This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 861111

DRONES4SAFETY

Drones4Safety

Research & Innovation Action (RIA)

Inspection Drones for Ensuring Safety in Transport Infrastructures

Implementation of the platform with data flow and visualization modules

D6.3

Due date of deliverable: 28.02.2022.

Start date of project: June 1st, 2020

Type: Deliverable
WP number: WP6

Responsible institution: SDU
Editor and editor's address: Lea Matlekovic, SDU

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 861111

Version 1.0
Release Date: November 30, 2021

Project funded by the European Commission within the Horizon 2020 Programme		
Dissemination Level		
PU	Public	✓
CO	Confidential, only for members of the consortium (including the Commission Services)	

1 Executive Summary

The following deliverable is focused on extending prototype software developed in D6.2. The deliverable presents the software architecture of Drone Inspection as a Service platform (DIaaS) and is part of the WP6 – Mission control and navigation. It describes the software architecture including newly developed modules for complete data flow and visualization. Specifically, three new microservice modules were developed to support delivering inspection data from the drones to the cloud platform. The drone delivers images to the cloud for the AI analysis. The images are stored in the Object Storage and further processed by the Alteia platform using the Image Uploader. Satellite images are gathered within a microservice, retrieving the data from available sources Copernicus Open Access Hub.

The document is structured in six sections. After an introduction, we present the software architecture including the modules for downloading satellite images and uploading images received from the drones to the Alteia AI platform which is under development in the WP4. The section describes the functional and non-functional requirements of the cloud system identified in D2.5. Furthermore, developed services are functionally described. The fourth section describes the design of the databases enabling data storage and the flow within the system. The fifth section describes the implementation of additional services developed in T6.5. The last section depicts possible data flows within the system and provides visualization examples.

Contents

References

MongoDB, https://www.mongodb.com/

PostgreSQL, https://www.postgresql.org/

PostGIS, https://postgis.net/

Gazebo, http://gazebosim.org/

GitLab https://docs.gitlab.com/ee/ci/

Copernicus Open Access Hub https://scihub.copernicus.eu/

Pymongo https://pymongo.readthedocs.io/en/stable/

Sentinelsat https://sentinelsat.readthedocs.io/en/stable/

Alteia https://alteia.readthedocs.io/en/latest/index.html

Acronyms

Acronym	Description
D4S	Drones 4 Safety
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
SQL	Structured Query Language
REST	Representational State Transfer
AI	Artificial Intelligence
CI	Continuous Integration
CD	Continuous Deployment
OSM	Open Street Map
SDK	Software Development Kit
UID	Unique Identifier
DIaaS	Drone Inspection as a Service

2 Introduction

The main scope of the Drones4Safety (D4S) project is to develop a system of autonomous, self-charging, and collaborative drones that, inspecting an extensive portion of transportation infrastructures in a continuous operation, can increase the safety of the European civil transport network. The project outcomes, the software services and the hardware drone system, will offer railway and bridge operators the chance to inspect their transportation infrastructure accurately, frequently, and autonomously.

The main purpose of this document is to provide a description and specification of additional software services enabling data flow and data visualization. Services are developed as an extension of the Drone Inspection as a Service (DIaaS) platform, presented in the previous WP6 deliverables, D6.1 and D6.2. The platform architecture was designed to support modifications and additions, therefore old features can be easily modified, and new features can be easily added. Modularity facilitated the addition of new services for integrating satellite imagery and enabling data flow through the platform. For a complete understanding of the data flow, we described all services developed in D6.2 providing a better overview of the platform. Further sections of this deliverable describe the implementation of the additional services, data flow through the platform, and visualizations on the web interface.

3 Software Architecture

Software architecture in general describes application or platform organization and structure. Architectural decisions impact application quality, performance, maintainability, and usability. Software architecture chosen at the beginning of the development can have a high impact on the application's future and affect its success. Architecture should be considered and planned if the application needs to be scalable, maintainable, and easily upgradable. When developing the application from scratch, the focus is usually on having a working product as soon as possible. However, that approach can become unsustainable when the number of platform features grows fast. If the architecture is not reconsidered, the development slows down and the platform becomes difficult to maintain. The complete implementation process is presented in deliverable D6.2. Here we give a brief overview of platform architecture as it is important for the development of additional modules which are the focus of this deliverable.

3.1 Software requirements

In order to design the architecture suitable for the scope of this project, we focused on system requirements defined in D2.5. From these system requirements, we identified cloud software requirements and developed the software in the appropriate architectural style. The following subsections present software functional and non-functional requirements. Functional requirements describe how the platform should work and how the user can interact with it. Non-functional requirements describe characteristics the platform should have to assure satisfying performance and stability when users and developers interact with it.

3.1.1 Functional requirements

The main actors interacting with the platform are developers and platform users such as drone operators. The developers should be able to easily add features; without spending time on understanding the codebase. They should not be concerned with deployment. Therefore, the deployment should be automatized. However, it should be easy to test new features by interacting with previously implemented functionalities and drones. The operator should be able to select the inspection targets and plan the safe inspection route. At the operator's request, the routes should be calculated and stored in the database as a set of waypoints representing locations with latitude and longitude. The route calculation determines the order of visiting the targets for each drone participating in the mission. The routes should be visualized on the 2D map and sent to the drones as waypoints used for navigation. The operator should be able to monitor the inspection progress and receive alerts in real-time. Inspection data and drone telemetry should be stored in a database and visualized on the web interface upon request. After the mission, the operator should be able to retrieve the mission plan, mission data, and mission results.

