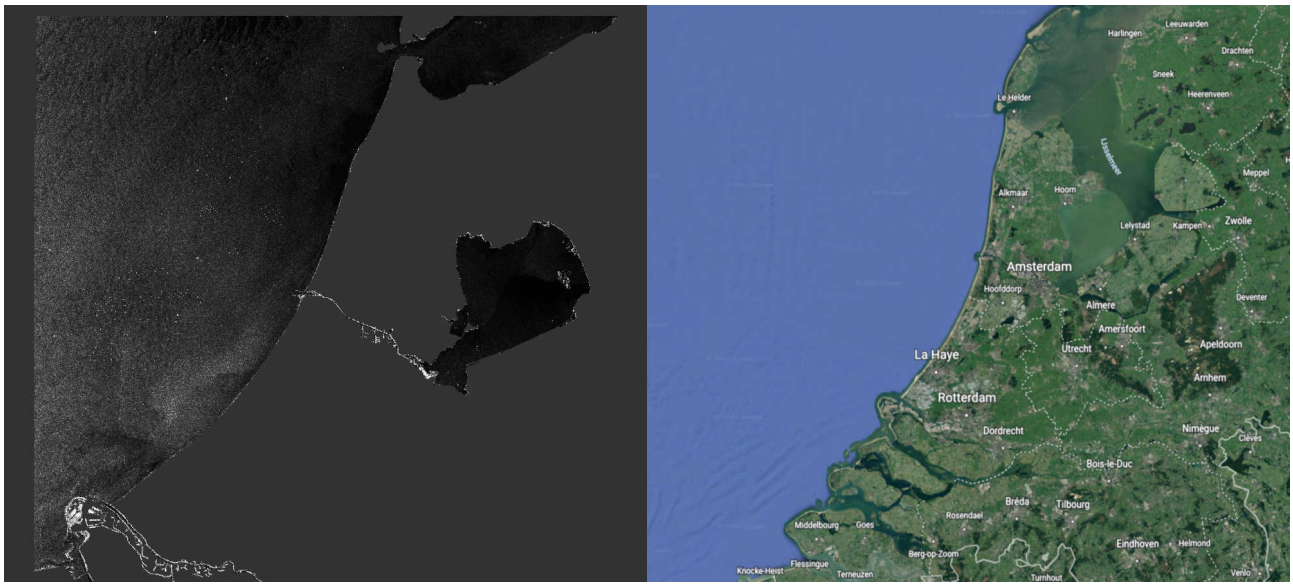


Figure 6: Result of applying a land-sea mask on a SAR product and comparison with a satellite image from Google Earth. Interestingly, the mask makes a difference between the sea and IJselmeer, a freshwater lake, only separated by a dam.

2.2.3 Radiometric Calibration

The second computational task is named "Calibration", here, we apply radiometric calibration to the image. SAR calibration's objective is to provide imagery in which the pixel values can be directly related to the radar backscatter of the scene. Typical SAR data processing, which produces level-1 images, does not include radiometric corrections, and significant radiometric bias remains. The radiometric correction is necessary for the pixel values to truly represent the radar backscatter of the reflecting surface and therefore for comparison of SAR images acquired with different sensors or acquired from the same sensor but at different times, in different modes, or processed by different processors [20].

This operator is a black box from SNAP that will automatically call the calibrator precisely created for the product type.

2.2.4 Adaptive Thresholding

We are now coming to the core of the graph. Indeed, "Adaptive Thresholding" is the operator responsible for detections. This operator is based straight on the description of the algorithm found in Crisp, David. (2004) [5]. The adaptive thresholding algorithm used in SNAP assumes that targets appear as bright on a dark background. It uses a moving window to analyze each pixel, determining a new threshold value based on the statistical characteristics of the local background. If the pixel value is above the threshold, it is classified as a target pixel. The specific algorithm used is a Two-Parameter Constant False Alarm Rate (CFAR) Detector. The user can set the parameters of the moving window, which includes the Target window, Guard window, and Background window. The Target window corresponds to one or multiple pixels under test, the Guard window is used to prevent contamination of the background values by the target pixels, and the Background window represents the local background. The window size and detection criterion are defined by user input. We use the following configuration:

Parameters	**Values**
Target window size (in meters)	50
Guard window size (in meters)	500
Background window size (in meters)	800
PFA	12.5

Table 1: Parameter choices for the Adaptive Thresholding operator.

The user parameters are used to determine the probability distribution function (PDF) of a fitted Gaussian distribution. Then the PDF and user-defined probability of false alarm (PFA) is used to determine the detection design parameter t as follows:

$$PFA = \frac{1}{2} - \frac{1}{2} erf\left(\frac{t}{\sqrt{2}}\right)$$

The decision criterion is then expressed as: $x_t > \mu_b + \sigma_b$ if the target window is a single pixel, or as: $\mu_t > \mu_b + \sigma_b$ if the target window contains multiple pixels. Where x_t is the value of the target pixel, or μ_t is the mean value of the target window, μ_b is the background mean and σ_b is the background standard deviation [20].

2.2.5 Object Discrimiation

The final operator in our process is the "Object Discrimination" operator. The CFAR algorithm can produce a significant number of false positives, which is a problem that needs

to be addressed both within the SNAP tool and later in Python. This operator serves as a filter to remove any eccentric detections. Objects that fall outside of certain size boundaries are automatically discarded. The current parameters for this operator are a minimum size of 30 meters and a maximum size of 600 meters.

2.2.6 Additional Considerations

Note on Improving Land Mask

A good mask is crucial for this project as the land often stands out next to the sea and results in false positives, almost all the time. We found two solutions, but both of them have impactful drawbacks. The first one uses the shorelineExtension parameter of the operator. This operator pushes the shoreline to cover more of the land. It is overly beneficial but substantially impacts performance as it slows down the process. The other solution is to use another source for the mask, and OpenStreetMap provides one [6]. This mask is more precise on the shoreline but doesn't include as many estuaries and harbors, which are very important to us.

Note on Width and Length Detections

To compute the width and the length of detections, the algorithm tries to fit the detection in an axis-aligned-bounding box. It means a ship heading at 45°against a parallel or meridian will appear as a square.

2.3 Saving Results on Persistent Storage

Now that we have some results, we need to store them in a way so we can easily retrieve them. In this part, we will first discuss how we read the results from gpt, then the data scheme we used, and finally, the advantages of using a SQL database.

2.3.1 Extracting Information

As gpt is writing its output to the disk, we had to write functions to retrieve the information. We tried to change the output file type in the graph to ease this task, but it turned out to be very inconsistent. When requesting, for example, a NetCDF4 or even a simple CSV file, gpt will sometimes spin on the operation and never write the result to the disk. Only the default format, named BEAM-DIMAP, seems to give consistent results. So, we are left with a BEAM-DIMAP file to parse.

The BEAM-DIMAP Format As we can see in figure 7, using a file is an imprecision as the output is composed of a file .dim, which contains the principal metadata and a directory with the actual data, along with related metadata. The .dim file is a rebranded .xml with a bunch of information on the product. The policy for software products such as SNAP is to keep most data when outputting a result. It explains why we see all those .img[1] files, which came from the original product file. Last but not least, we can see a directory named vector_data. It contains additional data generated by operators. For this section, the more important one is ShipDetections.csv, which contains the detection results.

[1].img are data files and similarly named .hdr file contains their metadata

```
.
├── S1A_IW_GRDH_1SDV_20230122T034428_20230122T034453_046891_059F84_BBD3.data
│   ├── Sigma0_VH.hdr
│   ├── Sigma0_VH.img
│   ├── Sigma0_VH_ship_bit_msk.hdr
│   ├── Sigma0_VH_ship_bit_msk.img
│   ├── Sigma0_VV.hdr
│   ├── Sigma0_VV.img
│   ├── tie_point_grids
│   │   ├── elevation_angle.hdr
│   │   ├── elevation_angle.img
│   │   ├── incident_angle.hdr
│   │   ├── incident_angle.img
│   │   ├── latitude.hdr
│   │   ├── latitude.img
│   │   ├── longitude.hdr
│   │   ├── longitude.img
│   │   ├── slant_range_time.hdr
│   │   └── slant_range_time.img
│   └── vector_data
│       ├── ground_control_points.csv
│       ├── land_polygons.csv
│       ├── pins.csv
│       └── ShipDetections.csv
└── S1A_IW_GRDH_1SDV_20230122T034428_20230122T034453_046891_059F84_BBD3.dim
```

Figure 7: Tree of the files contained in a BEAM-DIMAP output.

Now that we have roughed out the task of retrieving the result, we know which files to read: the rebranded .xml and a csv.

Reading the .dim File Sine Python is a well-known and versatile language, we can easily find a library to parse and walk across the nodes of an XML tree. We even used a standardized module: xml.etree.

This module provides the ability to find a descendant easily. Using the findall method, we can also target multiple levels of belonging. As these files tend to be huge - roughly 30,000 lines, we can directly retrieve the correct node. Using each node described in Table 2, we fill almost all elements of the Tile[2] object. We infer the remaining elements either given by an external function (file path or current time) or deduced from these nodes.

Node name	Description	Example
PRODUCT	Product name	S1A_IW_GRDH_1SDV_20230122T034428 _20230122T034453_046891_059F84_BBD3
SPH_DESCRIPTOR	Product type	Sentinel-1 IW Level-1 GRD Product
PASS	Orbit type	DESCENDING
num_output_lines	Image height	16697
num_samples_per_line	Image width	25992
first_line_time	First zero doppler azimuth time	22-JAN-2023 03:44:28.538943
last_line_time	Last zero doppler azimuth time	22-JAN-2023 03:44:53.538668
PROC_TIME	Processed time	22-JAN-2023 06:29:36.760169

Table 2: Used node of the .dim file.

[2]The object is described in the subsection 2.3.2

Reading the .csv File Now that we have parsed the metadata of the Tile, we need to read the CSV file to retrieve all detections. Python's standard library also provides a module to read these files, again a considerably time-gaining solution.

The file provided by gpt is an XSV file, which means that values are not separated by a comma but by a tabulation. Yet, it is an easy issue as Python's csv.DictReader supports user-custom symbols. We only add a filter to remove commented lines, and that's it for the hard part.

