BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

IMAGE MOSAIC FROM A SEQUENCE OF DRONE IMAGES

OBRAZOVÁ MOZAIKA ZE SEKVENCE SNÍMKŮ Z DRONU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **DMYTRO KHODAREVSKYI**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. VÍTĚZSLAV BERAN, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

162667

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Khodarevskyi Dmytro**
Programme:	Information Technology
Title:	**Image mosaic from a sequence of drone images**
Category:	Computer vision
Academic year:	2024/25

Assignment:

1. Learn about methods for mosaicing and registering images.
2. Design a system that will create a terrain map on the fly by sequentially compositing images from a drone. Consider flat terrain.
3. Select appropriate datasets for developing and testing the proposed solution.
4. Implement the proposed system using existing libraries.
5. Demonstrate and evaluate the features of the solution.
6. Present the key features of the solution in the form of a poster and a short video.

Literature:

- Richard Szeliski. *Computer Vision: Algorithms and Applications, 2nd ed.* ISBN: 978-3-030-34371-2, 2022, doi: 10.1007/978-3-030-34372-9.
- Joseph Howse and Joe Minichino, *Learning OpenCV 4 Computer Vision with Python 3*, ISBN: 9781789531619, 2020.

Requirements for the semestral defence:
Points 1, 2, 3 and in part point 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Beran Vítězslav, doc. Ing., Ph.D.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	12.5.2025

Abstract

This thesis presents an image-based approach to drone mapping, using only aerial photographs without relying on GPS or depth sensors. A custom pipeline was developed using computer vision techniques such as keypoint detection, feature matching, and image stitching. The work explores common challenges like perspective distortion, offering practical solutions for improved alignment. The results show that surface maps can be generated from images alone, with potential for real-time applications.

Abstrakt

Tato práce představuje obrazový přístup k mapování pomocí dronů, který využívá pouze letecké snímky, bez použití GPS nebo hloubkových senzorů. Vlastní pipeline byla vyvinuta s využitím technik počítačového vidění, jako je detekce klíčových bodů, porovnávání prvků a spojování obrazů. sběru obrazů. Práce zkoumá běžné problémy, jako je perspektivní zkreslení, a nabízí praktické řešení pro lepší zarovnání. Výsledky ukazují, že mapy povrchu lze generovat pouze z obrázků, což má potenciál pro aplikace v reálném čase.

Keywords

Computer Vision, Image Stitching, Drone, Keypoints, Drone Mapping, Feature Matching, Mosaicing, Image Blending

Klíčová slova

Počítačové Vidění, Sešívání Obrazu, Dron, Klíčové Body, Mapování Dronem, Porovnávání Bodu, Mozaikování, Prolínání Obrazu

Reference

KHODAREVSKYI, Dmytro. *Image mosaic from a sequence of drone images.* Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Vítězslav Beran, Ph.D.

Rozšířený abstrakt

Tato práce se zabývá obrazovým přístupem k mapování pomocí dronů, který využívá pouze letecké snímky bez podpory GPS, IMU nebo hloubkových senzorů, jako je LiDAR. Cílem je vytvořit dvourozměrnou mapu povrchu pomocí série snímků pořízených dronem, přičemž hlavní výzvou je správné zarovnání obrazů a minimalizace perspektivního zkreslení v situaci, kdy vstupní data tvoří výhradně RGB obrázky.

Mapování prošlo v posledních letech zásadní proměnou díky rozvoji moderních technologií. Zatímco dříve byly mapy vytvářeny ručně během dlouhého a pečlivého terénního průzkumu, dnes je možné vytvářet podrobné a přesné mapy rychle a efektivně za pomoci dronů a technik počítačového vidění. Automatizované systémy dokáží v krátkém čase generovat mapy s vysokou přesností, což je klíčové zejména v časově kritických situacích, jako jsou přírodní katastrofy, kdy aktuální mapa může významně přispět k záchraně lidských životů.

V této práci je představen vlastní pipeline pro obrazové mapování, který byl vyvinut s využitím tradičních metod počítačového vidění. V rámci vývoje byly použity techniky jako detekce klíčových bodů, porovnávání prvků a spojování obrazů. Tyto postupy umožňují vytvořit konzistentní mapu z posloupnosti snímků pořízených dronem, přičemž velký důraz byl kladen na řešení perspektivního zkreslení a přesné spojení obrazů navzdory různým úhlům pohledu.

Práce je rozdělena do tří hlavních částí. První část se věnuje teoretickému pozadí problematiky obrazového mapování, včetně algoritmů a výzev spojených s použitím čistě obrazových dat. Druhá část popisuje navržený přístup a podrobně vysvětluje zvolené metody a rozhodnutí, která byla učiněna během vývoje pipeline. Třetí část se zaměřuje na technické detaily implementace a zahrnuje také testování a vyhodnocení dosažených výsledků.

Výsledky této práce ukazují, že je možné generovat povrchové mapy výhradně z obrazových dat, což představuje praktické řešení pro reálné aplikace. Navržený přístup poskytuje potenciál pro rychlé mapování v prostředích, kde není možné použít tradiční senzory, a otevírá možnosti pro využití v různých oblastech, jako je krizové řízení, zemědělství nebo environmentální monitoring.

Image mosaic from a sequence of drone images

Declaration

I declare that I have prepared this bachelor's thesis independently under the supervision of doc. Ing. Vítězslav Beran, Ph.D. I have listed all literary sources, publications and other sources from which I drew.

.....
Dmytro Khodarevskyi
May 13, 2025

Acknowledgements

I would like to thank Mr. doc. Ing. Vítězslav Beran, Ph.D. for his help and understanding during the creation of this thesis, and I would also like to thank all my friends and parents for believing in me and supporting me in difficult times.

Contents

Chapter 1

Introduction

The process of mapping has undergone a major transformation with the advancement of technology. Where maps were once drawn manually through long and careful fieldwork, today they can be created quickly and efficiently using drones and computer vision. When properly organized, an automated system can generate highly detailed and accurate maps faster than any human could produce manually. This becomes especially important in time-sensitive situations, such as during natural disasters, when having an up-to-date map can help save lives.

This thesis focuses on the use of image-only drone mapping, where the goal is to recreate a two-dimensional surface map using only aerial photographs captured by a drone, without relying on additional metadata such as GPS, IMU, or LiDAR. The main challenge in this task is handling image alignment and perspective distortion when the input data is limited to RGB images.

To solve this task, I explored the theoretical foundations of image stitching, keypoint detection, and feature matching. Based on this research, I developed my own processing pipeline that uses traditional computer vision techniques to build a consistent map from a sequence of drone images.

The thesis is divided into three main parts. The first part explains the theoretical background, including the algorithms and problems related to image-based mapping and also the reason for such an idea. The second part presents my approach, where I describe the methods and decisions I made during development. Finally, the third part covers the technical details of the implementation, as well as tests and evaluation of the results.

Chapter 2

Theory

In this section I will describe what this algorithm can be useful for and what it is used for in different fields of activity. I will also describe the key information necessary to understand and solve the problem. For easier understanding, I will talk about the main parts of computer vision related to this topic and also talk about its mathematical part.

2.1 Orthomosaic mapping problematic

Drone mapping is an incredibly convenient method for relatively quickly obtaining information from a given area, especially if it works in real-time. All you need is a drone with a camera and a good algorithm for comparing images, and that's it.

Construction site map

Let us imagine a situation where you are a builder and you need to build a building on a fairly complex terrain, where some terra-forming may have been done recently. In a good team, the work can go quite quickly, so everything can change very quickly, and in order to be able to track the process easily, people can use maps made by a drone.

This can also help calculate the materials used or the materials that will need to be spent in a particular place of the building, although for this you need to be sure of the accuracy of the map, so the most accurate algorithm is needed in order to avoid delays, unnecessary losses, and unstable areas.

People are still people, so incidents or misunderstandings can occur on the construction site, which will also have to be quickly resolved to avoid serious costs of time and money. Here, among other things, it will be easier than ever to assess the situation with the help of a map that is quickly thrown together by a drone.

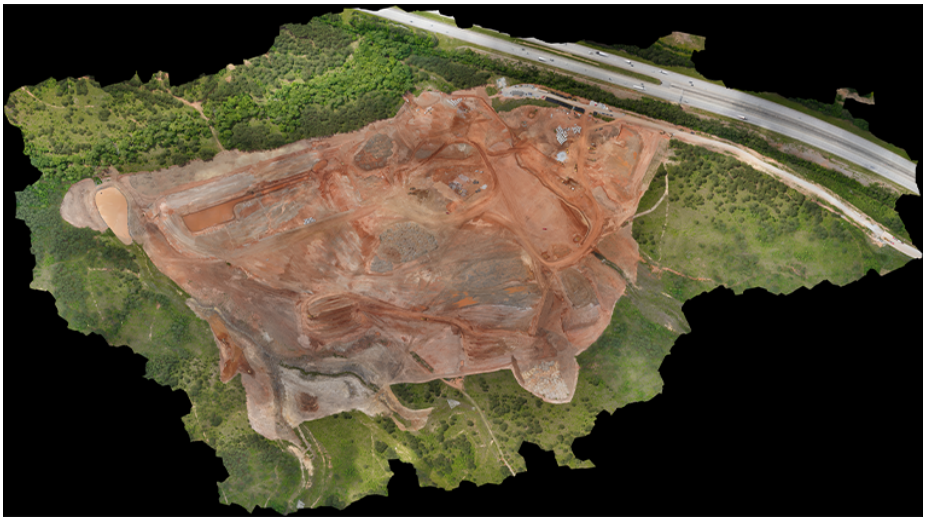


Figure 2.1: Bridgeport Moreland Construction Orthomosaic [19]

The figure 2.1 shows an orthomosaic for a construction site.

Orthomapping for Vegetation Areas

Let us move to a slightly different area – farming. Most fields where a variety of crops grow are very large plots of land. To analyze the condition of such a large area, you will have to spend a lot of people and time when you already need to fertilize, water, and weed the plants. What if it were possible to automate all this? It would be enough to collect information with a drone and combine it all into a common, accurate image, and it would be possible to analyze, including with the help of algorithms, all the necessary areas. Changes in the color of plants would indicate diseases or lack of moisture if the drone can collect additional information for the map, such as, for example, usually collected by multispectral NDVI sensors that can distinguish an additional spectrum of colors, thereby significantly increasing the efficiency of analyzing the health of the plant.

