Real-time Aerial Photograph Alignment using Feature Matching

Placering av flygfoton i realtid utifrån bildegenskaper

Andreas Magnvall
Alexander Henne

Supervisor : Peter Dalenius
Examiner : Jody Foo

External supervisor : Joakim Andersson
Upphovsrätt

Detta dokument hålls tillgängligt på Internet - eller dess framtida ersättare - under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Andreas Magnvall and Alexander Henne
Real-time Aerial Photograph Alignment using Feature Matching

Andreas Magnvall
Department of Computer and Information Science
Linköping University, Sweden
andma228@student.liu.se

Alexander Henne
Department of Computer and Information Science
Linköping University, Sweden
alehe451@student.liu.se

1 INTRODUCTION

1.1 UAV mapping
Unmanned aerial vehicles (UAV) are commonly used in a number of areas for various practical purposes. One such area is the forest industry, where camera-equipped UAVs can assist forestry by providing high-resolution aerial photographs which can be used to detect forest damage and monitor overall health. UAVs can fly above a forest at low altitude in order to capture aerial photographs of the area. The photos can then be stitched together to create a high-resolution top-down map of the entire forest. This is a cheaper and more environmentally friendly alternative compared to gathering imagery from a high altitude by use of a helicopter, airplane or other vehicle [4].

1.2 IT-Bolaget Per & Per
IT-Bolaget Per & Per is a software company located in Valla, Linköping, Sweden. They focus on developing mobile applications targeted at the forest industry, providing Geographic Information Systems (GIS) to its customers. Some of their apps make use of UAVs for capturing aerial photographs over areas of forest, which are then stitched together in order to create a map of the forest in real-time.

1.3 The problem
In the paper [11] which laid the groundwork for IT-Bolaget Per & Per’s UAV-based mapping, several methods for image stitching were discussed in order to achieve a visually attractive stitched image in a reasonable amount of time. The methods presented in that paper were the following.

- **Image crop**
  - Image crop is described as the simplest method. It works by first aligning the photos according to the position, rotation and height of the UAV while the photo was taken, and then removing the overlap between two adjacent photos. This produced an image with clearly visible seams and placement errors.

- **Alpha blending**
  - This is image crop with blending of the seams between the photos in the mosaic, creating a seamless result, but with parts of the landscape such as trees being visibly cut off as a result of the cropping.

- **Pixel/feature matching**
  - This way of stitching leverages image analysis to find common features in overlapping photos, and aligns, rotates and warps the photos to create a seamless mosaic by picking pixels from both photos. This produced the most accurate result of the three methods but was both unreliable and computationally expensive, making it infeasible at the time.

The **Image crop** method closely resembles the method used today at IT-Bolaget Per & Per. While being fast, the photos have clearly visible placement errors after being stitched, such as roads not connecting, and as such there is room for improvement in this area. IT-Bolaget Per & Per wants to see if the more powerful hardware available in mobile devices today can be leveraged in order to improve the visual accuracy of the resulting map.

Much has improved performance-wise in modern computing devices since the paper was written in 2016. In this paper we are therefore going to investigate how the accuracy of aerial photograph stitching can be improved by leveraging modern hardware and stitching techniques, specifically feature matching. This will be done by researching some of the more prominent feature matching algorithms and evaluating these in regards to both accuracy and performance. While some are faster, others yield a more accurate result.
1.4 Goals
We will revisit feature matching and investigate how suitable it is for real-time image stitching of aerial photographs. A prototype will be developed which can stitch aerial photographs taken from a UAV using feature matching. The goals of this report will therefore be to answer the following questions by comparing the results of newer methods compared to the current method in use by IT-Bolaget Per & Per.

- How do different combinations of feature detection and matching algorithms perform when applied to aerial photograph alignment in real-time?
- Which variant of feature matching yields the best result when stitching aerial photographs in real-time?

1.5 Delimitations
Since IT-Bolaget Per & Per primarily targets iOS devices, the work on this report will be focused on iOS and the prototype will be created for iOS only. Also, while many advanced UAVs exist with useful technologies such as LiDAR sensors which could be used for 3D-mapping of the surface, those are usually expensive and not what IT-Bolaget Per & Per is currently targeting. Therefore, the work on this report will only target the more common UAVs which are mounted with RGB cameras. As the photos should be stitched and presented on the device in real-time, the stitching has to complete within a limited time frame.

Also, because of the limited project timeframe, the work will be focused on purely image alignment and how this can be done.

2 BACKGROUND
In order to select a suitable method for image stitching of aerial photographs, some requirements have to be established.

2.1 Real-time stitching
Since the stitching is being performed on an iPad operated by a user, there are limitations on how much time can be spent stitching the captured photographs. Ideally photographs should be stitched and displayed in the application immediately after they have been captured, but stitching can be a time-consuming process. It cannot be deferred too long into the future because the operator would then have to keep the application open for a long time after the UAV has finished capturing all the photographs. For this reason, stitching should be performed whenever a photograph has been taken and received from the UAV. This could possibly limit how well the method performs, as less information is available compared to if the stitching would be performed only after the last photograph has been received.

