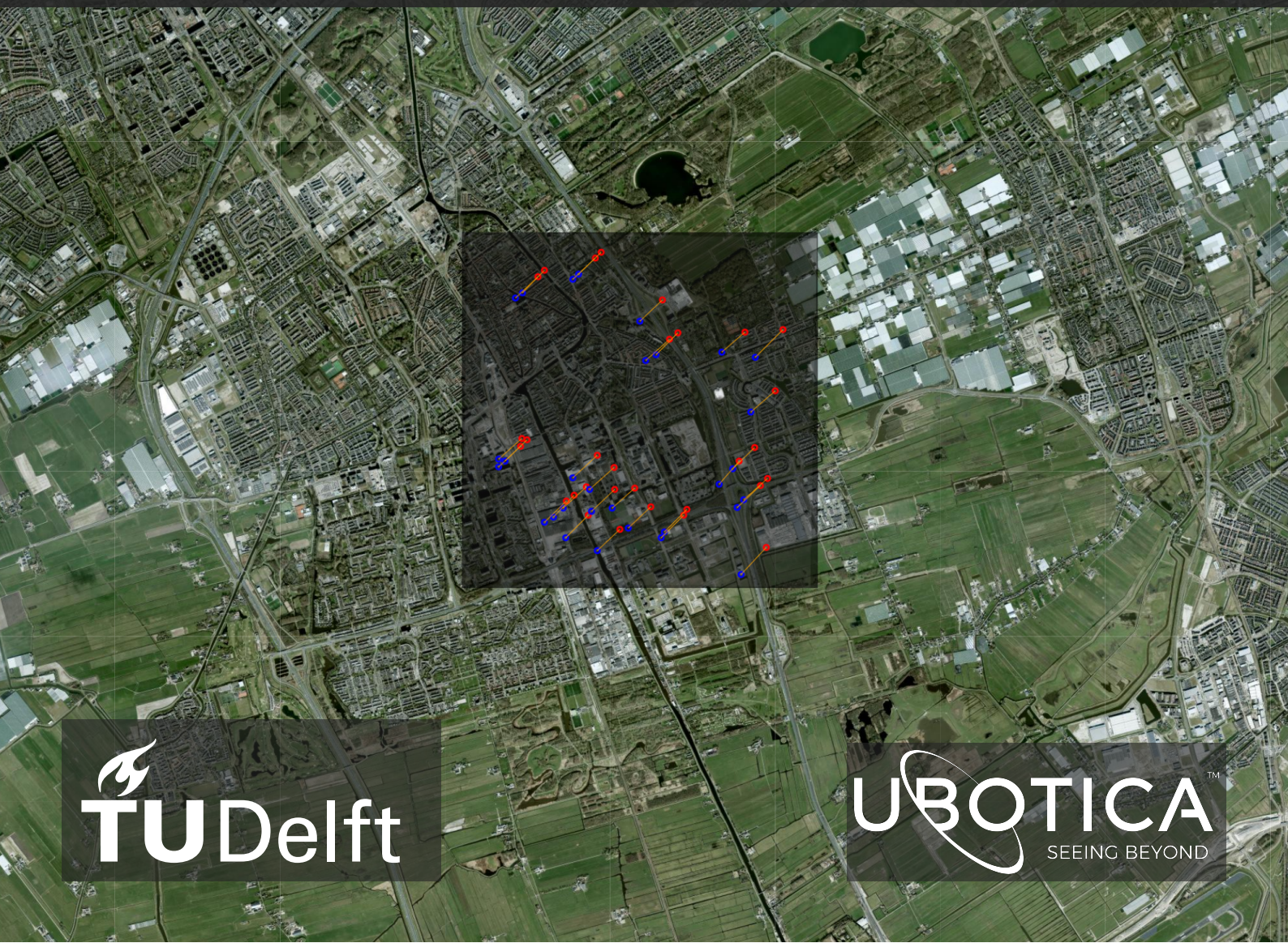
# A Novel Onboard Georeferencing System for Small Satellites

## Enabling Real-Time Localized Insights

MSc Thesis

L.H. Maiorano



TU Delft

UBOTICA™ SEEING BEYOND

# A Novel Onboard Georeferencing System for Small Satellites

## Enabling Real-Time Localized Insights

by

L.H. Maiorano

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday, November 17th 2023, at 15:00.

Student Number: 4431138
Project Duration: February 15, 2023 – November 17, 2023
Faculty: Faculty of Aerospace Engineering, Delft

Thesis committee:

| | |
|---|---|
| Prof. Dr. E.K.A. Gill, | TU Delft, Chair |
| Dr. A. Menicucci, | TU Delft, Supervisor |
| Ir. D. Rijlaarsdam, | Ubotica Technologies Ltd., Supervisor |
| Ir. K.J. Cowan MBA, | TU Delft, Examiner |

Version: v2
Date: November 14, 2023

*This thesis is confidential and cannot be made public until November 17, 2025.*

Cover: Sentinel-2 image of Delft, georeferenced using this novel pipeline, overlaid on ESRI World Imagery [15, 64]

# Preface

As a culmination of my studies, this real-world georeferencing problem was an intriguing combination of multiple aspects of aerospace engineering. A result of many hours of work and some valuable lessons learned, the completion of this thesis marks the end of an amazing study period here in Delft and the beginning of a new chapter. I would like to take this opportunity to thank all those who have helped me along the way.

First and foremost, I would like to thank my supervisors, David Rijlaarsdam and Alessandra Menicucci. You have been an incredible source of support and guidance, always asking challenging questions and providing me with insightful feedback. You presented me with a broad problem, and gave me the freedom to explore and develop my own solutions. I was extremely fortunate to work with you both, for which I'm very grateful.

To all my Ubotica colleagues, thank you for the opportunity to join such an incredible and talented team. Working alongside you has taught me so much more than just my thesis. To everyone in the Delft office, thank you for the good times and memories. Our team activities, lunches together, and humor in the office have made every day a pleasure. While there are too many to mention by name, I would like to emphasize a few who were always there to think along and share their expertise. In no particular order, thank you: Alberto, Maria, Jose, Cristopher, Tom, and Thijs.

I would also like to thank my family and friends for their support throughout my studies. To my mother, Heleen, who, since I was in elementary school, has helped me find my missing commas: thank you for all the years of reading my work and teaching me how to write.

Last but not least, I would like to especially thank my girlfriend, Esmée, for her love and unwavering support. You have stood by my side every day, every step of the way. I could not have done this without you.

*Luigi Maiorano*
*Delft, November 2023*

# Executive Summary

This research is focused on designing a system architecture for small satellites, independent of a specific mission, capable of significantly improving the timeliness of the insights generated from captured images. Building on recent technological developments which enable images to be analyzed on board the satellite to detect points of interest or to extract meaningful information, and the advent of on-demand communication networks; the next step is being able to attribute the precise geographic locations to these image insights.

Basic trigonometric methods that estimate the location are standard with an image capture, but dedicated georeferencing is still required to remove distortions and accurately determine the transformation between the image and spatial reference frames. The ability to receive both a description and location of insights from space, within minutes of image capture, will revolutionize the way time-critical monitoring applications are performed. This research aims to develop a system architecture that enables this, and to demonstrate its feasibility using a proof-of-concept implementation.

Starting with a detailed background study, the fundamental concepts of georeferencing are studied. This ensures a necessary understanding of the different types of image resolutions, common terminology, and accuracy metrics, before detailing the differences between direct and indirect georeferencing. Where direct georeferencing relies on sequential reference frame transformations and position/attitude measurements made at the time of capture, indirect georeferencing uses a database of reference images to compare the captured image to; both with the goal of determining a precise transformation between the image and the spatial reference frames. While image-based georeferencing systems are widely available, none are designed for edge computing on board small satellites in space.

In order to bridge this knowledge gap, a series of potential use cases are identified. These are time-critical applications such as wildfire and earthquake detection, which would both directly benefit from a system that can locate areas of interest identified in images and transmit this information within minutes of capture. The technical challenges and limitations imposed by small satellites and current-generation Inter-Satellite-Link (ISL) networks are then identified: mostly related to the minimal computational resources available on board and the extremely limited communication bandwidth. From these the system requirements are derived, which are then used to drive the design process.

The first design phase generates options regarding the overall approach to be used. This identifies that the georeferencing can be executed entirely on board the satellite, on terrestrial resources, or a combination of the two. Additionally, the type of georeferencing is inspected, and a resulting matrix is constructed between the two aspects. This matrix produces a total of nine high-level options.

An in-depth analysis and trade-off is performed, which evaluates each option based on their concept maturity, global coverage, and on-demand capability. As a result, the general approach is determined to be one where a combination of direct and indirect (image-based) georeferencing is used, and the computational processes are distributed across onboard and terrestrial resources.

The detailed design phase then determines the exact architecture to be used by this distributed system. Starting with a functional flow diagram, the main processes of the indirect georeferencing pipeline are drawn based on their operational sequence. All possible combinations of distributing these processes between onboard and terrestrial resources are then generated, within logical reason, to form eight detailed design options. Some are directly eliminated due to their obvious inferiority, and the remaining options are evaluated based on their technical feasibility. An additional subset of options is eliminated due to unnecessary complexity, and the final three are ranked based on estimated resource usage, independence of mission-specific constraints, and the potential for future development. While all three are considered acceptable solutions, the highest ranked option is selected as the final architecture.

The architecture is based on the idea that the reference keypoints with spatial metadata can be pre-extracted using terrestrial resources, and uploaded to the satellite prior to image capture operations. Once an image is captured, onboard keypoint extraction is performed using state-of-the-art deep learning models. The resulting keypoints can then be matched to the reference keypoints, and are used to compute polynomial coefficients of a transformation function between the image and spatial reference frames. Fundamental to this architecture design is its reliance on multiple independent state-of-the-art technologies, such as the keypoint extraction

model, feature selection filters, or the reference database; which are combined in a modular fashion. As research in the respective fields progresses, this novel georeferencing system can adopt these improvements and continue to perform at the highest possible accuracy.

In order to demonstrate the potential of the chosen architecture and verify the system requirements, a proof-of-concept analytical model is developed. A detailed study on potential keypoint models is conducted, and the D2-Net is selected as the most suitable for this research due to its state-of-the-art performance and low computational requirements. The analytical model is then experimentally evaluated using a variety of Sentinel-2 Level 2A datasets. The spatial accuracy of the implementation is analyzed by changing the model's parameters such as the image size, band combinations, and applying RANSAC, a well-known outlier removal algorithm.

Idealized images, radiometrically corrected to be equivalent of bottom-of-atmosphere values, are used to ensure the systematic development of a baseline georeferencing system not plagued by problematic input data such as cloud coverage, atmospheric effects, and geometric distortions. The results of the evaluations confirm the feasibility of the architecture, and the spatial accuracy achieved is comparable to current state-of-the-art georeferencing systems. However, some limitations regarding offset tiles are also identified and are investigated further. As noted in related work, the D2-Net suffers from localization errors. This is suspected to be an artifact of its original training, which was targeted to teach the model be invariant to illumination changes on cityscape images. The problem is expected to be resolved in future iterations by retraining D2-Net on representative multispectral satellite imagery.

Finally, a detailed list of recommendations for future research has been proposed regarding the novel architecture in general. These include the use of more advanced keypoint extraction models designed specifically for satellite imagery, alternative methods to address the issue of offset tiles (in the event retraining D2-Net does not prove to be a perfect solution), and detailed steps on how to improve the system to operate on non-idealized data in order to confirm the remaining system requirements.

As a result of this research, an object or point of interest detected by using state-of-the-art computer vision algorithms on board the satellite, can now be reduced to a simple label with the geospatial coordinates. Contrary to current data pipelines relying on full images, these concise messages have the potential to be transmitted via low-bandwidth inter-satellite communication networks, enabling the insights to be received on the ground within minutes of image capture.

# Contents

# List of Figures

# List of Tables

# Acronyms

| | | |
|---|---|---|
| 1B | Single-Band | 50, 53 |
| 3B-Comb | Three-Bands Combined | 50, 53 |
| 3B-Sep | Three-Bands Separated | 50, 53, 54 |
| ADCS | Attitude Determination and Control System | 27 |
| AHP | Analytical Hierarchy Process | 33 |
| AI | Artificial Intelligence | 14 |
| AMCT | Adaptive Mixed Context Triplet | 45 |
| AOCS | Attitude and Orbit Control System | 22 |
| ASIC | Application-Specific Integrated Circuit | 42 |
| ASPRS | American Society for Photogrammetry and Remote Sensing | 8 |
| BoA | Bottom of Atmosphere | 26, 27 |
| BRIEF | Binary Robust Independent Elementary Features | 15, 42 |
| CBIR | Content Based Image Retrieval | 16, 17, 28 |
| CE90 | Circular Error at 90% confidence level | 7, 8, 18, 48, 55 |
| CLI | Command Line Interface | 75 |
| CML | Concept Maturity Level | 34 |
| CNN | Convolutional Neural Network | 16, 43 |
| COTS | Commercial-of-the-Shelf | 24 |
| CV | Computer Vision | 14, 16, 17, 31 |
| D2D | Describe-to-Detect | 17 |
| DCN | Deformable Convolutional Network | 43 |
| DG | Direct Georeferencing | 5, 8, 11–13, 19, 29–32, 35 |
| DL | Deep Learning | 14 |
| DRRD | Detected, Repeatable, Reasonable, and Distinguishable | 43, 45, 63 |
| DSM | Digital Surface Model | 11, 13, 31, 32 |
| DTM | Digital Terrain Model | 11 |
| EO | Exterior Orientation | 8–13, 18 |
| ESA | European Space Agency | 34 |
| FAST | Features from Accelerated Segment Test | 15 |
| FLANN | Fast Library for Approximate Nearest Neighbors | 46 |
| FP16 | half-precision Floating-Point | 61, 64 |
| FP32 | single-precision Floating-Point | 61, 64 |
| FPGA | Field Programmable Gate Array | 42, 43 |

| | | |
|---|---|---|
| FREAK | Fast REtinA Keypoint | 15 |
| GCP | Ground Control Point | 5, 6, 13, 14, 17–20, 28, 31, 32, 35, 36, 38, 42, 44, 47, 49, 53, 54, 59, 60, 63 |
| GDAL | Geospatial Data Abstraction Library | 51 |
| GPS | Global Positioning System | 8, 9, 13 |
| GRD | Ground Resolve Distance | 6 |
| GSD | Ground Sample Distance | 6, 47 |
| HSRI | High Resolution Satellite Imagery | 6, 12 |
| ICP | Independent Check Point | 6, 18, 19, 48–50, 54, 60, 61 |
| IFOV | Instantaneous Field Of View | 6 |
| IG | Indirect Georeferencing | 5, 13, 14, 18–20, 28, 30–32, 35, 36, 41 |
| IMU | Inertial Measurement Unit | 8, 13 |
| INS | Inertial Navigation System | 8, 9 |
| IO | Interior Orientation | 8–11, 13, 18 |
| IoT | Internet of Things | 29, 31, 33 |
| IRS | Image Reference System | 44, 48–50, 59 |
| ISL | Inter-Satellite-Link | v, 1, 2, 23, 25, 36, 38, 41, 61, 67 |
| JPL | Jet Propulsion Laboratory | 34 |
| K-D Tree | K-Dimensional Tree | 17 |
| LE90 | Linear Error at 90% confidence level | 8 |
| LMS | Least Mean Squares | 20 |
| MCT | Mixed-Context Triplet | 45 |
| ML | Machine Learning | 31 |
| NCS | Neural Compute Stick | 61 |
| NMAS | National Map Accuracy Standard | 8 |
| NN | Neural Network | 32 |
| NTNU | Norwegian University of Science and Technology | 43 |
| OBC | OnBoard Computer | 25, 31, 61 |
| ORB | Orientated FAST and Rotated BRIEF | 14, 15, 44, 46 |
| OTS | Off-the-Shelf | 52, 55 |
| PCA | Principal Component Analysis | 16, 28 |
| POI | Point of Interest | 29–32, 44 |
| PRISMA | the Hyperspectral PRecursor of the Application Mission, Italian Space Agency | 55 |
| QI | Query Image | 5, 17, 28, 36–38, 41, 42, 44–60, 65, 66 |
| RAM | Random-Access Memory | 51, 55, 61 |
| RANSAC | RANdom SAmple Consensus | 18, 20, 43, 47, 49–55, 61, 62, 67 |
| REGIS | REal-time Georeferencing in Space | 75, 77 |
| RF | Reference Frame | 10, 44 |

| | | |
|---|---|---|
| RFDB | Reference Feature Database | 44, 46–55, 57–61, 63–66, 75 |
| RFM | Rational Functional Model | 11, 18–20, 36, 38 |
| RI | Reference Image | 17, 36, 41, 42 |
| RMSE | Root Mean Square Error | 6–8, 12, 18, 19, 48, 49, 52–54, 60, 61 |
| RPC | Rational Polynomial Coefficient | 11, 18, 30, 32 |
| RPF | Rational Polynomial Function | 12 |
| SIFT | Scale Invariant Feature Transform | 15, 44 |
| SNN | Siamese Neural Network | 32 |
| SotA | State-of-the-Art | 25, 26, 31, 42, 45, 54, 55, 64 |
| SRS | Spatial Reference System | 44, 48–50, 53, 59, 61 |
| SSC | Suppression via Square Covering | 65, 66 |
| SURF | Speeded-Up Robust Feature | 15, 42, 44 |
| TRL | Technology Readiness Level | 34 |
| USM | Universal Sensor Model | 31 |
| VPU | Vision Processing Unit | 25, 45, 61 |

# 1 Introduction

In the past decades, the space industry has seen a rapid increase in the number of small satellites launched into orbit. Continuous developments in the fields of electronics and computer science have enabled the miniaturization of satellite components, and has led to a new generation of small satellites which are significantly smaller and cheaper than their predecessors. The commercialization of this industry has furthermore led to an explosion of data-driven services, which rely on the vast quantities of data generated by these satellites.

One of the most common applications of small satellites is Earth observation. This is the process of collecting data about the Earth's surface, atmosphere, and oceans. The data is collected using a variety of sensors, such as cameras, spectrometers, and radar; and is then used to monitor and predict changes in the environment, such as climate change, natural disasters, and human activity. Key to the success of these applications is the ability to accurately locate the data. This is known as georeferencing, and is the process of assigning geographic coordinates to the collected data.

## 1.1. Motivation

In the field of remote sensing and data science, a recent trend has been towards processing data on the edge. Contrary to traditional pipelines where the data captured by a spacecraft is stored on board until it can be transmitted to a ground station for processing, this approach aims to filter and preprocess the data directly at the source in order to prioritize only the most relevant information. This is especially important for low-powered small satellites, which have limited downlink bandwidth and are therefore not always able to transmit all the data they capture.

One such example is cloud detection or object detection, in which the captured images are analyzed to filter out clouds that obscure the ground, or to identify objects of interest. While this is computationally more expensive than just storing the images on board, it allows a satellite to identify the subset of images with potential value, and discard the rest. The satellite can thus continue to capture images, retaining only those that are of value; thus increasing the return and total value of the mission.

Regarding satellite imagery, the value of the insights they produce is related directly to its positional accuracy. Being able to identify an object of interest in an image is only useful if the spatial location of the image is known. Take for example a satellite tasked with monitoring wildfires. The appearance of a smoke plume or thermal hotspot in an image indicates the presence of a fire, and can be detected on board the satellite using state-of-the-art computer vision models. However, these models will return the location of the fire within the image, and not the location of the fire on the ground. This is where georeferencing comes into play: by determining the location of the image, the geoposition of the fire can also be determined. This information can then be used to alert emergency services, and to track the fire's progress over time.

A second aspect of this use case is the timeliness of the information. Current data processing pipelines involve waiting for the satellite to establish direct contact with a ground station, in order to downlink the images stored on board so that they can be georeferenced. Depending on the orbit and distribution of ground stations across the globe, this can take an hour or more before the satellite can establish contact.

With the advent of cheaper satellites and ability to construct low-cost constellations, the emergence of real-time data services has become an affordable communication option. They establish a network of Inter-Satellite-Links (ISLs) to enable on-demand communication with the client satellite, regardless of its position in orbit. However,

the data rates of these services are multiple orders of magnitude lower than direct downlinks.

Combining the ability to preprocess data on board the satellite and the advances in ISL technologies, the goal of this research is to develop a novel georeferencing system that enables insights extracted on the edge to be attributed geospatial information and be delivered to the end user in real-time. This has significant potential for a multitude of time-critical applications such as the detection of natural disasters, deforestation or illegal fishing, to name only a few.

## 1.2. Research Question

An extensive literature study and background research determined that onboard real-time georeferencing systems do not yet exist. This is due to the significant technical challenges of performing what is considered a computationally expensive task on a small satellite with very limited resources. Only with recent developments in specialized hardware to accelerate computer vision tasks has this become a realistic possibility.

Due to the novelty of this technology, a high-level systems' engineering approach was followed that considered all aspects of the data pipeline: from image capture to delivery of the final georeferenced result. Additionally, a focus was placed on developing a system architecture, in order to ensure that the final system design would be compatible with the current state of the art, and would be able to adapt to future developments in the field.

The research question of this thesis was formulated as following:

> How can an image georeferencing pipeline for small satellites be distributed between onboard and terrestrial computing resources to enable real-time results?

To further steer the research and emphasize which aspects of the system needed to be addressed, two sub-questions were also formed.

1. How can the various elements of the georeferencing pipeline best be distributed between onboard and terrestrial resources, while still meeting the data downlink limitations imposed by an ISL network?
2. How does the spatial accuracy of this pipeline compare to what is achieved with current georeferencing systems?

The first sub-question steers the research to design a system limited by constraints concerning the downlink bandwidth. This limitation originates from real-world small satellite mission capabilities: inter-satellite-links currently do exist, but at very low data rates. The system design must therefore take this into serious consideration.

The second sub-question concerns the evaluation of the resulting pipeline. In order to demonstrate that the system architecture is a viable solution, a proof-of-concept implementation must be developed. This implementation is then tested and evaluated to determine its accuracy and potential for future improvements.

## 1.3. Methodology

The research has been organized into three main parts: a literature study, the system architecture design, and a proof-of-concept implementation. First, a thorough literature study was conducted to gain a fundamental understanding of the current state-of-the-art methods used for georeferencing. The results of this have been summarized in Chapter 2. Additionally, the literature study established that no complete end-to-end onboard georeferencing system compatible with real-time satellite communications appears in the public domain. This is described in the section "Related Work" of the research paper in Chapter 4.

To fill this technical gap, a top-down systems' engineering approach was followed. Chapter 3 documents this process and the key design decisions and their respective justifications. Starting with potential use cases of the system, a concept of operations was created to identify the interfaces and constraints. These were formalized into system-level requirements. A systematic design option exploration and trade-off process was then followed to narrow the possible solutions to those with the highest potential of success.

Fundamental to the selected design is its capacity to fuse multiple independent state-of-the-art technologies. As research in the respective fields progresses, the novel georeferencing system will be able to adopt these improvements. To demonstrate the feasibility and verify the requirements of the resulting architecture design, a proof-of-concept implementation was developed. This is presented in the form of a standalone research paper in Chapter 4.

This research paper details the investigation into specific underlying technologies used for the first-order implementation of the system. An analytical model was developed, which allowed the georeferencing pipeline to be simulated and evaluated. Although only tested on idealized data, the results of these experiments were promising and demonstrated the potential of the system. The results of these experiments are presented in the research paper, and the limitations of the current implementation are discussed. Furthermore, a number of recommendations for future work regarding this implementation are made, based on the experimental results.

Regarding the complete system architecture, the research paper was used as the verification of the design. Chapter 5 discusses which system requirements are met, and which need additional development. Finally, Section 5.2 details the aspects of the system design that need dedicated research and development. These items were identified throughout the design process, and recorded as areas for potential improvement.

Chapter 6 concludes the thesis by summarizing the results of the research, and answering the research question.

Appendix A contains larger versions of the diagrams found in the text, Appendix B describes the usage of the analytical simulation, and essential source code of the implementation can be found in Appendix C.

#### Scope of the Research Paper

The research paper (Chapter 4) has a slightly different focus than the rest of this report: it is primarily concerned with implementing and evaluating the proposed system architecture, rather than the system's design process. For this reason, the discussion and future work sections of Chapter 4 are limited to that specific implementation, while similar sections in Chapter 5 focus on the scope of the full system architecture.

The research paper and the overall thesis report are therefore complementary; together forming a complete overview of the research conducted.

# 2

# Background

This chapter aims to establish a basic understanding of the key topics regarding georeferencing. General concepts such as common terminology, satellite image resolution types and general georeferencing accuracies are introduced first. Then, the two main approaches to georeferencing are presented: direct and indirect georeferencing. While the former is ultimately not used in the final design, knowledge of the method is necessary to fully understand the system design process. For each approach, the basic principles are presented, followed by an assessment of the accuracy and a discussion of the potential error sources.

## 2.1. General Concepts

Basic concepts and terminology regarding remote sensing and georeferencing are presented here. These fundamentals are used throughout this thesis.

### 2.1.1. Common Terminology

The following terms are commonly used throughout this research, and are defined here for clarity.

**Georeferencing** is the process of attributing geospatial information to an image. A coordinate transformation function is computed which translates the position and orientation of each pixel in the image reference frame to the geographic reference frame. The methods to achieve this are either considered to be rigorous (using exact geometric relations, see Section 2.2) or empirical (using variable ground control points, see Section 2.3).

**Orthorectification** on the other hand, improves on georeferencing by accounting for the distortions caused by terrain or other surface objects. This greatly improves the accuracy of the georeferenced image, but requires additional inputs such as a digital terrain model and/or known Ground Control Points (GCPs).[44].

**Direct Georeferencing (DG)** is defined as the georeferencing process which relies directly on the imaging sensor's orientation and position parameters to perform a series of sequential coordinate system transformations to achieve the final mapping function. Discussed further in Section 2.2.

**Indirect Georeferencing (IG)**, in contrast to direct georeferencing, relies on matched pairs of ground control points in both the captured image and a reference image to calculate the mapping from the pixel-space to the geographic reference frame. This can be performed without knowledge of the image system parameters. This is further explained in Section 2.3.

**Keypoints** are specific points detected in an image that are considered significant, have a very precise location, and are invariant to changes in scale, rotation, and illumination. These are used to match points between similar images, and are typically detected using feature detectors. The keypoint itself consists of a vector descriptor, as well as the coordinates of its location in either the image or the geographic reference frame.

**Ground Control Points (GCPs)** are pairs of matched keypoints, and consist of a pixel coordinate pair in the image and a geographic coordinate pair in the spatial reference frame.

**Query Image (QI)** is the captured image with no geospatial information. This is the input to the georeferencing process.

### 2.1.2. Image Resolution Metrics

When analyzing the aerial or satellite imagery, the quality and accuracy of an image is defined by four different types of resolution: spectral, radiometric, spatial, and temporal.

**Spectral resolution** describes the ability for the sensor to measure finer wavelengths, and defines the width and number of spectral bands it detects. Panchromatic sensors, found on many sensor systems, have a single wideband covering multiple spectral wavelengths. Multispectral sensors refer to 3 - 10 bands, while hyperspectral sensors measure hundreds to thousands of narrow bands.

**Radiometric resolution** is defined as "... the capacity of the instrument to distinguish differences in light intensity or reflectance".[1] "The greater the bit depth (number of data bits per pixel) of the images that a sensor records, the higher its radiometric resolution."[48] Since each bit is either a 0 or 1, it records in exponents of 2; 4-bit resolution has $2^4 = 16$ values.[2] An example of different radiometric resolutions can be seen in Figure 2.1d. Typical radiometric resolutions of High Resolution Satellite Imagery (HSRI) cameras are in the order of 11 to 16 bits per pixel (WorldView-110 multispectral camera).[3]

*Bit depth*, as explained above, is the quantification of radiometric resolution. The distinction is that the bit depth directly describes a digital parameter of the image, while radiometric resolution refers to the differences in radiation magnitude recorded. However, these terms are commonly interchanged, due to their direct relationship.

**Spatial resolution** refers to the "measure of the smallest object that can be resolved by the sensor" [32]. This is equal to the ground surface area imaged by the sensor's Instantaneous Field Of View (IFOV), and is measured in linear units. This is referred to as the Ground Resolve Distance (GRD). Another form of measurement is the Ground Sample Distance (GSD), which refers to the distance between pixel centers. In ideal conditions, depending on sensor and acquisition parameters, the IFOV of the detector's cells do not overlap, in which case the GRD and GSD are equal. GSD and GRD are typically measured at nadir, as distortions occur at oblique angles. Figure 2.1c shows an example of Reykjavík, Iceland, at different spatial resolutions. Satellite imagery is commonly categorized into different (spatial) resolutions [32]:

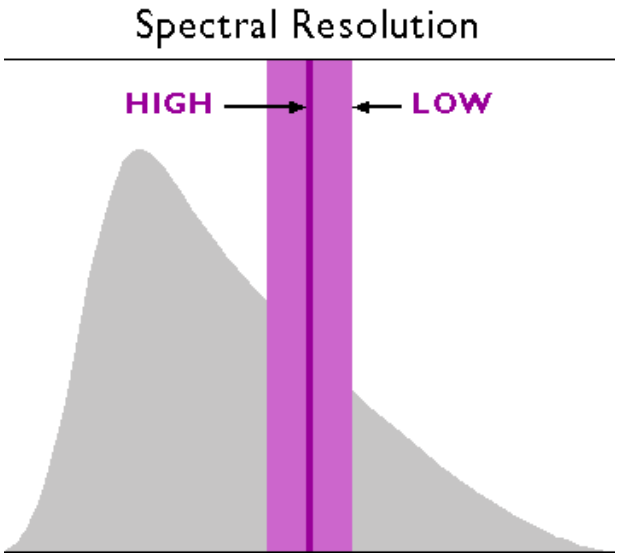
- Coarse (>1000m)
- Medium (100m - 1000m)
- Fine (5m - 100m)
- Very high (<5m)

**Temporal resolution** is the frequency or time interval that a sensor revisits the same ground segment of the Earth's surface. This is determined by the orbit of the satellite, latitude of the ground segment, and characteristics of the sensor such as swath width. Temporal resolution is important for certain applications where changes over time need to be recorded. For most remote sensing platforms this is on the order of days to weeks.

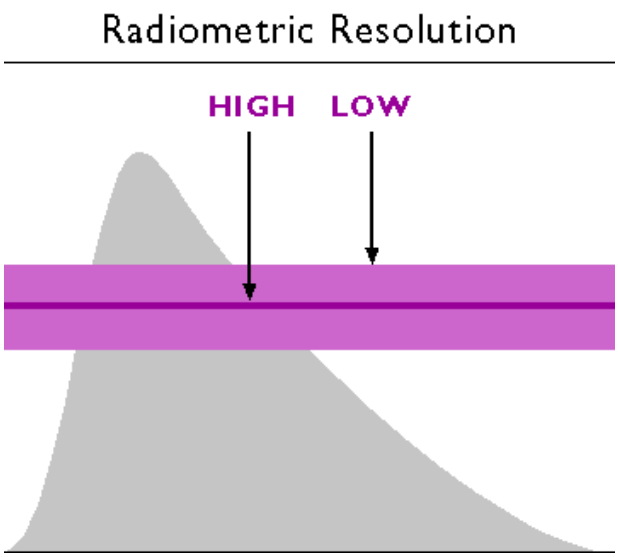
### 2.1.3. Georeferencing Accuracy

Accuracy in the scope of georeferencing is defined as "the distance between the actual geographic location of an object or detail compared to the position of the object in the image".[5] To evaluate georeferenced datasets, a subset of GCPs are selected as Independent Check Points (ICPs) and excluded from any potential georeferencing calculations. These GCPs are keypoints in the image that have a known geolocation. Once the image has been georeferenced, the predicted geospatial positions of the ICPs are compared to their known geospatial positions. The difference between the predicted and known positions is the error, and is used to calculate the accuracy of the georeferencing process. Calculating the Root Mean Square Error (RMSE) results in an overall measure of accuracy for that image [3]. RMSE can sometimes be given in two values $\text{RMSE}_x$ and $\text{RMSE}_y$ to differentiate between elliptical differences caused by oblique imagery, or simple RMSE which refers to a radial value (typically for on-nadir).

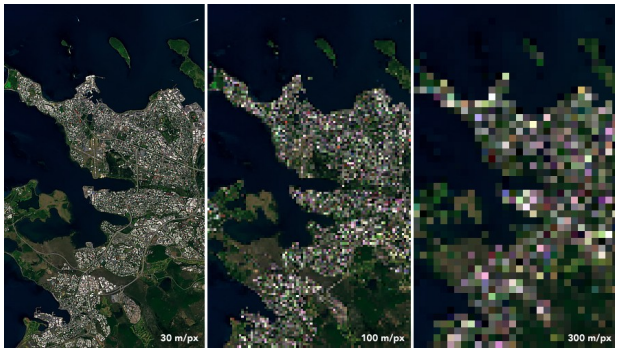
[1] https://sentinels.copernicus.eu/ca/web/sentinel/user-guides/sentinel-2-msi/resolutions/radiometric
[2] https://natural-resources.canada.ca/maps-tools-and-publications/satellite-imagery-and-air-photos/tutorial-fundamentals-remote-sensing/satellites-and-sensors/radiometric-resolution/9379
[3] https://earth.esa.int/eogateway/missions/worldview-3
[4] https://www.earthdata.nasa.gov/learn/backgrounders/remote-sensing
[5] https://www.intermap.com/blog/satellite-imagery-resolution-vs.-accuracy



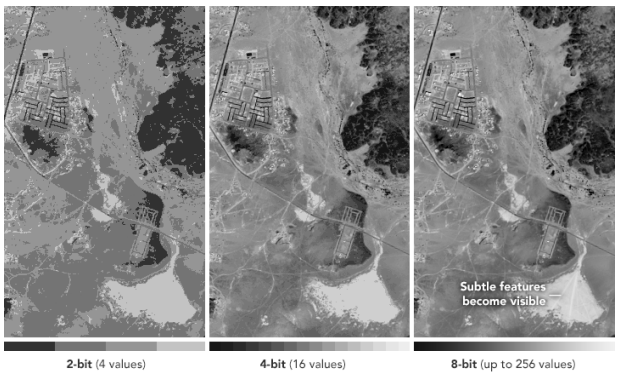

**(a)** Spectral resolution, illustrated as the width of wavelength bands captured by the sensor. High resolution sensors capture a narrow band [48].




**(b)** Radiometric resolution, illustrated as the difference in energy magnitudes that the sensor is able to discriminate. Low resolution sensors can only capture large differences in energy [48].

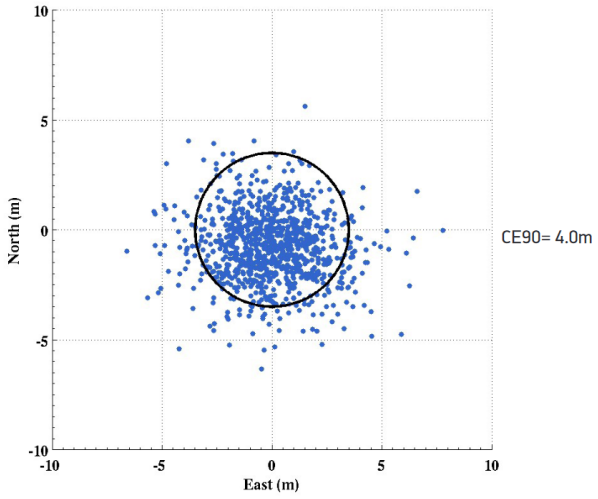



**(c)** Spatial resolution example, Landsat 8 imagery of Reykjavík, Iceland in 2019.[4]

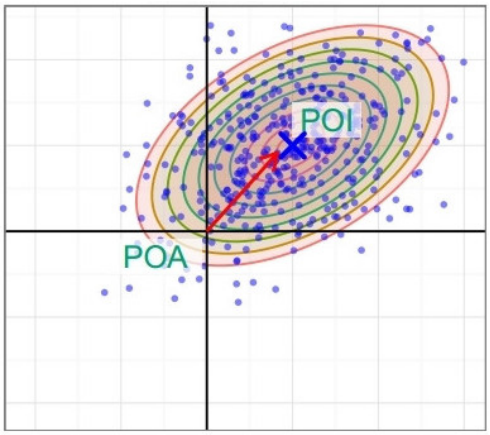



**(d)** Bit depth examples (different radiometric resolutions) [45].

**Figure 2.1:** Examples of spectral, spatial and radiometric resolutions. The gray areas in 2.1a and 2.1b represent the magnitude of electromagnetic energy at different wavelengths.




**(a)** Example of horizontal 4m Circular Error at 90% confidence level (CE90) accuracy [13].




**(b)** Example of a non-normal error distribution, demonstrating the superiority of RMSE as an accuracy metric.

**Figure 2.2:** Comparison of error distributions and metrics, CE90 vs RMSE[6]

Although RMSE is the current standard measure of accuracy defined by the American Society for Photogrammetry and Remote Sensing (ASPRS) in 2014, legacy standards are still commonly used. Known as the National Map Accuracy Standard (NMAS), it differentiates between horizontal and vertical accuracy. The CE90 refers to a sample where no more than 10% of the points (all of which should have the same geolocation) will fall outside a certain radius. Figure 2.2a illustrates this clearly. CE90 and Linear Error at 90% confidence level (LE90) are measured in meters or an equivalent linear measurement, to indicate the radius or height range in which the points fall.

However, CE90 relies on two critical assumptions: the points must have a bivariate normal distribution about a zero mean, and sufficient check points are used to accurately estimate the variances [4]. If the distribution is not normal, as seen in Figure 2.2b, the RMSE is a better measure of accuracy. For the purposes of this research, RMSE is used as the measure of accuracy since the distribution of the points cannot be assumed to be normal.

## 2.2. Direct Georeferencing

Direct Georeferencing (DG) is an approach that has been used for a relatively long time, since the early twentieth century [68]. As soon as military and civil institutions began photographing the surface via aerial surveys, a need arose to locate and identify the relation between the image and its geographic coordinates. In the period of 2000 - 2004, significant research was performed to determine if direct georeferencing could accurately be used to replace the traditional approach of that time: triangulation. It was established that off-the-shelf Global Positioning System (GPS) and Inertial Measurement Unit (IMU) could provide sufficient accuracy, especially if calibrated correctly [51, 65]. Currently, methods to perform georeferencing can be classified into two categories: rigorous geometric processing models (often referred to as direct georeferencing), and empirical geometric processing models (the indirect georeferencing variant) [33]. The purpose of this section is to describe and evaluate current state-of-the-art direct georeferencing systems, in the spaceborne remote-sensing domain. The results can then be compared to its counterpart, indirect georeferencing.

In the most simple terms, direct georeferencing transforms the image reference frame inside the camera to the geographic surface. This method is considered rigorous, because it is independent of the image taken, and does not require a best-fit approximation based on extracted image features. Present-day implementations of this approach still exist, as real-time onboard sensors can provide the necessary orientation data to perform this transformation. The intent is to determine the Exterior Orientation (EO) and Interior Orientation (IO) parameters of the camera at the moment of capture, using the onboard GPS and Inertial Navigation System (INS) to the highest possible accuracy. Through a series of coordinate transformations and projections, the image plane can then be mapped to the surface.

Direct georeferencing has two primary steps: determination of EO and IO parameters of the camera at the instant of recording, and performing scene restitution. Scene restitution transforms the pixel coordinates in the camera reference frame to the geographic coordinate frame, and corrects for residual distortions.

### 2.2.1. Orientation Parameters

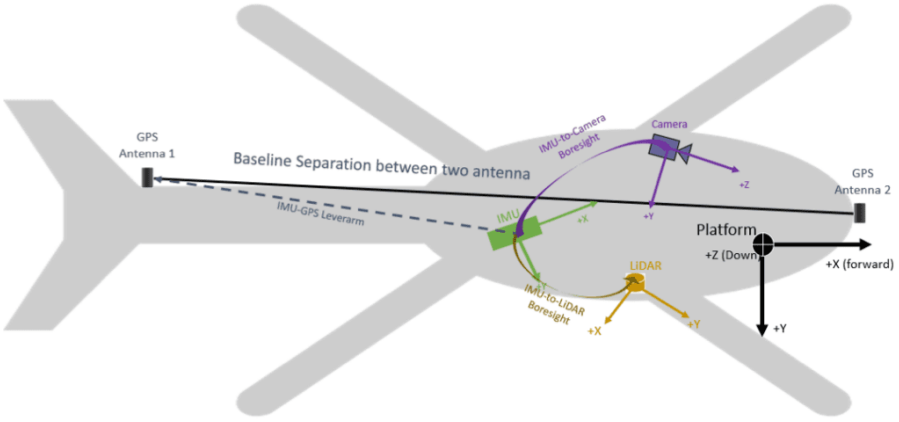



**Figure 2.3:** An example of the various reference frames onboard an aerial imaging platform. Note the depiction of boresight angles, which are a common source of systematic error.[7]

[6] http://ballistipedia.com/index.php?title=Circular_Error_Probable

The first step in direct georeferencing is determining the **Exterior Orientation (EO)** parameters of the imaging device. The EO parameters are defined as the position and orientation of the camera reference frame at the instant the image was taken, with respect to the world reference frame. For a spaceborne imaging system, this is determined using the onboard GPS and INS measurements and may utilize interpolation for an even more precise result, as analyzed in Poli [49]. These exterior orientation parameters consist of the attitude (pitch, yaw, and roll) and position (latitude, longitude, and altitude) of the spacecraft. Figure 2.3 depicts these reference frames on board an aerial platform, but is equally valid for a satellite system.

Additionally, the **Interior Orientation (IO)** parameters must be known, which are defined as the parameters relating the pixel coordinates of the image reference frame to the spacecraft reference frame [10]. This may include other intermediate reference frames such as the camera or mounting platform, and can also describe calibration parameters such as the position of EO sensors within the spacecraft, distortions caused by thermal loads, or mechanical dynamics of imaging hardware. Depending on the type of imager used, these interior orientation parameters may be fixed or variable. As explained in a 2004 study of various imaging scanners [29], whisk-broom scanners inherently have changing internal orientation parameters due to the dynamic nature of its imaging system. Figure 2.4 depicts these IO parameters as the positional vectors within the dotted box. The vectors originating from outside (i.e. $\mathbf{r}_{sensor}^{Earth}$) are the EO parameters.

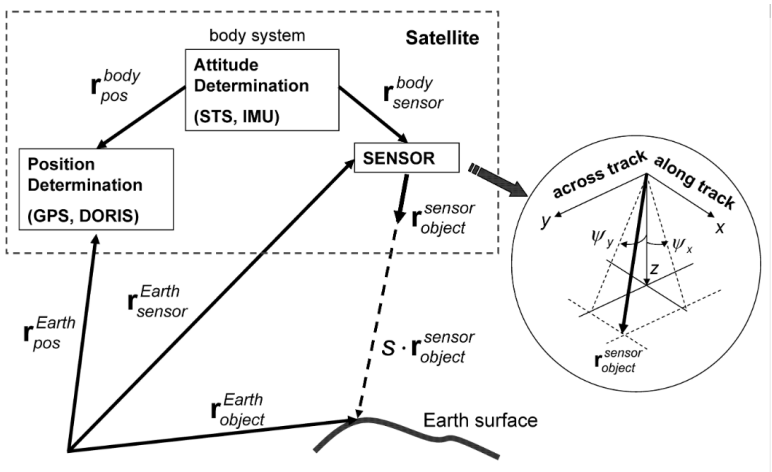



**Figure 2.4:** Vector representation of reference frames used in direct georeferencing [42]

## 2.2.2. Scene Restitution

Scene restitution refers to the overall process of transforming and correcting the image in such a way that the original scene can be restored. Within the scope of remote sensing, the original scene is the Earth's surface, imaged from a spaceborne scanner. The goal is to create a 3D model of the image, such that each pixel in the image is mapped to the correct geographic coordinates. Three main aspects are addressed: reference frame transformations, distortion correction, and orthorectification.

### Reference Frame Transformations

The first aspect of scene restitution, in essence, consists of a series of coordinate transformations (dependent on which Reference Frame (RF) the IO parameters are expressed in) to map the pixel coordinates of an object in the image to the Earth's geographic coordinates. Depending on the approach used, a number of intermediate reference frames may be defined for the system, through which the transformations are performed. Each platform will have its own definitions of reference frames, as can be seen in Figure 2.3.

Müller *et al.* [43] give a detailed overview of the exact steps used in direct georeferencing. It analyzes the different corrective processes required for common line scanners, such as pushbroom and whiskbroom sensors. The theory presented in this study is independent of platform, such that it can be utilized on both aircraft and spacecraft. The developed direct georeferencing program is also validated on both airborne and spaceborne cameras.

Referring to Figure 2.4 as an example, the starting RF is the pixel-space inside the camera (labeled as "sensor"). The position vector of the object of interest within the sensor RF is denoted as $\mathbf{r}_{object}^{sensor}$. The final goal is to

[7] https://geodetics.com/georeferencing-and-uavs/

determine the georeferenced position of this object: $\mathbf{r}_{object}^{Earth}$. Given an image captured at a certain time $t_j$, the following relation can be used to transform the object of interest from the sensor RF to the geographic RF:

$$\mathbf{r}_{object}^{Earth}(t_j) = \mathbf{r}_{sensor}^{Earth}(t_j) + s(t_j) \cdot \mathbf{R}_{body}^{Earth} \cdot \mathbf{R}_{sensor}^{body} \cdot \mathbf{r}_{object}^{sensor}(t_j) \tag{2.1}$$

$$\mathbf{r}_{sensor}^{Earth}(t_j) = \mathbf{r}_{pos}^{Earth}(t_j) - \mathbf{R}_{body}^{Earth}(t_j) \cdot \mathbf{r}_{pos}^{body} + \mathbf{R}_{body}^{Earth}(t_j) \cdot \mathbf{r}_{sensor}^{body} \tag{2.2}$$

Where the position vectors and transformation matrices are described below in Table 2.1.

