electronics

Communication

# Implementation of an FPGA-Based System to Process Images and Match Keypoints on High-Resolution Pictures

**Sina Bundschuh *, Jan Kunze and Klaus-Dieter Kuhnert**

Institute of Real-Time Learning Systems, University of Siegen, 57076 Siegen, Germany; jan.kunze@uni-siegen.de (J.K.)
* Correspondence: sina.timmermann@uni-siegen.de

**Abstract:** Processing scenery and finding points of interest is crucial for applications in robotics and aerospace missions. Those areas require efficient and reliable visual input processing. Here, field programmable gate arrays (FPGAs) offer essential advantages, like low power consumption compared to CPUs, performing a large number of calculations simultaneously, and having compact hardware. This paper presents an FPGA system that processes incoming camera data, finds points of interest, and matches them across different images on high-resolution images (2048 × 1088). It is a novel approach to implement the complete image processing pipeline on high-resolution images within the FPGA fabric without additional hardware. For keypoint detection and matching, our work uses a modified SIFT algorithm optimized for FPGA implementation processing and a nearest neighbor-based matching method. It was implemented on a Xilinx Kintex-7 FPGA and partially on a NanoXplore NG-Ultra to evaluate a radiation-hardened FPGA for space applications. On the Kintex-7, the keypoint detection achieves a speed of 33 ms per image, and its features are matched on up to 5 images per second. Judging by the resource utilization of one image processing module on the NG-Ultra, porting the entire system on a radiation-hardened FPGA appears feasible.

**Keywords:** field-programmable gate array (FPGA); image processing; scale-invariant feature transform (SIFT); keypoint detection; keypoint matching; radiation hardened

**Citation:** Bundschuh, S.; Kunze, J.; Kuhnert, K.-D. Implementation of an FPGA-Based System to Process Images and Match Keypoints on High-Resolution Pictures. *Electronics* **2024**, *13*, 4774. https://doi.org/10.3390/electronics13234774

Academic Editor: Silvia Liberata Ullo and Li Zhang

Received: 13 November 2024
Revised: 29 November 2024
Accepted: 30 November 2024
Published: 3 December 2024

## 1. Introduction

Finding points of interest in a visual scenery is a key issue in many applications that benefit from visual input, for example, robotic applications or aerospace missions. Implementing algorithms on a field programmable gate array (FPGA) allows for more parallel calculations than on a CPU and brings advantages like low power consumption and compact hardware, which is especially beneficial in robotics and aerospace applications. Our research focuses on developing hardware for those fields of research. The proposed architecture in this work is part of a modular sensor fusion system. The whole sensor fusion system's purpose is to fuse different sensors (image sensors and other sensors like radar and LIDAR). The presented work here implements an image processing chain for incoming camera data. The system was initially designed as a proof-of-concept for short-term missions using energy-saving and commercial-off-the-shelf (COTS) hardware but is actively developed further for long-term space missions with funding through the German Space Agency (DLR). To meet our hardware requirements, we decided to implement the algorithms on an FPGA. As the hardware should be compact and not have a lot of power consumption, it should be a smaller FPGA in a cost-saving range. This comes with the downside of a reduced number of resources in the FPGA fabric.

To implement algorithms that process a visual scenery on an FPGA for space applications, two iterations of optimizations are necessary. The first step is to make the system efficient enough to run on a small FPGA, the next step is to optimize the algorithms enough to be able to run them on radiation-hardened hardware. In our proposed image processing architecture, we focus on keypoint detection and feature matching. Our system follows a modular design, and the matched keypoints can be used by different applications. The output of images and matched keypoints can be flexibly integrated into further processing chains and calculations.

Implementing a keypoint detection comes with two major challenges. One problem is that detecting keypoints in an FPGA is resource intensive. The other is that the resulting keypoints from images with large resolution can result in a very large amount of data that can be difficult to handle in FPGAs. This is especially a factor in smaller and cost-saving FPGAs that come with a more limited amount of resources.

For keypoint detection, multiple algorithms exist. SIFT (scale-invariant feature transform) [1] is a commonly used algorithm for detecting keypoints and is a good asset for many applications due to its robustness regarding rotation and scaling. On the downside, this robustness comes at the cost of needing a larger amount of resources and a slower performance compared to other, less robust approaches. Prokop and Połap [2] mention the SIFT algorithm as one of the most popular feature detection algorithms besides ORB for applications like image stitching. Refs. [2,3] both mention that neural networks are another commonly used method to detect keypoints. However, neural networks for feature detection are not a viable option for many critical space applications because they need a large amount of computational power. Since hardware for critical space applications and long-term missions is very limited in its computational capabilities, classical feature detection methods like SIFT, SURF, or ORB are still used for image processing in space [3,4]. Hong and Shin [4] compare feature detection algorithms for lunar rover missions. Their work evaluates SIFT, SURF, BRISK, ORB, and AKAZE. SIFT and AKAZE showed the best results, although SIFT has the disadvantage of a high computation time. In our previous work [5], we also evaluated several keypoint detection algorithms in software: SIFT, SURF, ORB, AKAZE, BRISK, BRIEF, and KAZE. Similar to [4], we found that SIFT provided the best results with the drawback of having a high runtime. In our presented work, we overcome this problem by implementing an adapted version of SIFT on FPGA hardware to achieve a higher speed.

Existing works usually focus on using an FPGA as an accelerator for algorithms in software. Those solutions commonly implement either keypoint detection or keypoint matching and do not implement both on a single FPGA.

Works that use a stand-alone FPGA commonly work on very low resolutions or high-cost FPGAs that, in many cases, have support from an on-board processor, like the XILINX Zynq FPGA family.

We optimize and adapt both a keypoint detection algorithm and a keypoint matching algorithm so that both can be implemented together on an FPGA as a stand-alone system with no additional processor and still work on high-resolution images.