Such simplification, if it is also subsequently automated, will significantly facilitate the process of caring for plantations, which will reduce the cost of effort, time, and, of course, finances. In the figure 2.2 you can see an example of orthomosaic and metadata for plantations.

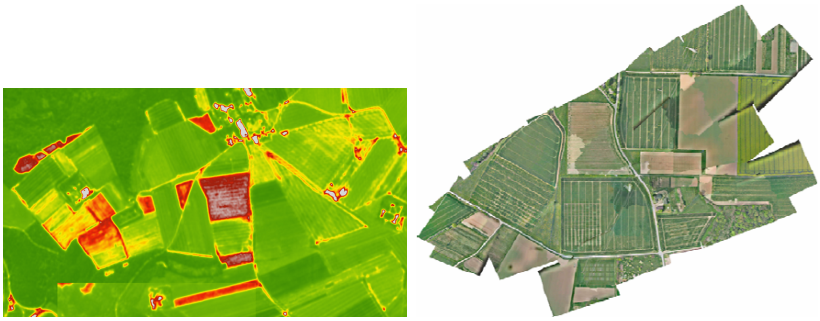


Figure 2.2: Example of NDVI spectral map [11] and agricultural orthomap [3]

Mapping the cataclysms

Gradually, we got to the reason why sometimes not only the accuracy of the map but also the speed of its construction can be important. In such situations, the surface changes completely, so only relevant and accurate information from a drone can help.

Hypothetically, there is a huge fire. In such situations, it is necessary to act quickly and decisively; every minute counts. It is necessary to assess the scale of the fire in order to calculate how much material needs to be delivered to eliminate the threat. In particular, by knowing the information from the map (such as the current geolocation of the fire's edge), you can calculate the time it can take to reach a particular place.

It also happens that the fire may have already occurred and will end; in such situations, speed is not so important, but you still need to act. If it has affected the infrastructure, then a map will be very helpful to assess the damage and plan a solution to the consequences. All this can apply to absolutely any cataclysm – a flood, a fire, an earthquake, or a volcanic eruption.

Use cases conclusion

The most common applications of this technology have been described above, but there may be more applications. Naturally, this method has its own shortcomings, which we will discuss in a little more detail later. From all of the above, we can draw the most important features in constructing maps that play a critical role in real applications. **Accuracy** is probably the most important factor in the map, as it is one of the priorities in all the above applications. Consequently, it will be the most important goal of this work. **Additional photo information (metadata)** can also be crucial very often; a simple picture may not be enough, so many similar algorithms make maps of various types, from depth maps to high-spectrum maps. Although this topic is very useful, in this project, I aimed to work with regular photographs, so depth maps or high-spectrum maps will not be discussed here. Finally, **algorithm speed** is another important aspect. Sometimes you need to get information quickly, even if you have to sacrifice accuracy. Although such situations are much rarer, they also have a right to exist. Therefore, such algorithms may consider the option of creating a map in real time, perhaps even in the drone itself.

2.2 Orthorectification

One of the key problems that can arise when shooting objects is perspective. The fact is that the map itself is a two-dimensional representation of the plane, but when we take pictures, although they are two-dimensional, they capture the world in its 3-dimensional form. This feature can bring certain artifacts to our results.
So, for example, in the first picture of the figure 2.3 we can see the roof of a building at one angle, as well as its wall almost completely, but after a few pictures its roof will be visible to us from a different angle, that wall will not be visible at all, but we will begin to see the wall behind the house.

The higher the object, the greater the difference in perspective relative to the ground, and the stronger this effect will be accordingly.

The best way to eliminate the problem is to eliminate the perspective, that is, to transform the original photo into its orthogonal copy. This means we will consider only those

Figure 2.3: Same building from different perspective [19]

pixels that are perpendicular to the camera. At the same time, those that have deviated significantly and are at an angle will distort the image. Orthorectification can be performed through different approaches: hardware-assisted methods, purely algorithmic solutions, or a combination of both.

In the figure 2.4 on the left, you can see a representation of the view from a regular drone photo. It displays all the light rays that hit the matrix, even those that hit at an angle, which means that we can see objects that are not directly under the camera, which can interfere with further mixing of photos, since we will accept different variations of objects in different shots.

In turn, on the right, we select for ourselves only those rays that are perpendicular to the lens, which makes the image consistent – both buildings have the same final image, as if we were looking at one building from above and the second building from above. So, if each photo contains such objects, the algorithm will have no trouble stitching them together. Ortho-cartography methods can perform such an operation based on information about the height of each pixel. In this work, however, I will try to get closer to the result of orthography solely by stitching images without additional information.

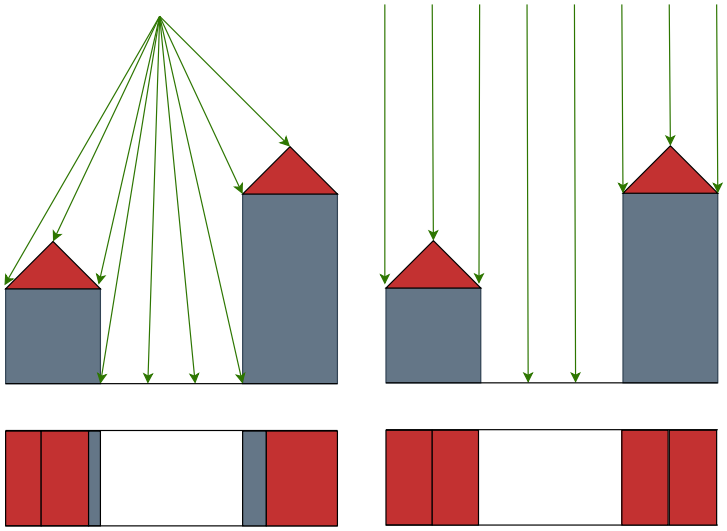


Figure 2.4: Perspective image view and Orthorectified view

2.3 Image stitching

The process of photo stitching means using new available data from additional photos to complement the previous ones. In this way, we can get a new, larger photo. This is how, for example, creating a digital panorama works on modern phones. Mapping with a drone is essentially a more rigorous version of image stitching.

The merging of photographs occurs through finding similar keypoints and descriptors, after which a transformation is calculated to transfer them. This means that the pictures should not contain little information from which similarities cannot be identified and should not contain completely different information relative to the pictures we are stitching together, since it will simply be unclear where this picture belongs. You can think about this from the point of view of a puzzle; a puzzle contains sections that fit onto another puzzle for fastening and form a whole picture. Without similarities in the picture and shape, we will not be able to add the puzzle to the overall picture. If the puzzle has a different picture on itself, it will be difficult for us to determine where it is; we most likely will not be able to place it accurately without relying on the previous puzzles.

The figure 2.5 shows an example set of canyon images for stitching, the first and second images clearly show that the stone arch is present in both, but offset, adding new information. The third and fourth images move step by step further, gradually adding to the image, but not jumping too far so as to not get too distinctive an image that does not relate to the previous ones. The result can be seen in the following figure 2.6, completely traceless stitching. I will tell you more about the process in the following parts.

Figure 2.5: Example of photo set for image stitching [14]

Figure 2.6: Example of panorama created by image stitching [14]

2.4 Image feature detection

In this section I will try to describe classical and key methods of image feature detection using computer vision algorithms.

Corner Detection

Corner detection is a fundamental technique in computer vision used to identify points in an image where the intensity changes sharply in multiple directions. These points, often referred to as „corners“ or „interest points,“ are useful for object recognition, image matching, and motion tracking. The detection typically involves analyzing the gradients of image intensity using convolution matrices (kernels), such as in the Harris [4] or Shi-Tomasi [13] methods. By identifying regions with significant variation in both the horizontal and vertical directions, corner detection provides a way to locate distinctive and stable features within an image.

Scale-Invariant feature detection

One of the most popular algorithms for finding keypoints or distinctive features in images is SIFT, also known as *Scale Invariant Feature Transform* [7]. For the information provided by distinctive features to be effective in image processing and computer vision tasks, the features must exhibit certain invariance properties. Specifically, they need to be **scale invariant**, meaning that a distinctive feature, such as one found in a window, should not be confused with a similar feature on a window of a different size. Additionally, they must be **rotation invariant**, ensuring that the feature is reliably recognized regardless of the object's orientation; in other words, it should not be mistaken for a rotated version of a similar object.

The process of working with distinctive features generally consists of two main stages: first, detecting keypoints in the image, and second, computing descriptors for these keypoints. These descriptors capture the local image structure around each keypoint in a way that supports robust matching between images despite changes in scale, rotation, or other variations. This algorithm finds characteristic features in a black-and-white photo and therefore does not distinguish colors.

SIFT output

At the output of the SIFT algorithm, we get a set of keypoints and descriptors that identify certain features of the photograph, and are also scale invariant, rotation invariant, and in some cases illumination invariant. Here is how the output of the SIFT algorithm is displayed using built-in methods 2.7.

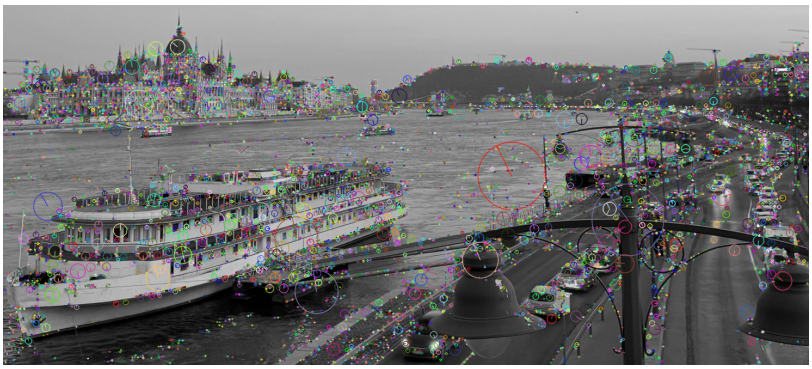


Figure 2.7: Example of keypoints from a photo of Danube river in Budapest

Although the SIFT was chosen for this project, there are other, more modern algorithms for detecting photographic features. There is also a good, quite new paper that compares some of the algorithms listed below on the same problem and explains how they work in more detail [12].