**Table 2.1:** Variables used in equations 2.1 and 2.2

| Variable | Description |
|---|---|
| $t_j$ | Time of image capture |
| $s$ | Scaling factor; can be determined through orthorectification or with use of EOs parameters [50, 68] |
| $\mathbf{r}_{sensor}^{Earth}$ | Position vector of the sensor projection center in the Earth RF. See Equation 2.2 |
| $\mathbf{r}_{object}^{sensor}$ | Position vector of the object in the sensor RF |
| $\mathbf{R}_{sensor}^{body}$ | Transformation matrix from sensor RF to the body (spacecraft) RF; described by rotation about Euler angles $(\epsilon_x, \epsilon_y, \epsilon_z)$ |
| $\mathbf{R}_{body}^{Earth}$ | Transformation matrix from body RF to Earth RF; described by rotation about Euler angles $(\omega(t), \varphi(t), \kappa(t))$ |
| $\mathbf{r}_{pos}^{Earth}$ | Position vector of satellite in the Earth RF, measured by the orbit position sensors (GPS) and stored as EO parameters |
| $\mathbf{r}_{pos}^{body}$ | Position vector of the GPS measurement sensor in the body RF |
| $\mathbf{r}_{sensor}^{body}$ | Position vector of the imaging sensor in the body RF |

#### Distortion Correction

A second aspect to address is distortion correction. Figure 2.5a illustrates oblique image distortion caused by images captured off-nadir. Since the Earth's surface is an ellipsoid, a planar image will also inherently be distorted towards the edges. These distortions can be corrected for using robust geometry (see Figure 2.5b); where the exact equations depend on the order of transformations and reference frames used. Additionally, the atmosphere will cause refraction to occur for the electromagnetic waves travelling through it. As detailed in Liang and Wang [33], this distortion is nonlinear and causes errors in the spatial relationship of the image pixels and the perspective center. This atmospheric distortion can be corrected using empirical models and measurements of the air pressure at the moment of image capture.

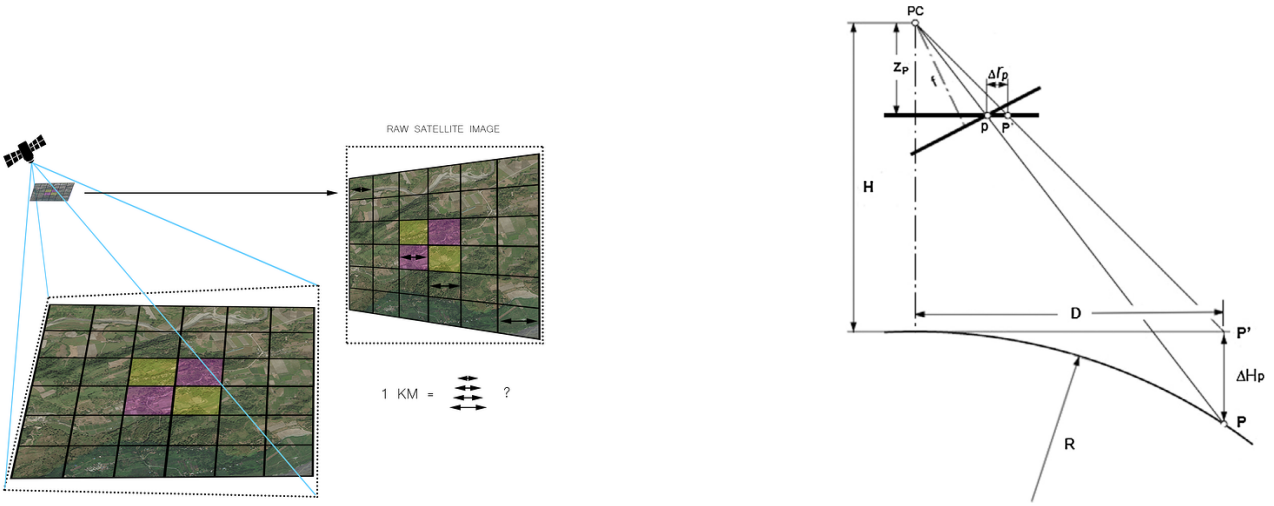
#### Orthorectification

Third, to further improve the accuracy of the georeferenced image, orthorectification may be applied; see Figure 2.6. This process adds a third dimension to the image data, such that each pixel gains an elevation coordinate (in addition to latitude and longitude). The georeferenced planar image is fit to a 3D Digital Terrain Model (DTM) or Digital Surface Model (DSM). This step can be computationally intensive, as an iterative algorithm is used to find the intersection point of a pixel's "sensor look direction" (a vector from the imager to the image plane) and the DSM; depicted in Figure 2.6b. An intersection point is found by first starting with an initial horizontal reference plane and projecting the pixel onto this to determine a horizontal position. The position is then retrieved from the DSM, which returns a new plane at a more accurate elevation. The process is repeated with the given pixel and the new plane, which continue to refine the resulting elevation of the point of intersection, explained in detail by Chen *et al.* [11]. The iterations are stopped once the changes in horizontal direction fall under a certain error threshold.

### 2.2.3. Alternative Methods

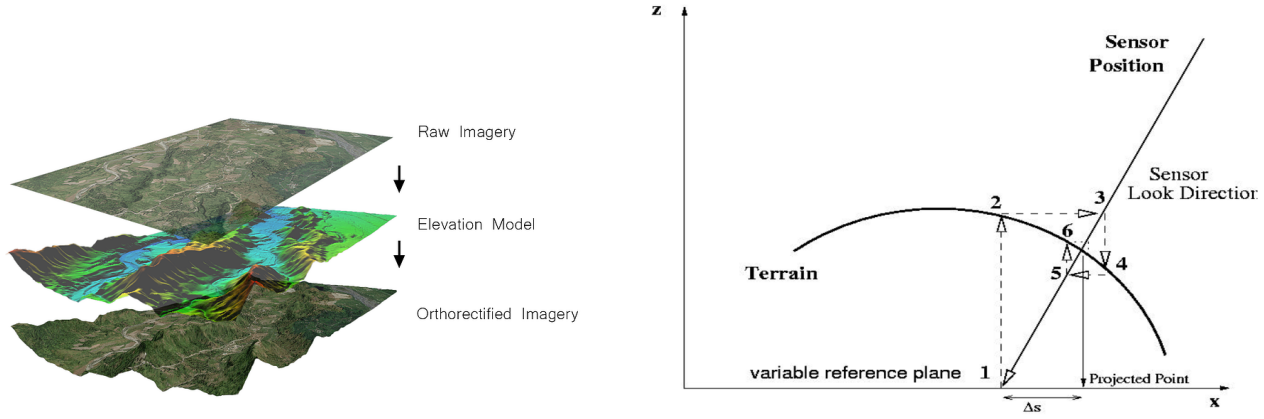
Alternatively, direct georeferencing can be performed using something called the Rational Functional Model (RFM). Explained in detail in Section 2.3.4, the RFM is essentially a generic empirical model relating image-

[8] https://www.intermap.com/blog/orthorectification-in-a-nutshell




**(a)** Oblique imagery.[8] **(b)** Curvature-induced distortion, [68].

**Figure 2.5:** Oblique and Earth curvature distortions of satellite imagery




**(a)** Orthorectification visualization, [55]. **(b)** Iterative intersection point determination, [43].

**Figure 2.6:** Orthorectification process

coordinates to geographic coordinates. The rational functions contain Rational Polynomial Coefficients (RPCs), which effectively encode the EO and IO parameters. Satellite image providers commonly supply these RPCs as ancillary data, so customers can independently georeference the data. The RPCs encode both persistent parameters determined during initial calibration of the sensor (IO parameters), and variable parameters measured at the moment of image capture (EO parameters); similar to the direct georeferencing process described above. Using IO and EO parameters in the form of RPCs allows georeferencing to be performed without the use of feature data extracted from the images, thus it can be considered an alternative DG method. However, the persistent values encoded in the RPCs are determined during the sensor calibration process, which relies on using ground control points to compare camera images with reference data; thus this method is not regarded as purely direct georeferencing.

### 2.2.4. Accuracy

As discussed in Section 2.1.3, the final georeferencing accuracy of the image data is an important figure to consider. Due to the various measurement errors and sensitivities of the satellite platforms, exact geo-localization accuracies will differ. By inspecting specific satellite imagery datasets that rely on DG, and analyzing studies that aim to improve the geo-localization accuracies, a common range of accuracy values for DG can be established.

Regarding available satellite imagery datasets, Table 2.2 summarizes a few existing sensor systems which utilize direct georeferencing. It can be seen that the geolocation accuracies are given in terms of upper bounds, since the accuracy typically decreases for greater off-nadir angles.

**Table 2.2:** Accuracies of sensor systems using direct georeferencing for geolocalization [8, 16, 18, 41]

| Sensor System | GSD @ nadir [m] | Geolocation Accuracy (CE90) [m] |
|---|---|---|
| SPOT-5 HRG | 10 | 50 |
| WorldView-3 | 1.38 | 5.6 |
| GeoEye-1 | 0.5 | 4.4 |

DigitalGlobe's Worldview satellite constellations can be considered state-of-the-art commercial HSRI. The values listed in Table 2.2 refer to basic mono images direct-georeferenced with Rational Polynomial Functions (RPFs) (without the use of ground control points). Using additional processing methods such as stereo imagery and orthorectification, these geolocalization errors can be refined to 3.0m horizontal and 2.8m vertical with RMSE of 1.9m and 1.6m respectively [41].

### 2.2.5. Error Sources

Although direct georeferencing is very robust and relatively simple to implement, this process can be subject to a multitude of errors, especially at high resolutions. Of primary concern is the propagation of measurement errors due to projection from the spacecraft camera to the surface. Since direct georeferencing is an extrapolation of the camera's EO parameters, these errors will be amplified by the satellite's orbit altitude [30]. Liang and Wang [33] describe most of the systematic errors present in direct georeferencing, which is summarized below:

- Satellite position measurements (including velocity and altitude)
- Sensor scan rate
- Clock synchronization between imager and other sensors
- Atmospheric refraction
- Earth curvature and terrain
- Earth rotation
- Earth ellipsoid transformation

In addition to the various distortion correction processes which address major sources of error, smaller errors are also prevalent. Zhao *et al.* [68] analyzes methods to decrease systematic errors from different reference frames for oblique imagery. Regarding the camera platform, a number of errors can be corrected for using calibration: lens distortion, boresight misalignment, IMU-GPS lever arm, and other interior orientation parameters. A visual depiction of these parameters can be seen in Figure 2.3.

#### Digital Surface Model

The accuracy of the DSM will also contribute to the overall georeferencing accuracy. As depicted in Figure 2.7, the spacecraft in the upper right corner images the surface at various oblique angles.

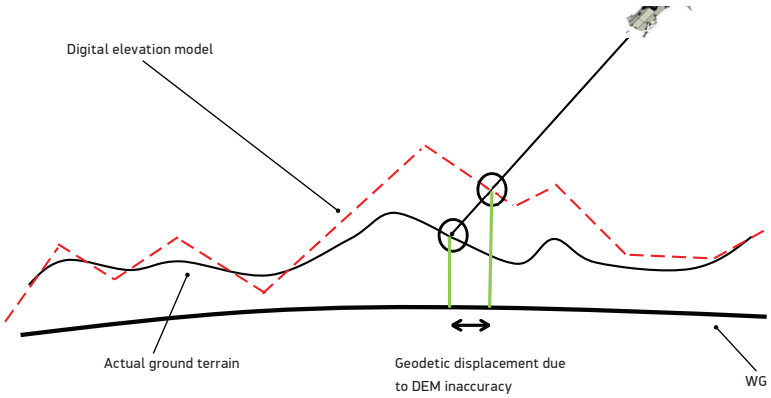



**Figure 2.7:** Georeferencing error due to DSM inaccuracies [13]

This error is generally corrected by either using a higher resolution DSM, or running more iterations as illustrated in Figure 2.6b. However, a high resolution DSM can be costly in terms of computational resources, or difficult to obtain from commercial sources.

In conclusion, direct georeferencing can be very accurate; $< 5m$ for some of the most modern platforms. Not being dependent on the image resolution for processing, the direct georeferencing spatial accuracy is tied directly to the accuracy and precision of the onboard positional and orientation instruments. Determination of EO and IO parameters is paramount, as well as reducing noise in measurement data. The accuracy of the georeferenced image can further be improved via orthorectification, which depends largely on the density of the DSM. Most studies conclude that while DG can be sufficiently accurate for certain applications, indirect georeferencing can provide a substantial improvement, as discussed in the following section.

## 2.3. Indirect Georeferencing

This section explores a newer and more accurate method: Indirect Georeferencing (IG). Relying on the processing of raw image data, this approach helps eliminate the dependency on the EO and IO parameters found in direct georeferencing, but introduces new challenges and uncertainties.

Image-based georeferencing is based on matching the captured image to a reference image with known position data. As depicted in Figure 2.8, key features are identified in the query image (left), and matched to the most similar keypoints of the known (previously georeferenced) image (right). These keypoint pairs are referred to as Ground Control Points (GCPs), and form correspondences between pixels in the image reference frame and pixels with known geographic coordinates. Using these pairs of coordinates, a general polynomial transformation function can be derived to map specific pixels (or points of interest) in the image to geographic coordinates.

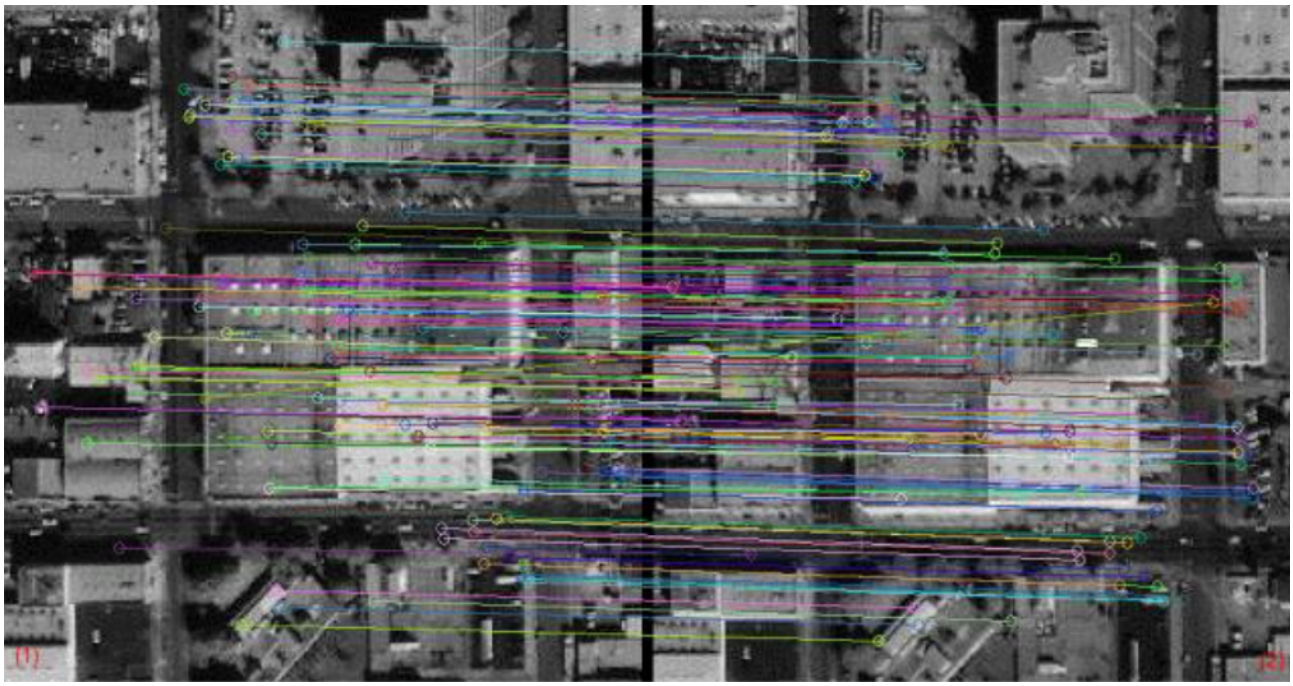


**Figure 2.8:** Example of GCP tie-points between two similar images [34]. Note how the matched features are predominantly permanent structures that exist in both images.

This approach can produce much better results than direct georeferencing, achieving even sub-pixel geolocalization accuracies. Since this process is significantly more complex, it also comes at a cost of higher computational resource usage. Basic use cases require only a few images to be georeferenced, and rely on manual selection of the ground control points to minimize error. However, for automated data pipelines, the detection, description, and matching of the GCPs depends on advanced Computer Vision (CV) algorithms.

With the latest advancements in Artificial Intelligence (AI) for computer vision, recognition and matching of features and objects has become a common application. Recent studies have demonstrated their potential to be used with IG, which is explored below. Naturally, traditional non-AI CV algorithms are proven technologies and are still common in georeferencing pipelines. Much research today continues to improve the properties of these classical CV algorithms.

In short, the IG general process consists of three primary steps: detecting and describing key features in the images, matching the features to generate tie-points (GCPs), and computing the coefficients of the transformation

model.

Starting with the first step, both traditional computer vision algorithms and deep learning methods are introduced below, as both have been proven to be effective in this application. The storage and preselection of reference images is then briefly discussed, followed by a brief overview of the final coordinate transformation model. Finally, state-of-the-art IG accuracy is presented, with a discussion on sources of errors specific to IG and how to mitigate them.

### 2.3.1. Traditional Computer Vision Algorithms

Traditional computer vision algorithms have predominantly been used in the past decades for identifying and describing key features in images. These algorithms are commonly referred to as being "handcrafted" since their performance is dependent on the tuning of specific parameters. These parameters are optimized for specific use cases, and therefore tend to be extremely efficient, yet very sensitive to changes in input data. Traditional algorithms also typically require separate functions for feature detection and feature description, whereas many modern Deep Learning (DL) algorithms combine these steps into a single process. Note: there are a few exceptions to this such as Orientated FAST and Rotated BRIEF (ORB); as is explained below.

#### Feature Detectors

The first step to be able to compare two images is recognizing key features within the images. This is typically done either by identifying corners and edges, local blobs, or larger regions within in an image. Many traditional algorithms have been developed and improved over the past decades to automatically detect these features. Deep learning algorithms to identify features are also rapidly emerging, due to their ability to be tuned for specific properties such as robustness and specific types of feature extraction. Most importantly, as described by Merkle [38], feature detectors must be *robust* to local distortions between comparable images, must reliably extract features *repeatably* (a single detector should detect the same features with varying viewing conditions), and must be *accurate* in identifying the locations of the features.

#### Feature Descriptors

In order to allow for features to be matched between images, they must be characterized in such a way that enables them to be directly compared without the use of spatial information. This is important because if two images of the same subject are significantly deformed relative to one another, the correspondence of extracted key features must still be very similar.

According to Zitová and Flusser [71], feature descriptors must adhere to four primary conditions:

1. *Invariance* - the local deformation of the image should not affect the feature description
2. *Uniqueness* - the description should solely identify a single feature, and be able to differentiate from other features
3. *Stability* - if the feature is slightly deformed, the new description should also be very similar to the original description
4. *Independence* - if the description is stored as a vector, the individual elements should be functionally independent

These criteria form the basis of most descriptor algorithms. As with feature detectors, there exist a multitude of variations, each having slight advantages or improvements. One such descriptor frequently used for georeferencing applications is the Binary Robust Independent Elementary Features (BRIEF) algorithm. BRIEF analyzes the surrounding pixels around a feature point and produces a 128-bit vector of relative intensities [56]. However, this descriptor performs poorly if rotations are present, due to the order of values in the vector [27]. Improvements such as Fast REtinA Keypoint (FREAK) utilize a more efficient sampling pattern and comparison method to reduce the computational cost. Mukherjee *et al.* [40] provide an in-depth comparison of detectors and descriptors.

##### *Efficiency improvements*

Certain feature detection algorithms are developed specifically to optimize for computational efficiency. Speeded-Up Robust Feature (SURF), a blob detector developed by Bay *et al.* [6], is a detector and descriptor that accelerates the Hessian-Laplace detector using box filters. The features are also described with orientation information,

derived from wavelet responses by applying Gaussian weights [27]. Additional optimizations such as those described by Cai *et al.* [9], report a computational decrease of up to 50% with only a minor drop in recall and precision.

The Features from Accelerated Segment Test (FAST) descriptor, proposed by Rosten and Drummond [52], is a corner detector, which can also be used to track features. Compared to similar detectors, this was optimized for computational speed, which helped it become a favored detector. Many machine-learning algorithms are based on or use this detector due to its efficiency.

Another descriptor, ORB, is a fusion of FAST detection and the BRIEF descriptor with modifications to improve orientation description. Karami *et al.* [27] compare the performances of three popular algorithms: Scale Invariant Feature Transform (SIFT), SURF, and BRIEF.

#### Feature Matching

Once the key features have been identified and extracted from the images in the form of descriptor vectors, they must be matched to form tie-points. Assuming one of the images has already been georeferenced (the reference image (RI)), the key features from that are considered to be ground control points. The other image is then referred to as the query image (QI). Matches are identified by computing a similarity measure between the descriptor vectors of the features, the most common being the Euclidean distance.

### 2.3.2. Deep Learning Methods

Recent developments in machine learning have led to a rapid increase in the use of neural networks to extract features. These non-traditional algorithms have the added benefit of being trainable to recognize a specific type of feature, and can therefore be extremely beneficial in extracting and matching tie-points.

Being a field of study within itself, the models listed here are only a small subset of what is available. These are provided to give an overview of the general process and capabilities of deep learning models in the scope of image matching and feature extraction. For a more detailed discussion on the topic, Szeliski [56] provides a very comprehensive overview of the field of computer vision, including deep learning methods.

#### Trainable Multipurpose Models

Combining methods used in classical computer vision algorithms, such as filters and convolutions, with machine learning techniques described above have resulted in the development of Convolutional Neural Networks (CNNs). This is the most widely used type of deep learning model for feature extraction and recognition. CNNs are typically trained on a large dataset of images, which are labeled with the desired output. The network then learns to recognize the features that are associated with the desired output. This ability to train the model for specific applications is the fundamental difference with traditional CV algorithms.

CNNs are used for a wide variety of purposes involving images, and often combine the convolutional layers that extract features with fully connected layers that process the information to produces classification or segmentation results. From a single image, the complete model can therefore not only identify and describe the pixels as features, but also compare them to neighboring pixels to identify larger objects, and then classify the objects. This ability to combine multiple steps into a single model is one of the main advantages of CNNs.

In the scope of image matching, and ultimately georeferencing, two main approaches can be taken. The first is to train a model which identifies and describes key features in the set of images, and then matches the features using either a traditional algorithm or another CNN. The second approach is to train a model to directly match the images, without the need for feature extraction. This approach is known as Content Based Image Retrieval (CBIR), which essentially encodes the image in such a way that the entire image can be compared to other images. This process is known as Principal Component Analysis (PCA), and is used for image retrieval applications such as Google's reverse image search [56, Sec. 7.1.4]. In Section 3.5 it is chosen to focus on keypoint extraction methods.

For keypoint matching, a common approach is to use a Siamese network, which are two parallel networks that share the same weights. The two networks are fed two images, and extract features from both independently. Subsequent fully connected layers then compare the features and produce a similarity score of the two images [37].

#### Loss Functions

Key to the success of this process is the creation of a representative loss function. This function quantifies the correctness of the model's output with respect to the desired output, and produces a value that should be minimized. During the training phase, optimizers are used to adjust the weights of the network to minimize the loss function. The weights are initialized randomly, and are adjusted iteratively until the loss function is minimized. The weights are then frozen, and the model can be used for inference on new data.

#### Common Models

Some of the first successful CNN architectures used for feature extraction include L2-Net, HardNet, and SOSNet, which had matching performances comparable to SIFT [60]. Each of these networks proposed new improvements, such as better loss functions or novel training strategies. One such function still common today is the Triplet Loss Function, which trains the network to "learn representations by distance comparisons"[25] rather than minimizing a single loss value.

Subsequently, other models emerged that are designed with the "end-to-end" training approach, which combine several parts of the detection, description and matching pipeline [23]. The improved loss functions teach the models to extract only features likely to be matched between images, and describe them in generic but repeatable ways. A couple of notable examples are LIFT [66], LF-Net [46], SuperPoint [12] and SuperGlue [53].

More recently, a new approach has been proposed called Describe-to-Detect (D2D), which reverses the traditional process "by first describing and then detecting the keypoint locations"[59]. This has shown to be effective in identifying keypoints that contain salient information of the underlying image, rather than of the specific pixels themselves. For example, a keypoint will only be detected if it describes a significant region of the image, rather than every edge or corner being detected. Dusmanu *et al.* [14] introduced a network D2-Net, which used this principle to create a feature extraction model robust to significant illumination changes. In the scope of satellite imagery, models using the D2D approach have significant potential, as the low spatial resolution produces "grainy" images where each pixel can be considered a corner or edge.

#### Transfer Learning

A final benefit of deep learning models is the potential for transfer learning. Training models is a computationally expensive process, especially on large datasets such as ImageNet. However, once the models weights are optimized, they can be used as a starting point for other models. The model can be repurposed for other inference tasks, and a new training cycle can be started using new representative datasets. The model's underlying ability to extract meaningful features will still be present, but will be fine-tuned to perform on the newer dataset. This is especially useful for satellite imagery, as the objects and features present in the images are very different to those in the original training dataset, but the main process of detecting salient features is still the same.

#### Feature Matching

Once keypoints have been extracted from both the QI and Reference Image (RI), they are matched to produce a set of corresponding points called Ground Control Points (GCPs). This can either be performed within the keypoint model, as is the case with Siamese networks, or with a separate algorithm. In most cases, the feature extraction model operates on only one input image at a time, and returns a set of features descriptors. As with the traditional CV algorithms, these sets of descriptors can be matched using a similarity measure such as the Euclidean distance, cosine similarity, or Hamming distance.[9]

The two most common matching algorithms are Brute-Force and FLANN-based matchers. Brute-Force simply matches all descriptors from the QI to all descriptors from the RI, and returns the best matches. This is a computationally expensive operation and thus not suitable for large datasets. FLANN is based on a K-Dimensional Tree (K-D Tree), which is a binary data structure that stores the descriptors in a tree-like structure. The tree can be traversed to find the closest matches, allowing for faster and more efficient matching, at the cost of accuracy. Bian *et al.* [7] provides a detailed explanation of the matching process, as well as additional lesser-known matching algorithms. In general, FLANN only provides real benefits for datasets greater than 1000 descriptors.

### 2.3.3. Reference Images Database

As part of creating the GCP tie-points, a reference image of the same geographic region as the query image must be selected. This requires a library or database of images that have previously been georeferenced, such that for

[9] https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

each extracted feature, the corresponding geographic coordinates are known. Depending on the operational region size of the satellite (of which query images will be taken) and the resolution of the reference images, the storage requirements can be extensive. For example, a worldwide land surface dataset of 10m GSD color images (water bodies excluded), requires roughly 200TB of data storage [19].

Once an image has been captured and requires processing, the preselection of the appropriate reference images is a crucial step to obtain accurate georeferencing results. This can be done in two ways:

1. By providing a first-estimate of where the image is likely to be, typically using a direct georeferencing estimate, or other image-capture metadata
2. By performing CBIR, which searches the database to find similar matching images. For this approach there are both traditional and machine learning based algorithms

This reduces the number of reference images that need to be processed to a manageable subset. While CBIR is technically possible, an initial location estimate is more commonly used, since visually similar images are not necessarily geospatially similar and can return more false positives.

#### Initial Location Estimate

Initial direct georeferencing estimates of the query image location can provide a quick solution of selecting the correct reference image. As explained in Section 2.2, the orientation and position parameters of the camera at the moment of capture are used to calculate where the field of view was pointing and what the geographic location of the object is. Based on this estimate, one or more reference images closest to this geographic point can be selected, and merged to form a single large image that is likely to encompass the complete region of the query image. However, this requires the imaging IO and EO parameters to be known.

### 2.3.4. Coordinate Transformation Model

A large collection of matching tie points, as depicted in Figure 2.8, can finally be used to compute a transformation model between the two images. This model can then be used to map known geographic coordinates from the reference image to specific pixels in the query image, thus completing the georeferencing process.

As mentioned prior, these are known as empirical geometric processing models, because they perform a best-fit estimation given the set of tie-points. Liang and Wang evaluated the current state-of-the art methods in 2020, and summarized: "Common empirical geometric processing models mainly include the general polynomial model, the DLT model, the affine transformation model, and the RFM" [33, p. 77]. The RFM is the most generalized and robust model, because it "is an extended expression of the general polynomial model, the DLT model, and the affine transformation model. . . " [33, p. 78].

Empirical models can generally be categorized as either global or local [38]. Global models use the whole set of tie-points to generate a universal transformation between the two images. If more tie-points are present than necessary for the generation of model parameters, a least-squares estimation is used. This adds robustness to possible feature deviations or errors. As explained below in Section 2.3.7, RANdom SAmple Consensus (RANSAC) is commonly used to eliminate outliers.

For high accuracy applications, the local transformation models are preferred if enough tie-points are present and distributed sufficiently throughout the image. These models produce different mappings for various regions between the images, with the major benefit that it is robust to localized distortions. Common models include piece-wise linear and cubic functions, elastic models, and local weighted mean models [38]. A drawback of these local models is the increased computational cost compared to the global alternative. Depending on the available resources, this may be an important factor to consider.

A second-order polynomial transformation is ultimately used in this design, and is detailed in Section III of the research paper below.

### 2.3.5. Accuracy

Determining an exact value to describe the geolocalization accuracy of images processed using the IG method is very difficult, as the process is dependent on many parameters. Most notably, the quality (GSD, filtering, noise, etc.) of the query and reference images will largely influence to what extent features can reliably be extracted and matched. Thus exact accuracy values may not directly reflect the capabilities of the indirect georeferencing methodology. However, one method to identify current achievable accuracies is to analyze the results of studies

done on the refinement or improvement of common georeferencing processes, and generalize between studies to gain a ballpark estimate.

Aguilar *et al.* [2] dedicated a study directly on the improvement of direct georeferenced images using extracted ground control points. As described previously in Section 2.2.4, these authors found that some of the best achievable accuracies without the use of GCPs is around 4m (CE90). The study was performed on two separate datasets, WorldView-2 and GeoEye-1, to ensure reliability of the results. Furthermore, it extensively evaluates the accuracy of the reference ground control points used, which is important as performing IG based on incorrect GCPs will create erroneous results. A certain number of GCPs were reserved as ICPs, and intentionally not used for the georeferencing process. The images were indirect georeferenced using the RFM, in which the RPCs were estimated from a selection of seven GCPs. The results (summarized in [2, Tab. 3]) conclude that the horizontal accuracy improvements were between 54% and 77%, depending on the precise approach used (reference datasets). This resulted in an absolute accuracy of 0.78m CE90 or 0.52m RMSE.

Another study, by Zheng *et al.* [69], focused on evaluating the geometric accuracies of the images before and after using ground control points acquired in the field by GPS surveying. For each of the five high resolution satellite image datasets (Pleaides, SPOT6, ALOS, ZY-3, TH-1), pixel RMSEs are calculated using ICPs. Expressing the improvements in terms of pixels helps generalize between sensors that have different resolutions. As a result, in all cases the accuracies improved significantly. The best could be seen for the ALOS dataset, which achieved sub-pixel resolution of 0.775px RMSE. Absolute spatial accuracies could also be found to be around 1.5m.

In conclusion, indirect georeferencing tends to be more accurate compared to direct georeferencing, but at the cost of greater computational complexity. The reliance on the extraction of image features effectively abstracts the process from the satellite's geometric configuration, allowing it to easily be applied to various platforms. However, these additional steps introduce additional risks and sources of error. Luckily, much recent research has been focused around improving and optimizing the required algorithms. Depending on the final system requirements, different parameters can be tuned to meet the necessary accuracy and final performance specifications. The following section builds on this in the scope of a full satellite image processing pipeline.

### 2.3.6. Error Sources

As described in Section 2.1.3, the accuracy of the final georeferenced image is a matter of how close the calculated geographic coordinates of a feature in an image are to its actual location in the real world. Since indirect georeferencing is a combination of image processing and mathematical transformations, the sources of error can be traced back to these categories. The errors listed below are not exhaustive, but aim to establish the most common sources and solutions proposed in literature.

All sensor data is subject to the possibility of errors and therefore the data processing pipelines must account for these. Errors are typically classified as one of three types, based on their source: systematic, accidental, or gross. Systematic errors are considered identifiable and therefore also fixable; once addressed they do not return. Accidental (also referred to as random) errors, can be attributed to noise or other uncontrollable factors. Gross errors are considered to be large blunders, often human error or a significant unexpected deviation. This can be a result of data corruption, recording mistakes, inadvertent mixing of coordinate systems, or incorrect dataset selection; to name a few examples.

Assuming that the data only contains accidental errors is unrealistic, thus researchers in the field of photogrammetry have since the 1970s focused on mitigating the existence of gross and systematic error. The overall ability for the system to detect and handle these errors is referred to its robustness. The following parts discuss systematic and gross errors in the scope of indirect georeferencing, and present the common methods used to minimize their influence on the final result.

#### Systematic Errors

Due to the dependence on features extracted from the images, indirect georeferencing is limited to the existence of unique characteristics present in both the reference and query images. Textureless or generic images such as those over water, snow, sand, or clouds are extremely difficult to automatically extract unique features from. Artificial ground control points can be placed on these surfaces, such as proposed by Padró *et al.* [47] and Zhang *et al.* [67], who evaluate the best placement and location for a limited set of GCPs. Additionally, clouds create a significant challenge, as they block visible features in the optical spectrum, and if not properly removed, cloud edges are detected as (false) distinct feature in an image. Clouds can be mitigated by masks, which simply remove the obscured parts of the image. Ground regions with texture-less surfaces can also be avoided when utilizing the indirect georeferencing pipeline. This can be considered a downside to choosing IG over DG.

Because the IG transformation model is considered empirical (a best-fit estimation of the provided parameters), it is also a source of systematic errors. Additionally, generic computation and program implementation errors (bugs) are always a risk. These are best mitigated by model verification and validation.

Ill-posedness and over-parameterization are phenomena attributed to the RFM due to highly correlated rational polynomial coefficients, and make it difficult to solve the model accurately [31]. Because these are known issues, they can be corrected for and are classified as systematic errors.

#### Gross and Accidental Errors

Since systematic error can be corrected for by adjusting the georeferencing model and overall system, it is the combination of gross and accidental error that require additional steps to be added to the indirect georeferencing pipeline. A common approach is to recognize that if the performance of georeferencing system is robust to unexpected gross errors, this will inherently also cover the accidental errors [35]. Considering the critical importance of GCPs in the IG pipeline, the existence of gross errors in the coordinates (both geographic and image-space) can have substantial impact on the final accuracy.

### 2.3.7. Error Mitigation - RANSAC and LMS

For typical datasets where the errors follow a normal (Gaussian) distribution, the regular least-squares method can be used. However, data containing outliers require more robust methods. Two main approaches are used in the context of indirect georeferencing to account for gross error: Least Mean Squares (LMS) and RANSAC. Both start by randomly selecting subset of data points, from which the model is generated. The model is then used to calculate estimated locations for all the data points. These estimates can then be compared to the actual locations, expressed as a residual. This process is then repeated for multiple random subsets of the data.

RANSAC scores each of the iterations by counting the number of data points with a residual under a certain threshold, declaring these points as “inliers”. The iteration with the most inliers is considered to be the correct solution, and the remaining outliers are regarded as gross errors to be removed. LMS functions similarly, but finds the median of the residuals squared for each iteration, and selects the smallest median value. This process is explained in detail in Szeliski [56, Sec. 8.1.4] For IG, for each iteration a subset of the available GCPs is selected, from which the RFM is derived. The RFM is then used to compute estimated GCP coordinates for each of the GCPs in the complete set. The estimations are thus geographic coordinates derived from image-space coordinates using the RFM, or vice-versa. This is explained by Marsetič *et al.* [36], who implemented RANSAC in their automated georeferencing pipeline.

# 3

# Systems Engineering

The overall design process of this georeferencing system is based on the classical V-model, an example of which is shown in Figure 3.1. Due to the academic nature of this project, the approach is slightly different from a typical system design, as the need is derived from a lack scientific knowledge rather than a client requesting a specific problem to be solved. Nonetheless, the V-Model could easily be adapted to this, by first defining potential uses-cases and thus potential "users". These are covered in Section 3.1.

The scope of this thesis covers the approximately two-thirds of the V-model: from the "concept of operations" through "system verification". "System validation" and "operations & maintenance" are not conducted in this research, as these require the implementation to be mature enough to accept non-idealized data; a phase reserved for future work.

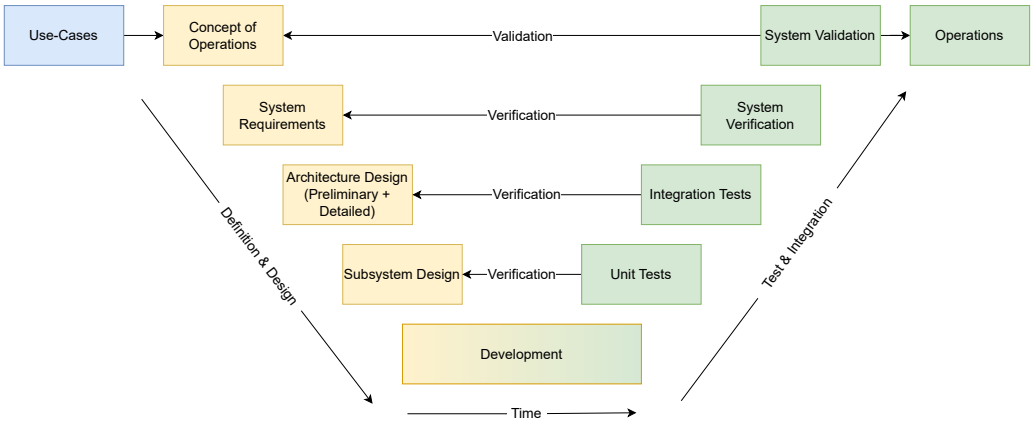



**Figure 3.1:** Systems engineering V-model approach[1]

The following sections detail the design process of the system. Based on the potential use cases, the concept of operations is outlined with the help of context diagram, which defines the system's boundaries and interfaces in the context of the identified use case. This is followed by a formalized list of system requirements which are derived from the functional and performance requirements of the use cases. A list of constraints are also established based on the context diagram, which help define the scope of this research.

Subsequently, two levels of design take place: first, a preliminary design is conducted to establish which general method of georeferencing is most suitable for the system, and where the main processing will take place. This is followed by a detailed design of the system architecture, which uses a functional-flow diagram distribute the georeferencing steps between terrestrial and onboard resources in the most efficient manner.

The development and verification of the system are covered in the research paper, in Chapter 4. Unit tests implemented throughout the development phase were used as verification of the individual components, and

[1]See Appendix A for an enlarged version.

the results of the tests performed in the research paper serve as verification of the system architecture as a whole. These requirement verification results are presented in Section 5.1.

## 3.1. Use Cases

Identifying specific use cases that could directly benefit from a real-time georeferencing pipeline are essential to the design process. This section presents a few examples of such use cases. The examples are not exhaustive, but are intended to gain quantitative values on the georeferencing accuracies needed to successfully function.

The real-time component of this system filters the potential uses-cases significantly. Many standard earth-observation satellites provide valuable monitoring information to their users, but classical examples such as agricultural monitoring, geologic and oceanographic research, or temporal change detection are not time-critical. Obtaining data within a few hours of the image capture is sufficient for these applications.

Time-critical applications are those that require the data to be available within minutes of the image capture. This is the case for disaster monitoring, where the data is used to coordinate emergency response teams. Another example is the monitoring of illegal activities, such as deforestation or illegal fishing. In these cases, the data is used to identify the location of the activity, and coordinate an immediate action. The sooner the data is available, the more effective the response can be.

For this first iteration of the system, the focus is to design a system that can make use of identifiable features in the images. This filters the potential use cases to those over land. The most obvious are natural disasters, such as floods, fires, or earthquakes. Although these events are often accompanied by a large amount of smoke, dust, or water, which can obscure the view of the ground, and cause additional challenges for feature extraction algorithms, the purpose here is to identify *potential* uses-cases, and most importantly the required positional accuracies.

**Table 3.1:** Potential real-time georeferencing system use cases and their required geospatial accuracies

| Application | Dataset | Spatial Resolution | Geospatial Accuracy (RMSE) |
|---|---|---|---|
| Earthquake - Temporal Changes[20] | Landsat8 OLI | ≈30m | 12m or better |
| Earthquake - Structure Damage[39] | Spot 6 & 7 | ≈1.5m | 6.6m |
| Wildfire - US/Canada FIRMS[2] | Landsat8 OLI | ≈30m | 12m or better |
| Wildfire - Australian (Onboard AI)[57] | PRISMA | ≈30m | 9.8m |

A brief study has been conducted on the existing satellite-based systems for these use-cases. The results are presented in Table 3.1. For this study, the applications were limited to wildfire detection and earthquake damage assessment. Flood detection was omitted, since the introduction of water bodies to the images would be similar to applying the georeferencing system to coastal or marine regions, which is outside the scope of this research. Wildfire detection is already a proven application, relying on infrared bands to identify thermal hotspots. Earthquake damage assessment is a more recent application, which uses the high-resolution optical imagery to identify damaged buildings and infrastructure. The detection of earthquakes is ground-based using seismic sensors, the localization of where the damage occurred is the time-critical component, and can often take hours or days to be determined using aerial surveys. Thus, satellite-based systems are currently being developed to prioritize data to be downlinked in order to provide this information within minutes of the event [21, 57].

Table 3.1 presents either application, with two implementation examples. For each, the data source used is identified, and from that the image's spatial resolution and positional accuracies can be derived. For the purpose of this research, these values are considered baseline accuracies to which the final system will be compared.

## 3.2. Concept of Operations

Based on the potential use cases a general concept of operations is constructed. Since the system is designed to be used in a variety of applications, the concept of operations is not specific to any one use case. Instead, it is a general description of the system's boundaries and interfaces. The general operations are as follows.

An image is captured by the multispectral imager of a satellite. This image contains no geospatial information, but sensor data from the Attitude and Orbit Control System (AOCS) on board the spacecraft can provide a

[2] https://www.earthdata.nasa.gov/faq/firms-faq#ed-geo-spatial

rough estimate of where the camera was pointed when the image was captured. This information is used to determine the approximate initial location estimate of the image. Next, the novel georeferencing system is tasked with determining the precise location of the image. The final result must be such that the coordinates of each pixel of the image can correctly be converted to the geospatial coordinates of the corresponding point on the ground. Additionally, the result must be available within five minutes of the image being captured, on an external platform. Ensuring the results are transmitted off-platform within the five-minute time constraint is a key challenge of the system. Since for the majority of an operational orbit the satellite is not in direct contact with the ground station, the results must be transmitted via one or more relay satellites, referred to as an Inter-Satellite-Link (ISL). Current generation ISL systems are extremely bandwidth limited, such that only a few hundred bytes can be transmitted at a time. This is a significant challenge for the system, as the captured images are typically in the order of several megabytes in size.

For the use cases listed above, the external platform would likely be the ground station. However, in order to future-proof the system such that it can also be used for other novel mission architectures, the receiving platform is generalized to be another system which is not the satellite itself. For example, this would enable the system to operate with distributed space systems, a field of mission architectures currently under significant research and development [58].

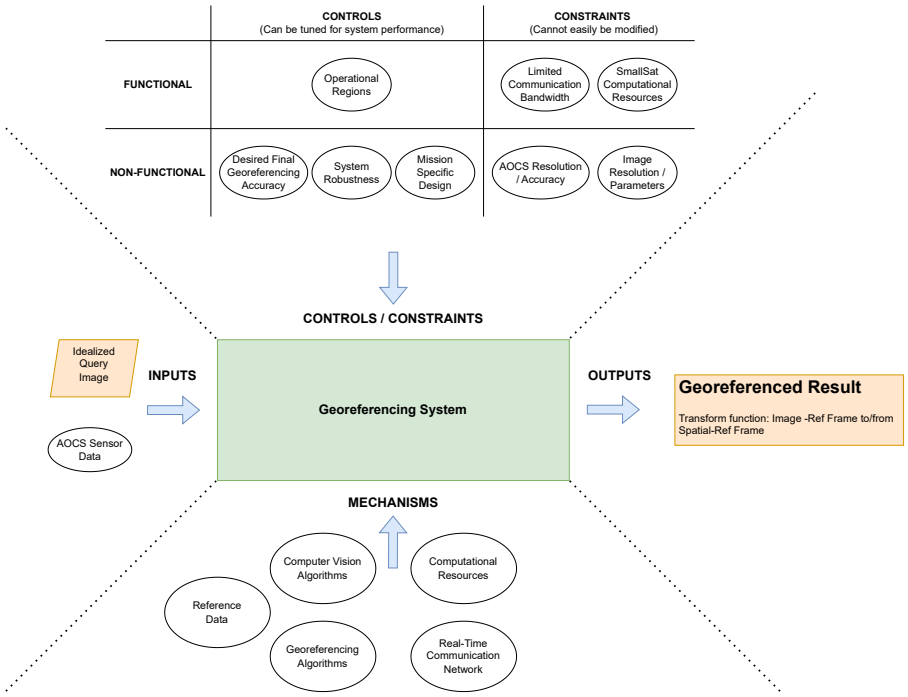



**Figure 3.2:** Context of operations diagram, used to generate requirements.[3]

Because the concept of operations is intentionally generalized, a context diagram is used to define the system's boundaries and interfaces [28]. Figure 3.2 shows this georeferencing system as a green box in the center, where the general concept of operation flows from left to right as described above. The partition the system represent the controls and constraints imposed on the system by the external interfaces, and the items below indicate the mechanisms and resources that enable the system to be implemented.

