GNSS-Free Localization for UAVs in the Wild

Smart Systems
Turku Intelligent Embbedded and Robotic Systems (TIERS) Lab
Master's Degree Programme in Information and Communication Technology
Department of Computing, Faculty of Technology
Master of Science in Technology Thesis

Author:
Marius-Mihail Gurgu

Supervisors:
MSc. (Tech) Jorge Peña Queralta
Assoc. Prof. Tomi Westerlund

July 2022

UNIVERSITY OF TURKU
Department of Computing

MARIUS-MIHAIL GURGU: GNSS-Free Localization for UAVs in the Wild

Master of Science Thesis, 55 p.
Turku Intelligent Embedded and Robotic Systems (TIERS) Lab
July 2022

Considering the accelerated development of Unmanned Aerial Vehicles (UAVs) applications in both industrial and research scenarios, there is an increasing need for localizing these aerial systems in non-urban environments, using GNSS-Free, vision-based methods. This project studies three different image feature matching techniques and proposes a final implementation of a vision-based localization algorithm that uses deep features to compute geographical coordinates of a UAV flying in the wild. The method is based on matching salient features of RGB photographs captured by the drone camera and sections of a pre-built map consisting of georeferenced open-source satellite images. Experimental results prove that vision-based localization has comparable accuracy with traditional GNSS-based methods, which serve as ground truth.

Keywords: GNSS-Free, ROS 2, UAV, PX4, Photogrammetry, Vision-based localization

Contents

List of Figures

List of Tables

List Of Acronyms

AGL	Above Ground Level
BVLOS	Beyond Visual Line of Sight
DDS	Data Distribution Service
FMU	Flight Management Unit
FPS	Frames Per Second
GE	Google Earth
GNSS	Global Navigation Satellite System
LIDAR	Light Detection and Ranging
MAE	Mean Absolute Error
MAV	Micro Aerial Vehicle
MAVLink	Micro Aerial Vehicle Link
QGC	QGroundControl
RGB	Red Green Blue
ROS	Robotic Operating System
SIFT	Scale Invariant Feature Transform
UAV	Unmanned Aerial Vehicle
UART	Universal Asynchronous Receiver-Transmitter
UDP	User Datagram Protocol
UWB	Ultra-Wideband
VI-SLAM	Visual-Inertial Simultaneous Localization and Mapping

1 Introduction

Unmanned Aerial Vehicles (UAVs) are currently used in a wide range of scenarios and commercial applications. They already have reliable localization methods based on the Global Navigation Satellite System (GNSS), Visual-Inertial Simultaneous Localization and Mapping (VI-SLAM) or Ultra-wideband (UWB) systems. The first method is usually used in outdoor environments, where it is relatively simple to use a GNSS sensor to obtain accurate positioning data [1]. The second and third methods prove their utility in environments where GNSS signal is unreliable, for example inside buildings, where localization systems such as UWB [2] or VI-SLAM are preferred [3].

Autonomy of aerial drones and their hardware capability have significantly improved in the last years, enabling them to safely fly autonomously over relatively long distances, beyond visual line of sight (BVLOS). However, less attention had been given to providing accurate positioning data during long distance UAV flights without using GNSS. This is an essential matter since UAV are complex systems in need of a failsafe mechanism that can be automatically engaged when GNSS signal becomes unavailable due to various reasons (e.g., jamming, spoofing [4] or severe weather conditions).

At the same time, image processing capabilities using deep neural networks have been constantly improving in the last decade, enabling even real-time recognition of objects and segmentation. This observation also holds for the processing of satellite image data, where artificial intelligence (AI) models are trained to differentiate between important objects such as buildings and rivers [5].

Building on top of available open source technology and algorithms, this paper aims to provide a reliable positioning mechanism for UAVs by only using RGB photographs provided by a camera mounted on the drone and satellite images of the flight area. The method is based on deep neural image segmentation that enables the localization of the vehicle by extracting recognizable features of buildings, roads, rivers and forest edges. These features will be further mapped to static satellite images using techniques such as template search, scale-invariant feature transform (SIFT) and graph neural networks based matcher.

1.1 Significance and motivation

The main goal of the project is to develop a localization algorithm that does not rely on GNSS for long-distance flights. The result will prove its utility in situations where GNSS signal cannot be reliably used and as a failsafe alternative that enables the drone to reach its goal position or at least land safely in a pre-established location. New commercial implementations such as autonomous drone delivery could use the new localization method to improve the reliability of navigation.

The core attribute of the project is the environment in which vision-based localization is provided: *in the wild*, denoting natural (non-urban) environments where artificial structures such as buildings and roads are sparse. This characteristic transforms what would be a trivial feature matching and homography computation problem inside a city into a challenging process, due to the difficulty of finding salient features in natural environments. Nevertheless, the final implementation is able to provide accurate localization results using a neural network for feature matching between drone photographs and satellite images.

Another significant feature of the implemented localization algorithm is that the pre-mapping process does not require the UAV to fly. The map is built using exclusively open source satellite images, with the objective of enabling autonomous localization in any area where the drone can fly, assuming it is legally and physically possible (due to harsh environmental conditions).

1.2 Related works

Previous studies directly related to this project include semantic segmentation based path planning [6], localization using open source Google Earth (GE) aerial images [7] and pose estimation with neural networks trained with georeferenced satellite photographs [8]. In addition to these, an open source image segmentation model originally used for building virtual worlds [9] proved its utility in providing useful input both for localization and navigation purposes.

Additionally, a graph neural network that provides feature computation and matching for outdoor images [10] became part of the implementation in the current project due to its improved performance, as described in Section 3.4. The model proved its efficient application not only for matching features, but also for the perspective transformations (namely homography) used in computing geographical coordinates when running the implemented vision-based localization algorithm.

Another related paper worth mentioning is [11], which uses an advanced template-based matching algorithm to compute the pose estimation of a UAV using only images. Because of insufficient computation resources at the time, no implementation is provided on the onboard computer of a drone. Another major difference, compared to this project, is that the drone navigates specifically urban environments, where computing and matching visual features is much less of a challenge, compared to natural, *in the wild*, flying areas.

1.3 Contribution

This thesis covers the study of different methods for vision-based GNSS-Free localization as well as the practical implementation and test of a selected approach, which was deemed to be the most technically feasible and robust. The core contributions of this thesis are the following:

1. Study three different approaches (template matching, SIFT features and deep features) to provide GNSS-Free vision-based localization for UAVs.
2. Design and implement a system for GNSS-Free localization that can run on an onboard a computer of a UAV based on matching deep features from live images with a pre-built satellite map made of georeferenced open-source images.
3. Provide a working framework based on 5G connectivity that enables remote operation of a drone and also video streaming over the mobile network.
4. Gather datasets consisting of RGB photographs and solid state LIDAR point cloud that enables the implementation and testing of future vision-based and LIDAR-based GNSS-Free localization algorithms for UAVs.

1.4 Structure

This thesis is organized in 6 chapters, as described below:

- **Chapter 2** offers a brief description of the open source hardware and software systems that represent the foundation on which this project was developed.
- **Chapter 3** describes the individual components used in implementing the system, motivates the essential design choices and provides an overview concerning the functioning of the entire project.

- **Chapter 4** is mainly concerned with the detailed description of the software and hardware subsystems that represent the main contribution of the thesis.
- **Chapter 5** presents the experimental results of the project.
- Finally, the conclusion of the project and ideas for further research can be found in **Chapter 6**.

2 Background

This chapter provides an overview of the main software and hardware building blocks on which this project is based on. Section 2.6 is especially important since it describes the feature matching mechanisms used in the developed localization algorithm.

2.1 Robot Operating System

The Robot Operating System (ROS) is an open-source middleware platform that enables and accelerates the development of software designed specifically for application in robotics. The main advantages of using ROS is the possibility to reuse code from previous projects and its communication framework that facilitates sensor fusion and considerably reduces the time needed for implementing algorithms. For example, replacing a hardware component (i.e. sensor) with a different version on a robot running ROS does not involve tedious and time-consuming tasks such as rewriting firmware components, but simply updating an interface – called package – that interacts with the component. The backbone of ROS is represented by its asynchronous communication via messages published by different nodes on multiple topics, depending on the data type and its origin. A simplified diagram of this type of communication can be observed in Figure 2.1.

The widespread use of ROS 1 lead to the development of ROS 2, which is an improved version used both in research [13] and commercial applications. There are many reasons for which the second version was rebuilt on different principles compared to the legacy one, such as security concerns, modularity and decentralization. To demonstrate

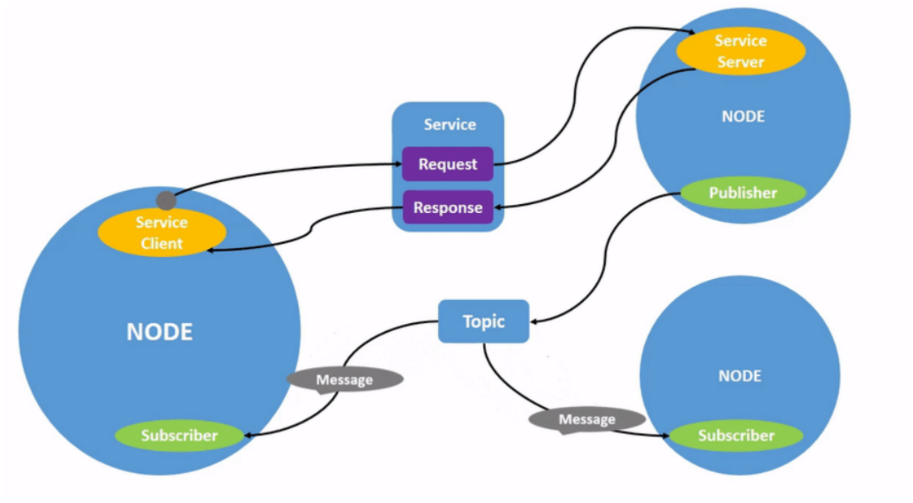


Figure 2.1: Communication framework of ROS [12]

the reliability and performance of the system, it is sufficient to mention that ROS already plays an essential role in developing mission-critical code for space missions organized by NASA [14]. One of the main improvements is the use of an open communication standard based on the publish-subscribe model called Data Distribution Service (DDS). Following its wide acceptance in both research and commercial application, a multitude of ready-to-use software packages based on ROS were developed, considerably accelerating the development of new applications in the field of robotics.

In this project, both versions of ROS were used for simulations, data gathering and experiments involving autonomous flight of the UAVs. Using a programmable flight controller called PX4 (see Section 2.2 for details), ROS is a fundamental software system of a UAV, enabling the real-time analysis of data from different sensors such as cameras, GNSS rovers, IMUs etc. The main advantage of using ROS 1 together with the PX4 flight controller is the robustness of its implementation (it rarely crashes compared to the ROS 2 version) and the simplicity of setting up the development environment (it features pre-built Linux binaries for the installation). On the other hand, ROS 2 offers a multitude of new features and a communication bridge with the flight controller based directly on DDS, instead of being reliant on MAVLink protocol, which is supposed to reduce the overhead of exchanging messages.

2.2 PX4 Autopilot

PX4 Autopilot [15] is an open-source flight controller software intended to run on a multitude of hardware boards called Flight Management Units (FMU) designed by different manufacturers using the same implementation standards. It can be easily integrated with ROS and simulation environments such as AirSim or Gazebo, enabling the development and testing of autonomous navigation algorithms. The PX4 controller uses an extended Kalman filter to aggregate and process data from different sensors: GNSS unit, barometer, inertial measurement unit (IMU). The sensor fusion capabilities allow the developer to focus on high level algorithms like path planning and tracking, instead of dealing with the complexities of controlling a hard real-time system with 6 degrees of freedom, which is the UAV.

2.2.1 Flying modes

One of the most important features, especially from a safety standpoint, of PX4 is the possibility of using different so-called *Flight Modes* to control the drone. For example, the flight mode featuring the easiest manual control is *Position mode*, which represents an important failsafe mechanism when autonomous navigation is no longer feasible. Position mode relies on accurate GNSS localization data to maintain the current altitude and position of the drone, in spite of external factors such as wind speed.

2.2.2 Flight parameters

Due to the high complexity of flight control algorithms which rely on multiple sensors and control loops, there is a need to easily adjust certain parameters in order to safely and reliably fly depending on external conditions. This is the reason the PX4 controller offers the possibility of modifying a set consisting of hundreds of different parameters, some of them being of critical importance. For example, a parameter called *RTL_RETURN_ALT* controls the altitude to which the drone will ascend or descend once the UAV engages the return-to-home action (can be activated manually or automatically as a failsafe mechanism upon sensor failure). More details about critical PX4 parameters and their values can be found in Section 4.3.2.

2.2.3 Advanced features

In addition to the necessary features that enable basic flying and control characteristics, PX4 also offers optional advanced modules that can be used to integrate additional sensors or activate special flying modes. For instance, a precision landing mode can be used if the UAV is equipped with either a LIDAR (Light Detection and Ranging) or infrared sensor [16]. Another useful advanced feature is the use of RTK GNSS localization systems that can provide centimeter level accuracy for reliable ground truth measurements and precise control of the UAV.

Another important feature provided by PX4 is the automatic logging of parameters and sensor states for each flight. Essential variables such as GNSS location, heading, velocity are recorded, but the logs also provide other meaningful data (such as vibration levels of the frame) that can be used for improving the design and autonomy level of the aircraft. In case of accidents or mid-flight sensor malfunctions, the flight log provides data that can be used for tracing back the root cause of technical or human errors in order to improve future designs and algorithms.