3.1.2 Non-functional requirements

We identified scalability, interoperability, modifiability, and performance as the platform's non-functional requirements. When load increases, the platform should adapt to the load and uninterruptedly continue working. Interoperability describes the platform’s ability to interact and exchange data between components within the system as well as with the outside systems. For autonomous drone inspection mission planning, the platform should be able to exchange information between components and communicate with the drone systems. The software architecture should be designed to facilitate the development, communication between components, and addition of new features without modifying the rest of the system. Modifiability specifies the degree to which the platform is robust to changes. The platform should perform to satisfy the requirements and to do so efficiently. We expect the platform’s growth in features and increased traffic in the near future. These requirements, when met, will facilitate the platform's growth.

3.2 Architecture design

Based on functional and non-functional requirements, we designed the platform in microservice software architecture and automatized the delivery process. The architecture was designed to facilitate the extension of the platform with modules enabling data flow and visualization. This subsection describes the architecture overall as well as additional modules developed during T6.5.

Microservice architecture, also known as Microservices, is a software architecture that structures an application as a collection of small, loosely coupled services. The services are independently deployable, highly maintainable, and testable. The concept was developed to overcome the downsides of monolithic architecture. Microservices have clear boundaries between each other and communicate through the HTTP protocol, usually by exposing a REST API and sending requests. Each service represents one capability that makes it easier to understand and locate the code. It also makes them robust to changes. Since they are small and deployed independently, they are easy to update and maintain. Services can be scaled independently and automatically, depending on the load. They can use different technology stacks, including programming language and data storage. That gives high flexibility to the development teams. However, there are some drawbacks of microservice architecture. The fact that a microservices application is a distributed system requires handling of fallacies the distributed computing carries. It means that developers must deal with the additional complexity. Microservice architecture, developed in D6.2, facilitated the addition of services for image storage, satellite imagery, and uploading images for further analysis.

There are many practices and tools developed to facilitate the testing, integration, and deployment of microservices. DevOps is a combination of practices and tools designed to facilitate the delivery of applications. It aims to increase an organization's ability to deploy applications faster by removing the barriers between development and operations teams. DevOps practices automatize the delivery process and are implemented as a part of a production pipeline. Applied tools and practices depend on the application delivery requirements and goals. Continuous integration is the practice of merging changes to the main branch as often as possible. When developers commit local changes to the remote repository, automated build and tests can run there before proceeding to production. Remote repository platforms with built-in version control, like GitLab and GitHub, facilitate the collaboration between developers and enable continuous integration. These platforms also integrate with different DevOps tools to enable continuous delivery and deployment. The practice of continuous delivery includes continuous integration and after a successful build, automatically propagates the application to staging. In staging, the application is deployed to the testing environment. There, the application including all services can be run and tested. Continuous delivery requires manual approval for release into production. However, the continuous deployment includes all the steps described in continuous integration and delivery, but instead of manual, the release process is also automated. Described process can vary in complexity depending on the number of services, test requirements, and in general, the deployment strategy. By enabling continuous integration and continuous delivery, described in D6.2, newly developed modules could easily be integrated and deployed to production.

3.2.1 Platform architecture

This subsection describes microservices developed in D6.2. based on platform requirements. Services are designed to enable scaling of specific processes which we expect to experience, e.g., route calculation. Figure 1 depicts architecture and communication flow and it includes the extension to the platform developed during the T6.5. Newly developed services are depicted in orange while previously developed services are in blue. The drone operator interacts with the system using the web interface. Web interface communicates with the backend by sending requests and serving responses to the user. It is implemented and deployed as a microservice; therefore, it is described in this section. The operator selects inspection targets on the web interface, which creates a POST request to the Missions service. The request contains target locations. Mission service has drone locations stored and, with received target locations, sends the POST request to the Routing Solver. Routing Solver uses A* Pathfinder to determine the order of visiting all the targets. A* Pathfinder requests the graph created from data stored in Towers, Railways, Bridges, and No-fly services to determine the shortest path for each combination of target location and drone location. Calculated routes are stored in the Missions service, visualized on the web interface, and sent to the drones Simulation through the Message Broker. While the mission is in progress, the Message Broker sends drone telemetry data to the Drone Log service, where the data is stored. Drone Estimator service uses telemetry data to estimate drone location in periods when drones are not reporting to the cloud. Estimation data is stored in the Drone Log service database for mission simulation after the mission finishes. Message Broker updates Missions service database with inspection results. Images from the drones are stored into the Object Storage and transfer is handled through the Message Broker. Satellite Imagery service accesses the satellite data from the Copernicus Open Access Hub and provides the user with the download option. The Image Uploader uploads images to the Alteia AI platform where the images are analyzed using deep learning techniques for fault detection and definition. The individual services are described in the following subsections as well as the web interface.