Other feature detection algorithms offer various trade-offs compared to SIFT. SURF (Speeded-Up Robust Features) and ORB (Oriented FAST and Rotated BRIEF), for example, are optimized for speed and are frequently used in real-time applications like SLAM (Simultaneous Localization and Mapping) and robotics. While they are faster than SIFT, they may sacrifice some robustness and accuracy. KAZE and AKAZE improve upon SIFT's localization accuracy by using nonlinear scale spaces, making them suitable for high-precision tasks such as image registration. In contrast, deep learning-based detectors, although more computationally demanding than SIFT, offer enhanced robustness and adaptability in complex environments, which is particularly beneficial for applications like autonomous driving and augmented reality.

2.5 Images feature matching

To understand how to transfer one photo to another, we must understand how the location of the keypoints in the photo has changed, thus we will be able to recreate a replica of the transformation. That is, for example, we photographed a window at one angle, then at another angle, and we will also be able to accurately identify this window in two photos using the above-mentioned keypoints, thus we can find out how the location of these keypoints has changed by coordinates, and how they have changed in orientation, for example. Here is an example of keypoint mappings in the figure 2.8.

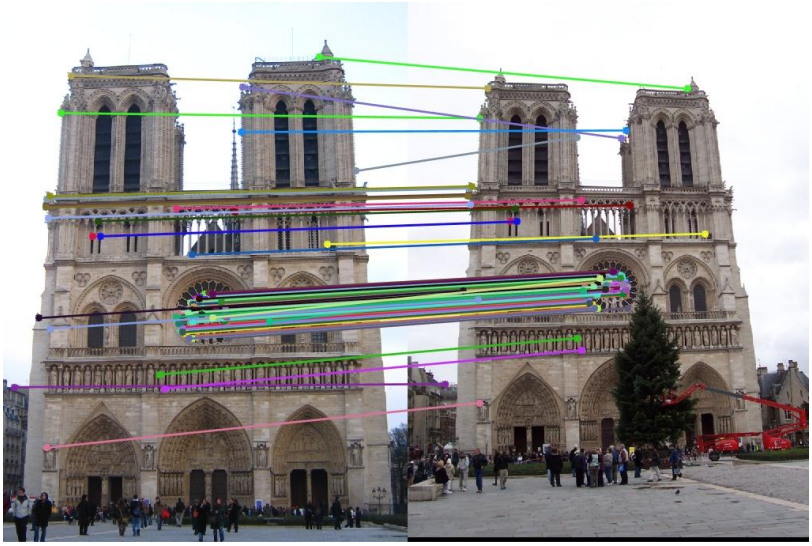


Figure 2.8: Feature matching [8]

Brute force matcher

The Brute-Force (BF) Matcher [10] is a straightforward method for feature matching that compares each descriptor from one image to every descriptor in another image. It computes a distance metric, such as Euclidean distance for SIFT or Hamming distance for binary descriptors like ORB, to identify the best matches. Although computationally intensive, the BF Matcher is highly reliable and works well for small datasets. It is often used in combination with cross-checking or the ratio test (e.g., Lowe's ratio test) to filter out poor matches. The simplicity and adaptability of the BF Matcher make it a popular choice for prototyping and applications where computational resources are not a constraint.

FLANN-Based matcher

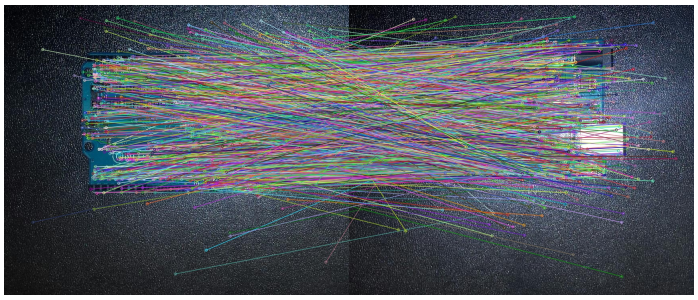
The Fast Library for Approximate Nearest Neighbors (FLANN)-Based Matcher [9] is designed to address the computational inefficiency of the BF Matcher for large datasets. FLANN employs optimized algorithms, such as KD-trees and hierarchical clustering, to approximate the nearest neighbors efficiently. This method significantly reduces the time required for feature matching while maintaining high accuracy. The FLANN-Based Matcher is particularly well-suited for applications involving large-scale datasets or real-time systems, such as 3D mapping and augmented reality.

Matchers conclusion

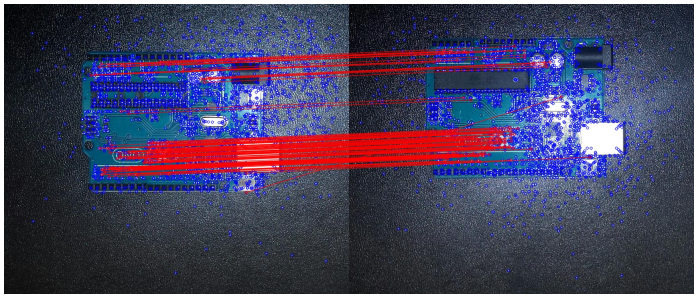
Both of the above algorithms are good in certain situations. Despite the fact that the BF matcher works much longer, and is simpler (which is what the name suggests), thus it is able to find almost all similarities available in the images, thereby improving potential stitching in the future, but here you need to be careful since not all matches are correct, so it is usually used in tandem with filtering through the D. Lowe's coefficient [18].

FLANN, in turn, finds **approximate** neighbors, which means not absolutely exact similarities, but in this way it could be faster to find similarities. However, this has the right to exist only in large datasets, if we are talking about speed, since in them the speed of the BF matcher may be low, and there is also the required amount of data after FLANN's work. Otherwise, when the dataset is small, FLANN does not find so many similarities and works not much faster than the BF matcher, thereby losing to it. When it comes to accuracy, BF is the best choice. You can see what the results of comparing points by both algorithms look like in the figure 2.9.

> FLANN provides similar results to the BFMatcher algorithm using the Ratio Test. While the matches are excellent, there are not enough matches to be able to then find the differences between the images.



(a) Brute-force matcher



(b) FLANN-based matcher

Figure 2.9: Feature matches using brute-force matcher (a) and FLANN-based matcher (b) [20].

Filtering features

When comparing descriptors at source and target keypoints, David Lowe derived a ratio that allows one to remove some of the unsuitable pairs with minimal loss of suitable ones. Let's call it the *D. Low'e ratio* [18].

To be more precise, the test finds similarities by the two nearest neighbors to a certain descriptor by its vector. The condition for accepting a match is defined as:

$$d_1 < d_2 \times r \tag{2.1}$$

Where:

- d_1 – Euclidean distance from the requested point to the closest of the matched points
- d_2 – Euclidean distance from the requested point to the second closest of the matched points
- r – D. Lowe's ratio constant (commonly 0.8)

The ratio test ensures that no two candidates in the dataset are similarly close to the query point. Only if the nearest neighbor is significantly closer than the second nearest can the match be considered reliable.

I prepared the explanation of it on figure 2.10. There is a visual example to show the work of such filtering on two-dimensional vectors (descriptors are 128-dimensional vectors, but the logic is the same except for finding the Euclidean distance). The points relative to which the comparison is made are marked in orange, while the blue ones are the points that denote neighbors relative to the orange ones.

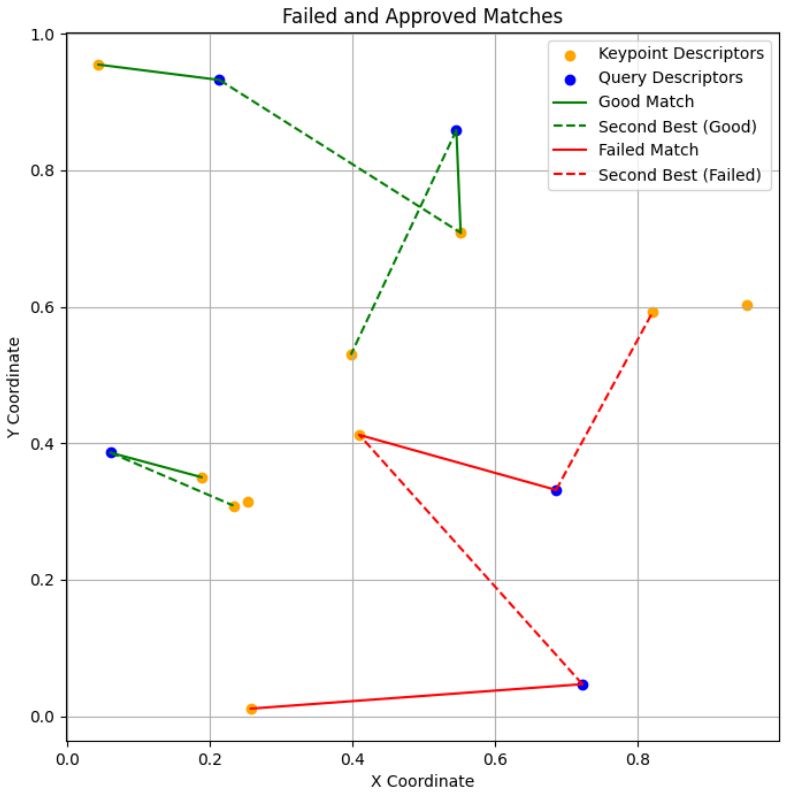


Figure 2.10: Visual example of Lowe's ratio test filtering.

Figure shows the points that were found as similarities for the orange ones. Those points that are clearly closer to their competitors are considered reliable similarities, while those that have a neighbor very similar in distance are considered unreliable. Increasing $\underline{r}$ we reduce the filtering, thereby obtaining more similarities – but less reliable ones. When decreasing, the accuracy increases, but by decreasing the number of final similarities, it may subsequently be more difficult for the algorithm to find further homography if there are not enough similarities.