2.2 Top-down stitching
The aerial photographs that are taken by the UAV capture the ground and forests from up in the air, with the camera aimed directly down towards the ground. This differs significantly from many applications of image stitching which assume the photos are panoramic. This means that they were captured while rotating the camera around a fixed point in space. By contrast, for the top-down photos in this application, the UAV’s camera is translated in space above the forest rather than rotated around a fixed starting point. The method must be developed with this taken into account.

2.3 Requirements for feature matching
First of all, the method has to be able to work well on aerial photographs taken above forest areas. This means that affine invariance is an important property, i.e. the ability of the method to match pixels in the overlap between photos taken from different angles. Rotation invariance is not as important, because we expect the rotation of the UAV taking the photos to be roughly the same between photos. Scale invariance could be important in specific situations, such as when capturing photos above areas with varying heights, where the photos would have been scaled to achieve a certain resolution.

The methods need to finish in a timely manner such that the operator of the map application can still see the map being created in real-time. Currently, IT-Bolaget Per & Per’s GPS-based method does not take much more than 20 milliseconds per photo. While less time expenditure is positive, it is not a major problem if the method takes several seconds per photo. There are also considerations that have to be made in regards to the hardware of the device the stitching will be performed on. In particular, system memory usage is limited by the maximum memory available on the device, so the method cannot exceed that.

Finally, the method has to be able to deliver a visually correct map. This is one of the primary problems set out in this paper. The map should have no greatly offset or misaligned images, or gaps, unless gaps are intentional feature of the data set. These types of errors are easy to spot at a glance for anyone observing the map.

3 THEORY

3.1 Stitching using feature matching
Features are distinct areas in an image which can be extracted and used to find the same areas in other images [13]. Feature detectors are algorithms that analyse the pixels in an image in order to find features. It first detects keypoints, which are essentially pixel coordinates for distinct areas. The keypoints and surrounding pixels are then passed into a feature descriptor algorithm. This produces feature descriptors for the areas around each keypoint. These features are more or less invariant to rotation, scaling and other transforms, so that the same feature can be detected in other images.

Matching is the process of matching the features found in two images, such that parts that are visually similar between the images are paired together.

3.1.1 Stitching process
The process we are going to use borrows from previous research in Computer Vision and works as follows [13].
1. Use a feature detector to find features in the two photos that are going to be stitched together. These photos require a certain amount of overlap, where usually the more overlap there is, the more reliable results the feature matcher can deliver.

2. Use a feature matcher to find matching features in the overlap between the photos. An visualisation of this process can be seen in figure 1. A full-size version is present in appendix A.

3. Create a homography matrix\(^1\) based on the keypoints that match.

4. Transform one of the images using the homography matrix.

5. Overlay the transformed image over the other one at the place where the features match.

### 3.1.2 Selection of feature detection algorithms

The computer vision software library OpenCV supports a vast amount of feature detection and descriptor extractor methods using its 2D Features Framework\(^2\) and xfeatures2d\(^3\) namespace. AKAZE, KAZE, BRISK, FAST, ORB, SIFT, SURF, FREAK and DAISY are some examples.

The following feature detection and descriptor extractor methods were selected based on experiments in other literature [1][10][14] and the previously outlined requirements in the Background section on page 2.

Scale Invariant Feature Transform (SIFT) is a feature detection and feature descriptor method created by David G. Lowe and presented in 1999 [6]. SIFT is one of the most widely used feature detection algorithms [15] and was patented until recently\(^4\). The algorithm is invariant, which means that it is consistent in finding features no matter the scale, translation or rotation. It is also fast and robust.

Speeded-Up Robust Features (SURF) is a patented\(^5\) feature detection and feature descriptor method created by Herbert Bay, Andreas Ess, Tinne Tuytelaars and Luc Van Gool, which was presented in a 2007 paper [2]. SURF is built on previous methods for detecting and describing features in visual imagery, specifically 2D images. SURF outperforms, according to the authors, previous methods in both computation and comparison. SURF, or its predecessor SIFT, has been found to be one of the best feature detection methods in several studies [1][10][14]. SURF is capable of feature detection, descriptor extraction and feature matching.

Oriented FAST and Rotated BRIEF (ORB) is an alternative to SIFT and is a feature detector and descriptor. It was developed because the authors thought the primary alternatives, SIFT and SURF, were too performance-heavy. The authors instead wanted a faster algorithm with similar performance [12]. The technique used is built on BRIEF, which is a binary descriptor. This is different from the other one normally used, float descriptor. This also means that different types of distances has to be used when matching.

While being substantially faster than SIFT, there are many cases where SIFT performs better accuracy-wise [14]. However, since performance is of importance, especially because of the limited capability of iOS devices, ORB may prove to be a good candidate. If it would be “good enough” accuracy-wise, and at the same time be the fastest, it would be a perfect match for image stitching.

