Institute for Computer Science VII
Robotics and Telematics

Bachelor's thesis

A Quantitative Evaluation of Feature Matching Algorithms for CubeSats

Bence Barthó

March 2023

First reviewer: Prof. Dr. Andreas Nüchter

Abstract

This work provides an overview of the potential applications and functionality of feature matching in Earth Observation (EO) with nanosatellites.

First, a meta-analysis of previous research into the area is performed to establish a general baseline.

Subsequently, a simulated CubeSat is prepared, with strict resource limitations, to approximate the satellites in use by the University of Würzburg.

The simulation also includes the acquisition of realistic Earth Observation (EO) images.

This set-up then evaluates commonplace OpenCV feature matching algorithms, especially regarding resource use and software performance. All algorithms examined were found to be compatible with the limited resources. Lightweight methods such as FLANN and RANSAC were used to approximate the solutions.

The conclusions reached here support the general consensus that SIFT remains the best choice for most applications, especially when robustness is vital. However, for different use cases, faster algorithms such as ORB and TEBLID are preferred.

Zusammenfassung

Diese Arbeit gibt einen Überblick über die potenziellen Anwendungen und Funktionen des Merkmalsabgleichs in der Erdbeobachtung mit Nanosatelliten.

Zuerst wird eine Meta-Analyse der bisherigen Forschung in diesem Bereich durchgeführt, um eine allgemeine Grundlage zu schaffen.

Danach wird ein simulierter CubeSat mit strikten Ressourcenbeschränkungen vorbereitet, um sich den von der Universität Würzburg verwendeten Satelliten anzunähern.

Die Simulation umfasst auch die Aufnahme von realistischen Erdbeobachtungsbildern.

Mit diesem Setup werden dann gängige OpenCV-Feature-Matching-Algorithmen evaluiert, insbesondere im Hinblick auf Ressourcenverbrauch und Softwareleistung. Alle untersuchten Algorithmen erwiesen sich als kompatibel mit den begrenzten Ressourcen. Leichtgewichtige Methoden wie FLANN und RANSAC wurden zur Lösungsannäherung verwendet.

Die hier gezogenen Schlussfolgerungen stützen den allgemeinen Konsens, dass SIFT für die meisten Anwendungen die beste Wahl bleibt, insbesondere wenn Robustheit entscheidend ist. Für verschiedene Anwendungsfälle werden jedoch schnellere Algorithmen wie ORB und TEBLID bevorzugt.

Contents

Acronyms

ADC Attitude Determination and Control. 37, 47, 50

AI Artificial Intelligence. 4

AKAZE Accelerated-KAZE. 8, 27, 40, 41, 46, 47

BRIEF Binary Robust Independent Elementary Features. viii, 8

BRISK Binary Robust Invariant Scalable Keypoints. 8, 40, 41, 46

CGI Computer-Generated Imagery. 3

CPU central processing unit. 10, 17, 27, 47

CV Computer Vision. 1, 17, 18, 50

EO Earth Observation. iii, 1, 4, 8, 14, 47, 50

FAST Features from Accelerated Segment Test. 8

FLANN Fast Library for Approximate Nearest Neighbors. iii, 10

FOSS free and open-source software. 4, 17

GPU graphics processing unit. 28, 30

IR infrared. 10

kNN k-Nearest Neighbours. 10

MASINT Measurement and signature intelligence. 50

MSAC M-estimator Sample Consensus. 18

OpenCV Open Source Computer Vision Library. 2, 4, 5, 18, 46

ORB	Oriented Binary Robust Independent Elementary Features (BRIEF). iii, 2, 8, 13, 27, 40, 41, 46, 47
OS	Operating System. 11
PROSAC	Progressive Sample Consensus. 18
radar	radio detection and ranging. 13
RAM	Random Access Memory. 10, 28, 47
RANSAC	Random Sample Consensus. iii, 18, 19, 32
SBC	single-board computer. 3, 50
SIFT	Scale Invariant Feature Transform. iii, 2, 8, 13, 14, 27, 38–41, 46
SIMD	Single Instruction, Multiple Data. 10
SoC	System on a Chip. 30
SURF	Speeded Up Robust Features. 2, 8, 13, 14, 27, 40, 46, 47
TEBLID	Triplet-based Efficient Binary Local Image Descriptor. iii, 9, 27, 38, 39, 41, 46, 47
UAV	unmanned aerial vehicle. 13
University of Würzburg	Julius-Maximilians-Universität Würzburg. iii, 1, 3, 10
VGA	Video Graphics Array (640×480). 21, 29
ZfT	Zentrum für Telematik e. V.. 3

Chapter 1

Background and Motivation

1.1 The Goal

With the growing capabilities of CubeSats dedicated to Earth observation applications, an increasing number of people are considering such missions for scientific or commercial uses. To make the best possible use of resources, a lightweight feature detection algorithm is needed that can run on low-power hardware.

Earth Observation (EO) refers to the practice of using technology to monitor and analyse the natural environment of the Earth to better understand our planet. EO provides information about the natural and man-made changes in society that help us to meet environmental challenges and plan for disasters. This work focuses on purely visual features extracted through Computer Vision (CV).

1.2 The Setup

The testing system is made up of a Raspberry Pi 3 Model B. The Raspberry Pi is a hobbyist microcomputer that has become ubiquitous as a platform for many various projects. It was chosen specifically for this work because the University of Würzburg operates a satellite using a Raspberry Pi Zero with similar specifications. This means that the findings will have direct use in the real world and a practical advantage.

To simulate an Earth Observation camera, a model globe was set up to be photographed with the official Raspberry Pi High Quality Camera module.

This globe is technologically advanced enough to rotate using solar power automatically. It uses satellite images taken by NASA of Earth, which more accurately simulate real-life conditions, making the test setup sufficiently similar [1].

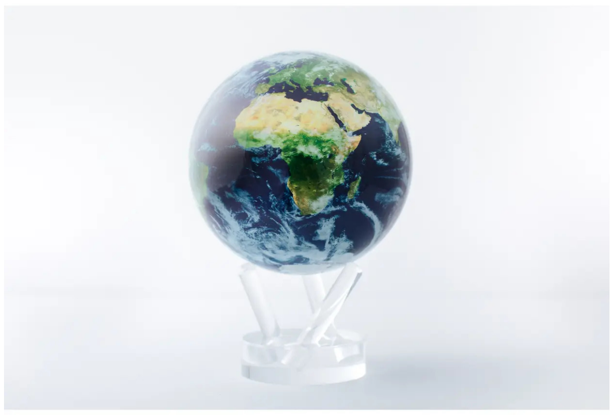


Figure 1.1: The MOVA Earth with Clouds Globe [2]

1.3 Software

The OpenCV library was used with C++ code to keep the codebase as lightweight as possible. This keeps the code small in memory usage, runs at a low hardware level, and is relatively portable. Well-known feature detection, extraction, and description algorithms include SURF, SIFT, ORB and many more [3]. Some of them have existed for decades. Some are cutting-edge. Recency, however, does not automatically mean better. Reliability and efficiency are more important than absolute accuracy in this use case. As satellite missions usually last for years, long-term planning is crucial. Robustness is a necessity.

1.4 Test Criteria

To compare the algorithms, a scoring system is needed directly. Quantifiable aspects include execution time, the overall number of features detected, the overall and peak memory usage, repeatability, the matching ratio with other algorithms, and the ease (time) of deployment. This work intends to compare as many facets as possible, to help different kinds of future missions.

1.5 CubeSats

California Polytechnic State University, San Luis Obispo, and Stanford University's Space Systems Development Lab created the CubeSat standard. The official Design Specification provides specific definitions [4]. As a subclass of nanosatellites, the mass of CubeSats generally falls into the 1 kg to 10 kg range. The standard 1 U unit has a dimension of 10 cm $\times$10 cm $\times$11.35 cm. As of August 2022, 1893 CubeSats had been launched. The industry is rapidly expanding [5]. The estimated number of upcoming launches is shown in Figure 1.2.

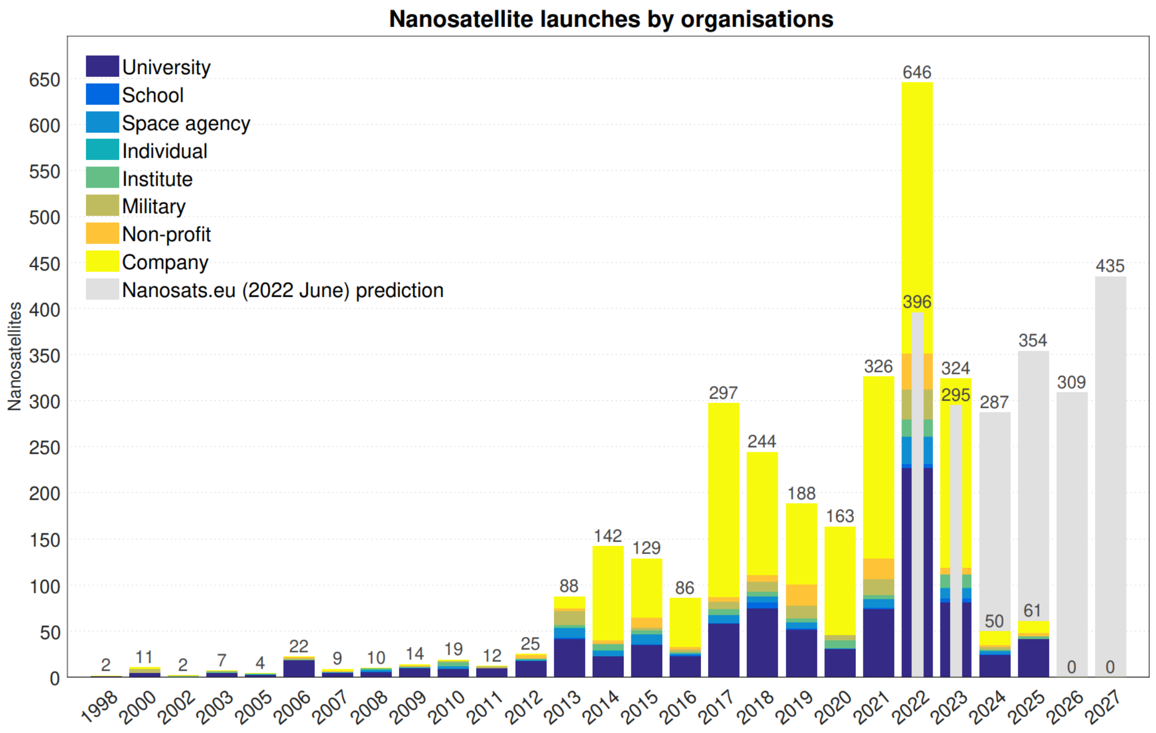


Figure 1.2: Nanosatellite launches per year [5]

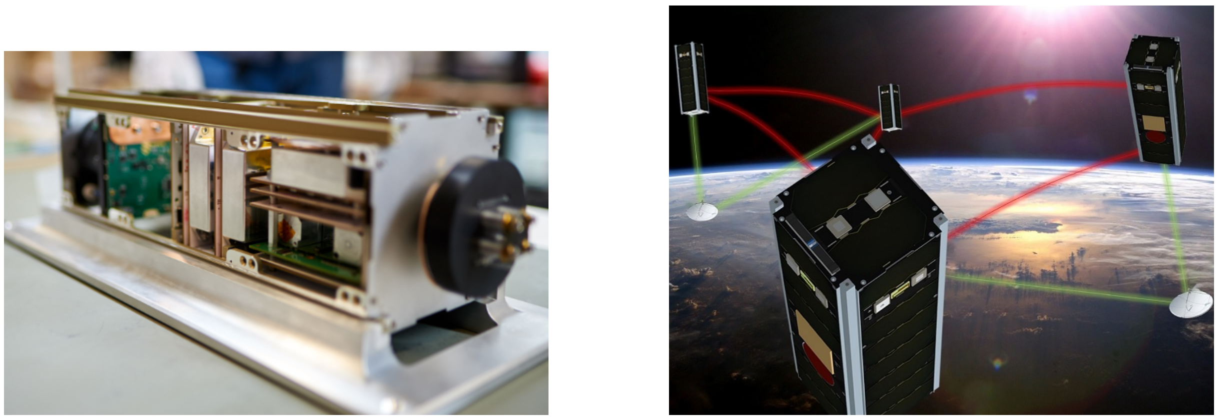
1.6 The Satellites of the University of Würzburg

The NetSat project is part of an ongoing cooperation between the Zentrum für Telematik e. V. (ZfT) and the Julius-Maximilians-Universität Würzburg (University of Würzburg) [6]. As part of the project, four CubeSats were launched in 2020. The mission aims to test the feasibility of a distributed constellation of CubeSats.

As each satellite weighs only 4 kg, the payload capability is minimal. On-board each CubeSat is a Raspberry Pi Zero equipped with its Camera Module. A Raspberry Pi is a widely adopted single-board computer (SBC).

The camera is capable of taking photos up to 12.3 MP in resolution; however, due to the limited bandwidth available, the actual broadcast resolution is much lower [7].

Figure 1.3a shows the internal hardware of a NetSat CubeSat, while figure 1.3b shows a Computer-Generated Imagery (CGI) rendering of a CubeSat formation.



(a) NetSat Hardware **(b)** NetSat Render

Figure 1.3: Images of NetSat [8, 9]

Figure 1.4 shows a typical example image, of what kind of imagery EO one expects from a CubeSat.

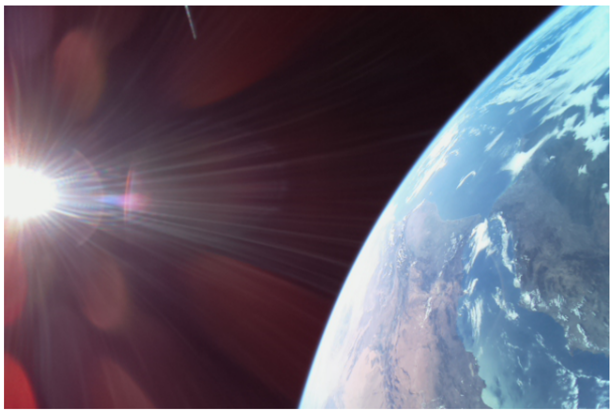


Figure 1.4: Image taken by NetSat-2 in November 2020 [8]

1.7 OpenCV

The Open Source Computer Vision Library (OpenCV) was launched over 20 years ago in 1999. It soon established itself as an industry standard. Designed from scratch to be free and open-source software (FOSS), it supports a wide variety of platforms and operating systems whilst retaining portability. Desktop platforms such as Windows, Linux, macOS, and FreeBSD and mobile platforms such as Android and iOS are all supported. The library has more than 2500 dedicated algorithms, from simple image processing to cutting-edge Artificial Intelligence (AI) applications.

Along with the open-source developers, the main contributor to the library is Intel®. These factors helped establish OpenCV with hobbyists and market-leading companies. Even if the intended platform differs, OpenCV is well documented, enabling straightforward migration.

Under the hood, OpenCV implements all algorithms using C and C++. However, it also offers bindings for other languages such as Python, Java and MATLAB. This lets the code be used by many more developers, with a more accessible syntax but maintains most of the high speed and efficiency of C/C++ code[10, 11].

Chapter 2

System Specification and Setup

2.1 Comparison of Algorithms

In order to validate this work's data, comparisons must be made with existing work, wherever possible. This approach measures the need for new results and areas to be improved.

Since the specific topic is a niche, a more general overview and extrapolation are necessary. Some authors use more advanced types of EO methods, utilising Deep Learning, which has a much higher resource cost.

As the lightweight property of the code is essential, these methods are generally outside the scope of this work. Although most authors compare a select few algorithms, this work intends to evaluate the entire spectrum. In general, the findings indicate that the repeatability and general robustness of the original SIFT and SURF algorithms remain unsurpassable [12]. However, more modern algorithms offer better rotation invariability, such as ORB and BRISK. In addition, these algorithms are open-source instead of proprietary. This helps to keep code affordable, accessible, and future-proof.