2.6 Images geometrical transformation

In order to find the geometric transformation between two images representing a flat object, we must find a homogeneous matrix transformation, for the calculation of which we use the *RANSAC algorithm* [2] described below.

The sequence of actions is listed below:

1. At the input, we have a set of pairs of points, denoted as (p_i, p_i'), where p_i represents the source coordinates and p_i' represents the target coordinates. Note that these points may include a false transformation.

2. A *homogeneous matrix* is a 3×3 matrix, represented as:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}, \tag{2.2}$$

$$p_i' \sim H \cdot p_i \tag{2.3}$$

   where $\sim$ denotes equality with minimum error.

3. To calculate the homogeneous matrix, we require at least four pairs of points. Using the *RANSAC algorithm* [2]:

   (a) Randomly select four pairs of points.

   (b) Construct the homogeneous matrix H using these points.

   Matrix is constructed by the Direct Linear Transform method (DLT). I will not specify the solution here, but it is described in a book [5] and an example of it is in a similar paper [6].

4. Transform the source points p_i using the matrix H and compute the distances (errors) between the transformed points and the target points p_i'. Formally:

$$\text{Error} = \|H \cdot p_i - p_i'\|. \tag{2.4}$$

5. Remove points with error greater than a specified threshold (outliers). Points with errors below the threshold (inliers) are retained.

6. Repeat steps (1)–(5) for a specified number of iterations. In each iteration, the model with the highest number of inliers (or best error score) is retained as the best hypothesis. After all iterations, the final model is recomputed using all inliers from the best hypothesis.

2.7 Image blending

When stitching photos, we can consider a variety of ways to overlay them. Since we are looking at photos that overlap each other, the place where the photos meet is repeated twice. So, which option to use to join the photos is decided by mixing at photos that

overlap each other; the place where the photos meet is repeated twice, so which option to use to join the photos is decided by mixing the photos.

The final photos, even after serious processing by algorithms, may contain slight color distortions or objects that could not be perfectly combined. There are also situations when the object is not very important to us, so we can neglect accuracy and take into account the visual representation.

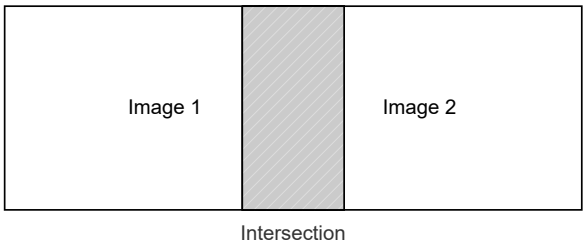


Figure 2.11: Image blending region

The figure 2.11 shows an example of the intersection of images when stitching; at the intersection, you have to select and distribute the weight of pixels between the images to produce a blend.

From Richard Szeliski's work [15], several methods of blending images were considered for this paper. These include the usual image overlap, average (50/50 blending) and feathered average (gradient blending), all of which you can see on figure 2.12.

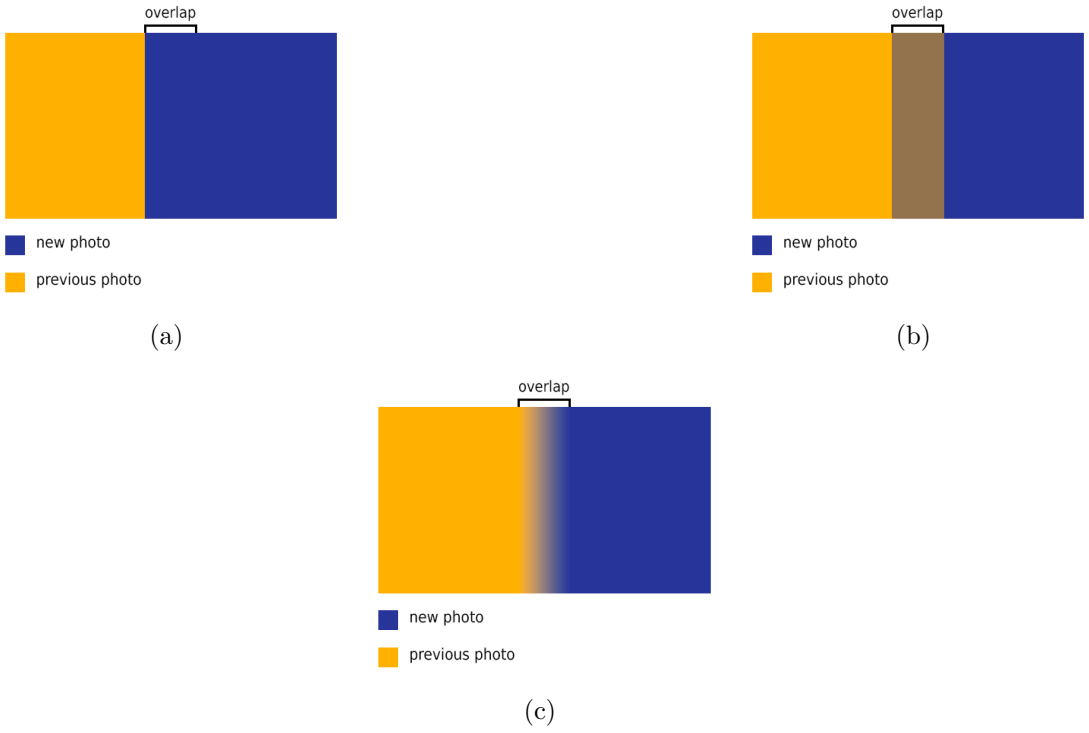


Figure 2.12: Examples of overlapping blending methods: overlap (a), average (b), and feathered average (c).

Chapter 3

Solution design

This chapter outlines the complete pipeline developed for this task, starting from raw image input and ending with a fully stitched image mosaic. In it, I will discuss the problems that need to be addressed and then explain how I approach them in my implementation without going into technical details yet, focusing purely on the theoretical and conceptual aspects.

3.1 Goal and Problems to be solved

The goal of this algorithm is to create the most consistent map possible based on drone images. Note that the images will not undergo orthorectification or color alignment.

In order for us to be able to effectively compare images one by one, we need to somehow process them one by one and collect the necessary information from each image, based on which we can transform new ones. From each image we can obtain a certain set of keypoints and descriptors; they need to be saved in some structure, from which we can easily obtain the necessary information about existing points. We will also need to correctly match the points to determine a homogeneous matrix. Based on homographies, we can transform each image and then superimpose them on each other in a certain way using one of the blending methods. To fully display the image, we also need to perform a general transformation from negative coordinates, which I will discuss in more detail in the following parts.

Here is an abbreviated list of things that need to be addressed:

- Capturing and preparing image data for analysis.
- Initializing the first frame and reference parameters.
- Defining a predictive area based on previous image transformations to guide the search for the next frame.
- Matching keypoints between images to estimate spatial relationships.
- Calculating homography matrices to define the transformation between images.
- Gradually building the full mosaic by warping and combining frames.
- Determining the final canvas size and correcting offsets introduced by cumulative transformations.
- Blending overlapping images using several techniques to reduce seams.

3.2 Algorithm for collecting information (matrices)

In this section I will describe the basic idea of the implementation of the algorithm for generating the necessary information for mapping, separating some details for a more detailed description in the following sections. I will describe the solutions to the above questions, namely how they work and why they are needed in a common environment.

Initialization, start of execution

The first step for the algorithm to work will of course be the initialization of the structures. As I described above, in order for us to be able to gradually add images, we need to save information in the structure. This information will be keypoints and their descriptors for each image, because with their help we will subsequently determine the further position of the newly arriving image. Therefore, the first thing we must do is initialize a certain map of keypoints. For now, we will leave this idea in an abstract concept; I will tell you more about it later. Now it is enough for us to know that this will be our storage and food for compiling transformation matrices. The next step will be the initial analysis of the image. The very first image has no predecessors, which means it has nowhere to transform, so we have the right to leave it in the position in which it will come to our algorithm, or it will be transformed to the unit matrix. In the future, all subsequent calculations and transformations will occur relative to the first image; in other words, the first image is the global coordinate system of images. Like any other image (except for transformation), the first image will go through a processing stage, where we will use SIFT to obtain a set of its keypoints and descriptors, and we will be able to save them in our map as the first reference set.

Matrix calculation cycle

Each subsequent image will also be processed using SIFT, but its transformation matrix will be calculated based on the similarities between the images. Let me give you an example: we received the second image. After initialization, we have access to information from the first image – the reference part, the original coordinates. The second image should overlap with the first in reality, so we can find similarities and calculate the transformation. As I already said, we get keypoints and descriptors using SIFT and get new data that we can compare with the previous ones, and the similarities will determine our matrix. Knowing which points match, we can find out how they were transformed from the coordinate system of the new image to the coordinate system of the first. I will tell you more about working with keypoints later. So, we were able to calculate the transformation of the image. We will remember this information for the future, and we can continue calculating matrices in a similar way, image by image. I used all these transformation matrices when creating the map.

Diagram

Here is a simplified visual description of the transformation matrix collection part in the form of a diagram, that you can see in figure 3.1. We cyclically go through all the images after the first one and gradually add information to the keypoint map and find transformation matrices for the images, and so on until we run out of images or the proposed amount of processing.