We have implemented a method that uses a modification of SIFT that was optimized for an implementation on FPGAs. This modified SIFT works on pictures with a high resolution of 2048 $\times$ 1088, and the keypoints are subsequently matched. The matching is done with an adaptation of a nearest neighbor matching method whose computational operations are suitable for an FPGA. Our image processing system receives images from a camera, processes the picture to resolve the camera's RGB-Bayer pattern, finds keypoints on those images, and matches them across different pictures while using a cost-saving FPGA as a stand-alone system with no additional computational hardware. Furthermore, we discuss the use of a NanoXplore FPGA for this use-case, which is a radiation-hardened FPGA compared with the Xilinx Kintex-7 FPGA that is currently used in our research. This paper focuses on the matching part of the processing chain, as this is a challenging task for FPGAs. The matching is completed on high-resolution pictures, which results in a large amount of data. The data, in order to find the best matching pair, need to be compared. That limits the possible parallelization of the task and implies that a lot of data needs to be stored for a longer time span. This is a challenging task for FPGAs, which are well-suited for parallel tasks on a smaller amount of data and have only limited resources to handle and store large batches of information.

There are different FPGAs that were considered for this system: The SIFT was first implemented on a Virtex-7 evaluation board, and the whole image processing chain was eventually implemented on a Xilinx Kintex-7 FPGA. Later, a part of the image processing chain was additionally implemented on a NanoXplore NG-Ultra FPGA. The Virtex-7 FPGA offers a higher amount of resources which lends itself to a faster development, while the Kintex-7 FPGA is a more cost-saving approach. The NG-Ultra FPGA is considered to be used for this image processing chain in the future because, as a radiation-hardened FPGA, it would be suitable for longer space missions.

Our goal is to develop a compact, modular, and energy-saving image processing system that lays the groundwork for applications in space missions. The goal is to make the system efficient enough to be run both on smaller FPGAs and on radiation-hardened FPGA hardware.

Our contributions are the following:

- An image processing system that is exclusively implemented in the FPGA fabric with no need for additional processors (e.g., Intel CPU, Microcontroller Boards).
- Optimizing the keypoint finding and matching algorithms so that they can be implemented together in the FPGA. This way we can receive and process camera data and give out matching pairs of keypoints on one single FPGA as a stand-alone system in contrast to systems that do not implement the whole processing pipeline.
- Matching keypoints in the FPGA fabric on images with a high resolution of 2048 $\times$ 1088 pixels.
- Evaluation of the NanoXplore NG-Ultra for the presented image processing system. To the best of our knowledge there are no related works utilizing this FPGA

## 2. Related Works

There are several related works that implement feature detection on an FPGA. Bonato et al. [6] implemented a feature detection based on SIFT, which detects features in up to 30 frames per second. Chang et al. [7] developed a SIFT-based detection of keypoints based on FPGA using a Xilinx Virtex II Pro device (XC2-VP30-5FF1152). In [8], an architecture was created for detecting features with FAST for an Xilinx Zynq XC7Z020 device. De Lima [9] improved the ORB (oriented FAST and rotated brief) feature extraction through FPGA acceleration, which was also synthesized for a Xilinx Zynq XC7Z020 device. However, all of the aforementioned solutions are limited to a resolution of 320 $\times$ 240 pixels, which restrains the use of sensors with greater resolution.

Some solutions achieve a higher resolution of 640 $\times$ 480, [10] designed an improved implementation of the SIFT algorithm on a Xilinx Virtex-5 FPGA, [11] implemented the BRIEF and FAST algorithms on a Zynq-7000. Kuo et al. [12] achieved a framerate of 205 FPS with an enhanced SIFT implementation on an Altera development board that is equipped with a Cyclone IV FPGA.

Approaches that work with a higher resolution rely on a combination of FPGA with additional hardware such as to be found in the Xilinx Zynq-Series. In [13], a Xilinx Zynq-7045 was used to accomplish feature matching with a real-time implementation of SIFT with a resolution of 1280 $\times$ 720. Ref. [14] utilized an algorithm called SYBA (synthetic basis) for finding and describing features on pictures up to1920 $\times$ 1080 pixel on a Xilinx Zynq XC7Z020.

A resolution of up to 1920 $\times$ 1080 was reached by Fularz et al. [15] on a Xilinx Zynq-7000 SoC based on the FAST and BRIEF algorithm proposed in [16], using a coprocessor for the matching core units. Peng et al. [17] worked with a resolution of 1920 $\times$ 1080 with the SIFT algorithm implemented on a development board (ZC702) from the Xilinx Zynq-series. Other architectures are implemented as a co-design between an FPGA and a CPU, as in [18], who introduced a design that extracts SIFT features on 800 $\times$ 480 images, based on an Altera Cyclone IV FPGA, where part of the computations are run on a PC with an Intel Core i7-3770 CPU. Similarly, [19] extracted SIFT features used for object tracking on pictures up to a size of 800 × 480 pixels by using a combination of FPGA (Altera DE2i-150 Development Board) and a Nios II CPU.

Works that include feature matching commonly use the FPGA as an accelerator, as seen in [20], where an accelerator for SIFT feature matching is implemented on a Xilinx Zynq-based FPGA development board on images with a resolution of 512 × 384 pixels. Ref. [21] also implemented an accelerator for SIFT feature matching, on a Stratix IV (EPS4SE820F) with a slightly higher resolution of 640 × 480 pixels. Ref. [22] reached a resolution of up to 1000 × 700 pixels for the ORB feature detection and matching on a Zynq device with a Xilinx XC7Z020CLG484-1.

Recent works tend to focus on using the FPGA as an accelerator and not as a stand-alone system, and/or use newer generations of FPGA development boards with onboard processors, such as the Zynq UltraScale+ series or the Avnet Ultra96-V2 development board. An example of such an architecture can be found in [23], where a FAST feature module was combined with a pyramid module for enhanced robustness, implemented on a Zynq UltraScale+ MPSoC ZU15EG. Another work utilizing the Avnet Ultra96-V2 development board can be found in [24], where the FPGA is used as an accelerator for ORB-based SLAM.