Binary Robust Invariant Scalable Keypoints (BRISK) was proposed in 2011 in an effort to create a robust method for detection, description and matching of features while also achieving good computational speed. It turned out to be faster than SURF while being comparable in matching performance [5], making it a possible alternative to SURF if computational performance is lacking. When it comes to affine invariance, this method generally performs better than SURF, making it an interesting method to investigate for our use case. It does however perform worse when downscaling [14], which might affect the result in specific cases such as when the UAV captures terrain with varying heights.

### 3.1.3 Selection of matching algorithms

The following matching algorithms are supported by OpenCV.

Brute-Force Matcher (BF) is a simple feature matcher [7]. The matcher uses the descriptor of one feature from one set of features and matches this with another feature from another set. The feature in the second set which was closest in distance to the first one is returned. This matcher has \(O(n^2)\) time complexity.

Fast Library for Approximate Nearest Neighbors (FLANN) is a matcher which finds approximate nearest neighbours [7][9]. This matcher is faster than the BF matcher, but it is not guaranteed to find the best match. The precision of the FLANN matcher can be improved by adjusting its parameters, at the cost of making the algorithm slower.

\(^1\)A matrix describing how coordinates in one plane can be represented in another plane
\(^2\)https://docs.opencv.org/3.4/d5/d51/group__features2d__main.html
\(^3\)https://docs.opencv.org/3.4/d6/d66/namespacecv_1_1xfeatures2d.html
\(^4\)https://patents.google.com/patent/US6711293B1/
3.2 Related works
There exists a lot of research in the area of image stitching. However, many focus on other applications such as panorama creation instead of mapping using top-down aerial photographs. Most are also not as forest specific as our work. We also have not seen any research on real-time stitching where the stitching is performed on a iOS device except for the previous paper accomplished at IT-Bolaget Per & Per. However, there is still related literature to be found.

Brown and Lowe [3] stitch together multiple panoramic images into one image, also using invariant features to match between the images. Several of the panoramas contain similar elements such as ground covered completely in snow, and this can present a challenge to a feature matcher because of the great similarity between the images. Their approach manages to stitch together such images with good results, and this supports the idea that it can be feasible to stitch together images containing similar elements such as aerial photographs of forests.

Moussa and El-Sheimy [8] present an approach for improving processing time of image stitching. Similarly to our work, they were also researching image stitching in combination with the use of UAV. While using a similar feature detection algorithm as the ones previously mentioned, they afterwards created a triangulation system. This made it possible to only look at nearby images during the matching stage, and thus saved processing time. However, the approach works by waiting for the last image to arrive before the stitching is performed, and this does not allow for real-time stitching.

4  METHOD
This chapter presents the method used for aligning photographs in real-time using feature matching. The chapter starts with a prestudy introducing the tools and some background to the method. Then it moves on to the implementation. Finally, it describes what experiments have been done and how the method was evaluated.

4.1 Prestudy
During the start of this project, we did a prestudy evaluating how suited the widely used computer vision library OpenCV was for the purpose of image stitching. One of the goals of this paper was to develop a fully suited prototype which could be used to render a real-time map while using the methods researched. However, this results in a lot of overhead in comparison to just exploring if and how basic stitching can be done. For this reason, we did a small study to find out if OpenCV can be used to solve the problem, and to get an understanding for the subject.

OpenCV has built in support for the feature matching methods we investigate, which lends it well to be used in a quick proof-of-concept implemented in the Python programming language. Python is in our opinion simple to use, very portable and capable of acceptable performance. It also has a OpenCV implementation unlike Swift, the language that is used while programming iOS applications.

The prestudy showed that implementing these techniques is feasible, and that an attempt can be made to implement a solution in a demo application written in Swift, Objective-C and C++.

4.2 UAV
The DJI Matrice UAV used has a RGB camera which can be activated to capture photos of the ground while in the air, with a maximum altitude of 120 metres above the ground. It features a barometer and GPS functionality which can give relative height changes with high precision and absolute heights above the ground with low precision.

4.3 Mapping procedure
IT-Bolaget Per & Per’s mapping app works roughly as follows. An operator draws out a flying path on an iPad. The UAV then takes off and starts flying in its path. While flying, the UV takes photos and sends those to the iPad. While images are received the stitching process starts, and the resulting stitched map is shown in real-time in the application. The process continues as long as more photos are received, adding more photos to the map.

4.4 Implementation
This section describes the steps for implementing the method, starting off by describing the development environment and existing code, and then presenting our solution.

4.4.1 About
A prototype was implemented for iOS using the programming languages Swift, Objective-C and C++. The computer vision library OpenCV was used for calculations because it already had support for the devised methods we decided to evaluate. Since OpenCV is written in C++ and lacks complete wrappers for Swift we used a Objective-C bridging header to make it possible to call C++ functions from Swift code. All of the code using OpenCV was then written in C++ and called from Swift through the Objective-C bridge. We used OpenCV version 4.5.2, and compiled it with the OpenCV contrib module to get access to the proprietary SURF detector.

4.4.2 Existing solutions
During development we first tried stitching using the OpenCV stitching_detailed example6. We also tested the high level stitching API7. However, none of those were capable of stitching together a whole map as our goal was. This led us to having to research more and further look up existing implementations. We thought that we would find tons of literature on the topic but actually almost came up empty handed. While image stitching is a major area, image stitching in this context is not. We therefore chose to develop our own solution while reusing several parts from the OpenCV feature matching examples.