Following the last step, we just needed to copy each relevant data to a Detection[3] object, and that's it. Those data are:

- ⋆ width and length - the dimension of the boat;
- ⋆ latitude and longitude - the position of the boat;
- ⋆ pixel_x and pixel_y - the position of the boat inside the image.

2.3.2 Data Scheme used in the Database

SQL databases can efficiently save high volumes of data or retrieve some particular subsets. So it makes a perfect fit for our project. However, the SQL syntax can be relatively hard to understand. We use SQLAlchemy to counterbalance it. This tool allows creating and managing database objects as if they were classic Python objects.

Now we are going to discuss the data scheme we used. It is often challenging to describe a database, but they are easy to understand when drawn. As a heads-up, we will shortly describe all used tables, but for more details, the reader should refer to this annex: C - UML Scheme of the Database. Moreover, the reader can also see the Python implementation with SQLAlchemy on GitHub.

The database is composed of three tables. One of them, the "ports" table, is static and contains the data for each port. It is created if absent from the database and filled from this file ports_unique.json[4]. The two remaining tables are "tiles" and "detections". Each attribute of the former represents a Copernicus product and contains its metadata. Each element in the latter corresponds to a detection with its position and size. These two tables are bound so we can retrieve each detection from a specific tile and find the tile parent of a detection.

2.3.3 Other Advantages to a Database

Using a SQL database has been a decisive choice for us. It relieves us from the burden of writing such a system. But it also comes with its advantages.

By this, we are mainly thinking of the support for the SQL language, a powerful tool to make complex requests to a database. From the beginning, we wanted to support it as a future opportunity to add features. However, its need started to appear before we thought.

In some cases, the CFAR algorithm, previously discussed, can return exact duplicates. It is a bug that needs to be fixed upstream, but we had to find a workaround to remove these from our system.

Even if this can be implemented directly in Python quite efficiently, it can also be done in a unique SQL request, as we can see in Source Code: SQL request to delete duplicates in the database.. This request will filter out any detections that do not match the list of all detections with distinct latitude and longitude coordinates.

[3] Again, it is described 2.3.2

[4] Its origin is explained in 3.1.2

```sql
DELETE *
FROM detections
    WHERE (detections.id
        NOT IN (SELECT detections.id
                FROM detections
                GROUP BY detections.latitude, detections.longitude
        )
    )
```

Source Code 1: SQL request to delete duplicates in the database.

And the SQLAlchemy version of this (which was implemented in aee1f34) is:

```python
duplicates = (
    db.query(Detection)
    .where(
        Detection.id.not_in(
            db.query(Detection.id).group_by(Detection.latitude,
            ↪ Detection.longitude)
        )
    )
    .all()
)

for duplicate in duplicates:
    db.delete(duplicate)
```

Source Code 2: Python equivalent of the request above.

3 The Geographic Information System

The purpose of the GIS component in BOATMAN is to provide a visual representation of the data collected by satellite imagery. The GIS allows for the easy identification and tracking of boats. It also gives the ability to view additional information, such as the size of the vessels. The GIS displays other relevant information, such as harbors, and coastlines, to help users understand the context of the boat movements.

Additionally, the GIS component provides interactive features that allow users to navigate the data and analyze the information in different ways, such as zooming in and out, panning, and querying specific data points. The user can also download the data directly from our server as a CSV file if they want to analyze the data with further depth. It should propose an intuitive and user-friendly interface for users to navigate and interact with the data.

Overall, the GIS component is an essential tool for understanding and interpreting satellite imagery data, enabling the monitoring of boats in near real-time.

Our web GIS consists of two components: the client and the server. The client side, which is executed on the user's machine, handles front-end processes. On the other hand, the server side is where backend scripts are run. The creation of these two components presented a significant challenge for us, which we will discuss in further detail later on.

3.1 Creating an Intuitive Web GIS: An In-Depth Look at Our Frontend Design

From the start of this project, we add the idea of creating a web GIS. Many famous GIS software exists as ArcGIS, QGIS, GRASS GIS, etcetera..., but they all need to be installed and run locally. For us, this project was more than just processing data. We aimed to make this data accessible to as many users as possible.

GIS software can be overwhelming for the public (as well as for us), and as such, we quickly jumped on the idea of developing our web GIS that could be accessible to everyone on the Internet once deployed. It also has the advantage of giving us a large margin for customization. Our web GIS would only have the required features that we would need. As for GIS software, a handful of web GIS development tools are available. We can cite Leaflet, OpenLayers, or the JavaScript API Esri ArcGIS.

Out of the tools we found, we decided to go with Leaflet. We chose Leaflet because of its lightweight and easy-to-use design. Contrary to other web-GIS tools, Leaflet allows developers with little GIS experience to create maps in just a few lines of code. While other tools, such as OpenLayers, may be better suited for larger projects, their advanced feature set was not necessary for our project, and the learning curve would have been steeper.

3.1.1 Designing a User-Friendly Web GIS: Our Approach and Implementation

This section will describe each UI feature implemented and justify its usage in the web GIS.

Basic Configuration of Leaflet

The Leaflet library creates the map background and mouse controls without us doing anything. We only had to specify which base maps Leaflet had to use. In this first screenshot, the base map used is Google Hybrid Satellites.

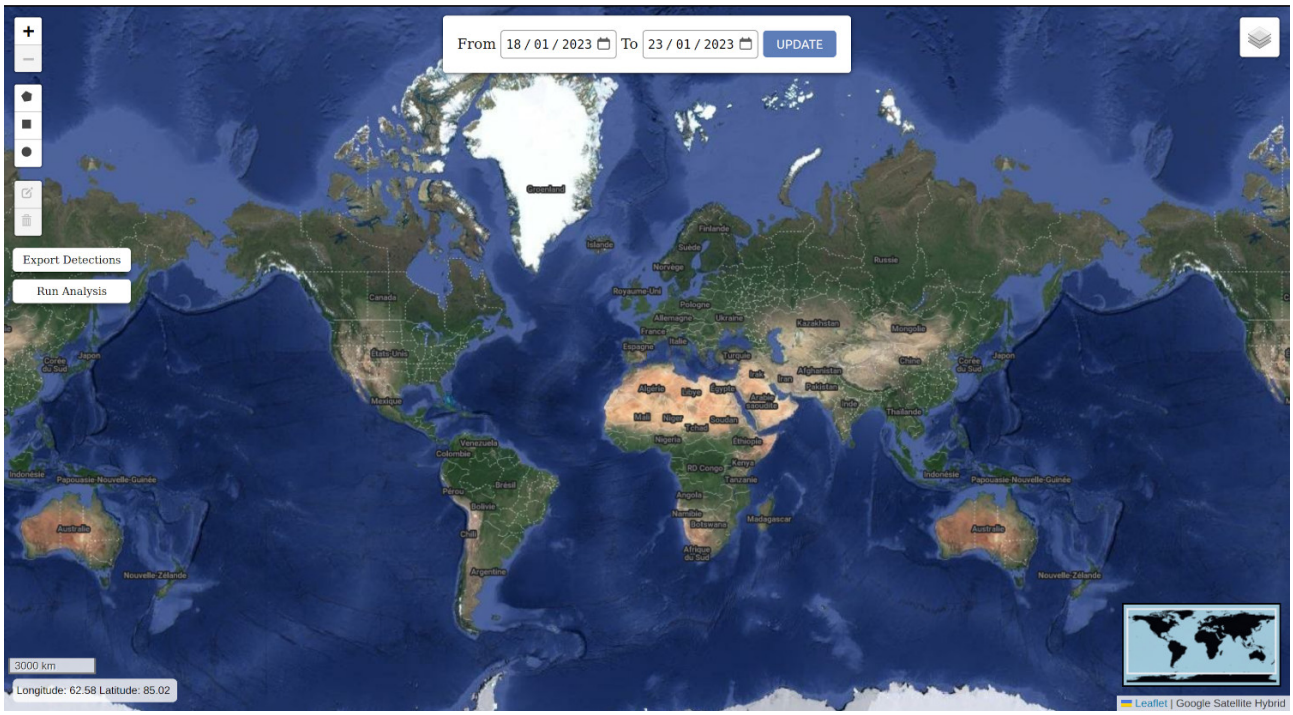


Figure 8: A first glance at the empty web GIS.

This map combines satellite imagery with road and labels overlay. We decided on this base map because it has high-resolution imagery, which is quite friendly for GIS users. Yet, it has some limitations. For example, the resolution is not that good on the ocean and seas. For this reason, we also decided to add the OpenStreetMap base map. It is an open-source and free base map that provides a detailed world map. This base map does not rely on images, meaning its resolution is limitless, even at sea. You will be able to see this base map in the following screenshots.

When creating the map, we set the default view to the one seen in the previous screenshot to allow the users to see the world in its globality. It is also easier to scroll to a region of interest, and they can quickly look at everything on the map.

Leaflet creates base map replicas on the left and right of the main base map displayed. We set the map boundaries to prevent the user from going on those maps (where there is no data).

Overlays

The Leaflet library also allows us to define overlays and layers of data. We can see this button in the top-right corner of the first screenshot.

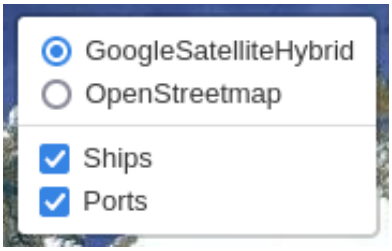


Figure 9: Base map and overlay selection.

Here the user can define which base map and which data to display.