2.3 Bridging PX4 and ROS

There are two possible ways to enable communication between the PX4 flight controller and ROS (either version 1 or 2). The first one, which has been in development for a longer

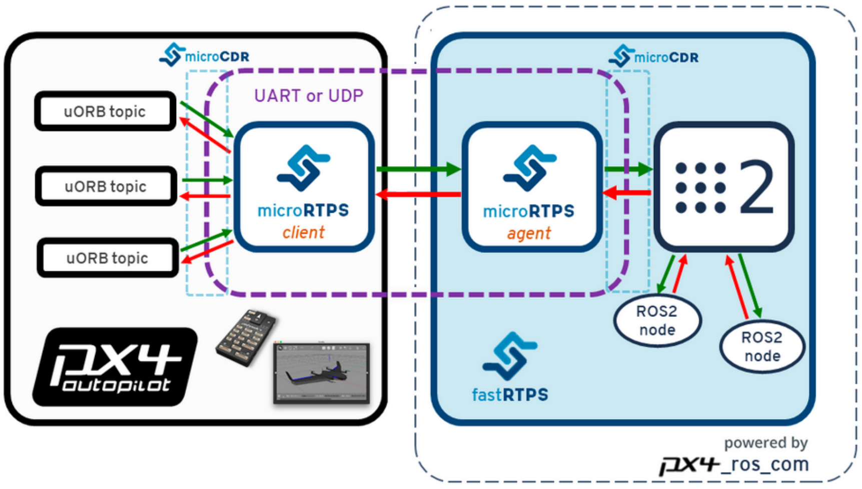


Figure 2.2: High level architecture of PX4 – ROS 2 bridge [17]

times and is more robust, relies on bridging ROS topics to MAVLink messages that can be directly interpreted by PX4. The second method, recommended by the PX4 developers, implies using a DDS implementation on the flight controller called *microRTPS bridge*, natively supported by ROS 2. This enables the flight controller and ROS 2 instance to communicate with less overhead compared to the first bridging method, making this solution more appropriate for real-time exchange of messages, as can be observed in Figure 2.2.

The reasons for adding this extra layer of complexity are stated in Section 2.1. Enabling real-time communication between ROS and PX4 allows the developer to make full use of the sensor fusion capabilities of the flight controller without considering low level control mechanisms when designing autonomous flight algorithms. In this way, a safe method of testing experimental localization, path planning and path tracking algorithms is being made possible. In case of malfunctions, it is always possible to trigger failsafe mechanisms, such as returning automatically to the taking off position or enabling manual control of the drone.

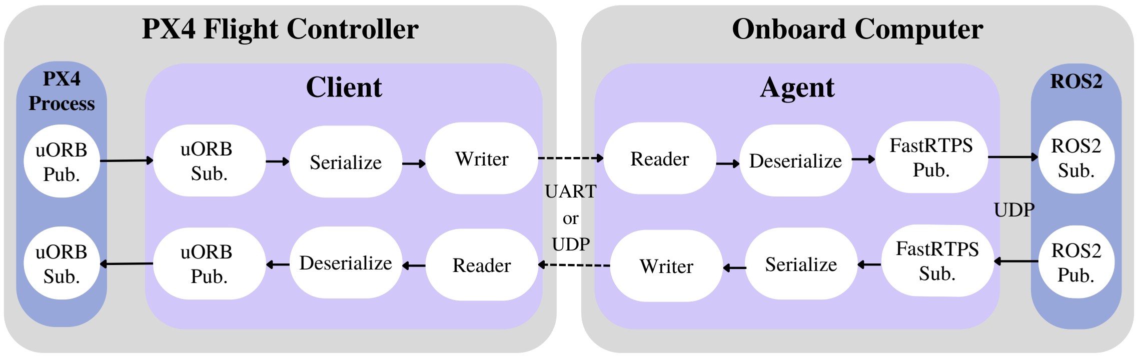


Figure 2.3: Detailed interface of microRTPS bridge [17]

2.3.1 Interfacing PX4 with ROS 2

As mentioned above, instead of directly relaying MAVLink messages to PX4 – a major limitation since the protocol was initially designed for low bandwidth use which can be easily exceeded – ROS 2 uses an interconnection application called *microRTPS*. This program, instead of relying on MAVLink to transmit messages, uses either a Universal asynchronous receiver-transmitter (UART) or User Datagram Protocol (UDP) connection, which allows for more flexibility when designing the communication and processing system of a UAV. For instance, in addition to more bandwidth and lower latency, the microRTPS bridge also allows the PX4 controller to communicate with a computer which is not physically installed on the drone, but it is part of the same virtual network. More details about the advantages and the added flexibility of such a design can be found in Section 4.4.

Another major advantage of the PX4 – ROS 2 bridge implementation is that it enables the use of simulators such as Gazebo or AirSim where offboard control scripts can be rigorously tested before implementing them on real UAVs. This is an obvious safety advantage, since the algorithms that provide autonomous navigation on drones are mission-critical systems in which failure rates and bugs must be kept at a minimum. For this reason, PX4 can be used with either Software-in-the-Loop (SITL) or Hardware-in-the-Loop

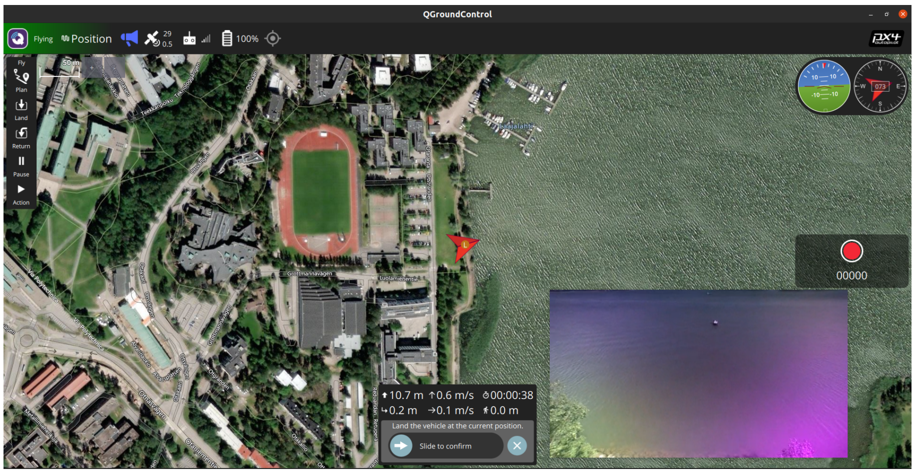


Figure 2.4: Interface of the ground control station during flight

(HITL) simulation modes [18]. The former uses a completely simulated environment (including as fake sensor data), while the latter runs the code on a physical PX4 controller and uses real sensor data. In the case of SITL simulation, the interfacing between the flight controller, QGC, path planner algorithm and the actual simulator is made through UDP ports. In addition, HITL simulation uses serial communication, namely UART, to exchange messages between the physical PX4 controller and the simulator.

2.4 QGroundControl

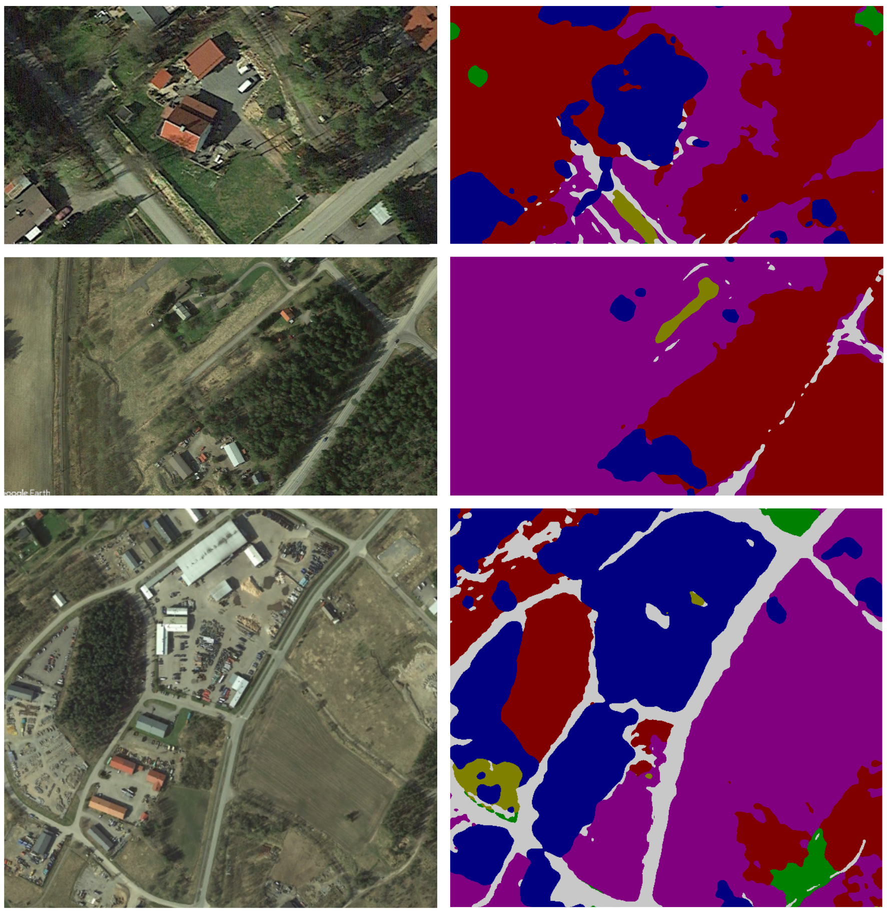
QGroundControl (QGC) is a cross-platform open-source (runs on Windows, Mac OS X and Linux) ground control station for manually controlled or autonomous micro aerial vehicles (MAV). Its main purpose is to provide a facile way to interact with the drone, supervise the correct functioning of the onboard sensors and act as a failsafe mechanism in case of emergency. Additionally, it allows upgrading the firmware of the flight controller and modify the flight parameters depending on the flight conditions. For example, when flying indoors, certain parameters need to be changed accordingly in order for the flight controller to use the localization data provided by the motion capture system instead of the GNSS sensor (see Section 4.3.2).

In Figure 2.4 the interface of QGC running on Ubuntu 20.04 can be observed. The drone is being manually controlled (using *Position* flight mode) and a third party application provides the real-time feed of the onboard camera. One of the essential features of the control station is that is uses a separate radio (or internet based) link for transmitting telemetry data – current GNSS position, heading, altitude and velocity being the most important. In addition, it allows for switching between flying modes, emergency land or return the MAV to its take off location independently of the radio transmitter used for manually controlling the drone. This essentially means that QGC also acts as a manual failsafe mechanism in case the UAV malfunctions while flying in autonomous mode and cannot be controlled using the main radio controller.

2.5 Satellite image segmentation

There is a multitude of remarkably accurate deep neural networks intended for segmenting aerial images taken with satellite cameras [19]. Most models are trained for specifically segmenting building footprints in urban areas [5], but in this project a reliable pre-trained model that could differentiate between at least 3 classes (buildings, forests, fields) was necessary. This specific requirement was the main reason for choosing an open-source model [9] based on Swin Transformer [20]. In short, [9] provides a model pre-trained using satellite photographs with 1000×1000 pixel resolution and 500×500 meters geographical footprint. The photographs used for training were taken in forest areas from France, so geographical zones that are quite similar to the ones in Finland, on which the localization algorithm developed in this paper was tested. The model effectively differentiates between 6 classes representing different ground surfaces as seen from an aerial perspective of a camera pointed directly towards the earth (90° relative to the ground surface).

Figure 2.5 shows 3 examples of segmented satellite images taken from different *eye*



Color	🟥	🟩	🟨	🟪	🟦	⬜
Surface	Dense forest	Sparse forest	Moor	Field	Building	Road

Figure 2.5: Segmentation of GE satellite images from different altitudes

altitudes (the approximated altitude of a virtual camera that would take the photograph) in GE. From top to bottom, the corresponding eye altitudes are approximately 100 m, 500 m and 1 km, respectively. Since the neural network that implements the segmentation [9] had been trained with square map sections with the side length of 500 m, the best results are provided for relatively high altitudes, of 1 km. When segmenting low altitude images, the model is unable to reliably distinguish roads and to differentiate between sparse and dense forests.

The most important classes are dense forests, buildings and roads because they provide good tracking features (especially artificial structures) used in feature matching of the photographs. Admittedly, there are certain disadvantages of using an already trained model, such as the fact that the training images are taken from a perspective corresponding to a higher altitude than the one used for collecting data with the UAV (120 meters due to legal considerations). This mismatch causes the model to sometimes output lower accuracy segmentation results, such as confusing a dense forest with a plain field. Nevertheless, the model proved to be reliable enough for enhancing the localization algorithm, especially when using template matching (see Section 2.6.1).

2.6 Image matching – useful concepts

The main objective of this project is finding a robust and fast enough technique of calculating the geographical location by matching photographs taken by a camera mounted on a drone with georeferenced satellite images stored on the onboard computer of the UAV. The core of the problem is implementing an image matching technique that finds the best correlation between what the drone camera captures at a given time and the onboard map. A visual representation of these concepts is found in Figure 2.6 that shows an example where a lower altitude image is identified as being part of a larger satellite photograph using the techniques introduced in the sections below.

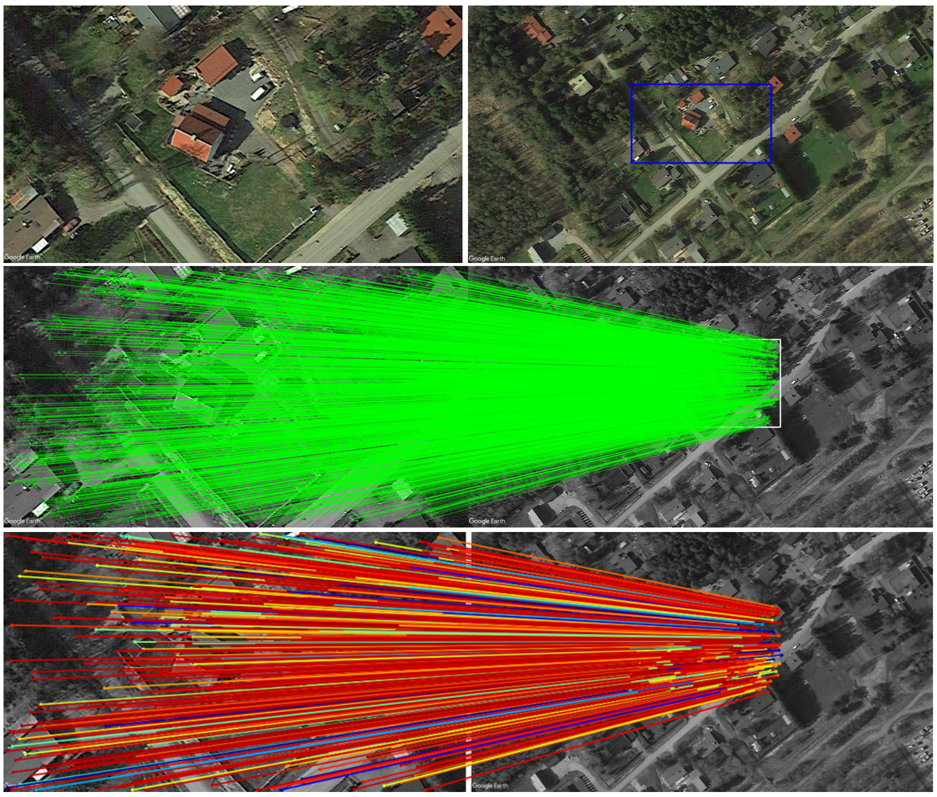


Figure 2.6: Comparison between the three studied methods: template matching (top), SIFT features (middle), deep features (bottom)

.