In related works, other authors implemented feature detection algorithms on FPGA, though mostly with resolutions as low as 320 × 420 pixels or with help from on-board microprocessors, which combine an FPGA with additional computational units. Spreading the algorithm and sharing it between FPGA and software usually presents a bottleneck and the additional hardware increases the power consumption. Approaches with a higher resolution used a high-cost FPGA with a large amount of logic units, as can be seen in [25], who achieved a resolution of 3840 × 2160 on an Intel Arria 10 GX 1150 FPGA. However, this FPGA is in the upper price range and is equipped with 1.150.000 logic cells, whereas our cost-saving approach uses a smaller, budget Kintex Series 7 FPGA with 406.720 logic cells. As Table 1 shows, our work presents a novelty as the feature extraction and matching is implemented in an FPGA without a co-processing unit and still achieves a higher resolution.

**Table 1.** Comparing resolution and utilized hardware in the proposed architecture and related works.

| | **Proposed Architecture** | **Belmessaoud 2022 [22]** | **Daoud 2020 [20]** | **Zhang 2023 [23]** |
|---|---|---|---|---|
| Algorithm | SIFT feature extraction and matching | ORB Feature detection and matching | Accelerator for SIFT-Matching | FAST features with pyramid module |
| Resolution | 2048 × 1088 | Up to 1000 × 700 | 512 × 384 | 1920 × 1080 |
| Hardware | Kintex410 | Zynq Device | Zynq-based Development board | Zynq UltraScale+ |

## 3. Materials and Methods

The image processing system consists of several cameras and a Kintex7 325 FPGA. The system on the FPGA has several modules that fulfill the different functions in the image processing chain. The modules receive images from an IDS industrial camera with a resolution of 2048 × 1088 pixels. It extracts the SIFT features and matches the correlating features. All of the modules are implemented on a Xilinx 7-Series FPGA without additional software. The complete pipeline, from the acquisition of the pictures and the subsequent operations on the image material, is implemented in the FPGA fabric. The setup is, therefore, very compact and energy saving. The setup with a custom-designed FPGA carrier board, which is equipped with a Kintex 410, is shown in Figure 1. In this example, it is connected via ethernet to receive data from an industrial camera, and to send the output data of the proposed image processing system to a PC.

The camera uses the GigE-Vision protocol and sends 24 frames/s to the fpga via an ethernet/udp (User Datagram Protocol) that supports up to 1GB/s. In the first step, the camera data received by the FPGA is processed in real-time by a module that resolves the camera's Bayer pattern. The incoming data from the camera have a Bayer pattern so that in order to receive a suitable image, the module takes the camera data, calculates the values of each pixel in the picture, and constructs both a monochrome greyscale and an RGB picture. The RGB picture is forwarded by the fpga to be used for different purposes by other user-specified hardware, while the grayscale picture is used in the fpga for the next step in the pipeline, the difference of Gaussian module (DoG). This module pre-processes the images before they are used in the SIFT-based feature detection module. The output of the DoG module consists of several octaves where the picture is scaled down and blurred to different degrees. Those octaves are being used for the next steps in the processing pipeline.

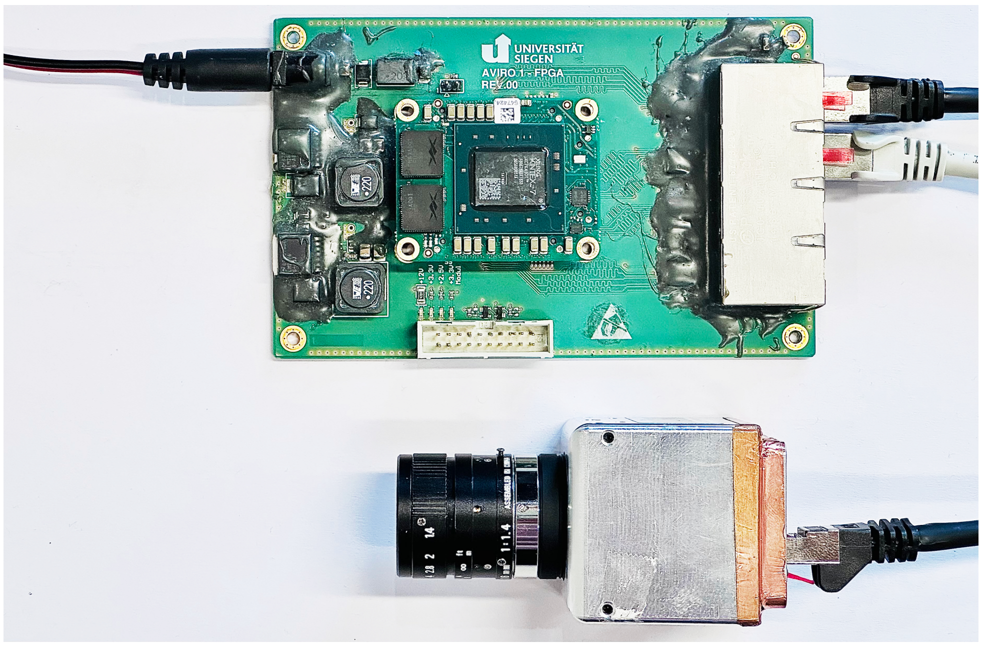

**Figure 1.** FPGA-board setup with a Xilinx 410 + industrial camera.

The DoG module delivers four octaves, which are used by the modified SIFT to find relevant keypoints. The resulting SIFT output consists of a stream of descriptors with 1800 bits each, which contain information about the octave, the position (column/row) in the picture, and the gradient vectors. Each keypoint descriptor has 17 vectors with 8 values each. The SIFT algorithm was slightly adjusted to the requirements of the FPGA and simplified without significantly losing accuracy. Floating point implementations in the FPGA fabric are immensely resource-costly and thus were replaced by fixed points. This SIFT adaptation is described in more detail in [26].

The flow between the FPGA modules is shown in Figure 2, which gives a short overview of the processing chain. The GigE Vision camera driver module communicates with the camera and processes the incoming udp packages, retrieves the camera data from those packages, and sends them to the Bayer pattern module. The Bayer pattern module receives a stream of camera data and resolves the images from it, which are sent to the difference of Gaussian module. This processes the images and sends the image pyramid with different octaves to the SIFT module. The SIFT module processes the image pyramid with different octaves for each picture and sends keypoints and its features with corresponding data (on which octave of the image the keypoint was found, and in which column/row, and the descriptor) to the feature matching module. The feature matching module matches the keypoints and sends pairs of keypoints with their feature information to the ethernet stack. There, the data are sent out via the udp over an ethernet interface.