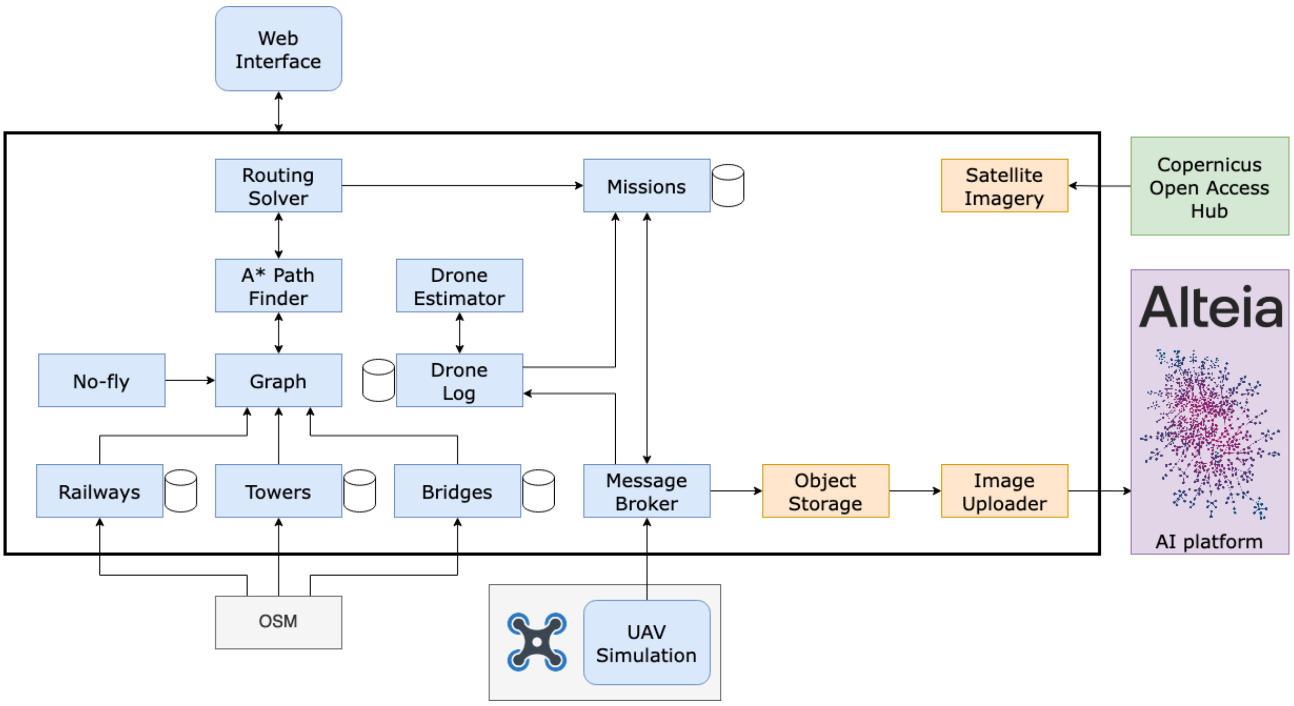


Figure 1 Microservices software architecture

3.2.1.1 Web interface

Based on the 2D map, the web interface visualizes inspection targets on real locations and enables user interaction with the system. It allows the user to select the targets and visualize calculated routes for each drone. The user can store and retrieve mission plans or execute them by sending the calculated navigation data to the drones.

3.2.1.2 Towers service

The service extracts power towers and power lines locations as points and lines from Open Street Map (OSM) and stores them in the database. The service communicates locations data to the Graph service to create nodes and edges in the graph. The service also provides the data to the web interface to visualize the inspection targets and enable the user target selection.

3.2.1.3 Railways service

The service extracts railways location as described in the Towers service, communicates with Graph service for graph creation, and provides the data to the web interface.

3.2.1.4 Bridges service

The service extracts polygons around the bridges from Open Street Map and stores them in the database. It communicates with Graph service for graph creation and provides the data to the web interface. On the web interface, each bridge is represented as a center point of the extracted polygon.

3.2.1.5 Graph service

The service creates a graph based on the data requested from Towers service, Railways service, Bridges service, and No-fly service. The graph contains nodes representing the infrastructure locations and edges containing pairs of neighboring nodes. The edges' cost is set up as the calculated distance between the neighboring towers. The service excludes nodes inside the no-fly areas using the data requested from the No-fly service. The graph is created only once, and it is served to the A* Pathfinder service to determine the shortest path between the set of inspection targets and drones. The new graph creation should run only when new locations are added to either towers, railways, or bridges databases.

3.2.1.6 Routing Solver service

The service receives inspection targets from the user via Missions service and drone locations from the vehicles. It determines which drone should inspect which target based on the shortest paths requested from the A* Pathfinder service. It returns the path for each drone as a set of waypoints containing graph locations in latitude and longitude. The results are stored in the Missions service database and sent to the web interface for routes visualization.

3.2.1.7 A Pathfinder service*

The service requests a graph from the Graph service and performs the A* algorithm on the targets and drones' locations received from the Routing Solver service. It returns the shortest path between a target and a drone as well as the total path distance. The shortest path is calculated considering costs assigned to the edges. The shortest path calculations are used by the Routing Solver service to solve the vehicle routing problem.

3.2.1.8 No-fly service

The service contains areas where drone flights are not allowed and provides the data to the Graph service. Web interface requests No-fly areas and visualizes them on the 2D map.

3.2.1.9 Missions service

The service provides mission data management and storage. It stores missions planned by Routing Solver service, defining the mission route and tasks for each drone. Upon the user's request, the mission routes and tasks are sent to the drones for navigation. This service stores inspection results in the database after the mission finishes. The user requests a mission plan from the service, visualizes the routes, monitors the mission progress on the web interface, or requests the mission visualization after the drone conducts an inspection.

3.2.1.10 Drone Log service

The service receives data from the drones and stores it in the database. The data consists of telemetry from drone sensors. For monitoring purposes, the service uses Drone Estimator to estimate the drone location and reports it to the web interface. The location is updated with drone measurements every time they are received.

3.2.1.11 Drone Estimator service

The service estimates the drones' location in periods between the location data is received from the Drone Log service. Using the estimations, the user can monitor the drone estimated movements on the web interface while the mission is in progress. Without the estimator, drone movements would be visualized only when received directly from the drones, which can be every 10 seconds or more. Estimations are sent back and stored in the Drone Log database.

3.2.1.12 Message Broker service

The service enables communication with drones and manages the data distribution from the cloud to the drones as well as from the drones to the cloud. It keeps track of the successful delivery of messages and assures the distribution based on priorities.

3.2.1.13 Drone Simulation service

The service runs a drone model in the Gazebo simulator and provides a testing environment for both developers and users. For developers, it assures faultless integration of newly developed features or services into the system. For operators, the simulation allows the mission testing before the execution with the real drones.

3.2.1.14 Object Storage service

The service stores images received from the drone through the Message Broker service into the object storage. Corresponding image metadata are stored in the database. The service enables connection and communication with the database and serves images upon request. The object storage also stores satellite images received from the Satellite Imagery service and serves them upon request.