World MiniMap

We decided to add a minimap in the bottom-right corner of the UI. We used the plugin leaflet.WorldMiniMap. It allowed us to add a small-scale map of the world, in which the actual view of the Leaflet map is shown:

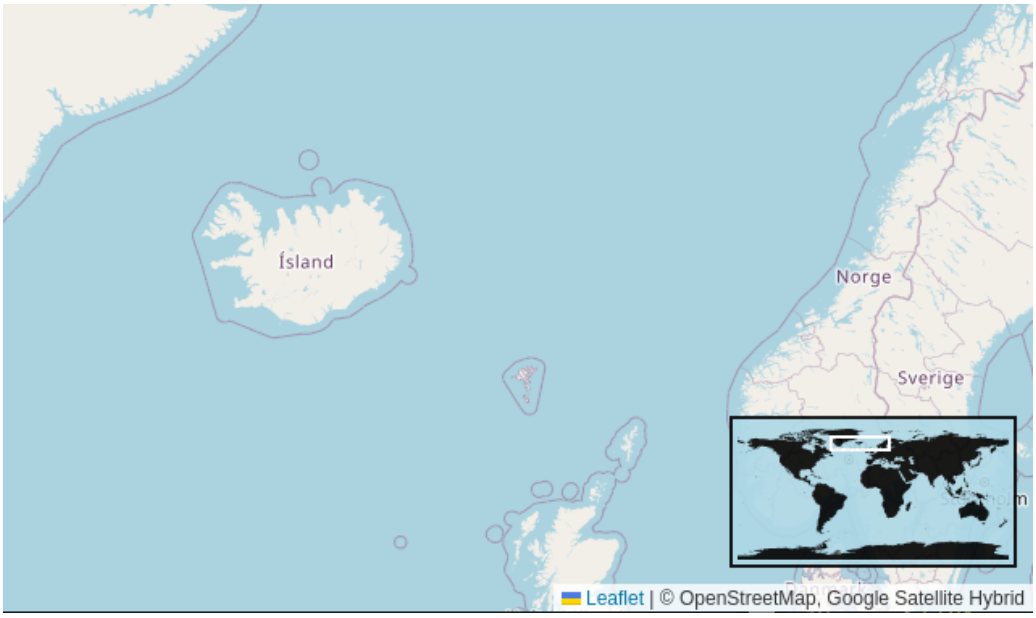


Figure 10: Map zoomed on Iceland, and minimap showing the region seen on screen (using OpenStreetMap base map).

Scale

Adding a scale to the UI seemed like an obligation to allow the user to know at which zoom level he is. It also helps to give the user a sense of the distance between locations, especially when zoomed in. This feature is also part of the basic leaflet library. We positioned the scale in the bottom-left corner:

Figure 11: Scale displayed under Iceland.

Mouse Coordinates

Leaflet.Coordinates is a plugin that lets us display the mouse coordinates at every instant. It can help the user quickly get a grasp of its position.

Figure 12: Mouse Coordinates.

It can also assist the user in knowing the location of a specific feature, which can be helpful for data collection. This plugin also can let the user define a temporary marker on the map. It can be rather practical for visualizing the data around a given point with precise coordinates.

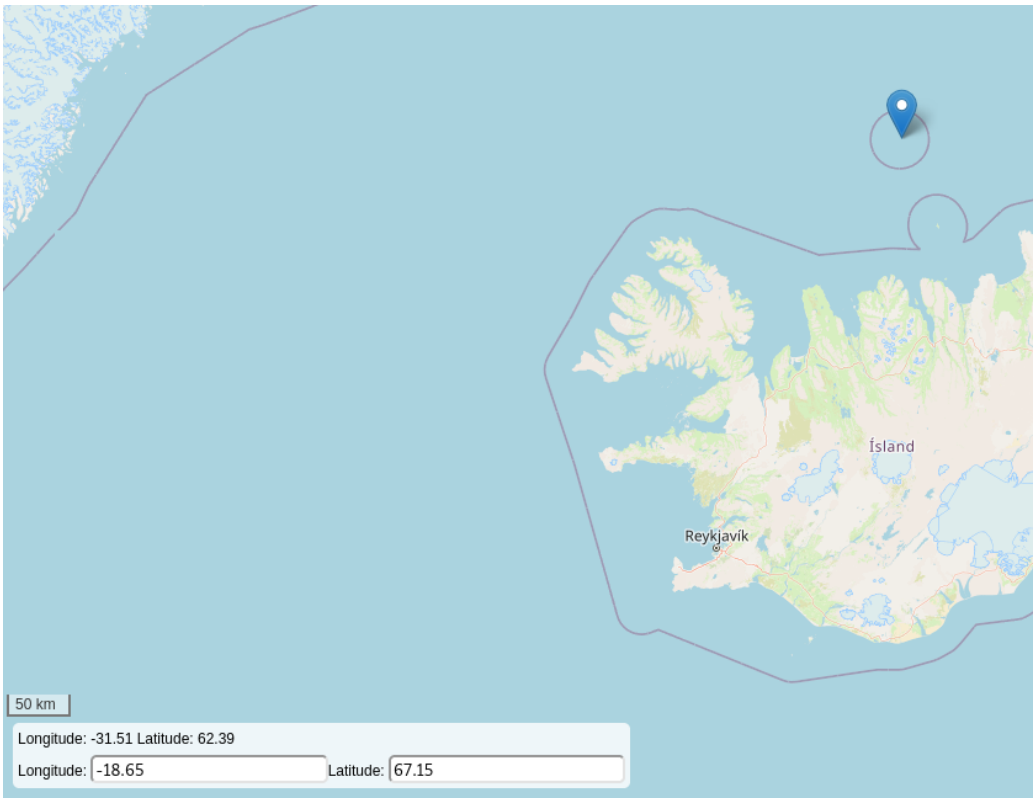


Figure 13: Adding a marker over Kolbeinsey, a little island north of Iceland.

The Draw Toolbox and Additional Buttons

On the top-left corner, we added a draw toolbox and some buttons. The plugin Leaflet.Draw allows the user to draw polygons on the map. The toolbox lets the user draw a circle, a rectangle, or a polygon. We limited to only one polygon drawn on the map at a time. The primary intent behind this toolbox is to let the user run data analysis on specific regions while the server is not running all the time. The Draw toolbox also allows the user to edit the drawn polygon. The user can utilize the buttons below the toolbox to detect boats in a specified region and download the data. The Export Detection buttons let the user download a CSV file with all the detections displayed on the map.

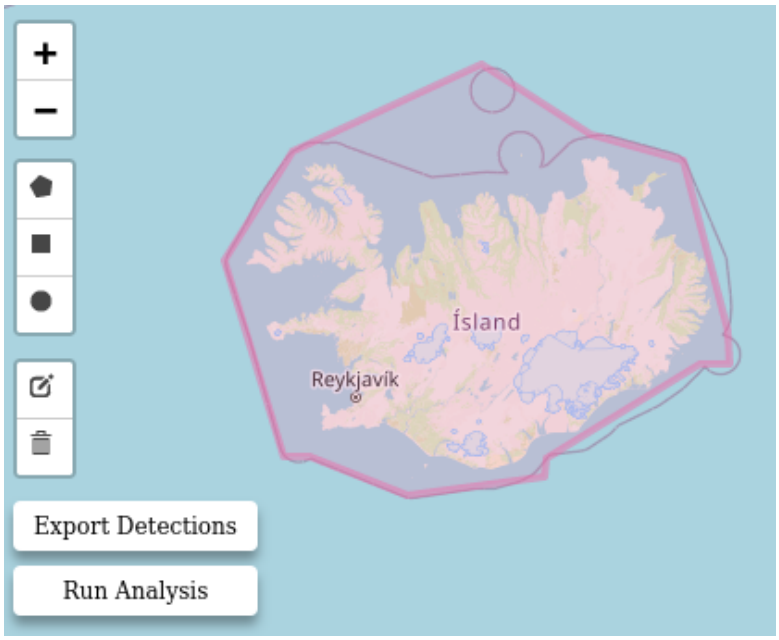


Figure 14: Drawing a polygon over Iceland using the draw toolbox.

We will detail these functionalities in the following sections because they concern the server.

The Date Range Picker

Finally, centered at the UI's top, we placed the data range picker form. This form lets the user choose two dates and query the detection database. We only show the detections from the last five days upon initial load. This form lets the user query the database on a longer or entirely different period.

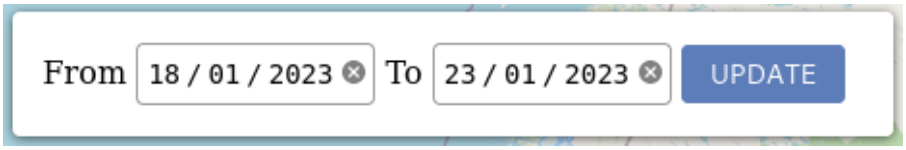


Figure 15: The date range picker form.

Its operation will also be detailed in the following sections.

3.1.2 Data Display in a Web GIS: Techniques and Strategies for Effective Communication and Understanding

The data display in a web GIS is crucial for effective communication and understanding of the information. How the data is displayed can significantly impact how easy it is to understand and interpret the information. A well-designed data display can make it easy for users to identify patterns and trends in the data, while a poorly designed data display can make it challenging to understand the information. Thus this step is crucial for our GIS development.

Boat Detections

Markers

The first approach to display information on a GIS is with markers. We merely add a marker at whatever position there is a detection. But this basic approach has at least a few drawbacks for plotting our ship detections.

One analysis of the region of one of the biggest ports in the world, such as Singapore, already gives us more than a thousand detections. Displaying every detection with a unique marker can make the map looks messy and considerably slows it down. And it is only one port, or just nothing, to put it another way.

To tackle this issue, we decided to use another Leaflet plugin: Leaflet.markercluster. This plugin provides an animated marker clustering functionality for Leaflet. It groups markers that are close together into clusters represented by a single feature. The number inside the cluster marker indicates the number of markers contained in the cluster. The clusters are updated each time the user zooms in or out, giving more or less information.

The plugin allows the user to click on the cluster markers to zoom into the cluster bounds making the map more interactive. Another cool feature is lazy loading. With this setting, the map will only load the markers on the screen and not bother about the unseeable ones. As in our case, with an extensive marker number, the plugin can improve map performance.