The simple scenario showcased in Figure 2.6 is characterized by some simplifications which make the feature matching process less challenging than using real datasets of drone photographs, with some difficulties listed and explained in Section 5.2. Nonetheless, this way of simulating drone camera perspective by using satellite imagery has proven its usefulness by providing a method to test and calibrate the three studied feature matching techniques.

2.6.1 Template matching

A widely used *classical* (in the sense that it does not involve machine learning) approach in image recognition and photogrammetry is **template matching**. Briefly, the concept is focused on finding a known pattern (small image) in a larger picture. The searched template, characterized by its size (p,q) is progressively shifted along the horizontal (j coordinate) and vertical axes (i coordinate) of the larger image for a given number of pixels. For each shifting iteration, a cost function is calculated, which returns the degree of similarity between the searched patch and the bigger picture [21]. At the end of the described algorithms, a pair of *(u,v)* coordinates is returned, representing the most probable location of the searched template (see Figure 2.7). An important note here is that the result is characterized by a confidence level, meaning that the found pattern might not be actually contained in the image. This statistical nature of the algorithm needs to be addressed in order for template matching to be a reliable method in mapping and localization.

There are two main advantages of using template matching:

- Invariance to contrast, brightness and translation
- Relatively simple implementation

The main assumption when performing template matching on two photographs is that they only differ in contrast, brightness and translation (the relative position of the template in the bigger image) [22]. Its relatively simple implementation allows speeding up the

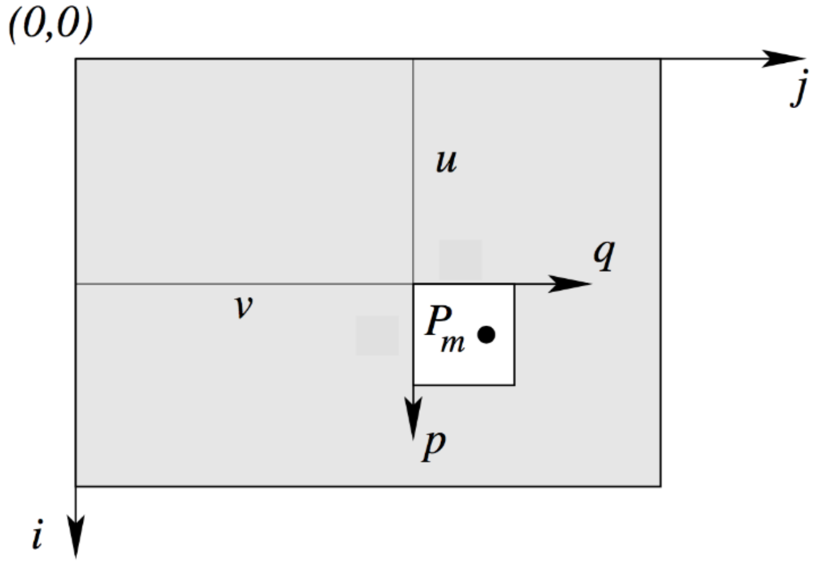


Figure 2.7: Main principle of template matching: the pattern $\mathbf{P_m}$ is matched against the gray area representing the larger image [22]

algorithm even on embedded systems, which is an essential feature for a technique used in real-time systems such as UAVs. In spite of its simplicity, the main drawback of using the template match technique is its lack of robustness for instances when the searched pattern is rotated, relative to its position in the larger image. This essentially means that the degree of similarity between the pattern and the bigger picture significantly decreases, even for small (under 5°) rotations. An obvious solution would be to run the template matching algorithm multiple times, for different rotations, but this greatly increases the processing time, making the technique inappropriate for a real-time application.

2.6.2 Scale invariant feature transform

Another widely known and extensively tested image matching technique is *Scale invariant feature transform* (SIFT) [23]. Originally used for finding partially occluded objects in a given image, the approach provides good results for matching two pictures of the same size. Firstly, the transform computes the SIFT keys for both images, which is a collection of salient features of the objects found in the photographs. Secondly, using a nearest-neighbor technique, the features found in the first image are matched to the ones in the second image. Two SIFT keys are considered to describe the same image feature if they have a minimum Euclidean distance between the two vectors corresponding to the keys [24]. This approach ensures that the probability of the matched keys to actually represent the same object feature in the two images is maximized.

Compared to template matching, SIFT is also partially invariant to rotation and provides reliable results even with partial occlusion of the searched object. Another major advantage is its relatively fast computation and matching of key features, allowing the processing of approximately 2 frames per seconds (FPS) [23]. These characteristics make the technique a good potential candidate for matching the drone camera feed to the map comprised of satellite images.

2.6.3 Deep feature matching based on neural networks

Classical heuristic approaches such as template matching and SIFT can be enhanced or altogether replaced by a neural network trained on a specific dataset, such as indoor images or outdoor landscapes. Since image matching consists of two main steps – feature detection followed by feature matching – either one of them (or both) can be computed using neural networks. For example, SuperGlue is a supervised graph neural network that takes as input previously computed features and matches them using the Sinkhorn algorithm [10]. SuperGlue can be used with features provided by classical approaches like SIFT or it can be coupled to another neural network, such as SuperPoint [25], which generates the features that need to be matched.

The main advantage of AI models used in matching are the speedup of the process (approximately 14 FPS [10]) and the improved precision of the technique, providing more accurate results and fewer false-positives when matching features. Obviously, this advantage is shadowed by the fact that using supervised neural networks requires a lot of resources for training the model in the first place, by manually annotating large datasets. Conversely, a pre-trained model can be used in a given application, even if the inference data is slightly different from the training dataset, with the effect that accuracy of the matching will decrease. Ideally, if time and financial resources allow it, the best approach would be to retrain the neural networks used in feature computing and matching using specific datasets that pertain to the given application of the model, in this case aerial images. Nevertheless, even with a pre-trained model, neural networks provide the best performance in terms of computation speed and matching accuracy, as can be seen in Chapter 3.

2.7 Legal considerations for flying UAVs

In all the European Union states, UAVs are highly regulated systems and different regulations apply to them based on their take off weight and level of autonomy [26]. There are 3 main categories of UAVs, divided into A1, A2 and A3 classes, based mainly on their weight (maximum of 900 grams, 4 kg and 25 kg, respectively) and the minimum flight distance to uninvolved persons. If a drone weighs more than 250 grams, it requires its remote pilot to have a license issued by the national authority for regulating aircraft systems, obtained after passing an exam.

The most important rules to follow when flying a drone involve never exceeding a maximum altitude of 120 m AGL, complying with no flight zones (inside urban or residential areas) and maintaining a minimum safety distance of at least 100 m between the UAV and uninvolved participants. These mandatory flying rules have been followed during the experiments in Chapter 5 and explain why flying at higher altitudes and above areas with multiple buildings (which could improve the vision-based localization accuracy) was not possible.

3 Design overview

In this chapter, the software design choices of the localization algorithm are discussed in detail and a brief survey of three feature matching methods for RGB photographs is provided. The chapter offers a qualitative analysis that aims to provide arguments for choosing algorithms used as essential building blocks described in Chapter 4. Finally, one of the feature matching techniques, based on a pre-trained graph neural network, is preferred for the final implementation due to its real-time performance and high accuracy.

3.1 Flowchart of the system

Before describing in detail each one of individual software modules on which the localization algorithm is based on, it is useful to offer an overview of the entire system. The basic requirements of the algorithm are to consider as input an image taken with any type of camera (wide angle is preferred) and compute the output, which is a pair of absolute coordinates *latitude, longitude*. Compared to localization methods based on visual features used for navigating indoor environments [27] [3], the constraints for providing accurate results are more relaxed. Centimeter level accuracy of the positioning mechanism is necessary when the UAV is navigating a building, but not when flying outdoors at high altitude (more than 100 meters AGL). There are multiple reasons to justify this, the main one being the difficulty of obtaining such a high accuracy considering that the tracked image features are situated tens of meters away from the camera. Another reason is that artificial objects, such as buildings, provide surfaces with more features to track,

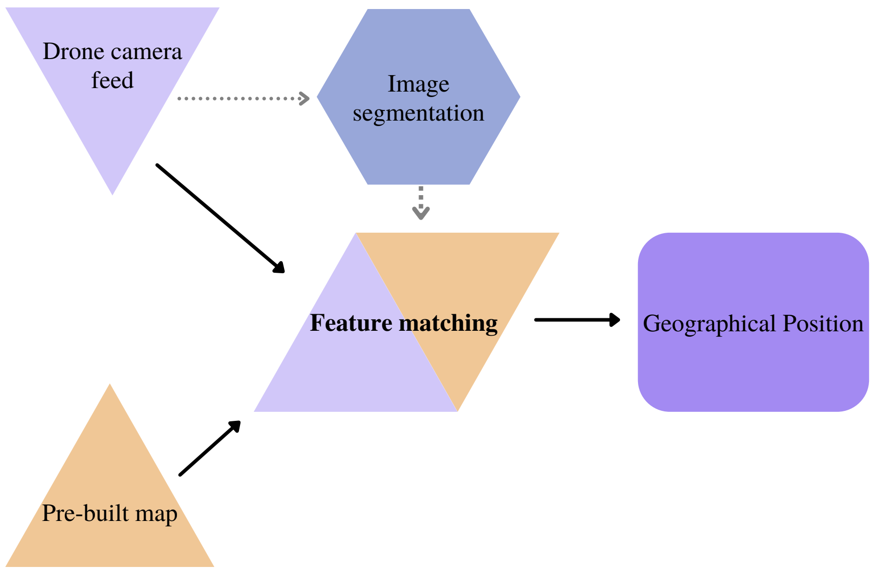


Figure 3.1: Overview of the system

compared to natural ones like forest edges.

Figure 3.1 provides a general understanding of the system, its inputs (RGB camera feed, prebuilt map) and output (geographical position of the drone). The core of the design is the feature matching mechanism which generates and matches noteworthy features of images from two sources: downward facing camera mounted on the drone and a list of aerial images taken from GE that basically comprise the map of the flight zone. The image segmentation plays in important role in this process because it enables the feature matching algorithms to only use parts of the camera feed which contain salient features. For example, if the camera feed is pointed at a building (high number of features to be matched) situated in a grassland (featureless area), the image segmentation module provides as input for the feature matcher only the part of the image which contain the building. This essential feature of the localization system improves both the processing speed of the images to be matched and the positioning accuracy.

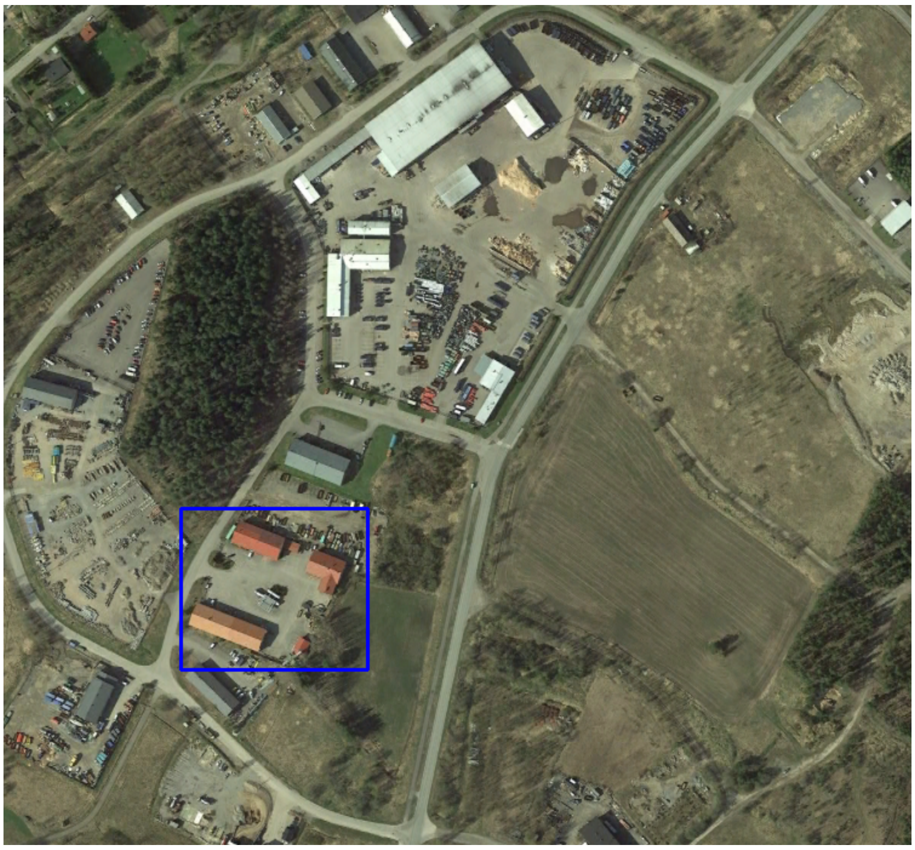


Figure 3.2: Template match localization of a small patch in a satellite image

3.2 Template matching – sensitive to rotation

In order to check if template matching represents a good approach for implementing the localization algorithm, a small test was designed using only satellite images from GE as data. A sample OpenCV implementation [28] was used as a baseline and modified accordingly for aerial images.

An important change was integrating the image segmentation model described in Section 3.5. Simply using the whole image provided by the RGB camera for matching would provide fairly inaccurate results. Consequently, the image segmentation model was used to identify buildings present in the image. After this important step, only the section of the image that contains buildings is used for template matching because artificial structures provide distinct features which yield accurate localization results. An example of the output of this implementation can be seen in Figure 3.2, where a small patch (inside the blue rectangle) was successfully located inside the satellite image, providing a high altitude perspective of the ground area.

