Universitetet i Stavanger

Feature Descriptors: Comparison of SIFT, FAST + BRIEF and ORB

PROJECT REPORT

In the **ELE510** course of the master program **Computational Engineering / Computer Science – Data Science**
at Universitetet i Stavanger

by

René König

Atanu Das

Fatema Tuz Zohora

13th November 2022

Student numbers	268127, 268128, 267981
Course	ELE510
Supervisor	Luca Tomasetti

Abstract

Three methods for detection and description of features are discussed and implemented in python using OpenCV: SIFT, FAST + BRIEF and ORB. Their behaviour regarding scale and rotation of features as well as time to detect and describe the features is observed across different cases.

The results of the experiments are in alignment with the expectations from the literature studies. FAST + BRIEF is the fastest method, however, does not provide scale and rotational information on the features. ORB is the second fastest while solving these issues, whereas SIFT provides more coherent results.

Table of Contents

List of Figures

List of Abbreviations

BRIEF	Binary Robust Independent Elementary Features
cv2	OpenCV library in python
des	feature descriptor
FAST	Features from Accelerated Segment Test
kp	keypoint
ORB	Oriented FAST and Rotated BRIEF
SIFT	Scale Invariant Feature Transform
SLAM	Simultaneous localization and mapping

1 Introduction

Feature detectors and descriptors are used to find features in an image and describe them, which is useful for a lot of computer vision applications. Some of these applications include object recognition and tracking, 3D object reconstruction and Robot Navigation, including Simultaneous localization and mapping (SLAM) [1], [2].

1.1 Problem Definition

A feature detector only detects features, providing their location in an image, while a feature descriptor describes those features according to their neighbourhood, which makes them useful e.g., for keypoint matching. A feature descriptor requires feature detection to be performed on an image first, hence it is hardly possible to talk about feature descriptors without mentioning their feature detectors. [1]

While one of the primary uses of feature descriptors is keypoint matching, this topic is beyond the scope of this project.

However, different methods for feature description (and detection) will be discussed and compared in terms of how they work, what results they provide and how they perform in terms of detecting and describing features and the time they take to do so.

Two factors to differentiate between algorithms for feature detection and description are scale invariance and rotational invariance. Scale invariance means that the detector can detect features at different scales. Rotational invariance means that the descriptor can describe a feature regardless of its orientation.

2 Methods

The methods discussed in this project are Scale Invariant Feature Transform (SIFT), Features from Accelerated Segment Test (FAST), Binary Robust Independent Elementary Features (BRIEF) and Oriented FAST and Rotated BRIEF (ORB).

They are implemented in Python using the OpenCV (cv2) library.

2.1 SIFT

SIFT is a combined feature detector and descriptor. SIFT is scale and rotationally invariant. SIFT consists of the following parts: scale-space peak detection, localization of key points, orientation assignment and key point descriptor [3].

First, an image's scale space is created by convoluting a Gaussian kernel (blurring) at different scales with the original image. The number of octaves in scale-space depends on the size of the source image. The result of the blurred image is:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

Where G is a gaussian blur operator, I is the image and σ is the scale parameter [4]. From these blurred images, Difference of Gaussians (DoG) is created as an approximation of Laplacian of Gaussian (LoG) to identify the key points.

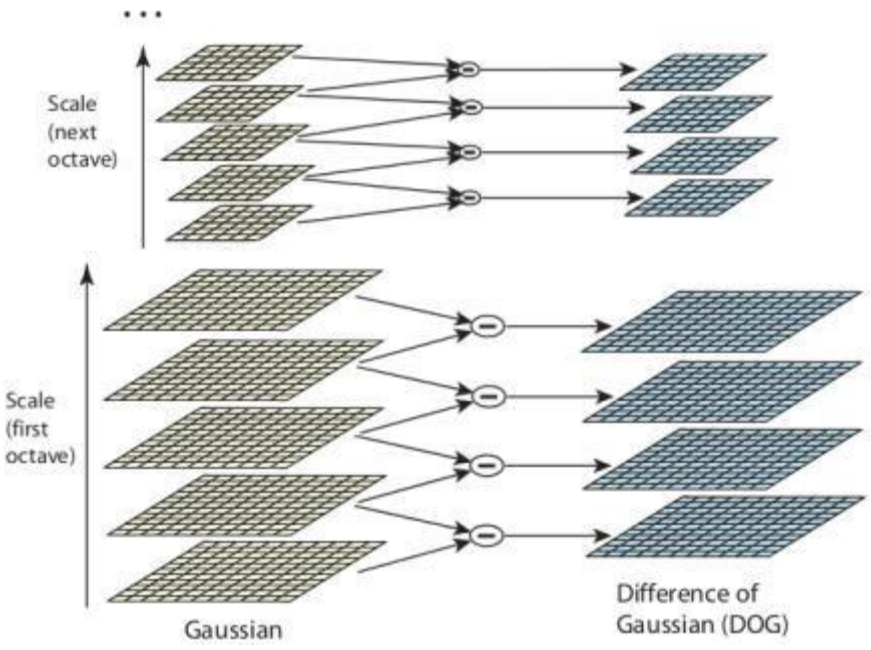

Figure 1: Difference of Gaussian [3]

In DoG the blurred images are subtracted from each other, as depicted in Figure 1. The DoGs are stacked on top of each other.