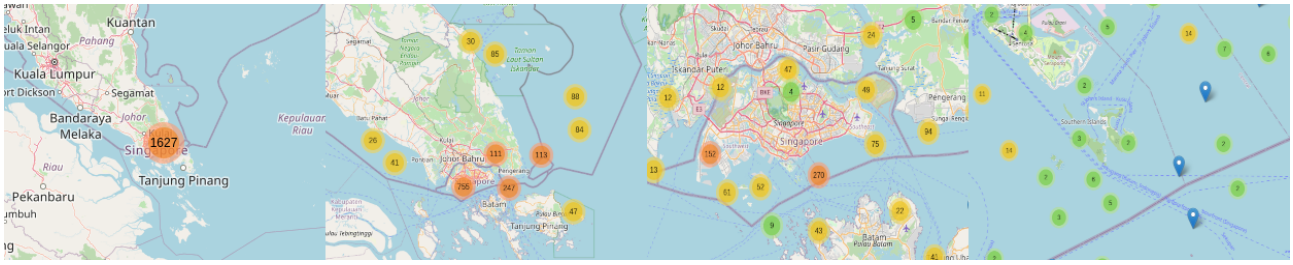


Figure 16: Different zoom levels on Singapore port.

Pop-Up Window

The analysis of the C-SAR data of Sentinel-1 allows us to retrieve some information about the boat. We decided to display this data on Pop-Up windows associated with each marker. Each boat marker is clickable, and we show the following map to the user:

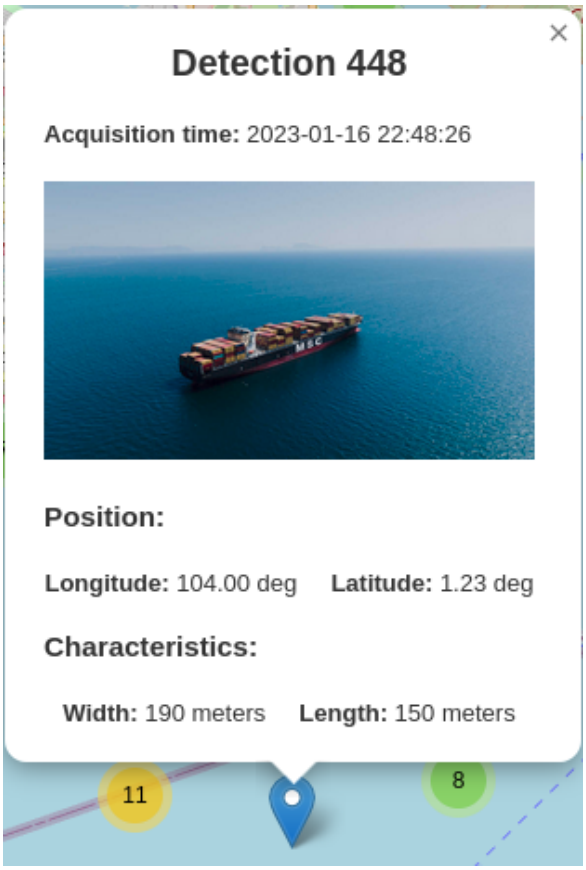


Figure 17: Pop-up window associated with a boat marker.

We display the main info, such as the position of the boat, the acquisition time of the satellite, and the characteristics that are extractable from C-SAR data: the width and length of the detection.

Ports

Another Data Source

We decided to add other data sources in our GIS to display some information about harbors. We made this choice because port data give context to the boat detections. It can give a better understanding of the data and how boats are operating with ports. It can be helpful for the user to identify hotspots in the data.

The data we decided to put is static and only gives basic information, but it is an area worthy of further development later. More complete data could provide insight into trade patterns, help logistics planning, and even help analyze and identify illegal activities and suspect practices, such as smuggling or illicit fishing.

Our port data comes from the data catalog of The World Bank. This dataset combines two distinct sources to provide detailed information about ports and their trade activity. The United Nations Economic Commission for Europe's (UNECE) global repository provides information about the locations of ports around the world, while data from the Logistic Performance Index provides information about the flow of goods and cargo through those ports. Specifically, the dataset includes information about the quarterly deployed capacity (measured in Twenty-foot Equivalent Units) for all ports that had reported international trade during the first quarter of 2020. A twenty-foot equivalent unit is an inexact unit of cargo capacity. It is based on the volume of a 20-foot-long (6.1 m) intermodal container, so it is basically the number of containers exported from the port.

We were able to add this dataset directly to our database, with a little bit of preprocessing to remove the duplicate ports from the list. The cleaned dataset is available on GitHub in the Meta folder.

Markers

The port database contains a little more than 800 harbors around the world. We only decided to display the 50 largest harbors to keep the GIS light. Yet, we made this a parameter, so we can easily change this behavior if needed.

Having information on the outflows of the port, which is correlated to trade activities, we decided to scale the port markers to reflect their importance in international trading. It allows the user to readily see which ports are the most active and provide a visual representation of the relative importance of each port.

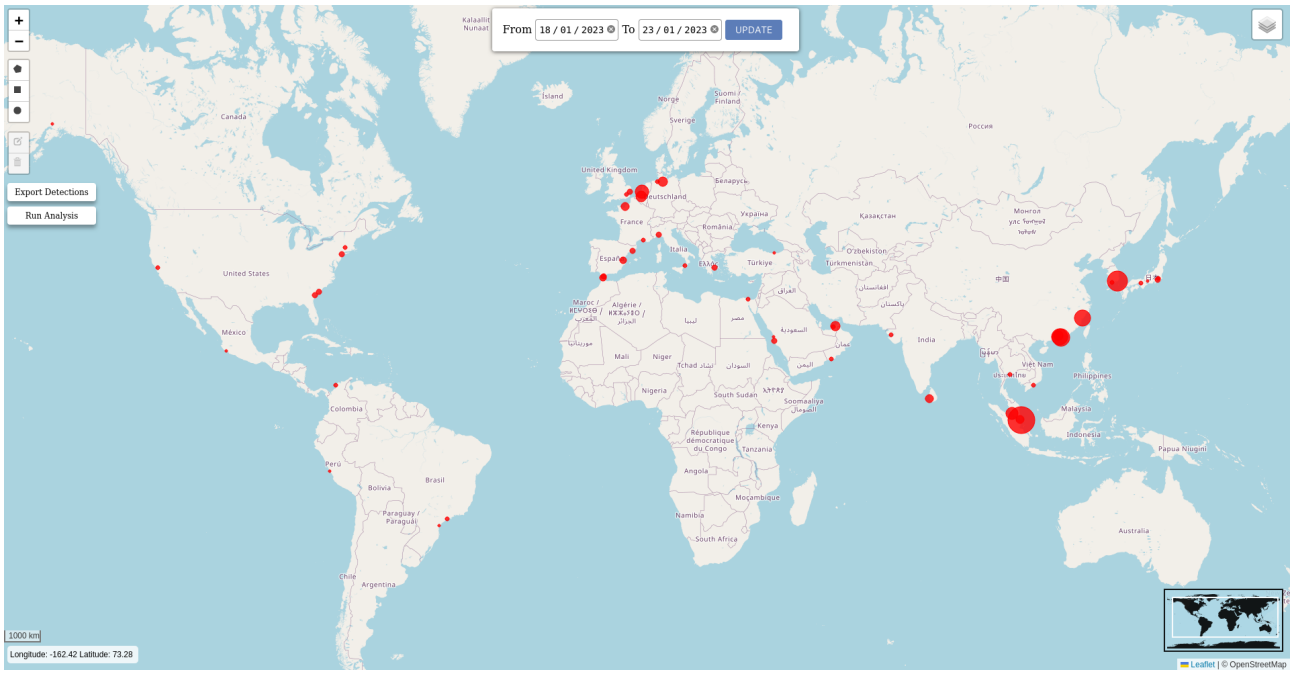


Figure 18: Global view of the GIS with only the port overlay active. Each port is represented by a red circle whose size is determined by their relative outflows.

Pop-Up Window

As for the boats, we add a pop-up window to each port marker. This pop-up keeps the same configuration as the one used for ships.

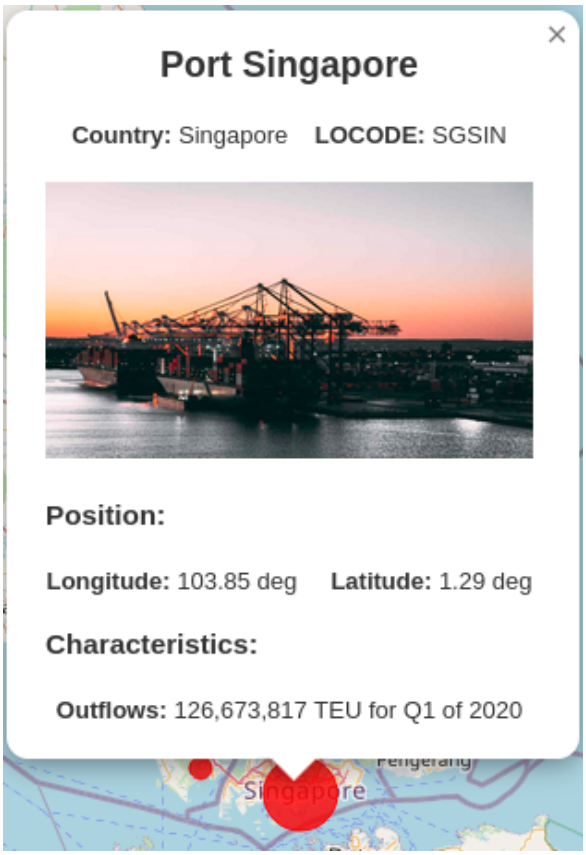


Figure 19: Pop-up window associated with Singapore port.

We display the name of the post, its country, and its LOCODE, a geographic coding scheme developed and maintained by the UNECE. We also give the position of the port and its outflows.

3.2 Connecting the Frontend and Backend: Managing Data Flow and Interaction in a Web GIS

The connection between the frontend and backend of a web GIS is essential for the proper functioning of the system. The frontend, which is the user interface, is responsible for displaying the data and allowing the user to interact with it. The backend, on the other hand, is responsible for handling the data processing and storage. The frontend and backend communicate with each other through an API (Application Programming Interface) that allows the frontend to request data from the backend and display it on the map. The backend also receives user inputs from the frontend, such as user interactions with the map, and processes them accordingly.

3.2.1 A First Approach with a Leaflet Plugin

In our project, we initially used the Leaflet.realtime plugin to display real-time data on the map. This plugin reads and displays GeoJSON from a provided source. The source is a URL on which the plugin will fetch data using AJAX. The plugin was quite convenient, and we used it for a long time, as it handled the AJAX requests to the server without us having to do anything. However, when adding the date range picker, some issues arose.