At first, template matching coupled with image segmentation provided encouraging results because the small patches (representing a simulated perspective of the drone camera) could be located successfully in the satellite images, provided that they have precisely the same orientation. If the orientation of the query image is even slightly different (less than 10°) from the one in which the small patch is matched, then the results are not reliable anymore. Of course, heading data from the integrated flight controller compass could be used to properly align the drone pictures with the on onboard satellite images, but not in all types of conditions. Since the magnetic compass of the drone can be easily affected by interference from sources such as high voltage power lines, a method which is more robust against rotation had to be found.

3.3 SIFT – inaccurate for real photograps

Template matching proved to be too sensitive to rotation to be used in a localization algorithm. As a result of this, a SIFT feature matcher was used, using the same test approach like in Section 3.2, locating a low altitude satellite image inside a higher altitude one. In spite of its relative slowness (only 2 images could be processed each second), SIFT seemed to be as accurate as template matching and, more importantly, more resistant against rotation. An example output of the implementation can be seen in Figure 3.3, where green lines represent the link between matched features in the two test images.

The results using simulated drone perspective from satellite data were encouraging enough, so that the SIFT approach could be tested on real data. When trying the SIFT feature matching algorithm on real drone photographs (see Chapter 5), it could no longer provide accurate results. The explanation for this expected behavior lie in the characteris-

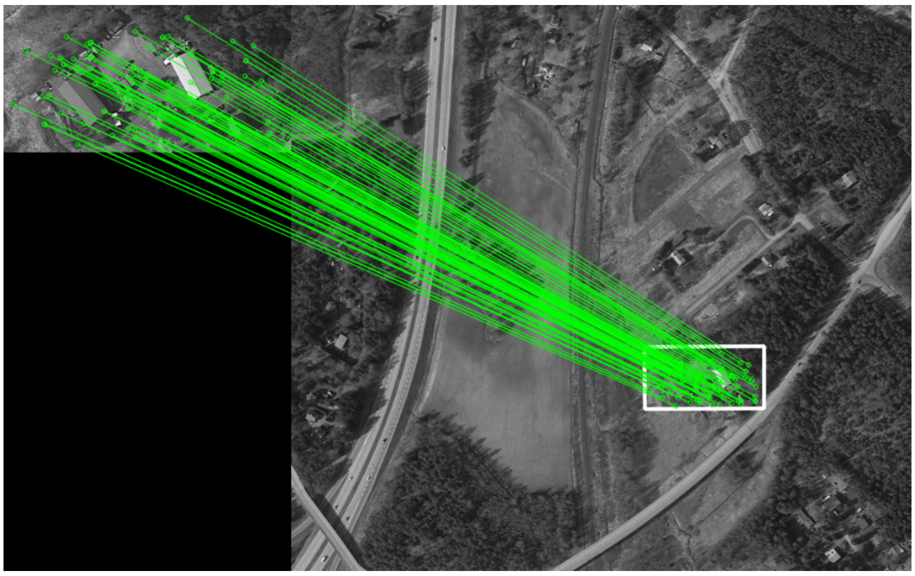


Figure 3.3: SIFT features match example

tics of real photographs, which are highly dependent on factors such as weather, seasons and time of day when the picture was taken. For example, shades caused by the relative position of the sun during the day can have a significant impact on feature matching due to occlusion.

SIFT based feature matching was so unreliable for real data, that it could not be effectively used to calculate geographical coordinates. When iterating through the satellite images that form the map of the flight area (see Section 3.6.1 for additional details), roughly the same number of features were identified between the real photograph and all the satellite pictures. This result made it impossible to actually locate more than a few of the real drone pictures, resulting in SIFT being an inappropriate method to be used in the localization algorithm. Consequently, a new method based on a graph neural network [10] for feature matching had to be designed and implemented, described in the next section.

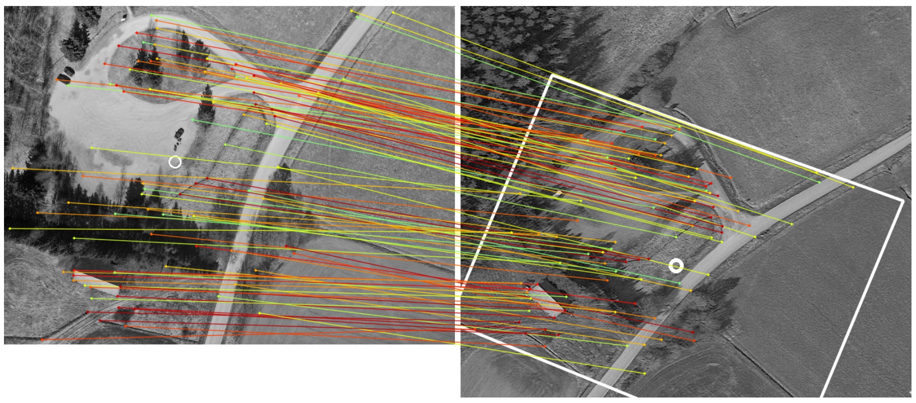


Figure 3.4: Superglue feature based localization

3.4 Matching salient features of aerial RGB photographs

After the tests performed with template matching and SIFT were considered not accurate enough for the localization algorithm, a new method, involving the use of neural networks for both calculating and matching image features, was considered. SuperPoint [25] – used for calculating features – and SuperGlue [10] – used for matching the features – are two coupled neural networks that can reliably find correspondences between two pictures of the same environment taken from different perspectives.

The main reason for choosing this architecture was that it can accurately compute the homography needed for locating an object present in two test images, even if the perspective and lighting conditions of the two samples vary widely. For example, Superglue was used in the original work to match building facades when one picture was taken during the day and another during the night, with significant changes in the perspective. Even so, the homography estimation achieved 90% accuracy [10], which was proof it could also be used for matching aerial images.

Figure 3.4 shows an example output of the modified SuperGlue algorithm, which provides the homography calculation for accurately locating salient features (computed with SuperPoint) that are present in both photographs. On the left (drone perspective), features provided by the distinctive shape of the road and the building can be used to find correspondences in the image on the right (satellite perspective). Even if the rotation is significantly different (more than 20°) and the pictures were taken one year apart from each other, at different times of day and from a different altitude, the feature matcher is able to find enough correspondences for reliable localization. The correspondences marked in green have a higher confidence level of confidence than those marked in red. An important note here is that only correspondences with a confidence level higher or equal to 50% were used to compute the homography, compared to the default value of 20%. The chosen value is high enough to disregard features which would lower the localization accuracy, but low enough so that enough points are found between pictures which vary significantly in rotation, translation and perspective.

3.5 Enhancing localization accuracy by image segmentation

Figure 3.5 shows the output of the image segmentation model [9] chosen for augmenting the template matching (Section 3.2) and SIFT based (Section 3.3) feature matching algorithms. Buildings (colored in blue) and roads (colored in gray) provide salient features which can be tracked with any feature matching algorithm. Forest edges (red) are another example of objects with distinctive shapes which can be reliably localized.

Consider the UAV would have to navigate from the buildings situated in the southern part of the map in Figure 3.5 to the constructions located in the northern side without using GNSS sensors. Accurate visual based localization can only be provided when the camera on the drone can keep track of buildings, roads or forest edges. Consequently, the image segmentation model offers a representation of the flight area which can be used by a path planning algorithm to avoid featureless areas unsuitable for path tracking, as suggested in [6], where only ground truth segmentation was used.

This approach also offers the possibility of using other types of sensors than cameras,

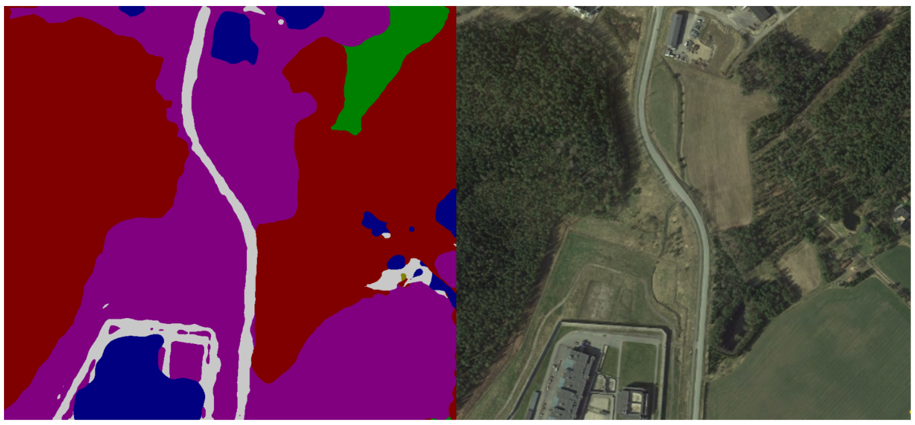


Figure 3.5: Example of a segmented satellite image

which are obviously only suitable during the day and in good weather conditions. As shown in [29], LIDAR sensors provide accurate distance measurements and object tracking even in adverse weather conditions. Compared to cameras (passive sensors), LIDARs are active sensors, which makes them suitable for use at night or when natural light is not sufficient for feature matching.

3.6 Map building

Accurate feature matching of images is only useful if the drone camera photographs can be precisely linked to geographical coordinates. An important assumption of this project is that the flight area of the UAV is known in advance, so a map for that specific zone can be built and uploaded on the onboard computer for offline use. The map is composed of rectangular sections representing RGB satellite images with an approximate resolution of 1400×1200 pixels, collected from GE. Each one of these sections are collected from the same perspective, with the camera view perpendicular on the ground surface and from an altitude that offers a similar field of view to a wide angle camera. One example of these sections can be observed in Figure 3.6, where the transparent white rectangle represents the georeferenced map tile.

3.6.1 Georeferencing map sections

Because the flight area is generally too large to be represented as only one image file, the map is split into different section, each one of them with two distinct geographical coordinates, a *(latitude, longitude)* pair for the top left corner and another one for the right bottom corner. Instead of using a unique pair of coordinates that correspond to the center of the image, it is more accurate to geographically reference two corners of the section. In this manner, any set of features located in the image can be accurately linked to geographical coordinates with a simple equation that relates pixels to latitude:

$$\mathbf{Lat} = Lat_t + \frac{C_y}{H} * (Lat_b - Lat_t) \tag{3.1}$$

where

$\mathbf{Lat}$	= computed latitude
Lat_t	= top left corner latitude
C_y	= vertical pixel coordinate of matched features
H	= height of the satellite image in pixels
Lat_b	= right bottom corner latitude

Using the same logic, longitude is computed as follows:

$$\mathbf{Lon} = Lon_t + \frac{C_x}{W} * (Lon_b - Lon_t) \tag{3.2}$$

where

$\mathbf{Lon}$	= computed longitude
Lon_t	= top left corner longitude
C_x	= horizontal pixel coordinate of matched features
W	= width of the satellite image in pixels
Lon_b	= right bottom corner longitude

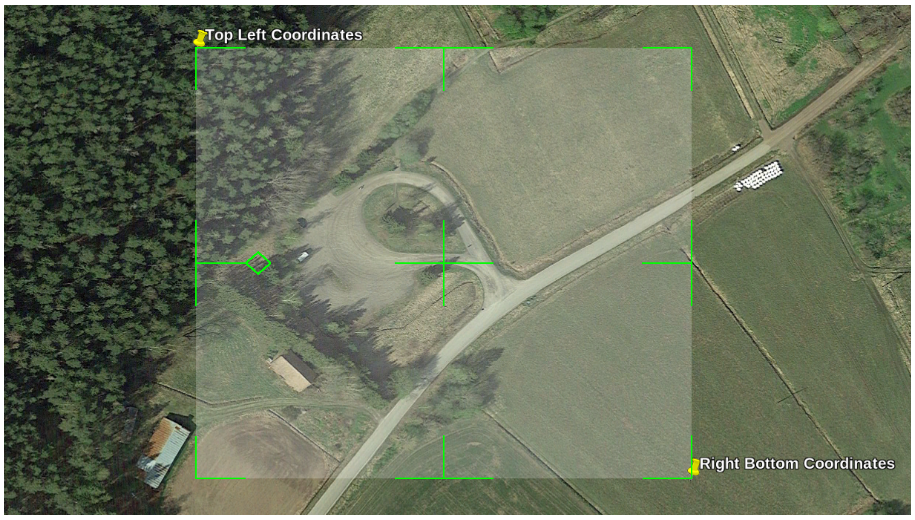


Figure 3.6: A georeferenced section of the map used for image matching

Since any set of located features can be enclosed in a polygon, the center of that polygon can be used as a reference to obtain the C_x and C_y pixel coordinates which represent the center of the polygon. This approach enables the algorithm to pinpoint accurately the geographical coordinates of an image region rich in salient features (e.g., a special shape such as the road in Figure 3.6).

3.6.2 Data Structures

In order to easily process a dataset containing hundreds of wide angle drone camera photographs, two classes were created. Their attributes are listed in Figure 3.7, where bold variables are mandatory values which serve as ground truth, underlined ones are computed and all other optional. All the data describing one drone photograph is read at runtime from a CSV file built by running a script which parses the metadata of each image. This metadata includes essential information such as the absolute GNSS location of the photograph, roll, pitch, yaw of the drone (and gimbal if present) and additional camera settings.