The SURF algorithm offers much higher speeds whilst maintaining robustness. However, this extra speed does add additional hardware requirements. Since our goal is not live transmission, longer processing times may be acceptable [13]. Previous work merely concentrated on either too diffuse comparisons or entirely different use cases. The goal is to examine and evaluate the algorithms available for the sole purpose of Earth Observation feature detection. Furthermore, algorithms must run on low-resource hardware. However, low speed is not necessarily a problem, as no real-time feature detection is required.

2.1.1 Algorithms in Detail

The Scale Invariant Feature Transform (SIFT) and Speeded Up Robust Features (SURF) algorithms are the oldest and most well known feature extraction algorithms. SIFT is more robust to geometric transformations and changes in scale, while SURF is faster and more efficient. Both use so-called float descriptors.

Oriented Binary Robust Independent Elementary Features (BRIEF) (ORB) is an open-source alternative to the traditional and patented SIFT and SURF algorithms. It is considered a fast and robust approach. It combines the Features from Accelerated Segment Test (FAST) corner detector with the Binary Robust Independent Elementary Features (BRIEF) descriptor and includes an orientation component for additional rotational invariance. The descriptor is fixed at 256 bits in size. ORB is, in general, very scale invariant.

Binary Robust Invariant Scalable Keypoints (BRISK) uses a more sophisticated and theoretically more robust detector and descriptor. However, these require more computations. BRISK is not designed to be strictly scale-invariant. BRISK includes a rotation-invariance step during the detection phase, while ORB only uses a rotation-aware descriptor that only accounts for small rotations. BRISK uses a configurable but larger descriptor than ORB. BRISK is considered to be highly viewpoint- and illumination-invariant.

KAZE and Accelerated-KAZE (AKAZE) are both unique by using a non-linear scale space for detection and description. AKAZE is an improved version of KAZE that is faster and more robust. AKAZE uses a binary descriptor, whereas KAZE uses a float. The descriptor of AKAZE is based on BRIEF, while that of KAZE is based on SURF.

Name	Release	Authors	Licence	Detection Type	Description Type	Stated Benefits
SIFT	2004	Lowe et al. [14]	Patented (expired 2020 [15])	DoG (blobs)	128D float	Highly invariant
SURF	2006	Bay et al. [16]	Patented (expires 2029 [17])	Hessian matrix (blobs)	64D float	Faster than SIFT
ORB	2011	Rublee et al. [18]	Open Source	FAST + Harris (corners)	Mod. BRIEF	Very fast
BRISK	2011	Leutenegger et al. [13]	Open Source	AGAST + FAST (corners)	DoG binary	Fast, more resource efficient
KAZE	2012	Alcantarilla et al. [19]	Open Source	Hessian matrix (blobs)	nonlinear float	Higher accuracy and distinctivity
AKAZE	2013	Alcantarilla et al. [19]	Open Source	Hessian matrix	Mod. Local Difference Binary	More efficient KAZE
TEBLID	2021	Suárez et al. [20]	Open Source	ORB (corners)	Box Average Difference Binary	Much lower computational cost

Table 2.1: Algorithm Comparison

Triplet-based Efficient Binary Local Image Descriptor (TEBLID) is a very recent descriptor algorithm. It relies on a different detection algorithm for its keypoints. The general recommendation is to use ORB. It uses the novel approach of extracting descriptors with texture and edge information, which should improve its effectiveness on low-contrast and weakly textured images.

2.1.2 Descriptor Types

The descriptors can be ordered into two major groups.

Float descriptors (SIFT, SURF, and KAZE) represent features using a vector of real-valued numbers. This leads to a high computational cost, with the benefit of retaining more information and thus the precision of the intensity values of the keypoints. This should offer higher accuracy and robustness with the trade-off of memory use and running time.

On the other hand, binary descriptors (ORB, BRISK, AKAZE, and TEBLID) represent the features with binary codes. This makes them more efficient to compute, but with the loss of precise information. This can make them unsuitable for images with high noise or low contrast.

2.1.3 Matching Methods

The following methods can be used to make matching more efficient for limited-resource applications:

Fast Library for Approximate Nearest Neighbors (FLANN) is a library to approximate k-Nearest Neighbours (kNN) search. It builds a hierarchical structure to search for neighbours more efficiently.

The L2 norm, otherwise known as the Euclidean norm, tends to be used for float descriptors.

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{2.1}$$

It calculates the distance of two vectors by taking the square root of the sums.

Binary descriptors offer a much more efficient way to determine neighbours. The Hamming distance is the number of bit positions in which the vectors differ. This is very fast, even for large-scale data, because it can be efficiently computed solely by bitwise operations. However, it is generally not as accurate as the Euclidean distance.

2.2 Exact Setup

2.2.1 Raspberry Pi

The hardware for the experiment was a Raspberry Pi 3 Model B. However, to account for the reduced resources of the Raspberry Pi Zero, the CPU was artificially limited to a maximum of 1 GHz on a single thread, and the RAM to a total of 512MB (with 128MB allocated to the GPU). This is the standard configuration of the camera modules onboard the satellites of the University of Würzburg. Simulating the platform as closely as possible is one of the main goals of this work.

An unavoidable difference lies in the CPU architectures. The Pi Zero has a 32-bit ARM11 processor using the ARMv6 architecture. The Pi 3 has a 64-bit processor based on ARMv8-A. Even accounting for the differences in cores, the Pi Zero lacks features such as out-of-order executions, hardware virtualisation, and advanced Single Instruction, Multiple Data (SIMD) instructions. These factors may lead to differences, but should be negligible.

In any case, the latest Pi Zero 2 W model has a CPU identical to the Pi 3. So this issue is irrelevant to any future missions using the latest hardware.

2.2.2 Raspberry Pi Camera

The Raspberry Pi High Quality Camera V1.0 camera module has a Sony IMX477R stacked back-illuminated sensor with a maximum resolution of 12.3 megapixels. It has a pixel size of 1.55 µm x 1.55 µm. A removable infrared (IR) filter is provided, which makes it suitable for advanced remote sensing operations. The sensor size is 1/2.3” (7.9mm) [7].

The lens is a CGL PT3611614M10MP. It is a telephoto lens with an aperture of f1.4 and a focal length of 16mm. With the 5.6 crop factor of the sensor, that is, the full-frame equivalent of an 89.6mm lens with an f7.8 aperture. However, it is important to note that the depth of field and light-gathering capability are not directly comparable.

2.2.3 MOVA Globe

The simulated Earth was created using a model globe. The globe rotates automatically using hidden solar cells and magnets to exploit the Earth's magnetic torque. The globe features NASA satellite images, which even include clouds [1].

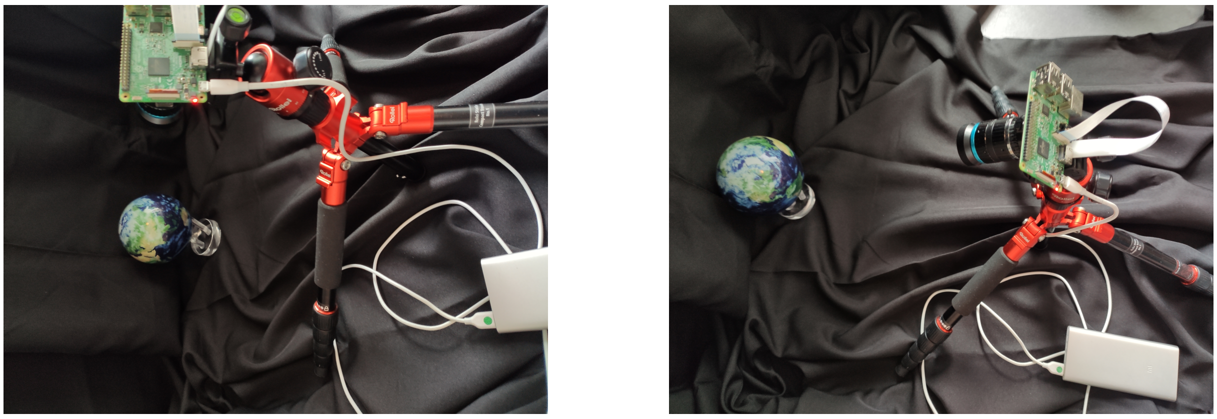


Figure 2.1: Experiment Setup

2.2.4 Software

The Operating System (OS) was Raspbian 10 (buster) for ease of use. This is a Debian-based OS specifically for the Raspberry Pi. The installation was modified to be as lightweight as possible by removing unnecessary packages.

The installed version of OpenCV was 4.1. The extra modules from the `contrib` library were also installed.

Chapter 3

Prior External Results and Baseline

3.1 Earth Observation Specific - Prior Results

There was already some research done on the topic of this work. However, a strict reliance on low-power hardware was rarely considered. Therefore, these results help to compare and validate the results, simultaneously showcasing the novelty and usefulness of the current work.

The most recent work from 2021 by Moghimi et al. concludes that SURF and SIFT are more accurate on average than the corner-detection-based ORB; however, they are slower to compute. The trade-off between accuracy and computing time must be considered. The authors also used specialised hardware with high computing capacities (a desktop computer with Intel®Core$^{\text{TM}}$ i7-3770 CPU@3.40 GHz, 12.00 GB RAM, running Windows). This means that any concrete quantitative comparisons are of limited use to the present application. The authors used publicly available images from previous satellite missions, ensuring repeatability. However, they implemented the algorithms for multispectrum analysis, which is outside the scope of this work [21].

Also, in 2021, Meng et al. evaluated various algorithms for multispectrum unmanned aerial vehicle (UAV) image registrations. They also used freely available satellite footage, thus making their findings relevant to this work. Their results conclude that SIFT performs better than ORB or SURF. This leads them to the conclusion that histograms of features based on local gradient directions are generally more robust than features based on local gradient magnitude or pixel value in multispectral UAV images. However, they noted that SIFT starts to perform poorly on soft textures. Their experiments used a high-end GPU and CPU (NVIDIA GeForce GTX 2080Ti, Intel i7 8700), making the actual quantitative results less relevant to the purpose of this work [22].

In 2016, Tahoun et al. compared many algorithms for matching optical and radio detection and ranging (radar) images. Their generalised conclusion was that ORB ran by far the fastest. However, this was due to finding the least number of keypoints, which led to a lower image matching accuracy rate. SIFT, SURF and FAST all performed similarly on optical images, whilst SIFT was the best suited for use with radar images [23].

Figure 3.1 shows an example of image registration between two satellite photos. The green lines connect the corresponding keypoints.

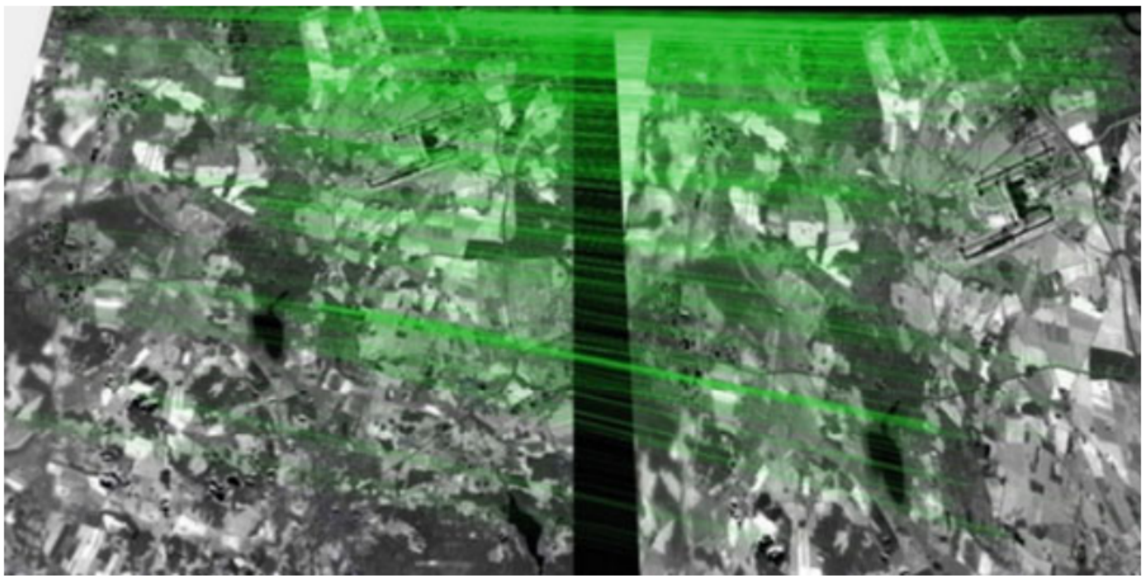


Figure 3.1: Image registration between two satellite images after translation by Tahoun et al. [23]

3.2 General Prior Results

Most of the available literature that compares feature detection algorithms is not related to EO or remote sensing applications. However, even seemingly unrelated topics help establish a base truth for comparisons. In addition, various forms of image manipulation techniques may benefit this work, so a detailed investigation is recommended.

For example, Hore et al. utilised the SURF algorithm for real-time detection, encryption, and decryption of dactylologic symbols (i.e., finger-spelling). This is shown in Figure 3.2. Their requirements aimed for an algorithm suited for robust, fast, and efficient work. Signs had to be detected as they were being shown in real time, and since their work used a secret language, they also had to be encrypted and decrypted simultaneously [24].

A more abstract and mathematical approach was the work of Bhaumik et al. They aimed to develop a better video compression process using feature detection to find redundant image sections in videos. Their comparison metrics and results may provide value for this work [25]. See Figure 3.3.

Khan et al. measured the performance of the SIFT and SURF algorithms using a public dataset that contains indoor and outdoor images of 2500 objects. This open dataset allows this work to serve as a benchmark for any other research group. The authors applied artificial deformations to the image to specifically examine rotation, illumination, and noise invariance. The most important conclusions were that SURF was on par with SIFT, except for scaling and significant blurring. They further improved the traditional 128D SIFT algorithm by creating a 64D descriptor. This proposal allowed similar accuracy while trebling the image matching speed and halving the memory requirements [26]. Figure 3.4 depicts the matching accuracy of their approach. This custom approach is worth considering in the current work. However, the

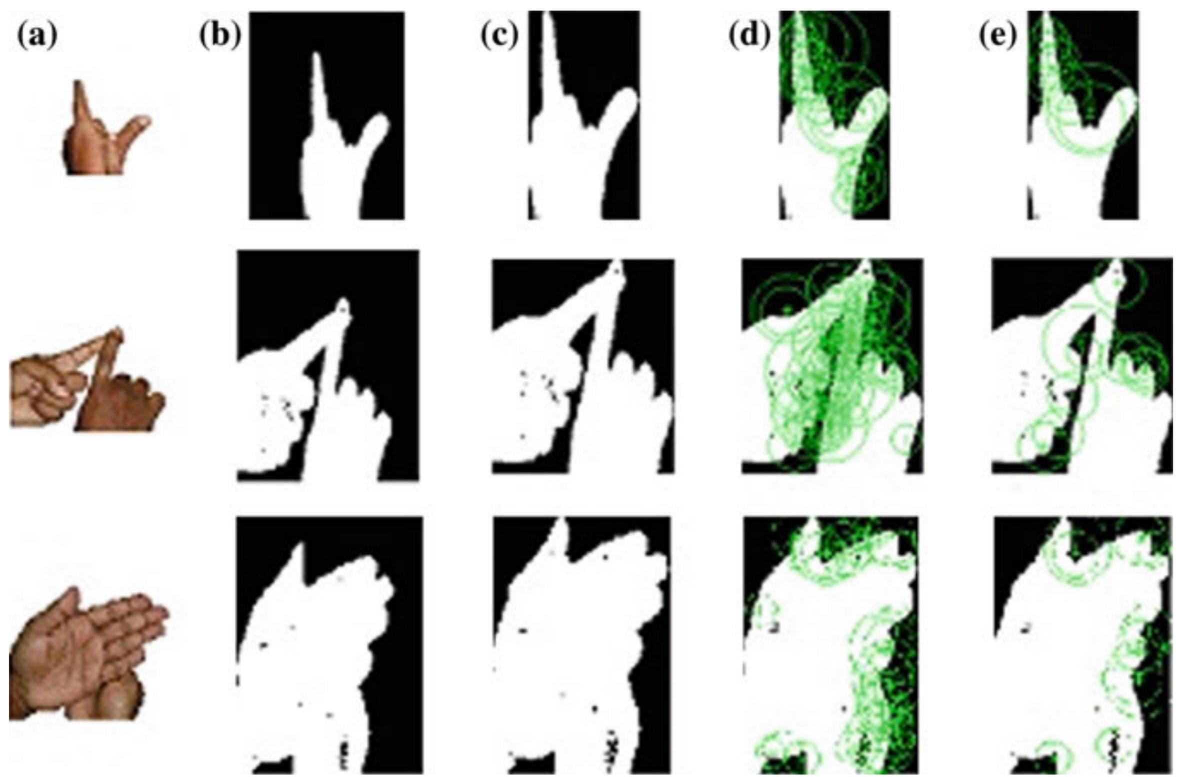


Figure 3.2: Image manipulation and feature descriptors to decode finger-spelling by Hore et al. [24]

benefits must be weighed against the potential ease-of-use drawbacks.