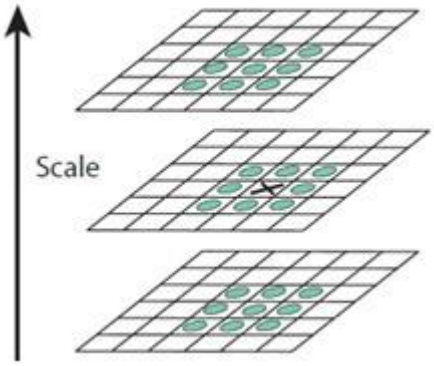

Figure 2: Finding Local Extrema in Scale Space [3]

As shown in Figure 2, local extrema in the scale space are calculated to find keypoints.

Contrast thresholding, Taylor series expansion of scale space and Hessian matrix are used to select the most accurate keypoints to represent a feature.

To calculate the descriptors, the local neighborhood is divided into small cells, as marked in red in Figure 3, and the gradient of these cells are computed. A histogram on the gradient orientation is used to find the primary direction of a cell.

The primary directions of 4x4 of these cells are summarized in an 8-bin orientation histogram, with each bin representing an angle of 45°, as shown in Figure 3

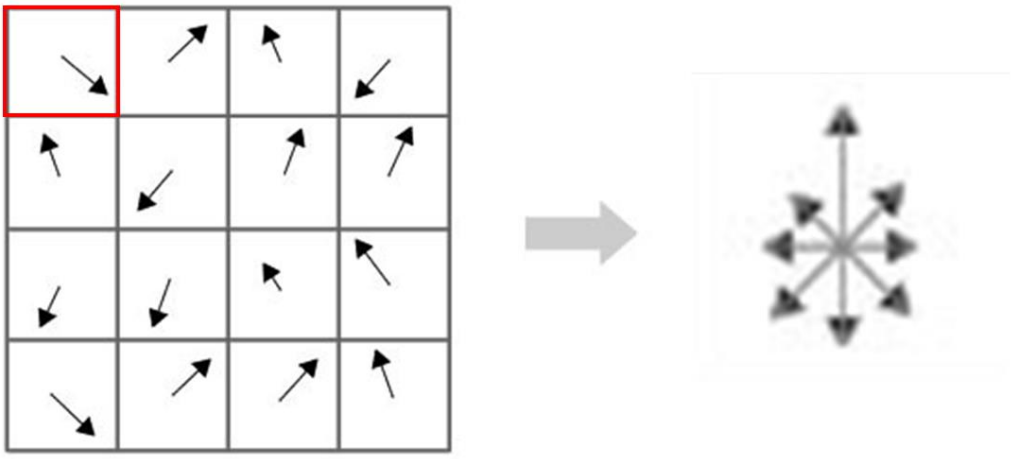

Figure 3: Orientation Histogram Creation (based on [4])

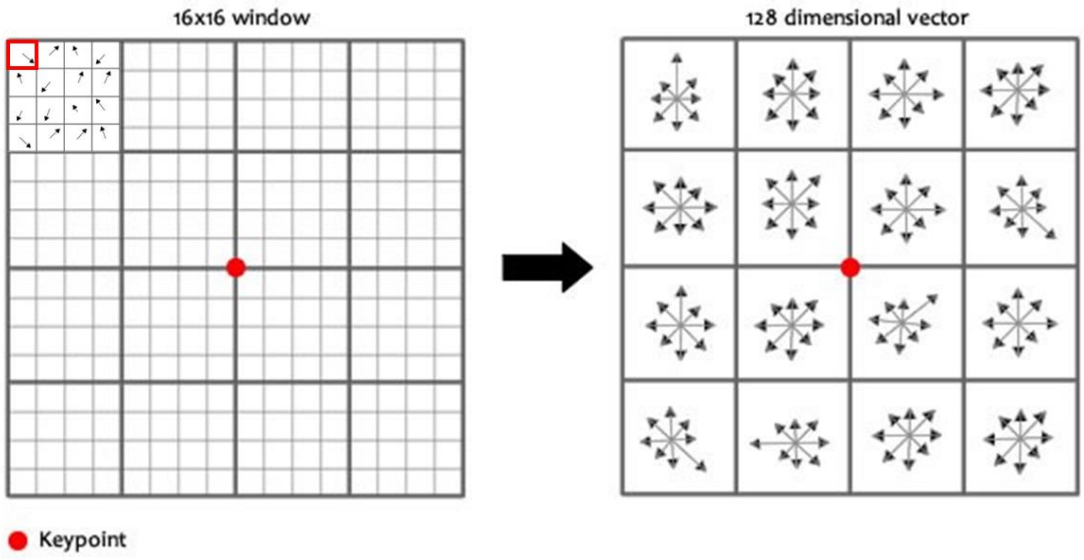

Figure 4: Creation of Feature Vector for Keypoint Descriptor (based on [4])

The previous step is repeated for 16 windows around the keypoint, as shown in Figure 4, to create a 128-dimensional feature vector which forms the keypoint descriptor.

2.2 FAST

FAST is an algorithm for corner detection, which can be used as a feature detector. [2], [5]

As an initial step, from an image, any pixel, p, is selected with its intensity value.