After running the algorithm, if the localization of a photograph was successful, the data will be saved in a CSV which will be used to provide a quantitative and qualitative

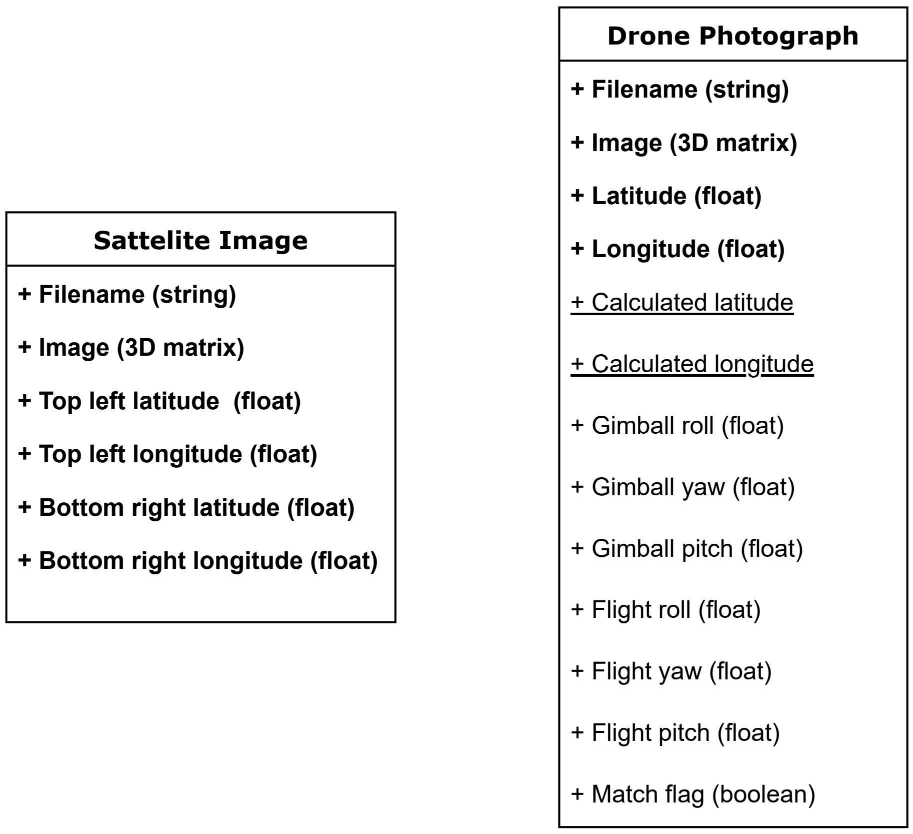


Figure 3.7: Classes used for representing georeferenced satellite and drone images

analysis of the results in Chapter 5. The use of specially designed data structures and CSV files provide a smooth framework for processing datasets and improving the algorithm. Since visualizing datasets consisting of absolute geographical coordinates (latitude and longitude) can prove to be difficult, the coordinates can be easily converted to the metric system using the haversine formula.

4 Implementation and experimental setups

This chapter will provide an in depth description of the localization algorithm based on the SuperGlue feature matcher and also describe the hardware setups which were used for experiments during the project.

4.1 MAVs used for experiments and gathered datasets

Three different drone models were used during the experiments, due to their different characteristics. Holybro X500 V2 (see Figure 4.1), a versatile quadcopter frame, is appropriate for testing different sensors and its relatively small size makes it suitable for safe flights indoors. This is an important aspect for testing new experimental setups, since flying outdoors requires careful planning due to the legal considerations explained in Section 2.7. DJI S1000 is a quadcopter with very limited autonomy, but with the possibility of carrying bigger payloads, such as a LIDAR sensor and its necessary hardware dependencies (see Figure 4.2). For taking photographs from high altitude (120 meters AGL), DJI Matrice 300 quadcopter equipped with a wide angle camera (see Figure 4.4) was considered the most appropriate solution, due to its stability, long range and autonomy, in addition to the possibility of controlling the camera gimbal.

The drones used in the experiments had various roles according to the size, technical specifications and ease of use. For instance, assembling of the Holybro model was

	X500 V2	Matrice 300	S 1000
Maximum takeoff weight	2.5 kg	9 kg	11 kg
Flight Time	18 min	55 min	15 min
Radio Control Range	300 m	8 km	300 m
Battery	4S LiPo	12S LiPo	6S LiPo

Table 4.1: Technical specifications of the MAVs used during experiments

particularly useful in understanding the general hardware architecture of the UAV and how different components (flight controller, telemetry radio, RC control receiver, GNSS sensor, power source etc.) interact together in real-time. Inherent challenges posed by complex hardware designs such as mounting different cameras, choosing the best position for the onboard computer and sensors and ensuring that a newly assembled UAV is ready to fly had to be overcome.

The LIDAR point cloud which can be observed in Figure 4.3 is part of a dataset gathered from different altitudes (50-100 m AGL) using a solid state Livox Avia LIDAR sensor with non-repetitive circular scanning. In optimal lighting conditions (cloudy weather) the range of the sensor reaches 450 m, which is more than enough to provide accurate mapping results even when flying using the maximum legal altitude of 120 m. In the provided figure, it is easy to visually differentiate between important classes of objects (such as trees, fields and buildings), therefore the point could provide a segmented output that can be matched against segmented satellite photos such as the ones found in Section 2.5. This kind of matching is not implemented in the current project, but the gathered point cloud is a building block that can used in further research to test the viability of the described method and enhance long range GNSS-Free autonomy.

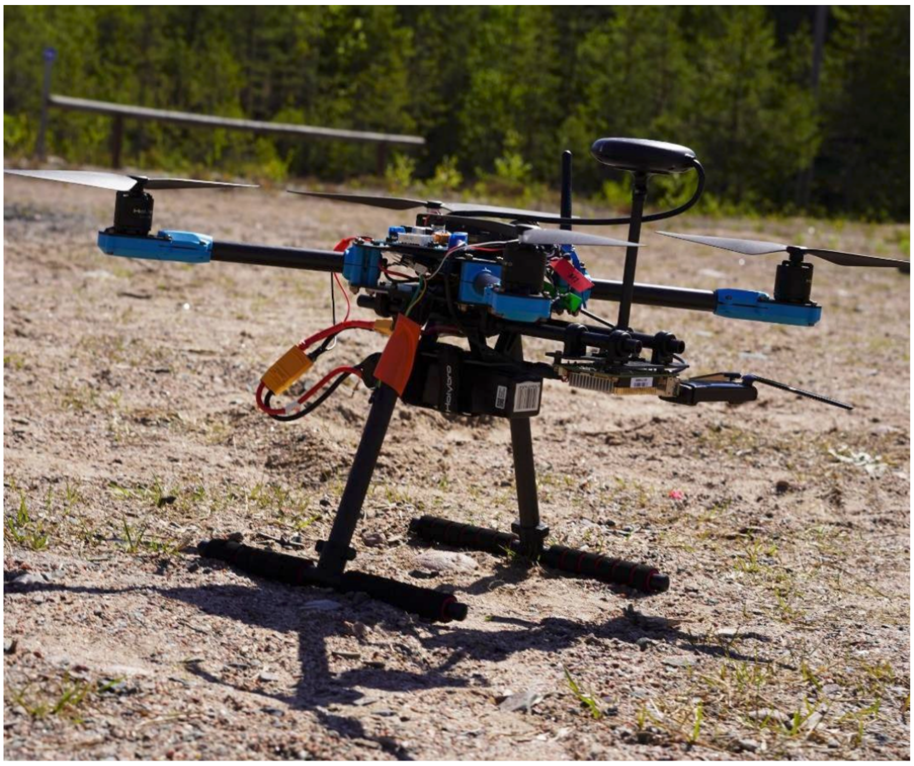


Figure 4.1: Holybro X500 V2 quadcopter equipped with onboard computer

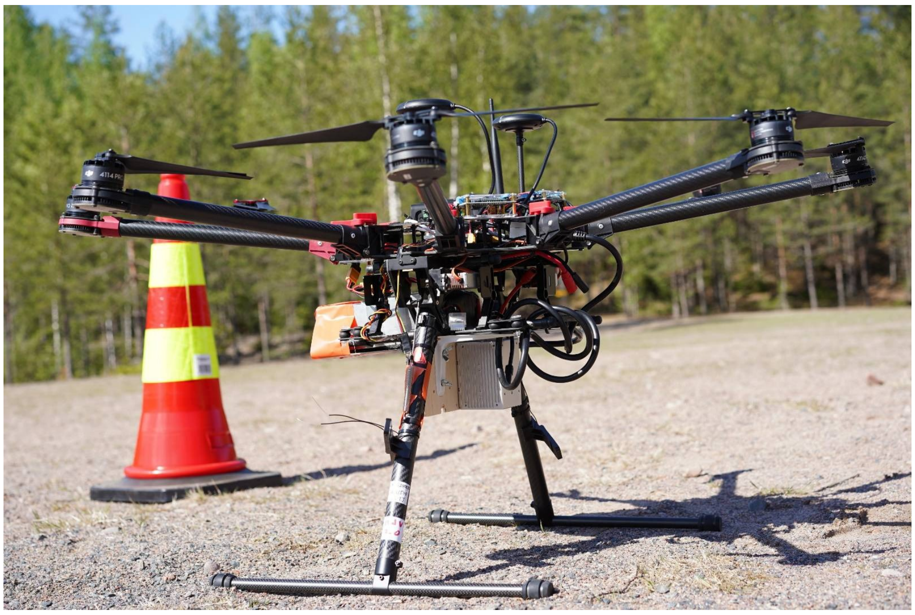


Figure 4.2: DJI S1000 equipped with solid state LIDAR sensor

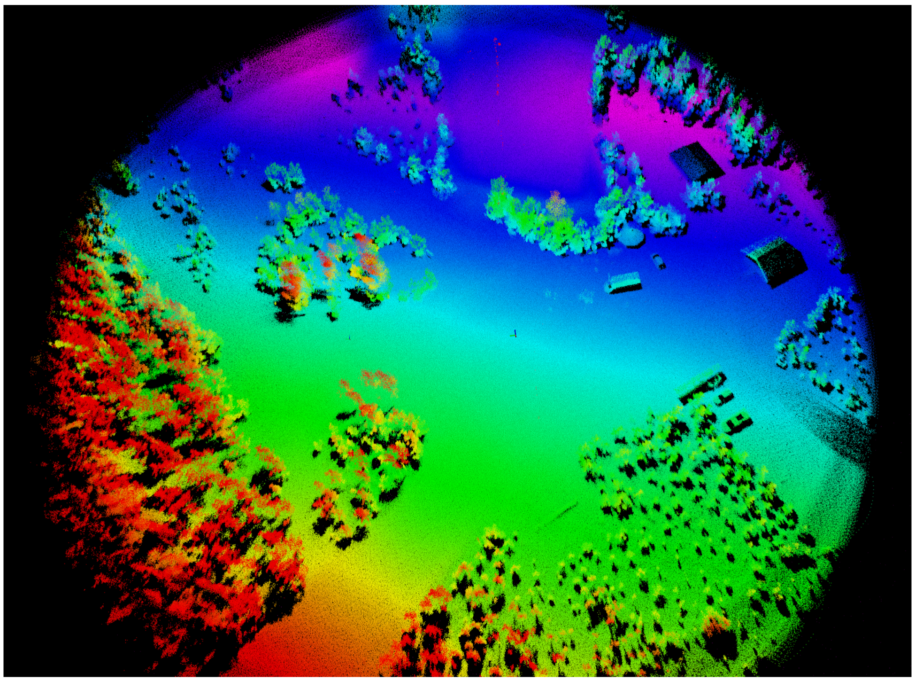


Figure 4.3: Visual representation of an aerial point cloud provided by the solid state LIDAR

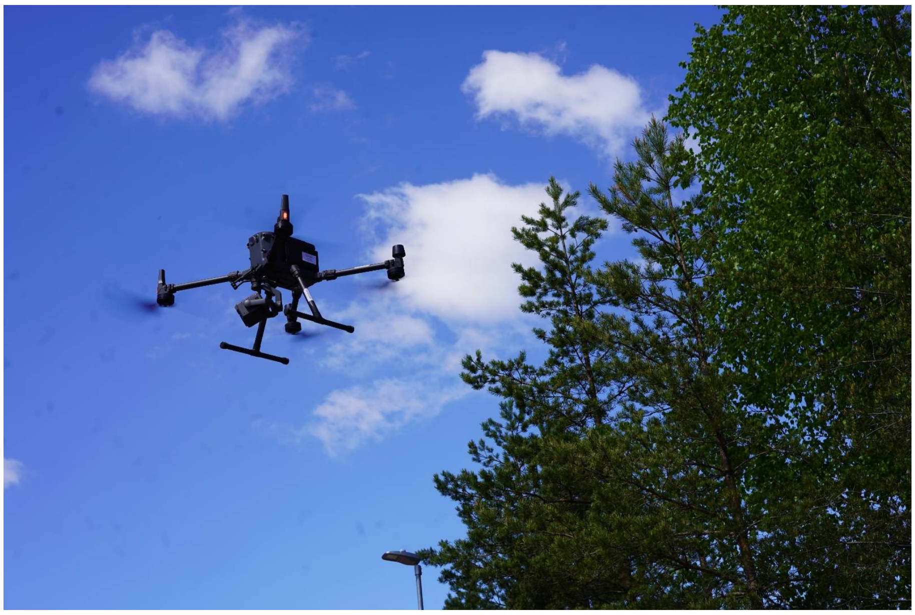


Figure 4.4: DJI Matrice 300 during the data gathering experiment

4.2 Vision-based localization algorithm

Algorithm 1 provides the core functionality of this project's software implementation. The developed method accepts as input RGB photographs (either JPEG or PNG) taken by the downward-facing drone camera and matches them against a pre-built map consisting of satellite georeferenced images (the map can be varying in size, depending on the size of the flying zone). Upon successful localization of a drone captured photograph, the output is a pair of absolute geographical coordinates (latitude and longitude), indicating the position of the aerial vehicle.

In order to compute geographical coordinates, the algorithm uses a fairly simple approach. It starts by rotating the drone photograph using the image metadata providing the orientation of the UAV and the camera gimbal to match the heading of the map sections (always oriented northwards). The rotation is necessary because it improves the number of features which can be matched for a pair of images. Secondly, the features of both images (drone and satellite) are computed and matched against each other. This process is repeated for all the satellite images and the drone is assumed to be located in the map section which has the most matches with the current camera photograph. Following the feature matching process, a perspective transformation is computed, that provides a link between the position of the matched features in the georeferenced map image and their location in the drone photograph. Since the matched map section is a rectangle with *a priori* known geographical coordinates, it is possible to convert from pixel coordinates to latitude and longitude in a straightforward manner (see Section 3.6.1 for the detailed calculation method).

An important parameter of the algorithm is the method used for differentiating between the matched features which represent inliers and the ones which are outliers. The method is called random sample consensus (RANSAC) and it improves accuracy of the localization algorithm by selecting only features which are relatively close together. These valid features are further used for the computation of the perspective transform and out-

Algorithm 1: Vision-based localization

```
Input:
    RGB camera drone photograph: $dronePhoto$
    Pre-built satellite map with $N$ georeferenced photographs
Output:
    Absolute coordinates: $(latitude, longitude)$
rotate$(dronePhoto, degrees = droneYaw)$
for $satellitePhoto = 1, N$ do
    compute_features$(dronePhoto, satellitePhoto)$
    matchesList = match_features$(dronePhoto, satellitePhoto)$
    if $length(matchesList) >= maxMatches$ then
        maxMatches = $length(matchesList)$
        matchedPhoto = satellitePhoto
        transMat = findHomography$(matchesList, method = RANSAC)$
        polygon = perspectiveTransform$(transMat)$
        pixelCoordinates = center_of$(polygon)$
compute_geo_coordinates$(pixelCoordinates, matchedPhoto)$
```

liers are discarded.