The matching algorithm is based on the nearest-neighbor approach, where an item is matched against a complete set of other items to find the best matching pair. It uses an FPGA-suited implementation of the Manhattan distance between two pairs of vectors. Matches between pairs, whose distance is above an adjustable threshold, are discarded. This threshold ensures that falsely matched pairs are rejected. It was found through iterative optimization to determine which threshold yields the most accurate results. The quality of the found matches was evaluated to determine which threshold ranges were either too low or too high. An ideal threshold should not filter out matches that are true positives and filter out matches that are false positives. The resulting threshold value ensures that the amount of false positives that are filtered out exceeds the number of filtered-out true positives. The latter happens to be erroneously sorted out when the threshold for acceptable matches is set too low.

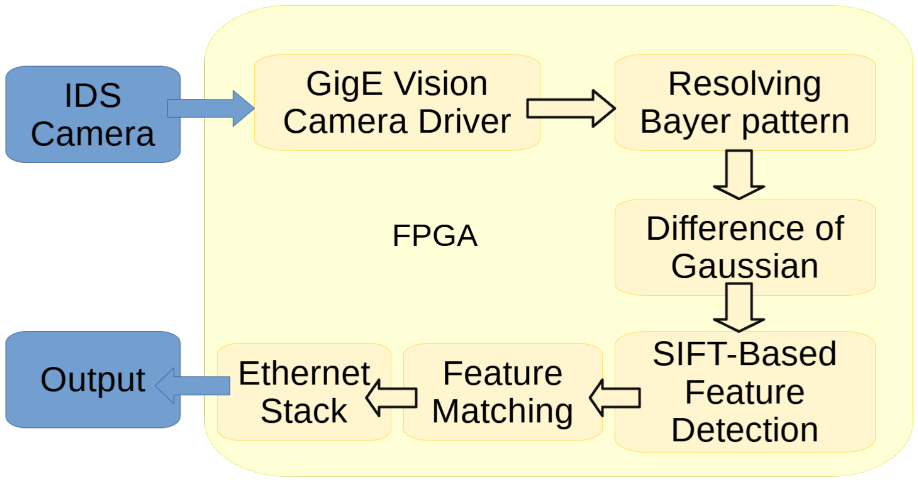

**Figure 2.** Overview of the FPGA modules and their workflow.

The descriptor consists of 1800 bits for each feature. It contains the data about the image coordinates of the keypoint and the described 17 vectors. This results in a large amount of data for several hundred to thousand keypoints, which need to be managed in the FPGA fabric. Those operations need to be parallelized in order to process images at an acceptable speed. However, parallelizing algorithms on the FPGA means that the parallel calculations take up more resources and more (already limited) space on the FPGA. This results in a trade-off between the available resources and the performance. The speed of the matching process is, therefore, limited by the amount of available resources on the FPGA, which will be explained in more detail in a later section.

The incoming features from the SIFT module are stored in the matching module until the complete set of features from the first image is stored. Depending on the settings of the module, which will be explained in the context of the performance and resource utilization, there is a limit to how many features per picture are stored. To reduce the number of stored features, those keypoints that are in close proximity to already stored keypoints are discarded. This optimization allows us to match features without requiring too many resources in the FPGA.

The keypoints from the first picture are then matched with the incoming keypoints of the next picture to find the best matching pair. This is repeated for every incoming keypoint of the second picture until its keypoints are completely processed and matched. When one picture is fully processed, its features will stay in storage to be compared with the features of the next pictures.

Therefore, each picture's features become matched first with the features of the preceding picture and then with the features of the succeeding picture. This approach allows us to track the position of a keypoint across multiple pictures.

Regarding the memory management within the FPGA, the incoming features of picture$_n$ that are matched with the features of picture$_{n\text{-}1}$ will overwrite the features of picture$_{n\text{-}2}$, as shown in Figure 3. Therefore, there are never more than two entire sets of features stored at the same time.

The algorithm that matches the features is based on a nearest-neighbor approach. Two different metrics were considered for this: the Euclidean distance and the Manhattan distance. Both metrics with their FPGA-suitable adaptations are described in the following.

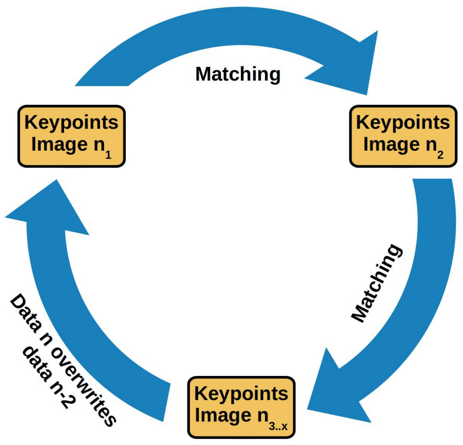

**Figure 3.** For memory optimization, features are saved in a ring buffer while their keypoints are being matched.

The first part describes the Euclidean distance between the vectors of the feature descriptors (vectors p, q, with a vector size of 8). Vector q is part of the descriptor of picture$_n$ while p is part of the feature descriptor of picture$_{n\text{-}1}$. The Euclidian distance extracts a root, which is resource costly in FPGA implementations:

$$\sqrt{\sum_{i=0}^{7}(p_\mathrm{i}-q_\mathrm{i})^2}$$

Therefore, this process was simplified, as the exact Euclidian distance is not required, and the necessary information for the matching is which pair of points have the smaller distance. As the values that are summed up are squared, the sums are positive, which implies that if the root of the sum$_1$ is smaller than the root of sum$_2$, then sum$_1$ is smaller than sum$_2$.

$$\sqrt{\sum_{i=0}^{7}(p_\mathrm{i}-q_\mathrm{i})^2} < \sqrt{\sum_{i=0}^{7}(a_\mathrm{i}-b_\mathrm{i})^2}$$

implies that

$$\sum_{i=0}^{7}(p_\mathrm{i}-q_\mathrm{i})^2 < \sum_{i=0}^{7}(a_\mathrm{i}-b_\mathrm{i})^2$$

Therefore, the root extraction is not necessary for the FPGA adaption of our algorithm and was not implemented on the FPGA to save resources. This leaves the following sum for one descriptor vector:

$$\sum_{i=0}^{7}(q_\mathrm{i}-p_\mathrm{i})^2$$

These values are summed up for all 17 vectors that belong to one descriptor:

$$\sum_{i=0}^{16}\sum_{i=0}^{7}(q_\mathrm{i}-p_\mathrm{i})^2$$

Another possible approach is the Manhattan distance:

$$\sum_{i=0}^{7}|q_\mathrm{i}-p_\mathrm{i}|$$

The Manhattan distance works with the absolute differences between the elements of a vector. This absolute value can be determined, as shown in pseudocode Algorithm 1.