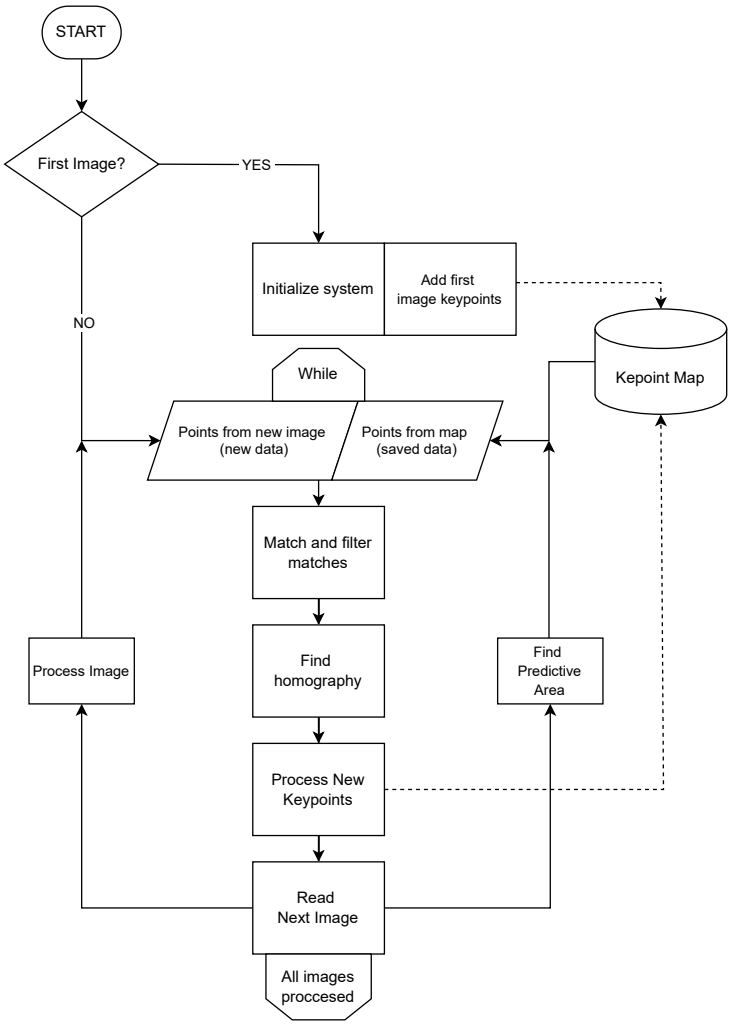


Figure 3.1: Capturing data pipeline diagram

3.3 Keypoints map

In this part I want to tell you more about the work of the keypoint map, as it plays an important role in calculating matrices. Initially, I thought to call this structure a keypoint map, but such a term is more suitable for three-dimensional structures containing points; our version is two-dimensional, as we work with two-dimensional images, and accordingly, build a two-dimensional map.

This map serves as a reference point for each new image, a perception factor of our algorithm. It is with the information from there that we collect similarities between the new image. Each keypoint is an object in two-dimensional space, which directly relates to the distinctive characteristic of the image, containing a pair of descriptors. When adding information from a new image, the keypoints are transformed according to a homogeneous matrix. This is necessary so that they retain their position according to the constructed map, because later it will be very useful for us to take points from a certain logical area.

Predictive area

This area is called the predictive area, from the word predict, that is, we kind of predict the area from which there may be potential similarities between the images. Since we are considering a gradual construction of the map, that is, the images follow one another, without sharp jumps, they are logically sorted, we can start from the fact that the predictive zone of searching for matches for a new image will be the area of the previous one. In order to avoid accidentally filtering out positive candidates from the keypoint map, we will increase our area by a certain constant. This is necessary in a situation where a new image, in addition to overlapping the last one, also overlaps one of the previously processed images, which is a fairly classic situation during a multi-line drone flight. When passing the first strip, the new image will not climb onto the previous ones, but when flying back, the images can climb onto the first strip, which means that we will start taking information not only from the previous image.

Based on the boundaries of the previous image, we can find out the area to search for points from the keypoint map. In order to get a predictive area of 4 points indicating we can use the following formula:

$$\mathbf{P}'_i = \mathbf{C} + s \cdot (\mathbf{P}_i - \mathbf{C}) \quad \text{for } i = 1, 2, 3, 4 \tag{3.1}$$

Where:

- $\mathbf{P}_i \in \mathbb{R}^2$ are the original corner points of the bounding box.
- $\mathbf{P}'_i \in \mathbb{R}^2$ are the corresponding points in the predictive (scaled) search area.
- $\mathbf{C} = \frac{1}{4} \sum_{i=1}^{4} \mathbf{P}_i$ is the centroid (geometric center) of the original bounding box.
- $s \in \mathbb{R}^+$ is the scaling factor that controls how much the area is expanded.

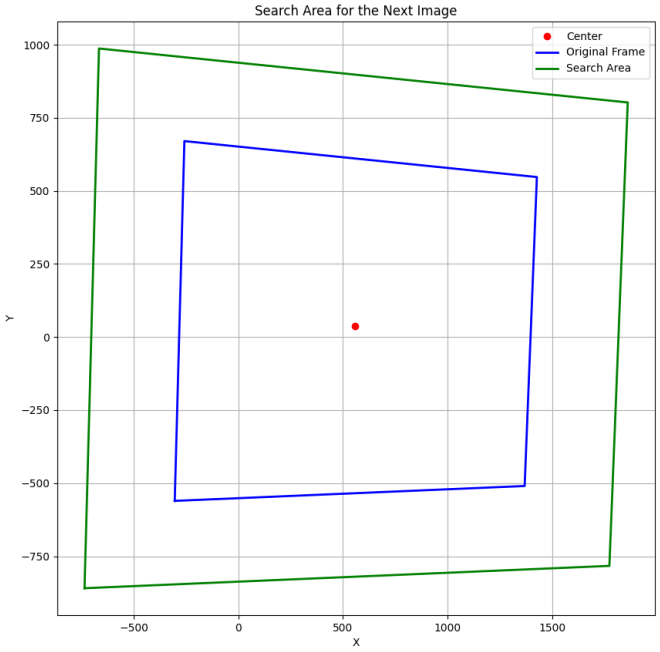


Figure 3.2: Predictive area example (scale ×1.5)

The figure 3.2 shows an example of a predictive region. The blue frame denotes the original boundaries of the previously processed image, the green frame denotes the calculated boundaries of the predictive region, increased by a constant. The red dot denotes the subtracted centroid of the image, relative to which the region was formed. The s in this image was 1.5.

Keypoints reliabilities

A small optimization based on the method proposed in [6] is also incorporated into this work. During the processing of keypoints, we can not only save them to the map, but also evaluate their performance, how useful they are when the algorithm works. If the keypoint turns out to be useless – that is, it was not used when constructing the matrix, that is, it did not find a pair, then in the future we can think about whether it is worth spending resources on it at all? Otherwise, if the point constantly finds a pair, and actively participates in the compilation of matrices, then it repeatedly proves its usefulness and claims to be a good candidate. This value, showing how useful the point is, we will call the parameter *Reliability*, which will be contained in each keypoint in the map. In my situation, when the keypoint reliability value drops below a certain value, which in my work was 0.3, when requesting keypoints in an area from the keypoint map, such a point will be ignored. Ideally, such a point could be deleted altogether, but for the sake of clarity of the process and further visualization, I decided to leave such points. When processing a new photo, both for new keypoints and those already in the keypoint map, there are 3 possible scenarios for reliabilities:

- **Increase reliability**

  If a keypoint that was already in the dataset was found as a pair (i.e. a new keypoint was transformed into its place or nearby), then its reliability is multiplied by u.

- **Decrease reliability**

  If a point lies in the predictive region and in the region of the new transformed image, but was not found as a pair, then the point was not used and thus doubts arise about its usefulness. Such points receive a reliability multiplication by a constant d.

- **Add as new**

  If both previous conditions do not fit the description of the keypoint, then we can simply add it to the keypoint map with the initial reliability value i.

- u – constant for increasing (1.5)
- d – constant for decreasing (0.7)
- i – initial reliability value (0.5)

Gathered information output

The output of these procedures is a set of homogeneous transformations for each image and a keypoint map in which the information is structured and filtered for use. In addition, I have attached a figure 3.3 below that visualizes an example of a keypoint map. The white rectangle indicates the predictive area, the last processed image is in red, and the green one is the boundaries of the previously transformed image, from which the predictive area was actually calculated. Also, for clarity, I normalized the reliability values of each keypoint to color, where red-violet color indicates reliable points, and blue-light blue indicates unreliable points; the lighter the color, the less reliable the point is considered, and, most likely, this point is considered no longer used.

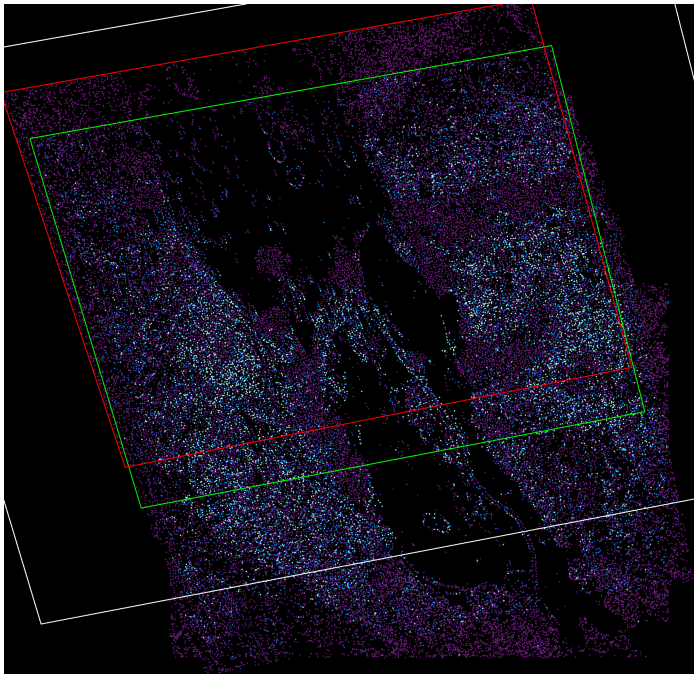


Figure 3.3: Keypoint map with colored reliabilities example

3.4 Algorithm for forming a map based on matrices

Most of the work is already behind us, but we still have to answer some questions that relate to the creation of the map. First of all, in order to calculate the frame of the final image, we can perform one operation. During the final transformation of the images, the corners of the images will determine the boundaries of the final image, so in order to calculate the sizes, we can transform the corners of each image to its homogeneous matrix, and calculate the minimum and maximum values. In this way, we will know what size the canvas should be to place the images without cropping anything. Specifically:

$$w = \max(x_i) - \min(x_i), \tag{3.2}$$

$$h = \max(y_i) - \min(y_i), \tag{3.3}$$

where x_i and y_i are the coordinates of the corners of all transformed images, w represents the final width, and h represents the final height.