---

1. https://docs.opencv.org/3.4/d9/dd8/samples_2cpp_2stitching_detailed_8cpp-example.html
2. https://docs.opencv.org/master/d8/d19/tutorial_stitcher.html
4.4.3 Our solution

After the first photograph has been received from the UAV, it is positioned on a Mapbox map according to the photograph’s embedded metadata. Mapbox is the map framework IT-Bolaget Per & Per uses to display stitched photos. The metadata contains the UAV’s GPS position, camera field of view, altitude above ground and flight yaw at the time of photo capture, which is used to estimate a quadruple containing the GPS coordinates of the four corners in the photograph.

The distance from the image’s center coordinate to the corners is calculated by considering the horizontal and vertical focal length of the UAV’s rectilinear lens, and the distance from the camera to the ground, i.e. the altitude: \( d = \text{altitude} \times \tan\left(\frac{\text{focal length}}{2}\right) \). The distance from the camera to the ground cannot be known for certain and can have a significant effect on the scale of the image when laid out on the map, therefore it is important that it is estimated as close as possible. The altitude of the UAV is estimated using three different methods. This is also the order the data is prioritised based on availability.

Firstly, the height can be set to a pre-defined constant number of metres such as 120 metres. This ignores the altitude differences between photographs, and results in the photographs having the exact same scale while viewed on the map. As such, this can introduce errors when photographs are side by side and the distance from the ground when the photographs were taken differs.

Secondly, height map data with data points every 50 metres is downloaded from Lantmäteriet and is used in combination with the absolute height reported from the UAV in the metadata. The terrain height from the height map is subtracted from the UAV’s absolute height in order to arrive at an estimate of the distance between the UAV and the ground.

As a last resort, the relative height from the UAV is used. This is the UAV’s reported height difference from where it started. A problem with this is that it does not take into account the height differences in the terrain.

After estimating the corner quadruple, features for the new photo are detected and stored, and the photo is placed on the Mapbox map.

4.4.4 Matching

For every photo after the first one, the feature matcher is run with a `match_conf` match distance threshold, individually matching with all previously processed photos which have corner quadruples that overlap with the new photo’s quadruple. The most confident match out of these is chosen. Confidence in this case is a decimal value with a higher value indicating more similarities between two compared photos. A minimal confidence value `conf_thresh` can be set in the code to optimise the trade-off between fewer good matches and more inferior matches.

If the feature matching for the new photo fails to yield a confident match, the photo is instead positioned purely according to the data present in the image metadata, like how the first photo was positioned.

To reduce the number of features and consequently reduce the execution time, we added an image scaling parameter `work_megapix` which downsizes the image to `work_megapix` megapixels. This parameter can be adjusted for a trade-off between match quality and time spent matching the features. The operation is additionally sped up by only considering the photos that overlap with the new image. This is determined by checking if the new image’s corner quadruple intersects with the ones of all other photos, then only attempting to match with those photos that overlap.

We could switch between the previously mentioned algorithms: SIFT, SURF, ORB, and BRISK for feature detection. For feature matching, we used and tested both the BF matcher and FLANN matcher algorithms.

We define the photo that the new photo matched with to be the parent photo of the new photo. This is similar to how a node in a graph has a parent. Since a new photo can match with any already processed photo that it overlaps with, it would be wrong to always define the parent image as being the previous image. Any image that is positioned using GPS, and as such is not positioned relative to another photo, does not have a parent.

4.4.5 Relative transformation

After matching the new photo to the parent photo, the homography matrix is extracted from the match result. A matching photo with a resolution of for example 5472×3648 has the corner points top-left (0, 0), top-right (5472, 0), bottom-left (0, 3648) and bottom-right (5472, 3648). These corners are multiplied by the matrix in order to scale, rotate and translate the corners to where the matcher suggests the new image should be positioned relative to the matching image.

4.4.6 Real-world rotation

After this stage, the photo needs to be rotated to match the real-world orientation, which is the rotation of the UAV relative to north on a compass. When extracting features, we pass in unaltered photos from the UAV’s camera without rotating them relative to north. This avoids challenges such as having empty areas on the sides caused by rotation by angles other than multiples of 90 degrees, and also potential image quality degradation or distortion of the images caused by transforming them.

To perform this type of rotation, the flight yaw degree found in the image metadata of the root parent is used. This way, all children are rotated in the same compass direction. For instance, the flight yaw of the first photo might be 90°, resulting in all its children also being rotated by 90°.

The rotation takes place around the parent photo’s top-left corner for every point in the photo to be rotated. The reason for the rotation being around the top-left corner is that OpenCV uses the top-left corner of passed in images as the origin, and agreeing on a common origin eliminates unnecessary translations.

4.4.7 Map placement

During the last stage of the process, the pixel-based corner points from the previous stages require conversion from pixels to GPS coordinates. This is done by first converting the points from pixels to
4.5 Experiments

Experiments were performed on a dataset containing 88 images with a 40% overlap between images in the flying direction of the UAV. The dataset was provided by IT-Bolaget Per & Per and contains a variety of relevant features, such as dense forest, sparse forest and paths. They were captured at the usual height of 120 metres and with occasionally varying ground level heights.