First, we need to understand how we query data on the server. Our server is a RestAPI. A REST (Representational State Transfer) API is a type of web architecture and a set of constraints usually applied to web services. RESTful APIs are built around the HTTP protocol and are often used to retrieve and manipulate data. RESTful APIs use HTTP requests to POST

(create), PUT (update), GET (read), and DELETE data. What the server does, is waiting for requests on given URLs.

In this case, when it receives a GET request, it fetches the SQLite Database and returns the detection in a GeoJSON format. We can add parameters to URLs, and the server will retrieve and use them to query the database. It means we can give some start_date and end_date parameter to the URL, and then the server will take those parameters and returns the data on this period. The last step is to set those parameter values to the ones defined in the date range picker by the user.

The problem with the L.realtime plugin is that the URL is defined at startup and can't be changed afterward. For a while, we stored the date parameters in the local storage. When the user submitted its date range, we stored the values and reloaded the web page. On reload, we tried to look for those parameters in the local storage, and if found, we would add them to the URL and create a new L.realtime layer with a connection to the URL with parameters.

This solution had one big problem that forced us to change our design. Reloading the whole page, even if Leaflet is lightweight, takes a few seconds as we also need to set the base maps. Adding more and more data would only make this time longer and longer. That is why we decided to use our own AJAX queries using the JavaScript library jQuery. It would require a bit more development, but it would be more flexible and adjusted to our needs.

3.2.2 Using AJAX Queries

AJAX stands for Asynchronous JavaScript And XML. In a nutshell, it uses the XMLHttpRequest object to communicate with servers. It can send and receive information in various formats, including JSON, XML, HTML, and text files. AJAX's most appealing characteristic is its "asynchronous" nature, which means it can communicate with the server, exchange data, and update the page without having to refresh it.

AJAX queries can be hard to handle, but here comes jQuery, a JavaScript library that handles AJAX more simply with an API.

Using jQuery, we can easily make GET and POST requests to the servers, receiving and sending data in the form of GeoJSON or text.

It solved the problem of the date range picker. With this solution, we don't need to reload the page every time the user submits the form. We merely send an AJAX query to the server with the user-defined parameters, clean our existing data, and fill the layer with the new data.

There are multiple interactions between the GIS and the server. At start-up, the GIS will send two GET queries to retrieve information about the boat detections and the ports. As we already saw, the date range picker will also generate a GET query to update the boat layer data. Finally, we also have two requests for the Run Analysis and Export Detections buttons. The Run Analysis button sends a POST request with the polygon drawn by the user using the toolbox as a GeoJSON. The Export Detections button request looks like the GET request used to get boat detections. The only difference is that it asks for CSV and not GeoJSON. Then, the text is processed with JavaScript, and a CSV file is downloaded.

Here's an example of a basic AJAX query using the jQuery API to retrieve our port data:

```
$.ajax({
  url: url_ports,
  type: "GET",
  dataType: "json",
  success: function (data) {
    // Process data and add to the map
  },
```

```
  error: function (xhr, status, error) {
    // Handle error
  },
});
```

Source Code 3: AJAX query using jQuery API to get our port data from the server.

3.3 The Server's Role in our Web GIS

The backend is where most of the algorithms and processes are executed. It is the backbone of our application. For the server side, we decided to use Python as our programming language and the library FastAPI. The server handles a lot of different things.

3.3.1 Building our API

We decided to use FastAPI, a modern, fast, and user-friendly web framework for building efficient and easy-to-maintain APIs. It is built on top of the Python standard library and is designed to be as simple and efficient as possible, with a minimalistic approach to building web applications. One of the main advantages of FastAPI is its speed, as it is one of the fastest Python web frameworks available. It also supports asynchronous programming, which allows for high performance and scalability.

Creating an API with this library is as easy as writing a few lines. Here is the code snippet used to define the behavior of the API when receiving a GET request on the endpoint "/ships.geojson":

```
@app.get("/ships.geojson")
def get_ships(
    start_date: date,
    end_date: date,
    data_type: str = "geojson",
    db: Session = Depends(get_db),
):
    assert end_date > start_date
    return crud.get_detections(db, start_date, end_date, data_type)
```

Source Code 4: Defining the GET behavior of the API for the endpoint "/ships.geojson".

Remembering the interactions defined in Using AJAX Queries, we need to define three endpoints for our API. We also create an endpoint to retrieve the port data using a GET request and one that will process the POST request sent using the Run Analysis button. The Export Button does not need a specific endpoint since we are just querying the ships' data. We only added a data_type parameter to the URL.

With that, our server is running and is able to send and receive data from the client.

3.3.2 Additional Server-side Tasks

In addition to handling client requests, the server must perform other tasks to provide a complete web GIS experience. These tasks include:

Communicating with the Database The server must query the database to retrieve the data needed to fulfill client requests. We use the SQLAlchemy library (as seen in Data Scheme used in the Database) to query our database and retrieve results. Finally, we encode the results in the desired format, such as GeoJSON or CSV, before being sent back to the client.

Running Analyses Our web GIS does not currently perform analysis in real time. Instead, the server only does analyses when the user selects a region on the web GIS and uses the Run Analysis button. This approach provides better flexibility during development. Nevertheless, the server can perform all the steps described in the Methodology and Data Processing section of our project.

3.4 Summary of the System Architecture

We created a summary graph to illustrate the data flows and the interactions between our web GIS, server, database, and the Copernicus API.

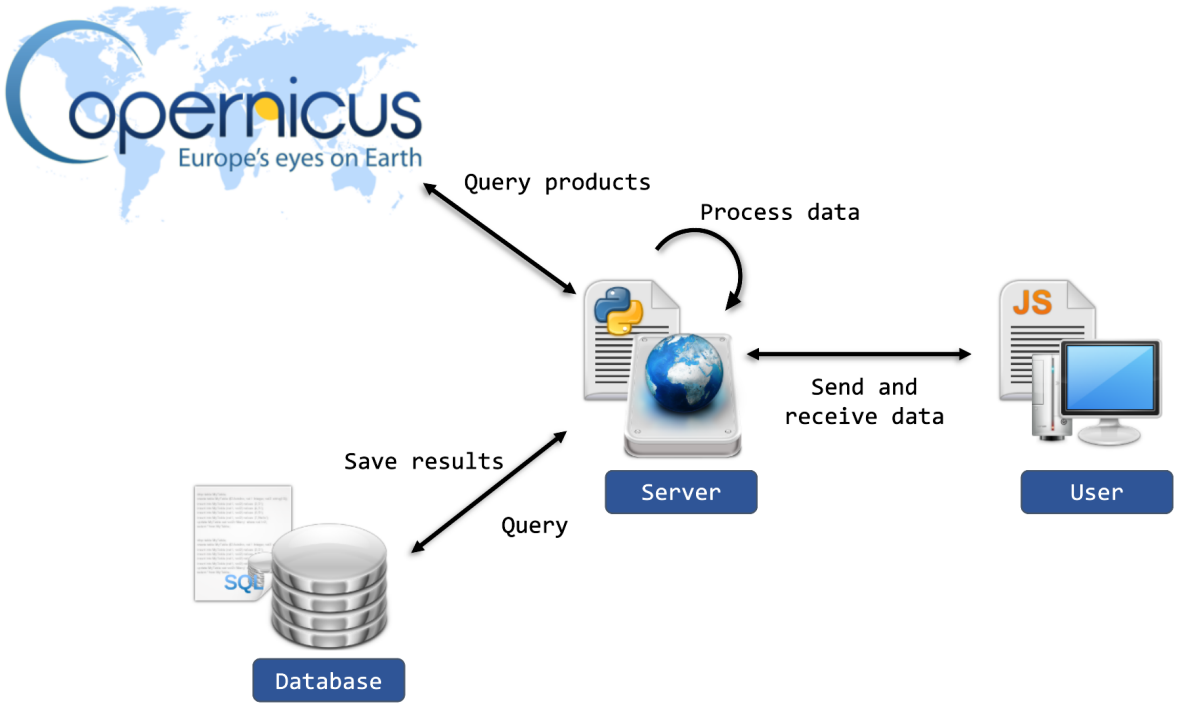


Figure 20: Visual representation of the system architecture of the web GIS with the Copernicus API, the Server, and the Database.

In summary, the system architecture of our web GIS includes several key components: a client-side interface, a server-side backend, and a database. The client-side interface handles front-end processes and displays data to the user. The server-side backend is responsible for processing requests, querying the database, running analyses, and performing other tasks to provide a complete web GIS experience. We use the database to store spatial data and other information required by the web GIS, and the Copernicus API is used to access and retrieve additional data from the Copernicus Earth Observation program.

4 Tools and Technical Issues

Computer calculations and automation are the base of this project. Therefore, we needed to provide solutions to any technical issue we found and build an ecosystem of tools to help us avoid errors. This section aims to show which tools we used in our workflow and why, as well as which issues we faced and how we solved them.

4.1 Team Organization

Behind each consequent software project, there is a team. To maintain good cohesion and efficiency, the team has to use some tools to be organized and ensure that the work done isn't diverging from the original will. We also need to verify that the code isn't regressing in terms of functionalities, readability, and maintainability. Now, we will detail each tool and process we decided on and how they helped us achieve each goal described above.

4.1.1 ClickUp - The Management System

ClickUp is a task management system. Many software are capable of this, and although it might not be the best alternative, it is simple, and we already used it for a previous project.

As only two people are working on the project, there is no explicit need for a task management system. Even a private conversation can be enough. Still, it allowed us to make lists of things we needed to do and share them easily. Especially when the project was at its first tentative steps, writing down goals and each step to achieve them is crucial.

Then, it also helps us organize tasks and explicitly show which are high-priority. Assigning people to tasks also reduces overlapping as we know who does what. Finally, even if we did not go too far with these features, ClickUp provides tools to bound tasks in time and make previsions.

4.1.2 Git - A Source Control Tool

We used a source code control tool to keep several developers working efficiently on the same code. We chose the open-source tool git, the industry's de facto standard for this task.

Git is practical because it helps us centralize the code. It is the only provenance of the code and provides a certified way to ensure we know which revision we work on. Git is essential as it makes people work on the same source code. It also wipes out issues caused by unnoticed code differences when discussing a subject. Also, it provides a rigorous way to solve merge conflicts when multiple people have changed the same code simultaneously.