**Algorithm 1** Determining absolute difference between two elements of a vector

```
1: if $q_i > p_i$ then
2:     $q_i - p_i$
3: else
4:     $p_i - q_i$
5: end if
```

This leaves the following calculation for 17 vectors of two matched features:

$$\sum_{i=0}^{16}\sum_{i=0}^{7}|q_i - p_i|$$

Both the approach based on Euclidean distance and the Manhattan distance were evaluated and implemented in the FPGA. The Manhattan distance utilizes fewer resources and was implemented in the final version of the matching module. Based on those calculations, the resulting values are compared to find the lowest value that determines the matching keypoint. For each keypoint in picture$_{n-1}$, the keypoint in picture$_n$ whose matching yielded the smallest sum, is marked as the best matching keypoint. This is depicted in Algorithm 2. This code was subdivided into small, primitive operations to be synthesizable by the FPGA synthesize tool. The best matching keypoints are paired together in a data structure and stored in the FPGA. When all keypoints of one image are matched, the matching pairs are sent as an output via the ethernet interface.

**Algorithm 2** Matching-module in pseudocode

```
1: $featureLimit \leftarrow 490$          ▷ Constant: limits feature per image
2: $setRadius \leftarrow 3$          ▷ Constant: radius outside of which to look for the next feature
3: $setThreshold \leftarrow x$          ▷ Discard matches with distance over this threshold
4: $featureCount \leftarrow 0$          ▷ Count: features per image
5: $lastOctave \leftarrow 0$          ▷ Stores the octave of the last feature
6: while featureStream.IsNotEmpty do
7:     featureStream.getFeature
8:     if feature.Octave < lastOctave then          ▷ Feature belongs to new picture
9:         send out keypoint matches from $img_{n-1}$
10:        featureCount = 0
11:    else
12:        featureCount = featureCount + 1
13:    end if
Require: featureCount < featureLimit
Require: $feature_{img\ n}$.ColumnRow > RadiusPreviousFeature
14:    for $Feature_i$ in $img_{n-1}$ do
15:        $distance \leftarrow 0$
16:        for $Vector_j$ in $Feature_{img\ n}$ do
17:            distance = distance+CalculateManhattanDistance($(Feature_{img\ n})_j$,$(Feature_{img\ n-1})_{ij}$)
18:        end for
19:        if distance > setThreshold then
20:            discardMatch
21:        else
22:            saveMatch
23:        end if
24:    end for
25:    linearSearch          ▷ Find match with lowest distance
26:    saveBestMatch          ▷ gets send out when picture is complete
27: end while
```

In summary, the following changes made it possible to place the image processing system, including the resource-intensive keypoint finding and keypoint matching module on the Kintex410 FPGA:

- Adapting the SIFT to be more suited for an FPGA implementation;
- Optimizing the memory management in order to keep as little information as is necessary within the FPGA storage;
- Optimizing the coding of the matching algorithm to break it down into a large number of small operations, to be easily and resourcefully synthesizable by the FPGA synthesis tool;
- Limiting the amount of keypoints per image optimizes the resources to achieve matching five images per second;
- Basing the matching on the Manhattan distance, which computes with very little FPGA resources

Both the Euclidean distance and the Manhattan distance were considered. However, the Euclidean distance uses more resources within the FPGA fabric. This, in itself, is an even greater problem, as the calculations of the distances between the feature vectors are parallelized. This parallelization within an FPGA means that the resources, like DSP slices, need to be multiplied by the number of parallel operations. This was not a feasible option for our design, as 71% of the DSP slices are taken by the DoG and SIFT operations.

## 4. Results

The resulting image processing system processes incoming camera data, resolves images from that data with a resolution of 2048 $\times$ 1088 px, detects SIFT keypoints in real-time, and subsequently matches those points on up to five images per second. For the experimental results in this section, we tested both the SIFT individually, as shown in Figure 4, and the whole image pipeline with the SIFT and the matching module implemented together on the Kintex-7 410 FPGA, as shown in Figure 5.

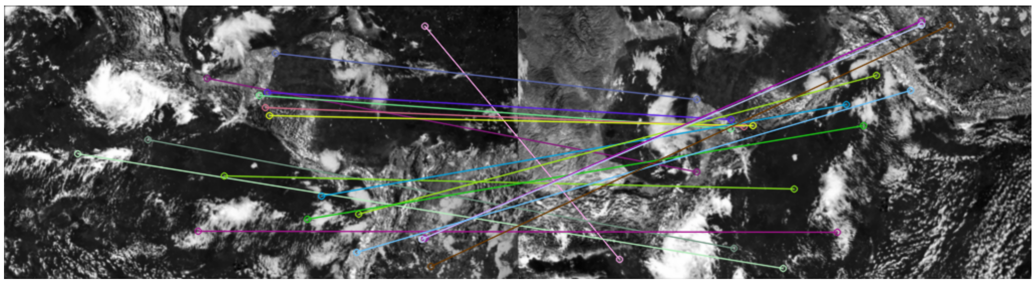

**Figure 4.** Results of the SIFT keypoint detection on the original image (**a**) and rotated image (**b**) with matched keypoints in early development stages matched on PC with OpenCV matching.

The adapted SIFT implementation on the FPGA takes 17.83 ms to detect keypoints on one image. This is a significant speedup compared to our tests on the same resolution (2048 $\times$ 1088 px), with an implementation of the OpenCV SIFT (380 ms) on an Intel workstation with an i7 processor.