In order to find out which feature detector performs the best, we benchmarked the four different feature detection methods. The following data was benchmarked: keypoints per image, matches per image, time per image, total time, total failed matches, and whether they were successful in matching. Successful in this context means that an experiment configuration fulfills the requirements on feature matching laid out in the Background section on page 2.

The following properties were set for our prototype when running the benchmark. For ORB, the maximum number of features \( n_{\text{Features}} \) was set to 100000. This way it included all the features in the photos in our data set. A lower value caused the matcher to fail at finding any matches. As advised by the OpenCV documentation \([7]\), LSH index was used for the binary descriptors ORB and BRISK while using FLANN, and Hamming distance was used while using BF. For the floating-point descriptors SIFT and SURF, KD Tree index with L2 norm was used for both FLANN and BF.

The \( \text{match\_conf} \), \( \text{work\_megapix} \) and \( \text{conf\_thresh} \) settings used for each detector can be seen in tables 1 and 2 below. The values were derived experimentally by starting off with the values used in OpenCV’s \texttt{stitching\_detailed} \( ^8 \) example. The original values did not result in a satisfactory number of matching photos. Perhaps they assume much greater similarities between photos than our data set offers. The values were modified until the detectors yielded the greatest number of matches without any gaps or obvious visual errors.

\[ \begin{array}{ccc}
\text{match\_conf} & \text{work\_megapix} & \text{conf\_thresh} \\
\hline
\text{SIFT} & 0.3 & 1 & 0.3 \\
\text{SURF} & 0.3 & 3 & 0.3 \\
\text{ORB} & 0.3 & 1 & 0.3 \\
\text{BRISK} & 0.3 & 1 & 0.4 \\
\end{array} \]

Table 1: Properties when running the benchmark using BF matcher.

\[ \begin{array}{ccc}
\text{match\_conf} & \text{work\_megapix} & \text{conf\_thresh} \\
\hline
\text{SIFT} & 0.3 & 1 & 0.3 \\
\text{SURF} & 0.3 & 3 & 0.3 \\
\text{ORB} & 0.3 & 3 & 0.3 \\
\text{BRISK} & 0.3 & 3 & 0.4 \\
\end{array} \]

Table 2: Properties when running the benchmark using FLANN matcher.

4.6 Evaluation

The configuration which performed the best in regard to total time spent and the number of matched images during the experiments was then selected for the evaluation stage.

For this stage, we created a Google Form which was sent out for people to fill. The Google Form consisted of questions related to the method. The primary goal of this form was to find out whether the previous GPS-based method or the new method performed better visually during image alignment in relation to the world.

The form consisted of 11 questions which can be seen in appendix D. There are 11 pairs of alternatives where one alternative shows the GPS-positioned image compared to a satellite image of the same area and the other alternative showing our method compared to a satellite image of the same area. Ten pairs were randomly selected from the matching pairs of images. Since it can be hard to see a difference in many cases, especially in pure forest images, one additional image was picked which features a wider variation of content including roads. This, we thought, would make it easier for the survey participants to compare the alternatives in that case. The alternatives were also randomly selected such that alternative A was the GPS-positioned image for some images and B for others.

5 RESULT

5.1 Experiments

The benchmarking was done on a M1 Mac Mini using the parameters specified in tables 1 and 2. The results are shown in tables 3 and 4.

SIFT, SURF and ORB with BF matcher produced maps that were successful according to the requirements for feature matching in the Background section on page 2. BRISK with BF matcher produced a map with errors which can be identified without doubt. For example, the road in the data set is disconnected, as shown in

---

\( ^8 \)https://docs.opencv.org/3.4.4/d/w/ddh/samples_c2pp_2stitching_detailed_c2pp_example.html
### Table 3: Benchmark data using BF matcher. Format: Mean±Standard deviation.

<table>
<thead>
<tr>
<th></th>
<th>SIFT</th>
<th>SURF</th>
<th>ORB</th>
<th>BRISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypoints/Image</td>
<td>11170±11418</td>
<td>32029±33560</td>
<td>64759±3591</td>
<td>24246±3528</td>
</tr>
<tr>
<td>Matches/Image</td>
<td>118±145</td>
<td>248±380</td>
<td>451±491</td>
<td>137±231</td>
</tr>
<tr>
<td>Time (ms)/Image</td>
<td>6874±3271</td>
<td>39840±27543</td>
<td>61801±19083</td>
<td>11471±5341</td>
</tr>
<tr>
<td>Total time (ms)</td>
<td>604 996</td>
<td>3 505 920</td>
<td>5 438 496</td>
<td>1 009 462</td>
</tr>
<tr>
<td>Matched images</td>
<td>57</td>
<td>57</td>
<td>52</td>
<td>60</td>
</tr>
<tr>
<td>Successful</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 4: Benchmark data using FLANN matcher. Format: Mean±Standard deviation.