The performance of the algorithm is tested on real drone photographs and the results are analyzed in Section 5.1. The minimum number of matched features between a pair of photographs that enable the computation of the drone position is four, otherwise the perspective transform cannot be calculated. If the drone flies over featureless surfaces such as fields or forests, an incorrect section of the map could be matched with the camera photograph, resulting in erroneous geographical coordinates.

4.3 Offboard control

4.3.1 Autonomous navigation

Algorithm 2 presents a simple finite state machine with four states that enable the UAV to fly through designated waypoints (can be defined by the user or by a path planner). In order for the drone to fly in *Offboard* (autonomous) mode, the FMU needs to receive waypoints in the form of three-dimensional spatial coordinates (x, y, z) with a frequency higher than 2 Hz. This represents a failsafe mechanism that ensures the safety of the flight in case of hardware or software errors that would otherwise cause the drone to lose control and crash.

For increased safety, offboard control can only be enabled if the drones has been previously armed through an independent channel (radio control) by a human pilot. Even if arming inside the script is possible, it is an inadvisable choice, since autonomous flight algorithms need to be rigorously tested before flying. This is the main reason the degree of autonomy is partly reduced and arming is done independently of the navigation algorithm.

The next step after arming is to issue the take off command by publishing a waypoint which is directly above (Z axis coordinate) the current position of the drone. A special PX4 take off command exists, but its use causes the drone to change the flight mode from autonomous to manual. The core of the algorithm is performed during the *START_MISSION* state, in which the drone flies through the designated waypoints. Finally, after reaching its final goal position, the land command is published.

4.3.2 Important PX4 parameters

As mentioned in Section 2.2, there are a number of essential parameters that dictate which sensors the FMU uses and what kind of failsafe mechanisms are activated in case of hardware or software malfunctions. It is not possible to compile an exhaustive list of parameters here, since there are hundreds of them, but essential ones should be part of the

Algorithm 2: Offboard control

```
Input:
    Arming command
    3D waypoint list in the form of local (x,y,z) coordinates
Output:
    Periodic ROS 2 messages containing intermediary local coordinates
while $True$ do
    switch STATE do
        case ARMED do
            enable_offboard_mode()
        case TAKE_OFF do
            publish_takeoff_command()
        case START_MISSION do
            for $waypoint = 1, N$ do
                compute_path($currentWaypoint, nextWaypoint, speedFactor$)
        case END_MISSION do
            publish_land_command()
    publish_trajectory_waypoint($x, y, z$)
```

take off checklist.

While many PX4 parameters are self-explanatory, there are quite a few which require additional clarification. SYS_MC_EST_GROUP determines if the extended Kalman filter (one of the most important feature of PX4 Autopilot) will be used in computing the multitude of sensor data (IMU, GNSS sensor, magnetometer, camera, LIDAR, etc.). This data is used for calculating the position, velocity and orientation of the drone in real-time. EKF2_AID_MASK is used to select between the GNSS sensor (usable outdoors) and the vision-based localization (available indoors with motion capture systems). EKF2_HGT_MODE determines which sensor (GNSS, barometric pressure sensor or vision-based) is used for determining the altitude. EKF2_EV_DELAY is the time between

	Value	
Parameter	**Outdoor**	**Indoor**
SYS_MC_EST_GROUP	ekf2	
EKF2_AID_MASK	1	24
EKF2_HGT_MODE	0 or 1	Vision
EKF2_EV_DELAY	-	50 ms
MAV_PROTO_VERSION	1	
RTL_DESCEND_ALT	50 m	2 m
RTL_RETURN_ALT	50 m	2 m

Table 4.2: Essential PX4 parameters

the IMU measurements and the pose estimation provided by a motion capture system. RTL_DESCEND_ALT (for fixed wing drones) and RTL_RETURN_ALT (for multicopters) determine the altitude used by the UAV when the autonomous return to take off position is triggered, either by a human pilot or a failsafe mechanism. This list is by no means exhaustive, but it represents a set of parameters which must be checked before each take off to minimize the risk of crashes and injuries.

4.3.3 Flight management architecture

In addition to the implementation of the vision-based localization algorithm, a ROS 2 framework that enables autonomous flight indoors and outdoors (with only changing a couple of PX4 specific parameters) was also created. Using the PX4-ROS 2 bridge described in Section 2.3 and a hierarchical control approach, the design described in Figure 4.5 enables the drone to fly autonomously through waypoints defined either manually by a human user, or by a path planner. The navigation node uses as inputs sensor data received from PX4 through the microRTPS interface and a list of waypoints. Its output is a continuous stream of local coordinates representing waypoints close enough so that the

UAV movement is smooth. The speed of the UAV can be easily increased or decreased (depending on the use case) by altering the $speedFactor$ in Algorithm 2.

The most important design choice of the framework is the fact that the *Navigation Node* (NN) does not directly publish coordinates and commands to the PX4 flight controller, but to an intermediary node that ensures the soundness of the data before feeding it to the FMU. In this sense, the NN directly receives sensor data from PX4 to ensure minimum latency, but its output, consisting of coordinates, is published to the *Check* topic to which the *Safety Check Node* (SFC) is subscribed. Basically, the safety node is simply relaying data (only after ensuring its correctness) between the NN and the trajectory topic that dictates the movement of the drone.

The SFC performs 4 different checks before relaying the trajectory data to the FMU:

1. The distance between two consecutive waypoints issued by the NN is lower than a specified threshold (default 0.4 meters)
2. The AGL altitude does not exceed a maximum value (default 3 meters)
3. The distance between that previous and the current location of the drone does not exceed a specified value (default 0.5 meters)
4. The distance between the take off location and the current location of the drone exceeds a maximum value (default 10 meters)

All the default values have been chosen for safe indoor flights, but can be easily changed to suit longer outdoor experiments. The chosen design of the framework obviously increases the latency by relaying command data through an additional node and performing safety checks, instead of directly publishing it to the topic to which PX4 is subscribed. This loss of performance is entirely motivated by a significant increase in the safety of drone flying experiments. In addition to the mandatory regulation briefly presented in Section 2.7, UAVs are potentially dangerous systems and the testing of ex-

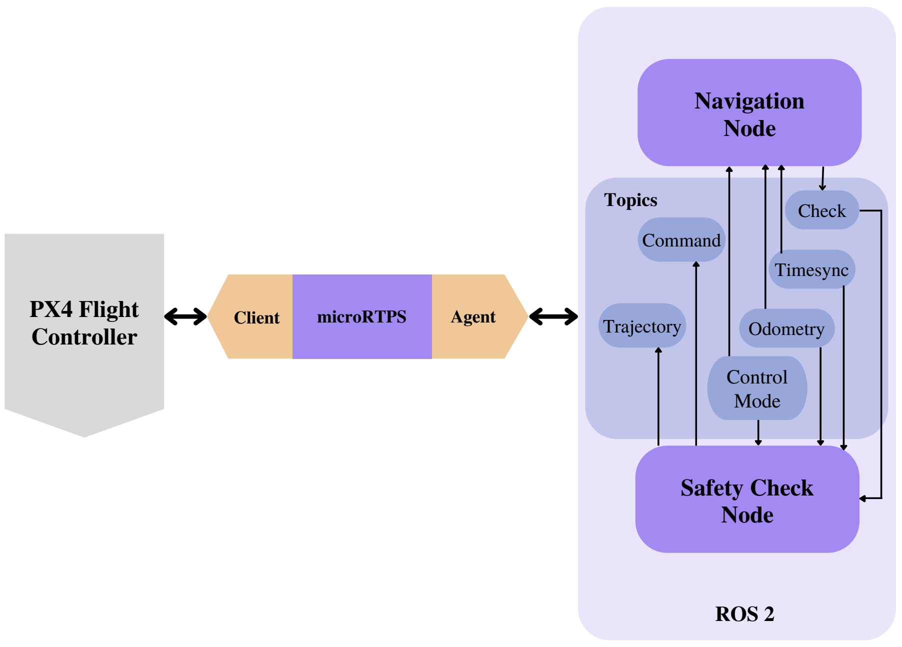


Figure 4.5: Interfacing of ROS 2 nodes and topics for offboard control

perimental software and hardware setups require hierarchical control approaches such as the one in Figure 4.5.

If any of the four safety checks fail, the SFC immediately stops publishing trajectory coordinates to the FMU, which promptly triggers a failsafe mechanism that forces the drone to land. In the case of failure, another choice is to switch the flight mode from *Offboard* (autonomous) to *Position* (manual) or *Hold* (maintain current position). Since the microRTPS bridge only works with unstable versions of the PX4 Autopilot software, the failure rate is higher than usual, so the hierarchical control mechanism plays an important role during flights.

4.4 5G Control and offloading

Considering the recent advances and the ubiquity of mobile networks, 5G technology became an enabling communication framework for UAVs. Due to its advantages such as high throughput and low latency – up to 1Gbps transfers with less than 50 ms delays

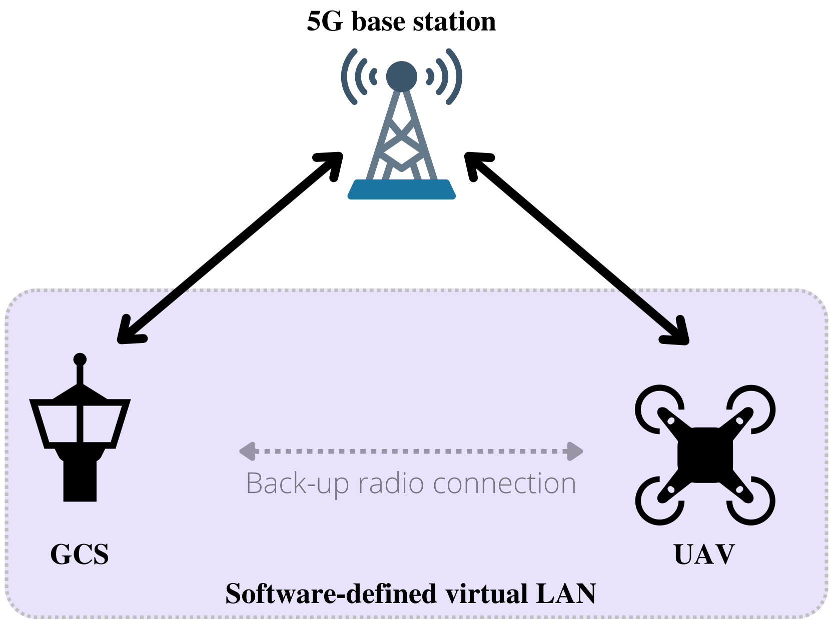


Figure 4.6: Network topology of a UAV controlled over 5G

[30] – 5G is a suitable standard for the control of UAVs. This is the main reason an experimental setup of an UAV entirely controlled through a 5G connection was designed during the data gathering sessions of this project.

Setting up telemetry over 5G is facilitated by an application called *mavproxy* [31] that can relay telemetry data to a ground control station through UDP in real-time. Additionally, both the onboard and the offboard computer are in the same software-defined virtual network to enable the exchange of data without the necessity of knowing the IP addresses of the SIM cards used for connecting to the 5G mobile network. This kind of setup extends the range of the drone to the boundaries of the mobile network coverage, because a direct radio connection between the UAV and the ground control station is no longer needed. Figure 4.6 introduces a simplified network topology that enabled a drone to be controlled over 5G.

Figure 4.7 shows a snapshot of the ground control station interface (QGC) together with video streaming while the UAV is being controlled exclusively through a 5G connec-

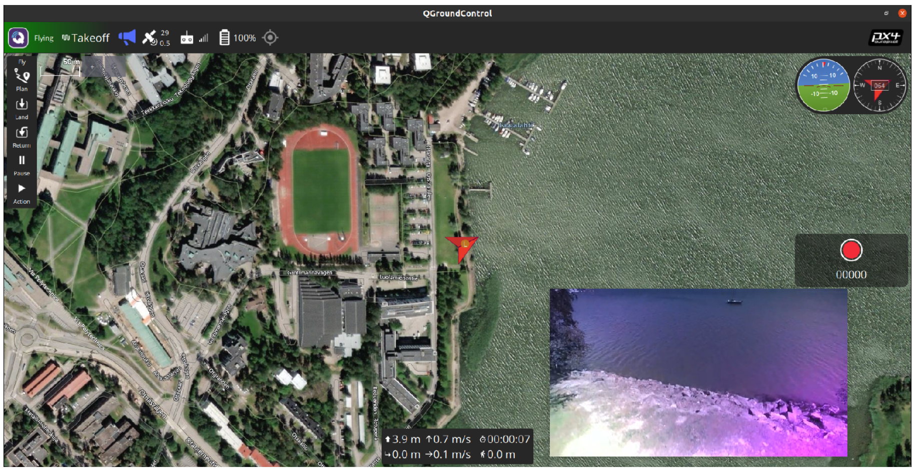


Figure 4.7: Autonomous control of UAV using 5G telemetry

tion. Even though a separate radio control connection was available due to safety reasons (as a failsafe), the control and monitoring of the drone is entirely done through the mobile network.

To demonstrate the viability of UAV telemetry over a mobile network, it was possible to connect the onboard computer to a 5G smartphone using USB tethering. In this case, the onboard computer featured 16 GB of RAM and a quad-core processor, making it feasible for very resource intensive processing (edge computing). In a different scenario, where a drone with a smaller payload would be used, it would be possible to process data captured by the drone in real-time on an *offboard* computer that has increased computing capabilities. For example, the localization algorithm presented in Section 4.2 could run in real-time on an offboard computer using photos from the drone camera and providing the UAV with GNSS-Free localization. This offloading scenario could be particularly useful in multi-robot systems, with a swarm of MAVs (having limited computing capabilities) connected to a 5G network where each aerial agent could be located using a GNSS-Free vision-based algorithm. In this sense, the tested 5G control of UAVs represents an important building block towards GNSS-Free localization and navigation.