The SIFT implementation, as part of the complete image processing pipeline, has a different speed compared to when it is tested as a single module. This is because the pre-processing (receiving camera images, the Bayer pattern, and the DoG module) increases the overall latency. In the complete architecture, we achieve feature detection on one image within 33 ms. Table 2 shows a comparison between the performance of related works and our feature detection. As our architecture works on a higher resolution, it is notable that our implementation reaches one pixel per clock cycle (8 ns) and, therefore, outperforms similar works.

**Table 2.** Comparing results among feature detections in FPGA hardware.

| Parameter | Proposed SIFT Module | SIFT in [10] | FAST+BRIEF in [11] | SIFT+LES in [19] |
|---|---|---|---|---|
| ms | 17.83/33 | 31 | 18 | 50 |
| Resolution | 2048 × 1088 | 640 × 480 | 640 × 480 | 256 × 256/800 × 480 |

Regarding the performance of the resulting module for the matching algorithm, the space on the FPGA is limited. Both a higher parallelity and a higher number of keypoints that are to be matched require a larger amount of resources. Regarding the placement of the module, parallelizing and pipelining the process becomes increasingly difficult for different reasons that lie in the possibilities of placing and routing resources on the FPGA. Therefore, it is important to weigh the speed of the module against the desired number of keypoints. This is why our module is adaptable and can be set to store more than 3000 keypoints per picture. Storing more keypoints per picture comes at the cost of a slower performance as the resources on the FPGA are not sufficient to perform this task at a high parallelity without needing a vast amount of resources. For the use of the module as part of the image processing chain, a lower number of keypoints per picture need to be determined in order to place the matching module together with the other image processing modules on the Kintex-7 FPGA. The image processing chain without the matching module takes up 86.18% of the LUT on the Kintex-7, 51.9% FF (flip flops), 51.46% BRAM and 71% DSP (digital signal processors). With the FPGA operating at 125 Mhz, this is already a challenging design to place and route.

Therefore, placing the matching module on the FPGA was also made possible by making adjustments in the memory storage and management. As a result, the FPGA can send out image data and keypoints separately, but not joined together, as the image does not stay in storage while the keypoints are being matched. After an image is processed by the SIFT module, only its resulting output is stored in the FPGA in order to match their keypoints. As the keypoints come in from the SIFT module, they are matched against the keypoints from the previous image, shortening the number of cycles during which image information is stored, further saving resources.

Additionally, as we wanted to match five pictures per second, the matching module in this implementation can be set to store only a specifically limited amount of features per image, eliminating clusters where a lot of features are close together on neighboring pixels. To implement this change, we had a closer look at the amount of keypoints per picture that are generated by the keypoint detection module. On pictures that do not have large amounts of textured surfaces, the SIFT module will, with the threshold it is operated on, generally not extract more than 1000 features per image. In addition, many of the keypoints are in close proximity to each other, correlate to the same object. Between storing nearly redundant keypoints, and considering the limited amount of RAM the Kintex410 FPGA offers and the steep incline of FPGA resources that more parallel computations require, we found that 490 features for each picture are ideal for the strict resource limits that our design requires. It allows the design to match five pictures per second while only needing a small amount of LUT resources. Additionally, the BRAM resources are a point of consideration. With the same speed of five pictures per second, the module needs only 18% of the available 18k-BRAM when matching 450 features per image, but it needs twice as much (36%) of the available BRAM when set to 490 features per image. Summed up with the 51% that the rest of the image processing chain needs, this gets close to 90% BRAM utilization. Matching more features per image would lead to complications when all the modules are implemented together on the FPGA. Therefore, the implementation of the module in our design was set to 490 features per picture. This allows for a resource-saving module that needs only a small amount of resources and can be placed on the FPGA as part of the complete image processing chain.

Figure 5 shows a result of the FPGA matching module (for better visibility, only the keypoints in the center of the images are displayed). Figure 5b) was rotated approximately 5 degree and scaled down to 75% size.

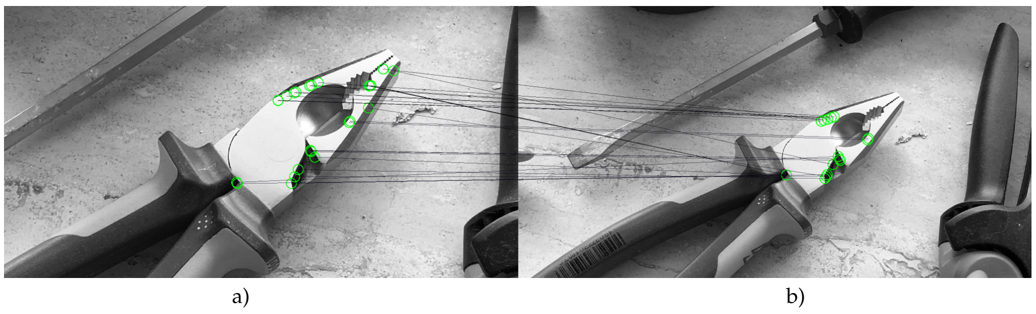

**Figure 5.** Result of the FPGA matching module between the (**a**) scene and (**b**) scaled and rotated scene.

Setting the module to match 490 keypoints per image makes the module resource-saving and allows the module to be placed on the FPGA as part of the image processing pipeline, which overall utilizes 90% of the LUTs in the Kintex410 FPGA, as Table 3 shows. The image processing chain is a stand-alone module that needs no further input other than the camera data. Therefore, it is adaptable and produces an output that can be used by any other application that uses the ethernet interface. The FPGA architecture runs with 125 mHz. In future applications, the matching needs to be implemented on a radiation-hardened NG-ULTRA FPGA. To evaluate the performance of this FPGA, a smaller part of the image processing chain, the module that resolves the Bayer pattern on the incoming camera data, was implemented on the NG-Ultra FPGA. The result is shown in Table 3 as well.