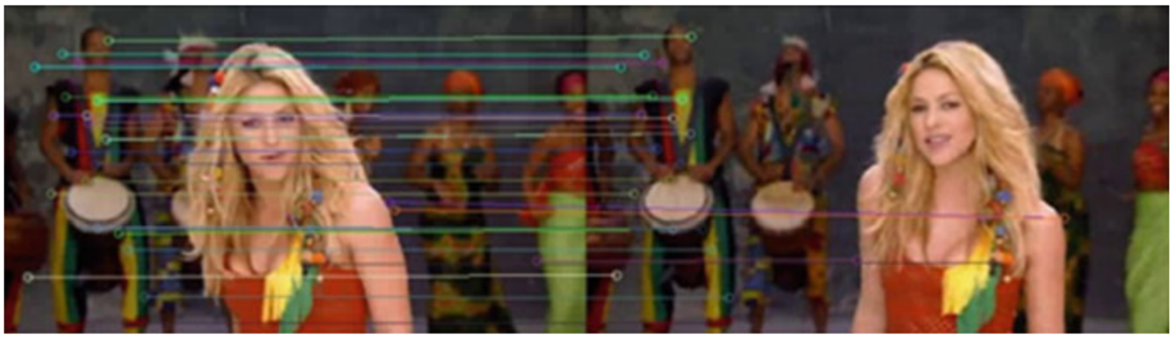


Figure 3.3: Using advanced feature descriptors to find correspondences between single video frames by Bhaumik et al. [25]

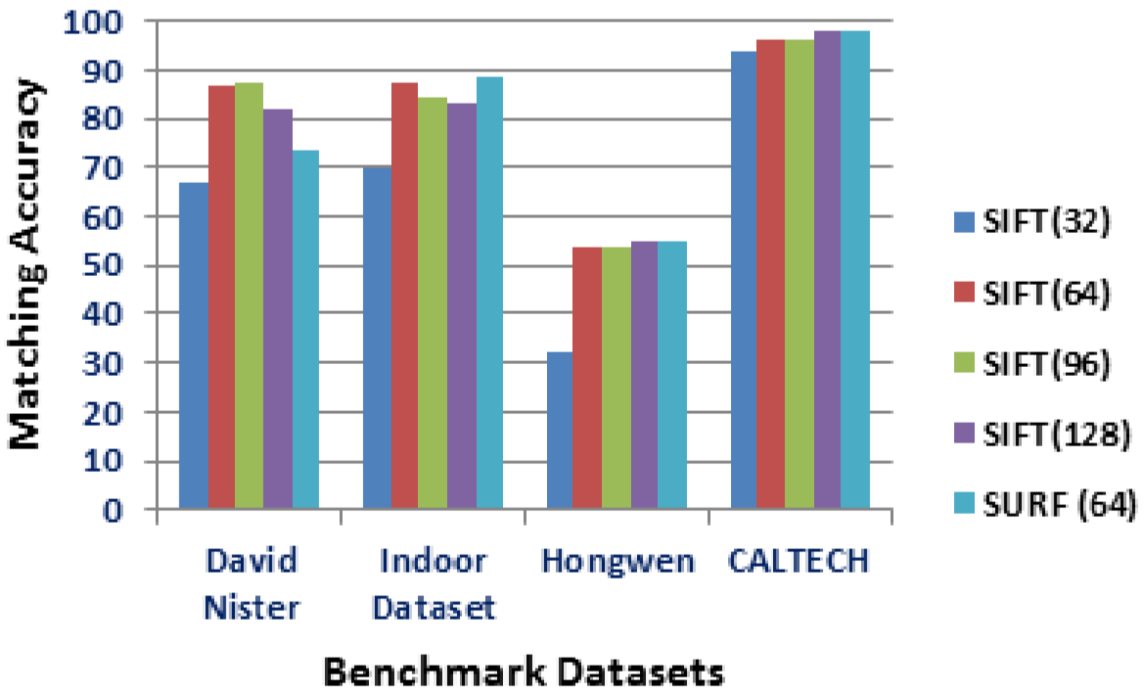


Figure 3.4: A diagram from the work of Khan et al. comparing the matching accuracy of the default SIFT and SURF algorithms with their novel proposal [26]

Chapter 4

Experiment Criteria

The following criteria were chosen to evaluate the algorithms. Each aspect was deliberately selected to measure both the usefulness of the current specific application and to enable comparisons with the broader literature. The resource use of each algorithm is vital due to the severely limited hardware available. The temperature of the CPU is also essential to consider, because the heat generated may negatively influence the other systems onboard an actual satellite. The CV parameters were selected to evaluate the robustness of an individual algorithm and to allow direct comparison of the algorithms. These metrics have also been widely studied in previous work; therefore, this work can be analysed compared to the broader literature.

4.1 Runtime

Runtime measurements were done using the same hardware and images for each algorithm to ensure repeatability. These values are essential mainly in relative terms as the specifics will change from mission to mission. Only the essential background processes were left alive during the measurements. The installation was kept as bare-bones as possible, e.g. with no graphical interface.

The runtime metric is also an essential contributor to the battery drain of a satellite. An algorithm cannot run for an arbitrary amount of time, as the batteries may deplete too soon.

4.2 Resource Use

A suitable lightweight resource monitoring method was needed to be found. Widely-adopted and platform-agnostic Linux tools were preferred for compatibility and future-proof reasons. The `nmon` package is over 15 years old, widely used, in active development and FOSS [27]. The package allows for monitoring the resources of a single process and generating easy-to-understand graphs for visualisation and assessment. The average and peak loads for the CPU and RAM were examined. In addition, the temperatures were also taken. However, these should only be considered relatively. The experiment was not executed in a controlled environment (i.e. a vacuum chamber). The heat balance of a satellite is a vital factor. During orbital phases in the shadow, heat must be generated. However, during sunlit phases, extra heat must be avoided. Excess heat is difficult to eliminate since the only method is radiative heat dissipation.

4.3 Feature Matching Metrics

4.3.1 Number of Features Found

The number of features is a straightforward metric. However, it mainly serves as a comparison, not as a direct evaluation.

4.3.2 Good Matches

Good matches were evaluated using the Lowe's ratio test, a lightweight and widespread pragmatic method [28]. In simple terms, the test posits that the two closest matches are unlikely to share the same descriptor distance. If the distance ratio is above a threshold, the matches are removed. This threshold is usually around 0.7-0.8.

4.3.3 Inliers

The Random Sample Consensus (RANSAC) algorithm distinguishes between inliers and good matches. This form of outlier detection is widely used in the Computer Vision sector [29]. One of the main advantages of RANSAC is its ability to handle a large number of outliers in the data, even if noise and outliers comprise more than 50 % of the data. It does this by randomly selecting a subset of the data points, fitting a model to that subset, and testing the model on the remaining data points. This process is repeated for multiple epochs and the best model is selected as the one with the highest number of inliers.

Additionally, in this use case, RANSAC has a further benefit. It is a computationally efficient algorithm, making it more suitable for use than, e.g. Progressive Sample Consensus (PROSAC) or M-estimator Sample Consensus (MSAC). Furthermore, it is also deterministic, which allows for repeatable comparisons without stochastic changes.

4.3.4 Precision

The precision metric is the ratio of true positives to total positives [30].

$$precision = \frac{\#\ True\ Positives}{\#\ True\ Positives\ +\ \#\ False\ Positives} \tag{4.1}$$

4.3.5 Homography

OpenCV offers the `findHomography()` method, which gives the perspective transform H between the source and destination planes:

$$s_i \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{4.2}$$

The method minimises the back-projection error using RANSAC:

$$\sum_i \left(x'_i - \frac{h_{11}x_i + h_{12}y_i + h_{13}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 + \left(y'_i - \frac{h_{21}x_i + h_{22}y_i + h_{23}}{h_{31}x_i + h_{32}y_i + h_{33}} \right)^2 \tag{4.3}$$

The perspective change can be made by applying the `perspectiveTransform()` to a sparse set of points. This is achieved by matrix multiplication of H with the input points (x, y) resulting in the output points (x', y') [31].

4.3.6 *k*-means clustering

k-means clustering is an unsupervised machine learning algorithm that groups a set of points into clusters based on their similarity. In the current work, 2D coordinate pairs were the inputs and outputs. In short, the algorithm randomly initialises k cluster centroids. Then it assigns each point to the closest centroid (e.g. using Euclidean distances). Afterwards, it recalculates the centroid using the mean of the assigned points. Then these steps are iterated until the centroids reach the optimal position. Clusters are identified by the points in the proximity of each centroid [32].

Chapter 5

Implementation

5.1 Input Images

The images visible in figure 5.1 were used to perform the measurements. The two images were taken three seconds apart, from the same position, using the built-in `raspistill` command. The images were captured directly in Video Graphics Array (640×480) (VGA) resolution. The lighting conditions were the same, and thus the rotation of the globe can be considered constant. The images show the entire globe, filling a large part of the frame while leaving the monochrome black background visible.

The distance of the camera was as close as the focal length allowed. The chosen aperture was f5.6, a fair compromise between the depth of focus and the exposure. The camera was positioned slightly above the globe, pointing downwards, thus capturing more of the northern hemisphere.

The images show a variety of landscapes: large monochrome ocean expanses, sharp continent outlines, clouds, and low-contrast and weakly textured Greenland and the Arctic. Thus these images are representative of what conditions might arise. However, varying illuminations were not tested.

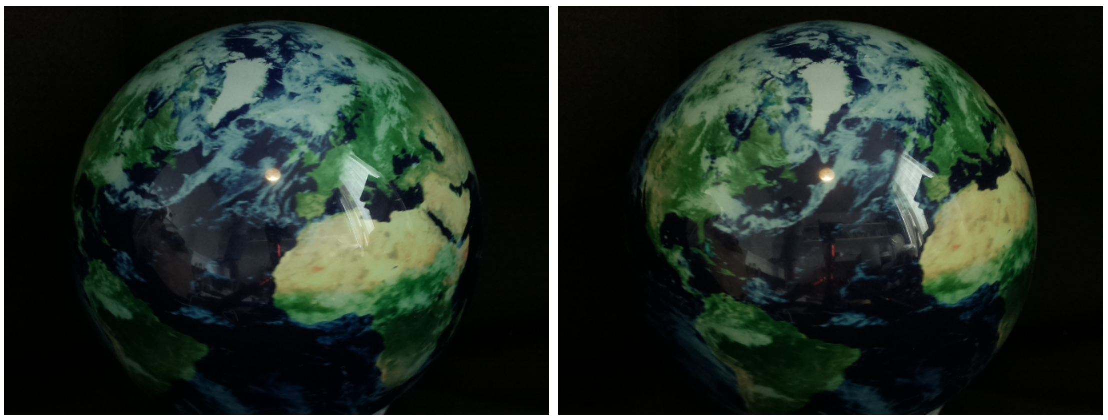


Figure 5.1: The two input images

5.2 Applying the SIFT algorithm

The following code snippet shows the implementation of the SIFT algorithm. The code was kept as close to the default settings as possible for all algorithms (e.g. 400 for the minimum Hessian in SURF, and maximum 1000 matches for ORB). This is done to simplify repeatability for any future work. Each algorithm's allocated objects were explicitly deleted, to prevent memory leaks and ensure better resource usage calculations. The standard implementation of Lowe's ratio test is also included. A standard value of 0.7 was used for the threshold.

```
void applySift(Mat src, Mat comparison, std::ofstream &output)
{
    // detect and compute keypoints
    Ptr<SIFT> detector = SIFT::create();
    std::vector<KeyPoint> keypoints, keypoints2;
    Mat descriptors, descriptors2;
    detector->detectAndCompute(src, noArray(), keypoints,
   descriptors);
    detector->detectAndCompute(comparison, noArray(), keypoints2,
    descriptors2);

    // match descriptors using flann
    Ptr<DescriptorMatcher> matcher = DescriptorMatcher::create(
   DescriptorMatcher::FLANNBASED);
    std::vector<std::vector<DMatch>> knn_matches;
    matcher->knnMatch(descriptors, descriptors2, knn_matches, 2);

    // filter matches using Lowe's ratio test
    const float ratio_thresh = 0.7f;
    std::vector<DMatch> good_matches;
    for (size_t i = 0; i < knn_matches.size(); i++)
    {
        if (knn_matches[i][0].distance < ratio_thresh *
   knn_matches[i][1].distance)
        {
            good_matches.push_back(knn_matches[i][0]);
        }
    }

    // show matches
    Mat img_matches;
    drawMatches(src, keypoints, comparison, keypoints2,
   good_matches, img_matches, Scalar::all(-1), Scalar::all(-1),
   std::vector<char>(), DrawMatchesFlags::NOT_DRAW_SINGLE_POINTS);
    putText(img_matches, "SIFT Matches", Point(10, 30),
   FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 255), 2);

    // save matches
    imwrite("SIFT_Matches.jpg", img_matches);

    // clear memory
    keypoints.clear();
    keypoints2.clear();
    descriptors.release();
    descriptors2.release();
    knn_matches.clear();
    good_matches.clear();
    img_matches.release();
}
```

Listing 5.1: Applying SIFT (C++)

5.3 Finding Inliers using RANSAC to Estimate the Homography

The application of RANSAC was kept as simple as possible, by relying on the built-in OpenCV method.

```

void findInliers(std::vector<DMatch> good_matches, std::vector<
   KeyPoint> keypoints2, std::vector<KeyPoint> keypoints, std::
   ofstream &output)
{
    std::vector<Point2f> obj;
    std::vector<Point2f> scene;

    // separate keypoints
    for (DMatch match : good_matches)
    {
        obj.push_back(keypoints2[match.queryIdx].pt);
        scene.push_back(keypoints[match.trainIdx].pt);
    }

    Mat inliers;

    // apply RANSAC and count inliers
    Mat H = findHomography(obj, scene, RANSAC, 3, inliers);
}
```

Listing 5.2: Finding Inliers (C++)