3.2.1.15 Satellite Imagery service

The service retrieves satellite images from the open-source SDK and serves them to the web interface after processing. Satellite images are used to visually validate locations of the relevant infrastructure i.e. power towers, power lines, bridges, and railways. Images can be used to check for safe landing zones near the infrastructure, where the drone can land in case of emergency. Upon the user's request, the images are downloaded according to the geolocation of the inspection infrastructure.

3.2.1.16 Image Uploader service

The service retrieves images from the object storage and uploads them to the Alteia AI platform where they are analyzed using deep learning algorithms. It uses Alteia APIs to establish communication with the platform. The platform enables fault detection from uploaded infrastructure images and it is currently under development in the WP4.

4 Database Design

This section describes databases within the microservices-based system described in the previous section. The section provides an overview of database design for storing infrastructure locations, planned missions, drones telemetry, and images sent from the drones.

4.1 Railways, Towers, and Bridges Database

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. We use it to store infrastructure data in JSON-like documents which are easily queried. MongoDB supports geo-queries when the database contains geographic locations. The user is able to query elements within a specified distance or range, which is very useful in our case for determining which nodes in the graph are neighboring nodes. Services that use MongoDB are Towers, Railways, and Bridges. The data is represented with models shown in Table 1. Data models determine a form the data is stored in the database. The database contains information about power towers, lines, bridges, and railways locations and specifications. Towers are defined as "nodes" and the data model contains location. Lines are defined as "ways" and contain an array with unique identifiers for nodes. Nodes' location is stored with the same data model used for towers. Bridges' locations are stored as ways containing an array of nodes forming the polygon around the bridge. Each node contains geolocation with latitude and longitude. Railways are stored using the same data model used for lines.

Table 1 Data models as stored in the database

Towers	Lines	Bridges	Railways
_id: ObjectId type: “node” id: tower_id location: Object	_id: ObjectId type: “way” id: line_id nodes: Array tags: Object _id: ObjectId type: “node” id: node_id location: Object	_id: ObjectId type: “way” id: bridge_id nodes: Array tags: Object _id: ObjectId type: “node” id: node_id lat: latitude lon: longitude	_id: ObjectId type: “way” id: railway_id nodes: Array tags: Object _id: ObjectId type: “node” id: node_id location: Object

4.2 Missions and Drone Log Database

PostgreSQL, also known as Postgres, is a free and open-source relational database management system emphasizing extensibility and SQL compliance. In Missions and Drone Log services, we created relations using PostgreSQL because this database will experience user queries and complex join queries for which NoSQL databases are not well adapted.

Data needs to be transferred between services in the cloud, to the drone, and from the drone to the cloud. Database relations specify mission data (including inspection tasks), telemetry data, and inspection results data. Therefore, we designed databases in Missions and Drone Log services to create global data space. Figure 2 shows relations implemented in these services and relationships between them. Relations contain attributes described as:

- **Missions relation -** the relation contains mission data, like mission name, the status of the mission, swarm participating in the mission, routes associated with it, geofence regions where the flight is permitted, and time information.
- **Task relation -** the relation specifies tasks given to the drone. It describes the inspection type, location of task execution, and the drone the task has been delegated. It stores specifications describing which sensors to use e.g., RGB camera, frequency of data acquisition, speed of the drone during data acquisition, etc.
- **Result relation -** the relation stores inspection results and related data. It contains time data when the results are received, the location where the result was obtained, the fault description, and the image UID associated with the faulty image.
- **Telemetry relation -** the relation stores drone telemetry data like position, orientation, velocity, speed from different sensors. It also stores the drone’s status, battery status, and onboard computer status.
- **Swarm relation -** the relation stores swarm identifiers for swarms participating in the mission.
- **Drone relation -** the relation stores drones participating in the mission.