The mechanisms are relatively self-explanatory. These are resources available to the system, generalized for small satellites. The controls and constraints are slightly more detailed. These elements are categorized as either functional or non-functional, where the latter refers to items that are qualities of the final system. "Mission Specific Design" for example refers to what extent the design of system is tailored to the precise parameters of a specific mission. How the elements are categorized is not an exact science, but rather a way to organize which are non-negotiable (such as the dependence on an extremely limited ISL bandwidth), and which may be adjusted during requirements and constraints formulation (i.e. the targeted final accuracy).

[3] See Appendix A for an enlarged version.

## 3.3. System Requirements

The system requirements are derived directly from the context diagram. Since the goal of this system design is to abstract from a specific mission, these requirements are formulated in a manner such that they only describe aspects which the design of the system has a direct influence on. External requirements, such as the data to be used by the system or other inputs, are defined as constraints in Section 3.4. The functional and performance requirements originate primarily from the controls and constraints, as well as the final required output of the system

To differentiate between the system-level and sub-system and specification-level scopes, the requirements have been structured with parent-child relationships. Fulfillment of all child requirements is necessary to meet the parent requirement. This structure also ensures traceability and lends to clear justifications of each. Finally, the method of how the requirement will be verified after the system is complete is included for each requirement. A glossary of key terminology used in the requirements is presented first.

### 3.3.1. Terminology

The terms defined here are to eliminate any ambiguity in the requirements.

- **Off-platform** Any entity capable of receiving and processing data that is not physically connected to the satellite on which the system operates. For example another satellite or a ground station.
- **The System** The complete georeferencing pipeline resulting from this design process, which processes a query image to produce a georeferenced result. This includes all necessary reference data and processing algorithms. Specific hardware is not considered part of the system, rather a medium on which the system operates.
- **Query Image** The image captured by the satellite and which must be georeferenced.
- **Small Satellite** A satellite platform weighing less than 180kg with the computational resources of a Commercial-of-the-Shelf (COTS) CubeSat or low-powered consumer laptop [54].
- **Operational Region** The geographic region of which the query images used by this system will be captured. Explained further in Section 3.4.

### 3.3.2. Formalized List

Based on the structured engineering approach outlined in NASA's Systems Engineering Handbook, the requirements have been divided into two categories: functional (what the system does) and performance (how well the system operates) [24].

#### Functional Requirements

FUNC-1 *The final result shall consist of a transformation function between the pixel reference frame and the geographic reference frame.*
This requirement defines the final goal of the system and what constitutes the ability to perform georeferencing.
(Verification by: test and analysis)

FUNC-2 *The system shall be applicable to a small-satellite mission*
This is a high-level requirement specifying that the system should operate with the limited resources of a small satellite. It does not specify whether the system must operate on board the satellite, but constraint CNSTR-3 specifies it must also be compatible with an ISL. Due to the nature of the academic research of this system, it was considered most beneficial to develop a system independent of a specific hardware platform, while keeping the general limitations of small-sat hardware in mind. For that reason, a consumer-grade laptop was used for development and testing, and all design choices were made in light of keeping the computational load as minimal as possible.
(Verification by: analysis)

FUNC-2.1 *The system shall function on a computational platform with at most 2GB of RAM, and should function on a computational platform with* 1 GB *of RAM*
This specifies a hard upper bound of 2 GB, which is already low for a consumer-grade laptop and very realistic for a SBC; and gives a target of 1GB which is a realistic value for small sats. The target value of 1GB is sourced from Table 8-2 in the "2022 NASA State-of-the-Art of Small

Spacecraft Technology" report.[54]
(Verification by: test and analysis)

FUNC-2.2 *The system shall function on a single core of a consumer-grade laptop.*
In combination with PERF-1, which specifies the real-time requirement of the system, this requirement is fairly representative of a State-of-the-Art (SotA) small sat and can be tested using a simple containerized environment on the development laptop.
(Verification by: test and analysis)

FUNC-2.3 *The system shall function on a computational platform with at most 8GB of long-term storage, and should function with 2GB of storage; for a system operational region of 12100 square kilometers.*
This requirement establishes limited storage capacity that is available on board the satellite. The lower target value is based on current generation SotA small sat OnBoard Computers (OBCs), which have 4 GB of mass storage [1, 26]. The upper limit is based on the assumption that the storage capacity of OBCs will increase significantly in the near future; based on the larger SpaceCloud iX10-100 OBC [62] which has a storage capacity of up to 8 TB and also includes a Myriad X Vision Processing Unit (VPU). The system operational region size is based on the Sentinel Level-2A granule size, and is specified since the system's storage is a function of how large the reference dataset must be.[4] CNSTR-1 elaborates on this further.
(Verification by: test and analysis)

FUNC-3 *All final results produced by the system shall be capable of being delivered off-platform via an ISL.*
All results must be received by a different entity than the satellite on which the system operates. This ensures that the insights produced by the system can be used for an application that benefits from the real-time capability of the system. However, since this system does not include the communication link itself, this requirement specifies that it must be compatible with real-time communication methods. This is further specified in FUNC-3.1.
(Verification by: analysis)

FUNC-3.1 *The system shall function with downlink rate of 10kB per minute, and should function with a downlink rate of 1kB per minute*
The data size of the system results must be small enough to be transmitted via an ISL and still meet the real-time requirement (PERF-1). This requirement does not explicitly require an ISL to be used. If another communication method (ie a direct downlink) is available, that could be used instead. The exact method of transmission used is specific to the mission and independent of this system design. The minimum data rate of this requirement is based on current SotA ISL specifications, but the upper bound accommodates near-future ISL's which are expected to be operational within the next five years.
(Verification by: analysis)

**Performance Requirements**

PERF-1 *The system shall operate in real-time, with at most 5 minutes measured between the input of a query image and when the result is received off-platform.*
The 5-minute value originates from the current latency of the FIRMS system, which was identified as one of the use case examples. The start time is defined as the moment when the query image is given to the system, since any delays caused by capture are within control of this system. The end time is defined as the moment when the result is received off-platform. Since this is dependent on the data rates of the exact ISL service to be used, and mission-specific design is preferably avoided, this requirement serves as additional motivation to make the final design as platform-agnostic as possible.
(Verification by: test and analysis)

PERF-2 *The final system accuracy shall have an error less than that of the initial location estimate, and should equal that which is attained using existing post-processing, terrestrial-based methods.*
This specifies that the system should at least provided an added benefit on board, compared to just using the initial location estimate.
(Verification by: test and analysis)

PERF-2.1 *The final geopositional accuracy of the query image shall be at least 15m (RMSE) relative to the reference image.*
Precise specification that defines a measurable accuracy. This is based on the use case analysis performed, which identified earthquake detection and wildfire detection to use these accuracy

[4] https://sentinels.copernicus.eu/ca/web/sentinel/missions/sentinel-2/data-products

requirements.
(Verification by: test and analysis)

## 3.4. System Constraints

Defining the scope of this research was essential to ensure the development of the system followed an organized and systematic approach, and ultimately answered the original research question. As with the system requirements, the simplifications and research bounds specified here were derived from the constraints identified in the context diagram. Topics of interest that fall outside the scope define here are listed as recommendations for future work in Section 5.2, with descriptions on how they can contribute to the further development of this system.

The information presented in this section is a collection of items originating from multiple instances in the research timeline. For example, assumptions and simplifications presented here may be a result of specific design decisions made in the systems' engineering process. Important to note is that all simplifications and assumptions *result* from design choices, and unless otherwise noted, never drove the decision-making process. The purpose of presenting them here is for clarity and traceability.

The sections below justify and explain the decisions made on scoping the system's development. Most decisions are based on the principle that a demonstration of the proposed *architecture* is the primary goal of this research. Fine-tuning and further improvements are reserved for following research.

#### *Image Capture Location*

Any earth observation spacecraft will require some form of scheduling regarding when images will be captured. Due to the operational duty-cycle, in which the spacecraft must alternate between different operational modes (capture, data processing, communication, etc), it will not be capturing images continuously throughout the entire orbit. The most realistic scenario is that a certain target region will be predefined during the mission operations planning, such that the images captured will be of this area. Therefore, regarding this system design, it can be assumed that the general location of the images is known, and certain regions can be included or excluded. Restricting the areas or types of land classes over which the initial system implementation is expected to perform can have major benefits. For example, water, desert, and ice regions or steep terrain may be excluded to help ensure that the development can work with images that contain distinguishable features. This is a common challenge in computer vision, and is discussed further in Chapter 4.

#### *Imager Properties*

The system is developed using images captured by a multi-spectral type imager, since most datasets are available in the optical and near-infrared domain. The bands captured by multi-spectral cameras contain the standard red-green-blue bands, which for the development process will be most straightforward to compare with existing datasets. The system is not limited to these bands, and if it is proven to work with multi-spectral images, further development can test its performance with hyperspectral images.

#### *Image Pre-Processing*

Images captured on board current SotA small satellites typically undergo either very limited or no preprocessing. For standard georeferencing pipelines, substantial image correction steps are generally applied before the images are exposed to feature detection or similar processes [61]. Due to the existing challenges of designing a novel georeferencing system, it has been chosen to assume the input query images are equivalent to Sentinel Level 2A products. Future studies can increase the complexity of the system by exploring the possibility of utilizing raw images as input.

Sentinel Level 2A multi-spectral products are considered to be Bottom of Atmosphere (BoA) equivalent, and are already orthorectified. While BoA images is a gross simplification not representative of the raw data found on board and orthorectification is considered an important step in the georeferencing pipeline (see Chapter 2), this simplification has been made to ensure any errors or challenges encountered during the development of the system could be attributed to the system itself, and not to the input data. Furthermore, the architecture of the system is designed to be compatible with raw data, and therefore the system can be adapted to work with raw images in future work.

**Constraints**

CNSTR-1 *The query image shall be captured over a scheduled region.*
This limits the complexity of the system design, allowing a region to be selected ahead of time, and enabling the possibility of uploading reference data to the satellite prior to capture. This is considered a realistic requirement, since small satellites typically schedule their image captures anyway due to resource limitations.
(Verification by: analysis)

CNSTR-2 *The system shall be provided with an initial location estimate of the query image.*
The system is limited to the image processing pipeline and not the processing of Attitude Determination and Control System (ADCS) and other satellite sensor data. The spacecraft has knowledge of its location and attitude, and therefore inherently can provide an initial location estimate with each captured image. Pre-processing this sensor data is considered out-of-scope for this system.
(Verification by: analysis)

CNSTR-2.1 *The query image initial location estimate shall have an accuracy of at least one-half the image's smallest dimension*
This defines the accuracy of the initial estimate to be a function of the image size, making it verifiable with multiple datasets.
(Verification by: analysis)

CNSTR-3 *The system shall utilize multispectral earth observation data*
Basic specification that the system will be for optical earth observation data only. Specific requirements of textures and corrections are listed as children. Each child requirement helps define the scope of the system design, ensuring that standard image pre-processing steps will not be necessary in the system architecture.
(Verification by: test and analysis)

CNSTR-3.1 *The query image shall contain unique textures, such that no tiles from a single image result in identical feature sets*
This is in place to enforce that the input image will contain enough distinguishable features such that when cut into tiles for processing, no two tiles will be identical. This prevents images of deserts, ice caps, water bodies, etc; in which there are no distinguishable features.
(Verification by: analysis)

CNSTR-3.2 *The query image shall be pre-processed to an equivalent of Sentinel Level 2A data*
The system should not be responsible for the pre-processing of input data. Sentinel Level 2A is considered BoA - equivalent; preliminary development of this system should first function with this before being subjected to more complex data.
(Verification by: analysis)

CNSTR-3.3 *The query image shall not contain clouds*
Edges of clouds can produce false features which will be detrimental to any feature-based georeferencing methods.
(Verification by: analysis)

CNSTR-3.4 *The query image shall be corrected for motion-blur*
The imager is in motion during capture, thus motion blur can be an artifact detrimental to any computer vision processes.
(Verification by: analysis)

CNSTR-3.5 *The query image shall only be captured during the daytime*
Most optical imagers do not perform well during nighttime. Furthermore, daytime images contain many more features than with poor illumination.
(Verification by: analysis)

CNSTR-3.6 *The query image shall have a spatial resolution of at least 30m*
The value of 30m is based on the identified potential use cases.
(Verification by: analysis)

## 3.5. Preliminary Design

As established above, there exist two primary methods of georeferencing: direct and indirect. Currently, the vast majority of research in this field is focused on optimizing existing algorithms or presenting novel ideas to accelerate various sub-steps of the georeferencing pipeline. This is seen in new feature recognition methods or

accelerations in the processing architecture. Literature on fundamentally different georeferencing methods is very limited; thus these approaches will not be considered valid candidates for system design options.

The primary purpose of identifying these design options is to establish which general approach is best suited for this application. These demonstrate potential solutions, yet abstract from precise implementation details to enable effective comparison with the other options.

This process is explained in the following sections. First, the generation and evaluation methodology of the different design options is outlined. Next, the options are presented and explained in detail. Finally, the obvious problematic candidates are eliminated, and using weighted criteria the remaining designs are ranked and selected.

### 3.5.1. Methodology

Since the conceptual architecture of this system is restricted to existing fundamental georeferencing technologies and specific satellite computational resources, the identification of the different design options could be limited to only those meeting these requirements.

The possible georeferencing methods are:

- Direct
- Indirect
- Hybrid (combination of direct and indirect)

Regarding the actual implementation of the pipeline, the data can be processed either:

- *On board* the satellite with extremely limited resources
- On a *terrestrial* computer, which has access to relatively unlimited resources
- *Distributed* - via a combination of the onboard and terrestrial resources

#### IG Image Matching Options

Within the scope of indirect georeferencing, as mentioned in Section 2.3, the use of deep learning models adds the possibility to match images using PCA. In generating the potential high-level design options, it must be determined whether PCA should be considered a viable design option, or if all indirect georeferencing options imply that features must be extracted from the images. In the scope of systems' engineering, the following risk assessment is performed.

While reducing the captured image to an encoded representation has the potential to significantly reduce the bandwidth required to send the image to ground stations, the actual georeferencing process must be considered. A major disadvantage of using PCA for the lookup of reference images occurs when the QI contains relatively generic features, such as forests, grasslands, or other simple repetitive textures. These QIs will match a large set of *similar* reference images, since an exact match is not present in the reference database. Selecting the most similar image will then be subjective of how unique the QI is. Literature also shows that implementations of PCA with satellite images are typically in the use of reducing the dimensionality of extracted features, rather than the entire image [70]. This demonstrates that precise image matching still needs more exact data points than a single encoded representation.

Furthermore, this approach presents an additional challenge if the query image is of a different scale or overlaps multiple reference images; a scenario almost guaranteed in the use case of georeferencing, since likelihood that the captured image has the exact same location as the reference image is nearly zero. In this event, the PCA model will return multiple matches, which would need to be filtered and spatially aligned using another means.

In selecting realistic preliminary design options, the use of PCA is not considered further. Due to the challenges listed above, it was determined that the use of CBIR specifically for georeferencing requires dedicated research and development before being considered a viable option. Using GCPs from extracted features is an established method, and was therefore considered to significantly reduce risk for the overall system architecture.

#### Generation

Whereas designs are traditionally explored via a design option tree, these solutions were identified using the matrix as seen in Table 3.2. This ensured that all possible combinations of georeferencing methods and

processing platforms were taken into account. The design options were identified in increasing complexity, such that terrestrial processing and DG are considered to be the most straightforward implementations. Each design is labeled with an alphanumeric ID, where the letter indicates the type/location combination and the numeral indicates the sub-design, which was needed in some cases to identify alternative solutions.

**Table 3.2:** Design options discovery matrix

| | | Processing Location | | |
|---|---|---|---|---|
| | | **Terrestrial** | **Onboard** | **Distributed** |
| **Georeferencing Type** | **Direct** | A | D | Z |
| | **Indirect** | B | E | G |
| | **Hybrid** | C | F | H |

***Note on design option details:***

The level of detail present in these design options is purely to establish and exemplify that the solution is realistic and potentially feasible. Variations, improvements, and alternatives within each option are certainly possible, but will only be considered at a later design phase. The scoring criteria are also established in Table 3.5.3 to evaluate the overall combination of georeferencing type and processing location relative to other options, rather than the specific implementation. The exact implementation should not directly impact the design option score, thus exact values for accuracy and performance are avoided.

The generation of design options followed an iterative process. Starting with the combinations defined by the matrix (Table 3.2), the base options were created ("P-A1, P-B1, P-C1", etc.). The major processing steps to be performed on board and/or on terrestrial hardware were listed, as well as the necessary downlink data.

#### Evaluation

Next, each option was evaluated for critical issues. This process differed from the ranking used for the trade-off; it aimed to identify major issues and determine if a plausible alternate solution exists. The following aspects were evaluated for each design option:

- General accuracy of the georeferencing method used
- On-demand capability
- Onboard resource usage

If a major issue was identified (such as significantly low accuracy), an alternative needed to be identified that still remained within the same georeferencing type. This could either be an existing design option or a new sub-design. For example, onboard DG in its simplest form does not include orthorectification. However, the resulting accuracy is significantly lower than that of industry standard, and in cases with large terrain variations is even unusable. A viable solution, which still remains in the same category, is to include orthorectification on board. This created an alternate design option "P-D2". This new design option was then evaluated in the same way.

As explained above, defining an alternate design was driven by a need for performance improvement, and to affirm whether that type/platform combination (i.e. "direct" + "onboard" had at least one viable solution for the overall system).

Note: All descriptions about "processing" regard the georeferencing pipeline only. Raw image filtering, corrections, compression, storage and other standard image processing steps fall outside the scope of this design.

### 3.5.2. Proposed Design Options

This iterative process produced a total of 11 design options, seen in Table 3.3. The following sections describe each in more detail, and provide information about its advantages and challenges.

#### Terrestrial Solutions

***Option P-A1 - (direct)***

This first option is one of the most simple, where the exterior and interior orientation parameters are downlinked directly, along with the Point of Interest (POI) image coordinates. This works over the limited Internet of Things

**Table 3.3:** Preliminary design options

| Option ID | Georeference Type | Processing Platform | Onboard Processing | Downlinked Data | Terrestrial Processing |
|---|---|---|---|---|---|
| P-A1 | Direct | Terrestrial | None | • POI pixel coords<br>• RPCs (EO + IO parameters) | Full DG pipeline<br>• with orthorectification |
| P-B1 | Indirect | Terrestrial | None | Full image | Full IG pipeline |
| P-C1 | Hybrid | Terrestrial | None | • Full image<br>• RPCs (EO + IO parameters) | • DG initial location estimate<br>• Select RI based on DG estimate<br>• Full IG pipeline |
| P-D1 | Direct | Onboard | Full DG pipeline<br>• no orthorectification | Final Result | None |
| P-D2 | Direct | Onboard | Full DG pipeline<br>• with orthorectification | Final Result | None |
| P-E1 | Indirect | Onboard | Full IG pipeline | Final Result | None |
| P-F1 | Hybrid | Onboard | • DG initial location estimate<br>• Select RI based on DG estimate<br>• Full IG pipeline | Final Result | None |
| P-G1 | Indirect | Distributed | • Identify GCPs<br>• Crop QI to min required GCPs | Cropped Query Image (QI) | • Lookup RI (CBIR)<br>• Full IG pipeline<br>- (with RI and Cropped QI) |
| P-H1 | Hybrid | Distributed | • Initial DG location estimate<br>• Extract GCPs (algorithm tbd) | • Initial location estimate<br>• POI pixel coords<br>• GCPs pixel coords | • Lookup RI based on DG estimate<br>• Full IG pipeline<br>- (with RI and GCP pixel coords) |
| P-H2 | Hybrid | Distributed | • Initial DG location estimate<br>• Extract feature vectors using CNN | • Initial location estimate<br>• GCP Feature vectors | • Lookup RI based on DG estimate<br>• Find matching tie-points for each QI feature vector<br>• Perform IG with all tie points |
| P-Z | Direct | Distributed | n/a | n/a | n/a |

(IoT) network, and the DG pipeline is executed on the ground. Using orthorectification the accuracy is not exceptionally high, but can be used for certain applications. No dedicated hardware is needed on board for this option.

#### *Option P-B1 - (indirect)*

This reflects of how most satellite image datasets are georeferenced. On board the satellite, no processing is required. The full image is sent to the ground station, where all georeferencing steps are performed using the IG process.

This requires little, to no dedicated hardware on board the satellite, and superior computational resources at the ground station can be leveraged for high-accuracy image-based georeferencing. A major drawback is that the necessary volume of data to be transmitted to the ground station is a maximum, relative to the other design options

#### *Option P-C1 - (hybrid)*

Combining the options P-A1 and P-B1, DG and IG can be used together to optimize the data pipeline. Considering the data volume of P-A1 (POI coordinates and RPCs) is considerably lower than the full image, including this in the downlink is a minimal increase. Executing DG on the ground gives and initial location estimate, which can assist in selecting the reference image.

### Onboard Solutions

In contrast, removing the need to transmit any form of image data to earth can significantly reduce the required downlink bandwidth. This can be accomplished by executing the full georeferencing pipeline on board the satellite. The result is that only the final result (the geographic coordinates of the point of interest) are transmitted.

Running the complete pipeline on board the satellite can be done via two approaches: using DG or IG.

#### *Option P-D1 - (direct)*

Full DG on board the satellite requires relatively little extra resources. The external orientation parameters are measured on board, and can be directly used in the Universal Sensor Model (USM) to perform the transformation. This design option is optimized to utilize minimum computational resources, thus the additional orthorectification step is omitted. Including this step would require storage of an extensive DSM, as well as extra computation for the OBC.

The resulting geo-localization accuracy of the system would be minimum, compared to other design options. This is due to the inherent lesser accuracy of DG as explained in Section 2.2, and the lack of orthorectification.

#### *Option P-D2 - (direct)*

Identified in the evaluation cycle, P-D1 has significantly substandard accuracy results. This alternative improves the accuracy by including orthorectification in the processing pipeline. While theoretically possible to run on board a satellite, the storage requirements of the necessary DSM exceed the capabilities of a small satellite.

#### *Option P-E1 - (indirect)*

Similarly, executing the full IG pipeline on board the satellite can enable sending only the final coordinates to the ground station. As explained in Section 2.3, this process relies on access to an extensive reference image library, which must be stored on board. Additionally, the generation of tie-points must be fully automated, requiring CV processing resources. If accomplished using Machine Learning (ML), dedicated hardware is likely necessary. Otherwise, traditional CV algorithms may be run on the OBC, although the computational requirements likely exceed the resources available.

The advantage of this approach is that the resulting geolocalization accuracy of the end result is likely much higher than from Option B, using DG. See Section 2.3.5. If the onboard resource requirements can be met, this solution could be the most desirable as it requires minimum complexity. Leveraging SotA ML algorithms, the model could be trained using terrestrial computational resources, only requiring inference to be run on board. Data transmission would be possible via an IoT network, meeting the goal to produce on-demand results.

#### *Option P-F1 - (hybrid)*

A combination of P-D1 and P-E1 could be executed on board. As with the terrestrial hybrid option (P-C1), first running the DG pipeline can provide an initial estimate at relatively little computational cost. This could significantly improve the IG process, as the reference image can trivially be selected. However, the accuracy will remain the same as option P-E1, and the onboard resource usage will still be high.

### Distributed Solutions

The final category divides the georeferencing pipeline into onboard and terrestrial segments. This allows certain pre-processing steps to be performed on board the spacecraft, while resource-intensive processes can be run on terrestrial hardware.

#### *Option P-Z - (direct)*

This is the only option directly not feasible. Direct georeferencing is a singular process which therefore cannot be distributed over multiple computational platforms. Thus, this design option was merely included for completeness, and is not explored further.

#### *Option P-G1 - (indirect)*

As Options D, E, and F aim to improve on Option P-B1 by removing the need to downlink full images, this solution aims to resolve the problem of requiring onboard storage of large reference datasets. Considering the IG process (Section 2.3), if only a limited number of GCPs are required, this option proposes to perform GCP identification on board the satellite and only send patches of the image containing these features to the ground station.

The image coordinates of the POI in question would be included in the downlink. Terrestrial processing would complete the IG pipeline; identifying the reference image, extracting reference GCPs, creating GCP tie-point pairs, and transforming the POI to geographic coordinates.

***Option P-H1 - (hybrid)***

In the event that the downlink bandwidth is too small for option P-G1, the transmitted data can be further reduced. Following the same initial process of extracting GCPs on board, this approach differs in that image-based IG is used to *improve* the geo-localization accuracy of an initial DG solution. Transmitted to the ground station are: the pixel coordinates of the POI, the RPCs produced by the initial DG, and the pixel coordinates of the GCPs but *without* the patched image features. This will result in a significantly smaller data package.

The exact feasibility and robustness of this process still needs to be investigated, as it is uncertain whether GCP-pairs can be established based purely on the relative spatial positions of the query image GCP (and no actual feature data). Normally, these GCP-pairs are determined by comparing the similarity of features.

***Option P-H2 - (hybrid)***

Addressing the concerns of establishing GCP-pairs in Option P-H1, an intermediate option between sending full image patches and no feature information should be considered. Since these image patches are ultimately processed to extract a "feature vector" that contains a measure of the respective feature, which can be compared in a similarity measure to other features, an alternative is to extract this vector on board the spacecraft. This feature vector will be smaller in terms of data storage than the raw (multispectral) image source. Since no image data is provided, an initial estimate of the geolocation must also be included, to allow the terrestrial processing segment to select the appropriate reference dataset.

Transmitted to the ground station are: the GCPs pixel coordinates and corresponding feature vector, POI pixel coordinates, and the initial DG location estimate.

This solution is based on the work done by Merkle [38], which demonstrates the process of producing these feature vectors through the use of a Siamese Neural Network (SNN). This approach is proven to be robust to multimodal image data, which includes reference data that may have a different scale and rotation than the query image. The feature vectors produced on board the spacecraft will be generated using a Neural Network (NN) that share parameters with the terrestrial NN analyzing the reference dataset. This ensures the resulting feature vectors will be comparable, thus resulting in a robust set of generated GCP tie-points.

The exact data size of these feature vectors needs to be further investigated. This will require an in-depth analysis of the generated satellite imagery, as well as a prototype of the SNN-based georeferencing pipeline.

### 3.5.3. Preliminary Trade-off

The preliminary trade-off performed here established the general approach to be used. This eliminated options that are infeasible, and identified those which progress to the detailed design phase.

#### Direct Elimination

The first step was to eliminate design options not meeting minimum requirements. Referring back to the evaluation step during the option generation, the three critical criteria are assessed: final accuracy, on-demand capability, and onboard resource usage. All design options directly failing these were eliminated, as described in Table 3.4.

**Table 3.4:** Elimination of design options

| Failed Criteria | Eliminated Option | Reasoning |
|---|---|---|
| Final Accuracy | P-D1 | Direct georeferencing without orthorectification results in sub-standard accurcies, especially in regions of high terrain variability |
| On-Demand Capability | P-B1 | Requires full image to be downlinked, not possible on limited IoT network |
| | P-C1 | Requires full image to be downlinked, not possible on limited IoT network |
| Onboard Resource Usage | P-D2 | Storage of DSM too large for satellite |
| | P-E1 | Storage of reference image database too large for satellite |
| | P-F1 | Storage of reference image database too large for satellite |

### Ranking Criteria

The four remaining design options were ranked using a set of weighted criteria. Three main categories were defined, each with a scoring scale of 0 to 6.

The weights were defined by comparing the relative importance of each. Roughly following the Analytical Hierarchy Process (AHP) framework, the criteria are compared to each other using the classical 1 - 9 scale [22]. A value of 2 indicates that one element has slightly more importance over another. Since the three criteria are nearly equally important (at this design stage) a comparison value of max 2 was used, resulting in the following pairwise comparison matrix was formed (Table 3.5):

**Table 3.5:** Criteria pairwise comparison

| Pairwise comparison | On-Demand Capable | Global Coverage | Maturity Level |
|---|---|---|---|
| On-Demand Capable | 1 | 1/2 | 1/2 |
| Global Coverage | 2 | 1 | 2 |
| Concept Maturity | 2 | 1/2 | 1 |

The table can be read as follows: [*row*] is [*value*] more important than [*column*]. For example: "Global Coverage" is "2" (slightly) more important than "On-Demand Capable".

Using the AHP eigenvector method described in Goepel [22], the weights of each criterion are computed. This results in a consistency ratio of 5.6%. Being below the standard threshold of 10% indicates the pairwise comparison is sound. The criteria are listed below, with the weights in parentheses:

- On-demand Capability (19.6%)
- Global Coverage (49.3%)
- Maturity Level (31.1%)

#### *On-Demand Capability*

This evaluates the likelihood whether the overall system could return results in real-time. High volume data downlinks were the typical bottleneck of this criterion, where full images are definitely not supported, thus resulting in a score of 0. The smallest data packets consist only of EO and IO metadata, therefore being able to be transmitted over the IoT link and resulting in a score of 6. The scores are defined below in Figure 3.3:

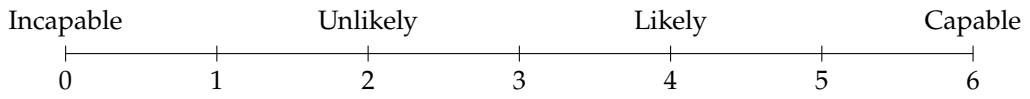



**Figure 3.3:** Trade-off scoring scale: on-demand capability

#### *Global Coverage*

This is an important combination of how effective the system is in operability around the world. Direct georeferencing is applicable irrespective of the ground surface, but suffers in terms of final accuracy. Indirect georeferencing on the other hand has a much higher accuracy, but is constrained to surface conditions that contain unique features. As such, indirect georeferencing does not function over water bodies. Since Earth's surface is roughly 70% water, this results in the latter only being applicable to 30% of the globe. Naturally, a combination of the two georeferencing methods would be ideal, maximizing accuracy and coverage. The scoring scale can be seen in Figure 3.4.

While simplifications regarding the operational region are defined in Section 3.4, this criterion is evaluated based on the general applicability of the system. The reasoning behind this is that the fundamental design choices should reflect the best possible architecture, and not be constrained by simplifications that are only applicable to indirect georeferencing. Since the design is independent of a specific mission or use case, general applicability is a priority that is supported by a system with global coverage.

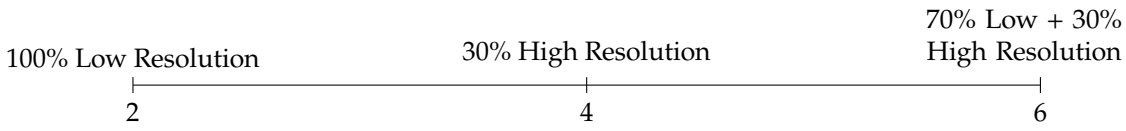
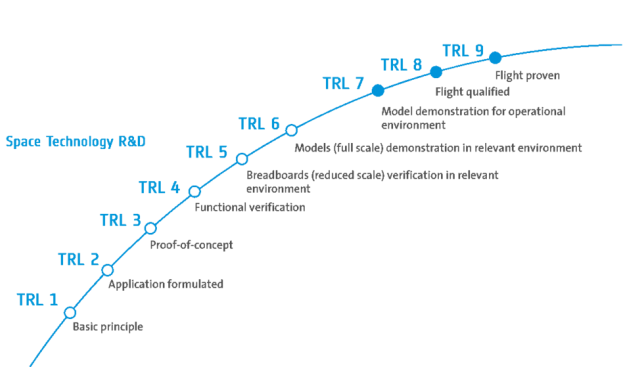
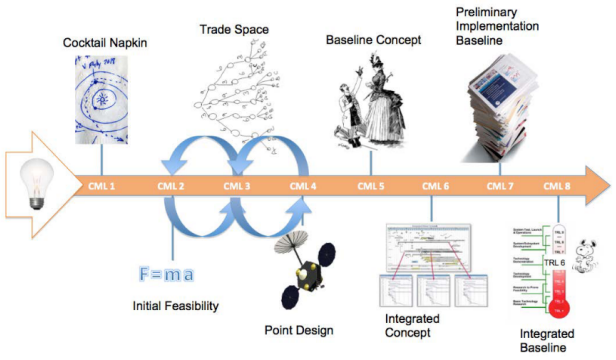



**Figure 3.4:** Trade-off scoring scale: on-demand capability

### *Concept Maturity*




**(a)** Technology Readiness Levels (TRLs) as defined by European Space Agency (ESA) [17]




**(b)** Concept Maturity Levels (CMLs) as defined by Jet Propulsion Laboratory (JPL) [63]

**Figure 3.5:** Common technology maturity scales

Determining the maturity level of each design option is performed using a customized maturity scale. Based on a combination of Technology Readiness Levels (TRLs) defined by ESA (Figure 3.5a) and Concept Maturity Levels (CMLs) by JPL (Figure 3.5b), the maturity levels evaluate the combination of key underlying technologies and the overall system integration. For example, one design option may rely on direct georeferencing which has decades of flight heritage, but its application in a system with onboard indirect georeferencing may still be considered novel. Each ranking also has explanation for support. These maturity levels are summarized in Table 3.6, and defined in Table 3.7.

**Table 3.6:** Maturity levels

| | | System Integration | | |
|---|---|---|---|---|
| | | **Concept** | **Verified** | **Validated** |
| **Key Tech** | **Concept** | 1 | - | - |
| | **Verified** | 2 | 4 | - |
| | **Validated** | 3 | 5 | 6 |

**Table 3.7:** Maturity terminology

| Stage | Definition |
|---|---|
| Concept | Concept is worked out to asses feasibilty, in terms of technical and programmatic aspects |
| Verified | Tested and verified in a relevant environment |
| Validated | Flight-proven |

### Option Trade-off

The ranking of the remaining design options can be seen in Table 3.8. The total score in the rightmost column is the sum of the individual scores multiplied by the criterion's weight. The valuation for each option is based on logical analysis of the process.

Regarding the on-demand capability, a score of 6 is assigned to P-H1 and P-A2 because the data generated prior to downlink is minimal, and will therefore meet the bandwidth requirements. Option P-H2 produces feature vectors, of which the data size is unknown. This will require additional investigation, but it is estimated that this can also be reduced to an acceptable level; therefore receiving a score of "highly likely". Pure distributed indirect

georeferencing (G1) requires a downlink of a resized image; which almost certainly exceeds the bandwidth constraints.

Global coverage is fully dependent on the georeferencing methods employed. Purely using direct georeferencing results in 100% coverage, but with a "Low" final accuracy; thus a low score of 2. Purely relying on indirect georeferencing does not provide global coverage, but has a higher final accuracy. A combination of DG and IG results in the best solution possible.

Concept maturity is evaluated on a case-by-case basis. Option P-H2 uses georeferencing technologies that have been verified in literature, but the complete system has not yet been developed. Option P-H1 on the other hand uses a conceptual georeferencing approach (relying only on GCP image coordinates), thus receiving a lower score. Direct georeferencing on terrestrial resources is a process that has been validated by heritage. However, no evidence of its use in a real-time system has been found, thus the system is considered to have a maturity of "verified". Finally, option G1 also relies on validated georeferencing technologies, but the process of reducing images to only contain the GCPs and transmitting these over the real-time network is purely conceptual.

**Table 3.8:** Trade-off of preliminary design options

| Option ID | Type | Georef Platform | On-Demand Capable | Weight 19.6% | Global Coverage | Weight 49.3% | Concept Maturity | Weight 31.1% | Total |
|---|---|---|---|---|---|---|---|---|---|
| P-H2 | Hybrid | Distributed | Highly Likely | 5 | 30% - High<br>70% - Low | 6 | Tech: verified<br>Sys: concept | 2 | 4.560 |
| P-H1 | Hybrid | Distributed | Yes | 6 | 30% - High<br>70% - Low | 6 | Tech: concept<br>Sys: concept | 1 | 4.445 |
| P-A1 | Direct | Terrestrial | Yes | 6 | 100% - Low | 2 | Tech: validated<br>Sys: verified | 5 | 3.717 |
| P-G1 | Indirect | Distributed | Unlikely | 2 | 30% - High | 4 | Tech: validated<br>Sys: concept | 3 | 3.297 |

It's clear that the "hybrid - distributed" solutions ("P-H1" and "P-H2") significantly outrank the other design options. This is a logical outcome, considering it combines the best properties of both direct and indirect georeferencing, and distributes the processing location to minimize the amount of data necessary to be downlinked from the satellite.

## 3.6. Detailed Architecture Design

Since the preliminary trade-off determined options "P-H2" and P-H1, both hybrid architectures, had the highest potential for success, the next step in the system design process was to determine the exact architecture to be applied. Combining a functional flow of the georeferencing process and the preliminary results, the different possible design options were generated. These were then evaluated and ranked based on their feasibility and criteria resulting from the specified system requirements.

### 3.6.1. Functional Flow

The first step to generating potential architectures is to define the functional flow of the system. This is a high-level functional flow diagram of the main processes that must take place, independent of processing location, depicted in Figure 3.6.

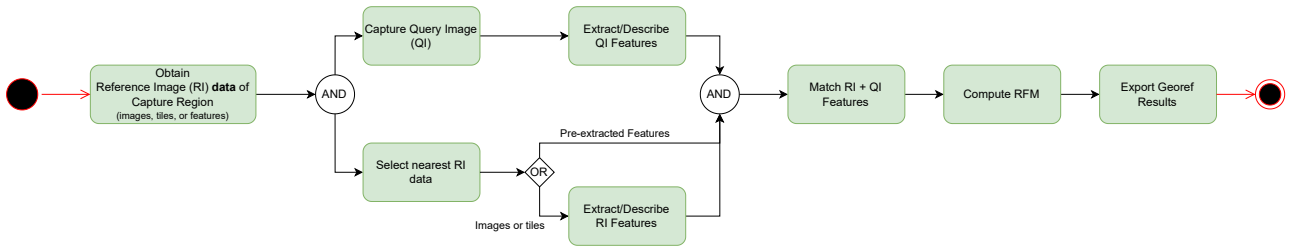



**Figure 3.6:** Functional flow diagram of system[5]

[5]See Appendix A for an enlarged version.

The diagram is based on the basic elements of an indirect georeferencing pipeline. To clarify that the QI processing can be done either before, in parallel, or after the processing of the reference image, "and" blocks are created to separate these sub-processes. Additionally, the option that the reference features have been extracted in advance from the reference image, an "or" block is included following the "select nearest RI data" element. Table 3.9 provides a description of each element in the functional flow diagram.

**Table 3.9:** Functional flow diagram element descriptions

| Element | Description |
|---|---|
| Obtain Reference Image Data of Capture Region | Acquisition of reference images and/or feature descriptors with geospatial information that can be used for IG. Must be of the region of that the spacecraft is scheduled to capture with the imager. |
| Capture Query Image | Image capture by the spacecraft of a geographic area within the scheduled region. Produces the (non-georeferenced) Query Image (QI). |
| Extract/Describe QI Features | Run the feature extraction and description algorithm on the query image to obtain a set of features. |
| Select Nearest RI Data | Select a subset of the reference data (images or features) which is known to spatially overlap the region of the query image. |
| Extract/Describe RI Features | If the reference dataset contains images, this element represents the extraction of keypoints from this dataset. |
| Match RI + QI Features | Match the reference and query feature descriptors to find corresponding sets of ground control points (GCPs). |
| Compute RFM | Using the sets of GCPs, compute the coefficients of the polynomial transformation model. |
| Export Georef Results | Prepare the results for transmission off-platform. I.e. saving the coefficients to a file or computing the geospatial coordinates of a point of interest identified in the image. |

Below are some additional justifications regarding the order of the elements in the flow diagram:

- *Reference Image Data* can be obtained prior to image capture, because the region of capture is known beforehand. See constraint CNSTR-1.
- Selection of RI data is dependent on the captured query image, because the QI provides an initial location estimate.
- *Extract/describe RI Features* comes after the selection of nearest RI data, because the feature extraction should only be done on the relevant (nearest) RI data. If the option to extract features from a full RI image dataset prior to selecting nearest RI data is desired, then a selection of nearest RI features needs to occur subsequently. This option is then functionally equivalent to starting with the first element *Reference Image data* in the form of obtaining features rather than full images.
- *Match features* is a separate element from *Compute RFM* because the processes are separate functions and depending on the computational load may not be able to be preformed on the same resource.
- *Export Georef Results* is its own functional block because it marks the output interface of the georeferencing pipeline. This for example takes care of formatting the results to be compatible with the ISL or the off-platform receiving system. See requirement PERF-1.

## 3.6.2. Design Options

Since each systems' engineering process is unique to its application, there is no standard method for generating design options. Rather than using a design option tree, the process applied here is based on the "Logical Decomposition Models" described by Hirshorn [24]. This method is used to generate all possible architectures based on the functional flow diagram. The process is described below.

### Constraints

Starting with the system requirements, a number of constraints to which each potential architecture must adhere to was established. These originate from the system requirements and general logic. These are presented in Table 3.10.

**Table 3.10:** Constraints regarding functional flow diagram when generating architectures

| | Constraint | Justification |
|---|---|---|
| 1. | Element "Capture Query Image" must be on board. | By simple logic: the image to be georeferenced is by definition the one captured by the satellite |
| 2. | Element "Obtain RI Data of Capture Region" must be on ground. | Reference data will be sourced from terrestrial databases |
| 3. | Element "Export Georef Result" must be off-platform | See requirement FUNC-3. |
| 4. | No data uplink after an ISL downlink | See requirement PERF-1. Data should not be transmitted back and forth with the ground station during the processing. The ISL is slow, thus this adds significant time delay. If a direct downlink is available, onboard georeferencing is unnecessary, since the QI can then also directly be sent to the ground for processing |
| 5. | No data uplink after "Capture Query Image" | System requirement: real-time capability of this system. Data uplink requires precise knowledge of when the image is taken, and the system will not be in communication mode during image capture |

### Discovery

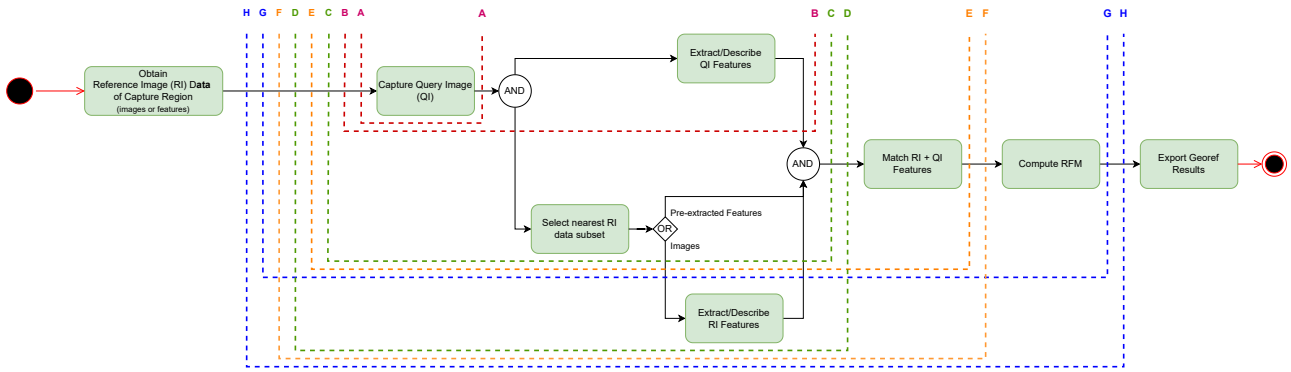



**Figure 3.7:** Discovery of all possible system architectures from functional flow.[6]

The architectures are formulated by drawing a segmentation line "- - -" through the flow diagram to divide the functional blocks between *ground* and *onboard* resources. Starting with the most simple architecture, (capturing the image onboard and sending the full image to the ground for georeferencing), functional blocks are progressively added while satisfying the constraints defined above. This results in the diagram depicted in Figure 3.7, with the options described in Table 3.11.

Note: An intersection of an arrow with the dotted line does not necessarily signify a data transfer. The arrows are indications of sequence. Also, the parallel actions between the AND-blocks can be asynchronous or synchronous in either order.

## 3.6.3. Direct Elimitations

Inspecting architectures based on functional coherence and relative value leads to some direct eliminations. Important to note, architectures are not eliminated here due to technical deficiencies (such as data packet too large for ISL). Elimination based purely on functionality whether the option has advantages over the others. The directly eliminated options and their reasons are listed in Table 3.12.

## 3.6.4. Technical Evaluations and Elminations

The six remaining architectures (A, B, E, F, G, and H) were converted to formal design options by appending a numeral to the ID. This process served as an initial feasibility study, with the goal to narrow the choices in order to select the most promising architectures for a final ranking. The two most critical factors considered in this process are the onboard resources required (computational, storage, etc.), and the data volume required to be transmitted to the ground station. Concerning the options in which only parts of the image are transmitted,

[6]See Appendix A for an enlarged version.