<table>
<thead>
<tr>
<th></th>
<th>SIFT</th>
<th>SURF</th>
<th>ORB</th>
<th>BRISK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keypoints/Image</td>
<td>11170±11418</td>
<td>32029±33560</td>
<td>64759±3591</td>
<td>24246±3528</td>
</tr>
<tr>
<td>Matches/Image</td>
<td>139±135</td>
<td>460±345</td>
<td>658±778</td>
<td>394±263</td>
</tr>
<tr>
<td>Time (ms)/Image</td>
<td>768±100</td>
<td>1951±282</td>
<td>36829±9068</td>
<td>24570±30633</td>
</tr>
<tr>
<td>Total time (ms)</td>
<td>67 602</td>
<td>169 941</td>
<td>3 241 006</td>
<td>2 188 610</td>
</tr>
<tr>
<td>Matched images</td>
<td>60</td>
<td>47</td>
<td>56</td>
<td>67</td>
</tr>
<tr>
<td>Successful</td>
<td>✔</td>
<td>✔</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

 BF matcher should in theory be able to deliver the greatest number of matched images possible but we saw that it was not the case. However, FLANN only succeeded for two of the four descriptors compared to three out of four for BF matcher. BF matcher resulted in a greater number of matched images on average for successful configurations, but FLANN matcher both took orders of magnitude less time, and also resulted in the most matched images for the SIFT configuration.

As described in the Background section’s requirements for feature matching, the purely GPS-based method took approximately 20 milliseconds per image. All the configurations resulted in a time expenditure many times higher than that. For instance, ORB with BF matcher took more than 3000 times longer than the GPS-based method. The fastest configuration, SIFT with FLANN, took almost 38 times more time than the GPS-based method. Still, SIFT with FLANN was done with each image under one second, which is well within the constraints.

SIFT with the FLANN matcher was found to both result in the greatest number of matching images of the successful configurations and also the least amount of time per image, so this configuration was chosen to be used as the basis for the evaluation of the method.

### 5.2 Map

Figure 3 displays the complete data set with 88 images positioned on the map using our method. From a glance the difference is not major when compared to the map derived from the GPS-based method visible in appendix B, but when zooming in the differences can be observed clearly. A full-size version of figure 3 is present in appendix C.
5.3 Evaluation

The image alignment survey showed that the feature matched image alignment performed slightly better visually overall. For the survey conducted, 15 responses were received. Out of the 11 images, 7 images had a majority selecting the feature matching alternative (64%). Three out of those 7 images had 60% or more answers in favour of feature matching. The selected image with a road was one of those three. The results can be seen in table 5.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>53.3%</td>
<td>60%</td>
<td>53.3%</td>
<td>53.3%</td>
<td>40%</td>
<td>40%</td>
<td>46.7%</td>
<td>26.7%</td>
<td>46.7%</td>
<td>40%</td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>46.7%</td>
<td>40%</td>
<td>60%</td>
<td>46.7%</td>
<td>46.7%</td>
<td>60%</td>
<td>60%</td>
<td>53.3%</td>
<td>73.3%</td>
<td>53.3%</td>
<td>60%</td>
</tr>
</tbody>
</table>

Table 5: Survey results for all 11 questions.

6 DISCUSSION

6.1 Result

We consider our result to be good but not perfect. The optimal scenario would be that every image pair would match and also do it well. This is however not always the case which can be seen in the results. From the data set tested, it can be seen that the number of matched photos varied from 47 to 67 photos in the data set of 88 photos. Even though BF matcher should in theory be the matcher capable of finding the greatest number of matches, this was not the case as shown in the result. Why this is the case, we are not exactly sure. It could be that since FLANN is an approximate algorithm, FLANN could wrongly match features.

Our result should be easy to replicate since the algorithms used are consistent and not relying on random factors. Not relying on random factors would also make this result reliable.

Regarding the real-world image alignment, which was the major problem set out to be solved in this paper, it can be said that it has improved but is not perfect. Previously, in the GPS method, several errors could be seen that were caused by GPS inaccuracies such as roads not connecting, parts of images being duplicated, and more which the feature matching method is capable of solving. On the other hand, the feature matching is sometimes unreliable and not always the ultimate answer, at least in its current form.

One thing that can be said is that mapping forestry is a difficult task. Since forests both have many similar parts yet also a lot of variances it is not hard to find features. However, the problem likely lies in the matching. This can be seen by looking at our benchmarks. Even though there are a lot of keypoints found per image, the matches are very few in comparison. Two photographs captured above a forest at even a small difference in position cause the trees to be shown from vastly different angles. In one photo, the left side of a tree might be shown, and on the other photo taken adjacently, the right side of the tree might be shown. This presents a challenge for the matching, as the feature descriptor might not be invariant enough for this level of difference between photos.
A data set with a greater overlap could potentially have led to more high-quality matches. The reason being that the areas which overlap between photos are more similar the larger the overlap is. The problem with a greater overlap is that more photos will need to be captured. An increase in the number of photos leads to a longer processing time. If it takes more time to process the photos than it takes for the UAV to travel and capture the next photo, the process can no longer be said to perform in real-time. Since it is desired that the images are shown in real-time, a greater overlap might not be feasible. More photos would also lead to an increase in memory usage.