Obviously, 5G connectivity is preferred due to its low latency and high bandwidth, but older technology such as 4G (or even 3G) mobile networks can be used to remotely

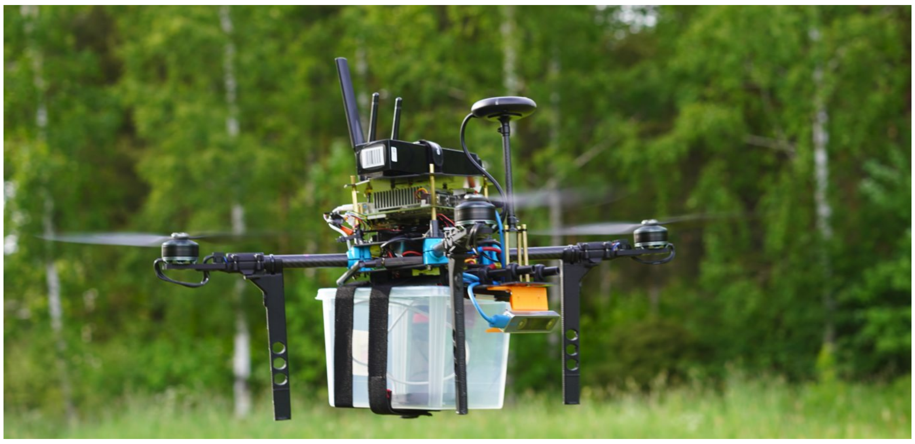


Figure 4.8: Drone connected to the 5G network through USB tethering

control a UAV. During the experiments, the Holybro 500 V2 in Figure 4.8 was also connected to the ground control station using 4G modems (one for the onboard computer and one for the laptop on which QGC was running).

5 Results

This chapter presents the experimental results for two photograph datasets on which the localization algorithm was tested. The photographs were taken on sunny weather in the wild (non-urban environment), from an altitude of approximately 120 meters AGL. Each dataset consists of tens of photographs taken from the same perspective (camera facing the ground). The ground truth is provided by the GNSS metadata embedded into each photograph.

Figure 5.1 provides a bird's-eye view of the flight area chosen for gathering the two datasets. The zone was chosen for a number of reasons, in particular because it contains a multitude of different surfaces (forests, fields, roads, sea shores, buildings) in a relatively small perimeter. In addition, it is also a sparsely populated area, which makes it suitable for UAV experiments in accord with regulation presented in Section 2.7.

5.1 Quantitative analysis

The experimental results of the two experiments are summarized in Table 5.1. A photograph was considered successfully localized if the error between the ground truth GNSS position of the drone and the location computed by the vision-based algorithm was less than 50 meters. The most important numbers are the mean absolute error (MAE) values that characterize the localization performance of the algorithm. For the first dataset, an average error of approximately 16 meters suggests the new localization method has an accuracy that can be compared to that of standard GNSS positioning methods. Ad-

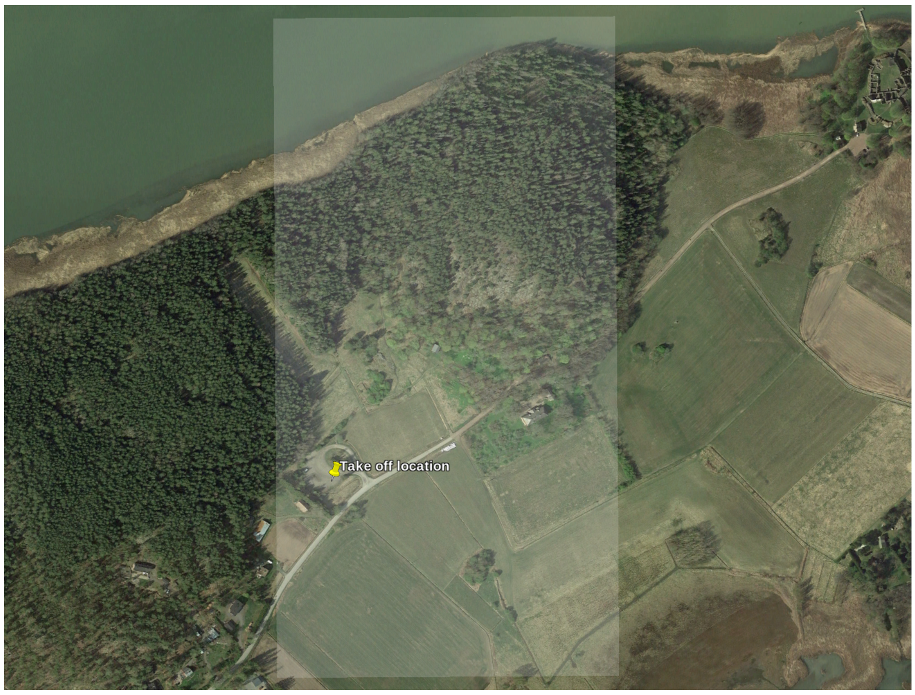


Figure 5.1: Satellite view of the flight zone (highlighted rectangle). The yellow pin is located at 60.403091° latitude and 22.461824° longitude

	Total	Localized	MAE (m)
Dataset 1	124	77 (62%)	15.82
Dataset 2	78	44 (56%)	26.58

Table 5.1: Localization algorithm performance metrics

mittedly, the accuracy is not high enough to be used in precision maneuvers or at low altitudes where obstacles are present, but the algorithm represents an important building block towards GNSS-Free navigation strategies for UAVs.

Figures 5.2 and 5.4 show the computed coordinates together with the ground truth GNSS values. For Dataset 1, the coordinates determined by the vision-based localization algorithm closely follow the GNSS ones, with a few exceptions where the error exceeds 30 meters, as can be seen in Figure 5.3. On the other hand, the values for the second dataset are less accurate (see Figure 5.5) and fewer photos had been successfully located.

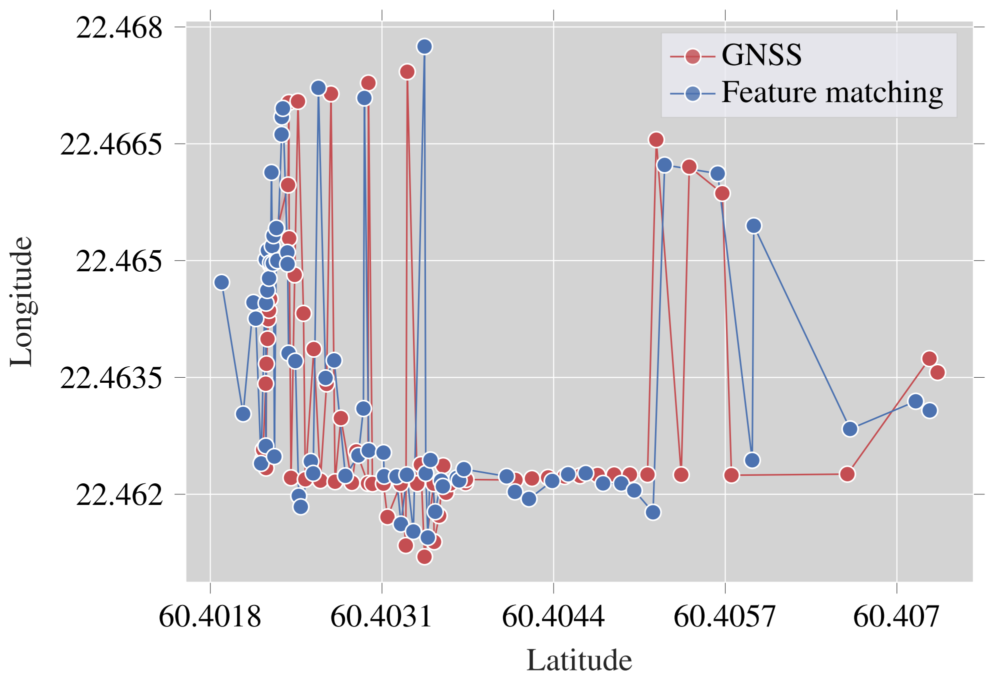


Figure 5.2: Computed coordinates and GNSS ground truth – Dataset 1

A comparison between the average error of the two datasets can be seen in Figure 5.6. The major difference in performance can be attributed to the fact that Dataset 2 contains more photographs with surfaces that have fewer features to match, such as field and forests.

As discussed in Section 3.4, the feature matching model is not rotation invariant, so it was necessary to use the metadata of the drone photographs to align them with the map images, which have the same heading (north). Because the UAV compass suffered from a systematic error, all the recorded photographs were rotated with 15°clockwise in addition to the gimbal and drone yaw. This small correction reduced the misalignment between the drone photographs and their satellite counterparts, maximizing the performance of the feature matching algorithm.

The boxplot of the error for the two datasets is of particular interest because it shows that most of the images for the first experiment are located with an error comparable to traditional GNSS services. The second dataset, however, features significantly increased errors, which suggests that navigation strategies must include mechanisms that allow the

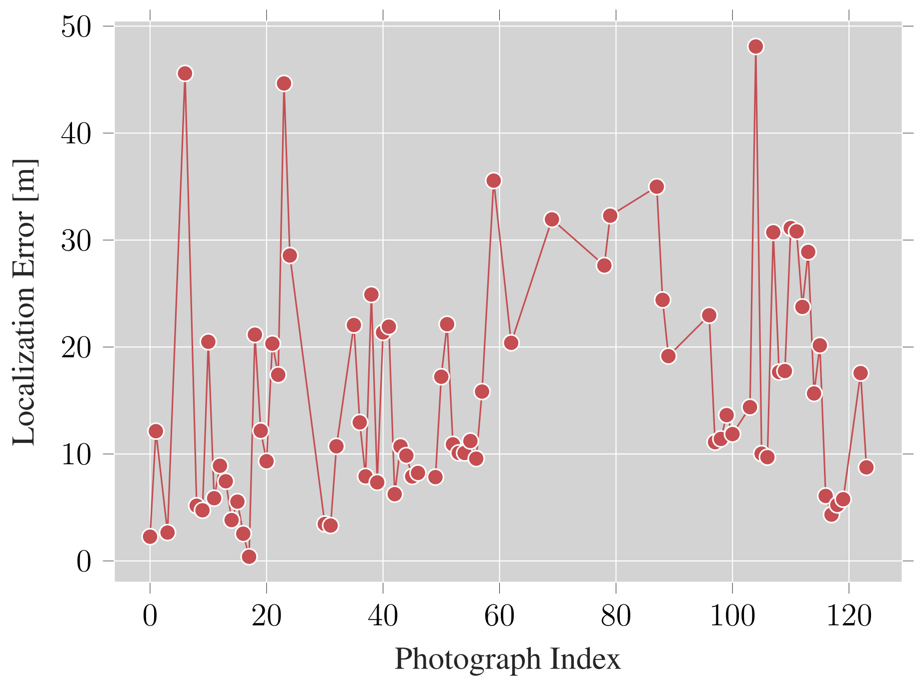


Figure 5.3: Localization error – Dataset 1

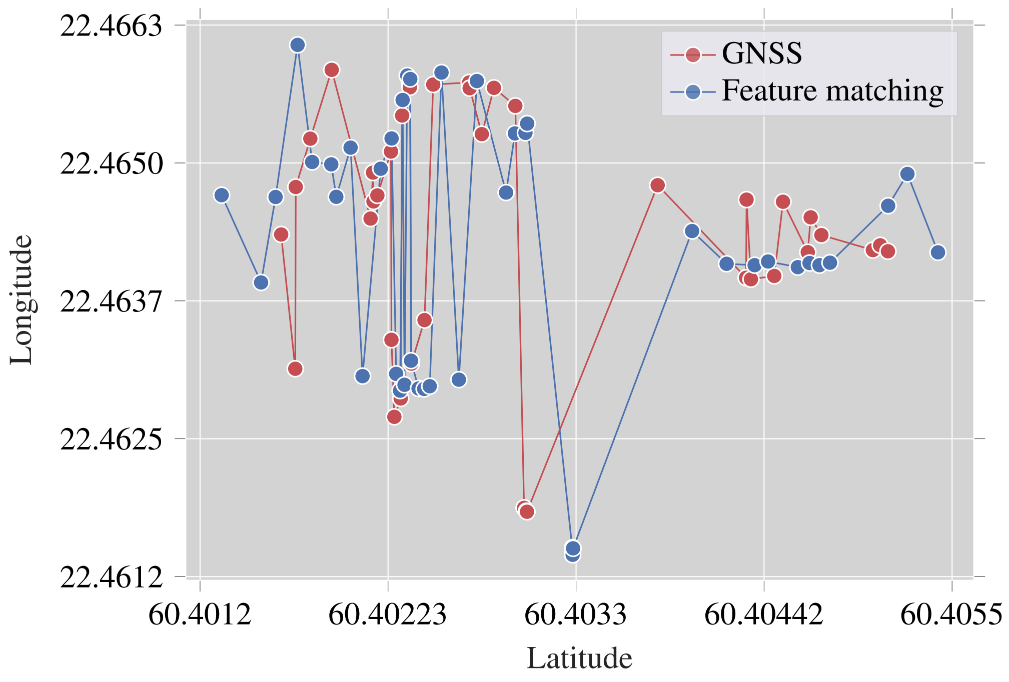
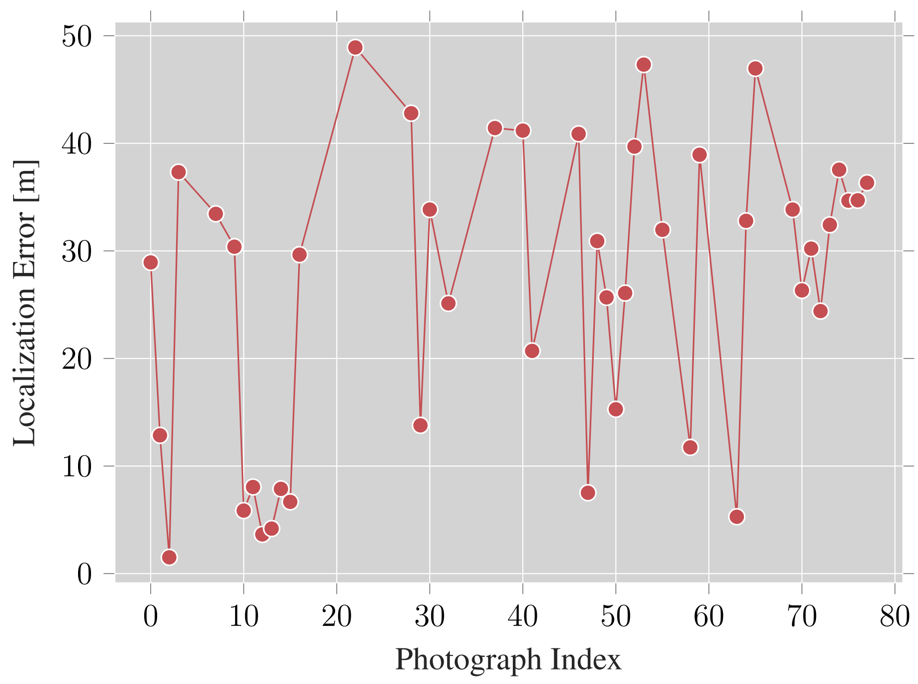


Figure 5.4: Computed coordinates and GNSS ground truth – Dataset 2



H

Figure 5.5: Localization error – Dataset 2

UAV to fly only over surfaces that offer salient features to be matched. The performance of the vision-based localization algorithm is directly dependent on the trajectory computed by the path planner.