Moreover, by enforcing more rules, we can reveal the power of git. In short, developers need to put their changes in atomic commits and write concise and descriptive messages to go along. By doing so, one can:

- ⋆ Keep a clean history, meaning that you can follow the evolution of the software over time.
- ⋆ Ease bug bisecting. As we know of any code modifications, we can tell when it started to bug and finally bisect to the actual problem.
- ⋆ Ease bug removal by reverting small atomic commits.

As we can see on the Source Code 5, each commit message justifies its own existence, by properly describing what it does and why. In the example, the commit is a bug fix of a broken URL that regressed in a previous commit.

```
commit cbcff65be9e3f021b78c420ef4ef24580bf39119
Author: VBoulenger <vincentboulenger@gmail.com>
Date:   Mon Jan 23 18:21:36 2023 +0100

    Fix broken url to export ship detections

    Regressed in 12dfb30.
```

Source Code 5: Extract of git log, showing one commit.

Atomic commits also have other advantages, as they ease the peer review process, which we will discuss in the following subsection.

4.1.3 Pull Requests - An Effective Way to Avoid Errors

Git gives you the ability to create branches. We create these branches from a point in history. On the new branch, we can commit modifications without impacting the main tree, allowing for history rewrites. This behavior can be seen in the figure 21, where four commits have been pushed to the "master" branch after that the "osm_data" branch diverged from it. Once you are sure of your changes, you can submit a Pull Request.

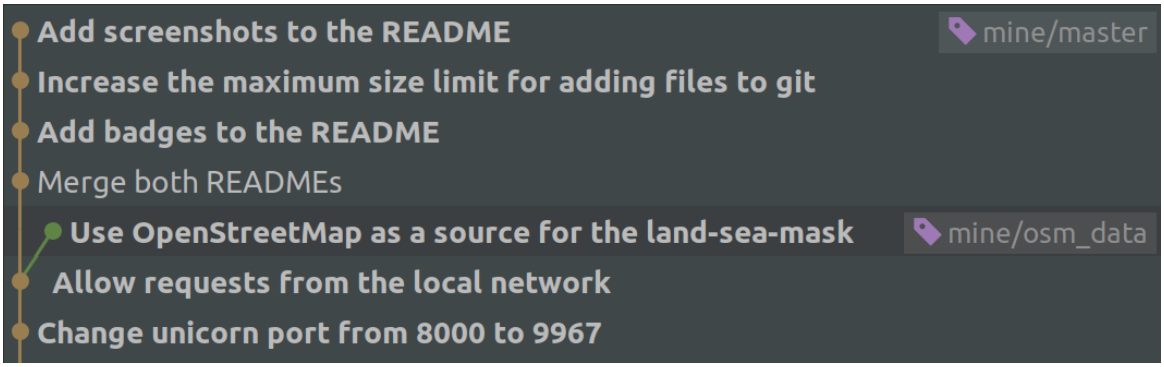


Figure 21: Image of a git tree with two branches.

A Pull Request is a way to ask a maintainer to merge your branch in the main tree. However, they provide an environment to discuss these changes, hence an appropriate time for peer reviews.

Early on in the project, we defined guidelines as to how we would modify the source code. The process was to submit a Pull Request and have the other person check it. This way, the reviewer could request changes or even modify stuff himself. We can not merge the Pull Request until we are both convinced that each commit is meaningful. But once it receives both approvals, changes can land on the main branch.

The time consumption of this procedure can make it seen as overkill, but the process can be done fast when you are used to it and has several benefits by ensuring:

- ⋆ Correct syntax and style - Reading code with weird syntax or different styles is draining. It is not something we want to spend time or energy on when we try to understand code.
- ⋆ Correct logic - A developer can not think of all possibilities or flaws in his code. An external point of view is good to limit these.
- ⋆ Good usage of git - It is quite challenging and often overlooked to write a proper commit description, especially on your modification. A neutral look can often suggest better rephrasing.

Also, as Pull Requests provide an independent environment, one can iterate on its changes regardless of the state of the main branch. It is especially significant as it refrains the desire to add a half-baked feature to be able to work on a new one.

Finally, we execute a pipeline, but we will discuss it in further detail in Continuous Integration - Ensuring Quality.

4.1.4 Precommit - An Help for Maintainers

On creating new commits, git allows the user to configure some hooks. These are user-customizable scripts that are run just before the actual committing operation. It is common practice to add checks to ease the reviewer's task. It is an essential tool as it increases productivity by removing a commute between the author of the commit and the maintainer that will point out the issue.

Moreover, it is, in fact, more effective than humans for this type of exercise. For example, typos can be unnoticed, and code style is too subjective. By letting a script do this work, we ensure way better results.

On code quality, human fallibility can manifest through carelessness and a lack of expertise. We are not Python experts, and using programs that can detect and expose bad practices is an advantage for maintainability.

Precommit is an open-source tool that builds and eases the installation of those checkers. It is so easy to configure and use that we ended up with more than 20 plugins. A list showing them all (with a description) is available here: List of Used precommit Plugins It is also visible on the GitHub repository.

4.1.5 Continuous Integration - Ensuring Quality

In the same vein as precommit, we also check that code submissions are correct. A pipeline performs this step. We refer to this step as Continuous Integration.

The pipeline is only composed of a 'flake8' check for now. It ensures that the code conforms to the Python standard. It is deplorable that we didn't have more time to enhance it. Indeed, we should at least check for as many things as in the 'precommit' hook.

Moreover, testing is a crucial part of Continuous Integration. We should add some unit testing - responsible for testing short parts of the code - and some global tests. It's very usual when modifying a part of a codebase to break something elsewhere. It explains why it is better to also use global tests to preserve the overall behavior.

4.2 Increasing Performances

One of the most significant technical issues we faced during this project was performance. Indeed, the server needs to answer any request at a suitable time. For example, a response time of around an hour would be a total dealbreaker for the project. To allow correct navigability on the GIS, we decided we needed to achieve a response time of fewer than five minutes per request. In this subsection, we will follow the chronological order of each attempt we made to reduce these times and go under the sub-five-minute mark.

To compare performances, we need to have a reference value. We will use the first working prototype to this end. The following subsection aims to describe the state of this first version.

4.2.1 A Matter of Language

Software performances are always tied (to some degree) to the programming language in which the application is made. For this project, we choose to use Python. There are various

reasons to this choice.

First, we are both familiar enough with it to start and write a project of this size. It is rather influential considering the few months allowed to complete the project.

Second, it is a well-known and tested language. Many tools exist for it, and one can be sure he will find help on the internet about a Python problem.

Finally, it is also essential that the language is one of the more accepted in the scientific community. Hence, we can find plenty of packages to use instead of re-writing everything ourselves.

We quickly developed a running prototype using the snappy package to interact with SNAP.

However, Python doesn't have only advantages. It is commonly considered as not very efficient. Some projects counterbalance this by the time gained when developing the application, but unfortunately, as we discussed above, we need to lower the runtime to its minimum.

At its best, this prototype needs a substantial time of 40 minutes to complete a tile. It was a big disappointment, but we had some ideas to reduce it.

4.2.2 Computation Efficiency with Multiprocessing

While the prototype was running, we realized it only used one CPU core. Modern computer architectures are designed with multiple cores to improve parallelization on a computer. Taking advantage of all the CPU's cores is the go-to way to decrease computation time.

Parallel computing is often not very straightforward to implement. Among other difficulties, the developer needs to ensure that shared data is correctly accessed and that concurrent algorithms are not stepping on each other's feet. To implement it in our project, we decided not to touch the actual detection algorithm but rather sandbox it in an agnostic environment. We do the multiprocessing upstream, where an algorithm splits a Copernicus product into tiles and feeds multiple concurrent workers with those.

Unfortunately, this solution does not work very well in practice. When we tested it, instead of not being used, CPU cores spent most of their time waiting. This problem has been quite hard to troubleshoot, but we finally figured out the technical details behind it.

To interact with SNAP, snappy uses jpy. The team behind SNAP develops this library, and it is helpful to communicate between the Java Virtual Machine and the Python interpreter. In other words, it means we can use Java code in Python and the other way around. In this light, snappy is only a little layer on top of jpy to give the user a python-friendly interface of SNAP, see Figure 22.

As jpy is very close to the Python interpreter, it is exposed to the Global Interpret Lock (GIL). This lock, essential to the current implementations of Python, is almost blocking jpy create concurrent calls. That, combined with the overload generated by dividing the product into tiles and making more calls through jpy, made this solution counter-productive and slightly slower than the streamlined, single-core version.

We discussed multiple solutions to resolve this problem:

- ⋆ Remove the GIL from the Python interpreter. It has been tried numerous times through the years but never made its way to the principal implementation. Regardless, this is way over our heads for this project.

- ⋆ Teach SNAP and jpy to work around the GIL. This solution is probably more manageable but remains a massive task with a high risk of being impossible.

- ⋆ Reimplement the whole stack of SNAP operators we are using in another language. Reimplementing them in Python or, better, C++ would have given us a significant performance boost. This solution is also quite time-consuming, but probably better than the two above.

Yet, it has another drawback: our versions won't have been thoroughly tested, unlike the one included in SNAP.

Finally, we found another solution, a package called snapista, that uses a clever solution to work around this issue.

4.2.3 snapista - A Partial Solution

Unlike snappy, which targets the Java Application Programming Interface (API), snapista relies on gpt, another tool provided with SNAP. As shown earlier, SNAP internally uses graphs as an abstraction to represent a process. Gpt, the acronym for *Graph Processing Tool*, is a command-line-based tool that uses SNAP in the backend to compute graphs. Snapista is an interface to create graphs and call gpt under the hood.