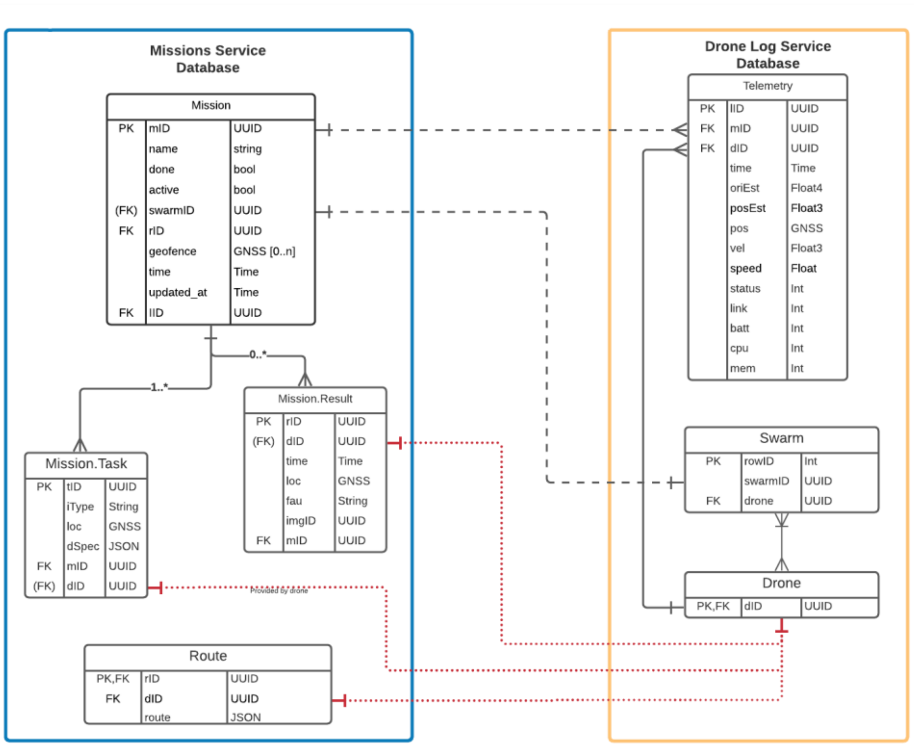


Figure 2 Database diagram

4.3 Object Storage Database

The Object Storage database is a MongoDB database storing images in a binary format. The database stores images received from the drones with corresponding metadata. MongoDB enables geo-querying which facilitates querying based on locations where the images were taken. For the proof-of-concept application, we use cloud storage offered by MongoDB similarly to databases storing power towers, power lines, bridges, and railways. The database contains a collection named "Image" storing useful data related to each image received. The data model is shown in Table 2. The data model determines the form in which each image is stored in the database. Object ID is a unique identifier automatically set by MongoDB when inputting a new entry. Image ID is an image unique identifier enabling search for a specific image. Mission ID stores the unique identifier of the mission during which the image was captured. Drone ID stores the unique identifier of the drone which captured the image. By storing the latitude and longitude of the location where the image was captured we enabled geo-querying and search for the images taken near the location chosen on the map. This design enables the user to search for images collected in previous autonomous missions either by the unique identifiers or by geolocation.

Table 2

Image
_id: ObjectId img_id: int mission_id: int drone_id: int lat: latitude lon: longitude metadata: Object img: Binary Object

5 Services implementation

This section describes newly developed services that enable data flow through the system and contribute to the data visualization through the web interface. The services handle data exchange within the system as well as the outside systems, i.e. Alteia AI platform and Copernicus Open Access Hub.

5.1 Object Storage

The service enables connection to the database and receives requests from the users based on their actions on the web user interface. When the user wants to upload an image to the Alteia platform for analysis, the service retrieves images from the chosen mission, captured with a specific drone, or based on geolocation where the images were captured and sends the selected image to the Image Uploader. The service connects to the database by using the pymongo library which enables querying as well. Before the image is stored in the database, it is saved as a binary object which allows inputting its value to the database. When the image is stored in the database, the user receives an alert on the web interface that the image from the drone was received. The service implements functions for retrieving the images based on the user’s criterium. It allows retrieving by geolocation, mission ID, drone ID, or mission ID and drone ID combined. The user can also trigger the image upload to the Alteia platform for AI analysis.

5.2 Satellite Imagery

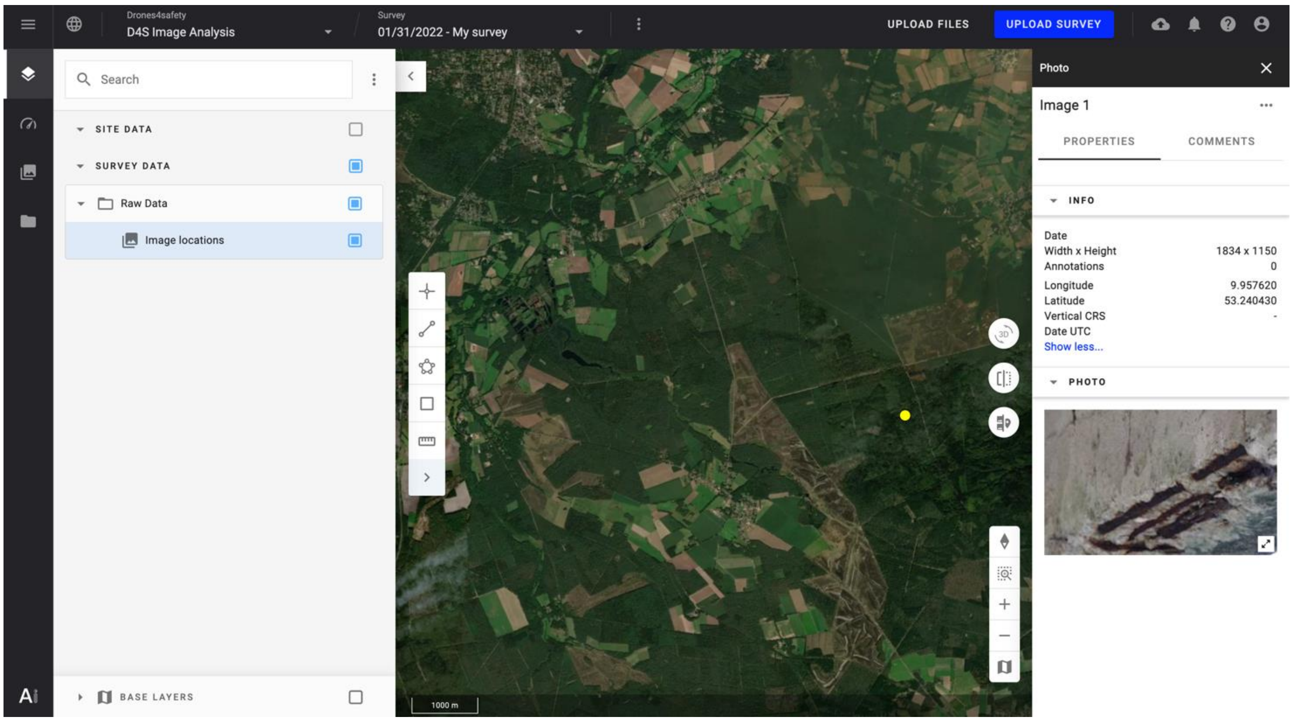
The service retrieves satellite images for the infrastructure location validation. It is implemented as a microservice and communicates with open-source SDK providing APIs for accessing sentinel satellite images. Sentinel satellites are part of the Earth observation mission from the Copernicus program that systematically acquires optical imagery at high spatial resolution (10 m to 60 m) over land and coastal waters. 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.