**Table 3.11:** All combinations of system architectures created from functional flow

| Architecture Design Option | Data to Uplink | Onboard Processing Steps | Data to Downlink for Terrestrial Processing |
|---|---|---|---|
| A1 | None | None | • QI Image |
| B1 | None | • Extract QI Features | • QI Features<br>• QI Location Estimate |
| C | • Ref. Features | • Extract QI Features<br>• Select RI Features Subset | • QI Features<br>• RI Features subset |
| D | • Ref. Images | • Extract QI Features<br>• Select RI Image Subset<br>• Extract RI Features | • QI Features<br>• Ref. Features subset |
| E1 | • Ref. Features | • Extract QI Features<br>• Select RI Features Subset<br>• Match RI + QI Features | • RI + QI Feature Pairs |
| F1 | • Ref. Images | • Extract QI Features<br>• Select RI Images Subset<br>• Extract RI Features<br>• Match RI + QI Features | • Ref. + QI Feature Pairs |
| G1 | • Ref. Features | • Extract QI Features<br>• Select RI Features Subset<br>• Match RI + QI Features<br>• Compute RFM | None |
| H1 | • Ref. Images | • Extract QI Features<br>• Select RI Images Subset<br>• Extract RI Features<br>• Match RI + QI Features<br>• Compute RFM | None |

**Table 3.12:** Directly eliminated architectures

| Architecture | Elimination Justification |
|---|---|
| C | • Only computationally more expensive than B.<br>• Selecting features onboard has no benefit if they only need to be downlinked |
| D | • Even more computationally expensive than C due to additional reference feature extraction.<br>• Selecting features onboard has no benefit if they only need to be downlinked |

the data volume is estimated to be approximately 50 features extracted from an image. This value is selected because an absolute minimum of 10 ground control points are required to create a third-order georeferencing polynomial, but studies have shown that a higher number of GCPs results in a more accurate model [33]. The quantity 50 is selected as a reasonable buffer.

The options eliminated due to technical challenges are described below.

### Option A1

This option requires the absolute bare minimum onboard resource usage, since the image is captured and transmitted without any georeferencing processes. Full images, even with exceptional compression, are too large to be transmitted via the ISL and will not meet requirement FUNC-3.1. Any preprocessing steps such as cropping or feature extraction fall under design option B1. This option is thus eliminated.

### Option F1

This option implements the feature extraction steps onboard, for both the captured QI and the reference images. The features are matched onboard, and the corresponding sets of GCPs are transmitted via an ISL to terrestrial resources to compute the RFM. While it's likely that this option would meet the requirements, it's functionally nearly identical option E1. The difference lies in that Option E1 works with pre-extracted reference features, which require significantly less onboard storage and do not need to be extracted using the limited onboard resources. Therefore, computationally inferior F1 is eliminated.

#### Option H1

This option is conceptually the most simple: the entire georeferencing pipeline is executed onboard the satellite. However, as determined in the analysis of Option F1, extracting features from the reference image onboard is an avoidable step that can also be done on terrestrial resources. In light of requirement Option G1 is thus a more efficient version of this option, and H1 is eliminated.

### 3.6.5. Ranking

Table 3.13 list the three criteria used to rank the remaining design options (B1, E1, G1). These are based on the overall goal to create a system with optimal performance that still fits within the constraints of a small satellite. The criteria are defined as follows:

**Table 3.13:** Design option ranking criteria

| Criteria | Scoring | Description |
|---|---|---|
| System Flexibility | 1 = Highly dependent<br>3 = Independent | System independence of platform constraints, such that it can be used by other missions for different applications |
| Innovation Potential | 1 = Little potential<br>3 = High potential | The potential for the system to be improved with more research outside the scope of this thesis. For example:<br>- to overcome simplifications of the current scope<br>- to make use of evolving supporting technologies in the near-future |
| Onboard Resource Compatibility | 1 = Less compatible<br>3 = More compatible | How suitable the proposed design is concerning the limited onboard resources of a small satellite, relative to the other design options. |

A traditional trade-off was not considered to be the most appropriate approach, because options B1, E1, and G1 are all variations on the same process. A trade-off would have been useful in the case where distinctly different processes were present between the design options, or in cases where selecting one option resulted in the forfeit of advantages present in another. As a result, the design options here are ranked in accordance to the criteria in Table 3.13.

**Table 3.14:** Final design option ranking

| Design Option | System Flexibility | Innovation Potential | Resource Compatibility | Total |
|---|---|---|---|---|
| B1 | 1 | 1 | 3 | 5 |
| E1 | 2 | 2 | 2 | 6 |
| G1 | 3 | 3 | 1 | 7 |

Reading the table such that higher scores are better, it can be concluded that the results are only logical:

- B1 has the least processing onboard but is dependent on terrestrial resources. Sending feature descriptors via the ISL is likely most difficult, due to the relatively large data packet size. Not being flexible is one of the greatest limiting factors
- E1 is a middle option. The ISL data packet size is significantly reduced, as it only contains coordinate pairs. This however comes at the cost of increased onboard computations. It is still slightly dependent on terrestrial resources to compute the RPCs of the final model
- G1 in terms of flexibility is the ideal choice. However, the feasibility of implementing this pipeline with models compatible with onboard resources requires further research.

Option G1 was thus selected as the optimal system architecture, with the understanding that its feasibility needed verification. This was conducted by implementing a proof-of-concept of the system, described in the next chapter. Had the results of the proof-of-concept been negative, then this ranking would directly indicate the next best option to implement. Section III of Chapter 4 provides an overview of the resulting system architecture, followed by an explanation of its implementation and experimental evaluation.

# 4

# Research Paper


## ABSTRACT

*While image-based georeferencing systems are widely available, none are designed for edge computing on board small satellites in space. Recent advances in autonomous data processing allow points of interest (POIs) to be identified in the captured images, enabling prioritization of data and reducing required downlink bandwidths. Directly attributing geopositional information to these POIs distills the generated insights to labeled locations, eliminating the need for images to be transferred off the satellite platform. Current georeferencing pipelines rely on databases of reference images and substantial computational resources to extract and match keypoints; an operation not feasible with the raw data and hardware limitations on board low-powered satellites. This work proposes a novel architecture which addresses these challenges. By pre-extracting reference keypoints using terrestrial resources, minimal processing on board is reserved for the captured images. Using a modular system architecture, the design is generalized for various small satellite platforms. A proof-of-concept analytical model is implemented based on D2-Net. This is evaluated on a variety of Sentinel-2A datasets, and the resulting state-of-the-art spatial accuracy confirms the feasibility and significant potential of this architecture. Finally, the limitations of this implementation are explored, from which specific recommendations for future improvements are proposed.*



***Keywords:*** *georeferencing; geospatial analysis; small satellites; edge computing; multispectral imagery; feature detection; convolutional neural networks (CNNs); D2-Net; RANSAC*


## I. INTRODUCTION

The use cases of satellite imagery continue to rapidly expand, driven by the increasing computational capabilities of small satellites and their capacity to extract insights in real-time. This information can be used to improve the efficiency of data analysis, from agriculture to disaster response. However, the value of this information is limited if it cannot be accurately georeferenced.

While there are many image-based georeferencing systems available, none are currently designed to operate on edge platforms. The limited computational resources of a small satellites and the challenges of raw satellite imagery make it difficult to implement such a system. Furthermore, obtaining the processed results from the satellite at any moment is also difficult, typically relying on an Inter-Satellite-Link (ISL) with serious bandwidth limitations which can relay the data to ground stations. However, the benefits of such a system are numerous, and would enable a wide range of new applications. For instance, disaster detection and response systems for events such as wildfires, floods, or earthquakes would be able to provide real-time information to first responders, enabling them to quickly and effectively allocate their resources.

This research proposes a novel system architecture that can overcome these challenges. The system is designed from a high-level system's perspective, ensuring the result takes into account all factors in the satellite image processing pipeline; from image capture in space to delivery of the georeferenced product. Indirect Georeferencing (IG) is selected as the most suitable approach, because it eliminates the dependency on physical parameters specific to the satellite platform and other errors due to sensor measurements.

The novelty of this system lies in its distribution of data processing between terrestrial and onboard resources. An essential step of georeferencing an image is identifying keypoints and matching these to known locations in a reference dataset. This process, which typically requires both the captured Query Image (QI) and a Reference Image (RI) to be present, is now optimized such that the keypoints are pre-extracted from the RI using terrestrial processing resources and uploaded to the satellite prior to image capture. On board, a keypoint detection and description method only needs to process the QI, and the resulting keypoints are then matched to the pre-extracted keypoints from the RI. Naturally, this implies the keypoint model must be

robust to variability in input data, since corresponding keypoints should be extracted from both the RI and QI. The selection of this model is therefore critical to the success of the system, and is covered in detail in the following sections.

Using a modular architecture, the system is designed such that its precise implementation can seamlessly be updated depending on the latest developments in the respective fields. The work presented here summarizes the proposed architecture, and implements a proof-of-concept to demonstrate the feasibility of the system. The system is evaluated on a variety of datasets, and the results are compared to existing State-of-the-Art (SotA) georeferencing products to compare its performance and evaluate its potential. Additionally, the limitations of this implementation are explored, from which recommendations for future work are proposed.

The rest of the paper is organized as follows. First, related work supporting both the overall system architecture and elements used in the proof-of-concept is presented. Next, the architecture and methodology of the system is explained in detail. The implementation is then systematically tested within the defined scope, in order to quantify its overall feasibility. Finally, the results are discussed and analyzed in depth to understand the current strengths and limitations of the system, and what is required to continue this work to obtain a deployable system.

## II. RELATED WORK

Since this research presents both a novel system architecture and a first-order implementation, this section aims to establish context for both categories. First, a number of previous studies regarding onboard georeferencing systems are presented, demonstrating the limited research available in this field. Second, the state-of-the-art in keypoint extraction and matching is presented, as this is a critical component of the proposed system.

### 2.1. Onboard Georeferencing Systems

At the time of writing, no full end-to-end onboard georeferencing systems could be found. However, a few studies have been performed which either indicate the need for such a system, or are performing research on foundational technologies such as Field Programmable Gate Arrays (FPGAs) that could be used for georeferencing. However, these studies do not propose a complete operational architecture, and are limited to a single component of the system.

Kothandhapani and Vatsal [17] directly evaluate the use cases and benefits of onboard autonomy for remote sensing applications. They establish that two stages are essential for onboard image processing: cloud segmentation and georeferencing. While the study clearly identifies the needs and challenges, such as computational restrictions, limited downlink bandwidth, and ultimately availability of reference Ground Control Points (GCPs), it does not propose any potential solutions or indicate ongoing related research.

#### 2.1.1. *FPGA Implementation*

Zhou *et al.* [36] introduced a novel method in 2017 to perform indirect georeferencing using a FPGA, and compared its computational performance with a traditional personal computer (PC)-based platform. Building on this, Liu *et al.* [23] proposes an FPGA-based automatic onboard georeferencing system. This approach aimed to optimize the design to minimize resource usage and overall latency, in order to demonstrate it could potentially be used in a real-time application. FPGAs are a logical choice for this experiment, as they are essentially Application-Specific Integrated Circuits (ASICs) which can be reconfigured in the field to be used for multiple applications.

A following publication by the same authors, addresses two critical steps in the automatic georeferencing pipeline: "reducing the computational complexity of matching feature points without losing precision" and "improving the processing speed using dedicated hardware"[22, p.2]. The GCP extraction is performed using Speeded-Up Robust Feature (SURF) for feature detection and the Binary Robust Independent Elementary Features (BRIEF) descriptor and matcher. Hardware optimizations such as parallel adders are used to further accelerate the processing. The results demonstrate that on this specific hardware platform, the FPGA can reliably and repeatably extract ground control points much faster and with less resources than a common PC.

The general approach of their proposed system is to extract and match GCPs from one QI with a particular reference image, compute the model coefficients with these reference points, and then extrapolate the model other QIs. This follows a similar approach used by Liang and Wang [19], who found that extrapolating a georeferencing model to image regions different to those used for GCP extraction may result in a significant loss in accuracy. This also highlights a critical weakness of this implementation: it requires reference images to be stored on board for GCP matching.

In practice, this novel approach would need significant storage resources on board to store the database of reference images, and its compatibility with more realistic raw images is questionable. Additionally, this is an implementation optimized for performance, rather than full system-integration. In contrast, the system proposed here takes all these additional factors into account.

#### 2.1.2. *HYPSO-1 Mission Application*

A paper by Mariusz E. Grøtte *et al.* [27] demonstrates an operational mission considering the use of onboard

georeferencing. The HYPSO-1 CubeSat mission from the Norwegian University of Science and Technology (NTNU) utilizes a FPGA to run onboard applications, and the paper mentions four additional candidate algorithms, of which georeferencing is one. The authors identify that the benefit of onboard georeferencing lies in "directly downlinking latitude and longitude coordinates of the image pixels together with extracted results from target detection or classification, requiring precisely time-synchronized attitude and position data"[27, p.13]. Elaboration on this algorithm is extremely limited, but their note concerning "time-synchronized data" strongly indicates a direct georeferencing approach will be used (one not reliant on image matching). A follow-up publication in 2023 directly states that "... image data are automatically downlinked at the next ground station pass by the MCS. Downlink of a single capture may require more than one ground station pass..." [6, p.6], confirming that onboard georeferencing is not yet implemented.

As a result, while the need for an onboard georeferencing system is well established, no complete system is currently available.

### 2.2. Keypoint Models

A few dedicated surveys have been conducted in recent years that investigate the challenges of keypoint extraction from satellite imagery [14, 31]. From these studies, a number of potential models and common methods have been identified that could be suitable for this application. The first is the use of self-supervised contrastive learning to increase the amount of available training data [15]. For example, Superpoint proposes a Siamese network that uses augmentations of an image to generate a large quantity of training pairs, since the corresponding pixels between two images is known [8]. A second recurring paradigm is the use of joint detection and description, rather than local feature description. Traditional keypoint detection pipelines use separate description and detection algorithms, while these newer deep learning models use the same network for both tasks [32].

While there is an abundance of deep learning models for keypoint extraction, two stand out in particular regarding the application to satellite imagery: D2-Net and a novel approach called the Detected, Repeatable, Reasonable, and Distinguishable (DRRD) keypoint framework. The DRRD framework proposes a generic set of modifications, that can be applied to existing classification and segmentation models to extract salient features from an image, and identifies the most distinguishable keypoints from these results. It also presents a promising method of self-supervised learning, in which co-registered multi-band images are used as training pairs. Since the corresponding pixels between bands refer to the same photographed surface feature, the model can learn to extract features that are robust to spectral changes [35]. Additionally, the DRRD framework claims it removes the need for a computationally expensive RANdom SAmple Consensus (RANSAC) filtering step, as the model is able to extract features that are statistically very likely to be repeatable across multiple images. Unfortunately, the exact implementation of the DRRD framework is not available. While this is an extremely promising approach and has the potential to be applied directly to models proven to be compatible with satellite platforms, the lack of details prevents it from being used in this research.

D2-Net is a feature extraction model that is designed to specifically address the challenges of illumination changes. While technically not the same, illumination differences in traditional images can be considered comparable to spectral band differences in satellite imagery. The architecture and approach of D2-Net is extremely similar to the Deformable Convolutional Network (DCN) proposed by Li *et al.* [18], which claims to extract keypoints from satellite imagery that is robust to geometric and radiometric changes. Furthermore, the code is open source and pretrained models are available, making this a very promising option [9]. The model is also based on a lightweight Convolutional Neural Network (CNN) architecture called VGG16, which is proven to be compatible with the limited computational resources of a small satellite [12].

One additional model was considered, the 2chADCNN network, which is focused specifically on matching satellite images that have significant seasonal changes [31]. While this network is very applicable to georeferencing, it is not compatible with the limited resources of a satellite. The 2chADCNN network relies on scaling and sliding windows with a CNN to compute the similarity between sub-scenes. This is a completely different approach, as it requires full reference images to be present. As a result, this model is not considered further.

## III. Novel Georeferencing System Architecture

The general architecture of this novel georeferencing system is designed to fit within a number of constraining requirements. The system must be capable of operating on the limited computational resources of a small satellite, must be able to deliver the results off-platform in real-time (within 5 minutes of image capture), and produce a transformation function between the image reference frame and the spatial reference frame with an accuracy consistent to current georeferencing systems. The scope of this research is to demonstrate that the proposed architecture is feasible using a proof-of-concept implementation.

In essence, the system uses indirect georeferencing of

[1]See Appendix A for an enlarged version.

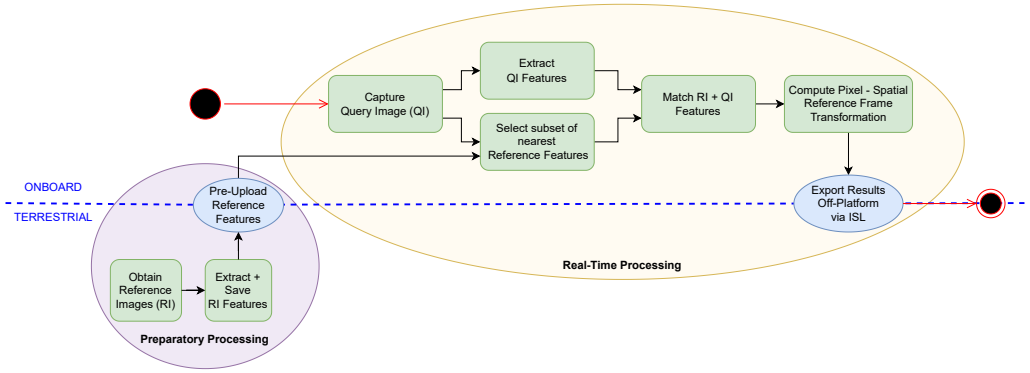



**Figure 4.1:** Overall system architecture diagram[1]

the captured QI, using pairs of GCPs based on matching features. These features are extracted directly from the QI, while the reference features are pre-extracted and stored in a database called the Reference Feature Database (RFDB). Each reference feature contains metadata of its location in the Spatial Reference System (SRS). Matching QI features located in the Image Reference System (IRS) to these Reference Frames (RFs) generates corresponding coordinate pairs between the two reference frames. These pairs are then used to calculate a transformation function, which is applied to the QI to produce a georeferenced image. The overall architecture is depicted in Figure 4.1.

Figure 4.1 is organized as a flowchart to represent the sequential nature of the system. The black and red dots signify the start and end of the real-time pipeline. A horizontal dotted line divides the elements that are executed on the terrestrial resources and those on board the spacecraft. From left to right, the pipeline starts with a preprocessing step: creation of the RFDB. This is performed on the ground, as it requires access to a large database of existing references images that are already georeferenced. The RFDB is then uploaded to the spacecraft, where it is stored on board.

The real-time pipeline begins as soon as an image is captured. It is assumed that the QI has an initial location estimate, which is used to select an appropriate subset of features from the RFDB. Additionally, the QI is passed to a keypoint extraction model, which is tasked with extracting the most important keypoints in the image, and describing these in the form of a feature vector. These features are then matched pairwise with the RFDB features, and the image and spatial coordinates are saved for each match. These coordinate pairs are passed to a least-squares approximation algorithm which computes the coefficients of a polynomial transformation between the two reference frames. This transformation function is the final georeferencing result and can convert image coordinates of any Point of Interest (POI) to geospatial coordinates.

It is clear that the architecture diagram does not specify exact algorithms or implementations used to demonstrate the concept. Fundamental to this architecture design is its reliance on multiple independent state-of-the-art technologies, which are combined in a modular fashion. As research in the respective fields progresses, this novel georeferencing system can adopt these improvements and continue to perform at the highest accuracy. These primary elements are: the feature extraction model (used for both the QI and RFDB), the subsequent filters to select the most relevant features as keypoints, the generation method of the RFDB, and the feature matching method. These modular elements are detailed below, as well as the polynomial transformation model used.

### 3.1. Feature Extraction Model

In order to create keypoint correspondences between the QI and reference image, most georeferencing systems either use manually identified GCPs, or extract features from the image using traditional computer vision algorithms such as Scale Invariant Feature Transform (SIFT), SURF, or Orientated FAST and Rotated BRIEF (ORB). A downside of these handcrafted algorithms is that they are fine-tuned to operate on very specific types of images, and not robust to significant changes such as spectral or temporal differences. A field currently under heavy development is the use of deep learning models for feature extraction and description. Recent advancements have shown these models are capable matching or exceeding the performance of traditional algorithms, while being more robust to changes in the image [13, 26].

For this proof-of-concept, it was determined deep learning models would be preferential to traditional feature extraction algorithms. Three main factors drove this decision: the overall system flexibility, model robustness to varying inputs, and the potential for future innovations. The diverse and rapidly growing selection of deep learning models each specialized for various use cases means there exists a high potential that at least one performs well on satellite imagery. Although this system implementation is evaluated on a simplified dataset, selecting a model that has the ability to per-

form on non-ideal data (ie. with spectral or perspective differences) significantly strengthens the claim of the overall system's feasibility. Furthermore, contrary to handcrafted algorithms, these feature extraction models can be re-trained on representative data in order to improve their performance, solidifying their potential to improve over time with new innovations. Finally, recent developments of space-grade dedicated Vision Processing Unit (VPU) hardware supports the use of these SotA models on board small satellites.

#### *3.1.1. Challenges*

The model must be capable of extracting features that are robust to challenges in satellite imagery, such as temporal, perspective, and spatial resolution differences, as well as various spectral bands. Reference features extracted from a reference image must also consistently be extracted from a temporally different query image of the same geographic region. Additionally, the model must be capable of running on the limited computational resources of a small satellite.

When comparing the current SotA deep learning models for feature description and extraction, it becomes apparent that most research is focused on classical applications, such as cityscapes, daily objects or medical images. Objects present in the images of these datasets are typically on a macro scale relative to the dimensions of the image. Satellite imagery on the other hand appears much more granular, as the individual pixels often represent an entire object on the ground.

As such, the model must identify and describe features in the satellite image that are representative of the latent surroundings. Temporary objects such as cars that move, or trees that lose their leaves in winter, are not suitable ground control keypoints as they will not be consistent over time. The model must therefore learn to identify features such as landforms or buildings that are more permanent. Furthermore, the QIs captured by the spacecraft are single band images, in contrast to the standard RGB images used to train the majority of deep learning models. Significant differences can exist between bands, thus the features must also be robust to spectral changes.

Finally, the scarcity of research dedicated to deep learning models for satellite imagery results in less publicly available pretrained models and datasets. This means the model used for this pipeline must be retrained, and the necessary training datasets must also be created.

#### *3.1.2. Model Selection*

An implementation of the DRRD framework applied to a U-Net with a MobileNetV2 encoder was tried, but did not achieve the expected performance. As a result, the D2-Net model was selected as the feature extraction model for this research. The pretrained model weights and clear architecture design gave confidence that it would actually function in the georeferencing system implementation. Additionally, D2-Net is commonly used in related works as a baseline reference, and consistently scores better than other baseline models such as Superpoint, DELF and SIFT [9]. It is noted that some research indicates that D2-Net has localization issues regarding the keypoint positions[35]; the impact on this system is discussed below. However, should the effects of these issues prove to be sufficiently detrimental such that D2-Net is deemed unsuitable for this architecture, the modular system design allows for it to be replaced with an alternative model if necessary.

#### *3.1.3. Loss Functions*

In addition to the numerous potential deep learning network architectures and training methods presented above in Section II, another consideration of significance, which ultimately defines the success of a model, is its loss function. Since this function defines exactly what and how the model will learn during its training, it is critical that it encapsulates how the model should perform. In this application, the loss function must both describe the features in a generic manner, and localize the most distinct features in the image. While doing so, the returned features must also be repeatable and robust to spectral, temporal, and geometric changes. Zhang *et al.* [35], the authors of DRRD, propose a novel loss function that claims to do exactly this. This Adaptive Mixed Context Triplet (AMCT) loss function supposedly combines the strengths of the Mixed-Context Triplet (MCT) introduced by Keller *et al.* [16] with an adaptive term that compensates for spectral variations found in the pairs of multi-spectral satellite images used for self-supervised training. Unfortunately, their exact implementation is unclear and could not be reproduced.

### 3.2. Keypoint Selection Filters

A second modular element of the system follows directly after the keypoint extraction pipeline. Deep-learning feature extraction models such as D2-Net produce dense feature maps, such that each pixel is a feature described with an n-dimensional vector. Each feature also has an attributed score, which is a measure of its distinctiveness compared to others and how significant the feature is in the image.

Unique to this system architecture is the retention of only a subset of the features, in order to only retain the most important keypoints. This implementation uses a "Top-K Layer", which effectively sorts the keypoints by their score and selects the best K (where K is a tunable parameter). This step is necessary to reduce the number of keypoints and the computational complexity during the matching process [24]. The inclusion of this step is recommended by similar pipelines applied to satellite imagery [35].

Alternative methods could also be implemented in the

future, such as filtering the keypoints based on a minimum score threshold. Being a modular architecture, this could easily be implemented without requiring changes to other parts of the pipeline. Details on these alternative methods are discussed below.

### 3.3. RFDB Generation

The current implementation of the Reference Feature Database (RFDB) uses the GeoPandas library to store a spatial DataFrame of the reference features.[2] The RFDB is generated by extracting the top-100 features from each tile of a reference image dataset covering a predefined region. The best configuration of bands to be used for this is determined below. The complete database is then saved to a geopackage file. This format is an open-standard, platform-independent SQlite database file, which supports reading subsets based on spatial queries.[3] This allows the system at inference-time to only load the necessary reference features from the database, rather than the entire file, which significantly reduces the memory requirements.

The current implementation uses Python, which is not optimized for computational efficiency. Alternative implementations could be used here in the future such as GDAL or QGIS and still be compatible with the generated databases. The storage size of these RFDBs is dependent on the number of features extracted per tile, and the number of tiles. Table 4.3 presents the exact results of the resulting file sizes.

### 3.4. Matching Method

Once the keypoints have been extracted from the QI and the subset of features from the RFDB is loaded, the descriptor vectors are matched to determine the correspondences. The current method is to use the OpenCV "brute force matching", which compares each descriptor in the QI to each descriptor in the RFDB. For each comparison, the L2 (Euclidean) Norm (see Equation 1) measures the similarity between the two vectors. Of these, a match is only considered valid if "i-th descriptor in set A has j-th descriptor in set B as the best match and vice-versa"; this is known as "CrossCheck".[4]

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2} \tag{1}$$

Where:

- $x$ and $y$ are the two descriptors being compared
- $n$ is the number of elements in the descriptor vector (512 in this case)

Other matching methods were considered, such as Fast Library for Approximate Nearest Neighbors (FLANN) which is commonly used for features extracted by the ORB algorithm. FLANN uses a KD-tree data structure to very efficiently identify and select the nearest neighbors in terms of euclidean distance. However, since FLANN shows little to no advantage when the number of keypoints is small (less than 1k), building this KD-tree is extra computational overhead. It is also known that FLANN prioritizes speed over identifying the exact best match, which in this application would be detrimental to the georeferencing performance [7, 29]. Therefore, the brute-force method was selected as an effective solution.

Future iterations of this system could opt to implement FLANN or other matching algorithms, if the density of the keypoints is increased and the trade-off between speed and accuracy is more favorable. Novel deep-learning matching models could potentially also be considered as alternative matching methods that can be trained to interpret the descriptors and be more robust to the spectral or temporal difference: SuperGlue and LightGlue are two such models [21]. However, these require the original images and use heavyweight transformer models that are not compatible with the limited computational resources of a small satellite.

### 3.5. Polynomial Transformation

The transformation between the image reference frame and the spatial reference frame is calculated using a polynomial function. This function can either be first- or second-order, depending on the number of GCPs available. A second-order polynomial is preferred, because it can rectify distortions in an image, while the first-order only corrects for affine transformations [1]. This system implementation always uses second-order transformations. The general forms of the polynomial functions are as follows:

First order:

$$\begin{aligned} X_i &= A_0 + A_1 x + A_2 y \\ Y_i &= B_0 + B_1 x + B_2 y \end{aligned} \tag{2}$$

Second order:

$$\begin{aligned} X_i &= A_0 + A_1 x + A_2 y + A_3 x^2 + A_4 xy + A_5 y^2 \\ Y_i &= B_0 + B_1 x + B_2 y + B_3 x^2 + B_4 xy + B_5 y^2 \end{aligned} \tag{3}$$

Where:

- $X_i$ and $Y_i$ are the spatial coordinates
- $x$ and $y$ are the image coordinates (column and row respectively)
- $A_0$ to $A_5$ and $B_0$ to $B_5$ are the coefficients of the polynomial function

The coefficients of the polynomials are estimated using the least-squares method. The first-order requires

[2] https://geopandas.org/en/stable/index.html
[3] https://www.geopackage.org/
[4] https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

a minimum of three GCPs, while the second-order requires a minimum of six GCPs. Equation 4 demonstrates how the $[A]$ coefficients are calculated (with 10 GCPs supplied) [23]:

$$\begin{bmatrix} 1 & x_1 & y_1 & x_1^2 & x_1y_1 & y_1^2 \\ 1 & x_2 & y_2 & x_2^2 & x_2y_2 & y_2^2 \\ 1 & x_3 & y_3 & x_3^2 & x_3y_3 & y_3^2 \\ 1 & x_4 & y_4 & x_4^2 & x_4y_4 & y_4^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{10} & y_{10} & x_{10}^2 & x_{10}y_{10} & y_{10}^2 \end{bmatrix} \times \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ \vdots \\ X_{10} \end{bmatrix} \tag{4}$$

Rewriting this in equation form, the system can be solved to find the coefficients using Equation 5. The process is repeated for coefficients $[B]$:

$$\begin{aligned} QA &= X \\ A &= (Q^TQ)^{-1}Q^TX \end{aligned} \tag{5}$$

#### 3.5.1. RANSAC

In order to remove false keypoint matches, RANSAC is applied during the calculation of coefficients. This algorithm identifies outliers by iteratively computing the polynomial model using random subsets of the GCPs, and computing the error with the remaining GCPs to identify data samples that do not fit the model. The process is repeated until a model is found that has the best fit for a subset of the data [11].

## IV. EXPERIMENTAL EVALUATIONS

In order to quantify the performance of the resulting system, a series of evaluations have been performed using industry-standard metrics. These experimental tests are designed to evaluate whether the proposed system architecture has the potential to function as an accurate georeferencing pipeline. This is done with three categories of evaluations: first, using simplified datasets to establish the best combination of parameters to use; second, using more realistic datasets to determine the robustness to non-ideal input data and identify the next steps for future development; and third, to measure if the computational resource usage is compatible with small sat hardware. The use of idealized data was intentional: it enabled the systematic development of a baseline georeferencing system not plagued by problematic input data, such as cloud coverage, atmospheric effects, and geometric distortions.

### 4.1. Datasets

Each test was run on Sentinel-2 Level 2A MSI data from four geographic regions: Spain (ES), South Africa (ZA), Germany (DE), and The Netherlands (NL) [10]. Every Sentinel-2 product contains 13 spectral bands, each saved in a separate image file of 10980 by 10980 pixels with a 10m Ground Sample Distance (GSD). To simulate the onboard processing of the georeferencing pipeline, the images were split into smaller tiles of 320 by 320 pixels; a parameter determined by the baseline evaluations explained below.

A distinction is made between a Sentinel-2 *product* and a *dataset*. Each *dataset* is constructed dynamically by this analytical model from a single *product*, using a certain subset of bands and split into tiles of a specified size. Different datasets can therefore be used to control the temporal, spectral, and/or spatial differences between the RFDB and the QI.

All products have a cloud cover percentage less than 1.4%, an essential requirement for the feature extraction model to perform consistently. The selection of products acquired via the Copernicus Open Access Hub was based primarily on minimizing cloud coverage, and second on a consistent temporal difference. However, only a limited selection of products are available online, and typically only the most recent acquisitions. Therefore, the temporal difference between the RFDB and the QI is not consistent across all products. Also, for certain geographic regions, such as the Netherlands, near-perfect cloudless days are rare and thus do not have many products available. Products used to create datasets for the evaluations are listed in Table 4.1.

**Table 4.1:** Sentinel-2 products used for evaluations

| Region | Cloud Coverage | Capture Date | RFDB Source | Temporal Difference w/ RFDB |
|---|---|---|---|---|
| ES | 0.85% | 28-8-2023 | Yes | - |
| ZA | 0.10% | 3-10-2023 | Yes | - |
| DE | 0.20% | 26-9-2023 | Yes | - |
| NL | 0.63% | 19-7-2022 | Yes | - |
| ES | 1.33% | 23-8-2023 | No | 5 days |
| ZA | 0.18% | 18-9-2023 | No | 15 days |
| DE | 0.71% | 11-9-2023 | No | 15 days |
| NL[†] | 1.36% | 7-9-2023 | No | 415 days |

[†]Only used in baseline evaluations to confirm that cloud cover is detrimental to the system.

A third NL product with a significantly more cloud coverage (4.68%) and a temporal difference of 105 days was used to demonstrate that cloudless data was indeed a necessary simplification for this design phase. This product was only used in a few evaluations and not considered a part of the main dataset since cloud-obscured images had already been defined to be outside the scope of the system design.

Because the operation of the georeferencing pipeline depends on pre-extracted features with their corresponding spatial coordinates stored in a RFDB, the products of each region (NL, ES, ZA, or DE) cover identical geographic footprints. The dataset used to create the RFDB processes all tiles (for each specified band) sequentially through the keypoint model to extract features, which are then saved to a geopackage

database file.

For the four baseline evaluations, the QI tiles' geographic footprint matched that of the tiles used to generate the RFDB. This ensured that each test strictly evaluated the effect of the temporal and spectral differences between the RFDB and the QI. For the robustness evaluation, the QI tiles' geographic footprint was shifted by a small amount to measure the effect of the spatial difference between the RFDB and the QI.

## 4.2. Metrics

The total accuracy of the system is measured using the industry-standard Root Mean Square Error (RMSE) metric. This is calculated in both the x- and y- directions, which in the case of the Sentinel-2 products, corresponds to easting and northing errors in meters, respectively. The RMSE is calculated for each Independent Check Point (ICP) using Equation 6 [33]:

$$\begin{aligned} RMSE_r &= \sqrt{RMSE_x^2 + RMSE_y^2} \\ RMSE_x &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2} \\ RMSE_y &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \end{aligned} \tag{6}$$

A secondary metric, "Circular Error at 90% confidence level (CE90)", was considered but ultimately not used due to its deprecation by industry standards and its required assumption that the errors are normally distributed about a zero mean [4]. This assumption cannot be made for this experimental system, and thus the RMSE is a more robust metric.

### 4.2.1. Relative vs. Absolute

The accuracy measured here is considered to be the "relative" accuracy, since the errors are calculated with respect to the positional coordinates derived from the reference images. Whether the reference images are georeferenced correctly is outside control of this system [30]. Nonetheless, it should be noted that the "absolute" accuracy of the system is thus the sum of the relative accuracy, and the absolute accuracy of the reference images. This behavior will always be present in any indirect georeferencing system.

## 4.3. Methodology

Establishing whether the georeferencing system is able to accurately transform between the IRS and the SRS is the primary goal of the evaluations. Since in reality the images used to create a RFDB will always be captured prior to the query image, the most realistic test condition is one with a temporal difference. The tests are therefore structured to evaluate the system's performance in cases of increasing complexity, and are based on the difference between the dataset used to create the RFDB and the dataset from which the QIs are sourced. The tests are summarized below:

### 4.3.1. Tests

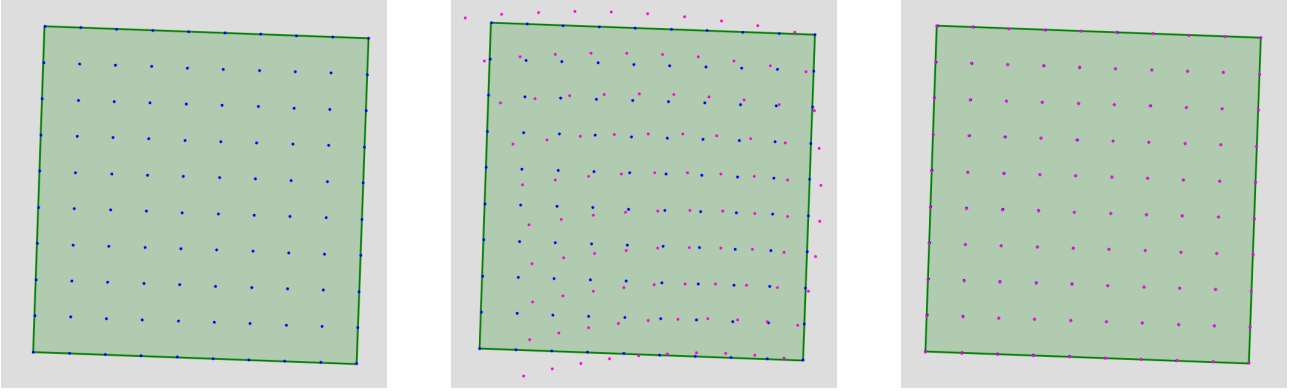
For a tile size of $320 \times 320$, each dataset contains 252 query image samples. The following tests were performed on all query image samples, and the RMSE is averaged to obtain the final accuracy. The tests are listed below, and the results are reported in Section V.

- T-1 **Spectral**: Only a spectral difference between the RFDB and the QI datasets. The QI and RFDB datasets are sourced from *the same* Sentinel-2 product, but the QI dataset only contains bands not present in the RFDB dataset. Only the red, green, and blue bands were used in these experiments (Sentinel bands 4, 3, and 2, respectively). Note: this test was only conducted for evaluations using single-band RFDBs.
- T-2 **Temporal**: Only a temporal difference between the RFDB and the QI datasets. The QI and RFDB datasets are sourced from *different* Sentinel-2 products, but contain the *same* bands.
- T-3 **Spectral & Temporal**: Combination of temporal and spectral differences between the RFDB and the QI. Note: this test was only conducted for evaluations using single-band RFDBs.
- T-4 **Spectral with RANSAC**: Same as the spectral test, but additionally using the RANSAC algorithm to filter out outliers. Note: this test was also only conducted for evaluations using single-band RFDBs.
- T-5 **Temporal with RANSAC**: Same as the temporal test, but additionally using the RANSAC algorithm to filter out outliers.
- T-6 **Spectral & Temporal with RANSAC**: Same as the Spectral + Temporal test, but additionally using the RANSAC algorithm to filter out outliers. Note: this test was also only conducted for evaluations using single-band RFDBs.

### 4.3.2. Independent Check Points (ICPs)

Similar to the generation of GCPs to compute the georeferencing transformation between the pixels in the IRS and the SRS, ICPs are created to independently verify the accuracy of the transformation across the whole of the image. Traditionally, these are pixels in the QI that have been manually selected, of which the spatial coordinates are known.

However, within the baseline development environment of this system, the ICPs could be generated automatically. Sentinel Level-2A products are already



**(a)** ICP 10 by 10 grid in the IRS.

**(b)** ICPs transformed to the SRS.

**(c)** ICPs transformed to the SRS with RANSAC applied during coefficient calculation.

**Figure 4.2:** Examples of testing the polynomial transformation function using ICPs, without and with RANSAC. The green squares are the actual bounds of the QI. Blue points are ICPs in the Image Reference System (IRS). Pink points are the ICPs transformed to the Spatial Reference System (SRS) using the georeferencing polynomial function.

georeferenced, and contain metadata on how to transform between the IRS and SRS. Thus, the ICPs can be generated by simply creating a regular grid over the QI. Transforming the grid points to the SRS using this metadata gives the actual spatial coordinates of each ICP. This can be seen in Figure 4.2a.

The QI is passed to the georeferencing pipeline as a plain image (containing only pixel intensity values and no metadata). The keypoints are extracted and matched to the corresponding reference features, from which the polynomial transformation function is derived. Each point in the regular grid of ICPs is sequentially transformed to the SRS using this function to obtain the predicted spatial coordinates, as seen in Figure 4.2b. The mean error between the actual and predicted coordinates of each ICP can then be calculated to obtain the accuracy of the georeferencing pipeline.

## 4.4. Evaluations

The three categories of evaluations are explained below. The first was used to identify the optimal configuration of parameters, the second to evaluate robustness to variable inputs, and the third to measure the computational resource usage.

### *4.4.1. Baseline Parameter Evaluations*

The baseline evaluations were designed to evaluate the effect of changing implementation parameters of the system on the mean accuracy. These are summarized in Table 4.2. Aspects such as model weights, combinations of bands used in the generation of the RFDB, tile sizes and the use of RANSAC are all parameters that must be chosen when developing the system. The goal of these evaluations was to determine the optimal parameters for the system, and to establish a baseline accuracy for the system.

Note: these evaluations were performed in sequence, and the resulting configuration that produced the lowest RMSE was applied to the subsequent evaluation. Evaluations with unknown parameters were initially selected based on best judgment and recommendations from literature, and later re-evaluated with the proper parameters once they had been determined. For example, the Model Weights evaluation was performed first with the single-band RFDB dataset, and then re-evaluated with the three-band separate RFDB dataset once that had been determined to be most performant. Only the final results are reported here, to consistently report the effects of each evaluation. RANSAC is an exception to this, since it does not have an effect on how the features are described or extracted.

**Table 4.2:** Overview of evaluations

| Evaluation | Modified Parameter | Tests |
|---|---|---|
| Model Weights* | Pre-trained weights of the D2-Net model | 1-6 |
| RFDB Bands* | Method of generating RFDB features (from a single, combination of, or three separate bands) | 1-6 |
| Tile Sizes* | Tile size used for both RFDB generation and QI samples | 2, 3, 5, 6 |
| RANSAC* | Applying RANSAC during polynomial coefficient computation | 2, 3, 5, 6 |
| Offset Tiles† | Spatial offset between QI tile and the tiles used to generate the RFDB | 2, 3, 5, 6 |

*Baseline Parameter Evaluation
†Georeferencing Robustness Evaluation

*Model Weights*

The first evaluation was to establish which model weights should be used by the keypoint extraction model. As reported by Dusmanu *et al.* [9], three pre-trained models were made available: `d2_ots.pt`, off-the-shelf Caffe VGG16 weights; `d2_tf.pth`, fine-tuned weights; and `d2_tf_no_photourism.pth`, fine-tuned

weights trained on data without PhotoTourism scenes. The goal of this evaluation was to confirm that the off-the-shelf weights were indeed the best choice for the keypoint extraction model, as suggested by the authors of D2-Net.

*RFDB Band(s)*

The feature extraction model, D2-Net, requires input images to be tensors of shape (3, H, W); three channels for the red, green, and blue bands and $H \times W$ pixels. However, multispectral images consist of more than three bands, and the proposed system is designed to georeference a single band at a time. As such, the input satellite query images of this implementation are repeated three times in the channel-dimension to fit this shape requirement.

The features stored in the RFDB should in theory be representative for any QI, regardless of the band. Since the D2-Net can accept multiband images, the hypothesis was made that combining the red, green, and blue bands together (as a single input image) during the RFDB generation could create feature descriptors representative of all three bands, thus reducing spectral matching error.

To test this hypothesis and determine the best approach, RFDBs were generated in three ways:

- **Single-Band (1B)**: Using single-band input images, from a dataset containing only band B02. This repeats the band thrice to create the required input shape.
- **Three-Bands Combined (3B-Comb)**: Using multi-band input images, from a dataset containing bands B02, B03, and B04. This combines the three bands into a single input image.
- **Three-Bands Separated (3B-Sep)**: Using single-band input images, but from a dataset containing bands B02, B03, and B04. This results in an RFDB with three times the number of features as the previous two methods.

Note, for method 1B, band B02 was chosen for the RFDB because it allowed for a secondary analysis objective: the effect of the spectral difference *magnitude* between QI and RFDB images on the overall system accuracy. Sentinel-2 band B02 has a wavelength of 490 nm and is spectrally more similar to B03 (560 nm) than B04 (665 nm) is.[5]

*Tile Sizes*

For the third evaluation, the size of images passed through the georeferencing pipeline was varied, with QI and RFDB tile sizes remaining consistent. Five sizes were selected, based on what is being used onboard small satellite hardware today and what was maximally possible given the computational resources available (Nvidia RTX 3050 4GB). All images are square, with side dimensions (in pixels) of: 256, 320, 480, 640, and 832. The goal of this evaluation was to determine the optimal image size for the georeferencing pipeline, across all geographic regions.

*RANSAC*

The final evaluation compares the effect of applying RANSAC to the georeferencing pipeline. As described in the previous section, the addition of RANSAC has been extensively proven in literature to significantly improve the results of georeferencing models. However, since the goal of this pipeline is to minimize the number of onboard computations required, ideally the effect of RANSAC on overall accuracy would be minimal. This would suggest that the keypoints extraction model is optimized to detect only inlier keypoints and RANSAC would not be necessary. The data for this evaluation was generated by running previous evaluations with and without RANSAC.

### 4.4.2. *Georeferencing Robustness Evaluation*

Once the baseline evaluations had determined the optimal parameter values of this implementation, the system was tested for its response to variations in input data. The results of this evaluation could be compared to the baseline measurements, in order to determine the limitations of the system and identify the next steps for future development.

In reality, once this georeferencing system becomes operational, the QI will never have the same spatial footprint of the tiles used to create the RFDB. The goal was to determine the effect of a spatial offset between the tiles used to create the RFDB and the QI. The QI was shifted by 5, 10, 80, 160, 270, 290, and 310 pixels in both the x- and y- directions, and the resulting accuracy was compared to the previous evaluations (with no offset). The hypothesis for this test was that the model would extract a different subset of keypoints from the QI, since each feature score is relative to the other feature scores in the descriptor map. An offset tile will be passed different input data, thus the descriptor maps will be different.

### 4.4.3. *Hardware Resource Consumption*

In order to determine if this novel pipeline would be compatible with small satellite hardware, the computational resource usage was also measured during the evaluations. In accordance with the requirements, two metrics were recorded: the Random-Access Memory (RAM) usage and the required disk space to store the RFDB. The results are reported below and analyzed with respect to the laptop on which the simulations were run.