Then there is another small problem. Images have the will to transform in all directions of two-dimensional space, but without the consequences of cropping the image, the way the pictures are presented from the technical side allows us to transform them to the right and down. In other directions, without intervention, relative to the first picture, all new ones would be sent to negative coordinates where the image is simply not displayed. In the figure 3.4 you can see how the first picture would look in the coordinate plane.

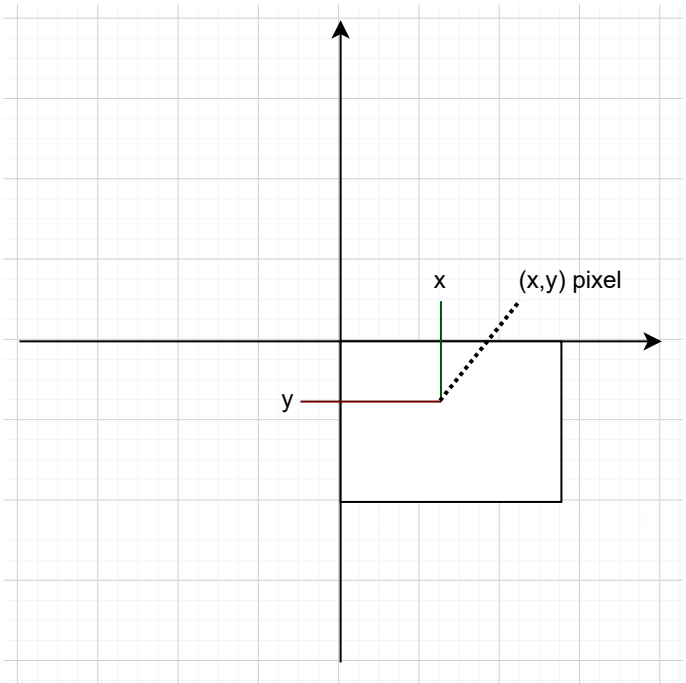


Figure 3.4: First image position on global coordinates

In other words, as I said, our first image is a global coordinate reference, and its very first pixel is at the location 0 by x and 0 by y. If we were to add an image, for example, to the left above the pixel, it would already be in the undisplayed negative coordinates. The solution to this problem is to find the farthest section (pixel) of all the corners of the images and multiply each homogeneous matrix by another translation matrix in the opposite direction.

To make it a little more clear, I have attached an image that describes the solution to this problem. Figure 3.5 shows the negative transformation without translation, figure 3.6 shows the final translation applied to each image, based on the distance x and y (Green indicates the distance y, red x. Only the negative part of the coordinates is used to calculate the translation.).

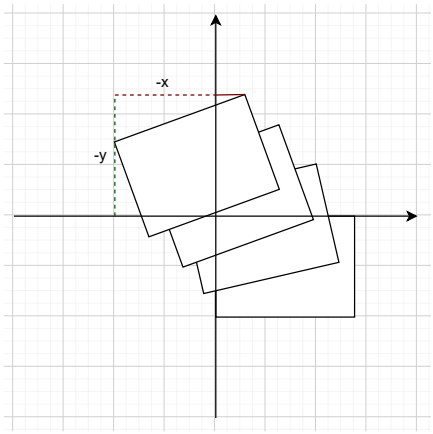


Figure 3.5: Example of negative transformation issue

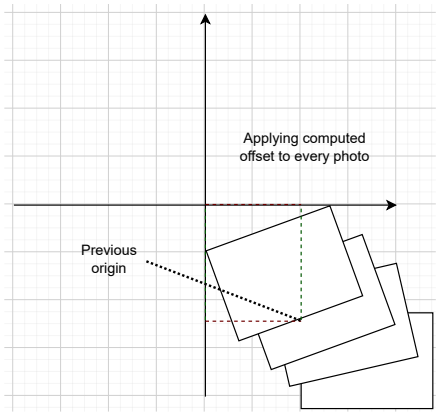


Figure 3.6: Transforming each consecutive image with offset

We have prepared everything necessary for compiling the map, the last step will be to choose how to superimpose the images on top of each other. From the options, we can choose one of the blending methods, and at the intersections of the image with the already connected previous ones, apply the calculated masks for blending. Each new image after the final homogeneous transformation and translation from negative coordinates will be added to the final calculated canvas, mixing with the previous images. Upon completion, we will receive the long-awaited result, a connected map based on images.

The figure 3.7 also shows a visual representation of the final stage of the algorithm, which gradually forms the final map based on the collected information. After some of the sizing and translation calculations, we can begin the cycle of adding each image to the map using blending methods.

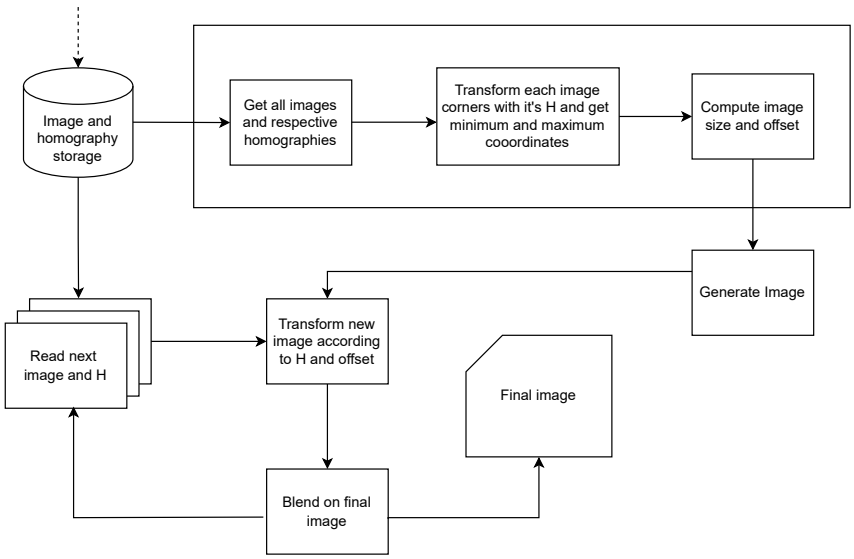


Figure 3.7: Forming resulting image

Chapter 4

Realization

In this section I will describe the implementation of the general algorithm from the program code side. It will describe the implementation features in the form of structures and functions or methods that were written in this project.

4.1 Software solution

In this project, I used one of the most common libraries for computer vision, OpenCV, to process visual information, which used SIFT, RANSAC, KNN, etc. methods. I chose Python as a programming language, as it seemed to me the most flexible for such tasks. Since there was a need to work with complex data types, the NumPy library was also used, and the Matplotlib library was used to visualize the process of creating a map, and some explanatory things for this paper.

The algorithm consists of the **main** module, which does the bulk of the information processing and uses auxiliary classes for the second phase of the algorithm, as well as for storing information in instance variables or saving processed information in files. Using information saved in files showed an acceleration of the algorithm by 10% to 23%, namely, homographies, descriptors, keypoints, and filtered matches are saved. In fact, when using optimization, only the second part of the algorithm works to the full extent. On such map creation, however, of course, old information is used; that is, when changing parameters for calculating new data, we better disable the use of saved data.

In the **main** itself, there is a sequential reading of photos, processing of command line arguments, and initialization of instances. Including the main course of the first phase of the algorithm, in which each photo is processed and whose information is added to the keypoint map for subsequent reference. Also, here I do the above-mentioned optimization of keypoints and update them directly in the storage.

All keypoints, descriptors, and additional information, which is usually a product of the SIFT algorithm, are stored using the **KeypointStorage** class. This class is able to save new keypoints and their data, as well as take existing ones in a certain area. It is important to note that if a new point appears within a certain threshold radius, it will be considered already existing and therefore, instead of adding a new point, the old one will be updated, namely its reliability according to the aforementioned optimization. To efficiently store keypoints in two-dimensional space, a K-D tree [17] was used as a data structure.

All information about the photos is saved through the **ImageStitchingData** class as the algorithm progresses. The class accumulates homographies and photos for further stitching. The class is also able to save the accumulated information in files for further reuse.

At the end of the **main**, the second phase comes into effect, where the **Stitcher** takes the accumulated information as an argument and begins mixing photos by homographies and by the selected mixing mode.

The diagram 4.1 shows the main part of the algorithm's operation methods, **main** processes the images and stores the necessary information in the class instances during the operation. When constructing a map, the necessary information is collected and processed to create the final image.