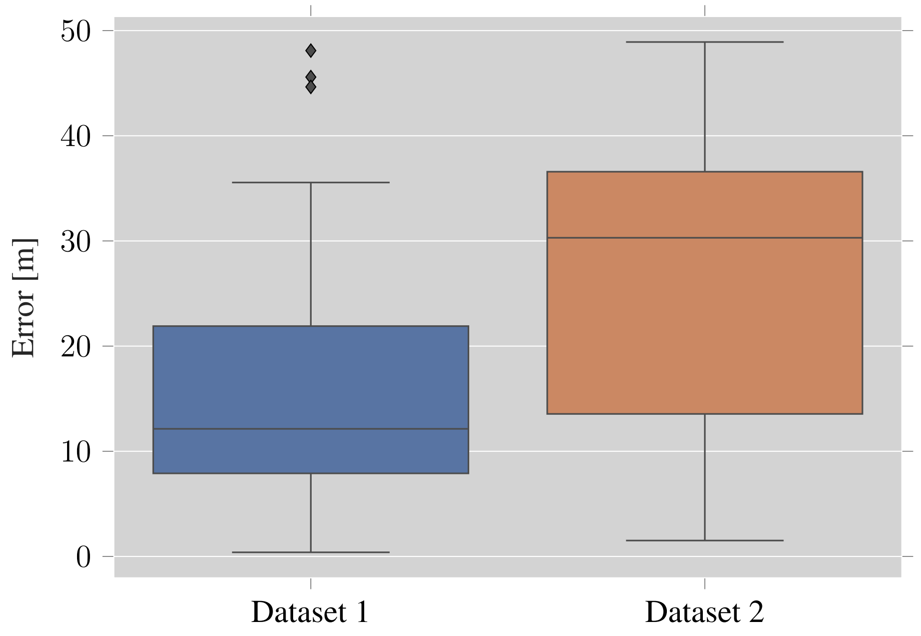


Figure 5.6: Boxplot comparing the localization error for the two experiments

5.2 Qualitative analysis

There are a number of important factors which affect the accuracy of the vision-based localization algorithm. Among these are weather and light – depending on the season and the amount of lighting, the drone photographs could have more or less features that can be matched with the onboard satellite map. The photos taken during the experiments were taken one year later, but in the same month (May) as the satellite images, in order to maximize the similarity between them.

Another important aspect is that artificial structures such as buildings and roads tend to have more features that can be tracked and matched, compared to natural objects. This is the reason that transforms feature matching into a challenging task when performed in the wild (outside urban environments). In addition to this, the open source photographs for non-urban environments are characterized by lower resolution and are more rarely updated than data available for cities. Nonetheless, Figure 5.7 shows that the model developed in this project is able to provide accurate matching results even when the drone photographs differ in perspective, rotation and contain surfaces sparse in salient features.

There are limitations that can lower performance, as the MAE value (approximately 26 meters) of Dataset 2 shows. This is explained by the fact that more photographs in the second dataset had been collected above featureless surfaces (such as fields and meadows), which can lower the accuracy or even make localization impossible.

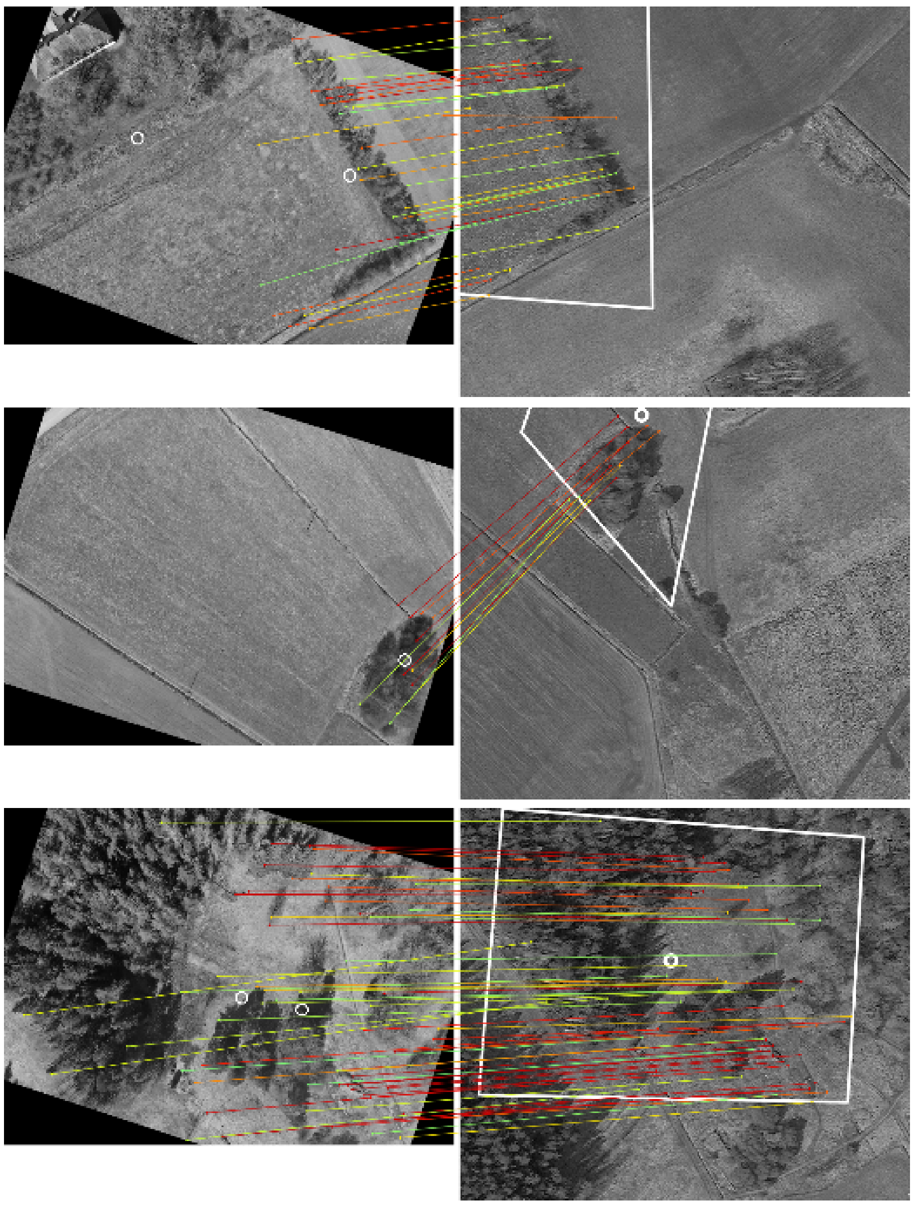


Figure 5.7: Examples of successfully matched drones photographs (left) and satellite images (right)

6 Conclusion

The goal of this project was to demonstrate that GNSS-Free vision-based localization of UAVs flying in the wild was not only possible, but a viable method which provides accurate results. To achieve this objective, three localization methods were studied and one of them was considered practical for a final implementation. To prove its effectiveness, the vision-based localization algorithm was tested on two experimental datasets consisting of RGB photographs taken with a drone flying at high altitude in non-urban environments.

6.1 Future research

The most important feature to be researched in the future is LIDAR-based localization using the gathered LIDAR dataset and the image segmentation model. This approach would enable localization of the UAV even during night or harsh weather conditions, significantly extending the applicability of the studied GNSS-Free localization algorithm.

Another improvement would be the implementation of a path planner that dictates the trajectory of the UAV, given an initial position and an end goal. The path planner would make use of the developed vision-based localization algorithm and also devise a navigation strategy to avoid flying the drone above featureless areas, where pose estimation using visual techniques becomes challenging or impossible.

An incremental improvement of the current implementation, given enough resources, could be done by retraining both the image segmentation model and the graph neural network used in feature matching using RGB photographs taken with the drone camera, instead of satellite images. Although time-consuming, the retraining process would significantly improve the accuracy of the localization algorithm.

References

[1] J. Tomastik, M. Mokroš, P. Surovy, A. Grznárová, and J. Merganič, "Uav rtk/ppk method—an optimal solution for mapping inaccessible forested areas?", *Remote sensing*, vol. 11, no. 6, p. 721, 2019.

[2] J. P. Queralta, C. M. Almansa, F. Schiano, D. Floreano, and T. Westerlund, "Uwb-based system for uav localization in gnss-denied environments: Characterization and dataset", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 4521–4528.

[3] J. Tordesillas and J. P. How, "Panther: Perception-aware trajectory planner in dynamic environments", *arXiv preprint arXiv:2103.06372*, 2021.

[4] M. L. Psiaki and T. E. Humphreys, "Gnss spoofing and detection", *Proceedings of the IEEE*, vol. 104, no. 6, pp. 1258–1270, 2016. DOI: 10.1109/JPROC.2016.2526658.

[5] J. Shermeyer, D. Hogan, J. Brown, *et al.*, "Spacenet 6: Multi-sensor all weather mapping dataset", in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 196–197.

[6] L. Bartolomei, L. Teixeira, and M. Chli, "Perception-aware path planning for uavs using semantic segmentation", in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 5808–5815.

[7] B. Patel, T. D. Barfoot, and A. P. Schoellig, “Visual localization with google earth images for robust global pose estimation of uavs”, in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2020, pp. 6491–6497.

[8] A. Shetty and G. X. Gao, “Uav pose estimation using cross-view geolocalization with satellite imagery”, in *2019 International Conference on Robotics and Automation (ICRA)*, IEEE, 2019, pp. 1827–1833.

[9] E. Guérin, K. Oechslin, C. Wolf, and B. Martinez, “Satellite image semantic segmentation”, *arXiv preprint arXiv:2110.05812*, 2021.

[10] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, “Superglue: Learning feature matching with graph neural networks”, in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4938–4947.

[11] A. Yol, B. Delabarre, A. Dame, J.-E. Dartois, and E. Marchand, “Vision-based absolute localization for unmanned aerial vehicles”, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2014, pp. 3429–3434.

[12] O. Robotics. “Understanding ros 2 nodes”. (2022), [Online]. Available: `https://docs.ros.org/en/dashing/Tutorials/Understanding-ROS2-Nodes.html` (visited on 06/27/2022).

[13] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild”, *Science Robotics*, vol. 7, no. 66, eabm6074, 2022.

[14] Y. Maruyama, S. Kato, and T. Azumi, “Exploring the performance of ros2”, in *Proceedings of the 13th International Conference on Embedded Software*, 2016, pp. 1–10.

[15] P. D. Team and Community. “Px4 autopilot user guide”. (2022), [Online]. Available: `https://docs.px4.io/master/en/` (visited on 06/28/2022).

[16] ——, “Px4 autopilot user guide”. (2022), [Online]. Available: https://docs.px4.io/master/en/advanced_features/precland.html (visited on 06/30/2022).

[17] ——, “Px4 autopilot user guide”. (2022), [Online]. Available: https://docs.px4.io/v1.12/en/ros/ros2_comm.html (visited on 06/30/2022).

[18] ——, “Px4 simulation modes”. (2022), [Online]. Available: https://docs.px4.io/v1.12/en/simulation/ (visited on 07/13/2022).

[19] S. P. Mohanty, J. Czakon, K. A. Kaczmarek, *et al.*, “Deep learning for understanding satellite imagery: An experimental survey”, *Frontiers in Artificial Intelligence*, vol. 3, p. 534 696, 2020.

[20] Z. Liu, Y. Lin, Y. Cao, *et al.*, “Swin transformer: Hierarchical vision transformer using shifted windows”, *arXiv preprint arXiv:2103.14030*, 2021.

[21] K. Briechle and U. D. Hanebeck, “Template matching using fast normalized cross correlation”, in *Optical Pattern Recognition XII*, SPIE, vol. 4387, 2001, pp. 95–102.

[22] C. Stachniss, “Image template matchingusing cross correlation”, *Photogrammetry I& Robotics Lab - Lecture Slides, University of Bonn*, 2021.

[23] D. G. Lowe, “Object recognition from local scale-invariant features”, in *Proceedings of the seventh IEEE international conference on computer vision*, Ieee, vol. 2, 1999, pp. 1150–1157.

[24] ——, “Distinctive image features from scale-invariant keypoints”, *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[25] D. DeTone, T. Malisiewicz, and A. Rabinovich, “Superpoint: Self-supervised interest point detection and description”, in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2018, pp. 224–236.

[26] Traficom. “Uas/rpas regulation”. (2022), [Online]. Available: https://droneinfo.fi/en/self-study-material-exams-accordance-eu-regulation (visited on 07/13/2022).

[27] Y. Wang, J. Ji, Q. Wang, C. Xu, and F. Gao, “Autonomous flights in dynamic environments with onboard vision”, in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2021, pp. 1966–1973.

[28] O. Documentation. “Template matching”. (2022), [Online]. Available: https://docs.opencv.org/4.x/d4/dc6/tutorial_py_template_matching.html (visited on 07/06/2022).

[29] A. Carballo, J. Lambert, A. Monrroy, *et al.*, “LIBRE: The multiple 3d lidar dataset”, *arXiv preprint arXiv:2003.06129*, 2020, (accepted for presentation at IV2020).

[30] A. Narayanan, E. Ramadan, J. Carpenter, *et al.*, “A first look at commercial 5g performance on smartphones”, in *Proceedings of The Web Conference 2020*, 2020, pp. 894–905.

[31] A. D. Community. “Mavproxy documentation”. (2022), [Online]. Available: https://ardupilot.org/mavproxy/ (visited on 07/19/2022).