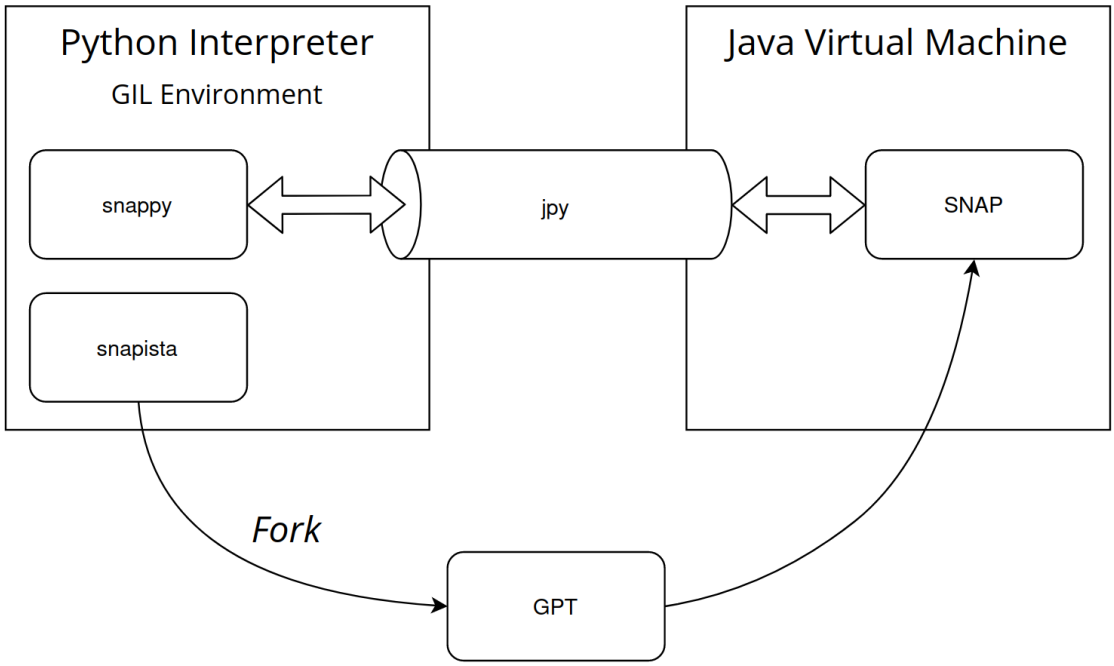


Figure 22: Scheme of the implemtation details of snappy and snapista.

It might look like an insignificant change, but moving the computational-heavy work outside the scope of the Python interpreter, and hence the GIL, provides a significant performance improvement. Due to the multiprocessing nature of this solution and the removal of jpy from the equation, we measured a considerable time reduction. The multi-language bridge also had a significant, and until this moment unnoticed, impact on performance.

With one of our machines with the most core count, we achieved a remarkable improvement, down to a processing time of less than 1 min 15s.

It might look like a perfect fit for this project, but regrettably, the package is a compromise as it refrains us. By its design, snappy provided a substantial panel of customizations. Also, as the gpt program runs in its process, we can't retrieve the results from memory. It forces us to write the result to a file and read it afterward.

As snapista is, by nature, very different from snappy, we had to rewrite a big part of the backend, which we did in this commit: d709badf.

5 Extending BOATMAN

After spending four months working on this project, we are reaching the end of its initial phase as a student project. However, we still have a list of ideas to improve the project. This part is devoted to exposing them and showing the reader the underneath potential of BOATMAN.

First, we are going to discuss how we think we can improve results and then how it can be used as a tool for future projects.

5.1 Improving Results

Sadly, the short time we had to complete the project didn't allow us to fine-tune our project. We couldn't spend time evaluating our detection results and couldn't do any parameter optimization. In this way, our project is lacking, and it should be one of the next major efforts.

Reducing False Positives

Currently, one of the major flaws of our process is the unreliability of the results due to the high number of false positives.

We can probably significantly reduce the number of false positives by removing points that occur multiple times on different images of a region taken at particular times. Stationary points are certainly not ships, and we can discard them without a doubt. This solution could also solve the precision issue of the sea mask, as the shore is not moving.

Changing the Core Algorithm

The CFAR-based algorithm used in SNAP was developed in the early 2000s, it is still relevant, but the research didn't stop with it. For example, the SUMO algorithm [8], or this other CFAR-based algorithm from 2017 [9], have way better results.

Gathering Data from more Satellites

As previously mentioned in section Possible Data Sources, the Copernicus API is not the only possible data source for our project. USING more SAR satellites would allow for closer proximity to real-time data. However, it should be noted that many reactive services are only available through purchase. Additionally, the project could expand to include other types of data beyond SAR imagery. By incorporating optical imagery from other satellites, and terrestrial sources, such as Automatic Identification System (AIS) signals broadcasted by registered boats, we could potentially identify ships and improve validation capabilities. Once the system performs well, we could also use it to detect boats navigating without AIS.

5.2 Retrieve more Information

For now, an approximation of the size of the boat is available. Adding more data would be great and enhance the capability of our web GIS. As shown in Li et al. (2022) [11], the field of ship detection is getting closer to computer vision and machine-learning-based models. These models can extract more information and would be very profitable for BOATMAN. For example, in Del Prete et al. (2021) [7], the team used deep learning to detect wake produced by traveling vessels. The wake can be exploited both for ship route and velocity estimation, as well as a marker of a ship's presence.

5.3 Enhancing the GIS

Our web GIS has many potential features we could add to enhance the user's experience and analysis capabilities.

External Data Streams

One important feature would be the integration of external data streams, as was done with the ports' data, to increase analysis opportunities. Currently, we do not have any tools for data visualization, including tools such as charts, maps, and graphs would allow the user to deepen their understanding of the data.

More Grained Filtering and Integrated Analysis

Additionally, adding historical ship tracking would give users a historical view of boat movements and the ability to track ships over a specific period. We could implement user management features such as user registration and access control to support collaboration and sharing of data among different users. Advanced filtering options, such as by ship type, size, and ownership, would allow users to find and analyze specific or groups of ships. We could also implement pre-analysis on some data flows. Overall, the more data available, the greater the opportunities for analysis.

5.4 Using BOATMAN as a base for future projects

Due to both its design and its nature, BOATMAN suits perfectly as a base for other projects.

An Accessible Backend

As the Python backend exposes a RESTful API, the GIS that we developed isn't the only application that can use it. This base can easily be used for any other projects based on ship detections.

Using the Data Collected by BOATMAN

As we discussed in the introduction, having real-time positions on boats has many applications. If we include many of the applications of the previous section, BOATMAN could be the perfect playground to develop applications for pirate detections or route optimizations.

Conclusion

During the last four months, BOATMAN evolved from an idea to a concrete project. Our project aimed to develop a web GIS for boat detection using satellite data from the Copernicus program. After more than a hundred commits and a thousand code lines, BOATMAN achieves to detect ships using SAR data from the Sentinel-1 mission and displays boat and port data on an interactive web GIS.

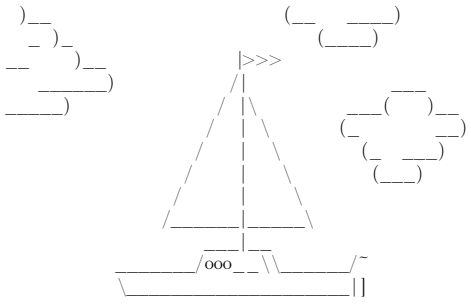
We used a combination of technologies to develop a client-side and a server-side application. We developed our frontend using the JavaScript library Leaflet. It allowed us to create a map and add attractive features to the user interface. We developed the server-side using Python and its library FastAPI. Finally, we used SQLite for the database management system. Retrieval and processing of the data are powered with the Python packages sentinelsat and snapista. We designed a processing routine to automate the extraction of boat positions from the SAR images.

The design of the interactions between each component of the project has needed careful consideration, and we faced several challenges over the course of this project. Those troubles, whether on the frontend or backend, were overcome through the help of different tools such as git. These tools helped us keep every issue under control. The organization we put in place has played a central role in the success of this project since it allowed for swift communication between us, even with other life constraints.

Our GIS allows users to easily view and analyze boat detection data in a user-friendly and interactive manner. It also has the potential for further development, such as integrating machine learning techniques and adding more data sources for an increased understanding of the data.

Over those few months, we had the chance to research and develop our understanding of various technologies and programmatic paradigms. From data management to web development to data processing, we saw very different domains and their specific constraints. Developing all the project's aspects was a considerable learning opportunity since it forced us to handle everything ourselves. It has specifically deepened our understanding of domains such as web development that were not familiar to us. Nevertheless, the fact that we were successful in our approach shows that the technique we used to stay on track and our ability to adapt was in line with the project's needs.

Some points in our organization could ask for improvement, such as task scheduling and deadlines, but overall, we are satisfied with the outcome of our project. Once fully developed, we believe it has the potential to bloom into a powerful tool for maritime surveillance and analysis, as well as give valuable insights for logistics, trade, and environmental protection.



Annexes

A Summary Sheet

<table>
<tr><th colspan="2">Summary sheet
Authors: Vincent Boulenger - Lucas Chollet</th></tr>
<tr><td>Subject</td><td>Targets</td></tr>
<tr><td>Creation of an interactive web-based GIS for displaying boat detections in realtime using satellite SAR images.</td><td>⋆ Deploying a web infrastructure
⋆ Creating an interactive and adapted interface
⋆ Implement scientific results to detect ships from SAR images.
⋆ Automate the above process and connect it to the user interface.</td></tr>
<tr><td>Main customer</td><td>Tools used</td></tr>
<tr><td>⋆ Any curious amateur
⋆ Organization that wants to make a survey on ships
⋆ Future research, using BOATMAN as a playground</td><td>Python, SQLite, FastAPI, SNAP, NodeJS, JavaScript, Leaflet</td></tr>
<tr><td colspan="2">Performed studies</td></tr>
<tr><td colspan="2">⋆ Study of the state of the art methods to perform ship detections
⋆ Study of the tools used in sattelite image analyzing
⋆ Study of the performance of our process
⋆ Test of the solution over different spot</td></tr>
<tr><td>Results</td><td>Explanation of the discrepancies</td></tr>
<tr><td>⋆ Good practices over the codebase
⋆ SAR-based detections of ships
⋆ A GIS over the web
⋆ A RESTful API server to support the GIS</td><td>The realtime aspect of the detection did not work out, the last update can be in a range of at worst three days and at best three hours.</td></tr>
<tr><td>Difficulties encountered</td><td>Work to be pursued</td></tr>
<tr><td>⋆ Lack of accesss to real-time data (paid services)
⋆ Lack of time to ehnance the scientific model for detections
⋆ Lack of time to polish the user-interface</td><td>⋆ Tune parameters and improve reliability
⋆ Implement more modern algorithm for boat detections
⋆ Add more data sources
⋆ Ehnance the user-experience</td></tr>
</table>