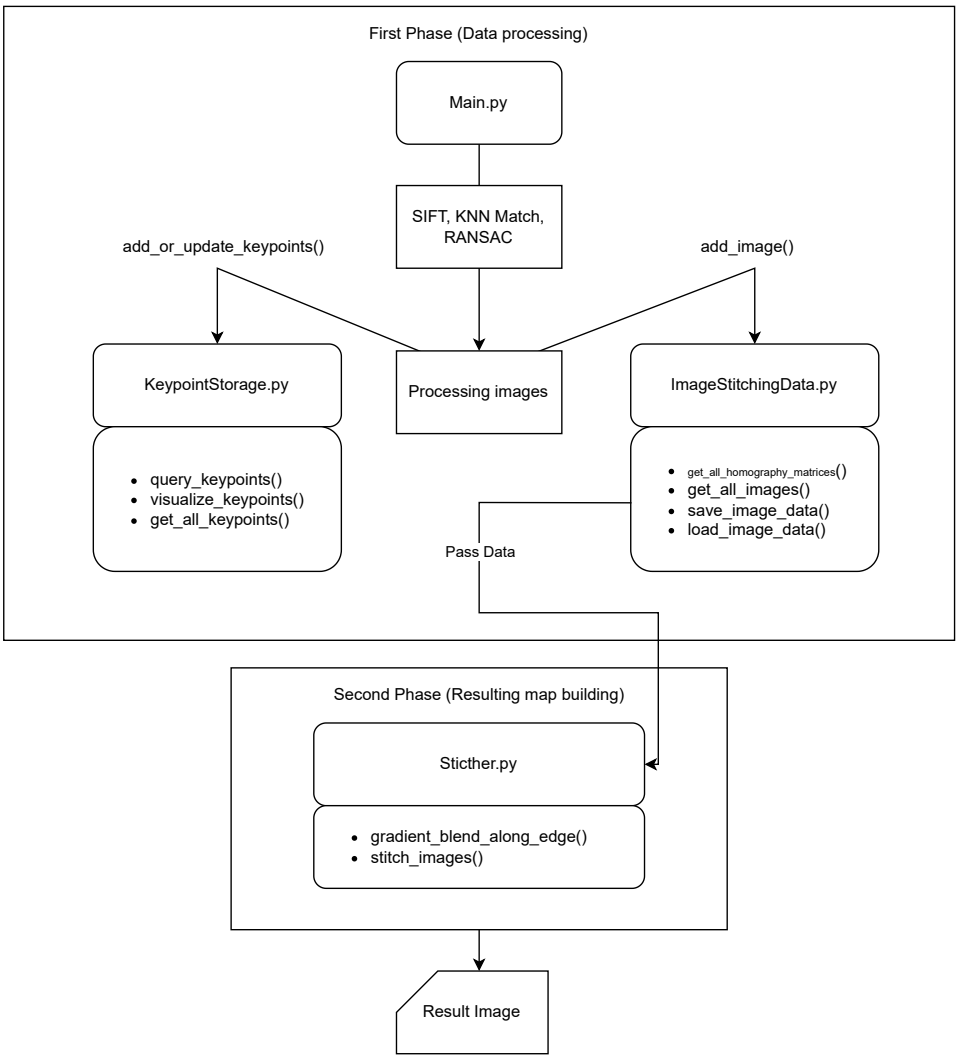


Figure 4.1: Program modules pipeline

4.2 Experiment overview

To evaluate the results, we will need to produce the result of my algorithm in several possible situations (different dataset options), as well as a reference on the basis of which we can compare the final result. For a more detailed assessment and analysis of similarities, the overlap blending mode was selected. The references that I managed to find will be considered ground truth since they are used for full-fledged calculations on maps in real life. To evaluate how similar the reference structure is to my result, the *SSIM* metric [16] was used.

This metric takes into account the geometric differences between objects and also carries them out in a manner similar to human perception. This metric is calculated using a formula like this, it combines luminance, contrast, and structural terms:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

- μ_x, μ_y – mean pixel intensity of images x and y
- σ_x^2, σ_y^2 – variance of images x and y
- σ_{xy} – covariance between x and y
- C_1, C_2 – small constants to stabilize the division (typically based on dynamic range)

Range: The SSIM value lies in the interval $[-1, 1]$, where:

- SSIM $= 1$: images are identical in structure
- SSIM $= 0$: no structural similarity
- SSIM < 0: structurally dissimilar (rare in practice)

Negative values are mathematically possible, especially when comparing highly uncorrelated or distorted images, but they are uncommon in practical image comparison. As such, SSIM values are typically interpreted within the interval $[0, 1]$.

It is important to note that SSIM is not a direct percentage of similarity. For example, an SSIM value of 0.6 does not imply that the images are *60% similar*. Instead, it suggests a moderate level of structural similarity, which should be interpreted relative to the context of the application or compared across multiple SSIM scores. This metric will allow us to objectively evaluate the work of the algorithm, regardless of any color deviations, but by looking more closely at the geometric component of the objects.

In order to achieve a fair metric assessment, the images must have the same objects at the same coordinates, which means that when superimposed, the images must match. Since the references contained very large, in their own way rotated orthomosaics, I had to perform several operations in order to match my images and the reference as much as possible. To do this, I initially reduced the resolution of the reference so that it matched my stitches, and rotated them in a roughly similar manner. Having cut off excess areas and left a reserve at the reference (so that there was more information from the reference than in my stitch), I used the same technique to transform the photo into the dimensions of my mosaic. To be more precise, I calculated keypoints, found similarities and calculated the homography for

the transformation, after which I transferred the reference to my result. In order to avoid areas where information from one version is present and information from the other is not, which would essentially distort the results, I made a conjunctive mask and applied it to both images. In the end, I got a reference based on which I could test my version fairly.

4.3 Datasets

I tried to use different datasets as sources of photos, as different terrains can help to notice different problems of the algorithm and also to reveal some shortcomings of the solution. Below I will list the datasets that I used and their features. On the table 4.1 you can see the original frames size, their downgraded resolution, total frames downloaded locally, and the number of frames used for my experiments.

Also, all of the following were reduced in resolution to speed up the work and development of the algorithm; in the original datasets, the quality of the photographs is significantly higher.

Datasets	**Original**	**Resized**	**Total frames**	**Used frames**
Golf Course	4000×3000 px	1440×1080 px	664	30
Highway	7952×5304 px	1619×1080 px	379	56
Quarry	7952×5304 px	1619×1080 px	85	50

Table 4.1: Datasets overview table

Colorado – Golf Course Dataset

This dataset examples on figure 4.2 are from DroneMapper [1] that captured a golf course in Colorado in 2016.

This dataset is very good for initial tests of the algorithm, as it contains quite a lot of distinctive features, and at the same time does not contain tall objects that can interfere with stitching photos due to perspective. Also, all objects on it are static, which means that during shooting the objects did not change their shape and position.

Figure 4.2: Golf Field [1]

Dominican Republic – Highway Dataset

This dataset examples on figure 4.3 are capturing the Dominican highway in 2022 by Wingtra [19].

In this situation we have a lot more buildings – which means more complex structures and distinctive features for the keypoints. However, there are moving cars that can interfere with the moving keypoints in the keypoint map, which in this situation was not a problem, since there are many fewer points from cars than from static objects in the form of buildings. Another problem was the perspective of the buildings. When flying over them, they look different in each photo, which is why it is not so easy to come to a consistent result.

Figure 4.3: Dominican Highway [19]

Quarry Dataset

This quarry dataset examples in figure 4.4 was filmed in 2018 by Wingtra [19].

I used only 50 photos from this dataset, since in subsequent photos there is a sharp jump that does not allow for consistent gradual stitching – thus, in 51 photos, the algorithm cannot localize similarities in keypoints. Of the other features in this dataset, there are significantly fewer keypoints due to the monotonous landscape, which in this situation did not greatly interfere with the result at the beginning and greatly accelerated the process, but subsequently the pictures were greatly distorted. Although the picture looks tolerably complete, it is clear that the relative sizes of some objects at the beginning and at the end can differ greatly from reality.

Figure 4.4: Quarry [19]

4.4 Evaluating results

The table 4.2 describes the metric values paired with datasets, indicating the results of comparisons. The third dataset does not have a value due to the lack of a mosaic reference.

Datasets	SSIM value
Golf Course	0.65
Highway	0.32
Quarry	N/A

Table 4.2: Results for different datasets

Golf Course Result

Figure 4.5: Reference vs Stitched image piece respectively golf

According to the evaluation metrics, the structural similarity index (SSIM) between the stitched image and the reference was approximately **0.65**. Despite this relatively high similarity, the reference image exhibited significantly desaturated colors.

When comparing the images side by side in Figure 4.5, the structural misalignment is not immediately apparent. However, upon closer inspection, it becomes evident that even in a seemingly flat region like a golf course, the reference image appears significantly more geometrically consistent than the stitched output. This deviation is quantitatively reflected by the observed structural difference, highlighting the challenges of accurately reconstructing even relatively planar surfaces under variable imaging conditions.

Highway Result

Figure 4.6: Reference vs Stitched image piece respectively highway 1st

Figure 4.7: Reference vs Stitched image piece respectively highway 2nd

In this scenario, the presence of numerous buildings and complex structural elements resulted in significant differences between the input and reference images. The calculated Structural Similarity Index (SSIM) was approximately 0.32, indicating a low level of visual correspondence. This value is due to strong distortion when comparing relative to the reference; the distances of objects are very different, and some objects are completely shifted, including perspective information in the form of building walls that is captured. Moreover, the comparison of keypoints, although affected by these factors, produced results comparable to those observed in simpler cases, such as the golf course dataset, suggesting some level of robustness in the feature-matching process.

In both examples presented in Figures 4.6 and 4.7 that are from the same created map, it is evident that certain features, specifically the walls of residential structures, are entirely absent from the reference images. Their absence prevents interference between frames, potentially aiding the matching process in those regions. However, in the second comparison example, it becomes apparent that the return flight introduced noticeable changes in structure, shifting objects further than the reference. These distortions result in misalignments and deformation of structural features, further complicating the image registration and stitching process. Figure 4.7 also shows how the size and position of some objects (for example, the main road on the right) have changed significantly in relation to the reference.

Quarry Result

Unlike the two previous datasets, unfortunately, there was no orthomosaic reference for the quarry, so I was unable to test it on the SSIM metric and can only visually evaluate the result. However, I still managed to notice some problems, which I will describe below.

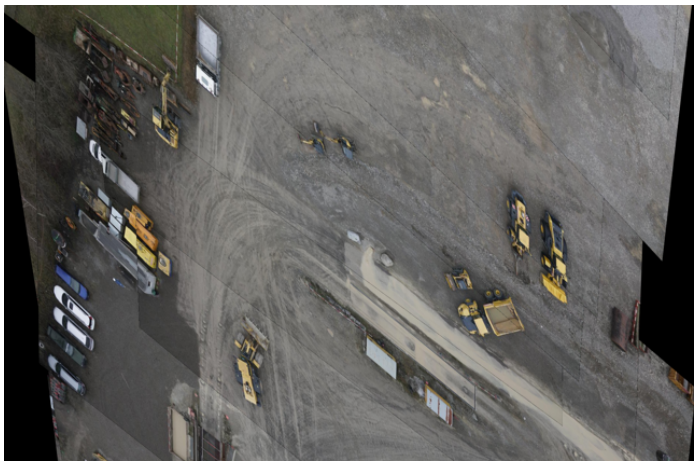


Figure 4.8: Quarry stitching distortion

One of the main details that catches the eye on figure 4.8 is the strong distortion of objects, their flattening during the z-transformation through a homogeneous matrix. Although the objects are superimposed at first glance naturally, perhaps due to the fact that the quarry has a lowered ground level, the image was distorted trying to adjust it to the starting shot, which can be seen in the image.