5.4 Measuring the Exact Execution Time

The runtime was measured from inside the code. This solution is dependent on a Linux based platform, however, it is easily adaptable for any other platform. The time was measured using the `chrono` library with the system time. This allowed better comparison for analysing the resource usage with different tools.

```
int64_t getExecutionTimes(std::chrono::time_point<std::chrono::
   high_resolution_clock> t1,
                              std::chrono::time_point<std::chrono::
   high_resolution_clock> t2){
    using namespace std::chrono;

    // get execution time in milliseconds
    auto duration = duration_cast<milliseconds>(t2 - t1).count();
    cout << "Execution time: " << duration << "ms" << endl;

    // print start and end time in HH:MM:SS:MMM format

    auto start = system_clock::to_time_t(t1);
    auto end = system_clock::to_time_t(t2);

    auto startms = duration_cast<milliseconds>(t1.
   time_since_epoch()).count();
    auto endms = duration_cast<milliseconds>(t2.time_since_epoch
   ()).count();

    std::tm broken_start = *std::localtime(&start);
    std::tm broken_end = *std::localtime(&end);

    std::cout << "start time: " << std::put_time(&broken_start, "
   %H:%M:%S") << ":" << startms % 1000 << std::endl;
    std::cout << "end time: " << std::put_time(&broken_end, "%H:%
   M:%S") << ":" << endms % 1000 << std::endl;

    return duration;
}
```

Listing 5.3: Measuring Execution Time (C++)

5.5 Clustering and Drawing Homography Matrices

The *k*-means clustering was also kept as simple as possible. The standard default parameters were used with the in-built OpenCV method.

```
void generateImageWithTransformations(std::string srcpath, std::
    string comparisonpath)
{
     // apply all homography matrices to a single 2d point using
    perspectiveTransform
     std::vector<Point2f> srcPoints;
     Point2f srcPoint(340, 176);
     srcPoints.push_back(srcPoint);

     // create a vector of dstPoints
     std::vector<std::vector<Point2f>> dstPointsVector;

     // apply each homography matrix to srcPoints
     for (int i = 0; i < homographies.size(); i++){
         std::vector<Point2f> dstPoints;
         perspectiveTransform(srcPoints, dstPoints, homographies[i
    ]);
         dstPointsVector.push_back(dstPoints);
     }

     // add each dstPoint to a Mat
     Mat dstPointsMat(dstPointsVector.size(), 2, CV_32F);
     for (int i = 0; i < dstPointsVector.size(); i++){
         dstPointsMat.at<float>(i, 0) = dstPointsVector[i][0].x;
         dstPointsMat.at<float>(i, 1) = dstPointsVector[i][0].y;
     }

     // use kmean to cluster points into 3 clusters
     Mat labels, centers;
     kmeans(dstPointsMat, 3, labels, TermCriteria(TermCriteria::
    EPS+TermCriteria::COUNT, 10, 1.0), 3, KMEANS_PP_CENTERS,
    centers);

     // display source image with srcPoint
     Mat srcWithPoint = imread(srcpath, IMREAD_COLOR);
     circle(srcWithPoint, srcPoints[0], 5, Scalar(0, 0, 255), -1);

     // display comparison image with dstPoints
     Mat comparisonWithPoints = imread(comparisonpath,
   IMREAD_COLOR);
    for (int i = 0; i < dstPointsVector.size(); i++){
        circle(comparisonWithPoints, dstPointsVector[i][0], 5,
   Scalar(0, 255, 0), -1);
    }

    // print cluster centers to comparison image
    for (int i = 0; i < centers.rows; i++){
        circle(comparisonWithPoints, Point2f(centers.at<float>(i,
    0), centers.at<float>(i, 1)), 5, Scalar(255, 0, 0), -1);
    }

    // print each dstPoint number and algorithm as text to the
   comparison image
    for (int i = 0; i < dstPointsVector.size(); i++){
        putText(comparisonWithPoints, std::to_string(i),
   dstPointsVector[i][0], FONT_HERSHEY_SIMPLEX, 1, Scalar(255,
   255, 255), 2);
    }

    // put the two images side by side
    Mat combined;
    hconcat(srcWithPoint, comparisonWithPoints, combined);

    // add text to the image
    putText(combined, "Input Point", Point(10, 30),
   FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 0, 255), 2);
    putText(combined, "Output Cluster", Point(10 + srcWithPoint.
   cols, 30), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 255, 0), 2);

    // save combined image
    imwrite("combined.jpg", combined);
}
```

Listing 5.4: Clustering and Drawing Homography Matrices (C++)

Chapter 6

Results

6.1 Resource Use

Table 6.1 gathers the stochastic data, i.e., the hardware resource usage. The results are as expected. ORB and TEBLID took the shortest times and were the least resource intensive. SURF was faster than SIFT, but not by a large margin. AKAZE was significantly faster and more efficient than its related algorithm, KAZE.

Name	Average Run Time (ms)	Average RAM (MB)	Maximum RAM (MB)	Average CPU (%)	Maximum CPU (%)	Average Temp. (°C)
SIFT	3982	175	219	78	95	55
SURF	3749	158	165	88	99	54
ORB	526	154	154	20	33	53
BRISK	3122	218	246	92	100	54
AKAZE	2699	262	271	72	98	54
KAZE	13386	308	326	86	100	57
TEBLID	1329	173	178	64	89	53

Table 6.1: Resource Use

Figure 6.1 visualises the average and maximum CPU utilisation in percentage form. The CPU had a maximum available frequency of 1000 MHz on a single core.

Except for ORB, all algorithms had a high CPU usage. This is expected behaviour and is not a problem. The CPU capacity is there to be used.

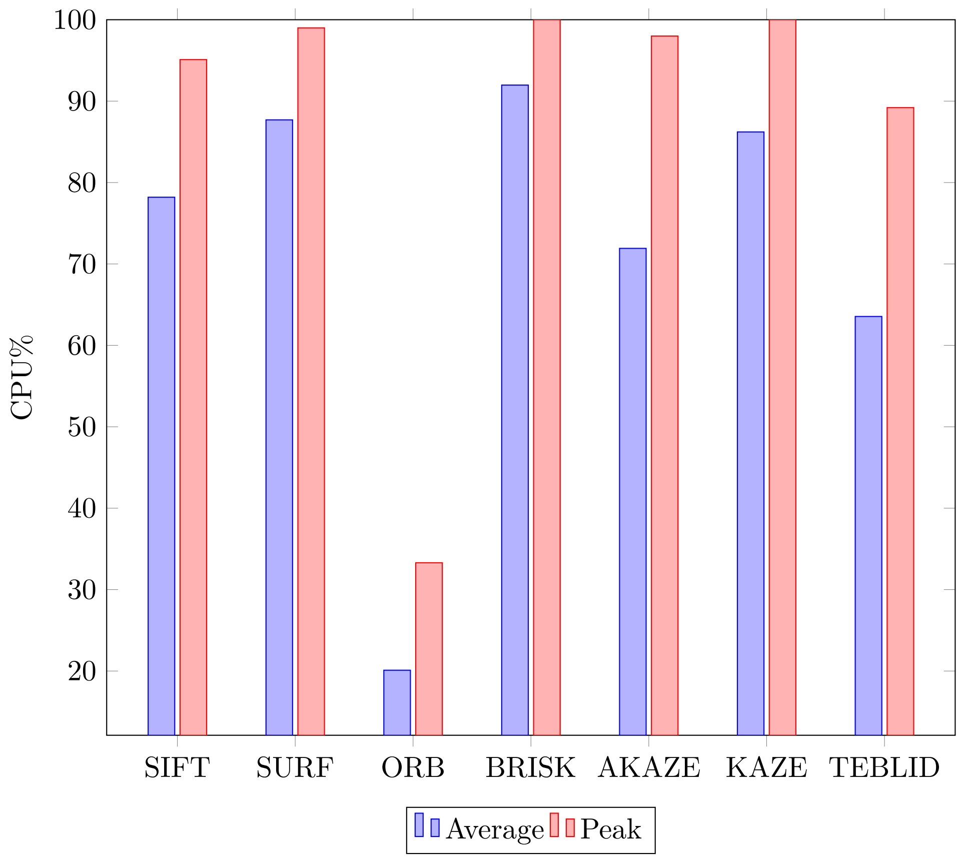


Figure 6.1: Average and Peak CPU Utilisation

Figure 6.2 visualises the average and maximum RAM utilisation in MBs. The RAM size was artificially limited to 512 MB, with 128 MB dedicated to the GPU. Furthermore, because the experiment was carried out on a platform as close to *in situ* as possible, the system processes used 40-50 MB of RAM in the background. Therefore, around 330-345 MB were free at any given time.

The maximum available RAM is marked with a red line and the average unloaded RAM usage is marked with a black line.

It is clearly visible that the available RAM was more than enough for our purposes. More RAM would only be necessary if examining images with higher resolutions. For example, a FullHD (1920 $\times$1080) resolution has 6.75 times more pixels than the VGA resolution used here.

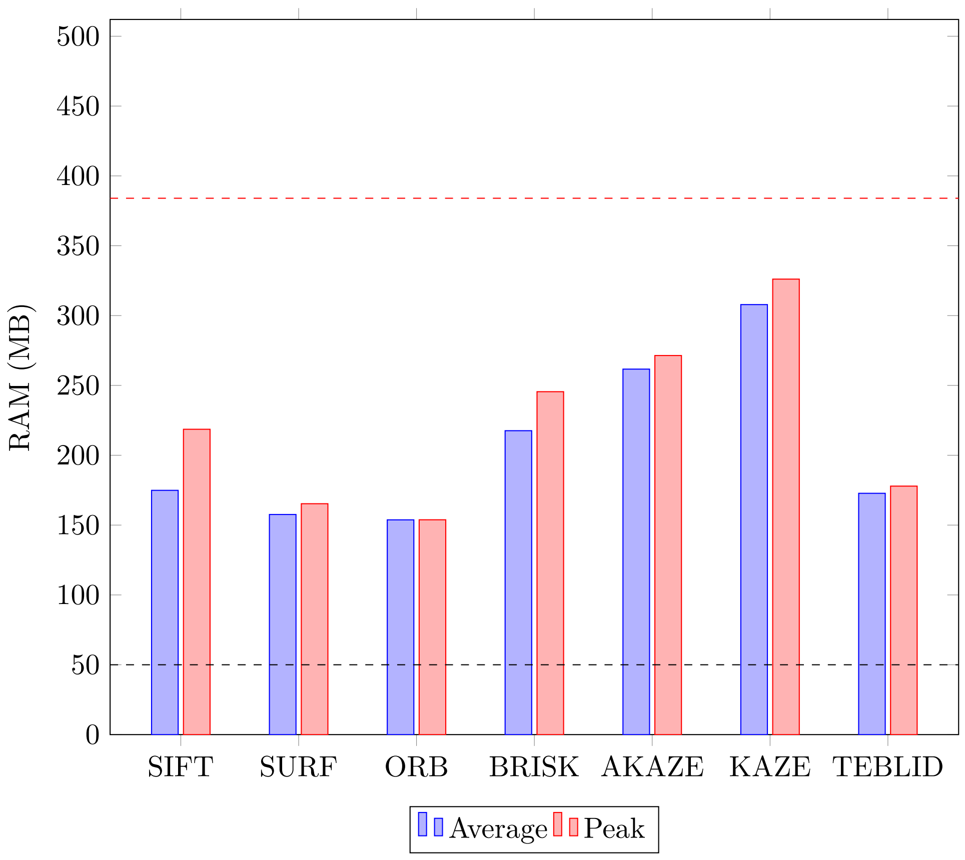


Figure 6.2: Average and Peak RAM use

Figure 6.3 shows the overall run time of the algorithms. The measurements were performed ten times on two VGA resolution images.

The deviations in the measurements were not large, thus the boxplots are thin and have no visible whiskers. This is because the runtime is generally deterministic. The slight variations may be due to other system processes interrupting.

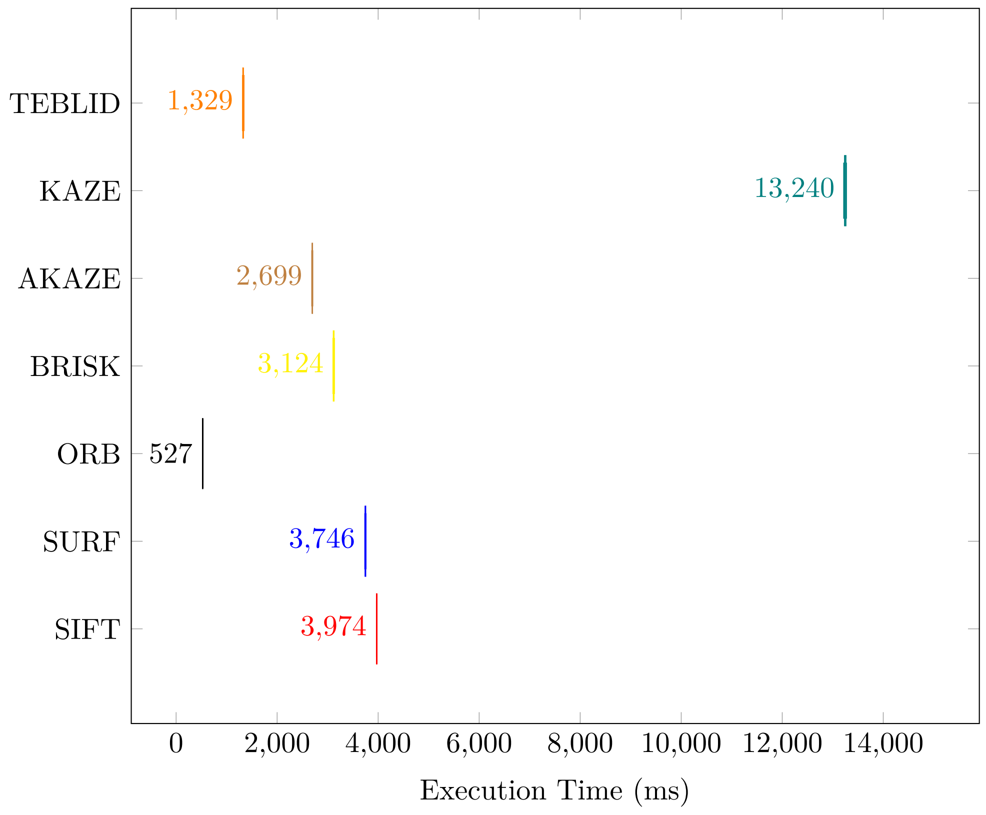


Figure 6.3: Box Plot of execution time ($n = 10$)

Figure 6.4 shows the reached temperatures of the Raspberry Pi System on a Chip (SoC) while running the algorithms. Ten measurements were made. The box plot shows the median values, with the standard deviations represented as whiskers. The `vcgencmd measure_temp` command was used. `vcgencmd` communicates directly with the GPU, to provide instantaneous and accurate temperature readings.

The temperature readings directly correlate with the resource usage and runtimes, i.e. the more computationally difficult algorithms heated the system up more.