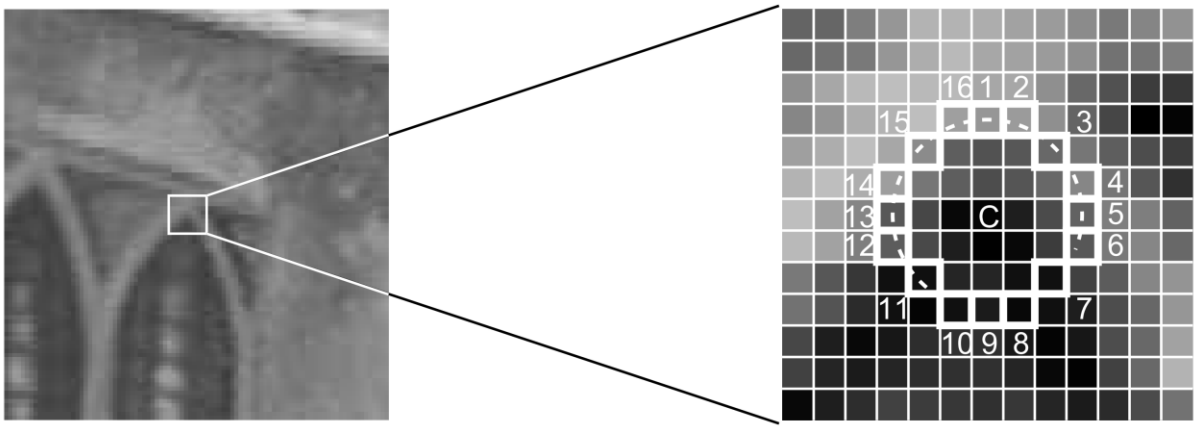

Figure 5: FAST Feature Detection [5, p. 8]

As shown in Figure 5, a circle of 16 pixels is considered around the pixel of interest. The pixel will be considered a corner if there are n contiguous pixels in the circle either brighter or darker than the intensity of the pixel of interest with the added or subtracted threshold value respectively. The authors of this algorithm selected the value of n as 12. [5, p. 7]

A comparison of the intensity between the pixel selected and the pixels 1, 5, 9, and 13 of the circle can be conducted as a first approximation to speed up the algorithm. Among these four points, if three of them satisfy the threshold criterion, then the pixel selected is a point of interest.

If the pixel is a point of interest, then the computation is performed on all the 16 pixels on the circle to detect if it is a feature or not, otherwise, the algorithm discards the pixel.

For feature detection, there are several good options available, nevertheless, in a real-time application, several of them (like SIFT, etc.) are not fast enough for application. FAST was developed to overcome this situation [6].

The algorithm is fast as it discards those pixels which are not points of interest at the first iteration of the algorithm and only computes on all 16 pixels of the circle for the pre-selected pixels of interest.

However, FAST does not operate as expected on computer-generated images, in which sharp edges align perfectly with the axes of the coordinate system. This is because it cannot find 12 contiguous darker or brighter pixels in the circle around the corner pixel. As a solution, the image needs to be blurred, preferably using Gaussian filter, to make the corners of these images detectable [2].

2.3 BRIEF

BRIEF is a feature descriptor that needs a separate feature detection algorithm to detect the features first.

The image is pre-smoothed to reduce noise sensitivity and increase stability, using a Gaussian kernel [7]. The neighbourhood of a feature will be converted into a binary feature vector of n = 128-512 bits.

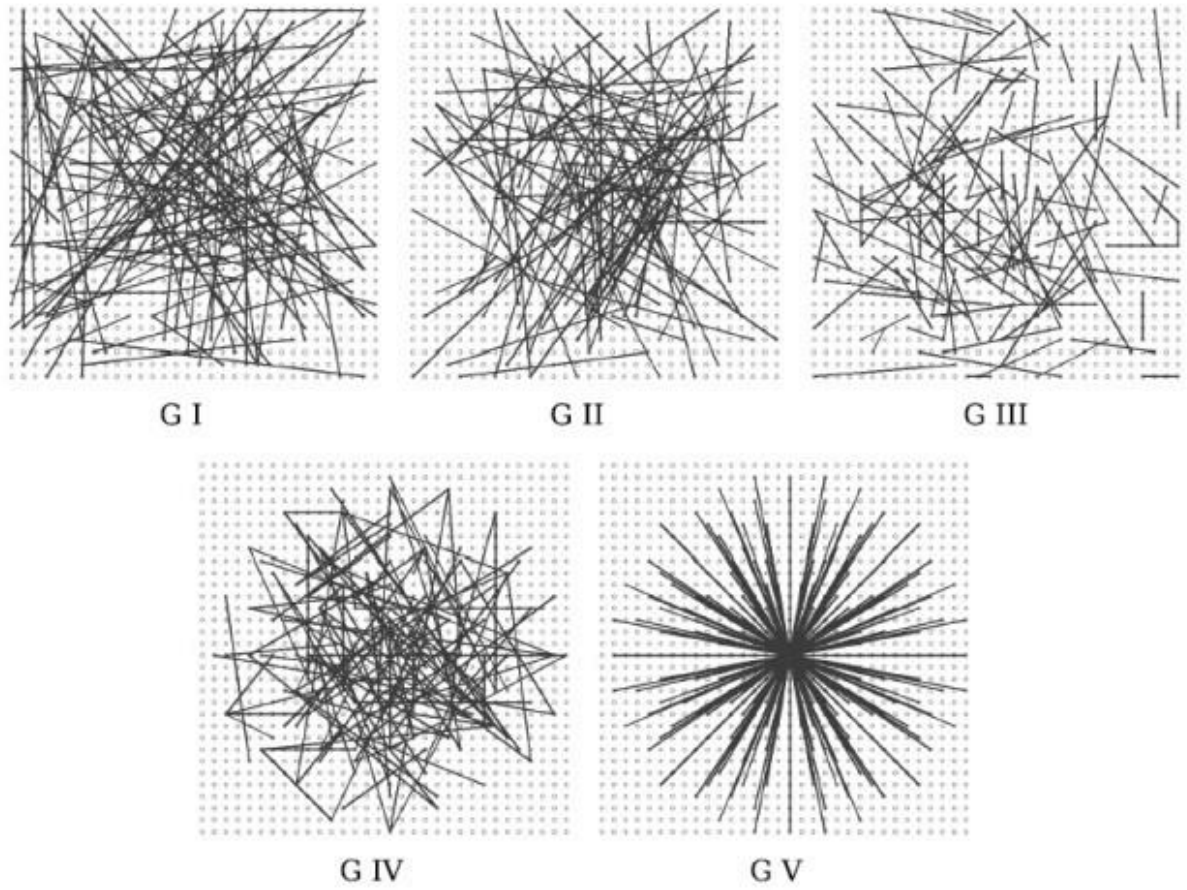

Figure 6: Sampling Geometries [8, p. 5]

For this a set of n test pairs are selected by one of the sampling geometries shown in Figure 6. On these a binary test is implemented to define the differences in intensities of the selected pixel pair. The binary test returns 0 or 1, depending on which pixel of the pixel pair has the higher intensity.