[5] https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-2-msi/resolutions/spectral

*RAM Usage*

Regarding the memory usage of the individual keypoint extraction, matching, and georeferencing steps, it was unfortunately not possible to measure this to an exact value. This is due to the complexities in the memory allocation of Python. The simulation of the georeferencing system relies heavily on the industry-standard `rasterio` library, which consists of Python wrappers around Geospatial Data Abstraction Library (GDAL) methods written in C++. Additionally, a number of custom CPython functions have been written to utilize GDAL's existing RANSAC methods. An attempt was made to profile the simulation's memory usage with Python's built-in `tracemalloc` library, but this library is limited to tracking only memory blocks allocated by the Python interpreter, and therefore excludes a significant portion of the simulation's operations.[6] Thus, an analytical approach was taken to estimate the memory usage of the georeferencing pipeline.

Using Python's `psutil` library, the system's memory usage was recorded throughout the simulation in 0.01 second intervals. The simulation was run to georeference a total of three standard 320×320 pixel single-band QI images, as used in previous tests.

*Disk Space*

The file storage was determined by recording the size of the RFDB file saved to the disk. Each RFDB contains features from a single Sentinel-2 product as explained above.

## 4.5. Additional Investigative Tests

The results of the offset tile tests were unexpected, and thus additional tests were performed to investigate the cause. This was done via a qualitative analysis to visually inspect the effect of the offset on the matched keypoints and to identify the source of the error.

The investigation followed three main steps. First, the system was tested with additional offsets, and the matched keypoints were plotted using the mapping visualization. The results were analyzed and a number of potential issues were noted. Subsequently, these concerns were addressed through a series of issue isolation tests:

1. Verification that the QI contains the correct pixel values
2. Verification that the visualization tool is correctly plotting pixel and spatial coordinates
3. Cause of clustered keypoint matches
4. Investigation of parallel match lines

These tests are detailed in the discussion section below. Finally, based on a thorough analysis, a theory is formed on the cause of the error and structured approach to resolve the issue is proposed.

[6] https://docs.python.org/3/library/tracemalloc.html

### 4.5.1. Visualization of the georeferencing process

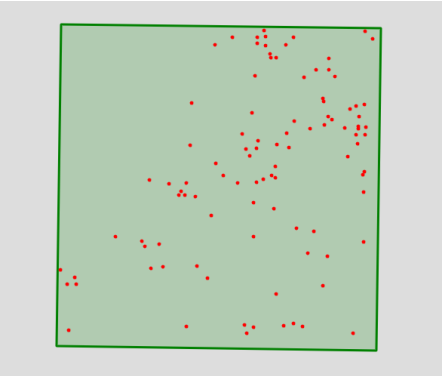


**Figure 4.3:** Visualization of QI extracted feature positions relative to the image

To visualize the accuracy of this matching process, the actual locations of the QI's bounds are added to a map as a green square, as well as the extracted QI keypoints (Figure 4.3). The spatial positions of these keypoints are known, based on the row/column pixel coordinates returned by the keypoint extraction model, and the metadata provided by the Sentinel-2 dataset. This allows the positions of the QI keypoints to be accurately placed on the map.

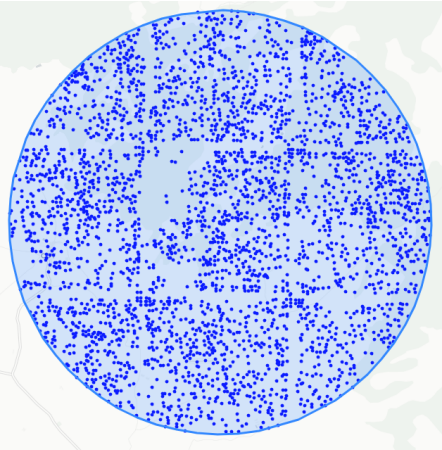


**Figure 4.4:** Visualization of RFDB subset selected, based on the QI initial position estimate

In Figure 4.4, the subset of RFDB keypoints around the QI initial position are similarly placed on the map, using the metadata provided by the RFDB dataset. Important to note is that the extraction of the QI and RFDB keypoints are independent processes, and the spatial locations of the QI keypoints are not used in the georeferencing process. The QI keypoints are only used for visualization purposes, to help understand the georeferencing process.

The grid artifact apparent in Figure 4.4 is noted, and is a result of the D2-Net architecture. The keypoints

localized exactly on the image edge are removed, in order to counteract the effects of padding in the convolutional layers. This effect is seen as the "gaps" of missing keypoints that form a grid-like structure in the RFDB. This effect would need to be addressed in future work, but is not considered a significant issue at this point.

## V. RESULTS AND ANALYSIS

The results of the evaluations are presented in this section. All results are presented with and without RANSAC applied during the georeferencing process. The dashed lines or cross-hatched bars indicate the results without RANSAC, while the solid lines or solid bars indicate the results with RANSAC. The results are presented in the following order: baseline evaluations, robustness evaluations, and hardware compatibility.

### 5.1. Baseline Evaluations

As noted above, all evaluations here are conducted on datasets with the same geographic footprint. Preliminary analyses of the results are also presented, in order to determine the best configuration of parameters for the system, before continuing to evaluations with offset QIs.

#### 5.1.1. *Pretrained Model Weights*

Comparison of the three available pretrained model weights for D2-Net can be seen in Figure 4.5. The measured RMSE is a mean value across all six tests.

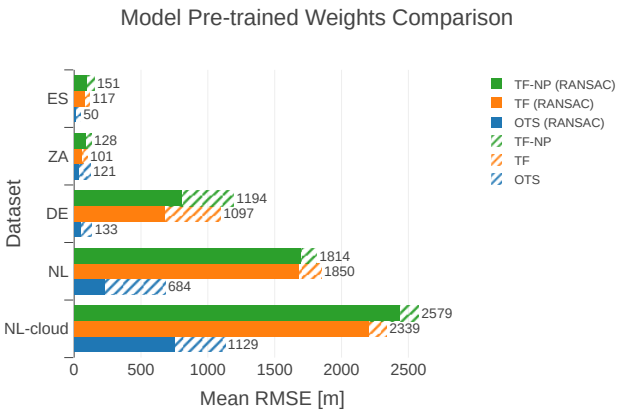



**Figure 4.5:** Pretrained model weights comparison, mean ICP RMSE across all tests

The fine-tuned (TF) weights and those fine-tuned on a dataset without phototourism images (TF-NP) produced significantly less accurate results, but is consistent with what Dusmanu *et al.* [9], the authors of D2-Net reported in their analyses. Therefore, for all subsequent evaluations, only the Off-the-Shelf (OTS) model weights were used.

#### 5.1.2. *RFDB Bands*

Figure 4.6 presents the three different configurations of constructing the RFDB with multiple bands.

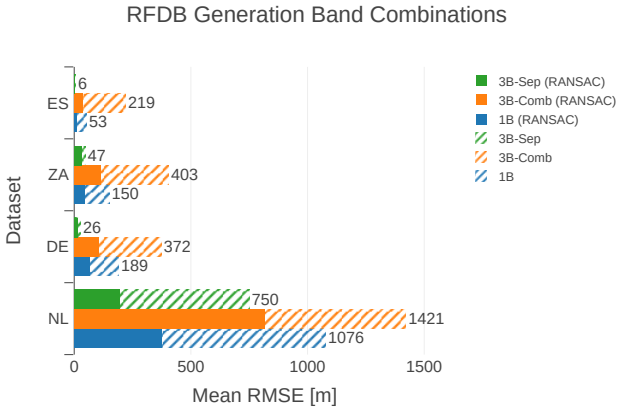



**Figure 4.6:** Mean RMSE of temporal difference tests

A clear conclusion can be drawn that the 3B-Sep method produces the most accurate georeferenced results. This is a logical outcome, as the generation of the features for both the RFDB and the QI are most similar. Both pass a single-band image through the D2-Net model, and thus the feature matching only contains temporal differences. A downside of using this method is the significantly increased size of the RFDB. However, as explained previously, the use of geopackage database files allows the system to only load the nearest reference features, thus keeping the memory footprint low during operation. Also, the 3B-Comb RFDB performs consistently worse than a single-band RFDB. This result is slightly unexpected, since the model was trained on RGB images.

As explain above, a secondary analysis was performed by selecting band B02 for the 1B RFDB method; the results of which are presented in Figure 4.7.

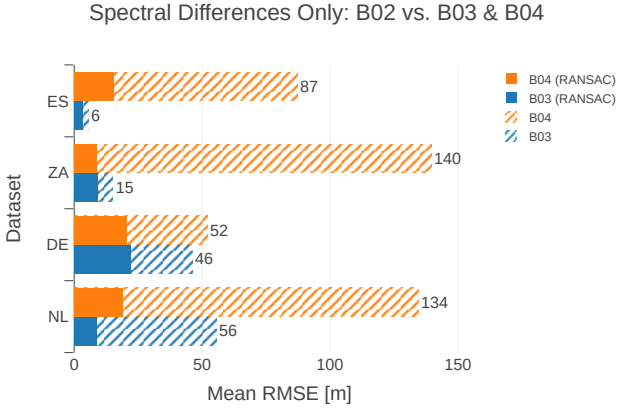



**Figure 4.7:** QI sampled from bands B03 and B04, using an RFDB generated using only B02. No temporal differences.

Inspecting Figure 4.7, shows that as hypothesized, band B03 has in general a better accuracy than band B04 for all cases without RANSAC. This is likely because a smaller difference in wavelength between the QI and RFDB results in more similarity in features extracted from the two images. If in future iterations of this system it is determined that the 3B-Sep RFDB occupies too much storage, a 1B RFDB should be used, since it performs better than 3B-Comb; and the results of this test indicate that a band that minimizes the spectral difference between the QI and RFDB should be selected.

### 5.1.3. *Tile Size*

All previous evaluations had been conducted on tiles of size 320 × 320 pixels. The results of this test aim to determine if a relationship exists between the tile size and the resulting system accuracy, and are presented in Figure 4.8.

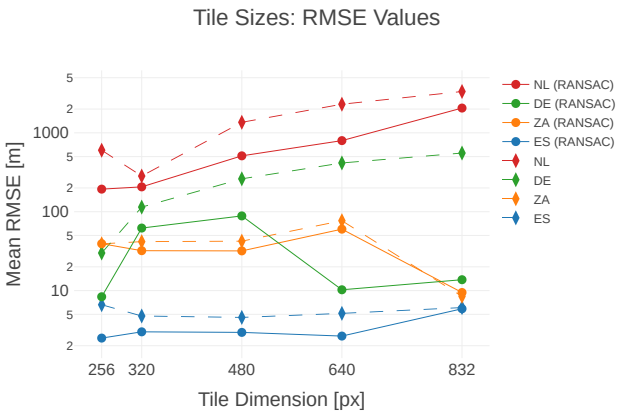



**Figure 4.8:** Mean RMSEs for various tile sizes, applied synchronously to the QI and RFDB

A slight trend towards better accuracies with smaller tiles is apparent when not applying RANSAC, but not significant enough to identify a clear relationship. Note the logarithmic scale on the vertical axis. This was used to help depict the relative improvements that tile-size may have, since certain regions have a consistently higher error than others.

A factor that may influence these raw RMSE values is the spatial distance between GCPs. While the final accuracy value is the metric of choice used to report the system's performance, the purpose of this evaluation is to identify the impact that tile size has on the generation of keypoints. As reported by Babiker and Akhadir [5], GCPs distributed evenly across an image will lower the overall error. Additionally, GCPs spaced far apart (with respect to the SRS) enable the model to achieve sub-pixel accuracies.

To investigate this effect, the results have been normalized relative to the dimension of the tile. Figure 4.9 presents the RMSE as a percentage of the tile's dimension. Although not consistent across all regions, a trend towards better accuracies with larger tiles is apparent. This is consistent with the findings of Liew *et al.* [20].

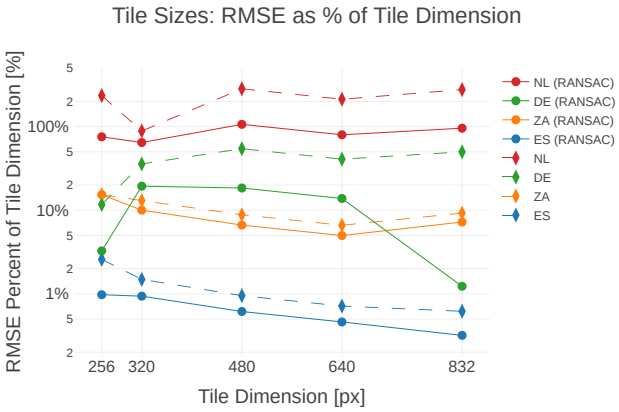



**Figure 4.9:** Mean RMSEs expressed as a percentage of the tile dimension, applied synchronously to the QI and RFDB

### 5.1.4. *Influence of RANSAC*

The effect of applying RANSAC to the system has been analyzed using the result of the previous evaluations. Temporal RANSAC tests (T-5 and T-6) have been summarized and compared to their respective non-RANSAC counterparts.

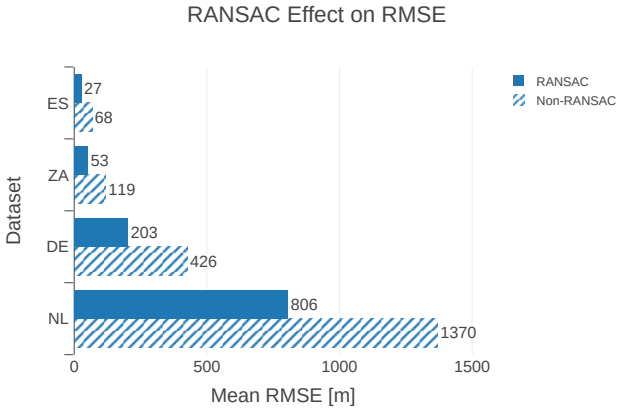



**Figure 4.10:** Effect of applying RANSAC to the matched GCP pairs. RMSE values are means of all previous tests.

Spectral tests have been left out of this evaluation, because the zero-valued 3B-Sep results from the RFDB evaluation are a misrepresentation of the system's actual performance. Furthermore, the spectral differences are not the focus of this evaluation. Clearly, as seen in Figure 4.10 and all previous evaluations, RANSAC provides a significant increase in total georeferencing accuracy. For iterations moving forward and the benchmark comparison to SotA products, the use of RANSAC is essential.

## 5.2. Robustness Evaluation

As all previous tests were conducted on QI tiles with the same geographic footprint as the tiles used to create the RFDB, these evaluations were aimed to establish whether the keypoint model could still accurately extract the same features if the input image was offset from the reference.

The results are presented in Figure 4.11, for a number of increasing offsets. The offsets are in pixels and are in both the northing and easting directions. Other offset combinations are discussed in Section VI. Since a tile size of $320 \times 320$ pixels was used, the offset of $160 \times 160$ corresponds to exactly one half tile, such that the QI overlaps four tiles of the RFDB equally.

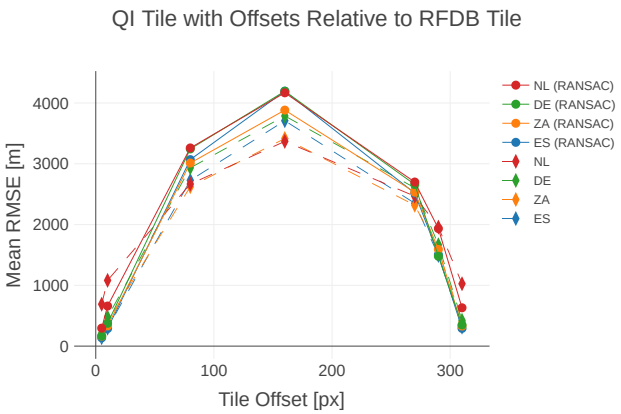



**Figure 4.11:** Impact of off-setting the QI tiles from the RFDBs tiles on the ICP RMSE

It is clear that the georeferencing accuracy of the system is poor for offset tiles, with an increasing offset leading to worse localization. Investigation into why this occurs is detailed below in Section VI. Interesting to note is that in contrast to the baseline evaluations, the RMSE error is higher when RANSAC is applied. This is likely due to the fact that the RANSAC algorithm removes outliers that do not fit the transformation polynomial model. However, the polynomial model has no knowledge of the actual geospatial position of the GCPs, and thus measures a different error than the RMSE of the ICPs.

## 5.3. Hardware Compatibility

### 5.3.1. Memory Usage

A graph of the total memory allocation can be seen in Figure 4.12. The start and end times of significant events are noted on the plot using colored brackets, such as: the initialization of the keypoint model (`__init__`), each QI inference (forward), the keypoints' matching (`brute_force_match`), and the ICP transformation computation including RANSAC (`icp_accuracy`). These are indicated by the colored brackets. Note, the memory usage for the first inference is significantly higher than for subsequent images. This is normal behavior for PyTorch; this is when the system allocates memory for the keypoint model's weights and tensors.

The smaller increases in memory during the first half of the plot correspond to other initialization functions, such as the loading of the QI dataset, setup of diagnostic logfile buffer, allocation of arrays to store the simulation results, and some additional memory used by `psutil` to monitor the usage. Excluding this "extra" simulation memory overhead, the georeferencing pipeline itself (in the current implementation) uses about 3 GB of RAM. This will be discussed further below.

Figure 4.13 depicts a closer inspection of the individual georeferencing steps performed after the keypoints have been extracted. It can be seen that minimal resources are required for matching and computing the polynomial transformation functions. This is because the tensors containing the keypoints and their respective coordinates are generated during inference, and only referenced during the computational steps. Also note the execution times of these steps are extremely short.

### 5.3.2. Disk Storage

**Table 4.3:** RFDB file sizes from a singular Sentinel-2 image (110km by 110km)

| Tile Size [px] | Total Number of Keypoints | RFDB File Size |
|---|---|---|
| $256 \times 256$ | 264600 | 2.2 GB |
| $320 \times 320$ | 173400 | 1.4 GB |
| $480 \times 480$ | 72600 | 602.8 MB |
| $640 \times 640$ | 38400 | 318.9 MB |
| $832 \times 832$ | 21600 | 179.3 MB |

The file size of the RFDB saved as a geopackage is dependent on the number of features extracted from each tile and the number of tiles used to generate the database. Table 4.3 shows the relationship between the tile size, total number of keypoints, and the resulting file size. One hundred keypoints are extracted per band, per tile. The geographic footprint is equal to a single Sentinel-2 Level 2A product which covers a square of 110km by 110km. The result is approximately 8.3 KB per feature.

## VI. DISCUSSION

The following sections contain detailed discussions regarding the results, and how they should be interpreted in the context of the complete system.

[7] See Appendix A for an enlarged version.
[8] See Appendix A for an enlarged version.

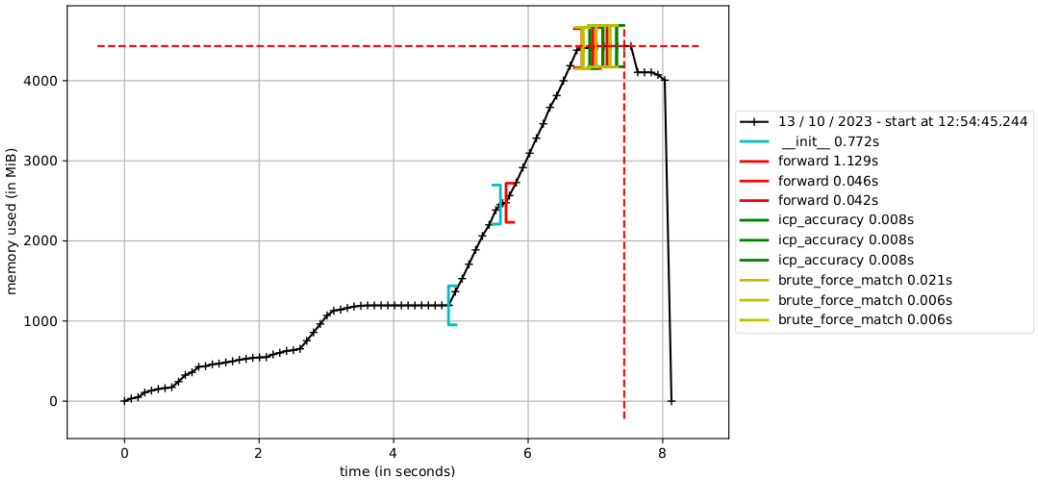



**Figure 4.12:** Total system memory profile of the georeferencing simulation for three QI samples[7]

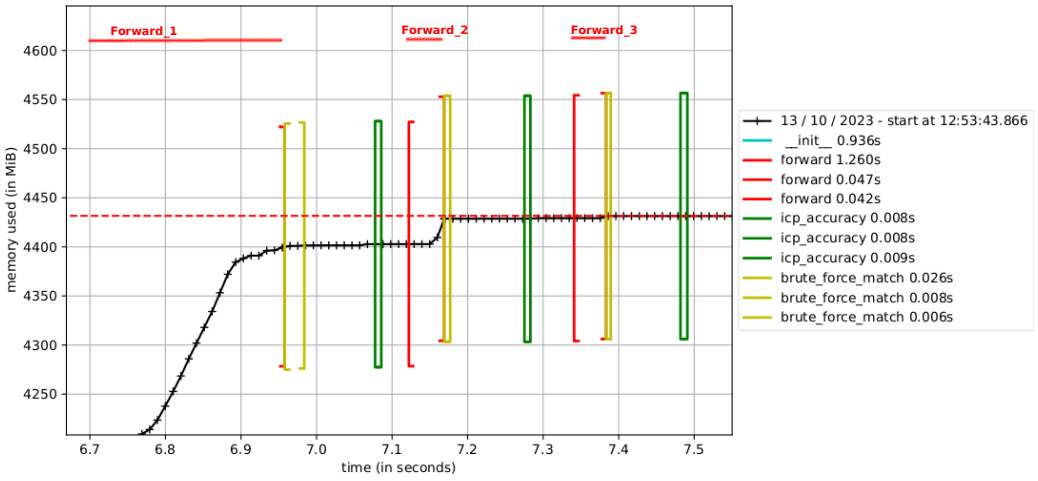



**Figure 4.13:** Close-up on the georeferencing steps in the simulation's memory profile[8]

### 6.1. Comparison to Existing State-of-the-Art Systems

Essential to being able to draw conclusions about the performance of the novel georeferencing system is to compare it to the performance of existing systems. The accuracy of this pipeline was evaluated across the four geographic regions, using the following parameters which have been identified in the baseline evaluation to produce in general the most accurate results: OTS pretrained weights, an RFDB generated using all three bands processed separately, a tile size of 320×320 pixels for both the QI and RFDB and the use of RANSAC to filter out outliers during polynomial computation. Note: although Figure 4.8 indicates that different tile sizes could improve accuracy for specific regions, the same tile size was used for all regions to keep the evaluation consistent.

**Table 4.4:** Comparison of the novel georeferencing system to existing products

| Dataset | Temporal Difference w/ RFDB | RMSE |
|---|---|---|
| NL | 415 days | 194.0m |
| ZA | 15 days | 32.5m |
| DE | 15 days | 13.3m |
| ES | 5 days | 3.5m |
| Landsat 8[9] | - | 12m |
| PRISMA [25] | - | 9.88m (15m CE90) |
| Real-time UAV-based fire detection [34] | - | 16.33m |

These results are compared to existing systems with use cases similar to what this novel system could be applied to: the Landsat 8 OLI and the Hyperspectral PRecursor of the Application Mission, Italian Space Agency (PRISMA) satellite imagery products, as well as a SotA UAV-based real-time wildfire detection system which operates much more accurately than satellite based systems. The results of this comparison are presented in Table 4.4.

PRISMA expresses the accuracy of their images using the CE90 metric. For a comparison to our results, this has been converted to RMSE using Equation 7 [33]. This assumes that their error has a bivariate normal distribution and meets the requirements for the CE90 metric.[10]

$$RMSE_r = \frac{CE90}{1.5175} \tag{7}$$

It is clear that under the idealized conditions, the model performs on par compared to other state-of-the-art systems. However, the current keypoint model will require further improvements in order to be able to perform well under more realistic conditions. Nonetheless, achieving sub-pixel accuracies even within the current scope demonstrates that the proposed architecture of storing features ahead of time and using them for georeferencing with their spatial metadata is indeed possible.

### 6.2. Geographic and Temporal Differences in Datasets

As seen in the results of all the evaluations, the specific QI dataset has a noticeable impact on the resulting georeferencing accuracy. Two aspects that should be inspected are the temporal differences between the QI and RFDB, and the geographic region. From only four datasets it is difficult to draw any conclusions, but some observations can be made.

Referring to Table 4.4, the datasets of Spain and Germany perform the best, with South Africa a close third. Only the dataset of the Netherlands performs significantly worse. This difference is likely due to the much larger temporal difference, 415 days compared to the 5-15 days found in the ES, ZA, and DE datasets. The reason for this discrepancy is the lack of cloud-free days in the Netherlands, and thus the difficulty to obtain overlapping datasets with a specific temporal difference.

Considering the final mission operations of an onboard georeferencing system deployed in space, the ability to upload a RFDB every 15 days is not unrealistic, but also not ideal. Uploading an updated RFDB prior to each capture would reduce the temporal difference during inference, but a system robust to larger temporal differences would significantly reduce the necessary uplink bandwidth and frequency of occurrence. Therefore, the NL dataset was retained to evaluate the system performance over large time differences.

Assessing the geographic differences is more difficult, as each region is a mix of urban, cropland, grassland, and forest. Spain and South Africa could be considered most similar in terms of land cover, but in the results the German dataset outperforms the South African dataset. Whether there is a correlation between geographic region and the accuracy is unknown at this point, and a systematic study with substantially more regions would be required to draw conclusions. This research has prioritized the development of the system architecture, and such a specific study should only be performed once the keypoint extraction model is able to perform well under more realistic conditions. The distribution of datasets does establish that the system's results are repeatable across multiple global locations.

[9] https://landsat.gsfc.nasa.gov/satellites/landsat-8/landsat-8-mission-details/
[9] https://www.eoportal.org/satellite-missions/prisma-hyperspectral
[10] https://earth.esa.int/eogateway/documents/20142/37627/4_4_Sebastien_Saunier_Geometry.pdf/dd36364a-2fca-bd18-6d9c-d7f421934a27

## 6.3. Offset Tiles

The current greatest limitation of the georeferencing system is to accurately extract keypoints from tiles with offset geographic footprints. This is a problem that could be considered a fundamental flaw of the pipeline, if not analyzed and addressed correctly. This analysis is conducted as follows: first the numeric results are analyzed regarding the effect the offset tiles has on the overall accuracy. Then, a qualitative analysis is conducted to visually inspect the effect of the offset and to try to identify the source of the error, and finally a number of potential solutions are discussed, which should be able to resolve the issue.

Referring to the results in Figure 4.11, it's clear that the offset tiles have a significant negative impact on the accuracy of the system. In worst-case conditions, where the QI is equidistant from four RFDB tiles (offset of 160 pixels), the error reaches a maximum of nearly 4km. Given that the QI has a dimension of 3.2 km × 3.2 km, this georeferencing error is worse than the original location estimate.

For the purpose of isolating the offset tile issue, the QI dataset is identical to the dataset used to generate the RFDB; there are no spectral or temporal differences. Thus, theoretically the RMSE error should be zero for all cases.

### 6.3.1. Increasing Offsets

Referring back to Figures 4.3 and 4.4, the QI is georeferenced by matching the red QI keypoints to the blue RFDB keypoints based on their descriptors. The matched keypoints are denoted by green and orange circles around the QI and RFDB keypoints respectively. In Figure 4.14a no orange circles are visible, since there is a perfect match with all the QI keypoints and the green circles are obscuring the orange circles. In the subsequent figures, the offset is gradually increased, and the matches between QI and RFDB keypoints are indicated with connecting lines.

### 6.3.2. Observations

A number of concerns can be observed regarding the samples in Figure 4.14:

1. The match lines between the QI and RFDB appear to be parallel in nearly all cases.
2. The relative position of QI keypoint clusters look to be identical to the distribution of matched RFDB keypoint clusters. For Figure 4.14b this cluster contains all QI keypoints, and for Figure 4.14e these clusters appear on the left and right sides of the QI.
3. Regardless if the offsets are to the north, east, or in both directions, the match lines are parallel and maintain a consistent angle.
4. As the offset increases, the number of non-parallel matches increases, although remains minimal.
5. For small offsets, the matched RFDB points are uniformly towards the lower-left of the QI keypoints. But at larger offsets, this is no longer the case and some RFDB keypoints are also located towards the upper-right.

These observations raise questions regarding the functionality of the georeferencing pipeline and whether this behavior is a result of a systematic error. The georeferencing pipeline has been extensively tested through both unit tests and integration tests. To isolate that these concerns are not a result of an error in the system simulation or georeferencing method, the following sections highlight a few of the verification tests conducted.

### 6.3.3. Issue Isolation Tests

*QI pixel values*

First and foremost, the query image is verified to contain the correct pixel values and is distinctly different from those used by the RFDB. This is verified by a series of five steps:

1. Loading the offset QI from the dataset
2. Plotting the QI to an image file
3. Separately plotting the QI bounds on the map, with ESRI World Imagery as reference
4. Visually verifying that the land features within the bounded region corresponds to the loaded image
5. Plotting the grid of RFDB tiles, to ensure these are indeed offset from the QI tile bounds

*Visualization tool verification*

Second, Figure 4.15 verifies that there are no faults in the mapping process for the visualization tool. The extraction of QI and RFDB keypoints are independent processes, and the spatial distributions of the keypoints in both cases are correct for all offset QIs.

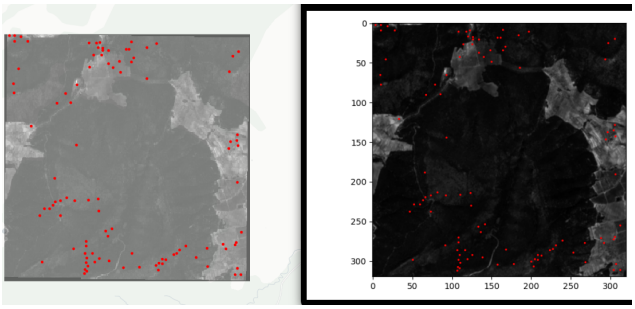
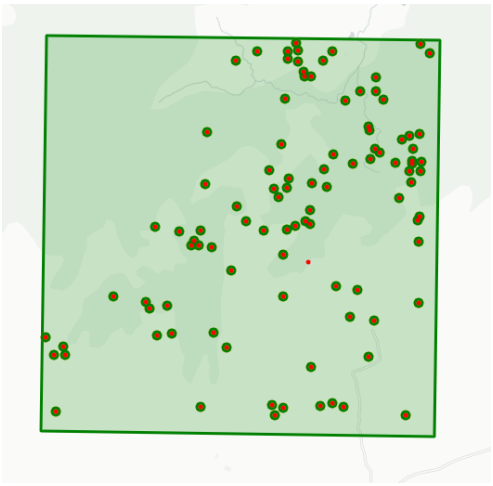
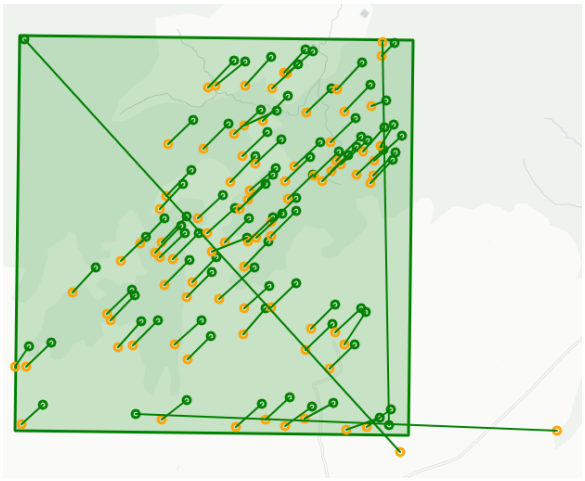



**Figure 4.15:** Verification that the spatial distribution of keypoints on the visualization map is correct, accomplished with two independent plots. Left is the QI and extracted keypoints plotted on the map usingSentinel-2 metadata to transform coordinates from IRS to SRS. Right is the QI plotted as a raster image in the IRS and overlaid with the keypoints. Note the spatial distributions and locations relative to underlying features are identical.

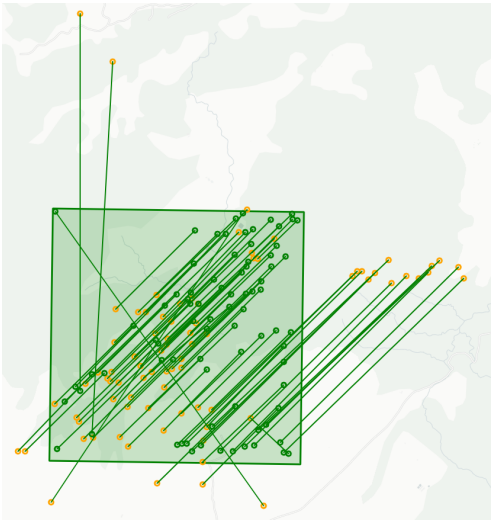
During keypoint extraction, the D2-Net model returns



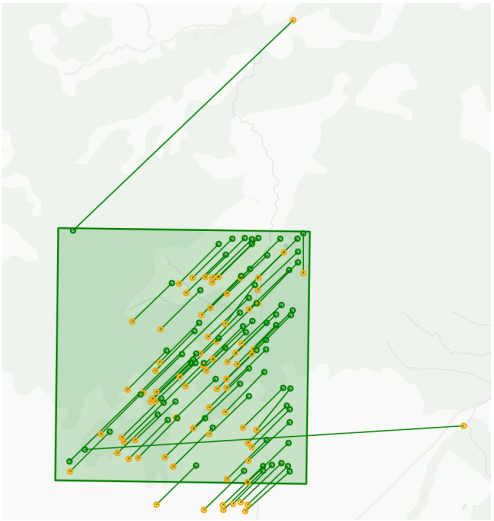
**(a)** No offset



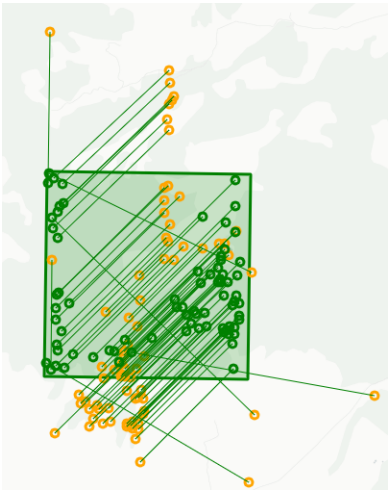
**(b)** 10 × 10 pixel offset



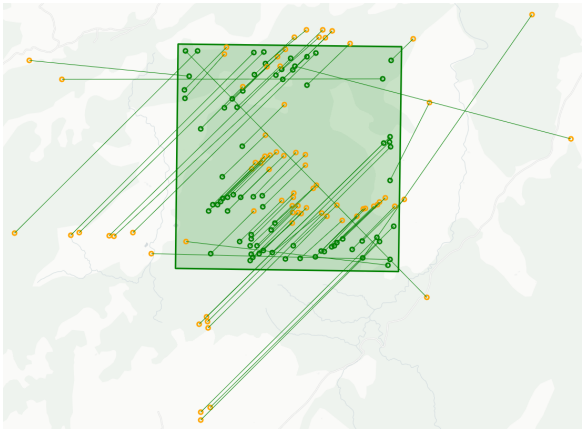
**(c)** 50 × 50 pixel offset



**(d)** 50px north × 0px east offset



**(e)** 150px north × 0px east offset



**(f)** 290 × 290 pixel offset

**Figure 4.14:** QI and RFDB keypoint matches with different offsets

three arrays: descriptors, pixel coordinates and feature scores. Directly after inference, these pixel coordinates are plotted on the input image as red dots, as seen on the right of Figure 4.15. The left map shows the results of the visualization tool: keypoints plotted in the spatial coordinate frame which are obtained using the keypoint pixel coordinates and the Sentinel-2 metadata. The distribution of keypoints is verified to be identical, confirming that the mapping process is correct.

*Clustered matches*

Next, observation 2 is addressed. Having verified that the keypoints extracted from the QI are indeed a result of the correct input data, and that the locations are correct, the question remains as to why the spatial distribution is so similar to that of the RFDB. This can best be explained using Figure 4.16. In the figure, the QI and extracted keypoints are identical to the previous verification step, but the map contains the addition of the RFDB keypoints in blue. The matched RFDB keypoints are highlighted with an orange circle, and the matches are drawn with green lines.

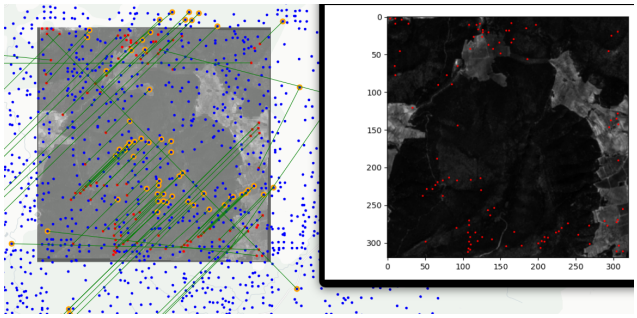


**Figure 4.16:** Addition of RFDB points to the mapping process verification

Since the RFDB is pre-generated independently of the QI, and the positions of mapped keypoints are verified to be correct, the locations of all keypoints present on the map are correct. Thus, the similar spatial distributions between the QI and RFDB keypoints must be attributed to the model.

*Parallel match lines*

Addressing concerns 3 and 4, it's important to note that while the matched lines are close to being parallel, upon close inspection they have variations in both angle and length. This can be clarified by inspecting the error distribution plot of the GCPs (Figure 4.17).

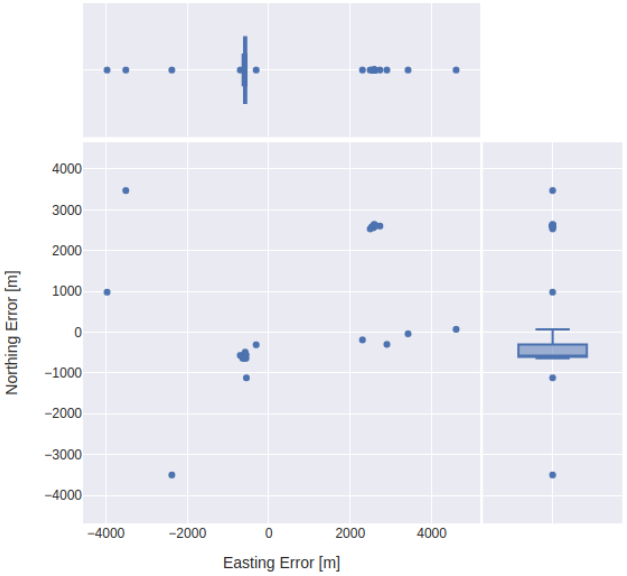



**Figure 4.17:** GCP error distribution of the 290 × 290 offset tile samples

Similar to the RMSE of the ICPs, these errors are the difference in spatial positions of matched keypoints. If all keypoint matches had an identical offset (resulting in perfectly parallel and equal length match lines), this distribution plot would have no variation. Additionally, such a consistent offset would point to a systematic error in the georeferencing process, which is not the case.

The two primary error clusters in Figure 4.17 can be attributed to the two main clusters of matched keypoints, those in the center with relatively short match lines, and those along the edges with longer match lines.

The keypoint pairs with larger match lines appear to refer to clusters originating from other RFDB tiles. This is behavior also seen by concern 5. As the tile offset increases and the QI approaches other RFDB tiles, the matched clusters transition to groups of RFDB keypoints from neighboring tiles.

### *6.3.4. Theory of Underlying Cause*

This consistent behavior of clusters of keypoints being matched to similar clusters in the RFDB forms the basis of the theory of the underlying cause. In essence, the descriptors produced by the keypoint model abstract from the precise features in the landscape. Although the goal of the keypoint extraction model is to produce descriptors that are invariant to minor changes in the image, the model is over-generalizing the landscape and producing descriptors that also implicitly describe their relative position to other keypoints.

In theory, this is a logical outcome of a descriptor. Take for example a simple image of a white square on a black background. A feature extractor such as D2-Net is designed to identify the square as the prominent feature, and will describe this with one or more keypoints.

D2-Net, specifically, will position the keypoints on the corners of the square, using its `DetectionModule`. Since each descriptor will describe the feature as a square, each will need to specify that the white square has corners at certain distances from itself. Since these corners are also extracted as keypoints, the descriptors have thus implicitly described their relative positions to each other.

For a network trained to be invariant to illumination changes (i.e. D2-Net), the reliance on these relative positions becomes extra important. Not all features will be extracted when the lighting changes, and the descriptions themselves will be radically different if based purely on the pixel values. However, relative positions will still be the same, and thus the descriptors will be able to match the same features in different lighting conditions.

The current georeferencing system implementation uses D2-Net model weights trained only on the illumination changes, and thus this effect is likely to be more pronounced than if the model was trained on a more representative dataset containing satellite imagery. Transfer learning to improve the model is a key step described in the future work section.

*Similar distributions of keypoint clusters*

As to why clusters of the QI keypoints have such a similar relative distribution to clusters of keypoints from the RFDB, this is a result of two factors: the poor localization of D2-Net, and the fact that the matched clusters are not identical.

The localization of keypoints extracted by D2-Net is major point of concern, already raised by Zhang *et al.* [35]. During the architecture design of this novel system, this was a factor that was taken into serious consideration when selecting the keypoint model. The localization issue is a result of the max pooling layers in the VGG16 backbone, which are used to reduce the dimensionality of the feature maps. The keypoints extracted by D2-net originate from a $79 \times 79$ pixel feature map (when using a $320 \times 320$ input image), and the final keypoint position and descriptors are interpolated to coordinates of the input image size.

Second, the matched clusters are *not* identical. Since the descriptors are based primarily on the relative position to neighboring descriptors, the keypoint matches are based on clusters with keypoints of similar relative positions. This explains why as the offset increases, the clusters of matched keypoints transition to neighboring RFDB tiles. This is also because with an increased offset, the QI tiles differ significantly from the image tiles used to create the RFDB, and thus the selection of keypoint descriptors is based on a different global feature map. The small number of keypoints matched in seemingly random directions are descriptors that describe other features not included in the Top-K selection, and thus the "clusters" to which they belong are not included in the RFDB.

### 6.4. Small Satellite Hardware Compatibility

The complete georeferencing system has been run on a standard laptop, with an Intel i5-1240P CPU, 16GB of RAM and an NVIDIA GeForce RTX 3050M GPU with 4GB VRAM. The laptop is capable of running the system in real-time, with a processing time of on average 0.19 seconds per image. This processing time includes computing the RMSE of the ICPs, since that is the stage at which RANSAC is applied. If operated on board a satellite, the RMSE computation would be omitted as the ICPs would not be computed. This would slightly reduce the processing load.

The 4 GB of RAM recorded by the memory profile above, is significantly more memory than what is available on board a satellite. However, this value can not be compared directly. First, the current implementation is not optimized for memory efficiency, as it was written in Python to enable rapid development. Any embedded satellite implementation would be written in a lower-level language such as C++, which allows for explicit and efficient direct memory management. Second, the model used in these simulations uses single-precision Floating-Point (FP32) weights, while an onboard implementation would use a half-precision Floating-Point (FP16) model. Converting the model to the lower precision weights would require mixed precision training and fine-tuning, but would halve the memory usage [28]. Third, the size of the input tiles can be reduced if necessary. As tested above, using smaller tiles is not significantly detrimental to the system's performance, and doing so will reduce the required memory footprint.

Regarding the onboard storage compatibility, the RFDB file with a maximum size of 2.2 GB, is easily small enough to be saved onboard the satellite. Furthermore, these values are based on FP32 descriptors. Converting the model (and therefore also the descriptors) to FP16 will approximately halve the storage size. If necessary, the RFDB file can be compressed to reduce the storage requirements even further, at the cost of increased processing time.

Attempts to compile the D2-Net model to run on an Neural Compute Stick (NCS) 2 were so far unsuccessful, but resulted in errors that can likely be solved with further research. The NCS 2 contains a Myriad X VPU, produced by Intel, which is also present on small satellite systems.[11] The VGG16 backbone of the D2-Net model has been proven to run on the Myriad X during radiation tests conducted by Furano *et al.* [12], and by the benchmark tests conducted by Almeida *et al.* [3]. Therefore, it is expected that the D2-Net model should be able to run on the NCS 2.

[11] https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html

Should this prove to be unsuccessful, an alternative embedded architecture can be considered, where the D2-Net is split into two parts: the VGG16 backbone used to produce the descriptors (verified to work on an NCS 2), and the three unique D2-Net layers used to score, interpolate, and select the descriptors as keypoints. The latter three layers are not tested on the NCS 2, but are significantly smaller than the VGG16 backbone, and could be run on the OnBoard Computer (OBC) of a satellite. In a worst-case scenario, the D2-Net model could be replaced with an alternative keypoint extraction model, thanks to the modular architecture of the system.

By analysis, it can therefore be justified that given that the georeferencing simulation requires minimal resources on a standard laptop and the keypoint extraction model is small enough to run on embedded hardware, the system as a whole is compatible to run on the hardware of a small satellite.

### 6.5. ISL Compatibility

Since the georeferencing pipeline is completed on board the satellite, resulting in the final transformation function, the ISL is effectively not required. The only data that would need to be communicated via the ISL to the off-platform end user, is either the coefficients of the polynomial function, so that the receiving party can apply the transformation, or SRS coordinates of a specific point of interest detected by other systems on board the satellite. The latter is the more likely option, as this would be the most useful information to the end user, and could send a simple message containing for example: "Fire detected at (lattitude = 51.989, longitude = 4.375)".