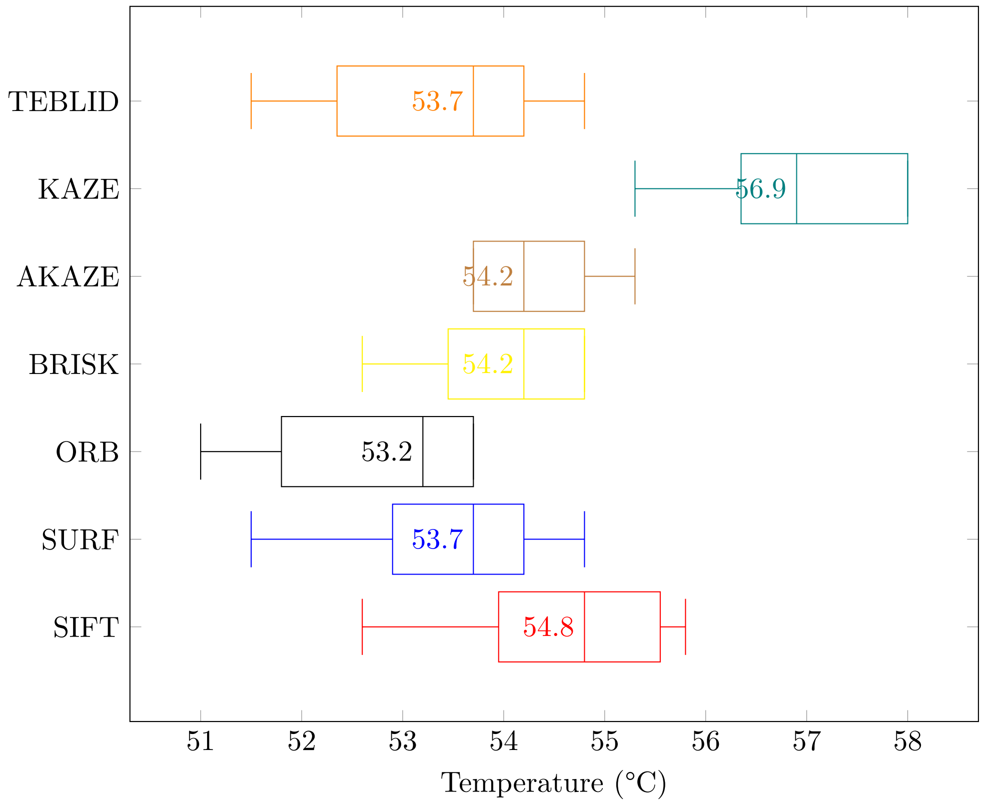


Figure 6.4: Box Plot of temperatures ($n = 10$)

6.2 Software Metrics

Table 6.2 summarises the software results. Due to their deterministic property, multiple measurements were not warranted.

The total number of keypoints is not a very important metric on its own. It is simply useful to see how algorithms generally behave. The number of keypoints of ORB and TEBLID was manually limited to the thousand best. This is a standard procedure and recommended for the TEBLID descriptor.

Name	Total Keypoints	Total Matches	Good Matches	Inliers
SIFT	2180	1102	202	8
SURF	1557	761	82	6
ORB	2000	1000	31	5
BRISK	2403	1171	75	7
AKAZE	1427	707	59	7
KAZE	1468	723	82	7
TEBLID	2000	1000	17	5

Table 6.2: Software Metrics

Table 6.3 shows the precision of each algorithm.

Precision is a classification metric that can be measured without available ground truth. This experiment is such a case. Usually, to generate ground truths, images are synthetically manipulated. This allows one to exactly measure the true and false positives and negatives. For CubeSat missions, this is not available, so relative comparisons are used.

The results of the fast algorithms are disappointing. TEBLID and ORB had the lowest Good Match precision. However, both achieved astonishingly high results in the Inliers-to-Good Match ratio. This suggests that their results are highly compatible with RANSAC and thus the homography matrices.

Name	Good Matches / Total Matches	Inliers / Good Matches	Inliers / Total Matches
SIFT	18%	4%	0.73%
SURF	11%	7%	0.79%
ORB	3%	16%	0.50%
BRISK	6%	9%	0.60%
AKAZE	8%	12%	0.99%
KAZE	11%	9%	0.97%
TEBLID	2%	29%	0.50%

Table 6.3: Precision

Figure 6.5 shows the number of matches found by each algorithm.

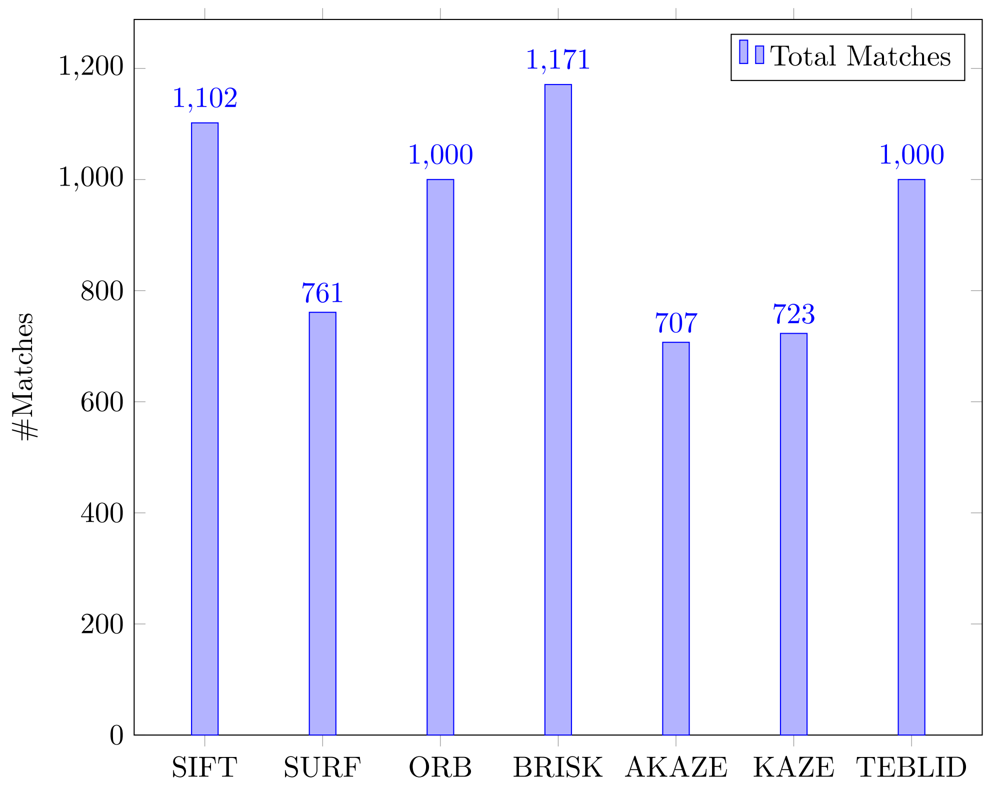


Figure 6.5: Overall number of matches found by the algorithms.

Figure 6.6 shows the number of good matches, that is, after applying Lowe's ratio test to the matches. This step is detailed in Section 4.3.2. The relevant code snippet is included in Section 5.2.

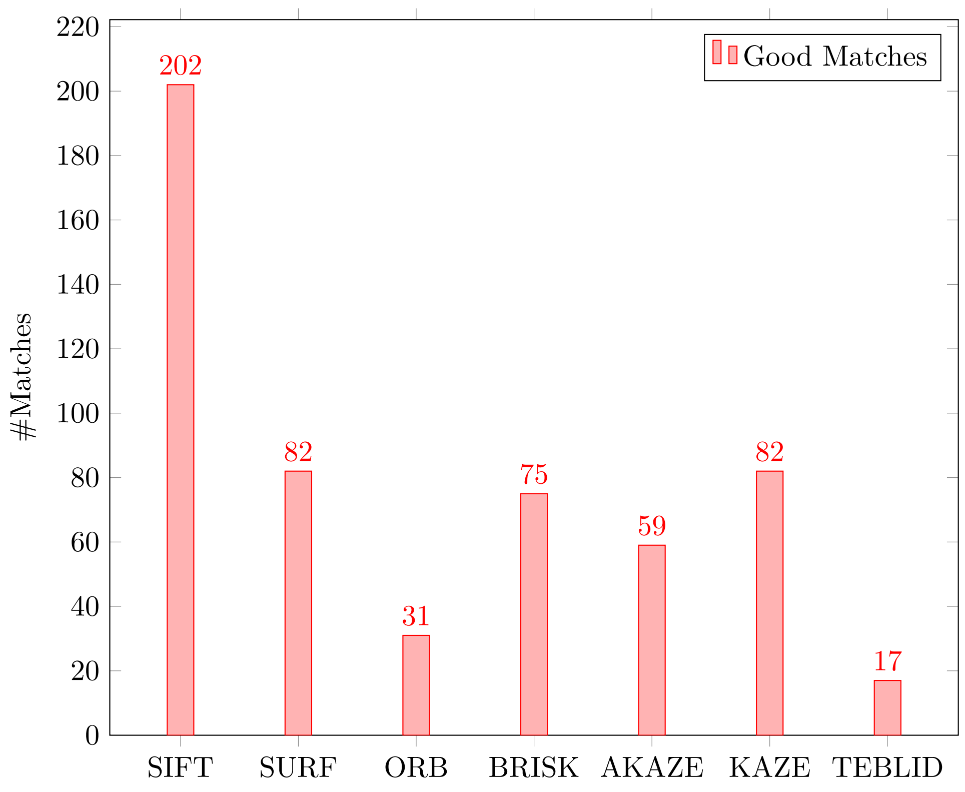


Figure 6.6: Number of Good Matches Found by the Algorithms

Figure 6.7 shows the percentage of good matches to overall matches.

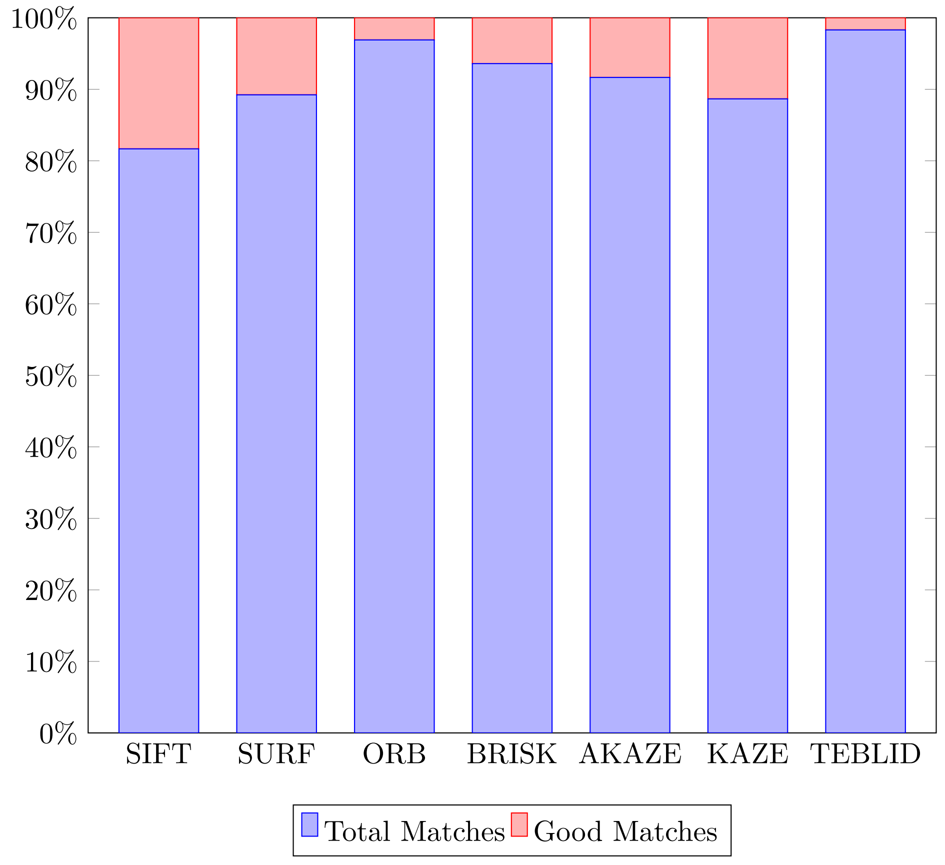


Figure 6.7: Ratio of Good Matches to Total Matches

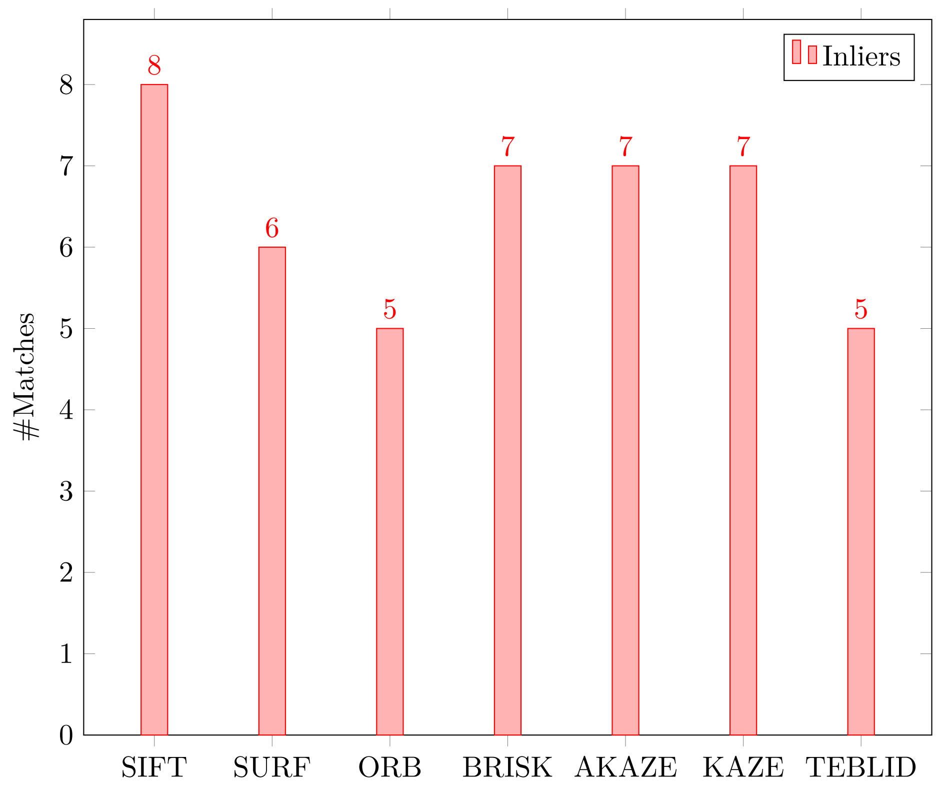


Figure 6.8: Number of Inliers via RANSAC

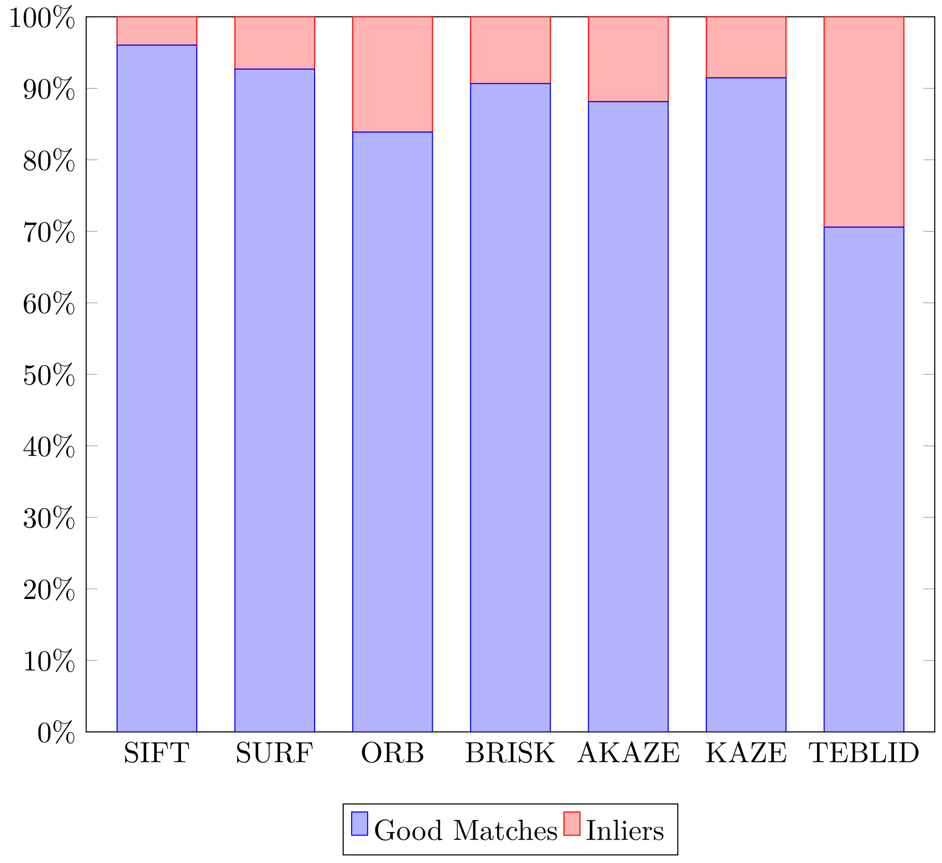


Figure 6.9: Ratio of Inliers to Good Matches

Figure 6.10 shows the inliers in relation to the computational time.