Satellite images are represented as rasters where each pixel holds information about the location and is stored in a matrix. The service access images through the Python library sentinelsat which makes searching, querying, retrieving, and downloading Sentinel satellite data possible and easy. We interact with the data from the

Copernicus Open Access Hub and retrieve images based on the user's request. User requests satellite data by clicking the target infrastructure on the web interface and confirming the request for download. When retrieving the images from the Copernicus Open Access Hub, we need to specify the bounding box which is created using the location the user inputs. When writing a query, we specify a time when the images were taken and make sure we are retrieving recently collected data. The Satellite Imagery service retrieves data from the previous two months where the cloud coverage is only up to 20%, assuring images visibility. The API returns an ordered dictionary containing satellite images bounding boxes where our bounding box is included or partially included. The algorithm for choosing the most appropriate image is implemented, selecting and downloading only the image with the best characteristics and visibility. The image is cropped to include only the relevant location and the user is able to download it. The result of the described process is a satellite image showing relevant infrastructure according to the limits of the satellite image resolution. The user can validate if the infrastructure coordinates are correct and if there are flat landing zones where the drones could land in the case of an emergency.

5.3 Image Uploader

The service uploads mission images to the Alteia AI platform, currently under development in WP4. To upload mission images, Delair provided us login credentials for using Alteia Python SDK. Image Uploader service creates a new project on the Alteia platform if one with the same name does not already exist. The project can contain multiple missions, providing information about drone routes stored in the Missions service. When drones collect images, the images are stored in the object storage and can be provided to the Image Uploader at users' request. The Image Uploader creates an image dataset on the Alteia platform where each image has the metadata with dimension values and location where they were taken. The Alteia SDK provides a method for uploading images to the previously created image dataset. For each location where drones gathered images, the Image Uploader creates a dataset where an image is uploaded. The final upload is visible through the Alteia platform and it is shown in Figure 3.



Date	
Width x Height	1834 x 1150
Annotations	0
Longitude	9.957620
Latitude	53.240430
Vertical CRS	-
Date UTC	

Figure 3 Infrastructure image uploaded to the Alteia platform through the Image Uploader Service. The map shows the location where the image was captured. The image is visible on the right side, including the corresponding metadata.

6 Data flow and Visualisation

This section presents two data flows within the system. The first one describes the user's actions and possibilities of interacting with the platform while the second one follows the data flow from the drones to the cloud. The data flows result in visualizations on the web interface, which are presented in this section.

The user interacts with the system through the web interface where different actions are possible. To calculate mission routes, the user selects target infrastructure on the web interface and requests route calculation. Route calculation is described in detail in Section 3. Routes are stored in the Missions database and visualized on the web interface. An example of route calculation is shown in Figure 4 where the user chooses bridges for inspection and the vehicle routing algorithm calculates near-optimal routes shown on the web interface. The map can also be visualized in the satellite view and the user can have a closer look at the infrastructure from aerial images. For sending the waypoints to the drone, the user selects a mission from the possible options stored in the Missions database and starts the mission. Inspection paths are sent to the drones. In the proof-of-concept application, a simulated drone starts executing the mission as shown in Figure 5. Before selecting the target infrastructure, the user has a possibility to visually validate the infrastructure location by downloading an up-to-date satellite image of the infrastructure location. To download the image, the user selects a location on the map. Images are retrieved from the Satellite Imagery service which accesses the data from the Copernicus Open Access Hub. An example of an image that was processed within the Satellite Imagery service is shown in Figure 6. To analyze images collected by the drones, the user selects the corresponding mission and uploads images gathered during that mission to the Alteia AI platform. Images are retrieved from the Object Storage. The data flow of users' interactions with the system is depicted in Figure 7.