6.2 Method

6.2.1 Prestudy

Doing a prestudy on the topic prepared us somewhat for the task ahead. The problem was much harder than we had anticipated. Much of the literature and internet resources on image stitching are focused on panorama creation, which is a similar problem but not the same as stitching aerial photographs.

After doing a prestudy, we initially set out to improve the appearance of the seam between photos and started with the implementation of a prototype. In the beginning we tried to implement a custom algorithm taking inspiration from existing OpenCV examples and our own prestudy. However, this was difficult and did not yield an adequate result. Both because of the lack of related literature and information but also because we thought that it was difficult to understand how the feature detection algorithms perform. Since there was barely anything to compare to, much of the work consisted of trial and error.

This early work did give us more experience and understanding which made it easier to progress. At one stage we thought that the first problem of image alignment was solved and progressed to improving the seam. Blending and multi-band blending in particular seemed like good candidates. However, multi-band blending was complex to work with. We also understood that our image alignment was in fact rarely working adequately, and especially not for more than just a few images. This led us to change our focus to purely image alignment and our current implementation.

6.2.2 Implementation

Our current implementation consists of many steps, but is also thorough and can be split into different parts. This division both makes it easier to understand, yet also easier to debug and analyse.

There are four parts of the algorithm that we do not control and can therefore have a negative impact. Firstly, since photos that do not have a parent are positioned according to the GPS, they could potentially be placed far away from where the photo should be. This error would then affect every sequential photo until the next non-matching photo is positioned according to the GPS. Nevertheless, the same error can repeat again. The second part the methods relies on and which can not be controlled is the height data from Lantmäteriet. While it would seem unlikely that this data was wrong, it could still lead to major errors related to for example scaling of the images. If the images have the wrong scales, they would not align correctly in relation to the real world. Thirdly, the keypoints detected are crucial to correctly matching the overlapping images. If the keypoints are wrong, perhaps caused by a fault in the detector, the images could be aligned incorrectly. If instead too few keypoints are found, the image would not align at all. Lastly, depending on the matcher, the matches could also be wrong which could cause similar errors as those in the previous part.

The first problem is not solvable since our method relies on a previous image in order to have anything to match with. It must therefore have a non-feature matched image in the beginning to start with. Any problems related to GPS could only be solved by a higher GPS precision, which is related to the hardware of the UAV. The second problem, if it is a problem, could perhaps be solved by a more accurate and improved height map. It is likely that a very small error is caused by the height map only having a 50 metres accuracy. While Lantmäteriet does have a 1 metre accuracy height map, this was not something that we had access to during the project. The third problem is hard to solve, since some of the feature detectors tested are the most popular and widely used feature detectors, and are found to perform the best in several studies [14][10]. The same applies for the last part related to the matching.

Apart from this, the implementation does not rely on anything else. The remaining part of the implementation is instead created by the authors and has also been tested. The implementation should also be relatively easy for someone else to recreate or test out using the source code.

6.2.3 Evaluation

Since it was hard to objectively decide which method aligned the best out of the two methods, we chose to conduct a survey. In the optimal scenario, a fully objective method would have been the best. However, such a method was difficult to find and it would have been hard to implement within our limited time frame. A survey evaluation was therefore chosen to get some sort of objective indication. For this survey, it was decided to not inform the participants about what different methods were in works since this could have affected the result. The goal was simply to determine what looks best. The decision to randomise the alternatives order served to remove bias.

This method for evaluation of our prototype does have some flaws. The major flaw is probably that there is little difference in many cases between alternatives. This could have caused the respondents to think that there is no difference and to pick an image randomly. Another flaw is that the images are quite small in Google Forms. This makes it harder to spot the difference between alternatives, especially if the form is viewed on a mobile device. Finally, with both a small number of images, only one data set, and not very many respondents it could be difficult to argue that the method is good or bad in general.

6.3 Source discussion

The sources picked were chosen with care. Peer reviewed literature was our primary source with some additional first-hand sources from OpenCV on how some parts of OpenCV work.
6.4 Work in a larger context
The work can be put in many different important contexts. Real-time mapping has many uses. It is not only limited to mapping forestry but could possibly also be used to quickly, accurately and cheaply map areas such as cities, mountains or a whole country using a UAV.

UAVs also only use electricity, and this could potentially be a lot more environmentally friendly than using for example an airplane, or even a satellite for mapping. When increasing the accuracy and quality of maps created by a UAV, perhaps more people would be open to use them. This could in return help the environment.

7 CONCLUSIONS
7.1 Summary
For the purpose of aligning aerial photographs in real-time, a prototype for mapping with the help of image alignment using feature matching has been implemented. In order to find the position for the image alignment, two overlapping photos are feature matched and then aligned. This is done by finding features in both images and then matching those features together. This way, a transformation matrix can be extracted which describes the translation, rotation and scaling which should be applied to one image in order to align that image correctly relative to the other. If the matching fails, the image is instead positioned using embedded GPS data and altitude. The best feature detector was found to be SIFT by evaluating the four different detectors laid out in this paper. This evaluation was done after benchmarking the different detectors. The two different possible OpenCV matchers BF and FLANN were also evaluated. In this case FLANN matcher performed better, being around 10 times faster than BF and yielding a similar number of matches.