The result of the binary test for each of the n pixel pairs is combined in a binary string to create the feature vector or descriptor.

Because no information on orientation is considered in BRIEF it is not rotationally invariant. However, the binary string allows fast feature comparison using hamming distance, which is just applying XOR and bit count on the binary strings of two features.

2.4 ORB

In ORB FAST is used to detect features on multiple scales of a scale pyramid. The Harris Corner measure is applied to find the strongest of these multiscale features, making ORB scale invariant. [9]

An orientation based on the intensity weighted centroid of the neighbourhood is assigned, as depicted in Figure 7.

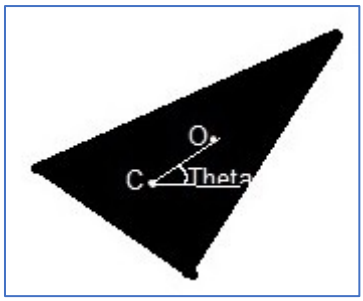

Figure 7: Intensity Weighted Centroid [10]

The angle θ of the vector from the center of the corner O to centroid C gives the primary orientation of the feature. The sampling geometry used in BRIEF is rotated according to θ to align with the feature orientation before applying BRIEF, making ORB rotationally invariant. [10]

3 Implementation

To implement the algorithms in python the full package of the computer vision library OpenCV `opencv-contrib-python`, which contains both main modules and contrib/extra modules, is used.

3.1 SIFT

```python
def SIFT(img,show=True,nfeatures=500,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS):
    # Initiate SIFT detector/descriptor
    sift = cv2.SIFT_create(nfeatures)

    # Find the keypoints and compute the descriptors with SIFT
    kp, des = sift.detectAndCompute(img, None)

    if show == True:
        imgWkeypoints = cv2.drawKeypoints(img,kp,None,flags=flags)
        cv2.imshow("SIFT",imgWkeypoints)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return kp, des
```

Figure 8: Code Snippet Implementation of SIFT

Figure 8 shows the implementation of SIFT in python using `cv2`. First the SIFT feature detector and descriptor is initialised using `cv2.SIFT_create`. Then the `kp` are detected and their `des` computed using `sift.detectAndCompute`.

The primary parameter to adjust is the number of features to detect (`nfeatures`).

The implementation is based on [3] and [4].

3.2 FAST + BRIEF

```python
def FASTBRIEF(img,thresh=10,show=True,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS):
    # Initiate FAST detector
    fast = cv2.FastFeatureDetector_create(threshold=thresh)
    # Initiate BRIEF descriptor/extractor
    brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()

    # find the keypoints with FAST
    kp = fast.detect(img,None)
    # compute the descriptors with BRIEF
    kp, des = brief.compute(img, kp)

    if show == True:
        imgWkeypoints = cv2.drawKeypoints(img,kp,None,flags=flags)
        cv2.imshow("FAST + BRIEF",imgWkeypoints)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return kp, des
```

Figure 9: Code Snippet Implementation of FAST + BRIEF

Figure 9 shows the implementation of FAST and BRIEF in python using `cv2`. First the FAST feature detector and BRIEF feature descriptor are initialised using `cv2.FastFeatureDetector_create` and `cv2.xfeatures2d.BriefDescriptorExtractor_create` respectively. Then the keypoints (`kp`) are detected using `fast.detect` and their descriptors (`des`) are computed using `brief.compute`.

The primary parameter to adjust is the threshold (`thresh`) for the FAST detector. The implementation is based on [6], [11], [2] and [7].

3.3 ORB

```python
def ORB(img,show=True,nfeatures=500,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS):
    # Initiate ORB detector/descriptor
    orb = cv2.ORB_create(nfeatures)

    # Find the keypoints and compute the descriptors with ORB
    kp, des = orb.detectAndCompute(img, None)

    if show == True:
        imgWkeypoints = cv2.drawKeypoints(img,kp,None,flags=flags)
        cv2.imshow("ORB",imgWkeypoints)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return kp, des
```

Figure 10: Code Snippet Implementation of ORB

Figure 10 shows the implementation of ORB in python using `cv2`. First the ORB feature detector and descriptor is initialised using `cv2.ORB_create`. Then the `kp` are detected and their `des` computed using `orb.detectAndCompute`.

The primary parameter to adjust is the number of features to detect (`nfeatures`).

The implementation is based on [9] and [10].

4 Testing, Analysis and Results

The algorithms are compared across different example images. Two of these cases are described and discussed in this section, other examples are available in the provided jupyter notebook.

4.1 Simple User Manual

```
def compare(imgNames,imgPath='./images/', rotate=False, angle='random',
            blurFAST=False,FASTthresh=1,nfeaturesORB=64,nfeaturesSIFT=16,show=False):
```

Figure 11: Code Snippet Comparison

As shown in Figure 11, a function called `compare` has been implemented to compare the different algorithms. This function takes a list of image names (`imgNames`) to perform the comparison on, as well as the primary parameters for the respective algorithms. Note that the same parameter for each algorithm will be used on all supplied images. It returns lists of the images and the respective keypoints and descriptors found by each algorithm.