## VII. FUTURE WORK

As identified previously, the keypoint extraction model is the main bottleneck of the system in terms of overall performance. Improving the model's ability to extract keypoints with descriptions more directly dependent on the underlying features should reduce if not eliminate the error regarding offset query images. In theory, this can be achieved via transfer learning, where the model is retrained on a dataset containing satellite imagery.

Following work from previous studies, a self-supervised approach can be used, which leverages co-registered multi-band images to extract features that are robust to spectral changes [2, 18, 35]. Applying this to the retraining of D2-Net, the VGG16 encoder layers should not be frozen, as the model should modify the descriptors it produces to be more representative of the underlying features. In essence, while D2-Net was an improvement over VGG16 in terms of illumination changes because it learned to ignore minute details from the pixels, this retraining process should try to revert that to some extent. The advantage of starting with the `d2-ots` weights is that the model already has a good understanding of extracting salient features from an image, something that most other keypoint extraction models lack.

The localization issue due to the `maxpooling` layers could still pose a challenge in the development of a highly accurate georeferencing pipeline. While it's impossible to predict the exact impact that these layers will have on the model's accuracy after retraining, implementing a completely different keypoint model in the georeferencing system can also be a viable option. Zhang *et al.* [35] claim to have produced a model that successfully detects repeatable keypoints from satellite imagery, and is also robust to spectral differences. Their reported results indicate that nearly all keypoints are retained after RANSAC filtering, which in terms of the overall georeferencing system provides an additional computational benefit. Extracting keypoints that statistically are more likely to be correct, removes the need for the additional RANSAC filtering step.

The work by Li *et al.* [18] uses a feature detection method very similar to D2-Net, and would be a good starting point if retraining does not prove to be successful.

Finally, the loss function used to (re)train the keypoint model should be taken into careful consideration. This field of research is currently under heavy development. A new loss function that correctly optimizes for the multi-spectral and temporal differences in satellite imagery, while ensuring that the keypoints are representative of the underlying data and equally distributed across the image, will ensure the best georeferencing accuracy.

## VIII. CONCLUSION

The novel georeferencing system as a whole has demonstrated exceptional results. With the use of an off-the-shelf keypoint extraction model, the system is able to achieve stat-of-the-art sub-pixel accuracy; and with future transfer learning, the model is expected to perform well under realistic conditions. The goal of this research was to introduce a novel system architecture capable of georeferencing satellite imagery in real-time using onboard hardware. The concept of pre-extracting keypoints and saving these for future matching differs significantly from traditional georeferencing systems, which have access to complete reference images when generating ground control points. The identified limitations of the current implementation in combination with the suggested areas for improvement have established a strong foundation for future research to build upon.

As a result, an object or point of interest detected using state-of-the-art computer vision algorithms on board the satellite can now be reduced to a simple label with the geospatial coordinates. Contrary to full images,

these concise messages have the potential to be transmitted via low-bandwidth inter-satellite communication networks, enabling the insights to be received on the ground within minutes of image capture.

## REFERENCES


[1] B. Abderrazak, D. Morin, G. BENIE, and F. Bonn, "A theoretical review of different mathematical models of geometric corrections applied to remote sensing images," *Remote Sensing Reviews*, vol. 13, Aug. 1, 1995. DOI: 10.1080/02757259509532295.

[2] G. Abdi, F. Samadzadegan, and P. Reinartz, "Spectral-spatial feature learning for hyperspectral imagery classification using deep stacked sparse autoencoder," *Journal of Applied Remote Sensing*, vol. 11, no. 4, 2017, ISSN: 1931-3195. DOI: 10.1117/1.JRS.11.042604.

[3] M. Almeida, S. Laskaridis, I. Leontiadis, S. I. Venieris, and N. D. Lane, "EmBench: Quantifying Performance Variations of Deep Neural Networks across Modern Commodity Devices," in *The 3rd International Workshop on Deep Learning for Mobile Systems and Applications*, Jun. 13, 2019, pp. 1–6. DOI: 10.1145/3325413.3329793. arXiv: 1905.07346 [cs, stat].

[4] "ASPRS Positional Accuracy Standards for Digital Geospatial Data," *Photogrammetric Engineering & Remote Sensing*, vol. 81, no. 3, pp. 1–26, Mar. 1, 2015, ISSN: 00991112. DOI: 10.14358/PERS.81.3.A1-A26.

[5] M. E. A. Babiker and S. K. Y. Akhadir, "The Effect of Densification and Distribution of Control Points in the Accuracy of Geometric Correction," *International Journal of Scientific Research in Science, Engineering and Technology*, vol. 2, no. 1, pp. 65–70, Jan. 16, 2016, ISSN: 2394-4099.

[6] S. Bakken *et al.*, "HYPSO-1 CubeSat: First Images and In-Orbit Characterization," *Remote Sensing*, vol. 15, no. 3, p. 755, 3 Jan. 2023, ISSN: 2072-4292. DOI: 10.3390/rs15030755.

[7] J. Bian *et al.* "MatchBench: An Evaluation of Feature Matchers." arXiv: 1808.02267 [cs]. (Aug. 7, 2018), [Online]. Available: http://arxiv.org/abs/1808.02267, preprint.

[8] D. DeTone, T. Malisiewicz, and A. Rabinovich. "SuperPoint: Self-Supervised Interest Point Detection and Description." arXiv: 1712.07629 [cs]. (Apr. 19, 2018), [Online]. Available: http://arxiv.org/abs/1712.07629, preprint.

[9] M. Dusmanu *et al.* "D2-Net: A Trainable CNN for Joint Detection and Description of Local Features." arXiv: 1905.03561 [cs]. (May 9, 2019), [Online]. Available: http://arxiv.org/abs/1905.03561, preprint.

[10] European Space Agency, *Copernicus Sentinel-2 MSI Level-2A BOA Reflectance*, European Space Agency, 2021. DOI: 10.5270/S2_-znk9xsj.

[11] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1, 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[12] G. Furano *et al.*, "Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities," *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, pp. 44–56, Dec. 2020, ISSN: 1557-959X. DOI: 10.1109/MAES.2020.3008468.

[13] H. Halmaoui and A. Haqiq, "Feature matching for 3D AR: Review from handcrafted methods to deep learning," *International Journal of Hybrid Intelligent Systems*, vol. 17, no. 3-4, pp. 143–162, Apr. 22, 2022, ISSN: 14485869, 18758819. DOI: 10.3233/HIS-220001.

[14] Z. E. Hussein, "Image Georeferencing using Artificial Neural Network Compared with Classical Methods," *Iraqi Journal of Science*, pp. 5024–5034, Dec. 30, 2021. DOI: 10.24996/ijs.2021.62.12.38.

[15] A. Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon. "A Survey on Contrastive Self-supervised Learning." arXiv: 2011.00362 [cs]. (Feb. 7, 2021), [Online]. Available: http://arxiv.org/abs/2011.00362, preprint.

[16] M. Keller, Z. Chen, F. Maffra, P. Schmuck, and M. Chli, "Learning Deep Descriptors with Scale-Aware Triplet Networks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT: IEEE, Jun. 2018, pp. 2762–2770, ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00292.

[17] A. Kothandhapani and V. Vatsal, "Methods to Leverage Onboard Autonomy in Remote Sensing," presented at the 2nd National Conference on Small Satellite Technology & Applications, Thiruvananthapuram, Dec. 12, 2020.

[18] L. Li, L. Han, H. Cao, and M. Liu, "A Self-Supervised Keypoint Detection Network for Multimodal Remote Sensing Images," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLIII-B2-2022, pp. 609–615, May 30, 2022, ISSN: 1682-1750. DOI: 10.5194/isprs-archives-XLIII-B2-2022-609-2022.

[19] "Chapter 2 - Geometric processing and positioning techniques," in *Advanced Remote Sensing: Terrestrial Information Extraction and Applications*, S. Liang and J. Wang, Eds., Second edition, Amsterdam: Academic Press, 2020, pp. 59–105, ISBN: 978-0-12-815826-5.

[20] L. H. Liew, Y. C. Wang, and W. S. Cheah, "Evaluation of Control Points' Distribution on Distortions and Geometric Transformations for Aerial Images Rectification," *Procedia Engineering*, International Symposium on Robotics and Intelligent Sensors 2012 (IRIS 2012), vol. 41, pp. 1002–1008, Jan. 1, 2012, ISSN: 1877-7058. DOI: 10.1016/j.proeng.2012.07.275.

[21] P. Lindenberger, P.-E. Sarlin, and M. Pollefeys. "LightGlue: Local Feature Matching at Light Speed." arXiv: 2306.13643 [cs]. (Jun. 23, 2023), [Online]. Available: http://arxiv.org/abs/2306.13643, preprint.

[22] D. Liu, G. Zhou, D. Zhang, X. Zhou, and C. Li, "Ground Control Point Automatic Extraction for Spaceborne Georeferencing Based on FPGA," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, pp. 1–1, Jun. 1, 2020. DOI: 10.1109/JSTARS.2020.2998838.

[23] D. Liu *et al.*, "On-Board Georeferencing Using FPGA-Based Optimized Second-Order Polynomial Equation," *Remote Sensing*, vol. 11, p. 124, Jan. 10, 2019. DOI: 10.3390/rs11020124.

[24] S. Liu *et al.*, "Rethinking of learning-based 3D keypoints detection for large-scale point clouds registration," *International Journal of Applied Earth Observation and Geoinformation*, vol. 112, p. 102 944, Aug. 1, 2022, ISSN: 1569-8432. DOI: 10.1016/j.jag.2022.102944.

[25] E. Lopinto *et al.*, "Current Status and Future Perspectives of the PRISMA Mission at the Turn of One Year in Operational Usage," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, Jul. 2021, pp. 1380–1383. DOI: 10.1109/IGARSS47720.2021.9553301.

[26] J. Ma, X. Jiang, A. Fan, J. Jiang, and J. Yan, "Image Matching from Handcrafted to Deep Features: A Survey," *International Journal of Computer Vision*, vol. 129, Jan. 1, 2021. DOI: 10.1007/s11263-020-01359-2.

[27] Mariusz E. Grøtte *et al.*, "Ocean Color Hyperspectral Remote Sensing With High Resolution and Low Latency—The HYPSO-1 CubeSat Mission," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–19, Jun. 4, 2021, ISSN: 1558-0644. DOI: 10.1109/TGRS.2021.3080175.

[28] P. Micikevicius *et al.* "Mixed Precision Training." arXiv: 1710.03740 [cs, stat]. (Feb. 15, 2018), [Online]. Available: http://arxiv.org/abs/1710.03740, preprint.

[29] M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014, ISSN: 1939-3539. DOI: 10.1109/TPAMI.2014.2321376.

[30] R. Müller, T. Krauß, M. Schneider, and P. Reinartz, "Automated Georeferencing of Optical Satellite Data with Integrated Sensor Model Improvement," *Photogrammetric Engineering & Remote Sensing*, vol. 78, no. 1, pp. 61–74, Jan. 1, 2012. DOI: 10.14358/PERS.78.1.61.

[31] Y. Ren, Y. Liu, Z. Huang, W. Liu, and W. Wang, "2chAD-CNN: A Template Matching Network for Season-Changing UAV Aerial Images and Satellite Imagery," *Drones*, vol. 7,

no. 9, p. 558, 9 Sep. 2023, ISSN: 2504-446X. DOI: 10.3390/drones7090558.

[32] J. Revaud *et al.* "R2D2: Repeatable and Reliable Detector and Descriptor." arXiv: 1906.06195 [cs]. (Jun. 17, 2019), [Online]. Available: http://arxiv.org/abs/1906.06195, preprint.

[33] K. Ross, Vicki Zanoni, Tom Stanley, Mary Pagnutti, and Charles Smith, "Geopositional Statistical Methods," in *Proceedings of the 2004 High Spatial Resolution Commercial Imagery Workshop*, St Louis, MS, United States: John C. Stennis Space Center, NASA, Jan. 30, 2006.

[34] B. Santana, E. K. Cherif, A. Bernardino, and R. Ribeiro, "Real-Time Georeferencing of Fire Front Aerial Images Using Iterative Ray-Tracing and the Bearings-Range Extended Kalman Filter," *Sensors*, vol. 22, no. 3, p. 1150, 3 Jan. 2022, ISSN: 1424-8220. DOI: 10.3390/s22031150.

[35] X. Zhang *et al.*, "Distinguishable keypoint detection and matching for optical satellite images with deep convolutional neural networks," *International Journal of Applied Earth Observation and Geoinformation*, vol. 109, p. 102 795, May 1, 2022, ISSN: 1569-8432. DOI: 10.1016/j.jag.2022.102795.

[36] G. Zhou, R. Zhang, N. Liu, J. Huang, and X. Zhou, "On-Board Ortho-Rectification for Images Based on an FPGA," *Remote Sensing*, vol. 9, no. 9, p. 874, 9 Sep. 2017, ISSN: 2072-4292. DOI: 10.3390/rs9090874.

# 5

# System Design Results

The current implementation of the proposed system architecture is one possible solution to the proposed novel system architecture. Although it has some specific challenges, it has certainly demonstrated the potential for this novel georeferencing approach. The following sections evaluate the overall system architecture and design. As part of the verification steps found on the right-hand side of the V-model (Figure 3.1), the system requirements are evaluated below to determine if the system meets the original specifications, based on the results of this implementation.

Throughout the development of the design, design choices were made based on best judgements and supporting literature. However, the final system still contains many parameters and aspects that should be tested and experimentally evaluated. Furthermore, idealizations used in this research, such as the use of idealized data, should also be replaced by realistic parameters. These aspects are ranked by their expected return in Section 5.2.

## 5.1. Requirements Verification

The verification of the system requirements established in Section 3.3 is performed based on the selected architecture and the results of the experimental evaluations. The results are summarized in Table 5.1, with a rationale for each requirement.

Had the results of the implementation been negative, such that the system did not attain reasonable accuracies even with idealized data or the measured resource usage was too high, the requirements would have been marked as "Failed". Likewise, if the suggested improvements do not succeed in fixing the D2-Net localization issues, an alternative method would need to be considered. The first step would be to try an implementation using a different lightweight feature extraction model, such as the DRRD model, which could reduce the resource consumption or improve the accuracy. Other recommendations are listed in the following section.

Requirements FUNC-2 and PERF-1 are marked as "To Be Confirmed" (TBC) since analysis of the experimental results strongly indicates the requirement will be met, but confirmation by testing on representative hardware is needed. PERF-2 on the other hand is marked "To be determined" (TBD), because while the current system implementation with the pre-trained D2-Net model produces promising results, it requires further development and fine-tuning before a decision can be made. As noted previously, the nature of the architecture of the system is such that the exact implementation is not fixed, and elements such as the feature extraction model, RFDB backend, and GCP matching algorithm can be replaced to create a more optimal system. The current implementation is a proof-of-concept. The results are promising, but further development is required to confirm the exact performance of the system.

**Table 5.1:** System requirements verification

| Requirement | Met | Rationale |
|---|---|---|
| FUNC-1 | Yes | Demonstrated with the experimental evaluations. |
| FUNC-2 | TBC | Analysis suggests the system can run on a small satellite, but requires testing on representative hardware to confirm. See the results discussion section 6.4 of the research paper for more details. |
| FUNC-2.1 | TBC | Analysis of memory usage indicates the requirement is very likely to be met when tested with an optimized model. See the results discussion section 6.4 of the research paper for more details. |
| FUNC-2.2 | TBC | Not tested on representative hardware, but analysis suggests this requirement will be met. See the results discussion of the research paper: section 6.4. |
| FUNC-2.3 | Yes | Storage requirements of the RFDB are less than 8 GB for all tile sizes (using a 110km by 110km region). If FP16 is used for the RFDB descriptors, the storage for tiles of size 256 by 256 pixels also reduces to 1.1 GB, which is within the requirement's 2 GB target value. |
| FUNC-3 | Yes | The complete georeferencing process is executed on board, and the child requirement has been met. |
| FUNC-3.1 | Yes | The final result consists of 12 coefficients of the polynomial transformation function. An array of 12 FP32 values would have a size of 48 bytes, which is well within the 1 kB per minute requirement. Using FP16 values will further reduce this to 24 bytes. |
| PERF-1 | TBC | On a consumer laptop the system has been tested to run in a fraction of a second; analysis of resource usage strongly suggests system will be produce a georeferenced result within 5 minutes on satellite hardware. |
| PERF-2 | TBD | The final system architecture shows promising results, but the model requires fine-tuning. Current implementation (using idealized data) meets the accuracy of SotA products. Further research is required to confirm if this also applies to non-idealized data. |
| PERF-2.1 | TBD | Under ideal conditions the system achieves sub-pixel accuracies of up to 3.5m. This needs to be tested with an improved feature extraction model on realistic data. |

## 5.2. Recommendations for Future Work

In addition to the specific topics recommended in the research paper to improve the keypoint model of this specific implementation, there are also areas of research that should be investigated to improve the overall georeferencing pipeline. From a complete space systems' engineering perspective, many other factors besides the keypoint extraction play a role in the overall performance of the system. A number of aspects requiring dedicated research have been compiled throughout the development and evaluation of this novel system. Table 5.2 summarizes these areas of research and ranks them based on the ratio of their expected return in improving the system's performance versus their estimated effort to implement/execute. The following subsections describe each of these topics in more detail.

**Table 5.2:** Ranking of topics for recommended future research

| **Item** | **Expected Return** | **Estimated Effort** | **$\frac{\text{Return}}{\text{Effort}}$** |
|---|---|---|---|
| Cloud / coastal masks | 5 | 1 | 5.00 |
| Neighboring tile consensus | 5 | 2 | 2.50 |
| Match on threshold | 4 | 2 | 2.00 |
| Dataset variety | 3 | 2 | 1.50 |
| Feature selection methods | 4 | 3 | 1.33 |
| Non-affine transformations | 4 | 3 | 1.33 |
| Remove grid artifacts in RFDB | 4 | 4 | 1.00 |
| Raw satellite imagery | 5 | 5 | 1.00 |
| Universal reference features | 4 | 4 | 1.00 |
| Feature descriptor size | 3 | 3 | 1.00 |
| Image preprocessing | 2 | 4 | 0.50 |
| Terrain effects | 1 | 4 | 0.25 |
| Land class effects | 1 | 4 | 0.25 |

#### Cloud / coastal masks

Concerning non-idealized data, one of the most common challenges in satellite imagery is clouds. Given that onboard cloud masking is already feasible, applying this as a preprocessing step, in combination with the adaptive approach mentioned above, could be a very effective way to improve the overall performance of the system. The same could be applied using coastal masks, which could be used to exclude tiles that contain a large amount of water. Water or clouds create artificial edges in an image, which confuse the keypoint detection algorithms.

#### Neighboring tile consensus

Currently, the system splits a captured image into smaller tiles, each for which the entire georeferencing pipeline is executed. Each tile will therefore have its own transformation function, based on the keypoints from that specific tile. However, from a system-wide perspective, the transformation function of a single image should be relatively uniform. This is because the image is captured at one moment from a single perspective, and the transformation function should therefore be the same for each tile. An adaptive or coordinated approach could be considered, in which the transformation functions are still computed per tile, but a consensus between the results from each tile could be used to remove outliers. This would be very effective for tiles containing difficult data, such as clouds or coastal areas.

#### Match on threshold

Instead of matching all extracted keypoints using the brute force matcher, setting a threshold value such that only those with high enough similarity to the reference features are selected as matches. The current implementation only retains the 30 best matches, but this could be modified to be dynamically select only the best matches.

#### Dataset variety

An experimental study of using other dataset sources, other than just Sentinel-2 is recommended. Applying a variety of data during training will help the model to generalize better. This can be an intermediate step to test the system on more challenging data before transitioning to raw unprocessed satellite imagery.

#### Feature selection methods

Instead of simply selecting the Top-K feature from each image, using a minimum score threshold to only select the most distinguishable keypoints could prove more effective. This could help in situations where the QI does not contain many features compared to other tiles.

#### Non-affine transformations

Once the georeferencing keypoint model has been improved regarding the offset tiles, and is able to perform on multi-scale images, the next step is to assess its robustness to non-affine transformations, such as perspective changes or local warping caused by terrain. Most current keypoint extraction models such as D2-Net already take this into account. In cityscape images, such as those taken from a car driving down a street, the primary challenge in keypoint matching is the differences in perspective. Regarding satellite imagery, since the distance between the imager and the surface is on a much larger scale, the effects would more likely be apparent in deformed ground features. Whether a keypoint model pretrained on cityscapes will be robust to these deformations needs to be investigated. This would also introduce the orthorectification process into the georeferencing pipeline, which is currently part of the basic image processing pipeline.

#### Remove grid artifacts in RFDB

As seen in Figure 4.4, the current keypoint model has a tendency to extract features near the edges of the image. This results in keypoint locations retaining artifacts of the regular tile grid, while on the contrary the RFDB should abstract from this and only return the most distinctive keypoints. Modifications should be researched into methods to ensure a relatively uniform distribution of keypoints across the image. For example an improved loss function that takes this into account, or extra filter layers.

During development of this pipeline, some preliminary experimentation was performed with using the Suppression via Square Covering (SSC) algorithm proposed by Bailo *et al.* [5]. Figure 5.1 demonstrates how this algorithm prevents clustering by specifying a minimum bounding box around each keypoint. However, caution must be taken that the suppression of keypoints due to the spatial position does not also suppress highly descriptive and significant keypoints representing underlying salient features.

**Figure 5.1:** Preliminary experimentation with SSC algorithm on QI keypoints

### Raw satellite imagery

Once the system performs well on all types of pre-processed satellite imagery, retraining the keypoint model to function on raw satellite imagery enables it to work directly on board. The raw satellite images have significant radiometric differences from the current images being tested, thus retraining is needed to adapt the model to work with these as well. An added challenge to this process is obtaining sufficient amounts of training data, and ensuring that the individual bands are co-registered such that they can be used for the self-supervised training pipeline.

### Universal reference features

Similar to the RFDB Bands evaluation, more advanced methods of creating the reference features should be explored. Examples could be combining feature descriptors from multiple datasets with temporal differences, or preprocessing the input images to combine the bands or temporal differences. Depending on how the model is trained, composite reference features could form a generalized representation of the underlying land features, which would help match new QI features.

### Feature descriptor size

Currently, D2-Net returns a feature descriptor of 512 values, but U-Net for example only uses 16 values. Reducing this dimension would reduce computational load, but the effect on matching performance is unknown.

### Image preprocessing

The effect of normalizing or otherwise pre-processing the satellite images before being passed to the keypoint model should be investigated. Especially in the case of raw images, simple filters or digital number normalization technique could be required to standardize the input into the model, in order to obtain consistent results. Precisely what and how this should be implemented will require a systematic and dedicated study.

### Terrain effects

An experimental study regarding the effects of variable terrain on the system performance should be conducted. This is a prerequisite to retraining on raw data, since orthorectification is a crucial step in the pre-processing of raw images. Theoretically, if the keypoints in the RFDB have altitude information included with their spatial coordinates, the orthorectification step will be conducted automatically. Terrain effects will likely also have overlap with the non-affine transformation research.

### Land cover effects

Last but not least, as noted previously, a dedicated study of the impact that land classes have on the overall georeferencing accuracy is needed. This should only be executed once a performant model has been obtained. This should investigate the effects of challenging land cover types, such as large water bodies, ice, or sand, which result in images with minimal unique or temporally persistent textures. The results of this study will enable the novel georeferencing system to effectively be applied to suitable space missions.

# 6

# Conclusion

Based on recent developments in computer vision and object recognition, the ability to preprocess images on board a satellite has emerged in recent years, enabling the prioritization of data to be downlinked for processing and thus significantly increasing a mission's value. However, the transfer of images to ground stations still remains a bottleneck for applications in which the timeliness of the insights is critical. This research has analyzed this technical challenge, and has proposed a solution with high potential of success. Using an architecture which enables the images to be georeferenced on board, the need for the images to be transferred off the satellite platform is removed entirely.

This research was driven by the primary research question, with two sub-questions to further steer the research and emphasize which aspects of the system needed to addressed.

> *How can an image georeferencing pipeline for small satellites be distributed between onboard and terrestrial computing resources to enable real-time results?*

1. How can the various elements of the georeferencing pipeline best be distributed between onboard and terrestrial resources, while still meeting the data downlink limitations imposed by an ISL network?
2. How does the spatial accuracy of this pipeline compare to what is achieved with current georeferencing systems?

The first sub-question drove the research to follow a systems' engineering approach. The potential use cases for such a georeferencing system were identified, as well as the technical challenges and limitations imposed by small satellites and current-generation ISL networks. The system requirements were derived from this, and a systematic design process was followed to generate a complete set of design options. Two main design phases were used: the first to identify which general georeferencing methodology to follow, direct or indirect; and the second to develop a specific architecture for the chosen methodology. As an answer to this question, the optimal distribution of the pipeline elements is one as follows: the reference keypoints are first pre-extracted using terrestrial resources and uploaded to the satellite, the image is then captured and the keypoints of this image are extracted using an onboard state-of-the-art deep learning model. The keypoints can then be matched and used to estimate the geopositional information of the image.

In order to demonstrate the feasibility of this approach, and answer the second sub-question, a specific implementation of this architecture was developed. This proof-of-concept is based on the D2-Net, which was selected as a keypoint extractor due to its state-of-the-art performance and low computational requirements. This implementation was evaluated using a variety of Sentinel-2A datasets, in order to establish its spatial accuracy. By changing model parameters such as the image size, band combinations, and applying RANSAC, the limitations and overall performance of the implementation was analyzed. Furthermore, compared to current georeferencing systems, this novel system achieves the same performance, confirming its feasibility as a proposed architecture. However, some limitations regarding offset tiles were also identified, and specific recommendations for future work were proposed to address these issues.

Because this research was focused on developing a first-order architecture design, it is not unexpected that the current iteration has limitations. The development intentionally used idealized test data, in order to ensure the systematic development of a baseline georeferencing system not plagued by problematic input data

such as cloud coverage, atmospheric effects, and geometric distortions. Naturally, these idealizations must be eliminated in subsequent design iterations before the system can be considered for operational use. Architectural improvements to address these were identified throughout the research and have also been detailed for future research.

Regarding the system requirements established in the preliminary phases of this research, half have been fully met, and the other half require confirmation with non-idealized data or representative hardware. Important to note is that the current architecture does not fail any of the requirements, suggesting that the design is successful and has significant potential for future development. As a result, an object or point of interest detected using state-of-the-art computer vision algorithms on board the satellite, can now be reduced to a simple label with the geospatial coordinates. Contrary to full images, these concise messages have the potential to be transmitted via low-bandwidth inter-satellite communication networks, enabling the insights to be received on the ground within minutes of image capture.

# References


[1] AAC Clyde Space, *Kryten-M3 Datasheet*, Jul. 28, 2020.

[2] M. A. Aguilar, A. Nemmaoui, F. J. Aguilar, A. Novelli, and A. García Lorca, "Improving georeferencing accuracy of Very High Resolution satellite imagery using freely available ancillary data at global coverage," *International Journal of Digital Earth*, vol. 10, no. 10, pp. 1055–1069, Oct. 3, 2017, ISSN: 1753-8947. DOI: 10.1080/17538947.2017.1280549.

[3] S. H. Alizadeh Moghaddam, M. Mokhtarzade, A. Alizadeh Naeini, and A. Amiri-Simkooei, "A Statistical Variable Selection Solution for RFM Ill-Posedness and Overparameterization Problems," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 7, pp. 3990–4001, Jul. 2018, ISSN: 1558-0644. DOI: 10.1109/TGRS.2018.2819136.

[4] "ASPRS Positional Accuracy Standards for Digital Geospatial Data," *Photogrammetric Engineering & Remote Sensing*, vol. 81, no. 3, pp. 1–26, Mar. 1, 2015, ISSN: 00991112. DOI: 10.14358/PERS.81.3.A1-A26.

[5] O. Bailo, F. Rameau, K. Joo, J. Park, O. Bogdan, and I. S. Kweon, "Efficient adaptive non-maximal suppression algorithms for homogeneous spatial keypoint distribution," *Pattern Recognition Letters*, vol. 106, pp. 53–60, Apr. 15, 2018, ISSN: 0167-8655. DOI: 10.1016/j.patrec.2018.02.020.

[6] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Computer Vision and Image Understanding*, Similarity Matching in Computer Vision and Multimedia, vol. 110, no. 3, pp. 346–359, Jun. 1, 2008, ISSN: 1077-3142. DOI: 10.1016/j.cviu.2007.09.014.

[7] J. Bian *et al.* "MatchBench: An Evaluation of Feature Matchers." arXiv: 1808.02267 [cs]. (Aug. 7, 2018), [Online]. Available: http://arxiv.org/abs/1808.02267, preprint.

[8] A. Bouillon, E. Breton, F. De Lussy, and R. Gachet, "SPOT5 geometric image quality," in *IGARSS 2003. 2003 IEEE International Geoscience and Remote Sensing Symposium. Proceedings (IEEE Cat. No.03CH37477)*, vol. 1, Jul. 2003, 303–305 vol.1. DOI: 10.1109/IGARSS.2003.1293757.

[9] S. Cai, L. Liu, S. Yin, R. Zhou, W. Zhang, and S. Wei, "Optimization of speeded-up robust feature algorithm for hardware implementation," *Science China Information Sciences*, vol. 57, no. 4, pp. 1–15, Apr. 1, 2014, ISSN: 1869-1919. DOI: 10.1007/s11432-013-4946-y.

[10] J. Li-Chee-Ming and C. Armenakis, "Fusion of optical and terrestrial laser scanner data," presented at the Canadian Geomatics Conference and Symposium of Commission I, ISPRS Convergence in Geomatics-Shaping Canada's Competitive Landscape, vol. 38, Calgary, Alberta, Canada, May 8, 2012, pp. 15–18.

[11] Q. Chen, M. Sun, X. Hu, and Z. Zhang, "Automatic Seamline Network Generation for Urban Orthophoto Mosaicing with the Use of a Digital Surface Model," *Remote Sensing*, vol. 6, pp. 12 334–12 359, Dec. 1, 2014. DOI: 10.3390/rs61212334.

[12] D. DeTone, T. Malisiewicz, and A. Rabinovich. "SuperPoint: Self-Supervised Interest Point Detection and Description." arXiv: 1712.07629 [cs]. (Apr. 19, 2018), [Online]. Available: http://arxiv.org/abs/1712.07629, preprint.

[13] DigitalGlobe, "Accuracy of Worldview Products," DigitalGlobe, White Paper, 2014.

[14] M. Dusmanu *et al.* "D2-Net: A Trainable CNN for Joint Detection and Description of Local Features." arXiv: 1905.03561 [cs]. (May 9, 2019), [Online]. Available: http://arxiv.org/abs/1905.03561, preprint.

[15] European Space Agency, *Copernicus Sentinel-2 MSI Level-2A BOA Reflectance*, European Space Agency, 2021. DOI: 10.5270/S2_-znk9xsj.

[16] European Space Agency. "SPOT 5 - Instruments," ESA Earth Online. (), [Online]. Available: https://earth.esa.int/eogateway/missions/spot-5.

[17] European Space Agency. "Technology Readiness Levels (TRL)," ESA Enabling Support. (), [Online]. Available: https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Shaping_the_Future/Technology_Readiness_Levels_TRL.

[18] European Space Agency. "WorldView-4 - Instruments," ESA Earth Online. (), [Online]. Available: https://earth.esa.int/eogateway/missions/worldview-4.

[19] European Space Agency and Copernicus. "Sentinel-2 MSI: MultiSpectral Instrument, Level-1C | Earth Engine Data Catalog," Earth Engine Data Catalog. (Feb. 19, 2023), [Online]. Available: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2.

[20] X. Fan *et al.*, "Estimating earthquake-damage areas using Landsat-8 OLI surface reflectance data," *International Journal of Disaster Risk Reduction*, vol. 33, pp. 275–283, Feb. 1, 2019, ISSN: 2212-4209. DOI: 10.1016/j.ijdrr.2018.10.013.

[21] G. Furano *et al.*, "Towards the Use of Artificial Intelligence on the Edge in Space Systems: Challenges and Opportunities," *IEEE Aerospace and Electronic Systems Magazine*, vol. 35, no. 12, pp. 44–56, Dec. 2020, ISSN: 1557-959X. DOI: 10.1109/MAES.2020.3008468.

[22] K. D. Goepel, "Implementation of an Online Software Tool for the Analytic Hierarchy Process (AHP-OS)," *International Journal of the Analytic Hierarchy Process*, vol. 10, no. 3, Dec. 6, 2018, ISSN: 1936-6744. DOI: 10.13033/ijahp.v10i3.590.

[23] H. Halmaoui and A. Haqiq, "Feature matching for 3D AR: Review from handcrafted methods to deep learning," *International Journal of Hybrid Intelligent Systems*, vol. 17, no. 3-4, pp. 143–162, Apr. 22, 2022, ISSN: 14485869, 18758819. DOI: 10.3233/HIS-220001.

[24] S. Hirshorn, *NASA Systems Engineering Handbook*, Rev2. Washington DC: NASA Aeronautics Research Mission Directorate (ARMD), 2016.

[25] E. Hoffer and N. Ailon. "Deep metric learning using Triplet network." arXiv: 1412.6622 [cs, stat]. (Dec. 20, 2014), [Online]. Available: http://arxiv.org/abs/1412.6622, preprint.

[26] ISIS - Innovative Solutions In Space B.V., *ISIS on-board computer (iOBC) Datasheet*, 2023.

[27] E. Karami, S. Prasad, and M. Shehata, "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images," p. 5,

[28] W. J. Larson, Ed., *Applied Space Systems Engineering* (Space Technology Series). Boston, Mass.: McGraw-Hill Learning Solutions, 2009, 895 pp., ISBN: 978-0-07-340886-6.

[29] S.-K. Lee and S.-W. Shin, "A Study on the Interior Orientation for Various Image Formation Sensors," *Korean Journal of Geomatics*, vol. 4, no. 1, pp. 23–30, 2004, ISSN: 1598-4699.

[30] K. Legat, "Approximate direct georeferencing in national coordinates," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 60, no. 4, pp. 239–255, Jun. 1, 2006, ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2006.02.004.

[31] C. Li, X. Liu, Y. Zhang, and Z. Zhang, "A Stepwise-then-Orthogonal Regression (STOR) with quality control for Optimizing the RFM of High-Resolution Satellite Imagery," *Photogrammetric Engineering & Remote Sensing*, vol. 83, no. 9, pp. 611–620, Sep. 1, 2017. DOI: 10.14358/PERS.83.9.611.

[32] "Chapter 1 - A systematic view of remote sensing," in *Advanced Remote Sensing (Second Edition)*, S. Liang and J. Wang, Eds., Academic Press, Jan. 1, 2020, pp. 1–57, ISBN: 978-0-12-815826-5. DOI: 10.1016/B978-0-12-815826-5.00001-5.

[33] "Chapter 2 - Geometric processing and positioning techniques," in *Advanced Remote Sensing: Terrestrial Information Extraction and Applications*, S. Liang and J. Wang, Eds., Second edition, Amsterdam: Academic Press, 2020, pp. 59–105, ISBN: 978-0-12-815826-5.

[34] D. Liu, G. Zhou, D. Zhang, X. Zhou, and C. Li, "Ground Control Point Automatic Extraction for Spaceborne Georeferencing Based on FPGA," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, pp. 1–1, Jun. 1, 2020. DOI: 10.1109/JSTARS.2020.2998838.

[35] T. Long, W. Jiao, and X. Jia, "A new method for automatic gross error detection in remote sensing image geometric correction," Aug. 1, 2010, pp. 1940–1945. DOI: 10.1109/FSKD.2010.5569465.

[36] A. Marsetič, K. Oštir, and M. K. Fras, "Automatic Orthorectification of High-Resolution Optical Satellite Images Using Vector Roads," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 53, no. 11, pp. 6035–6047, Nov. 2015, ISSN: 1558-0644. DOI: 10.1109/TGRS.2015.2431434.

[37] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, Cancun: IEEE, Dec. 2016, pp. 378–383, ISBN: 978-1-5090-4847-2. DOI: 10.1109/ICPR.2016.7899663.

[38] N. Merkle, "Geo-localization Refinement of Optical Satellite Images by Embedding Synthetic Aperture Radar Data in Novel Deep Learning Frameworks," Jan. 1, 2018.

[39] T. Miyamoto and Y. Yamamoto, "Using Multimodal Learning Model for Earthquake Damage Detection Based on Optical Satellite Imagery and Structural Attributes," in *IGARSS 2020 - 2020 IEEE International Geoscience and Remote Sensing Symposium*, Sep. 2020, pp. 6623–6626. DOI: 10.1109/IGARSS39084.2020.9324464.

[40] D. Mukherjee, Q. M. Jonathan Wu, and G. Wang, "A comparative experimental study of image feature detectors and descriptors," *Machine Vision and Applications*, vol. 26, no. 4, pp. 443–466, May 1, 2015, ISSN: 1432-1769. DOI: 10.1007/s00138-015-0679-9.

[41] D. Mulawa, C. Comp, and B. Clarke, "Geolocation Accuracy Performance of the DigitalGlobe Constellation During 2017 and 2018 H," presented at the Joint Agency Commercial Imagery Evaluation 2018 (College Park, MD), Aug. 29, 2018.

[42] R. Müller, T. Krauß, M. Schneider, and P. Reinartz, "Automated Georeferencing of Optical Satellite Data with Integrated Sensor Model Improvement," *Photogrammetric Engineering & Remote Sensing*, vol. 78, no. 1, pp. 61–74, Jan. 1, 2012. DOI: 10.14358/PERS.78.1.61.

[43] R. Müller, M. Lehner, R. Müller, P. Reinartz, M. Schroeder, and B. Vollmer, "A Program for Direct Georeferencing of Airborne and Spaceborne Line Scanner Images," *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, vol. 34, no. 1, pp. 148–153, 2002.

[44] Müller, Rupert. "DLR - Earth Observation Center - Orthorectification," German Aerospace Center (DLR): Remote Sensing Technology Institute, Photogrammetry and Image Analysis. (), [Online]. Available: https://www.dlr.de/eoc/en/desktopdefault.aspx/tabid-6144/10056_read-20918/.

[45] NASA. "What is Remote Sensing?" Earthdata. (Jul. 13, 2021), [Online]. Available: http://www.earthdata.nasa.gov/learn/backgrounders/remote-sensing.

[46] Y. Ono, E. Trulls, P. Fua, and K. M. Yi. "LF-Net: Learning Local Features from Images." arXiv: 1805.09662 [cs]. (Nov. 22, 2018), [Online]. Available: http://arxiv.org/abs/1805.09662, preprint.

[47] J.-C. Padró, F.-J. Muñoz, J. Planas, and X. Pons, "Comparison of four UAV georeferencing methods for environmental monitoring purposes focusing on the combined use with airborne and satellite remote sensing platforms," *International Journal of Applied Earth Observation and Geoinformation*, vol. 75, pp. 130–140, Mar. 1, 2019, ISSN: 1569-8432. DOI: 10.1016/j.jag.2018.10.018.

[48] Penn State College of Earth and Mineral Sciences. "6. Resolution | The Nature of Geographic Information," The Nature of Geographic Information. (), [Online]. Available: https://www.e-education.psu.edu/natureofgeoinfo/c8_p7.html.

[49] D. Poli, "A Rigorous Model for Spaceborne Linear Array Sensors," *Photogrammetric Engineering & Remote Sensing*, vol. 73, no. 2, pp. 187–196, Feb. 1, 2007. DOI: 10.14358/PERS.73.2.187.

[50] J.-Y. Rau, L. Chen, C.-C. Hsieh, and T.-M. Huang, "Static error budget analysis for a land-based dual-camera mobile mapping system," *Journal of the Chinese Institute of Engineers*, vol. 34, pp. 849–862, Oct. 1, 2011. DOI: 10.1080/02533839.2011.591914.

[51] P. Reinartz, R. Müller, M. Lehner, and M. Schroeder, "Accuracy analysis for DSM and orthoimages derived from SPOT HRS stereo data using direct georeferencing," *ISPRS Journal of Photogrammetry and Remote Sensing*, Extraction of Topographic Information from High-Resolution Satellite Imagery, vol. 60, no. 3, pp. 160–169, May 1, 2006, ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2005.12.003.

[52] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 2, Oct. 2005, 1508–1515 Vol. 2. DOI: 10.1109/ICCV.2005.104.

[53] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich. "SuperGlue: Learning Feature Matching with Graph Neural Networks." arXiv: 1911.11763 [cs]. (Mar. 28, 2020), [Online]. Available: http://arxiv.org/abs/1911.11763, preprint.

[54] Sasha Weston, "State-of-the-Art of Small Spacecraft Technology," NASA Ames Research Center, Small Spacecraft Systems Virtual Institute, Moffett Field, California, NASA/TP—2022–0018058, Jan. 1, 2023.

[55] E. Setyawan. "Orthorectification in a Nutshell." (Aug. 5, 2019), [Online]. Available: https://www.intermap.com/blog/orthorectification-in-a-nutshell.

[56] R. Szeliski, *Computer Vision: Algorithms and Applications* (Texts in Computer Science), 2nd ed. Cham, Switzerland: Springer International Publishing, 2022, ISBN: 978-3-030-34371-2 978-3-030-34372-9. DOI: 10.1007/978-3-030-34372-9.

[57] K. Thangavel *et al.*, "Autonomous Satellite Wildfire Detection Using Hyperspectral Imagery and Neural Networks: A Case Study on Australian Wildfire," *Remote Sensing*, vol. 15, no. 3, p. 720, 3 Jan. 2023, ISSN: 2072-4292. DOI: 10.3390/rs15030720.

[58] K. Thangavel *et al.*, "Trusted Autonomous Operations of Distributed Satellite Systems Using Optical Sensors," *Sensors*, vol. 23, no. 6, p. 3344, 6 Jan. 2023, ISSN: 1424-8220. DOI: 10.3390/s23063344.

[59] Y. Tian, V. Balntas, T. Ng, A. Barroso-Laguna, Y. Demiris, and K. Mikolajczyk. "D2D: Keypoint Extraction with Describe to Detect Approach." arXiv: 2005.13605 [cs, eess]. (May 27, 2020), [Online]. Available: http://arxiv.org/abs/2005.13605, preprint.

[60] Y. Tian, B. Fan, and F. Wu, "L2-Net: Deep Learning of Discriminative Patch Descriptor in Euclidean Space," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 6128–6136. DOI: 10.1109/CVPR.2017.649.

[61] T. Toutin, "Review article: Geometric processing of remote sensing images: Models, algorithms and methods," *International Journal of Remote Sensing*, vol. 25, no. 10, pp. 1893–1924, May 1, 2004, ISSN: 0143-1161. DOI: 10.1080/0143116031000101611.

[62] Unibap AP, *SpaceCloud iX10-100 Datasheet*, 1004028 v1.5, 2023.

[63] R. Wessen, C. S. Borden, J. K. Ziemer, R. C. Moeller, J. Ervin, and J. Lang, "Space Mission Concept Development using Concept Maturity Levels," in *AIAA SPACE 2013 Conference and Exposition*, San Diego, CA: American Institute of Aeronautics and Astronautics, Sep. 10, 2013, ISBN: 978-1-62410-239-4. DOI: 10.2514/6.2013-5454.

[64] *World Imagery*, in collab. with Maxar and E. Geographics, [basemap], Esri, Sep. 6, 2023.

[65] N. Yastikli and K. Jacobsen, "The Effect of System Calibration on Direct Sensor Orientation," presented at the ISPRS Congress, Geo-Imagery Bridging Continents, Jan. 1, 2004.

[66] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. "LIFT: Learned Invariant Feature Transform." arXiv: 1603.09114 [cs]. (Jul. 29, 2016), [Online]. Available: https://www.epfl.ch/labs/cvlab/research/descriptors-and-keypoints/lift/, preprint.

[67] K. Zhang, H. Okazawa, K. Hayashi, T. Hayashi, L. Fiwa, and S. Maskey, "Optimization of Ground Control Point Distribution for Unmanned Aerial Vehicle Photogrammetry for Inaccessible Fields," *Sustainability*, vol. 14, no. 15, p. 9505, 15 Jan. 2022, ISSN: 2071-1050. DOI: 10.3390/su14159505.

[68] H. Zhao, C. Wu, Z. Zuo, Z. Chen, and J. Bi, "Direct georeferencing of oblique and vertical imagery in different coordinate systems," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 95, pp. 122–133, Sep. 1, 2014. DOI: 10.1016/j.isprsjprs.2014.06.001.

[69] X. Zheng, Q. Huang, J. Wang, T. Wang, and G. Zhang, "Geometric Accuracy Evaluation of High-Resolution Satellite Images Based on Xianning Test Field," *Sensors (Basel, Switzerland)*, vol. 18, no. 7, p. 2121, Jul. 2, 2018, ISSN: 1424-8220. DOI: 10.3390/s18072121. pmid: 30004440.

[70] J. T. Zhu, C. F. Gong, M. X. Zhao, L. Wang, and Y. Luo, "Image Mosaic Algorithm Based on PCA-ORB Feature Matching," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-3-W10, pp. 83–89, Feb. 7, 2020, ISSN: 1682-1750. DOI: 10.5194/isprs-archives-XLII-3-W10-83-2020.