By inspecting the result, feature matching could in some cases be seen to drastically improve the connection between photos. To further compare the new and old method, a survey was conducted resulting in a slight favour of the new feature matching method.

Since the method succeeds in matching a large number of image pairs, which in return yields more often than not a better result than the original GPS-method, it can be seen as useful and effectively an improved version. While not perfect, it still manages to resolve the original problems of image alignment to a large extent.

7.2 Future works
It would be interesting to see how well our prototype performs on additional devices, for example iPads. The prototype could also be tested using more types of UAVs, and more settings could be experimented with. It is possible that the result would become even better by doing this.

Since our prototype does not succeed in finding a match for every pair of images, it is not perfect. In the optimal case every image should be matched together. To limit the scope of the work, the reason for not all image pairs matching was not researched extensively. However, when our prototype fails to match and falls back to GPS, errors start to show. If every image would match, this would likely not be the case. There are likely optimal combinations of feature detectors, matcher algorithms, and settings that result in more matching pairs. As of now, the current hardware is also in some cases the limiting factor, as even with severe downsampling of the images, the BF matcher takes many times longer than FLANN matcher with the same scaling.

Currently, when the alignment in many cases can be seen as perfect, the visual appearance of the seam could be improved if that is wanted. For this purpose, using multi-band blending, exposure adjustment and other techniques could be useful. We decided to not implement this further in our implementation since we believed it was time prohibitive to implement these, even when using OpenCV. We also primarily focused on the alignment and not on perfecting the seam. It also would not have made sense to improve the seam before the seam was correctly aligned.

REFERENCES
Appendix A  FULL SIZE FEATURE MATCHING EXAMPLE

Figure 6: Feature matching in the works. Blue dots are features detected and red lines show matches between features from both images.
Figure 7: The full data set using the GPS-based method.
Figure 8: The full data set using the feature matching-based method.
Appendix D  SURVEY QUESTIONS

“The image on the right of the red divider shows a satellite image of a forest. On the left, the same satellite image has photographs layered on top. For alternatives A and B, please compare the image on the left of the red divider with the image on the right. Then select which alternative you think best matches the real world satellite image on the right.”

Figure 9: Image 1 Alternative A (Feature matching)  Figure 10: Image 1 Alternative B (GPS method)

Figure 11: Image 2 Alternative A (Feature matching)  Figure 12: Image 2 Alternative B (GPS method)

Figure 13: Image 3 Alternative A (GPS method)  Figure 14: Image 3 Alternative B (Feature matching)
Figure 21: Image 7 Alternative A (Feature matching)

Figure 22: Image 7 Alternative B (GPS method)

Figure 23: Image 8 Alternative A (Feature matching)

Figure 24: Image 8 Alternative B (GPS method)

Figure 25: Image 9 Alternative A (GPS method)

Figure 26: Image 9 Alternative B (Feature matching)
Andreas Magnvall

BESLUT I ÄRENDE LM2021/013596

BESLUTSDATUM: 2021-05-12

Ansökan om Spridningstillstånd
Lantmäteriets beslut
Lantmäteriet beslutar att meddela Andreas Magnvall tillstånd ur totalförsvarssynpunkt att sprida en sammanställning av geografisk information enligt ansökan.

Beslutet gäller tills vidare.

Ärendet

I ansökan har Andreas Magnvall framförts bland annat följande:


Skälen för beslutet

Med geografisk information avses lägesbestämd information om förhållanden på och under markytan samt på och under sjö- och havsbottnen enligt 2 §, punkt 1. Lag (2016:319) om skydd för geografisk information (GiL).

Med en sammanställning av geografisk information avses geografisk information i form av avbildning, beskrivning eller mätning enligt 2 §, punkt 4 GiL.

Sammanställningar av geografisk information över svenskt landterritorium får inte spridas utan tillstånd av Lantmäteriet om den insamlats från luftfartyg genom fotografering eller liknande registreringar. Tillstånd ska ges om spridningen inte kan antas medföra skada för totalförsvaret. Detta framgår av 9 § GiL.
LANTMÄTERIET GÖR FÖLJANDE BEDÖMNING

Den information Andreas Magnus avser att sprida är att betrakta som en sammanställning av geografisk information som är tillståndspliktig.

Lantmäteriet gör bedömningen att spridning av den sammanställning av geografisk information som framgår av ansökan, inte kan antas medföra skada för Sveriges totalförsvar.

Ansökan ska därför bifallas.

Beslutande

Beslut i detta ärende har fattats av handläggare Helena Fyhr.

Om personuppgifter

Information om hur Lantmäteriet behandlar personuppgifter går att hitta på https://www.lantmateriet.se/personuppgifter eller genom att kontakta Kundcenter på telefonnummer 0771-63 63 63.