This is an interesting and unique metric and was computed by dividing the number of inliers by the average execution times of the algorithms. This metric provides an overview of how many valuable features to expect in a time frame. This may be mission-critical in some cases. For example, if a CubeSat has to use its camera images for emergency ADC in real time, large numbers of keypoints are needed in a short time for localisation.

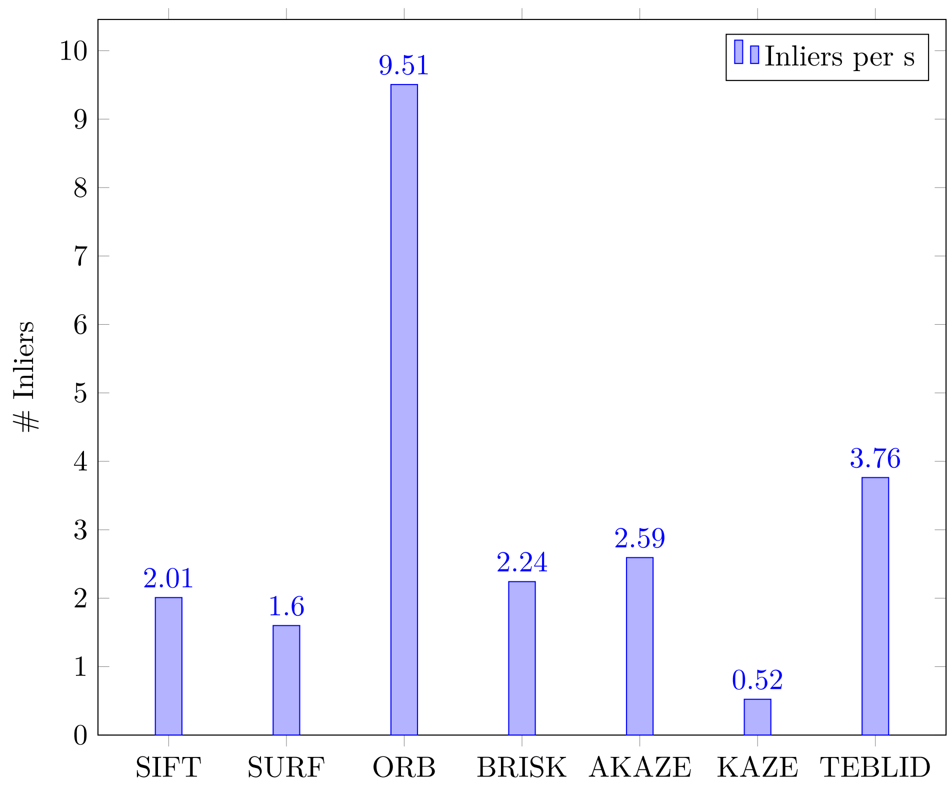


Figure 6.10: Extrapolated Inliers per second ($\frac{Inliers}{AverageDuration} \cdot 1000$)

6.3 Output Images

The following images serve as examples to illustrate the workings of the algorithms.

6.3.1 Good Matches

SIFT had the highest number of Good Matches, while TEBLID had the least. The comparison is visible in figure 6.11.

It is interesting to note that SIFT's keypoints were distributed fairly evenly on a vertical line, while TEBLID's matches were nearly all located in one specific region.

This leads to an unreliable and inaccurate homography matrix generation. The points near the equator move the furthest between the two images. In contrast, the points near the North Pole barely move place, and their rotation may be approximated with a translation. This leads to TEBLID ignoring the rotational component and generating an inaccurate perspective transform. Images for each algorithm are located in the Appendix, figures A.1 and A.2.

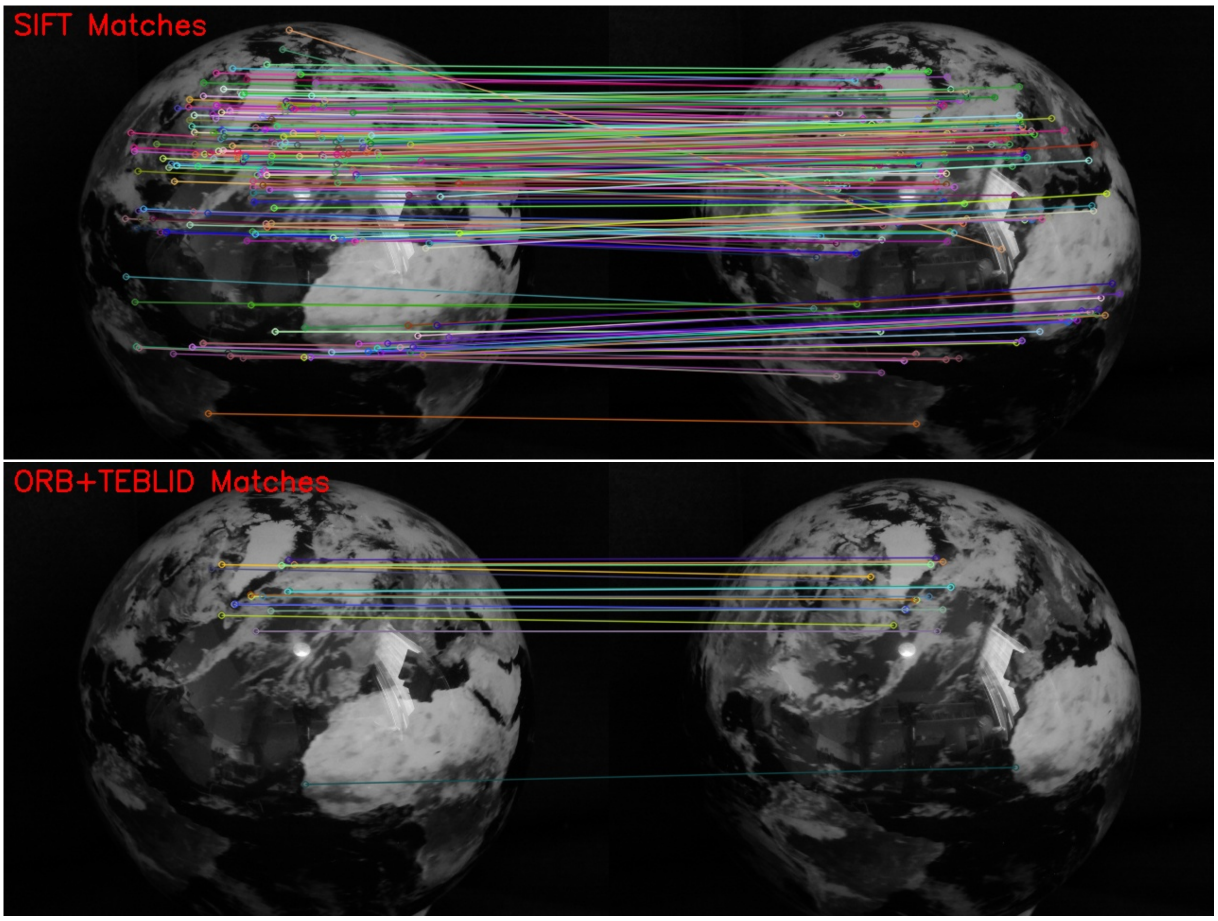


Figure 6.11: Example images with the highest and lowest number of Good Matches

Figure 6.12 shows the inliers of the SIFT and TEBLID algorithms. The inliers further exacerbate the distribution problem visible in figure 6.11. Even SIFT loses its vertical distribution and the points basically represent a simple translation. Images for each algorithm are located in the Appendix, figures A.3 and A.4.

6.3.2 Inliers

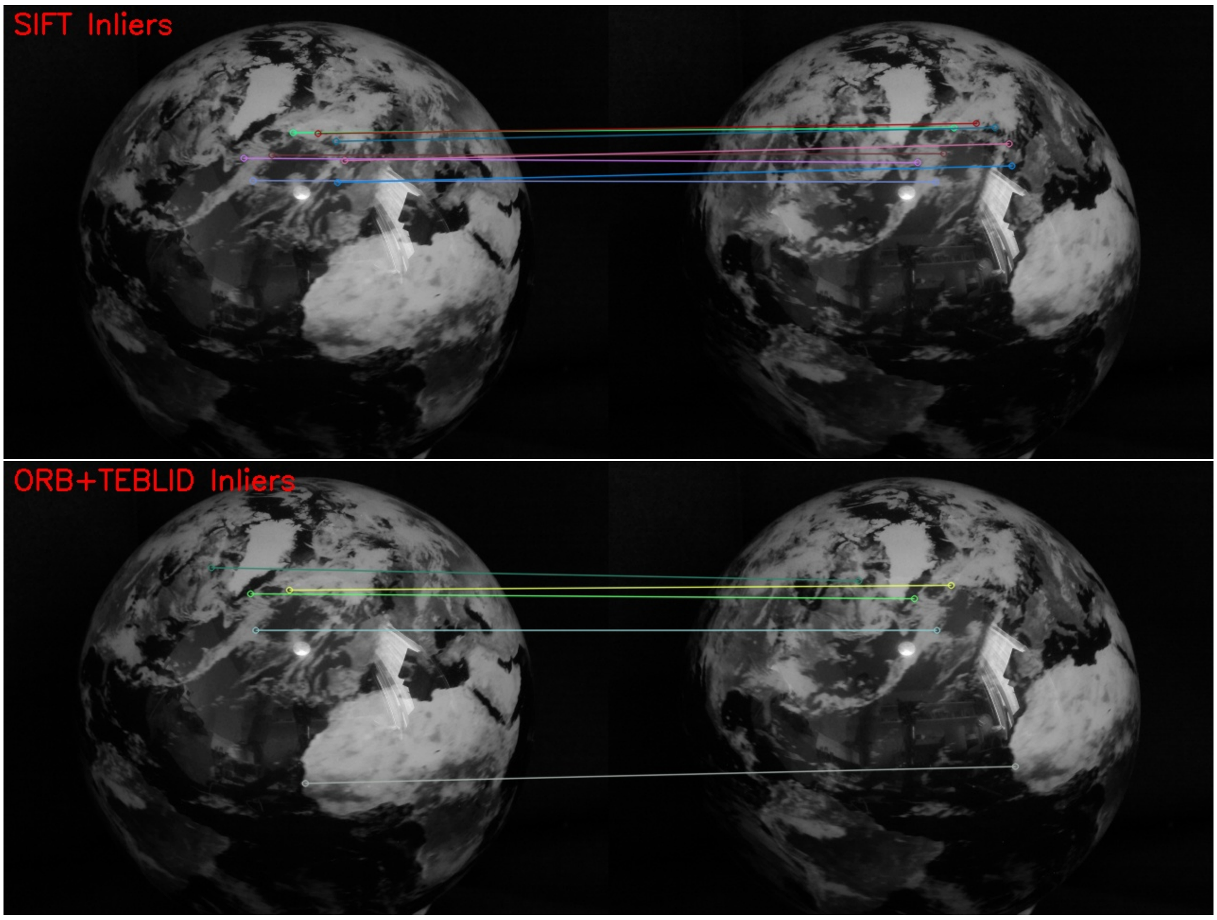


Figure 6.12: Example images with the highest and lowest number of Inliers after RANSAC

6.3.3 Homographies

To evaluate the accuracy of the homography matrices, multiple input points were manually selected. And each homography matrix resulting from each algorithm was applied. The results were grouped using k-means clustering into 3 clusters. The transformed points are drawn in green, and the cluster centres are drawn in red. The numbers labelling the green points represent the algorithm used with:

- 0 := SIFT
- 1 := SURF
- 2 := ORB
- 3 := BRISK
- 4 := AKAZE

- 5 := KAZE
- 6 := ORB + TEBLID

The analysis of the figures 6.13 and 6.14 leads to interesting conclusions.

Of all the tests, SIFT is the only algorithm that consistently provides fairly accurate outputs.

In general, it can be said that input points located near the centre of the image are able to be handled much better than inputs near the edges of the globe. This further points to an inaccuracy between the description of rotation and translation.

The BRISK algorithm performs so poorly that in most cases, the generated output point is outside the bounds of the image area.

KAZE has an interesting behaviour, the output point is always located in nearly the same spot, with small differences. This means that in addition to being by far the most computationally expensive algorithm, it is also inaccurate when it comes to generating homography matrices. The outputs of AKAZE are also located at similar points, but with larger variations than with KAZE.

With points located in the middle of the image, the output of ORB is also fairly reliable. This confirms the prior consensus that ORB is poor at rotation invariance.

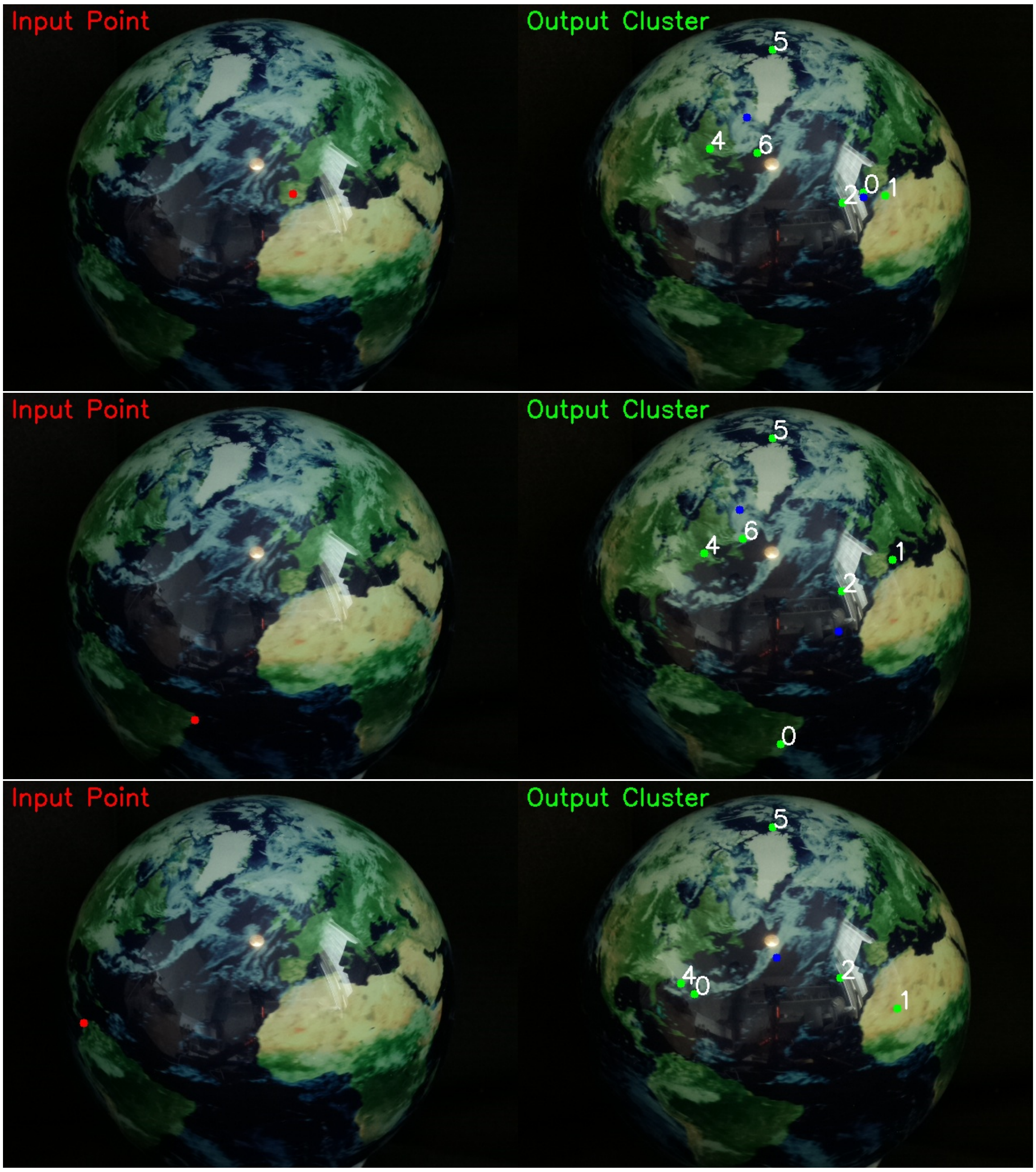


Figure 6.13: Homography accuracy with input point and corresponding output points