**Table 3.** Resource utilization of the whole image pipeline, the matching module, and the Bayer module.

| Image Pipeline on Kintex410 | Matching on Kintex410 | Bayer-Module on Kintex410 | Bayer-Module on NG-Ultra |
|---|---|---|---|
| 61% BRAM | 36% 18K-BRAM | 6 BMEM, 11 DMEM | 12 NX-RAM |
| 57,548 FF | 2781 FF | 1726 LUT | 2514 LUT |
| 90% LUT | 5% LUT | ~0.68% LUT | ~0.43% LUT |

The amount of needed resources on the NanoXplore NG-Ultra was compared to the resources the same module needs on the Xilinx Kintex410. The Bayer module was implemented on the Kintex410 with proprietary components such as the BRAM component, which had to be replaced by proprietary components the NanoXplore Impulse development tool offers. The resulting data were utilized to evaluate the feasibility of implementing the proposed image processing pipeline on the radiation-hardened NG-Ultra FPGA. In regards to the amount of resources that an implementation on the NG-Ultra FPGA needs compared with an implementation on the Kintex410 device, implementing more modules on the NG-Ultra appears feasible.

## 5. Discussion

In this paper, an architecture was presented that retrieves and processes images to find matching pairs of keypoints. It implements the extraction of keypoints based on the SIFT algorithm, and the matching between corresponding keypoints based on Euclidian distance. The whole pipeline is implemented completely within the FPGA fabric with no additional computational hardware. This image pipeline executes the following steps: the communication with a camera and the retrieval of live images, resolving the camera's Bayer pattern, a DoG algorithm, a SIFT feature extraction on a 2048 $\times$ 1088 px picture, and lastly the feature matching. This work presents a novelty as it implements keypoint detection and matching on high-resolution pictures on an FPGA as a stand-alone system without the need for additional or on-board processing units.

The SIFT implementation reaches a speed of 33 ms while up to 490 features per image can be matched across up to five pictures per second. The results show that the matching module is limited by the amount of resources the other processing modules need—the largest of this being the SIFT module. Other feature extraction methods need fewer resources but are less robust. Thus, future work could focus on feature extraction algorithms like SURF or ORB if it will become necessary to upscale the amount of features that can be matched within a reasonable time frame.

The system is implemented on a Xilinx Kintex 410, which significantly reduces the costs compared to approaches with Xilinx Virtex and Zynq-FPGA boards. For space missions, the Xilinx Kintex 410 represents a commercial off-the-shelf (COTS) solution and can be used in short time space missions and non-critical applications. Our goal for the future is to make the system usable in long-term missions and critical applications. For these use cases, radiation-hard FPGAs are needed. So our future research will focus on implementing the image processing chain on a radiation-hardened FPGA. Here, a few points have to be taken into consideration when implementing our modules on this new hardware. One of these is the amount of resources that differ between the various FPGA models. Another point is the FPGA architecture with its clocking system, which varies depending on the FPGA model and manufacturer. This might necessitate further changes in the implementation of the modules.

A first step was taken to evaluate the feasibility of implementing the proposed image processing chain on a radiation-hardened FPGA. The Bayer module was implemented on both a Xilinx Kintex-410 and a NanoXplore NG-Ultra FPGA. The results show that the same module needs more LUT resources in the NG-Ultra FPGA, but at the same time, the FPGA offers a larger total amount of LUTs. Percentage-wise, the modules need a lower amount of LUT in the NanoXplore NG-Ultra (∼0.43% compared with ∼0.68% on the Kintex-410 FPGA). Based on the estimation that is offered by implementing the same module on both FPGAs and comparing the amount of needed resources, it is manageable to reconstruct the whole image processing chain on a NanoXplore FPGA. Challenges with the clocking architecture did not arise, as the design met its timing requirements; however, as mentioned above, implementing more modules and reaching a higher degree of resource utilization might lead to new challenges.

**Author Contributions:** Conceptualization, methodology, software, validation, S.B.; writing—original draft preparation, S.B.; writing—review and editing, J.K. and K.-D.K.; supervision and project administration, K.-D.K.; funding acquisition, K.-D.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** The project is funded under the DLR, German aerospace research and technology centre, and the Federal Ministry for Economic Affairs and Climate Action (BMWK), funding references 50RA2210A and 50RA2401B, from 1 November 2022 to 30 April 2027.

**Data Availability Statement:** The datasets presented in this article are not readily available, as this project is part of ongoing research. Requests to access the datasets should be directed to sina.timmermann@uni-siegen.de

**Acknowledgments:** We would like to thank the reviewers for their time and effort and their insightful comments.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

| | |
|---|---|
| BRAM | Block RAM |
| BRIEF | Binary Robust Independent Elementary Feature |
| BRISK | Binary Robust Invariant Scalable Keypoint |
| COTS | Commercial Off The Shelf |
| DLR | German Space Agency (In German: Deutsche Luft- und Raumfahrt) |
| DoG | Difference of Gaussian |
| DSP | Digital Signal Processor |
| FAST | Features from Accelerated Segment Test |
| FPGA | Field Programmable Gate Array |
| FPS | Frames Per Second |
| LUT | Look-Up Tables |
| ORB | Oriented Fast and Rotated Brief |
| SIFT | Scale-Invariant Feature Transform |
| SLAM | Simultaneous Localization And Mapping |
| SURF | Sped Up Robust Features |
| UDP | User Datagram Protocol |