It is possible to blur the image before applying the FAST algorithm using the parameter `blurFAST`.

It is also possible to `rotate` the images, either by a provided or a random `angle`, and perform the same operations as on the non-rotated images.

```
def plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,
                    SideBySide=False,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS):
```

Figure 12: Code Snippet Plot Comparison

As shown in Figure 12, a function called `plot_comparison` has been implemented to plot the results of `compare`. This function takes the lists of the images and the respective keypoints found by each algorithm to plot.

It is possible to plot the images either above each other or side-by-side using the parameter `SideBySide`.

The flag `cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS` shows the scale and orientation of the found keypoints for ORB and SIFT. Note that BRIEF does not provide this information.

The flag can be replaced by `cv2.DrawMatchesFlags_DEFAULT` to not show this information for ORB and SIFT, which can help to de-clutter the plot, if only the positions of the detected features are of interest.

4.2 Case 1 Chessboard

```
imgNames = ['chessboard.png']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess = compare(imgNames,
                                                                                      blurFAST = True,
                                                                                      FASTthresh = 1,
                                                                                      nfeaturesORB = 64,
                                                                                      nfeaturesSIFT = 16)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

Figure 13: Code Snippet Setup Chessboard

Figure 13 Shows the setup for case 1, a computer-generated chessboard with sharp, edges parallel to the x- and y-axis. Because of these sharp, orthogonal edges the image needs to be blurred before applying the FAST algorithm, otherwise it does not detect any corners. The threshold for FAST is set to 1 and the number of features to detect for ORB and SIFT to 64 and 16 respectively.

4.2.1 Result Comparison

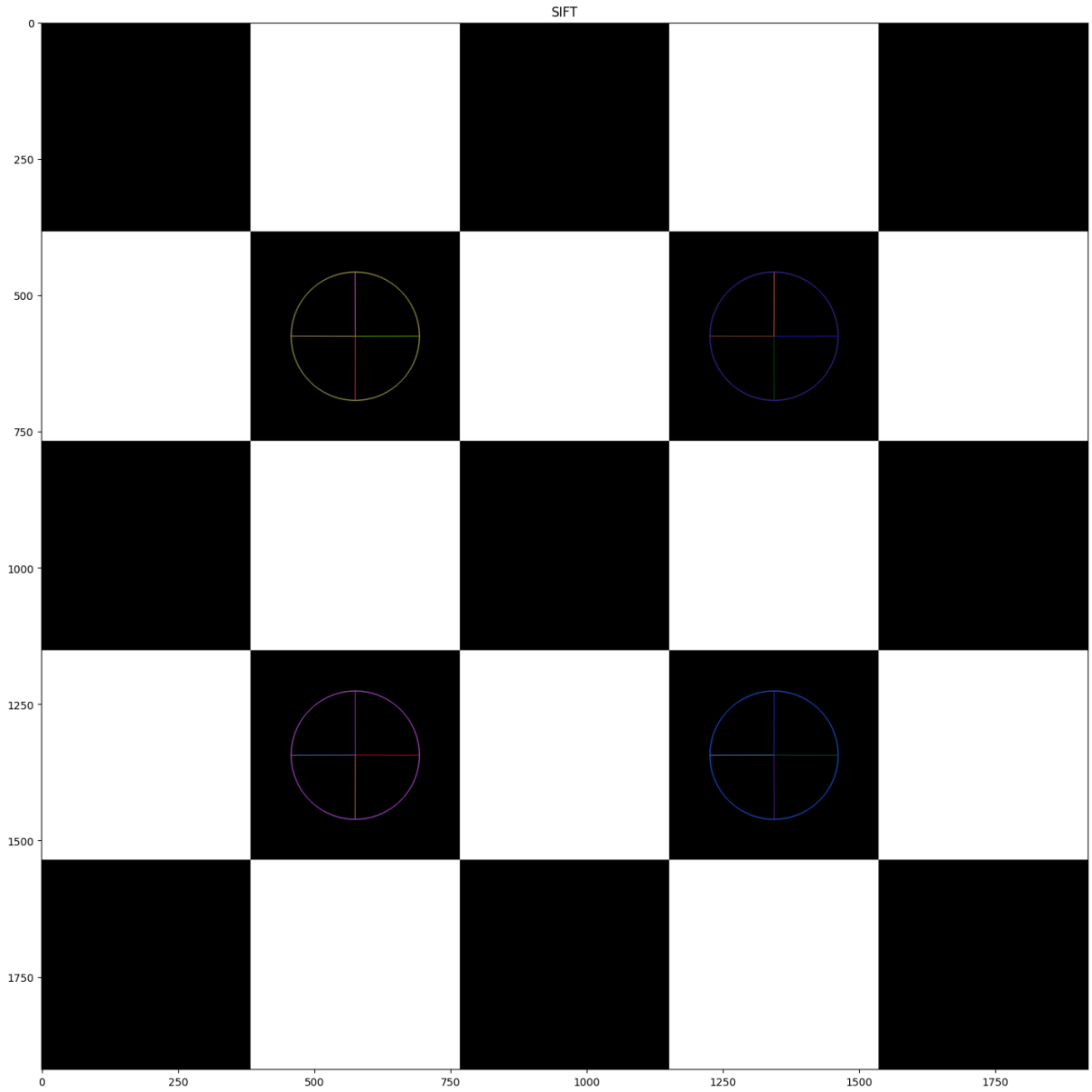

Figure 14: Chessboard SIFT

Figure 14 shows the features and descriptors from SIFT. Because of its scale invariance, SIFT detects the four central black squares as global features, placing the keypoints in the center of these squares. Because the number of features to detect is set to 16 we see four features in the center of each square, each slightly offset from the others, with the orientation vector pointing towards the strongest gradient, being the nearest edge of the square.

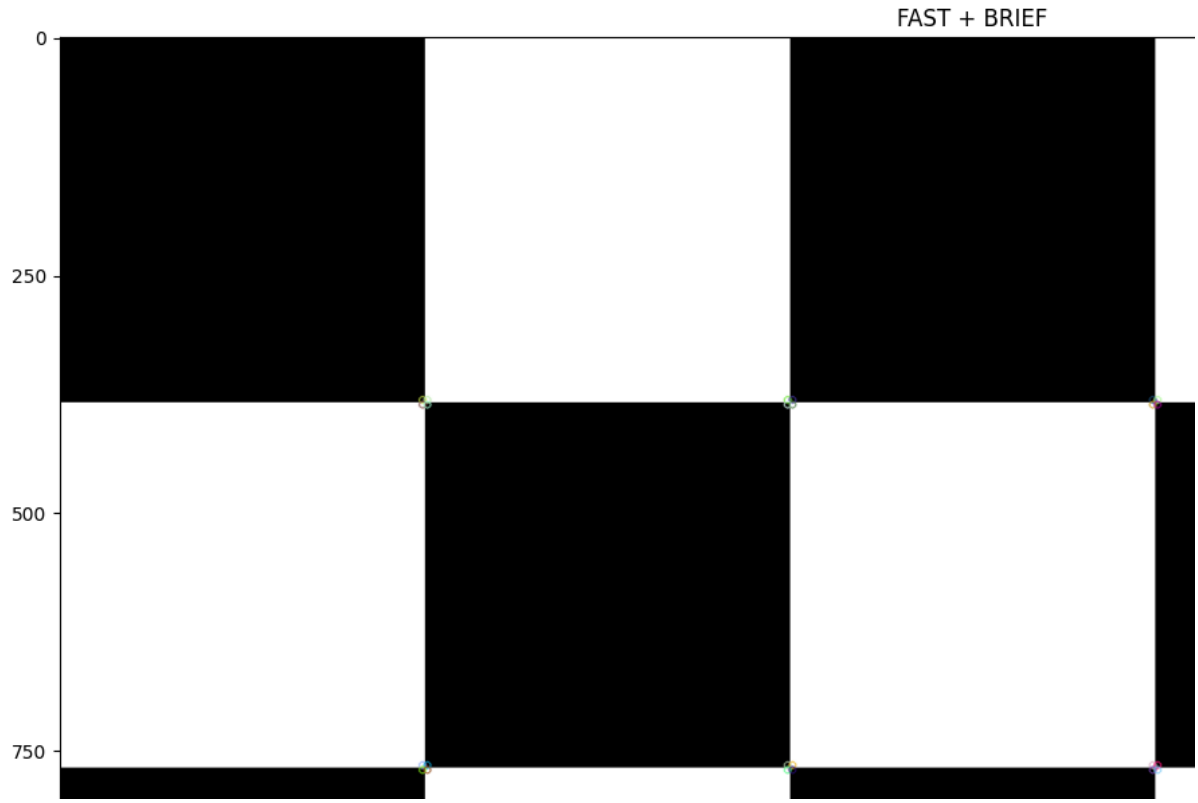

Figure 15: Chessboard FAST + BRIEF (Section)

Figure 15 displays a section the features detected on the chessboard by FAST. The features are represented as small circles and can be seen in the corners of each square. Since FAST detects the corners, there are four features at each crossing, each belonging to the corner of the respective black or white square.