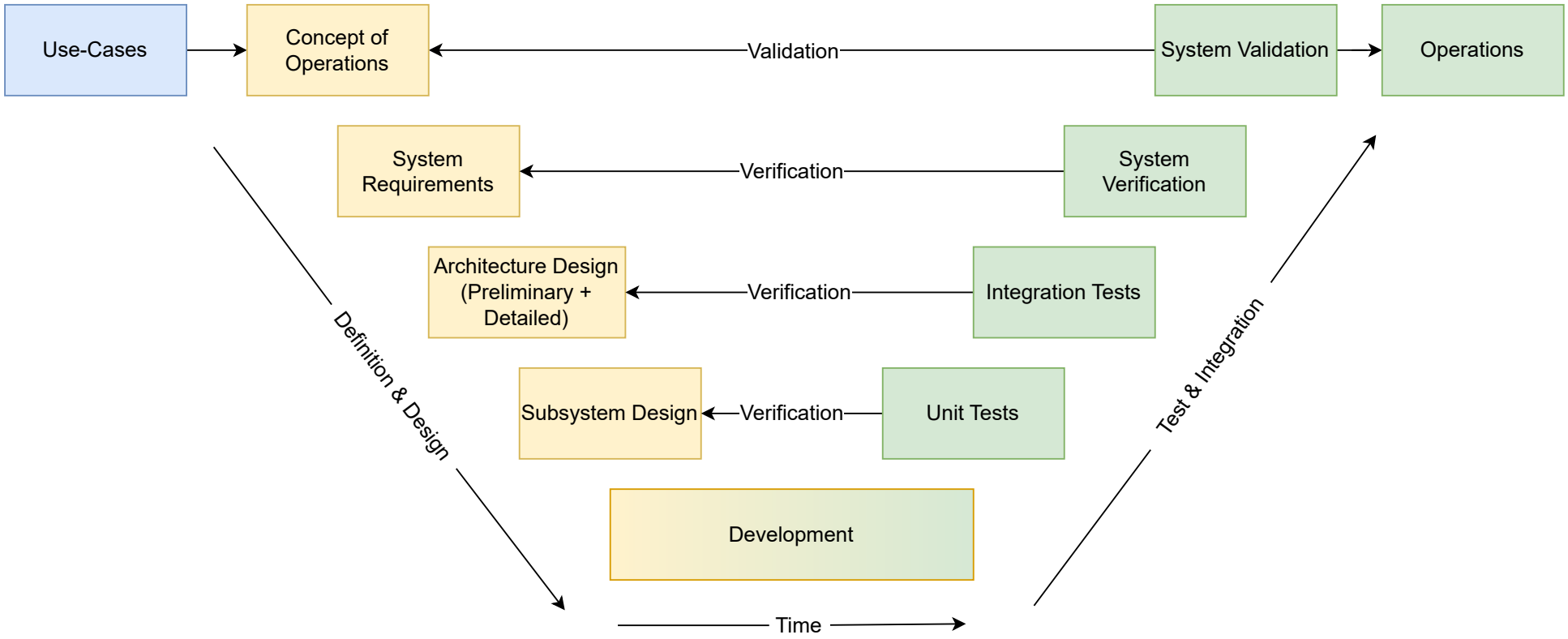
[71] B. Zitová and J. Flusser, "Image registration methods: A survey," *Image and Vision Computing*, vol. 21, no. 11, pp. 977–1000, Oct. 1, 2003, ISSN: 0262-8856. DOI: 10.1016/S0262-8856(03)00137-9.

# A

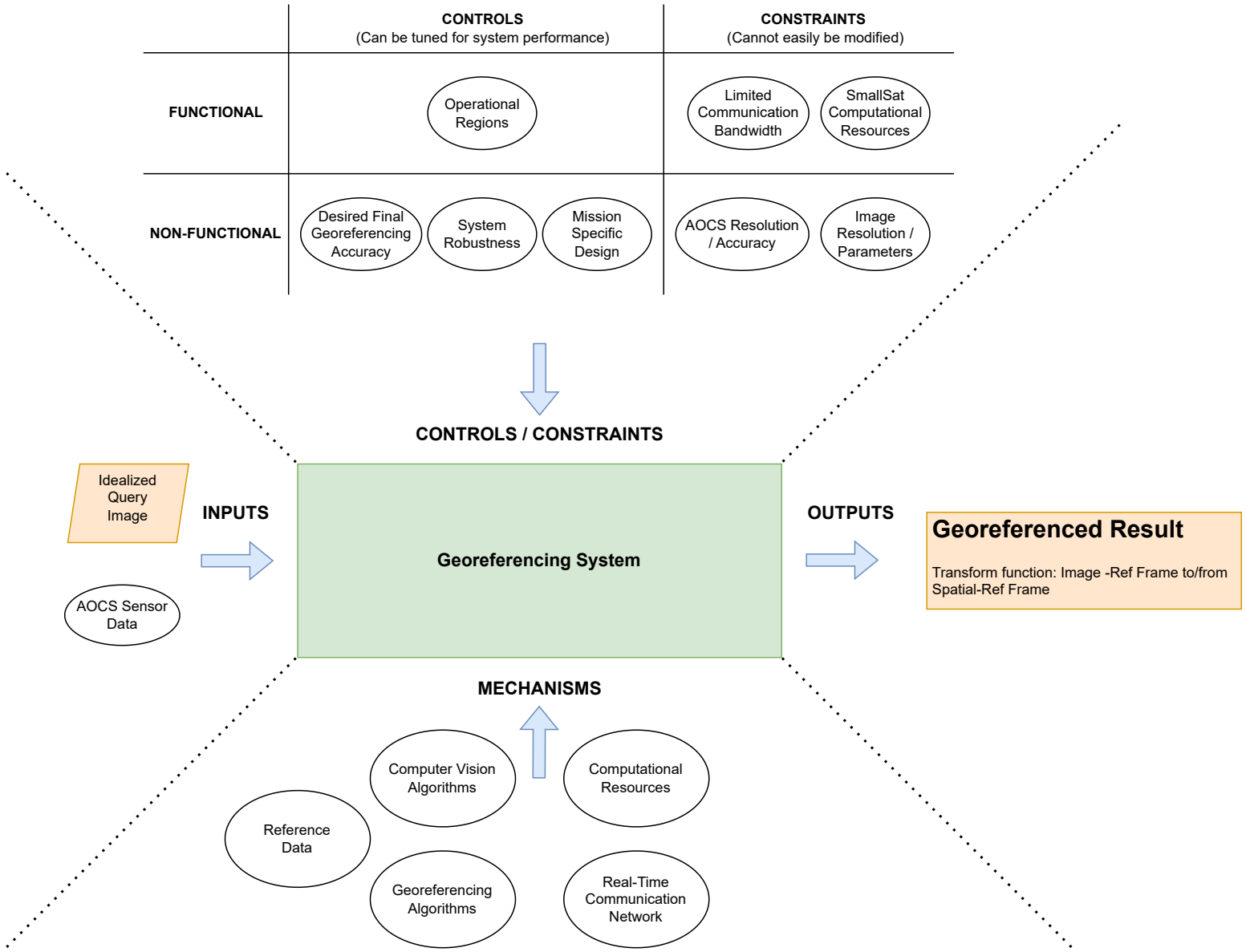
# Diagrams

The following diagrams are larger versions of those found in the text.

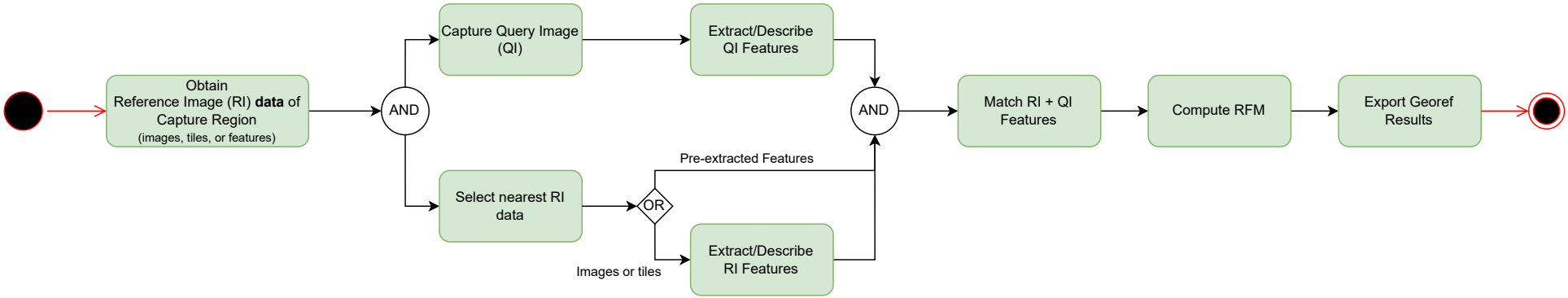
## Systems Engineering V-Model (Figure 3.1)



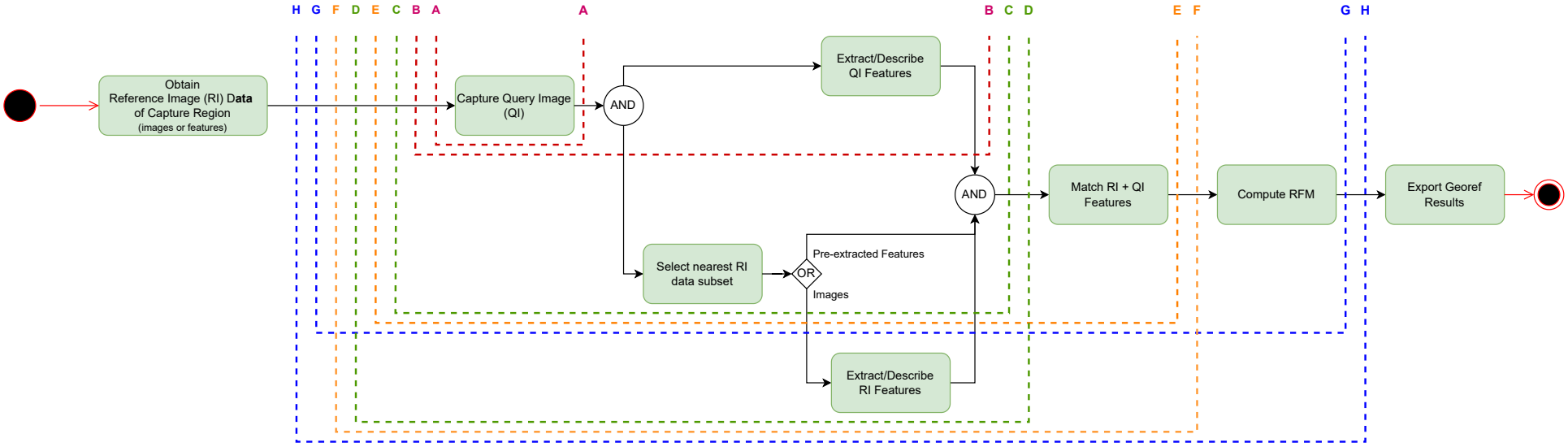
## Context of Operations Diagram (Figure 3.2)



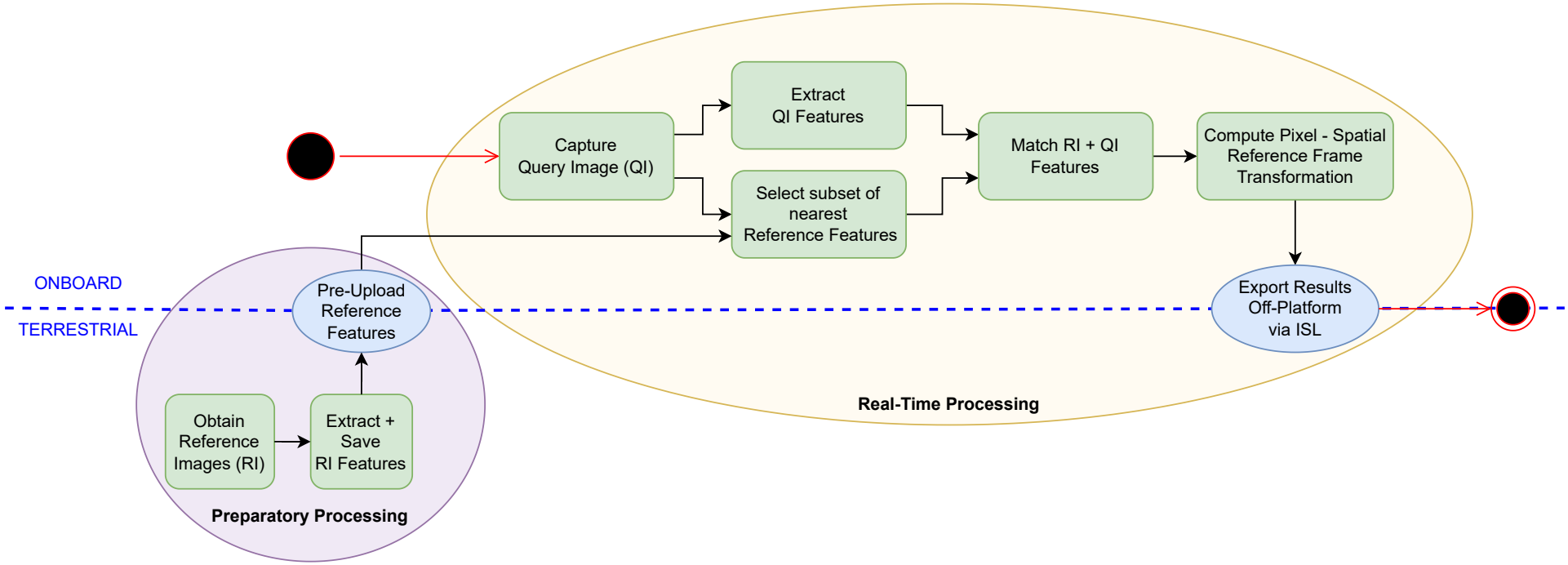
## Functional Flow Diagram (Figure 3.6)



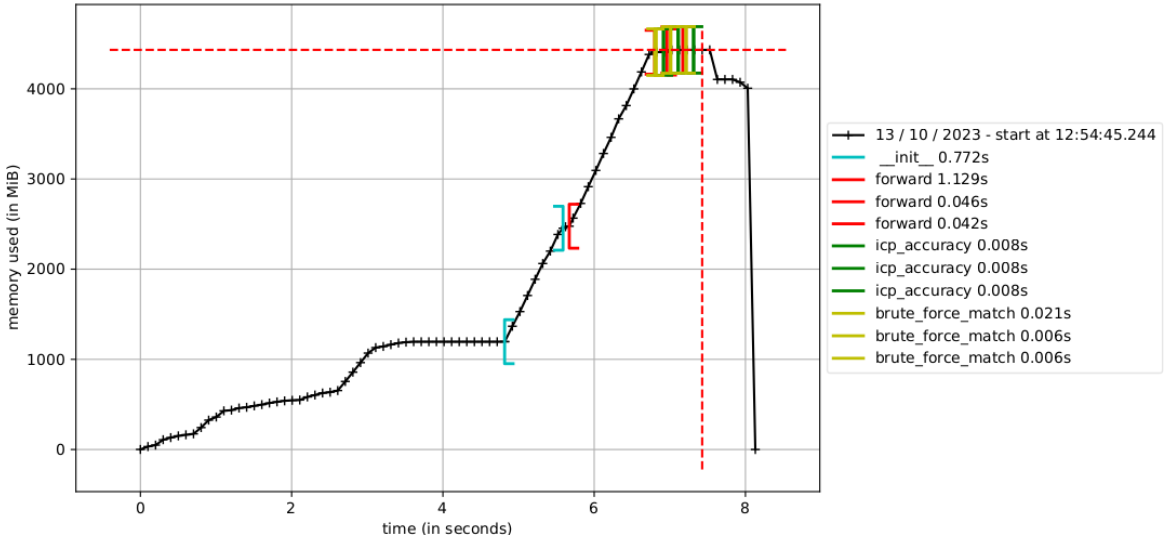

## Architecture Discovery Diagram (Figure 3.7)



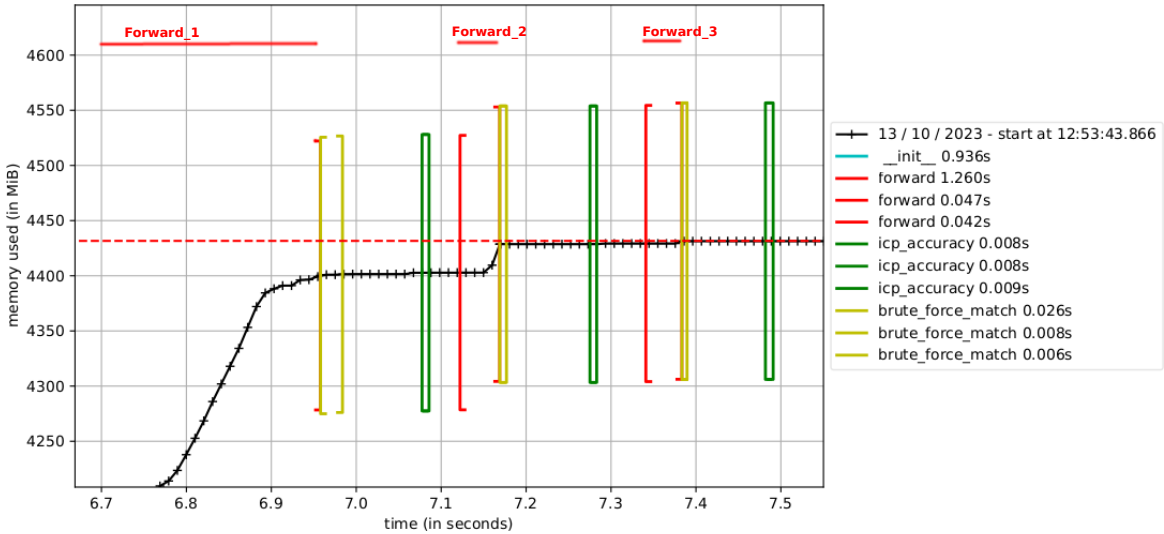
## System Architecture Diagram (Figure 4.1)



## Total System Memory Profile (Figure 4.12)




## Inference Memory Profile (Figure 4.13)



# B

# System Implementation Documentation

This appendix contains documentation on how to use the Command Line Interface (CLI) of the novel georeferencing system implementation, nicknamed REal-time Georeferencing in Space (REGIS). In addition to the system simulation, the program contains a number of additional tools used for generating the RFDB, training experimental keypoint models, data handling, results analysis, and more utilities.

The following sections describe the usage of the various tools. Each command or subcommand contains a help menu that can be printed using the `-h` or `-help` flag. The main help menus are listed here for reference.

The commands can also be loaded directly from a configuration file using the `--config` flag. The configuration file is a yaml file containing the arguments for the command and is automatically saved to the results directory for each run. Any arguments passed after the `-config <path/to/config>` will override those loaded from the config file.

## B.1. Requirements

- Python3 via Mamba (preferred) or Conda
- Ubuntu 22.04 recommended, not tested on other platforms

## B.2. CLI Usage

Once installed, the program can be run from the command line using the `regis` command. The program is split into a number of subcommands, each with their own help menu. The main help menu is shown below.

```
usage: regis [-h] [-v] [--config CONFIG] [--print_config [=flags]] [--logfile LOGFILE] [--log-level {TRACE,DEBUG,INFO,SUCCESS,WARNING,ERROR}]
             {keypt,georef,utils} ...

REaltime Geoferencing In Space (REGIS) Interface

options:
  -h, --help                                          show this help message and exit
  -v, --version                                       show program's version number and exit
  --config CONFIG                                     Path to yaml configuration file containing run arguments. (default: None)
  --print_config [=flags]                            Print the configuration after applying all other arguments and exit. The optional flags customizes the
                                                      output and are one or more keywords separated by comma. The supported flags are: comments, skip_default,
                                                      skip_null.
  --logfile LOGFILE                                   REGIS logfile path (default: /home/luigi/Documents/system_design/logs/regis.log)
  --log-level {TRACE,DEBUG,INFO,SUCCESS,WARNING,ERROR}  Specify logging level (default: INFO)

Commands:
  For more details of each subcommand, add it as an argument followed by --help.

  {keypt,georef,utils}                                Possible commands
    keypt                                             Keypoint extraction pipeline
    georef                                            Georeferencing pipeline
    utils                                             Additional utilities
```

### B.2.1. RFDB Generation

Command to generate the RFDB from a dataset. The RFDB is a geopackage database containing the keypoints and descriptors of a dataset.

```
usage: regis [options] georef [options] rfdb [-h] [--model {d2net,unet}] [-s SEED] [-k TOP_K] [-n N_TILES] [--rfdb-epsg RFDB_EPSG]
                                             [--n-bands-per-tile N_BANDS_PER_TILE] [--data-split {train,val,test,all}] [-b BATCH_SIZE]
                                             [--img-bands BAND_A [BAND_B ...]] [--img-res IMG_RES] [--img-tile-size HEIGHT_PX WIDTH_PX] [--augment]
                                             [--tile-offset-px TILE_OFFSET_PX TILE_OFFSET_PX] [--no-relu]
                                             data_directory weights_checkpoint_path db_save_dir
```

```
options:
  -h, --help                                    show this help message and exit
  --model {d2net,unet}                          Model to use for feature extraction (default: d2net)
  -s SEED, --seed SEED                          Seed all pytorch random functions (including selecting tiles) (default: 23)
  -k TOP_K, --top-k TOP_K                       Number of features to extract (default: 30)
  -n N_TILES, --n-tiles N_TILES                 Number of tiles to be processed from the source image. `0` results in all tiles being processed. (default: 1)
  --rfdb-epsg RFDB_EPSG                         EPSG code of the RF database common CRS. `0` uses the CRS of the dataset, if only one available (default:
                                                4326)
  --n-bands-per-tile N_BANDS_PER_TILE           Number of bands to use per feature descriptor. Cannot be more than the number of bands in the dataset
                                                (default: 1)

REQUIRED Arguments:
  data_directory                                Path to directory containing datasets
  weights_checkpoint_path                       Path to checkpoint (*.chkpt) or (*.pt) file to load model
  db_save_dir                                   Directory to save database as Geopackage

Dataset options:
  --data-split {train,val,test,all}             Dataset split type (default: all)
  -b BATCH_SIZE, --batch-size BATCH_SIZE        Number of samples per batch (default: 16)
  --img-bands BAND_A [BAND_B ...]               Image bands to be used from the dataset (default: ['B02', 'B03'])
  --img-res IMG_RES                             Spatial resolution of the dataset, used to find files (default: 10)
  --img-tile-size HEIGHT_PX WIDTH_PX            Size of image tiles to create, in pixels. U-Net requires a size divisible by 32. (default: [320, 320])
  --augment                                     Apply augmentations to dataset (default: False)
  --tile-offset-px TILE_OFFSET_PX TILE_OFFSET_PX  Offset the tiles in the dataset by this number of PIXELS. Calculated as [n_rows, n_columns] from the BOTTOM
                                                LEFT corner. This is because dataset is split based on south-west corner. Useful for creating a datset with
                                                offset tiles from the original dataset. (default: [0, 0])

D2Net options:
  --no-relu                                     D2-Net: remove ReLU after the dense feature extraction module (default: True)
```

## B.2.2. Georeferencing Simulation

Runs a parametric simulation of the georeferencing system. The arguments passed here are automatically saved to a configuration file in the results directory, which can be loaded directly at any later time.

```
usage: regis [options] georef [options] sim [-h] [--model {d2net,unet}] [--top-k TOP_K] [-s SEED] [-n N_TILES] [-m BEST_M_MATCHES]
                                            [--icps-grid-dim ICPS_GRID_DIM] [--gcp-match-distance GCP_MATCH_DISTANCE] [--px-xy-location {center,ul,ll,lr,ur}]
                                            [--ransac] [--ransac-tol RANSAC_TOL] [--no-relu] [--data-split {train,val,test,all}] [--qi-rotate QI_ROTATE]
                                            [--qi-shift-x QI_SHIFT_X] [--qi-shift-y QI_SHIFT_Y] [-b BATCH_SIZE] [--img-bands BAND_A [BAND_B ...]]
                                            [--img-res IMG_RES] [--img-tile-size HEIGHT_PX WIDTH_PX] [--augment]
                                            [--tile-offset-px TILE_OFFSET_PX TILE_OFFSET_PX] [--plot-matches] [--rfdb-source-image] [--map-full-rfdb]
                                            [--disable-map] [--last-map-only] [--no-auto-open]
                                            data_directory weights_checkpoint_path db_path results_dir

options:
  -h, --help                                          show this help message and exit
  --model {d2net,unet}                                Model to use for training (default: d2net)
  --top-k TOP_K                                       Number of top-k keypoints to extract from each image (default: 50)

REQUIRED Arguments:
  data_directory                                      Path to directory containing datasets
  weights_checkpoint_path                             Path to checkpoint (*.chkpt) file to load model
  db_path                                             Filepath to geopackage database
  results_dir                                         Directory path to save simulation results and run metadata

Simulation Options:
  -s SEED, --seed SEED                                Seed all pytorch random functions (including selecting tiles) (default: 23)
  -n N_TILES, --n-tiles N_TILES                       Number of tiles to be processed from the source dataset. `None` results in all tiles being processed.
                                                      (default: None)
  -m BEST_M_MATCHES, --best-m-matches BEST_M_MATCHES  Matching parameter to return m best keypoint matches where m is specified by the user. Default is to
                                                      return all matches (default: -1)
  --icps-grid-dim ICPS_GRID_DIM                       Square grid dimension of Independent Check Points (ICPs) to evaluate the accuracy. Default is 5, which
                                                      results in a 5x5 grid (25 samples). (default: 10)
  --gcp-match-distance GCP_MATCH_DISTANCE             Maximum distance that the spatial coordinates of GCPs may have to still be considereda correct match. In
                                                      meters. (default: 50)
  --px-xy-location {center,ul,ll,lr,ur}               Determines if the returned coordinates are for the center of the pixel or for a corner. (default: center)
  --ransac                                            Removes the worst GCP outliers iteratively until the minimum number of GCPs are reached or no outliers can
                                                      be detected (default: False)
  --ransac-tol RANSAC_TOL                             Tolerance used when refining the number of GCPs to use. According to `gdalwarp` documentation, "In pixel
                                                      units if no projection is available, otherwise it is in SRS (spatial reference system) units" (default:
                                                      1.0)

D2Net options:
  --no-relu                                           D2-Net: remove ReLU after the dense feature extraction module (default: True)

Dataset Options:
  --data-split {train,val,test,all}                   Dataset split type (default: test)
  --qi-rotate QI_ROTATE                               Rotate Query Image by R degrees counter-clockwise (default: 0)
  --qi-shift-x QI_SHIFT_X                             Shift Query Images by X meters to the East (default: 0)
  --qi-shift-y QI_SHIFT_Y                             Shift Query Images by Y meters to the North (default: 0)
  -b BATCH_SIZE, --batch-size BATCH_SIZE              Number of samples per batch (default: 16)
  --img-bands BAND_A [BAND_B ...]                     Image bands to be used from the dataset (default: ['B02', 'B03'])
  --img-res IMG_RES                                   Spatial resolution of the dataset, used to find files (default: 10)
  --img-tile-size HEIGHT_PX WIDTH_PX                  Size of image tiles to create, in pixels. U-Net requires a size divisible by 32. (default: [320, 320])
  --augment                                           Apply augmentations to dataset (default: False)
  --tile-offset-px TILE_OFFSET_PX TILE_OFFSET_PX      Offset the tiles in the dataset by this number of PIXELS. Calculated as [n_rows, n_columns] from the
                                                      BOTTOM LEFT corner. This is because dataset is split based on south-west corner. Useful for creating a
                                                      datset with offset tiles from the original dataset. (default: [0, 0])

Output Options:
  --plot-matches                                      Creates interactive pyplot of matched keypoints per QI sample. CAUTION: Pauses simulation for each sample.
                                                      (default: False)
  --rfdb-source-image                                 Path to RFDB dataset. Adds reference image to debug_map (default: False)
  --map-full-rfdb                                     Plot full Reference Feature Database keypoints on map. Significantly increases sim time, and slows map
                                                      loading (default: False)
  --disable-map                                       Disables creation of interactive html map for each QI sample. Does not pause simulation. (default: False)
  --last-map-only                                     Generate a map only for the last sample (default: False)
  --no-auto-open                                      Disables automatic opening of the interactive html map and plots (default: False)
```

# C

# Source Code

The source code included here are the most essential modules used to implement the georeferencing system. REal-time Georeferencing in Space (REGIS) is the software name of this implementation, and is used to differentiate portions of the code from existing libraries (such as D2-Net). The source code is split into appropriate sections below.

## C.1. Sentinel-2 Datasets

These classes are used to load Sentinel-2 products into data structures that can be used by both the georeferencing system and PyTorch training pipelines. The granules are segmented into tiles and the image data is only loaded into memory when the specific tile is requested. This is therefore extremely performant and allows for the use of large datasets without the need for large amounts of memory.

**Listing C.1:** sentinel_dataset.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Description   :
Author        : Luigi Maiorano (lmaiorano97@gmail.com)
Creation Date : 19-07-2023
References :
Based on TorchGeo Sentinel2:
- https://github.com/microsoft/torchgeo/blob/8b53304d42c269f9001cb4e861a126dc4b462606/torchgeo/datasets/sentinel.py
"""
import functools
import glob
import itertools
import os
import re
import warnings
from datetime import datetime
from pathlib import Path
from typing import Any, Callable, Optional, Tuple, Union

import numpy as np
import rasterio
import torch
from affine import Affine
from loguru import logger
from pyproj import CRS, Transformer
from rasterio.coords import BoundingBox
from rasterio.features import geometry_window
from rasterio.transform import AffineTransformer
from rasterio.warp import Resampling, calculate_default_transform, reproject
from rasterio.windows import from_bounds
from shapely.geometry import Polygon
from torch import Tensor
from torchgeo.datamodules import NonGeoDataModule
from torchgeo.datasets import NonGeoDataset


class S2Product:
    def __init__(self, bounds: BoundingBox, bandfile_srcs: dict[str, rasterio.io.DatasetReader]) -> None:
        """object containing the Sentinel 2 source datasets

        The `bandfile_srcs` contains OPEN `rasterio.io.DatasetReader` objects  for
        each band file. This architecture is used to circumvent a GDAL bug in which
        each open DatasetReader instance has a small memory leak. Keeping these objects
        open prevents the need to create new instances each time the RegisS2Dataset class reads a
        patch using __get_item__().

        Downside is that only ONE process can be used.

        Args:
            bounds (BoundingBox): Complete bounding box of the Sentinel-2 Product
            bandfile_srcs (dict[str, rasterio.io.DatasetReader]): Dictionary of each band name and
                the corresponding `open DatasetReader` object.
        """
        self.bounds = bounds
        self.bandfile_srcs = bandfile_srcs
        self.bands = list(bandfile_srcs.keys())
```

```
        self.name = Path(self.bandfile_srcs[self.bands[0]].name).parts[-6]
        self.crs = self.bandfile_srcs[self.bands[0]].crs

        # Check that CRS and transforms of all bands are identical
        for src in self.bandfile_srcs.values():
            assert src.crs == self.bandfile_srcs[self.bands[0]].crs
            assert src.transform == self.bandfile_srcs[self.bands[0]].transform

    def __str__(self):
        return self.name


class S2Patch:
    def __init__(self, bounds: BoundingBox, s2_dataset: S2Product, bands: list[str]) -> None:
        """A Sentinel2 dataset patch object

        The RegisS2Dataset index is be a list of these objects

        Args:
            bound_box (BoundingBox): Bounding box of the patch
            bands (list[str]): Pair of bands to be included in this patch
            s2_dataset (S2Dataset): Pointer to the S2Dataset object containing the rasterio
                `open DatasetReader` objects for each band (stored in attribute `bandfile_srcs`)
        """
        self.bounds = bounds
        self.s2_dset = s2_dataset
        self.bands = list(set(bands))
        self.srcs = {b: self.s2_dset.bandfile_srcs[b] for b in self.bands}
        self.date = self._get_date()

    def _get_date(self):
        s0 = self.get_src0()
        name = Path(s0.name).stem
        return datetime.strptime(name.split("_")[1], "%Y%m%dT%H%M%S")

    def get_src0(self) -> rasterio.DatasetReader:
        """Get src object of the first band of this patch.

        Each band has a different src, since each band has its own file"""
        return self.srcs[self.bands[0]]

    def get_crs(self) -> CRS:
        """CRS's of both bands are identical, therefore just return one"""
        return self.get_src0().crs

    def get_window(self) -> rasterio.windows.Window:
        """Returns window based on the bounds and transform of this patch"""
        return from_bounds(*self.bounds, self.get_src0().transform)


class RegisSentinel2Dataset(NonGeoDataset):
    """Sentinel 2 dataset for use in the REGIS keypoint pipeline

    The underlying principle of this DatasetClass is to construct an index that does not need to
    load the rasterdata into memory. Only once the item is queried using __get_item__, does the
    image get loaded using rasterio.

    All necessary splitting into patches is done using bounding boxes
    and other metadata.

    """

    # https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-2-msi/naming-convention
    # https://sentinel.esa.int/documents/247904/685211/Sentinel-2-MSI-L2A-Product-Format-Specifications.pdf
    filename_glob = "T*_*_{}*.*"
    filename_regex = r"""
␣␣␣␣␣␣␣␣^T(?P<tile>\d{{2}}[A-Z]{{3}})
␣␣␣␣␣␣␣␣_(?P<date>\d{{8}}T\d{{6}})
␣␣␣␣␣␣␣␣_(?P<band>B[018][\dA])
␣␣␣␣␣␣␣␣(?:_(?P<resolution>{}m))?
␣␣␣␣␣␣␣␣\..*$
␣␣␣␣"""
    date_format = "%Y%m%dT%H%M%S"

    # https://gisgeography.com/sentinel-2-bands-combinations/
    all_bands = ["B01","B02","B03","B04","B05","B06","B07","B08","B8A","B09","B10","B11","B12",]
    rgb_bands = ["B04", "B03", "B02"]

    index: list[S2Patch]

    @staticmethod
    def tile_band_tensor(tile, band) -> torch.Tensor:
        """Returns 4D single band tensor of a given sample"""
        band_img_idx = tile["bands_order"].index(band)
        return tile["image"][band_img_idx, :, :].unsqueeze(0).unsqueeze(0)

    def __init__(
        self,
        root: str = "data",
        split: str = "train",
        bands: list[str] = ["B02", "B03"],
        res: float = 10,
        px_tile_size: tuple[int, int] = (None, None),
        transforms: Optional[Callable[[dict[str, Tensor]], dict[str, Tensor]]] = None,
        cache: bool = True,
        download: bool = False,
        extended_getitem: bool = False,
        n_bands_per_tile: int = 2,
        tile_offset=[0, 0],
    ) -> None:
        """Initialize a new RegisSentinel2 dataset instance

        Attributes of importance:
            index: A list of S2Patch objects that can be sampled. Each S2Patch object contains the
                geographic bounds of the patch, and up to 2 bands of data. If only 1 band is passed
                to this __init__(), each patch will also contain only 1 band. If 3 or more bands are
                passed, this dataset will compute all pair combinations of bands, and generate a S2Patch for each pair.
            patch_bboxes: A list of patch bounding boxes and the corresponding index indices. This
                is useful for generating a map of the bounding boxes without repeats caused by
                multiple combinations of bands.

        Args:
            root (str, optional): root directory where dataset can be found. Defaults to "data".
            split (str, optional): train/val/test split to load. Choose from {'train', 'val', 'test'}.
                Defaults to "train".
            bands (str, optional): bands to return (defaults to all bands). Defaults to all_bands.
```

```
            res (float, optional): resolution of the dataset in meters (as specified by ESA).
                Defaults to 10.
            px_tile_size (tuple[int, int], optional): size of patches to be generated from full Sentinel-2 image.
                Should equal the size of the images fed into the network. T-V-T is fixed to 50%, 25%, 25%
                in order to significantly simplify and prevent the need of additional tiling functions.
            transforms (Optional[Callable[[dict[str, Tensor]], dict[str, Tensor]]], optional):
                a function/transform that takes input sample and its target as entry and returns a
                transformed version. Defaults to None.
            cache (bool, Optional): if True, cache file handle to speed up repeated sampling. Must be True if
                num_workers > 0. Defaults to True.
            download (bool, optional): if True, download dataset and store it in the root directory.
                Defaults to False.
            extended_getitem (bool, optional): Return extended dict with __get_item__, containing
                non-pythorch datatypes: tensors, numpy arrays, numbers, dicts or lists. Default False
            n_bands_per_tile (int, optional): Number of bands to include in each tile. For self-supervised
                training, it is essential that each tile includes exactly 2 bands. Defaults to 2.
            tile_offset (tuple[int, int], optional): Offset of the dataset to be created with respect to the
                original Sentinel2 products. Will result in less tile than the original. Defaults to None.
        """
        super().__init__()
        bands = bands or self.all_bands
        for b in bands:
            assert b in self.all_bands, "Only Sentinel-2 bands allowed"

        self.band0_glob = self.filename_glob.format(bands[0])
        self.filename_regex = self.filename_regex.format(res)

        self.root = root
        self.split = split.lower()
        self.bands = bands
        self.resolution = res
        self.px_tile_size = tuple(px_tile_size)
        self.txforms = transforms
        self.download = download
        self.cache = cache
        self.extended_getitem = extended_getitem
        self.name = root.name
        self.n_bands_per_tile = n_bands_per_tile
        self.tile_offset = tile_offset

        self._download()

        self.s2_products = self._find_products()

        if None in self.px_tile_size:
            self.patch_size = (np.Inf, np.Inf)
        else:
            # Patch size = resolution * pixels
            self.patch_size = (
                self.px_tile_size[0] * self.resolution,
                self.px_tile_size[1] * self.resolution,
            )

        self._set_max_patch_size()

        self._build_index()
        self._create_index_grids()

    def __getitem__(self, idx: int) -> dict[str, Any]:
        """Return a tile within the dataset.

        Args:
            index: index to return
        Returns:
            Dict containing: Image tensor, bands order along dim0, index of the sample (to be used for plotting)
        """

        tensor, bands_order = self._load_band_images(idx)

        if self.txforms is not None:
            tensor = self.txforms(tensor).squeeze()

        if self.extended_getitem:
            return {
                "image": tensor,
                "bands_order": bands_order,
                "dset_index": idx,
                "bounds": self.index[idx].bounds,
                "crs": self.index[idx].get_crs(),
                "window": self.index[idx].get_window(),
            }

        return {
            "image": tensor,
            "bands_order": bands_order,
            "dset_index": idx,
        }

    def __len__(self) -> int:
        """Return the number of data points in the dataset.

        Returns:
            length of the dataset
        """
        return len(self.index)

    def _load_band_images(self, idx, array_out=False) -> tuple[Tensor, list]:
        images = []
        bands = []

        item = self.index[idx]

        window = item.get_window()

        for band, src in item.srcs.items():
            array = src.read(indexes=1, out_shape=self.px_tile_size, window=window)

            # fix numpy dtypes which are not supported by pytorch tensors
            if array.dtype == np.uint16:
                array = array.astype(np.int32)
            elif array.dtype == np.uint32:
                array = array.astype(np.int64)

            images.append(array)
            bands.append(band)

```

```
        arrays: "np.typing.NDArray[np.int_]" = np.stack(images, axis=0)

        if array_out:
            return arrays, bands

        tensor = torch.from_numpy(arrays).float()
        # As done by __get_item__ in GeoDataset (base of Sentinel2 dataset class)
        tensor = tensor.to(torch.float32)
        return tensor, bands

    def _set_max_patch_size(self) -> None:
        """Patch size must at a maximum be half the bounds of the smallest image"""
        # find smallest bounds
        minh = minw = np.inf
        for dset in self.s2_products:
            bounds = dset.bounds
            w = (bounds.right - bounds.left) // 2
            h = (bounds.top - bounds.bottom) // 2

            if w < minw:
                minw = w
            if h < minh:
                minh = h

        if minw < self.patch_size[0] or minh < self.patch_size[1]:
            warnings.WarningMessage("Patch size reduced to fit smallest Sentinel Dataset")
            self.patch_size = (minw, minh)

    def _find_products(self) -> list[S2Product]:
        """Recursively find images in subdirectories

        Returns:
            list of dicts of band image files
        """
        products = []

        pathname = os.path.join(self.root, "**", self.band0_glob)
        filename_regex = re.compile(self.filename_regex, re.VERBOSE)
        for filepath in glob.iglob(pathname, recursive=True):
            match = re.match(filename_regex, os.path.basename(filepath))
            if match is not None:
                try:
                    fpath = Path(filepath)
                    dset_srcs = {}

                    for band in self.bands:
                        # Find all filepaths of the individual band files
                        band_files = [p for p in fpath.parent.glob(f"T*_*_{band}_{self.resolution}m.*")]
                        if len(band_files) > 1:
                            warnings.warn(
                                f"Multiple {band} - {self.resolution}m files found within the "
                                f"{fpath.parts[-8]} dataset. ONLY OPENING THE FIRST FILE"
                            )

                        if self.cache:
                            dset_srcs[band] = self._cached_open_file(band_files[0])
                        else:
                            dset_srcs[band] = self._open_file(band_files[0])

                    # Create offset bounds from bottom left corner, b/c split_bbox uses bottom left corner
                    if self.tile_offset != [0, 0]:
                        src_left, src_bottom, src_right, src_top = dset_srcs[self.bands[0]].bounds

                        bounds = BoundingBox(
                            left=src_left + self.tile_offset[1] * self.resolution,
                            bottom=src_bottom + self.tile_offset[0] * self.resolution,
                            right=src_right,
                            top=src_top,
                        )

                    else:
                        bounds = dset_srcs[self.bands[0]].bounds

                except rasterio.errors.RasterioIOError:
                    # Skip files that rasterio is unable to read
                    continue

                else:
                    # Add to S2Product item
                    item = S2Product(bounds=bounds, bandfile_srcs=dset_srcs)
                    products.append(item)

        if len(products) == 0:
            msg = f"No {self.__class__.__name__} data was found in `root='{self.root}'`"
            msg += f" with `bands={self.bands}`"
            raise FileNotFoundError(msg)

        return products

    @functools.lru_cache(maxsize=128)
    def _cached_open_file(self, filepath: Path) -> rasterio.DatasetReader:
        return self._open_file(filepath)

    def _open_file(self, filepath) -> rasterio.DatasetReader:
        """Based on torchgeo.datasets.geo.GeoDataset._load_warp_file()"""
        src = rasterio.open(str(filepath))
        return src

    def _build_index(self) -> list[S2Patch]:
        self.index = []

        # Optional other indices if self.split == 'all', used for plotting of the dataset
        if self.split == "all":
            val_index = []
            test_index = []

        if None in self.patch_size:
            raise NotImplementedError

        for s2dset in self.s2_products:
            # Split base image into patches
            # First into groups of 2x2 patches to ensure the equal distribution of T-V-T patches
            patch_groups = self._split_bbox(s2dset.bounds, (self.patch_size[0] * 2, self.patch_size[1] * 2))
            for pgroup in patch_groups:
                # Then split into 4 patches used for train, val, or test dataset
                patches = self._split_bbox(pgroup, self.patch_size)
```

```
            # Iterate through combinations of bands
            tile_n_bands = min(self.n_bands_per_tile, len(self.bands))
            for band_group in itertools.combinations(self.bands, tile_n_bands):
                # Create list of BoundingBox, dataset_index for possible items
                # This prevents creating additional instance of the rasterio.open() objects
                items = [S2Patch(bounds=patch_bbox, bands=band_group, s2_dataset=s2dset) for patch_bbox in patches]

                if self.split == "train":
                    # Top two patches
                    self.index.extend(items[2:])
                elif self.split == "val":
                    # Bottom Left patch
                    self.index.append(items[0])
                elif self.split == "test":
                    # Bottom Right patch
                    self.index.append(items[1])

                elif self.split == "all":
                    self.index.extend(items[2:])
                    val_index.append(items[0])
                    test_index.append(items[1])

        if self.split == "all":
            # Append val and test indices
            i_len = len(self.index)
            v_len = len(val_index)
            t_len = len(test_index)

            # Save the start and stop ranges of subsets
            self.val_index_range = (i_len, i_len + v_len - 1)
            self.test_index_range = (i_len + v_len, i_len + v_len + t_len - 1)
            self.index.extend(val_index)
            self.index.extend(test_index)

    def _split_bbox(self, bbox: BoundingBox, patch_shape: tuple[int, int]) -> list[BoundingBox]:
        """Split a BoundingBox into smaller BoundingBoxes

        Args:
            bbox (BoundingBox): original bounding box to be split
            patch_shape (tuple[int, int]): (column width, row height) of new patches, in units of the
                Bounding Box, meters if using UTM coord. system.

        Returns:
            list[BoundingBox]: A list of the smaller bounding box.
        """
        patches = []

        # Integers must be used to prevent unexpected behavior, as per documentation
        xrng = np.arange(int(bbox.left), int(bbox.right), int(patch_shape[0]))
        yrng = np.arange(int(bbox.bottom), int(bbox.top), int(patch_shape[1]))

        # Determine if split evenly with no margin on edge, drops margin if present
        if (bbox.right - bbox.left) % patch_shape[0] == 0.0:
            # Add last bound to xrng, not included by arange
            xrng = np.append(xrng, [bbox.right])

        if (bbox.top - bbox.bottom) % patch_shape[1] == 0.0:
            # Add last bound to yrng, not included by arange
            yrng = np.append(yrng, [bbox.top])

        for i in range(len(xrng) - 1):
            for j in range(len(yrng) - 1):
                patches.append(BoundingBox(left=xrng[i], right=xrng[i + 1], bottom=yrng[j], top=yrng[j + 1]))

        return patches

    def _create_index_grids(self):
        """Per S2 Product, returns a grid of indices.