## References

1. Lowe, D.G. Object recognition from local scale-invariant features. In Proceedings of the Seventh IEEE International Conference on Computer Vision 1999, Corfu, Greece, 20–25 September 1999. . [CrossRef]
2. Prokop, K.; Połap, D. Heuristic-Based Image Stitching Algorithm with Automation of Parameters for Smart Solutions. *Expert Syst. Appl.* **2024**, *241*, 122792. [CrossRef]
3. Borse, J.H.; Patil, D.D. Empirical Analysis of Feature Points Extraction Techniques for Space Applications. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 81–87. [CrossRef]
4. Hong, S.; Shin, H.-S. Comparative Performance Analysis of Feature Detection and Matching Methods for Lunar Terrain Images. *J. Korean Soc. Civ. Eng.* **2020**, *40*, 437–444. [CrossRef]
5. Krämer, M.-S.; Hardt, S.; Kuhnert, K.-D. Image Features in Space-Evaluation of Feature Algorithms for Motion Estimation in Space Scenarios. In Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods 2018, Madeira, Portugal, 16–18 January 2018; pp. 300–308. [CrossRef]
6. Bonato, V.; Marques, E.; Constantinides, G.A. A Parallel Hardware Architecture for Scale and Rotation Invariant Feature Detection. *IEEE Trans. Circuits Syst. Video Technol.* **2008**, *18*, 1703–1712. [CrossRef]
7. Chang, L.; Hernández-Palancar, J.; Sucar, L.E.; Arias-Estrada, M. FPGA-Based Detection of SIFT Interest Keypoints. *Mach. Vis. Appl.* **2013**, *24*, 371–392. [CrossRef]
8. de Lima, R.; Martinez-Carranza, J.; Morales-Reyes, A.; Cumplido, R. Accelerating the Construction of BRIEF Descriptors Using an FPGA-Based Architecture. In Proceedings of the International Conference on ReConFigurable Computing and FPGAs (ReConFig) 2015, Riviera Maya, Mexico, 7–9 December 2015; pp. 1–6. [CrossRef]
9. de Lima, R.; Martinez-Carranza, J.; Morales-Reyes, A.; Cumplido, R. Improving the Construction of ORB through FPGA-Based Acceleration. *Mach. Vis. Appl.* **2017**, *28*, 525–537. [CrossRef]
10. Yao, L.; Feng, H.; Zhu, Y.; Jiang, Z.; Zhao, D.; Feng, W. An Architecture of Optimised SIFT Feature Detection for an FPGA Implementation of an Image Matcher. International Conference on Field-Programmable Technology, Sydney, NSW, Australia, 9–11 December 2009; pp. 30–37. [CrossRef]
11. Heo, H.; Lee, K.-Y. FPGA Based Implementation of FAST and BRIEF Algorithm for Object Recognition. *IEEE Int. Conf. IEEE Reg.* **2013**, *17*, 202–207. [CrossRef]
12. Kuo, C.-H.; Huang, E.-H.; Chien, C.-H.; Hsu, C.-C. FPGA Design of Enhanced Scale-Invariant Feature Transform with Finite-Area Parallel Feature Matching for Stereo Vision. *Electronics* **2021**, *10*, 1632. [CrossRef]
13. Sanchez, H.A.; George, A.D. A Streaming Hardware Architecture For Real-time Sift Feature Extraction. In Proceedings of the International Conference on Field-Programmable Technology (ICFPT) 2021, Auckland, New Zealand, 6–10 December 2021; pp. 1–9. [CrossRef]
14. Lee, D.-J.; Fuller, S.G.; McCown, A.S. Optimization and Implementation of Synthetic Basis Feature Descriptor on FPGA. *Electronics* **2020**, *9*, 391. [CrossRef]
15. Fularz, M.; Kraft, M.; Schmidt, A.; Kasiński, A. A High-performance FPGA-based Image Feature Detector and Matcher Based on the Fast and Brief Algorithms. *Int. J. Adv. Robot. Syst.* **2015**, *12*, 141. [CrossRef]
16. Rublee, E.; Rabaud, V.; Konolige, K.; Bradski, G. Orb: An efficient alternative to SIFT or surf. In Proceedings of the 2011 International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; pp. 2564–2571. [CrossRef]

17. Peng, J.Q.; Liu, Y.H.; Lyu, C.Y.; Li, Y.H.; Zhou, W.G.; Fan, K. FPGA-based parallel hardware architecture for SIFT algorithm. In Proceedings of the 2016 IEEE International Conference on Real-time Computing and Robotics (RCAR) 2016, Angkor Wat, Cambodia, 6–10 June 2016; pp. 277–282. [CrossRef]
18. Li, S.-A.; Wang, W.-Y.; Pan, W.-Z.; Hsu, C.-C.J.; Lu, C.-K. FPGA-based hardware design for scale-invariant feature transform. *IEEE Access* **2018**, *6*, 43850–43864. [CrossRef]
19. Chien, C.-H.; Chien, C.-J.; Hsu, C.-C. Hardware-software co-design of an image feature extraction and matching algorithm. In Proceedings of the 2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS) 2019, Singapore, 28 February 2019–2 March 2019; pp. 37–41. [CrossRef]
20. Daoud, L.; Latif, M.K.; Jacinto, H.S.; Rafla, N. A fully pipelined FPGA accelerator for scale invariant feature transform keypoint descriptor matching. *Microprocess. Microsyst.* **2020**, *72*, 102919. [CrossRef]
21. Vourvoulakis, J.; Kalomiros, J.; Lygouras, J. FPGA accelerator for real-time sift matching with RANSAC support. *Microprocess. Microsyst.* **2017**, *49*, 105–116. [CrossRef]
22. Belmessaoud, N.M.; Bentoutou, Y.; Chikr El-Mezouar, M. FPGA implementation of feature detection and matching using Orb. *Microprocess. Microsyst.* **2022**, *94*, 104666. [CrossRef]
23. Zhang, J.; Xiong, S.; Liu, C.; Geng, Y.; Xiong, W.; Cheng, S.; Hu, F. FPGA-based feature extraction and tracking accelerator for real-time visual slam. *Sensors* **2023**, *23*, 8035. . [CrossRef] [PubMed]
24. Ichikawa, Y.; Shioda, A.; Kawamura, K.; Chu, T.V.; Motomura, M. An accurate FPGA-based orb feature extractor for slam with row-wise keypoint selection. In Proceedings of the 2024 IEEE International Conference on Consumer Electronics (ICCE) 2024, Las Vegas, NV, USA, 6–8 January 2024; pp. 1–2. [CrossRef]
25. Kreowsky, P.; Stabernack, B. A full-featured FPGA-based pipelined architecture for SIFT extraction. *IEEE Access* **2021**, *9*, 128564–128573. [CrossRef]
26. Hardt, S.; Krämer, M.-S.; Kuhnert, K.-D. Real-Time Implementation of a feature detection algorithm for usage in space applications. In Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS 2018), Madrid, Spain, 4–6 June 2018.