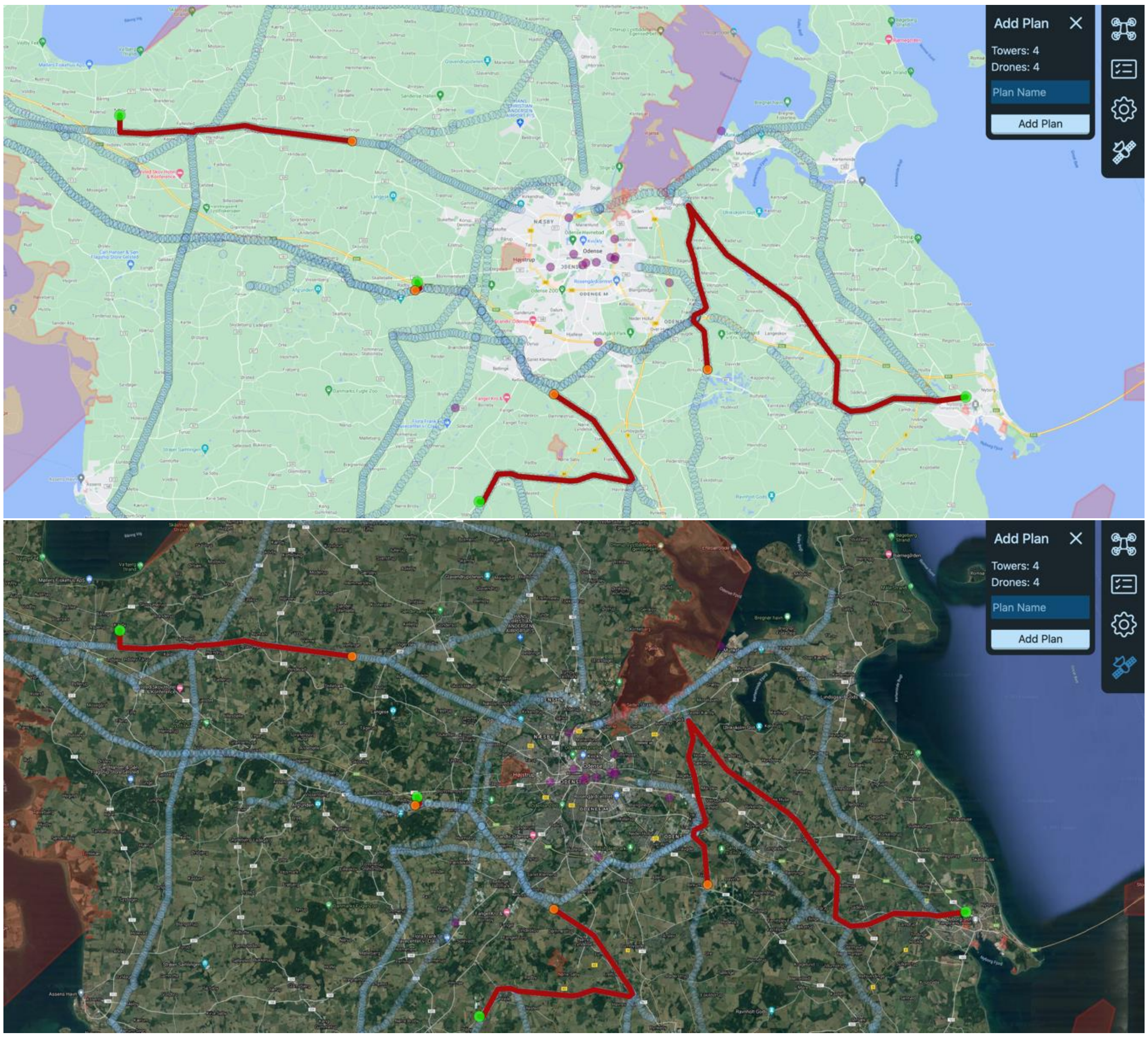


Figure 4 Map-based web interface showing calculated routes for four drones inspecting four selected bridges. Drones are shown in orange, selected bridges are green, power towers are blue, and bridges are purple. Result routes are shown in red.

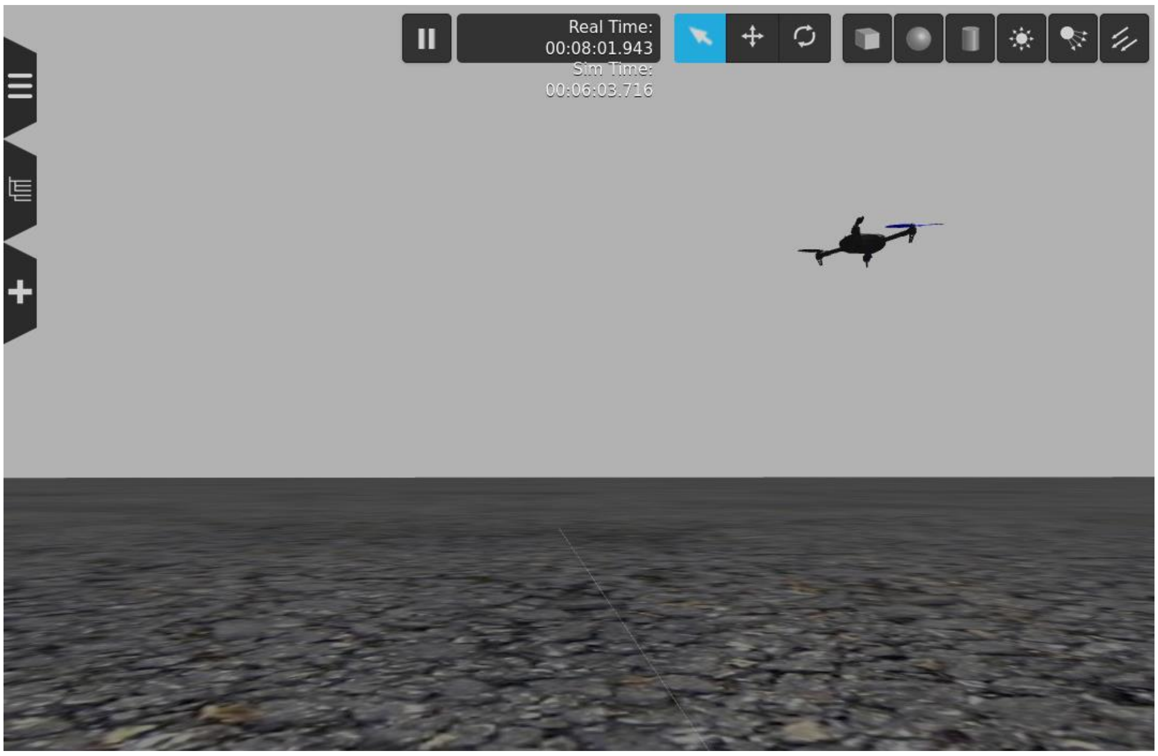


Figure 5 Simulated drone executing the inspection mission. The simulation is exposed to the web and the user can monitor the drone's movement both in simulation and on the web interface.