        The each cell in the grid corresponds to a patch in the dataset, and contains the index of
        that patch

        This can be used to determine which samples to take from a dataset.
            ie: only border patches, only center patches, top left patch, etc...
        """
        self._index_grids = {}

        # Create list of indices and (source img) pixel coordinates of the patch centers
        patch_ctrs_idx_row_col = {str(prod): [] for prod in self.s2_products}

        # Get center coordinates for each pixel
        for i, item in enumerate(self.index):
            w = item.get_window()
            patch_ctrs_idx_row_col[str(item.s2_dset)].append((i, w.row_off, w.col_off))

        for prod, idx_row_col in patch_ctrs_idx_row_col.items():
            # Group row indices and col indices
            _, rows, cols = zip(*idx_row_col)

            # Remove duplicates and order them
            row_set = sorted(set(rows))
            col_set = sorted(set(cols))

            r0 = min(row_set)
            c0 = min(col_set)
            # Compute difference in pixels between patch centers
            rstep = abs(row_set[1] - row_set[0])
            cstep = abs(col_set[1] - col_set[0])

            grid_rows = list(range(len(row_set)))
            grid_cols = list(range(len(col_set)))

            # Create array of indices
            arr = [[set() for i in grid_cols] for j in grid_rows]

            # Fill with values
            for idx, r, c in idx_row_col:
                arr[int((r - r0) / rstep)][int((c - c0) / cstep)].add(idx)

            # Save
            self._index_grids[str(prod)] = arr

    def get_exact_bounds(self, idx: int) -> Tuple[BoundingBox, CRS, Polygon]:
        """Gets the exact spatial coordinate bounds of the patch, derived from the corners of the pixels
```

```
        This differs from the "bounds" attribute, which is the bounding box calculated by splitting
        the parent dataset into smaller patches, and could potentially result in the patch bounding box
        intersecting a pixel. (but very unlikely, as bbox splitting is computed by multiples of the GSD)

        Args:
            idx (int): Dataset patch index

        Returns:
            BoundingBox,
            CRS: CRS of coordinates
            Polygon,
        """
        ul, crs = self.get_px_coords(idx, row=0, col=0, location="ul")
        ll = self.get_px_coords(idx, row=-1, col=0, location="ll")[0]
        lr = self.get_px_coords(idx, row=-1, col=-1, location="lr")[0]
        ur = self.get_px_coords(idx, row=0, col=-1, location="ur")[0]

        east_north = list(zip(ul, ll, lr, ur))

        bbox = BoundingBox(
            left=min(east_north[0]),
            bottom=min(east_north[1]),
            right=max(east_north[0]),
            top=max(east_north[1]),
        )

        return bbox, crs, Polygon([ul, ll, lr, ur])

    def get_folium_corners(self, idx: int, location="center", epsg=3857) -> list:
        """Retrieves image bounds in the form [lat_min, lon_min], [lat_max, lon_max]] for use
        with folium ImageOverlay

        Args:
            idx (int): Dataset patch index
            location (str, optional): Location to sample SRS coords in pixel. Defaults to 'center'.

        Returns:
            list: [[lat_min, lon_min], [lat_max, lon_max]]]
        """

        logger.warning("get_folium_corners() Tested only with rectangular images")

        ul = self.get_px_coords(idx, row=0, col=0, location=location, tgt_epsg=epsg)[0]
        ll = self.get_px_coords(idx, row=-1, col=0, location=location, tgt_epsg=epsg)[0]
        lr = self.get_px_coords(idx, row=-1, col=-1, location=location, tgt_epsg=epsg)[0]
        ur = self.get_px_coords(idx, row=0, col=-1, location=location, tgt_epsg=epsg)[0]

        lat, lon = zip(ul, ll, lr, ur)

        return [[min(lat), min(lon)], [max(lat), max(lon)]]

    def get_all_srcs(self):
        """Retrieves a dictionary of all src objects used in this dataset, per band"""
        srcs = {b: set() for b in self.bands}
        for p in self.s2_products:
            for b, s in p.bandfile_srcs.items():
                srcs[b].add(s)
        return srcs

    def warp_patch_band(
        self,
        idx_or_poly: Union[int, Polygon],
        band: str,
        dest_crs: str = "EPSG:4326",
        angle=0,
        shift_x=0,
        shift_y=0,
    ):
        """Retrieve patch band array and warp to a different CRS

        WARNING: Not robustly implemented. Use caution when using with multi-file datasets

        The returned transform accurately georeferences the coordinates of the rotated/translated
        image to SRS. This has been verified by mapping and checking that the lower left and upper
        right corner coordinates as returned by the second return argument are actually located on
        the correct map

        Args:
            idx_or_src (int or rasterio.windows.Window): either the known dataset index, or
                a Window object to calculate the subset
            band (str): _description_
            dest_crs (_type_, optional): _description_. Defaults to "EPSG:3857".
            angle (float): angle in degrees to rotate clockwise
            shift_x (int): meters to shift east
            shift_y (int): meters to shift north

        Returns:
            np array: Reprojected image
            list: [[lat_min, lon_min], [lat_max, lon_max]] of new image
            AffineTransform: transform from result pixel CRS to destination spatial CRS
        """
        if isinstance(band, list):
            if len(band) == 1:
                band = band[0]

        if isinstance(idx_or_poly, int):
            idx = idx_or_poly
            patch = self.index[idx]
            src = self.index[idx].srcs[band]
            window = patch.get_window()

        elif isinstance(idx_or_poly, Polygon):
            src = list(self.get_all_srcs()[band])[0]
            window = geometry_window(src, [idx_or_poly], 0, 0)

        source_arr = src.read(1, window=window).squeeze()
        source_h, source_w = source_arr.shape

        # Compute rotation transform and create empty destination raster with appropriate size
        cos_theta = np.cos(np.abs(np.radians(angle)))
        sin_theta = np.sin(np.abs(np.radians(angle)))

        new_height = int(source_h * cos_theta + source_w * sin_theta)
        new_width = int(source_w * cos_theta + source_h * sin_theta)

        # Calculate the position to paste the original image in the center of the blank image
```

```
        offset_x = (new_width - source_w) // 2
        offset_y = (new_height - source_h) // 2

        shift = Affine.translation(shift_x, -shift_y)

        # Bounds are still in the window crs, but including the explicit shift
        window_transform = src.window_transform(window) * shift
        win_transformer = AffineTransformer(window_transform)

        ll = win_transformer.xy(source_h + offset_y, -offset_x)
        ur = win_transformer.xy(-offset_y, source_w + offset_x)
        affine_bounds = [*ll, *ur]  # left, bottom, right, top Bounding coordinates in src.crs

        # Calculate the destination transform and dimensions for the CRS change
        tgt_crs_transform, crs_width, crs_height = calculate_default_transform(
            src.crs,
            dest_crs,
            new_width,  # Source raster width
            new_height,  # Source raster height
            *affine_bounds,  # left, bottom, right, top Bounding coordinates in src_crs,
        )

        # Define rotation transformations
        rotate = Affine.rotation(angle, pivot=(source_w // 2, source_h // 2))
        trans = Affine.translation(-offset_x, -offset_y)  # Compensate for the rotation, don't ask...

        rotate_transform = window_transform * rotate * trans

        # Manually create padded array with original image in the center
        padded_rotate_arr = np.zeros((new_height, new_width))
        padded_rotate_arr[offset_y : offset_y + source_h, offset_x : offset_x + source_w] = source_arr

        crs_arr = np.zeros((crs_height, crs_width))
        dest_arr, dst_transform = reproject(
            source=padded_rotate_arr,
            destination=crs_arr,
            src_transform=rotate_transform,
            src_crs=src.crs,
            dst_transform=tgt_crs_transform,
            dst_crs=dest_crs,
            resampling=Resampling.nearest,
        )

        # Reverse xy lists to produce lat, lon pairs of lower-left and upper right
        dst_transformer = AffineTransformer(dst_transform)
        n_rows, n_cols = dest_arr.shape

        ll = list(dst_transformer.xy(n_rows - 1, 0))[::-1]
        ur = list(dst_transformer.xy(0, n_cols - 1))[::-1]

        return dest_arr, [ll, ur], dst_transform

    def get_px_coords(self, idx: int, row: int, col: int, location="center", tgt_epsg=None) -> tuple[tuple[float], CRS]:
        """Get spatial coordinates of a pixel's center of a given patch.

        Args:
            idx (int): Dataset patch index
            row (int): row pixel coordinate
            col (int): column pixel coordinate
            location (str): location relative to pixel to get coordinates. Default "center". Options
                are: 'ul', 'll', 'ur', 'lr' (upper-left, lower-left, upper-right, lower-right).
            tgt_epsg (int): Target EPSG CRS to transform x, y coordinates to. Default None for no transform

        Returns:
            tuple[tuple[float], CRS]: (x-coord, y-coord), CRS

        Also returns the CRS of the patch

        Reference:
        https://rasterio.readthedocs.io/en/stable/api/rasterio.io.html#rasterio.io.DatasetReader.xy
        """

        if row < 0:
            row = self.px_tile_size[0] + row

        if col < 0:
            col = self.px_tile_size[1] + col

        src = self.index[idx].get_src0()
        window = self.index[idx].get_window()

        x, y = src.xy(
            row + window.row_off,
            col + window.col_off,
            offset=location,
        )
        if tgt_epsg is not None:
            to_crs = CRS.from_epsg(tgt_epsg)
            tx_fn = Transformer.from_crs(src.crs, to_crs)
            return tx_fn.transform(x, y), to_crs

        return (x, y), src.crs

    def get_all_crs(self):
        """Get a list of all coordinate reference systems used by this dataset"""
        return list(set([s2p.crs for s2p in self.s2_products]))

    def get_index_grids(self):
        return self._index_grids

    def get_interior_tile_indices(self):
        """Returns dataset indices of all non-edge tiles of each s2_product"""
        idxs = []
        for arr in self._index_grids.values():
            for row in arr[1:-1]:
                for idx_set in row[1:-1]:
                    idxs.extend(list(idx_set))

        return sorted(idxs)


class RegisS2DataModule(NonGeoDataModule):
    """LightningDataModule implementation for the RegisSentinel2Dataset

    Uses the train/va/test splits from the dataset
    """
```

```
    def __init__(
        self,
        batch_size: int = 16,
        num_workers: int = 0,
        cache: bool = True,
        **kwargs,
    ):
        """Initialize new RegisS2DataModule

        Args:
            batch_size (int, optional): Number of samples per batch. Defaults to 16.
            num_workers (int, optional): How many subprocesses to use for data loading. ``0`` means that the data will be loaded in the main process.
                Defaults to 0.
            **kwargs: Additional keyword arguments passed to ``RegisSentinel2Dataset`` class
        """
        # Most kwargs appear unused, but are automatically saved by the save_hyperparameters()
        #   method. The advantage of listing them here is that defaults can be set, and can be
        #   documented in the docstring for future implementations
        # PEP8 warning just need to be ignored
        self.save_hyperparameters(
            kwargs,
            ignore=[
                "download",
            ],
        )

        super().__init__(
            RegisSentinel2Dataset,
            cache=cache,
            batch_size=batch_size,
            num_workers=num_workers,
            **kwargs,
        )
```

## C.2. D2-Net Keypoint Extraction

The following code is a module used to interface the D2-Net model source code with the overall system simulation. All source code in the relative directory `.d2_net` is equivalent to that created by the original authors of D2-Net, which can be found at https://github.com/mihaidusmanu/d2-net/tree/master/lib

**Listing C.2:** regis_d2net.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Description   : Functions to interface Regis with the D2-Net model
Author        : Luigi Maiorano (lmaiorano97@gmail.com)
Creation Date : 01-10-2023
References :
"""
import torch
import torch.nn.functional as F

from .d2_lib.model_test import D2Net
from .d2_lib.utils import interpolate_dense_features, upscale_positions


class RegisD2NetModel(D2Net):
    def __init__(self, model_file, top_k, use_relu=True, dtype=torch.float32):
        assert model_file.suffix == ".pth", "D2-Net␣Model␣file␣must␣be␣a␣.pth␣file"
        super().__init__(
            model_file=str(model_file),
            use_relu=use_relu,
            use_cuda=torch.cuda.is_available(),
            dtype=dtype,
        )

        self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
        self.top_k = top_k

    def forward(self, x) -> tuple[torch.Tensor, torch.Tensor, torch.Tensor]:
        """Extracts features from an image using the D2-Net model

        Returns
        -------
        tk_combined : torch.Tensor
            Array containing the descriptors and coordinates of the top_k keypoints.
            shape = [1, descr_dim + 2, k] were the last 2 values of dim 1 are row and col

        tk_scores : torch.Tensor
            Tensor containing the scores of the top_k keypoints. Shape = [1, k]
        # r_c_loc_coords : list
        #     List of tuples containing (row, col, pixel_location) of the top_k keypoints.
        tk_descr : torch.Tensor
            Array containing the descriptors of the top_k keypoints. shape = [1, descr_dim, k]
        tk_coords : torch.Tensor
            Array containing the coordinates of the top_k keypoints, [row, col]. shape = [1, 2, k]
        """
        # Repeat x in the channel dimension to match the input size of the model
        x = torch.concat([x, x, x], dim=1)  # Must use concat for ONNX compatibiltiy
        x = x.to(self.device)

        dense_features = self.dense_feature_extraction(x)
        detections = self.detection(dense_features)
        displacements = self.localization(dense_features)

        # Process D2-Net output
        keypoints, scores, descriptors = self.process(
            device=self.device,
            dense_features=dense_features,
            detections=detections,
            displacements=displacements,
        )

        # Select top_k keypoints
        if self.top_k == 0:
            self.top_k = len(keypoints[:, 0])
```

```
        top_k_idxs = torch.argsort(scores.squeeze(), descending=True)
        top_k_idxs = top_k_idxs[: self.top_k]

        tk_coords = keypoints[top_k_idxs, :]
        tk_scores = scores[top_k_idxs]
        tk_descr = descriptors[top_k_idxs, :]

        # reshape to match RegisModel output
        tk_coords_f = torch.floor(tk_coords.permute(1, 0).unsqueeze(0))
        tk_coords_f = torch.flip(tk_coords_f, dims=[1]).clone()  # Clone operation required for later operations
        tk_scores = torch.unsqueeze(tk_scores, dim=0)
        tk_descr = torch.unsqueeze(tk_descr.permute(1, 0), dim=0)

        return tk_scores, tk_descr, tk_coords_f

    def process(self, device, dense_features, detections, displacements):
        """Copied from d2_lib/pyramid.py, but removed extra code since no scales are used"""

        fmap_pos = torch.nonzero(detections[0].cpu()).t()
        del detections

        # Recover displacements.
        displacements = displacements[0].cpu()
        displacements_i = displacements[0, fmap_pos[0, :], fmap_pos[1, :], fmap_pos[2, :]]
        displacements_j = displacements[1, fmap_pos[0, :], fmap_pos[1, :], fmap_pos[2, :]]
        del displacements

        # mask = torch.logical_and(torch.abs(displacements_i) < 0.5, torch.abs(displacements_j) < 0.5)
        mask = (torch.abs(displacements_i) < 0.5) & (torch.abs(displacements_j) < 0.5)

        fmap_pos = fmap_pos[:, mask]
        valid_displacements = torch.stack([displacements_i[mask], displacements_j[mask]], dim=0)
        del mask, displacements_i, displacements_j

        fmap_keypoints = fmap_pos[1:, :].float() + valid_displacements
        del valid_displacements

        raw_descriptors, _, ids = interpolate_dense_features(fmap_keypoints.to(device), dense_features[0])

        fmap_pos = fmap_pos[:, ids.cpu()]
        fmap_keypoints = fmap_keypoints[:, ids.cpu()]
        del ids

        keypoints = upscale_positions(fmap_keypoints, scaling_steps=2)
        del fmap_keypoints

        descriptors = F.normalize(raw_descriptors, dim=0).cpu()
        del raw_descriptors

        fmap_pos = fmap_pos.cpu()
        keypoints = keypoints.cpu()

        scores = dense_features[0, fmap_pos[0, :], fmap_pos[1, :], fmap_pos[2, :]].cpu()
        del fmap_pos
        del dense_features

        keypoints = keypoints.t().detach()
        scores = scores.detach()
        descriptors = descriptors.t().detach()
        return keypoints, scores, descriptors
```

## C.3. RFDB Generation

This code is used to create, save and load the reference features database.

**Listing C.3:** rfdb_database.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Description   : Fucntions for initializing and interacting with the database
Author        : Luigi Maiorano (lmaiorano97@gmail.com)
Creation Date : 26-07-2023
References :
"""
import os
from pathlib import Path

import geopandas as gpd
import lightning.pytorch as pl
import numpy as np
import pandas as pd
import torch
import yaml
from loguru import logger
from pyproj import CRS
from shapely.geometry import Point
from src.keypoint_pipeline import RegisD2NetModel, RegisSentinel2Dataset, RegisUNetModel
from tqdm import tqdm


def create_ref_database(args, rfdb_crs=4326):
    pl.seed_everything(args.seed)

    dataset = RegisSentinel2Dataset(
        root=args.data_directory,
        bands=args.img_bands,
        split=args.data_split,
        res=args.img_res,
        px_tile_size=args.img_tile_size,
        extended_getitem=True,
        n_bands_per_tile=args.n_bands_per_tile,
    )

    # Save s2_product names to a file for record
    products = {prod.name: prod.to_json() for prod in dataset.s2_products}
    prod_metadata_file = Path(os.getenv("RESULTS_DIR"), "products.yml")
    with prod_metadata_file.open("w") as f:
        yaml.dump(products, f)
```

```

    if args.rfdb_epsg != 0:
        rfdb_crs = CRS.from_epsg(args.rfdb_epsg)
    else:
        if len(dataset.get_all_crs()) > 1:
            logger.error("Multiple CRSs found in dataset, and no common CRS specified for the RF database.")
            raise ValueError()
        else:
            rfdb_crs = dataset.get_all_crs()[0]
            logger.info(f"CRS of the RFDB set to {rfdb_crs}")

    if args.model == "unet":
        model = RegisUNetModel.load_from_checkpoint(
            args.weights_checkpoint_path,
            top_k=args.top_k,
        )
        model.eval()  # disable randomness, dropout, etc...

    elif args.model == "d2net":
        model = RegisD2NetModel(
            model_file=args.weights_checkpoint_path,
            top_k=args.top_k,
            use_relu=args.use_relu,
        )
        model.eval()

    else:
        raise ValueError(f"Model {args.model} not recognized")

    # The dataset may contain products from multiple CRSes
    # Each should be loaded into its own GeoDF, which can then be transformed to a singular CRS and
    # combined
    data_per_crs = {
        str(crs): {
            "spatial_xy": [],
            "kpts": [],
            "bands": [],
        }
        for crs in dataset.get_all_crs()
    }

    n_tiles = len(dataset)
    if args.n_tiles != 0:
        n_tiles = args.n_tiles

    # for band in tqdm(args.img_bands, desc="Bands"):
    for patch_idx in tqdm(range(n_tiles), leave=False, desc="Patches"):
        sample = dataset[patch_idx]
        x_b = []
        for band in sample["bands_order"]:
            x_b.append(dataset.tile_band_tensor(tile=sample, band=band))
        x = torch.cat(x_b, dim=1)

        # Extract keypoints
        if args.model == "unet":
            _, _, keypts_descr, keypts_rc = model(x)

            # Get pixel coordinates of the keypoints relative to the input image,
            # such that spatial coordinates can be retrieved of those pixels
            feat_img_coords = model.map_featRC_to_imageRC(keypts_rc).squeeze()

            # Get Spatial Coordinates of each keypoint
            for i, p in enumerate(feat_img_coords.T.tolist()):
                spatial_coords, crs = dataset.get_px_coords(idx=patch_idx, row=p[0], col=p[1], location="center")
                desc = keypts_descr[0, :, i]

                crs_key = str(crs)

                data_per_crs[crs_key]["kpts"].append(desc.detach().numpy())
                data_per_crs[crs_key]["spatial_xy"].append(Point(*spatial_coords))
                data_per_crs[crs_key]["bands"].append(band)

        elif args.model == "d2net":
            keypts_score, keypts_descr, keypts_rc = model(x)

            # Get Spatial Coordinates of each keypoint
            for i in range(keypts_rc.shape[-1]):
                r = keypts_rc[0, 0, i].item()
                c = keypts_rc[0, 1, i].item()
                spatial_coords, crs = dataset.get_px_coords(idx=patch_idx, row=r, col=c, location="center")
                desc = keypts_descr[0, :, i]

                crs_key = str(crs)

                data_per_crs[crs_key]["kpts"].append(desc.detach().numpy())
                data_per_crs[crs_key]["spatial_xy"].append(Point(*spatial_coords))
                data_per_crs[crs_key]["bands"].append(band)

        else:
            raise ValueError(f"Model {args.model} not recognized")

    # Construct geodataframes of each CRS
    crs_dfs = []
    for crs_str, data in data_per_crs.items():
        df = gpd.GeoDataFrame(
            data={"band": data["bands"], "feat_descr": data["kpts"]},
            geometry=data["spatial_xy"],
            crs=CRS.from_string(crs_str),
        )
        df = df.to_crs(rfdb_crs)  # Convert to rfdb crs
        crs_dfs.append(df)

    df = pd.concat(crs_dfs)

    logger.info(f"Database contains {len(df)} features")

    if not args.db_save_dir.exists():
        args.db_save_dir.mkdir(parents=True)

    logger.info(f"Saving database to {args.db_save_dir}. This may take a while...")
    db_save_path = Path(args.db_save_dir, "reference_feature_database.gpkg")

    df["feat_descr"] = df["feat_descr"].apply(lambda x: np.array2string(x))

    df.to_file(db_save_path, driver="GPKG")

```

```

def load_db(db_directory, bbox=None, mask=None):
    assert bbox is None or mask is None  # only bbox or mask can be provided

    db_file = Path(db_directory, "reference_feature_database.gpkg")

    # Load Database
    df = gpd.read_file(str(db_file), bbox=bbox, mask=mask)
    # Convert string representation of np arr back to array, ignore first and last [ ] brackets
    df["feat_descr"] = df["feat_descr"].apply(lambda x: np.fromstring(x[1:-1], sep="␣"))

    return df
```

## C.4. Feature Matching

These modules are used to match features between images and perform the georeferencing process. The `transformer.py` module is the main module used to perform the georeferencing process, and the `matching.py` module is used to match features between images.

**Listing C.4:** matching.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Description   : Methods to match the sets of descriptors
Author        : Luigi Maiorano (lmaiorano97@gmail.com)
Creation Date : 11-08-2023
References :
"""

import cv2
import numpy as np
import pandas as pd
import geopandas as gpd
from loguru import logger
from rasterio.control import GroundControlPoint as GCP


def create_gcp_row(row):
    "Creates␣a␣Rasterio.control␣Ground␣Control␣Point␣by␣combining␣column␣values"
    return GCP(row=row.qi_px_row, col=row.qi_px_col, x=row.geometry.x, y=row.geometry.y)


def brute_force_match(
    descr_img_coords,
    ri_gdf: gpd.GeoDataFrame,
    ri_tnsr,
    qi_tnsr,
    m_matches: int = -1,
    **kwargs,
) -> gpd.GeoDataFrame:
    """Quick implementation of the OpenCV Brue-Force Matcher

    https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

    Args:
        descr_img_coords (_type_): Tensor of query image descriptors. Shape = [2, n_descr]
        ri_gdf (_type_): GeoDataFrame of reference image keypoints. With columns: [geometry (Point), feat_descr, band]
        ri_tnsr (_type_): Tensor of reference image descriptors Shape = [n_descr, descr_dim]
        qi_tnsr (_type_): Tensor of query image descriptors. Shape = [descr_dim, n_descr]
        k_matches (int, optional): Returns m best matches where m is specified by the user.
            Defaults to -1 to return all.

    Returns:
        _type_: _description_
    """

    # Convert to numpy arrays convert to float 32 for OpenCV compatibility
    ref_descr = (ri_tnsr.detach().numpy()).astype(np.float32)
    qi_descr = (qi_tnsr.detach().numpy().T).astype(np.float32)

    # create BFMatcher object
    bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

    # Match descriptors.
    matches = bf.match(qi_descr, ref_descr)

    # Sort them in the order of their descriptive distance.
    matches = sorted(matches, key=lambda x: x.distance)

    # Select only the top m_matches
    matches = matches[:m_matches]

    if m_matches > -1 and (tot := len(matches)) < m_matches:
        logger.info(f"Less␣successful␣keypoint␣matches␣made␣({tot})␣than␣was␣requested␣({m_matches}).")

    matched_ri_idxs = [x.trainIdx for x in matches]

    # # Also converts index of descr [0 to K] to index of the pixel in the image of each top-K descr
    # feat_img_coords = model.map_featRC_to_imageRC(keypts_rc).squeeze()

    match_idxs = [x.queryIdx for x in matches]
    matched_keypts_rc = descr_img_coords[:, match_idxs]

    # Get the subset of RI keypoints that match QI keypoints
    matched_ri_df = ri_gdf.iloc[matched_ri_idxs].rename_axis("ri_df_index").reset_index()

    matched_ri_df["qi_px_row"] = pd.Series(matched_keypts_rc[0, :].tolist())
    matched_ri_df["qi_px_col"] = pd.Series(matched_keypts_rc[1, :].tolist())

    # Create Rasterio GCP objects
    gcp_series = []
    for _, row in matched_ri_df.iterrows():
        gcp_series.append(create_gcp_row(row))
    matched_ri_df["gcp"] = pd.Series(gcp_series)

    return matched_ri_df
```

## C.5. Georeferencing

**Listing C.5:** transformer.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Description   : ctype implementation of the RasterIO GCPTransformer to fix a bug when transforming row,col to xy
Author        : Luigi Maiorano (lmaiorano97@gmail.com)
Creation Date : 11-09-2023
References :
https://gis.stackexchange.com/questions/261239/how-to-extract-transformation-coefficients-calculated-with-gdal-based-on-gcps
"""

import ctypes as ct
import ctypes.util as ctu

import numpy as np
from rasterio.control import GroundControlPoint

libgdal = ct.CDLL(ctu.find_library("gdal"))


class GeorefGCPs:
    def __init__(self, gcps: list[GroundControlPoint], refine=False, refine_tolerance=1.0):
        """Creates a GDAL GCPTransformer object that can be used to transform row,col to x,y.

        This should be used in a context manager to ensure the transformer is destroyed after use.
        (Python 'with' statement)

        Example:
            with GeorefGCPs(gcps) as transformer:
                for row, col in zip(rows, cols):
                    x, y = transformer.xy(row, col)
                    # do something with x, y

        For Refine=True, see following thread for documentation:
            https://trac.osgeo.org/gdal/ticket/4143

        Args:
            gcps (list[list]): List of GCP elements [[g.row, g.col, g.x, g.y]]
            refine (bool): Removes the worst GCP outliers iteratively until the minimum number of GCPs are
                reached or no outliers can be detected
            refine_tolerance (float): In pixel units if no projection is available, otherwise it is in
                SRS (spatial reference system) units.
        """

        gcp_vals = [[g.row, g.col, g.x, g.y] for g in gcps]

        GDALCreateGCPTransformer = libgdal.GDALCreateGCPTransformer
        GDALCreateGCPTransformer.restype = ct.POINTER(GCPTransformInfo)

        GDALCreateGCPRefineTransformer = libgdal.GDALCreateGCPRefineTransformer
        GDALCreateGCPRefineTransformer.restype = ct.POINTER(GCPTransformInfo)

        c_gcps = (Gcp * len(gcp_vals))()
        for i, pt in enumerate(gcp_vals):
            gcp = c_gcps[i]
            gcp.pszId = str(i).encode()
            gcp.pszInfo = "".encode()
            gcp.dfGCPLine = pt[0]  # row
            gcp.dfGCPPixel = pt[1]  # col
            gcp.dfGCPX = pt[2]  # X
            gcp.dfGCPY = pt[3]  # Y
            gcp.dfGCPZ = 0

        nReqOrder = 0  # highest order possible (limited to 2) with the provided gcp count will be used
        bReversed = 0

        if refine:
            # Min num gcps, 3 for 1st order 6 for 2nd order
            n_minimum_gcps = 6 if len(gcps) >= 6 else 3

            self.transformer = libgdal.GDALCreateGCPRefineTransformer(
                len(c_gcps),
                c_gcps,
                nReqOrder,
                bReversed,
                ct.c_double(refine_tolerance),
                ct.c_int(n_minimum_gcps),
            )

        else:
            self.transformer = libgdal.GDALCreateGCPTransformer(len(c_gcps), c_gcps, nReqOrder, bReversed)

    def __enter__(self):
        return self

    def __exit__(self, *args):
        """For use with context managers

        Based on:
        https://github.com/rasterio/rasterio/blob/5e001a08beb747fef0b60b5ccdbf2e5a08da7e05/rasterio/_transform.pyx#L352
        https://stackoverflow.com/questions/3724987/python-ctypes-c-object-destruction
        """
        libgdal.GDALDestroyGCPTransformer(self.transformer)
        self.transformer = ct.c_void_p

    def xy(self, row, col):
        """Python implementation of GDAL CRS_Georef

        Reference:
            https://github.com/OSGeo/gdal/blob/75134407dae4dff36b952c85d43f6e3e5a22f4ea/gdal/alg/gdal_crs.cpp#L608

        Args:
            row (int): _description_
            col (int): _description_
            tx (GDALCreateGCPTransformer): GDAL Transformer object

        Returns:
            tuple: easting, northing (x, y)
        """
        e1 = col - self.transformer.contents.x1_mean
        n1 = row - self.transformer.contents.y1_mean
        E = list(np.array(self.transformer.contents.adfToGeoX))
```

```
        N = list(np.array(self.transformer.contents.adfToGeoY))

        if self.transformer.contents.nOrder == 1:
            e = E[0] + E[1] * e1 + E[2] * n1
            n = N[0] + N[1] * e1 + N[2] * n1

        elif self.transformer.contents.nOrder == 2:
            e2 = e1**2
            n2 = n1**2
            en = e1 * n1

            e = E[0] + E[1] * e1 + E[2] * n1 + E[3] * e2 + E[4] * en + E[5] * n2
            n = N[0] + N[1] * e1 + N[2] * n1 + N[3] * e2 + N[4] * en + N[5] * n2

        else:
            raise NotImplementedError

        return e, n


class Gcp(ct.Structure):
    _fields_ = [
        ("pszId", ct.c_char_p),
        ("pszInfo", ct.c_char_p),
        ("dfGCPPixel", ct.c_double),
        ("dfGCPLine", ct.c_double),
        ("dfGCPX", ct.c_double),
        ("dfGCPY", ct.c_double),
        ("dfGCPZ", ct.c_double),
    ]


class GDALTransformerInfo(ct.Structure):
    _fields_ = [
        ("abySignature", ct.c_byte * 4),
        ("pszClassName", ct.c_char_p),
        ("pfnTransform", ct.c_void_p),
        ("pfnCleanup", ct.c_void_p),
        ("pfnSerialize", ct.c_void_p),
        ("pfnCreateSimilar", ct.c_void_p),
    ]


class GCPTransformInfo(ct.Structure):
    _fields_ = [
        ("sTI", GDALTransformerInfo),
        ("adfToGeoX", ct.c_double * 20),
        ("adfToGeoY", ct.c_double * 20),
        ("adfFromGeoX", ct.c_double * 20),
        ("adfFromGeoY", ct.c_double * 20),
        ("x1_mean", ct.c_double),
        ("y1_mean", ct.c_double),
        ("x2_mean", ct.c_double),
        ("y2_mean", ct.c_double),
        ("nOrder", ct.c_int),
        ("bReversed", ct.c_int),
        ("bRefine", ct.c_int),
        ("nMinimumGcps", ct.c_int),
        ("dfTolerance", ct.c_double),
        ("nRefCount", ct.c_int),
    ]

    def __str__(self):
        s = ct.string_at(self.sTI.pszClassName).decode() + "\n"
        s += "adfToGeoX:\n"
        for i in range(0, 20):
            s += "  %d: %g\n" % (i, self.adfToGeoX[i])
        s += "adfToGeoY:\n"
        for i in range(0, 20):
            s += "  %d: %g\n" % (i, self.adfToGeoY[i])
        s += "adfFromGeoX:\n"
        for i in range(0, 20):
            s += "  %d: %g\n" % (i, self.adfFromGeoX[i])
        s += "adfFromGeoY:\n"
        for i in range(0, 20):
            s += "  %d: %g\n" % (i, self.adfFromGeoY[i])
        s += "x1_mean: %g\n" % self.x1_mean
        s += "y1_mean: %g\n" % self.y1_mean
        s += "x2_mean: %g\n" % self.x2_mean
        s += "y2_mean: %g\n" % self.y2_mean
        s += "nOrder: %d\n" % self.nOrder
        s += "bReversed: %s\n" % (self.bReversed != 0)
        return s
```

# C.6. System Simulation

**Listing C.6:** simulation.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Description   : REGIS Simulation
Author        : Luigi Maiorano (lmaiorano97@gmail.com)
Creation Date : 26-07-2023
References :
"""
import os
import sys
from pathlib import Path

import geopandas as gpd
import lightning.pytorch as pl
import numpy as np
import pandas as pd
import torch
import yaml
from deepdiff import DeepDiff
from loguru import logger
from shapely.geometry import Point
from src.georef.analysis import (
    analyze_gcp_data,
    analyze_icp_data,
```

```
    gcp_accuracy,
    georef_accuracy,
    icp_accuracy,
    rmse_to_ce90,
)
from src.georef.classes import GCPCollection
from src.georef.database import load_db
from src.georef.matching import brute_force_match
from src.keypoint_pipeline import RegisD2NetModel, RegisSentinel2Dataset, RegisUNetModel
from tqdm import tqdm


def regis_simulation(args, suppress_stdout=False):
    pl.seed_everything(args.seed)

    os.environ["AUTO_OPEN_PLOTS"] = "true"
    if args.no_auto_open:
        os.environ["AUTO_OPEN_PLOTS"] = "false"

    if suppress_stdout:
        logger.disable("src")  # Disable logging for the module 'src' and all children

    db_args = load_rfdb_args(args.db_path)

    if "ots" in str(args.weights_checkpoint_path):
        args.use_relu = False

    # Sanity check the arguments used for RFDB creation vs currrent inference
    args_diff = compare_args_sim_rfdb(args, db_args, ignore_keys=["n_tiles", "data_split", "top_k"])
    if args_diff is not None:
        msg = "\nDifferent arguments passed between SIM (current) and RFDB-generation runs:"
        for k, v in args_diff.items():
            k = k.split("'")
            name = f"{k[1]}:"
            new = f'"{v["new_value"]}"'
            msg += f'\n\t{name: <15} sim = {new: <15} rfdb = "{v["old_value"]}"'
        logger.warning(msg)

    qi_dataset = RegisSentinel2Dataset(
        root=args.data_directory,
        bands=args.img_bands,
        split=args.data_split,
        res=args.img_res,
        px_tile_size=args.img_tile_size,
        extended_getitem=True,
        tile_offset=args.tile_offset_px,
    )

    if args.model == "unet":
        model = RegisUNetModel.load_from_checkpoint(
            args.weights_checkpoint_path,
            top_k=args.top_k,
        )
        model.eval()  # disable randomness, dropout, etc...

    elif args.model == "d2net":
        model = RegisD2NetModel(
            model_file=args.weights_checkpoint_path,
            top_k=args.top_k,
            use_relu=args.use_relu,
        )
        model.eval()

    assert len(args.img_bands) == 1, "Only single-band Query Images supported for now"
    qi_band = args.img_bands[0]

    interior_idxs = qi_dataset.get_interior_tile_indices()

    sample_indices = interior_idxs[args.n_tiles - 1 : args.n_tiles]  # Create range of indices to sample
    if args.n_tiles is None or args.n_tiles == 0:  # Sample ALL internal tiles if argument is left as none
        logger.info(f"Argument --n-tiles not set. Proceeding to sample all {len(interior_idxs)} internal tiles")
        sample_indices = interior_idxs

    icp_grid_dim = args.icps_grid_dim
    logger.info(f"Accuracy evaluations conducted with regular grid of {icp_grid_dim**2} Independent Check Points")

    # Raw data columns
    raw_data_gcp_cols = ["dset_sample_idx", "RF_eastnorth", "QI_colrow", "QI_eastnorth"]
    raw_data_icp_cols = ["dset_sample_idx", "truth", "predicted"]

    # Initialize parameters
    ICP_gdfs = []
    GCP_gdfs = []
    for idx in tqdm(sample_indices, leave=False, desc="Patches", disable=suppress_stdout):
        if args.last_map_only:
            args.disable_map = True
            if idx == sample_indices[-1]:
                args.disable_map = False

        # get QI
        sample = qi_dataset[idx]
        img_x = qi_dataset.tile_band_tensor(tile=sample, band=qi_band)

        # ------------------------------
        # Reference Keypoints
        # ------------------------------
        # Get QI center and diagonal dimension to simulate initial position
        qi_center = bbox_center_point(sample)
        qi_max_dim = get_tile_diag(sample)

        # Load reference keypoints, for now using the QI sample as a source of initial position
        ri_gdf, ri_mask_gs, ri_df = get_reference_keypoints(args.db_path, qi_center, qi_max_dim, sample_crs=sample["crs"])

        # Convert Ref Features Dataframe to tensor
        rf_tnsr = torch.from_numpy(np.stack(ri_gdf["feat_descr"])).float()

        # ------------------------------
        # Query Image Keypoints
        # ------------------------------
        # Extract QI features
        if args.model == "unet":
            _, _, keypts_descr, keypts_rc = model(img_x)
            qi_descr_tnsr = keypts_descr.squeeze().detach()  # shape = [descr_dim, k]
            feat_img_coords = model.map_featRC_to_imageRC(keypts_rc).squeeze()  # shape = [2, k]

        elif args.model == "d2net":
```

```
            keypts_score, keypts_descr, keypts_rc = model(img_x)

            # Match RegisUNet format
            qi_descr_tnsr = keypts_descr.squeeze().detach()  # shape = [descr_dim, k]
            keypts_rc = keypts_rc.squeeze().detach()  # shape = [2, k]
            feat_img_coords = keypts_rc

        # Match Keypoints
        # ---------------------------
        matched_ri_df = brute_force_match(
            descr_img_coords=feat_img_coords,
            ri_gdf=ri_gdf,
            ri_tnsr=rf_tnsr,
            qi_tnsr=qi_descr_tnsr,
            m_matches=args.best_m_matches,
            patch_idx=idx,
            dset=qi_dataset,
        )

        gcps = GCPCollection(matched_ri_df["gcp"], crs=matched_ri_df.crs)
        # Create record of GCPs for saving later
        gcp_gdf = gpd.GeoDataFrame(
            data=list(
                zip(
                    [idx for i in range(len(gcps))],
                    [Point(g.x, g.y) for g in gcps],
                    [Point(g.col, g.row) for g in gcps],
                    [
                        Point(
                            qi_dataset.get_px_coords(
                                idx=idx,
                                row=g.row,
                                col=g.col,
                                tgt_epsg=gcps.crs.to_epsg(),
                            )[0]
                        )
                        for g in gcps
                    ],
                )
            ),
            columns=raw_data_gcp_cols,
            geometry=raw_data_gcp_cols[1],
            crs=gcps.crs,
        )
        GCP_gdfs.append(gcp_gdf)

        # Evaluate GCP matching accuracy
        gcp_gdf, (gcp_rmse_x, gcp_rmse_y, gcp_rmse_comb, n_corr_match) = gcp_accuracy(gcp_gdf, success_match_dist=args.gcp_match_distance)

        logger.info(f"Using {len(gcps)} matched GCPs; {n_corr_match} considered 'successful' (within " f"{args.gcp_match_distance}m).")
        logger.debug(f"GCP RMSE_x: {gcp_rmse_x:.2f}m, RMSE_y: {gcp_rmse_y:.2f}m, " f"Combined: {gcp_rmse_comb:.5f}")

        if len(gcps) < 6:
            logger.warning(f"{len(gcps)} GCPs used to create 1ST ORDER POLYNOMIAL. " "2nd-order requires at least 6 GCPs.")

        else:
            logger.debug(f"{len(gcps)} GCPs used to create 2ND ORDER polynomial")
            logger.debug(
                "3rd-order polynomials unstable and currently unsupported, see GDAL docs: "
                "https://gdal.org/api/gdal_alg.html#_CPPv424GDALCreateGCPTransformeriPK8GDAL_GCPii"
            )

        # -------------------------
        # Evaluate Georef Accuracy
        # -------------------------
        try:
            (
                rmse_x,
                rmse_y,
                rmse_comb,
                ICPs_actual,
                ICPs_mapped,
                icp_df,
            ) = georef_accuracy(
                sample,
                img_x,
                gcps,
                dataset=qi_dataset,
                icps_gdf_cols=raw_data_icp_cols,
                icp_grid_dim=icp_grid_dim,
                xy_loc=args.px_xy_location,
                ransac=args.ransac,
                refine_tolerance=args.ransac_tol,
            )
            ICP_gdfs.append(icp_df)
            logger.info(
                f"Tile {idx} RMSE_x: {rmse_x:.2f}m, RMSE_y: {rmse_y:.2f}m, Combined: {rmse_comb:.5f}, " f"CE90: {rmse_to_ce90(rmse_comb):.5f}m"
            )

        except ValueError as ve:
            logger.error(f"Error processing tile {idx}, likely unable to compute solution with matched GCPs. See error: {ve}")

    # (Verified) All GDF CRS's are the same. This is by design and tested
    master_ICP_gdf = pd.concat(ICP_gdfs, ignore_index=True)
    master_GCP_gdf = pd.concat(GCP_gdfs, ignore_index=True)

    master_GCP_gdf, (gcp_x, gcp_y, gcp_rmse, nmatch) = gcp_accuracy(master_GCP_gdf)

    logger.success(
        "\nMatched GCP Averages:\n"
        f"\tAvg num correct matches (within {args.gcp_match_distance}m): {nmatch:.3g}\n"
        f"\tGCP_RMSE_x: {gcp_x:.5f}m\n"
        f"\tGCP_RMSE_y: {gcp_y:.5f}m\n"
        f"\tGCP_RMSE: {gcp_rmse:.5f}m\n"
    )

    master_ICP_gdf, (icp_x, icp_y, icp_rmse) = icp_accuracy(master_ICP_gdf)
    logger.success(
        "\nICP Averages:\n" f"\tRMSE_x: {np.mean(icp_x):.5f}m\n" f"\tRMSE_y: {np.mean(icp_y):.5f}m\n" f"\tRMSE: {np.mean(icp_rmse):.5f}m\n"
    )

    if os.getenv("SUPPRESS_OUTPUT") is None:  # Set by testing modules to prevent saving data to disk
        # Create Data visualizations and save analysis results
        analyze_icp_data(master_ICP_gdf)
        analyze_gcp_data(master_GCP_gdf)

```

```
        # Convert non-geometry columns containing Point objects to xy coordinate tuples
        gdf_Point_to_tuple(master_ICP_gdf)
        gdf_Point_to_tuple(master_GCP_gdf)

        # Save raw data to files
        master_ICP_gdf[raw_data_icp_cols].to_parquet(Path(os.environ["RESULTS_DIR"], "independent_check_points.parq"))
        master_GCP_gdf[raw_data_gcp_cols].to_parquet(Path(os.environ["RESULTS_DIR"], "matched_ground_control_points.parq"))

    return master_ICP_gdf, master_GCP_gdf


def gdf_Point_to_tuple(gdf: gpd.GeoDataFrame):
    point_cols = gdf.select_dtypes(include=["object"]).columns.tolist()
    for col_name in point_cols:
        gdf[col_name] = gdf[col_name].apply(lambda p: (p.x, p.y))


def load_rfdb_args(rfdb_dir):
    args_path = Path(rfdb_dir, "cli_args.yml")
    with args_path.open() as f:
        yml = yaml.load(f, Loader=yaml.Loader)
    return yml["georef"]["rfdb"]


def compare_args_sim_rfdb(sim_args, rfdb_args, ignore_keys=[]):
    """Compares simulation and rfdb arguments for differences in model parameters

    Sanity check to highlight differences and warn user.

    Returns a dictionary of values changed, where:
        'old_value' = rfdb args
        'new_value' = sim args
    """
    sim = dict(sim_args)
    db = dict(rfdb_args)
    return DeepDiff(db, sim, exclude_paths=ignore_keys).get("values_changed")


def get_reference_keypoints(db_path: Path, center: Point, radius, sample_crs):
    """Retrieve reference keypoint dataframe from saved database"""
    # Create a GeoSeries so the circle mask also has CRS information
    ri_kpt_circle = center.buffer(radius)
    ri_mask_gs = gpd.GeoSeries(ri_kpt_circle, crs=sample_crs)
    ri_df = load_db(db_path, mask=ri_mask_gs)

    return ri_df, ri_mask_gs, ri_df


def bbox_center_point(tile):
    """Returns a shapely Point object of center of the tile"""
    bounds = tile["bounds"]

    center_x = (bounds.right - bounds.left) / 2 + bounds.left
    center_y = (bounds.top - bounds.bottom) / 2 + bounds.bottom

    return Point(center_x, center_y)


def get_tile_diag(tile):
    """Retrieves diagonal dimension of the tile, in the units of the crs"""
    bounds = tile["bounds"]
    width = bounds.right - bounds.left
    height = bounds.top - bounds.bottom
    return np.sqrt(width**2 + height**2)


def load_icp_gcp_gdf(parq_path):
    parq_path = Path(parq_path).resolve()
    # set results directory to be consistent with data source
    if os.getenv("RESULTS_DIR") is None:
        logger.error('Environment Variable "RESULTS_DIR" not set. Must be specified before loading/processing data. Exiting')
        sys.exit()

    gdf = gpd.read_parquet(str(parq_path))
    point_cols = gdf.select_dtypes(include=["object"]).columns.tolist()
    for c in point_cols:
        gdf[c] = gdf[c].apply(lambda x: Point(x))

    return gdf
```