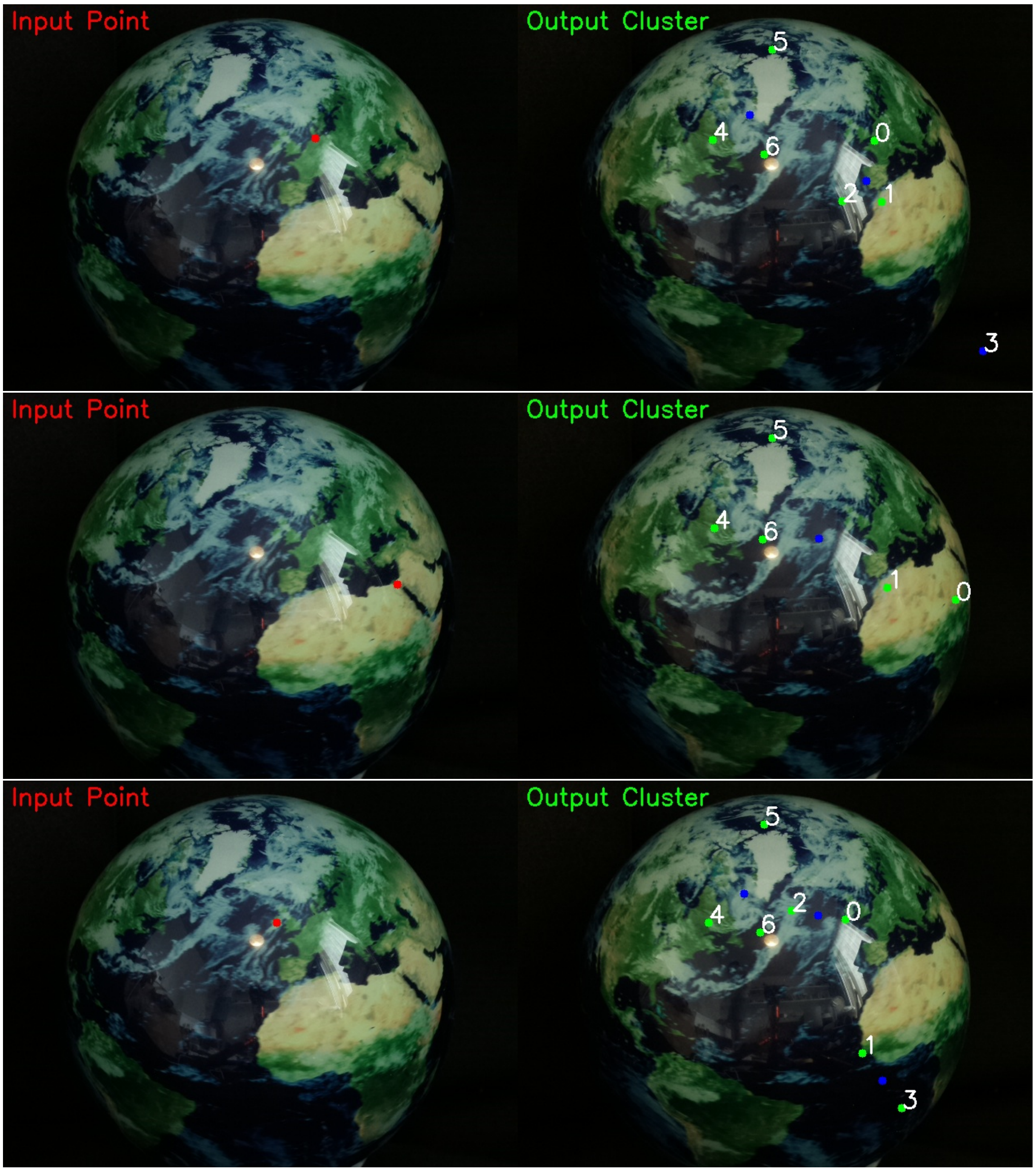


Figure 6.14: Homography accuracy with input point and corresponding output points

Chapter 7

Discussion

The experiment results confirm the prior consensus concerning the robustness and speed of the algorithms. SIFT has by far the most good matches. Its inliers are also the most, but not by such a large margin. Although its developers did not intend it to be a "fast" algorithm, its execution speed is still acceptable, especially compared to KAZE.

The SURF algorithm's ratio of Good Matches to all matches was lower than SIFT's. The execution time was comparable to SIFT but did not perform significantly better despite its name (i.e. "speeded up"). This is a curious finding and should be further investigated. A possible explanation is that due to the computational limits of the Raspberry Pi, it is not able to properly take advantage of the speeded-up calculations.

Compared with others' results, such as Pusztai et al. [33] and Tareen et al. [12], the results are broadly similar, with a few interesting differences.

Regarding the quantitative aspects, the previous authors found that ORB and BRISK generally found the most features, AKAZE and KAZE the fewest. In contrast, SIFT and SURF were in the centre field. The order was much the same on a computational time per keypoint basis, with SIFT performing worse than before. This order also persists for the matching performance.

In this work, ORB performed worse in all aspects except computational costs.

Pusztai et al. declare that, in their view, SURF was the most accurate and robust algorithm for matching. However, they mainly tested rotation, not translation variances, and it is questionable how far this is relevant for our purposes.

Tareen et al. conclude that SIFT is the most robust with respect to scale and rotational variances. ORB and BRISK performed acceptably for affine changes, with high efficiency. However, their overall accuracy and robustness are far behind SIFT.

This work includes a unique and novel factor, including a current (2021) cutting-edge algorithm, Triplet-based Efficient Binary Local Image Descriptor (TEBLID). This algorithm has yet to be thoroughly evaluated, but promises encouraging results. It has even been included in the official OpenCV code base. Thus, it will likely achieve widespread adoption shortly. Using the TEBLID descriptor instead of the default ORB improved the ratio of inliers to good matches. The absolute amounts were still lower than ORB's and the registration was unreliable. The added hardware use was minimal. This shows promising improvements, and the author's work should be followed for later developments. For now, however, it is not the best option.

The detailed and thorough hardware use evaluation in this work is also unique compared with prior research. However, this was easier because the platform was clearly and unambiguously defined.

Using figures 6.13 and 6.14, it is immediately obvious that SIFT had by far the most consistently accurate homography matrix. SURF and AKAZE also performed acceptably in some cases. However, the other algorithms all resulted in surprisingly inaccurate transformations. In particular, BRISK was far-off, with the resulting points usually were outside the image's bounds. KAZE projected the input point into almost the same output point in each case, with minimal variations.

Although SIFT is the oldest algorithm, its homography matrix results prove that it is still the most accurate and robust. SIFT's speed is on par with that of more modern algorithms, and its use of hardware resources is average.

The relevancy of the homography matrix is mission dependent. As a satellite orbits the Earth and the Earth rotates, translational, rotational, and affine transformations are all encountered. If the images are taken far apart in time, no previous features will be visible, and so the homography matrix is unusable. However, if the images are taken shortly after each other, the corresponding features are visible, and the matrix is relevant.

In addition, the orbit's height and the lens's focal length are also vital factors. The homography matrix is much more applicable for low orbits and high focal lengths (i.e. zoomed in). It is the opposite for higher orbits and wider lenses (i.e. zoomed out), as the changes are more complicated. However, images must be taken more rapidly at lower orbits, as the viewed areas move rapidly, and corresponding features may be easily lost.

As mentioned previously, the figure 6.10 shows the number of inliers found per second. If a satellite needs to use the images, not just for EO, but also for ADC, faster algorithms such as ORB and TEBLID must be considered. Delegating ADC tasks to the camera module simplifies the payload of a camera and frees up vital space and weight. However, care must be taken that the need for rapid keypoints does not negatively affect the quality of the EO. In this case, it may be advisable to use both a fast and a robust algorithm, depending on the current need.

If the computational platform's capacity allows for it, it is the most suitable algorithm for the task. For minimal CPU, RAM and speed savings, SURF and AKAZE may be considered. For the rapid acquisition of valuable keypoints, ORB, TEBLID and AKAZE are the most suitable.

This result matches the conclusion of the prior authors'.

Chapter 8

Conclusion and Outlook

Feature matching in Earth Observation CubeSats can be used to automate the analysis of large volumes of image data, making it easier to identify patterns and objects of interest in the images captured by these satellites. The most common current and future applications include:

- Change Detection: Feature matching can be used to identify changes in the Earth's surface over time. By comparing images taken at different times, feature matching algorithms can detect environmental changes such as urbanisation, deforestation, and changes in land use.
- Object Detection and Tracking: Feature matching can be used to detect and track specific objects of interest, such as ships, vehicles, and aircraft. This can be useful in surveillance and land monitoring applications.
- Terrain Mapping: Feature matching can create 3D maps of the terrain by matching features in multiple images. This can be useful for applications such as disaster response and urban planning.
- Visual servoing and Attitude Determination and Control (ADC): Using software for determining and controlling attitude enables the use of sensor fusion to improve the accuracy of hardware-based approaches. This is already used by star detectors, for example. In the future, it may be possible to completely eliminate the need for hardware by building robust feature matching systems. This leads to lower overall weights and dimensions, allowing for cheaper launches. In addition, hardware is more prone to breakdowns.
- Rapid response Measurement and signature intelligence (MASINT): optical features combined with spectrum analysis offer high-speed surveillance of events such as nuclear activities. When relying solely on features, response time may be significantly shortened.

In summary, feature matching can be a powerful tool for earth observation CubeSats. Automating the analysis of image data can help identify patterns and objects of interest in a timely and efficient manner, making it easier to monitor changes in the environment and respond to events.

Alongside the many advantages CubeSats offer, they also present many disadvantages that require changes in the usual software engineering approach. CubeSats have limited space and weight capacity, which can limit the performance of the payload. Microcontrollers and SBCs are needed to conform to the limits. CubeSats may also have limited communication capabilities due to their small size and power constraints. This means that they cannot broadcast any arbitrary amount of data.

Any software must be carefully evaluated and tested to ensure compatibility with these limitations. The experiments described in this work achieved this goal. With the present contributions, efficient and reliable Computer Vision is made possible for anyone who launches a Earth Observation nanosatellite. This serves to broaden access and further democratise space exploration.