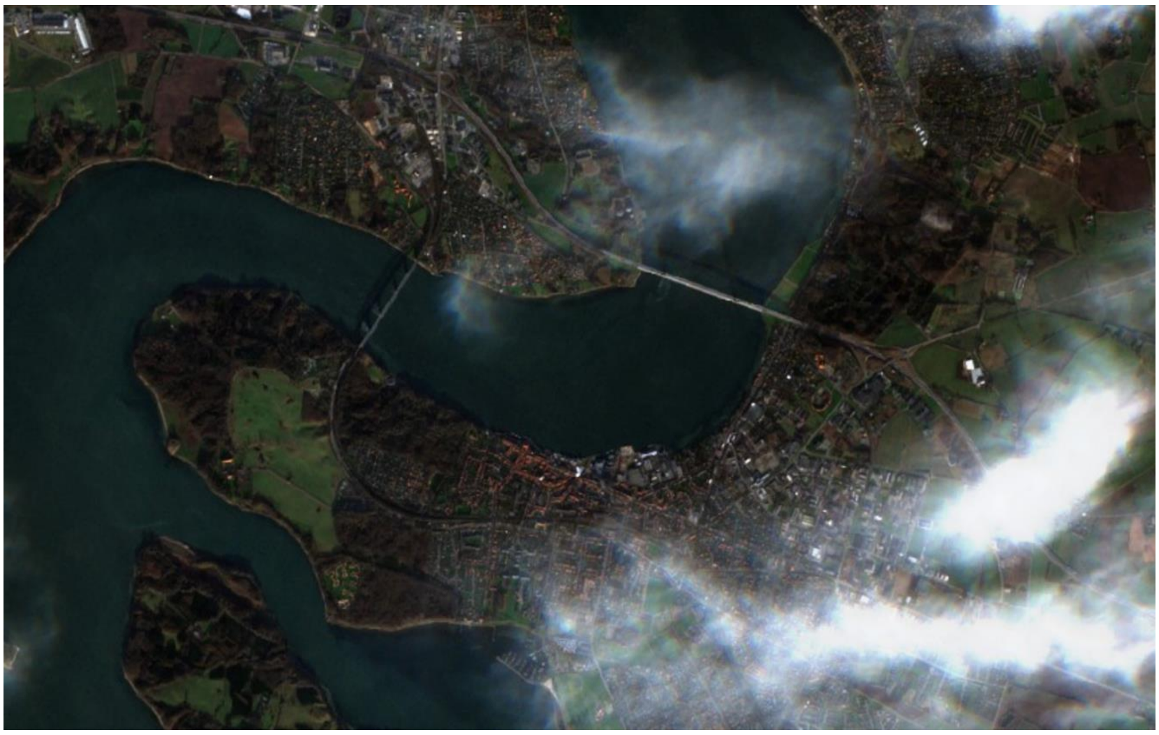


Figure 6 Satellite image of a bridge downloaded from the Copernicus Open Access Hub using the Satellite Imagery service.

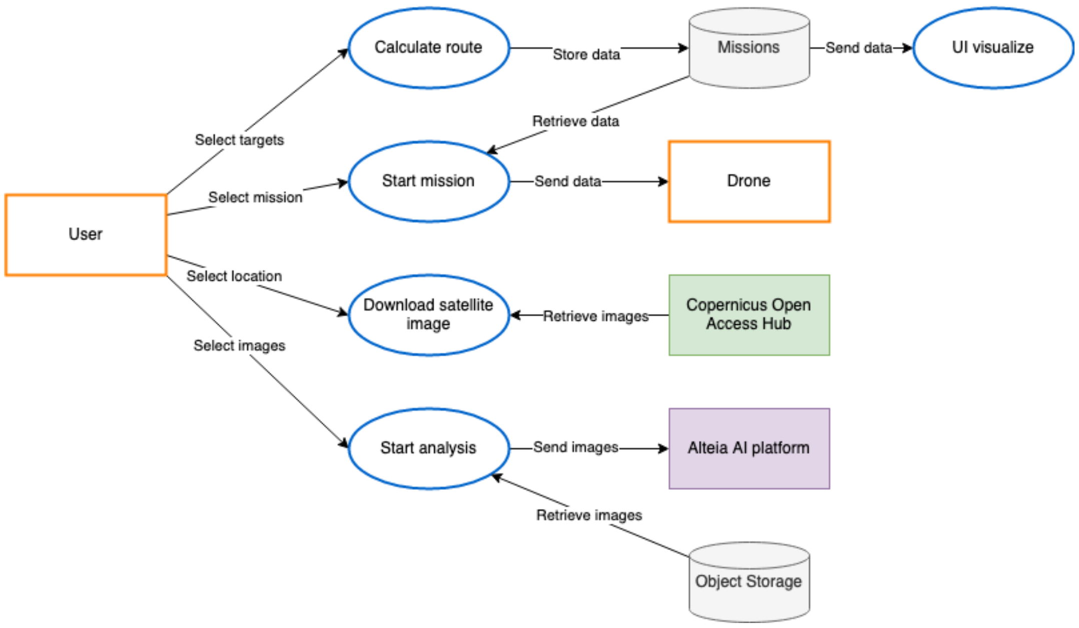


Figure 7 Data flow of user interactions with the cloud platform and drones.

Drones use sensors to measure the telemetry data and send it to the cloud. The data is stored in the Drone Log database. The data is supplemented with estimation values for the periods when the drones do not report the telemetry to the cloud. The Drone Log reports the telemetry to the web interface and saves the mission status to the Missions database. Image flow begins with taking the image using a camera mounted on board the drone. Images are sent to the cloud and stored in the Object Storage notifying the user through the web interface that the image has been received. The data flow of drone data transfer to the cloud is depicted in Figure 8.

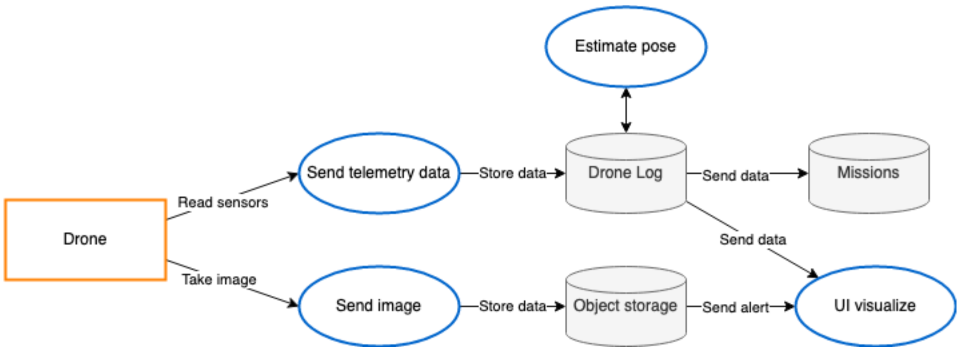


Figure 8 Data flow of transferring the data from the drone to the cloud.