In order to verify the distortion, I decided to take a small measurement based on the Google Maps satellite. I found roads from this quarry and took a measurement using the built-in function. The real width of the road from the first picture on figure 4.9 was 8.5 meters (with an additional road on the left), the real width of the road from the second picture was 6.5 meters. Then I took a relative measurement in my mosaic without changing the scale and got a road ratio of 3.3 to 4.5, respectively.

The ratio in the roads from the Google Maps reference was 0.73, which means that the second road is narrower in the real world. In my result, a distortion occurred that led to the expansion of the road almost twice (if we imagine that the scale of the first picture is the reference) since the ratio became 1.35, which means that the deformation greatly expanded the road in pursuit of pleasing the coincidences of keypoints. This shows that my method is more suitable for quick map sketches than for full-fledged accurate maps intended for measurement. On the next page, I left a full compressed picture of the stitching on the figure 4.10.

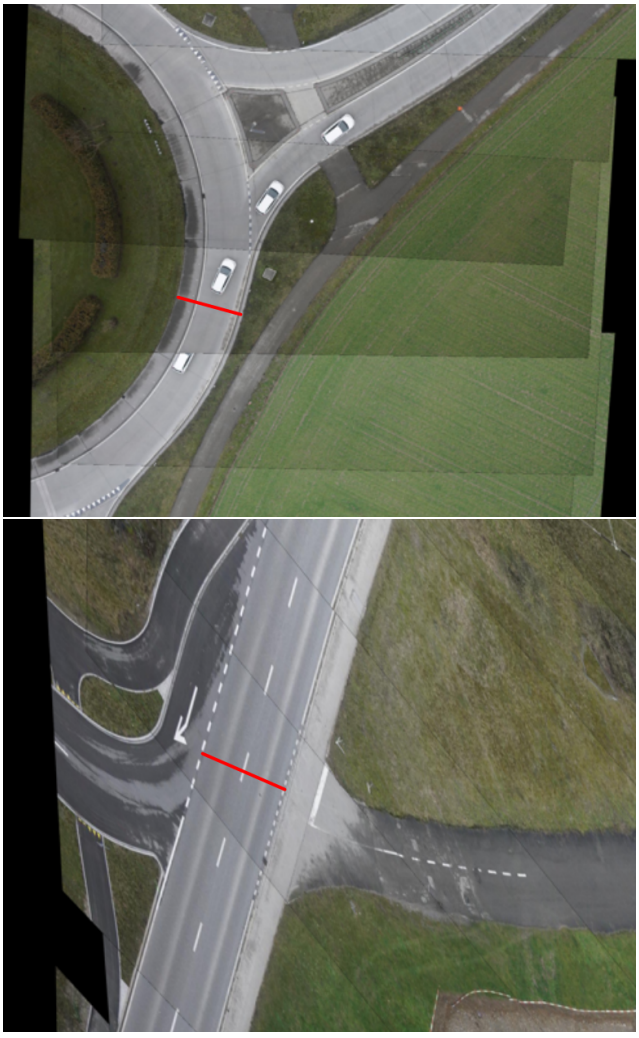


Figure 4.9: Quarry roads width comparison

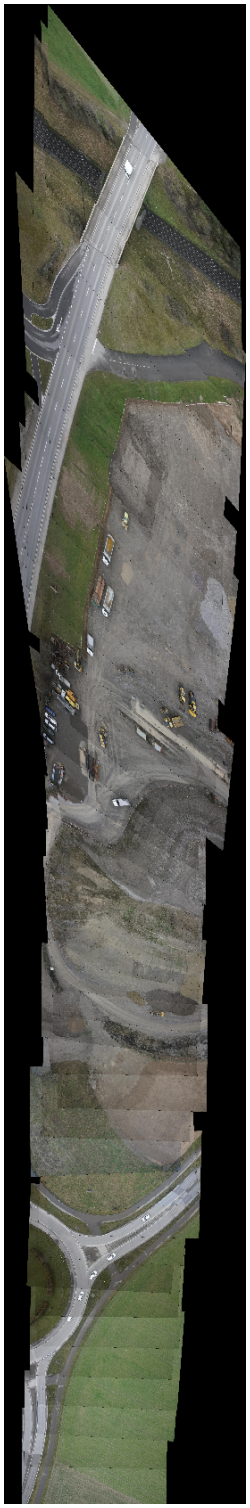


Figure 4.10: Quarry 50 images stitch

Chapter 5

Conclusion

The purpose of this work was to study the necessary techniques and theory of computer vision and to create an algorithm based on them that would be able to create a complete map based on images without metadata. Respectively, I did not take into account the height of pixels, geodata, etc. An additional goal was to study what problems may arise during the process and how they can be solved. One of the key problems was perspective distortion, which is solved by orthorectification, but the most classic solution to this problem would be to use the height metadata of pixels. Also, the problem of consistent image distortion in some situations was discovered, potentially caused by unstable shooting or non-uniform shooting height.

To create the algorithm, I studied many methods of image processing and using the obtained information, such as SIFT, BF Matcher, RANSAC, and others, based on which I created my algorithm, including additional techniques to optimize the process,
for example – keypoint map.

Experiments have shown that for complex surfaces and not particularly sparing in terms of quality of shooting images, the algorithm can produce distorted results, which are more suitable for a quick assessment of the surface, and the creation of a rough map, rather than a map that can be used for measurements. For consistent shooting of flat objects, the algorithm can be taken into account for measurements, but this depends on the purpose. In the future, for accuracy up to the possibility of measurements, the stability of the algorithm can be increased using height data and pixel transformations to get rid of non-orthogonal image objects. In particular, it is possible to come to some kind of color balancing, which can potentially facilitate the stitching process. In the case of creating your own dataset, there is an option to add image processing based on camera parameters to get rid of lens distortions, etc.

Bibliography

[1] DroneMapper. *DroneMapper Datasets*. 2018. Available at: https://dronemapper.com/sample_data/.

[2] Fischler, M. A. and Bolles, R. C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*. ACM, 1981, vol. 24, no. 6, p. 381–395.

[3] Grenzdörffer, G. and Teichert, B. The photogrammetric potential of low-cost UAVs in forestry and agriculture. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.* online. 1st ed., version 1.0, january 2008, vol. 37. Available at: https://www.researchgate.net/figure/Orthophoto-mosaik-Merklingsen-2-M2_fig3_242084686.

[4] Harris, C. and Stephens, M. A combined corner and edge detector. In: University of Sheffield. *Proceedings of the Alvey Vision Conference*. 1988, p. 147–151.

[5] Hartley, R. and Zisserman, A. *Multiple View Geometry in Computer Vision*. 2ndth ed. Cambridge University Press, 2004. ISBN 9780521540513.

[6] Kapsa, J. *Automatic construction of a terrain map by a drone* online. Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of information technology. Available at: https://www.vut.cz/studenti/zav-prace/detail/150224. [cit. 2025-01-17].

[7] Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* online. 1st ed., version 1.0. Vancouver, B.C., Canada: [b.n.], january 2004, p. 28. Available at: https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

[8] Malche, T. What is Image Matching? An Introduction. *Roboflow* online. 20. may 2024. Available at: https://blog.roboflow.com/image-matching/.

[9] Muja, M. and Lowe, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2009, vol. 2, p. 331–340.

[10] OpenCV team. *Brute-Force Matcher – OpenCV Documentation*. OpenCV, 2024. https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html.

[11] OpenWeather. Custom palettes for agricultural applications. *Visualisation of the NDVI index on satellite maps.* online. 22. august 2019. Available at: https://openweather.co.uk/blog/post/visualisation-ndvi-index-satellite-maps-custom-palettes-agricultural-applications. Path: Blog; Agro; Page 3; Visualisation of the NDVI index on satellite maps.

[12] Se Yun Hwang, C. S. H. Real-Time 2D Orthomosaic Mapping from Drone-Captured Images Using Feature-Based Sequential Image Registration. *ResearchGate, Preprints* online. 1st ed., version 1.0. Republic of Korea: Inha University, december 2023. Available at: https://www.researchgate.net/publication/376210202_Real-Time_2D_Orthomosaic_Mapping_from_Drone-Captured_Images_Using_Feature-Based_Sequential_Image_Registration.

[13] Shi, J. and Tomasi, C. Good features to track. In: IEEE. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, p. 593–600.

[14] Singh, C. D. Panorama Stitching. *CMSC426 Computer Vision* online. 1st ed., version 1.0, 2025. Available at: https://cmsc426.github.io/pano/.

[15] Szeliski, R. *Image Alignment and Stitching: A Tutorial*. Now Foundations and Trends, 2006. 116 p. ISBN 9781933019994.

[16] Wang, Z.; Bovik, A. C.; Sheikh, H. R. and Simoncelli, E. P. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*. IEEE, 2004, vol. 13, no. 4, p. 600–612.

[17] Wikipedia contributors. *K-d tree — Wikipedia, The Free Encyclopedia*. 2024. Available at: https://en.wikipedia.org/w/index.php?title=K-d_tree&oldid=1251096500. [Online; accessed 28-April-2025].

[18] Wikipedia contributors. *Scale-invariant feature transform — Wikipedia, The Free Encyclopedia*. 2024. Available at: https://en.wikipedia.org/w/index.php?title=Scale-invariant_feature_transform&oldid=1263577481#:~:text=The%20way%20Lowe%5B,number%20of%20correct%20matches. [Online; accessed 5-April-2025].

[19] Wingtra. *Wingrtra Datasets*. 2025. Available at: https://wingtra.com/mapping-drone-wingtraone/aerial-map-types/data-sets-and-maps/?srsltid=AfmBOorSyk1tDLi--1K7yoegLHMZpWooOLL--irnn51M5VQJ5Bil9OG5.

[20] Zat, V. Compare the objects in the two images and spot the difference. *Comparing Descriptor Matching Algorithms and Options* online. 24. november 2017. revised 23.7.2018. Available at: https://vzat.github.io/comparing_images/week5.html.