Appendix A

All Images

A.1 Good Matches Images

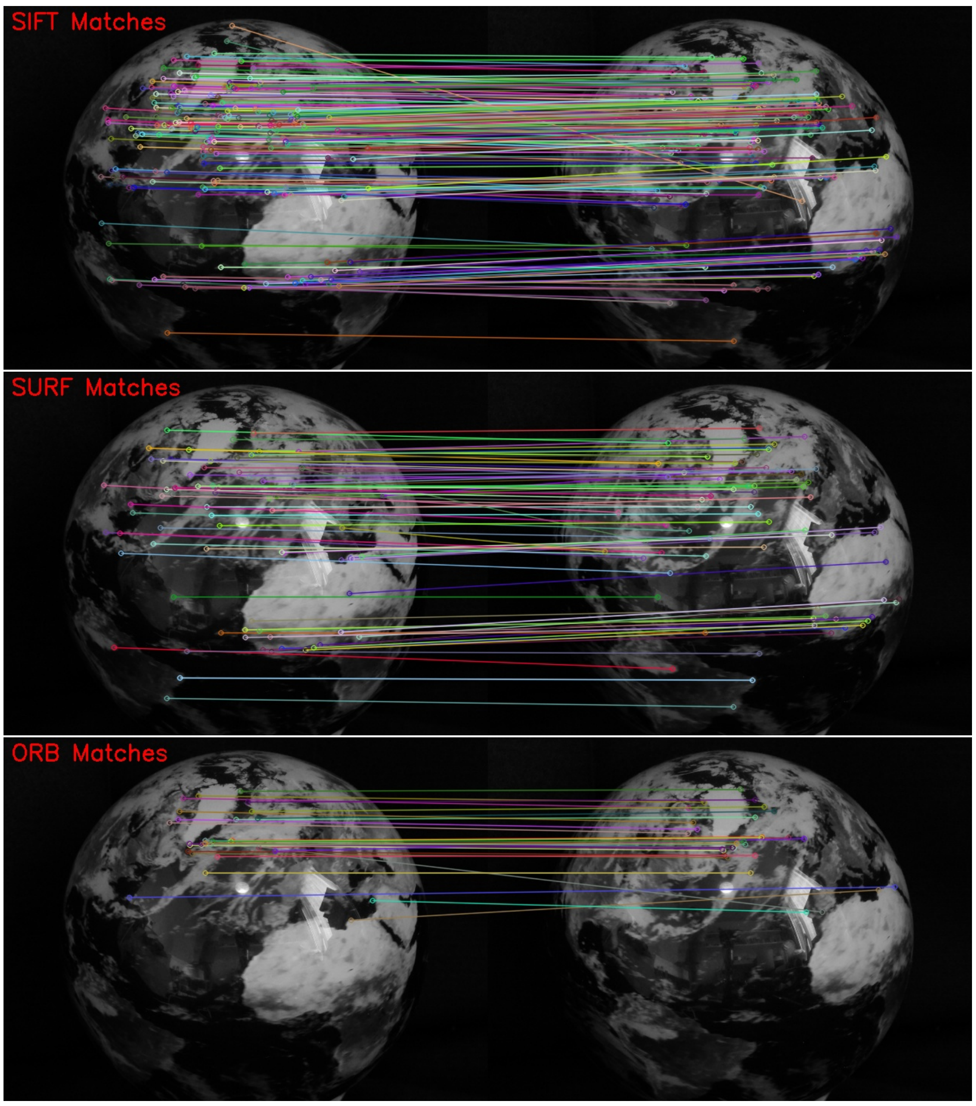


Figure A.1: Good Matches of each algorithm

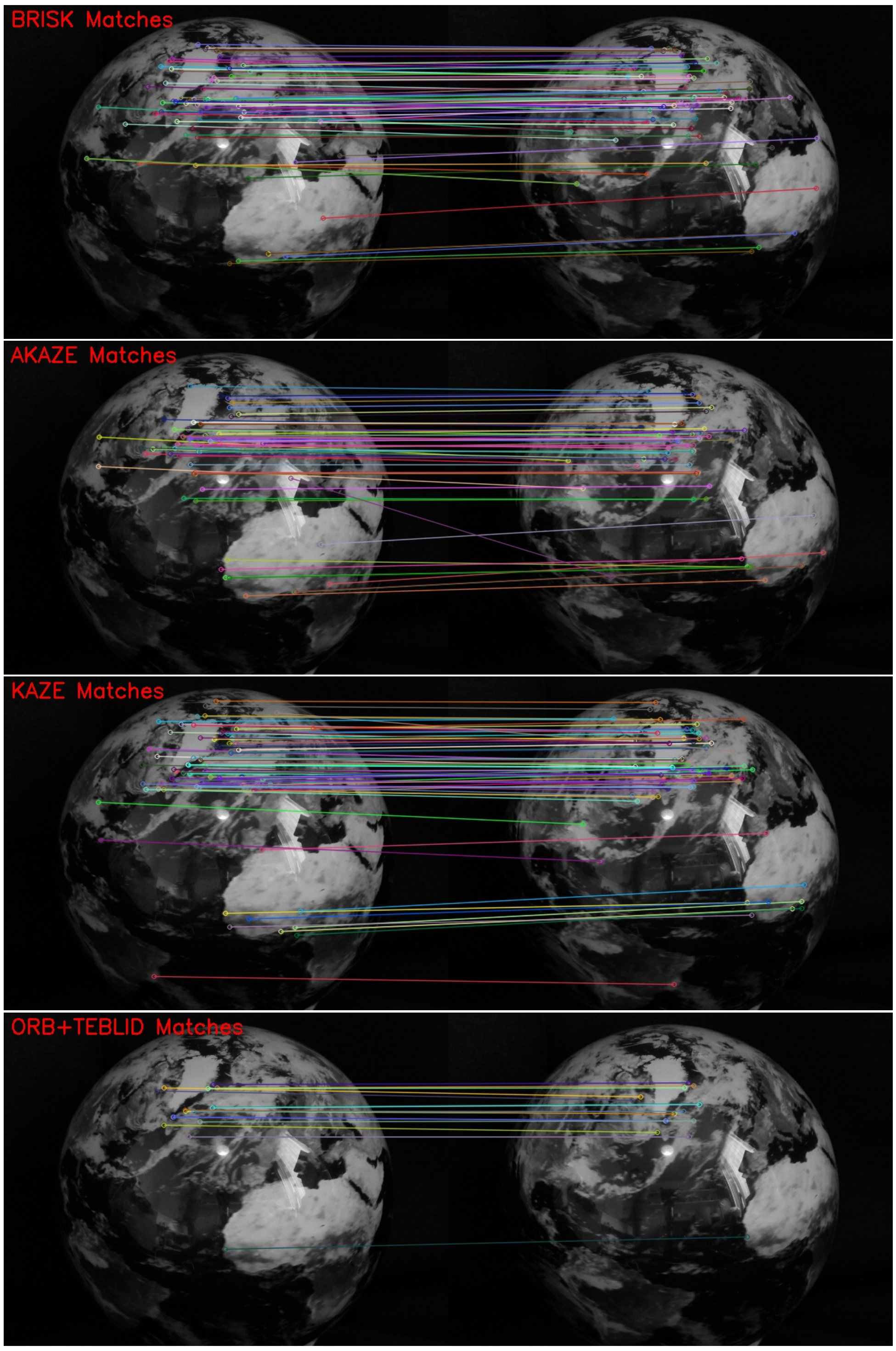


Figure A.2: Good Matches of each algorithm

A.2 Inliers Images

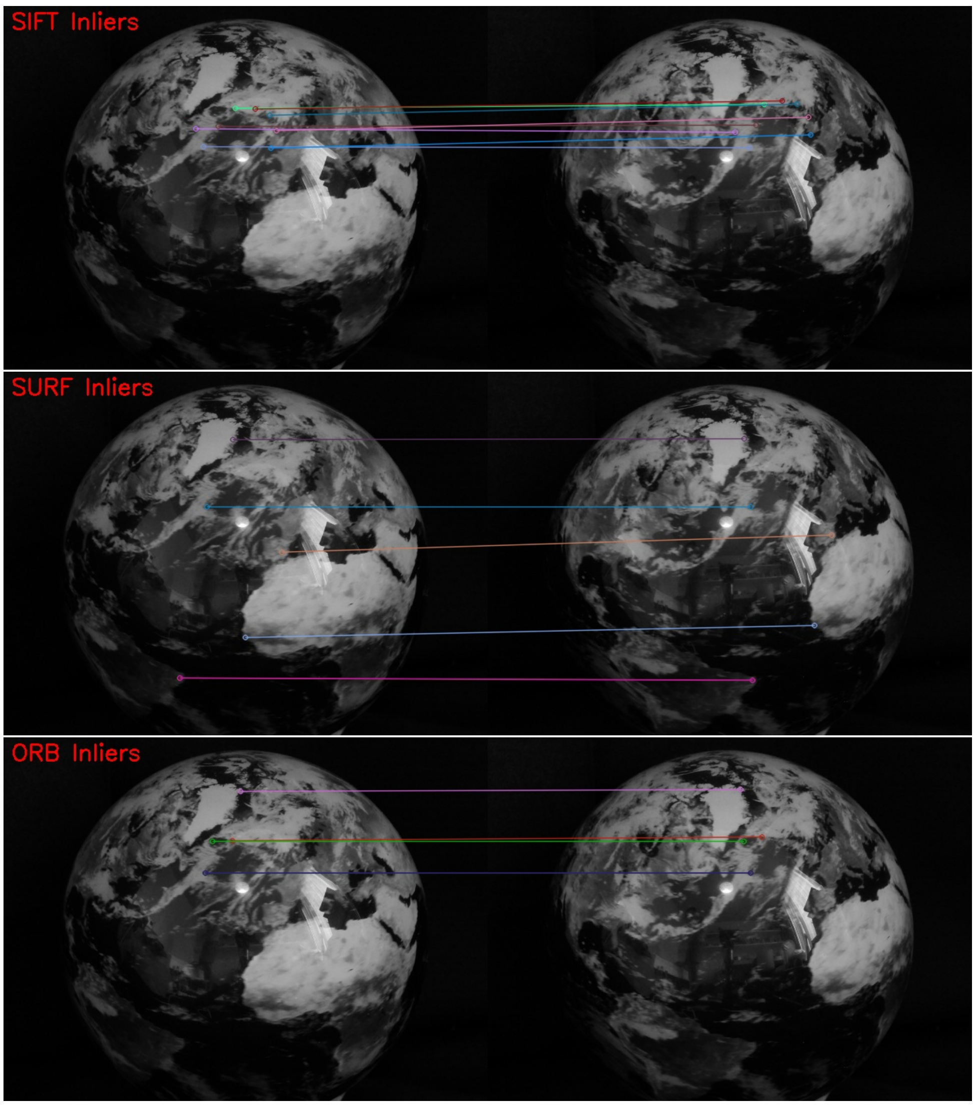


Figure A.3: Inliers of each algorithm

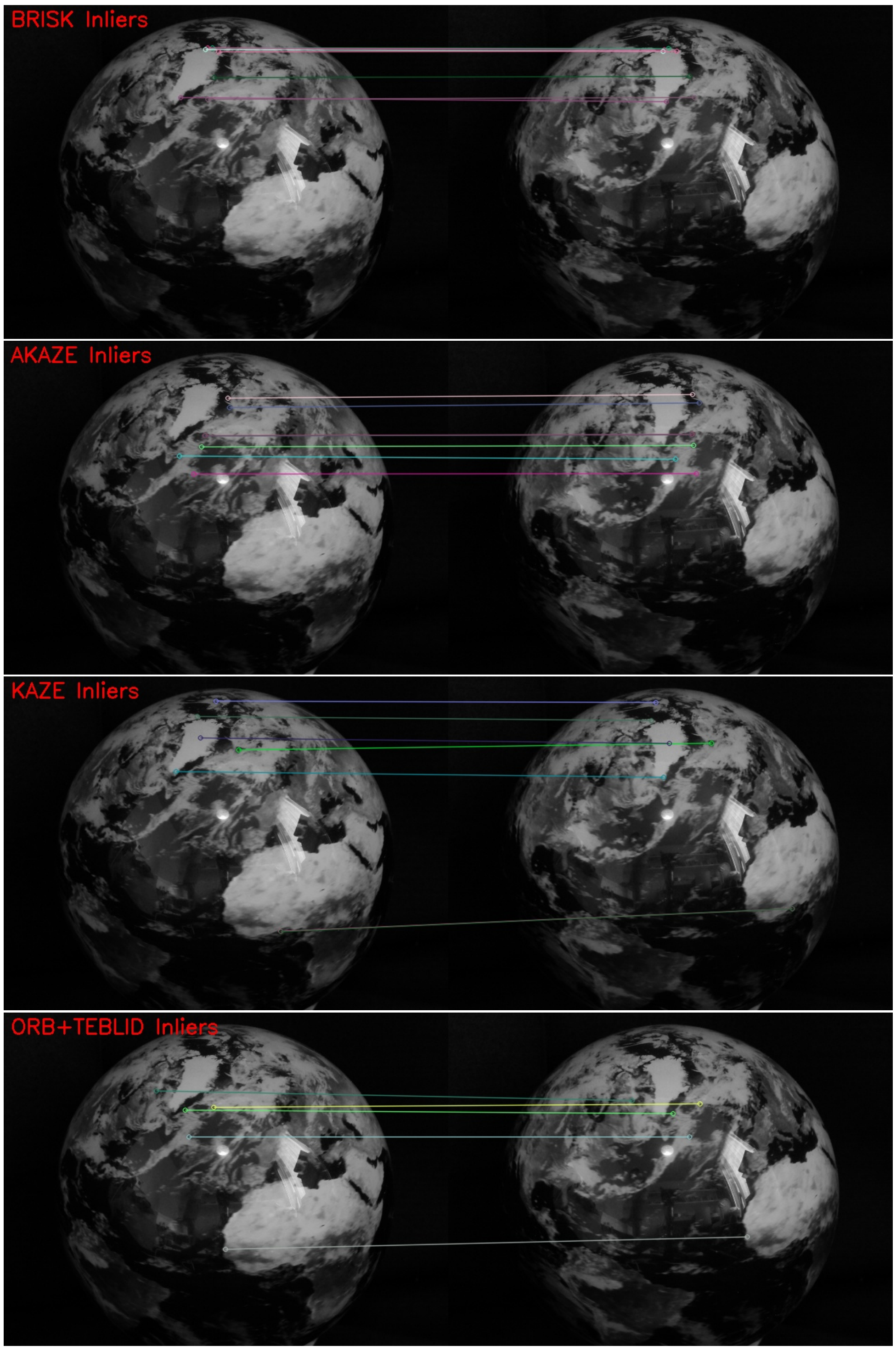


Figure A.4: Inliers of each algorithm

Bibliography

[1] European Patent Office. EP 1224653 A2 20020724 - FRICTIONLESS SELF-POWERED MOVING DISPLAY — data.epo.org. `https://data.epo.org/gpi/EP1224653A2`, 2023.

[2] MOVA Globes. Earth with Clouds Globe — movaglobes.com. `https://www.movaglobes.com/earth-clouds/`, 2023.

[3] OpenCV. OpenCV: Feature Detection and Description — docs.opencv.org. `https://docs.opencv.org/4.x/d5/d51/group__features2d__main.html`, 2023.

[4] S Lee, A Hutputanasin, A Toorian, W Lan, R Munakata, J Carnahan, D Pignatelli, A Mehrparvar, and A Johnstone. CubeSat Design Specification Rev. 14 The CubeSat Program, Cal Poly SLO (No. CP-CDS-R14.1). *Cal Poly, San Luis Obispo, CA*, 2022.

[5] Erik Kulu. Nanosatellite Launch Forecasts-Track Record and Latest Prediction. In *Proceedings of the 36th Annual Small Satellite Conference*. Utah State University, 2022.

[6] Julian Scharnagl, Roland Haber, Veaceslav Dombrovski, and Klaus Schilling. NetSat—Challenges and lessons learned of a formation of 4 nano-satellites. *Acta Astronautica*, 201:580–591, 2022.

[7] Raspberry Pi Foundation. Raspberry Pi Documentation - Camera — raspberrypi.com. `https://www.raspberrypi.com/documentation/accessories/camera.html#hardware-specification`, 2023.

[8] Zft. NetSat - Pionierforschung fuer Formationskontrolle — telematik-zentrum.de. `https://www.telematik-zentrum.de/netsat/`.

[9] University of Würzburg. NetSat: Der erste Kontakt — uni-wuerzburg.de. `https://www.uni-wuerzburg.de/aktuelles/pressemitteilungen/single/news/netsat-der-erste-kontakt-1/`.

[10] OpenCV. About - OpenCV — opencv.org. `https://opencv.org/about/`.

[11] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with OpenCV. *Communications of the ACM*, 55(6):61–69, 2012.

[12] Shaharyar Ahmed Khan Tareen and Zahra Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*, pages 1–10. IEEE, 2018.

[13] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. BRISK: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. Ieee, 2011.

[14] G Lowe. Sift-the scale invariant feature transform. *Int. J*, 2(91-110):2, 2004.

[15] PATENTSCOPE. US6711293 Method and apparatus for identifying scale invariant features in an image and use of same for locating an object in an image — patentscope.wipo.int. `https://patentscope.wipo.int/search/en/detail.jsf?docId=US40341179`.

[16] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[17] Google Patents. US20090238460A1 - Robust interest point detector and descriptor - Google Patents — patents.google.com. `https://patents.google.com/patent/US20090238460A1/en`.

[18] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[19] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part VI 12*, pages 214–227. Springer, 2012.

[20] Iago Suárez, José M Buenaposada, and Luis Baumela. Revisiting binary local image description for resource limited devices. *IEEE Robotics and Automation Letters*, 6(4):8317–8324, 2021.

[21] Armin Moghimi, Turgay Celik, Ali Mohammadzadeh, and Huseyin Kusetogullari. Comparison of keypoint detectors and descriptors for relative radiometric normalization of bitemporal remote sensing images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14:4063–4073, 2021.

[22] Lingxuan Meng, Ji Zhou, Shaomin Liu, Lirong Ding, Jirong Zhang, Shaofei Wang, and Tianjie Lei. Investigation and evaluation of algorithms for unmanned aerial vehicle multispectral image registration. *International Journal of Applied Earth Observation and Geoinformation*, 102:102403, 2021.

[23] Mohamed Tahoun, Abd El Rahman Shabayek, Hamed Nassar, Marcello M Giovenco, Ralf Reulke, Eid Emary, and Aboul Ella Hassanien. Satellite image matching and registration: A comparative study using invariant local features. In *Image Feature Detectors and Descriptors*, pages 135–171. Springer, 2016.

[24] Sirshendu Hore, Tanmay Bhattacharya, Nilanjan Dey, Aboul Ella Hassanien, Ayan Banerjee, and SR Bhadra Chaudhuri. A real time dactylology based feature extraction for selective image encryption and artificial neural network. In *Image feature detectors and descriptors*, pages 203–226. Springer, 2016.

[25] Hrishikesh Bhaumik, Siddhartha Bhattacharyya, and Susanta Chakraborty. Redundancy elimination in video summarization. In *Image Feature Detectors and Descriptors*, pages 173–202. Springer, 2016.

[26] Nabeel Younus Khan, Brendan McCane, and Geoff Wyvill. SIFT and SURF performance evaluation against various image deformations on benchmark dataset. In *2011 International Conference on Digital Image Computing: Techniques and Applications*, pages 501–506. IEEE, 2011.

[27] nmon. nmon for Linux — Main / HomePage — nmon.sourceforge.net. `https://nmon.sourceforge.net/pmwiki.php`, 2023.

[28] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[29] Jinliang Liu and FanLiang Bu. Improved ransac features image-matching method based on surf. *The Journal of Engineering*, 2019(23):9118–9122, 2019.

[30] Şahin Işık. A comparative evaluation of well-known feature detectors and descriptors. *International Journal of Applied Mathematics Electronics and Computers*, 3(1):1–6, 2014.

[31] OpenCV. OpenCV: Camera Calibration and 3D Reconstruction — docs.opencv.org. `https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga4abc2ece9fab9398f2e560d53c8c9780`, 2023.

[32] J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297. University of California Los Angeles LA USA, 1967.

[33] Zoltan Pusztai and Levente Hajder. Quantitative Comparison of Feature Matchers Implemented in OpenCV3. *21st Computer Vision Winter Workshop*, 2016.

Proclamation

Hereby I confirm that I wrote this thesis independently and that I have not made use of any other resources or means than those indicated.

Würzburg, March 2023