B Code Snippet to Download Copernicus Products

```python
def download_sentinel_data(
    request_geojson: FeatureCollection,
    platformname: str = "Sentinel-1",
    producttype: str = "GRD",
):
    """
    Parameters
    ----------
    request_geojson: FeatureCollection
        Geojson object containing a polygon in which data in queried.
    platformname: str
        Indicates the platform used (default to 'Sentinel-1').
    producttype: str
        Indicates product type (default to 'GRD').
    """
    api = SentinelAPI(None, None)
    footprint = geojson_to_wkt(request_geojson)

    products = api.query(
        footprint,
        platformname=platformname,
        producttype=producttype,
    )

    latest_product_id = products.popitem(last=False)[0]
    result = api.download_all([latest_product_id])
```

Source Code 6: Python code to download the last product corresponding to a region

C UML Scheme of the Database

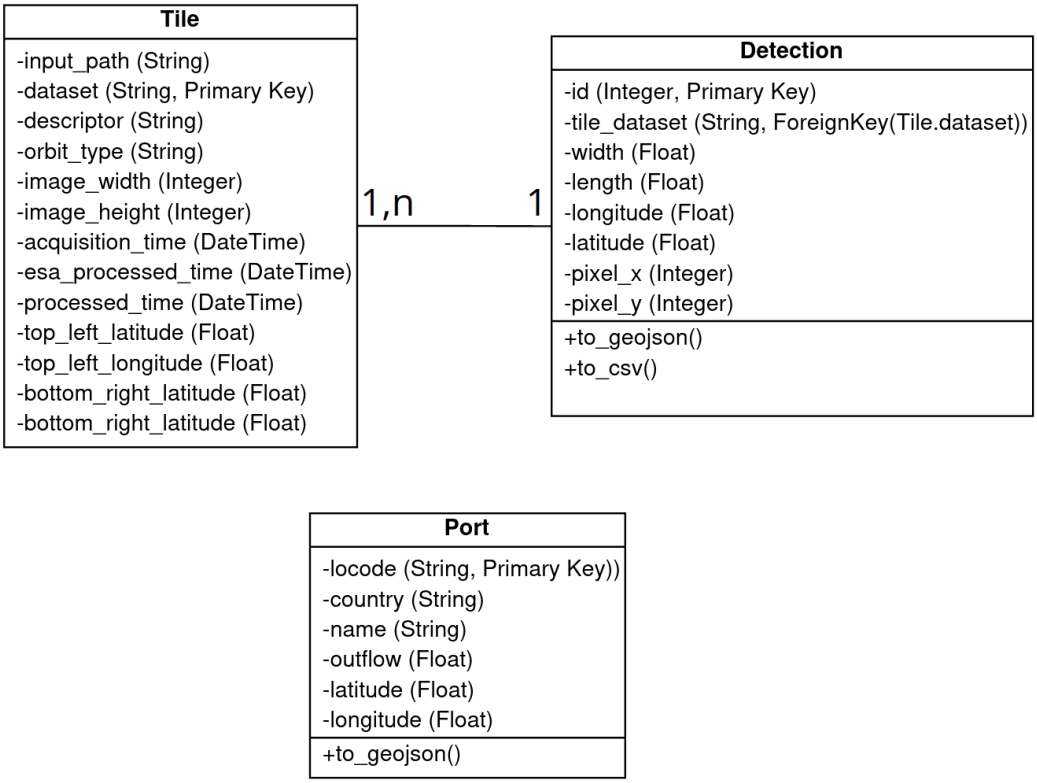


Figure 23: UML representation of the database.

D List of Used precommit Plugins

- ⋆ check-added-large-files: Prevents giant files from being committed.
- ⋆ check-case-conflict: Checks for files that would conflict in case-insensitive file systems.
- ⋆ check-json: Checks JSON files for parsable syntax.
- ⋆ check-merge-conflict: Checks for files that contain merge conflict strings.
- ⋆ check-symlinks: Checks for symlinks that do not point to anything.
- ⋆ check-yaml: Checks yaml files for parsable syntax.
- ⋆ detect-private-key: Detects the presence of private keys.
- ⋆ destroyed-symlinks: Detects symlinks that are changed to regular files with a content of a path to which that symlink was pointing.
- ⋆ fix-byte-order-marker: Removes utf-8 byte order marker.
- ⋆ end-of-file-fixer: Ensures that a file is either empty or ends with one new line.
- ⋆ mixed-line-ending: Replaces or checks mixed line ending.
- ⋆ trailing-whitespace: Trims trailing whitespace.
- ⋆ csslint: CSS linter.
- ⋆ prettier: JavaScript linter.
- ⋆ reorder-python-imports: Put standard import first, then third-party, and finally first-party.
- ⋆ mypy: Static type checker for Python.
- ⋆ autoflake: Removes unused imports and unused variables.
- ⋆ gitlint: Git commit message linter.
- ⋆ typos: Finds and corrects spelling mistakes among source code.
- ⋆ black: Python code formatter.
- ⋆ flake8: Ensure code conformance to PEP.

References

[1] *Accueil | Copernicus*. [Online; accessed 4. Jan. 2023]. Jan. 2023. URL: https://www.copernicus.eu/fr.

[2] European Space Agency. *Copernicus Open Access Hub*. [Online; accessed 24. Jan. 2023]. URL: https://scihub.copernicus.eu/dhus/#/home.

[3] European Space Agency. *Copernicus Open Access Hub - APIs Overview*. [Online; accessed 24. Jan. 2023]. URL: https://scihub.copernicus.eu/userguide/APIsOverview.

[4] Andreas Braun. "Radar satellite imagery for humanitarian response. Bridging the gap between technology and application". In: *Universität Tübingen* (Aug. 2019). URL: https://publikationen.uni-tuebingen.de/xmlui/handle/10900/91317.

[5] David Crisp. "The state-of-the-art in ship detection in Synthetic Aperture Radar imagery". In: (May 2004).

[6] *Data Derived from OpenStreetMap for Download*. [Online; accessed 24. Jan. 2023]. Jan. 2023. URL: https://osmdata.openstreetmap.de/data/land-polygons.html.

[7] Roberto Del Prete, Maria Daniela Graziano, and Alfredo Renga. "First Results on Wake Detection in SAR Images by Deep Learning". In: *Remote Sensing* 13.22 (2021). ISSN: 2072-4292. DOI: 10.3390/rs13224573. URL: https://www.mdpi.com/2072-4292/13/22/4573.

[8] Harm Greidanus et al. "The SUMO Ship Detector Algorithm for Satellite Radar Images". In: *Remote Sensing* 9.3 (2017). ISSN: 2072-4292. DOI: 10.3390/rs9030246. URL: https://www.mdpi.com/2072-4292/9/3/246.

[9] Pasquale Iervolino et al. "SAR Ship Detection for Rough Sea Conditions". In: (Aug. 2019). DOI: 10.1109/IGARSS.2019.8900332.

[10] *Landsat Science*. [Online; accessed 19. Jan. 2023]. Jan. 2023. URL: https://landsat.gsfc.nasa.gov.

[11] Jianwei Li et al. "Deep Learning for SAR Ship Detection: Past, Present and Future". In: *Remote Sensing* 14.11 (2022). ISSN: 2072-4292. DOI: 10.3390/rs14112712. URL: https://www.mdpi.com/2072-4292/14/11/2712.

[12] Marza Marzuki et al. "Fishing boat detection using Sentinel-1 validated with VIIRS Data". In: *IOP Conference Series: Earth and Environmental Science* 925 (Nov. 2021), p. 012058. DOI: 10.1088/1755-1315/925/1/012058.

[13] *MODIS Web*. [Online; accessed 19. Jan. 2023]. Jan. 2023. URL: https://modis.gsfc.nasa.gov/data.

[14] Rose Njambi. *How SAR data is complementary to optical*. [Online; accessed 6. Jan. 2023]. Jan. 2023. URL: https://up42.com/blog/tech/sar-data-complementary-optical.

[15] *Overview | Get to Know SAR – NASA-ISRO SAR Mission (NISAR)*. [Online; accessed 6. Jan. 2023]. Jan. 2023. URL: https://nisar.jpl.nasa.gov/mission/get-to-know-sar/overview.

[16] *Polarimetry | Get to Know SAR – NASA-ISRO SAR Mission (NISAR)*. [Online; accessed 6. Jan. 2023]. Jan. 2023. URL: https://nisar.jpl.nasa.gov/mission/get-to-know-sar/polarimetry.

[17] *Sentinel Online - ESA - Sentinel Online*. [Online; accessed 4. Jan. 2023]. Jan. 2023. URL: https://sentinels.copernicus.eu/web/sentinel/home.

[18] *Sentinel-1*. [Online; accessed 3. Jan. 2023]. Jan. 2023. URL: https://www.esa.int/ESA_Multimedia/Images/2014/02/Sentinel-1#.Y7QY8vCROZM.link.

[19] *Sentinel-1 - Overview - Sentinel Online - Sentinel Online*. [Online; accessed 4. Jan. 2023]. Jan. 2023. URL: https://sentinels.copernicus.eu/web/sentinel/missions/sentinel-1/overview.

[20] Serco Italia SPA. "Ship detection with Sentinel-1 – Gulf of Trieste (version 1.3)". In: (2018). URL: https://rus-copernicus.eu/portal/wp-content/uploads/library/education/training/OCEA01_ShipDetection_Trieste.pdf.

[21] *TerraSAR-X and TanDEM-X - Earth Online*. [Online; accessed 19. Jan. 2023]. Jan. 2023. URL: https://earth.esa.int/eogateway/missions/terrasar-x-and-tandem-x.

[22] *WMO OSCAR | Details for Instrument SAR-C (Sentinel-1)*. [Online; accessed 4. Jan. 2023]. Jan. 2023. URL: https://space.oscar.wmo.int/instruments/view/sar_c_sentinel_1.