Note how the descriptors of feature 0 and 4, which are displayed in Figure 16, both look like this: `[224 234 4 138 52 237 214 103 157 202 85 135 75 40 223 181 82 51 159 76 111 103 26 213 196 33 76 160 65 186 239 199]` . This is the binary, 32 byte long feature vector provided by BRIEF, with 8 bits summarized to 1 byte, represented in decimal.

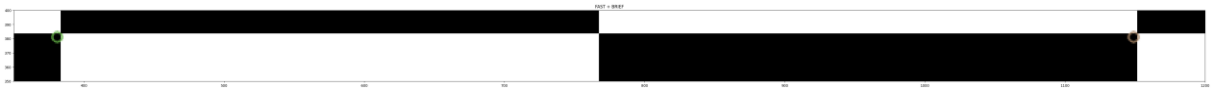

Figure 16: Chessboard FAST + BRIEF feature 0 and 4

Since feature 0 and 4 are both in the same position on the top-right corner of a black square, their surrounding neighbourhood looks the same and therefore the pixel-pairs which are compared by BRIEF result in the same feature vector.

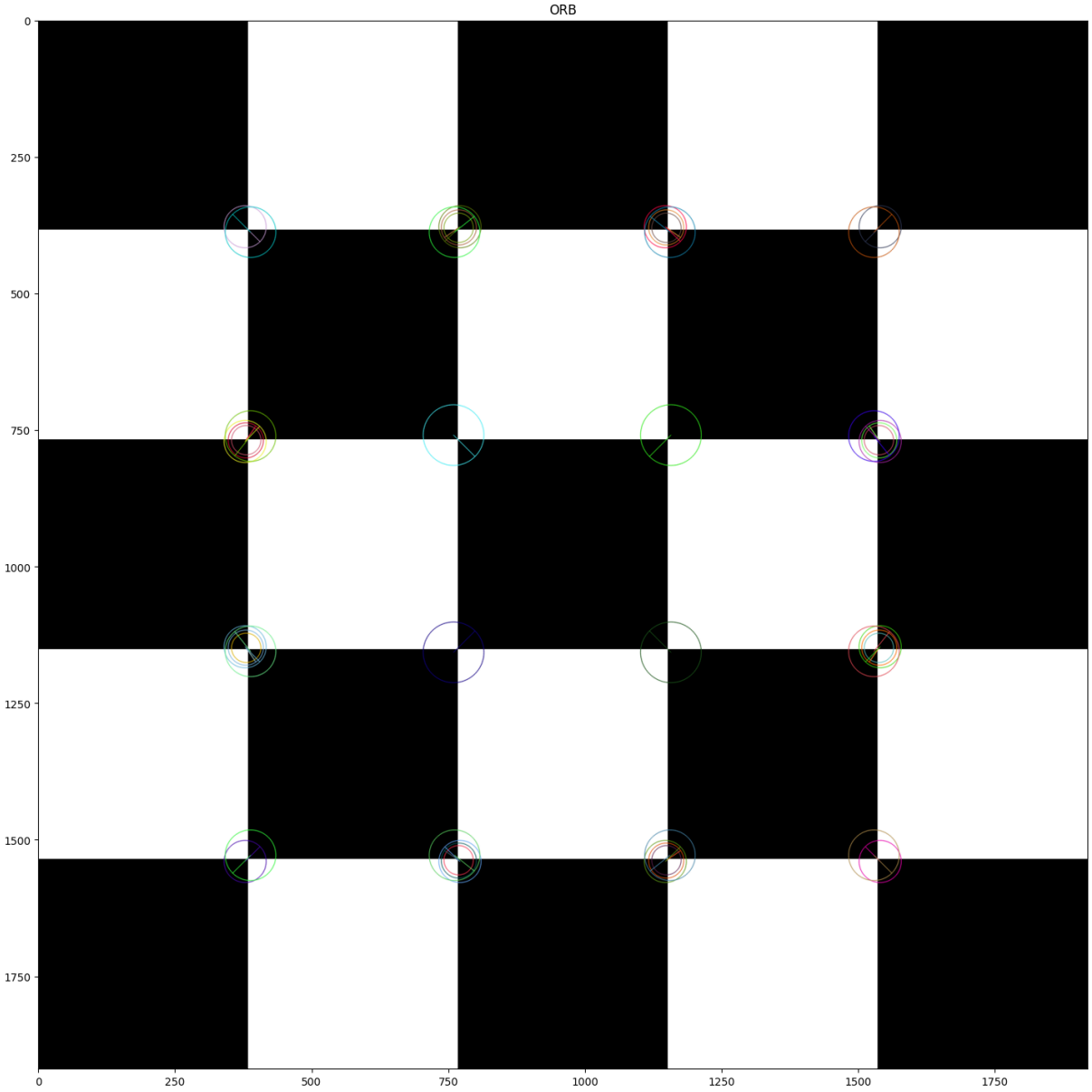

Figure 17: Chessboard ORB

Figure 17 shows the features and their descriptors from ORB, represented by circles of different sizes indicating the scale of the feature and a line representing the orientation of the feature. Note that since ORB is based on FAST the detected features are also located in the corners of the squares. Because of ORB´s scale invariance and selection of features based on Harris corner measure in the global context of the image, some corners may have multiple features while others only have one. From the four central features it is easily visible, that the orientation vector points to the center of intensity of the neighbourhood, away from the center of the black square in which´s corner the feature is located.

4.2.2 Performance Analysis (Time)

Using `%timeit` the following performance results (in terms of time per run) for this case have been found:

```
FAST + BRIEF:
7.02 ms ± 542 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
ORB:
27.9 ms ± 2.7 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
SIFT:
649 ms ± 82 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

FAST + BRIEF is clearly the fastest but lacks information on scale and orientation of the features. It is about 4x faster than ORB and 92x faster than SIFT.

ORB is still 23x faster than SIFT and provides information on scale and orientation of the features.

4.3 Case 2 Comet Neowise

Case 2 is a photograph of an illuminated glider on a grass field below a stary night sky with comet Neowise prominently above the horizon in the center of the frame, shown in Figure 18.

Figure 18: Comet Neowise (Original RGB)

The threshold for the FAST algorithm is set to 50 and the number of features to detect for ORB and SIFT to 3000. Note that the high number of features to detect is required by the many stars in the night sky.

Figure 19: Comet Neowise SIFT

Figure 19 shows that SIFT is detecting a lot of the small-scale features such as the stars in the background and grass blades in the foreground.

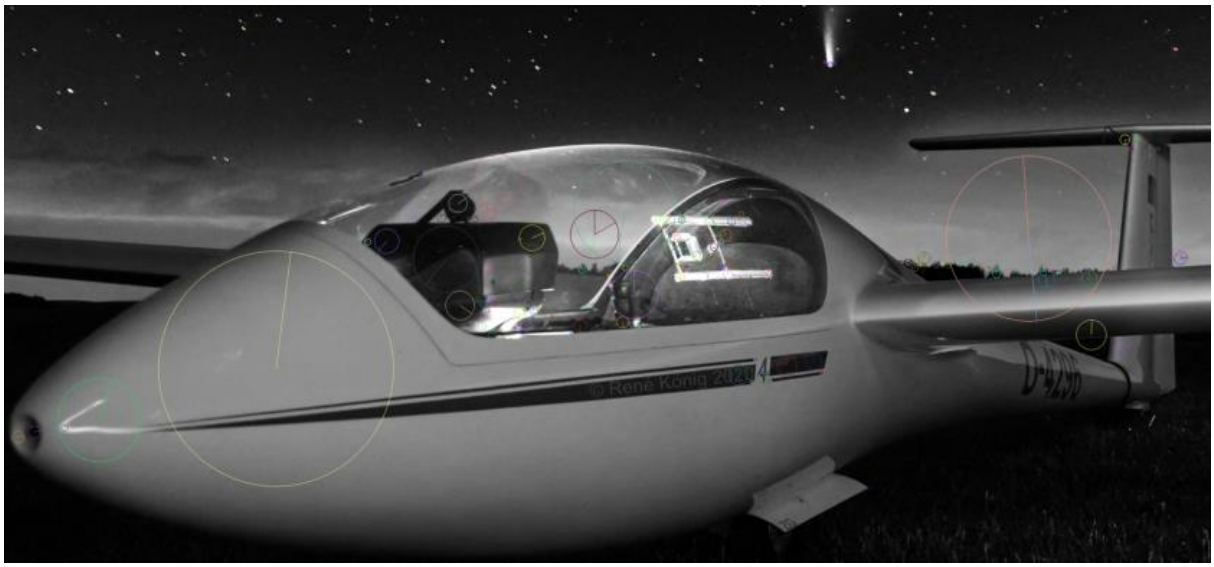

Figure 20: Comet Neowise SIFT (Zoomed In)

However, Figure 20 shows that SIFT is also detecting a lot of the higher scale features in and around the glider. The rotation vector is pointing towards the strongest gradient, which is especially visible in features like dark instrument panel in the cockpit in front of the bright horizon or the large-scale features detected between wing and the horizontal stabilizer, where the orientation vector is pointing towards the strong gradient alongside the sharp leading edge of the wing or the dark horizontal stabilizer in front of the illuminated horizon respectively.

Figure 21: Comet Neowise FAST + BRIEF

Figure 21 shows that FAST is detecting every star which has a high enough contrast to the background to pass the defined threshold. The features are visible as small colored points on top of the stars. Similarly, it also detects some of the illuminated blades of grass in the foreground, which is a bit easier to see in Figure 22.

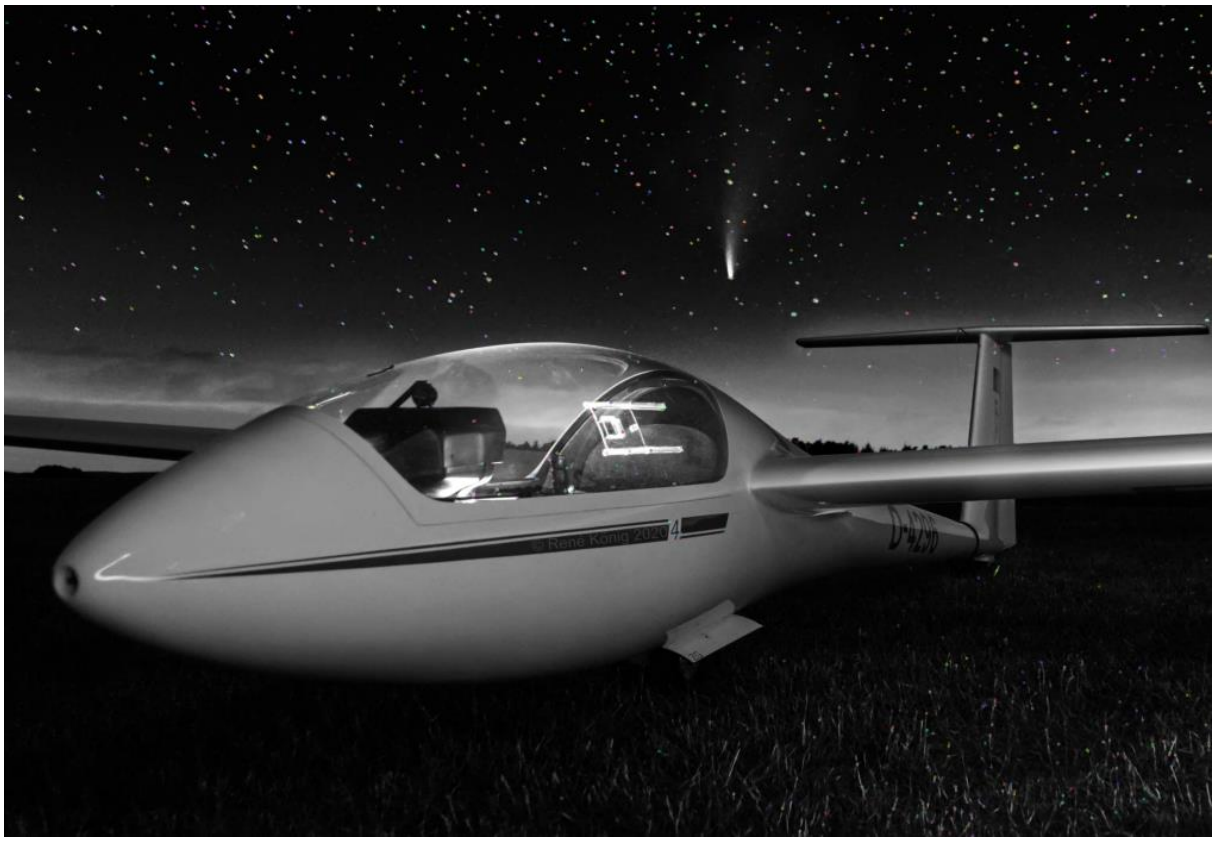

Figure 22: Comet Neowise FAST + BRIEF (Zoomed In)

FAST is also able to detect some of the sharper, high contrast corners on the glider such as the illuminated window in the canopy, which are visible in Figure 22, but struggles with softer, low-contrast or higher scale features on the glider such as the nose or the wing-root.

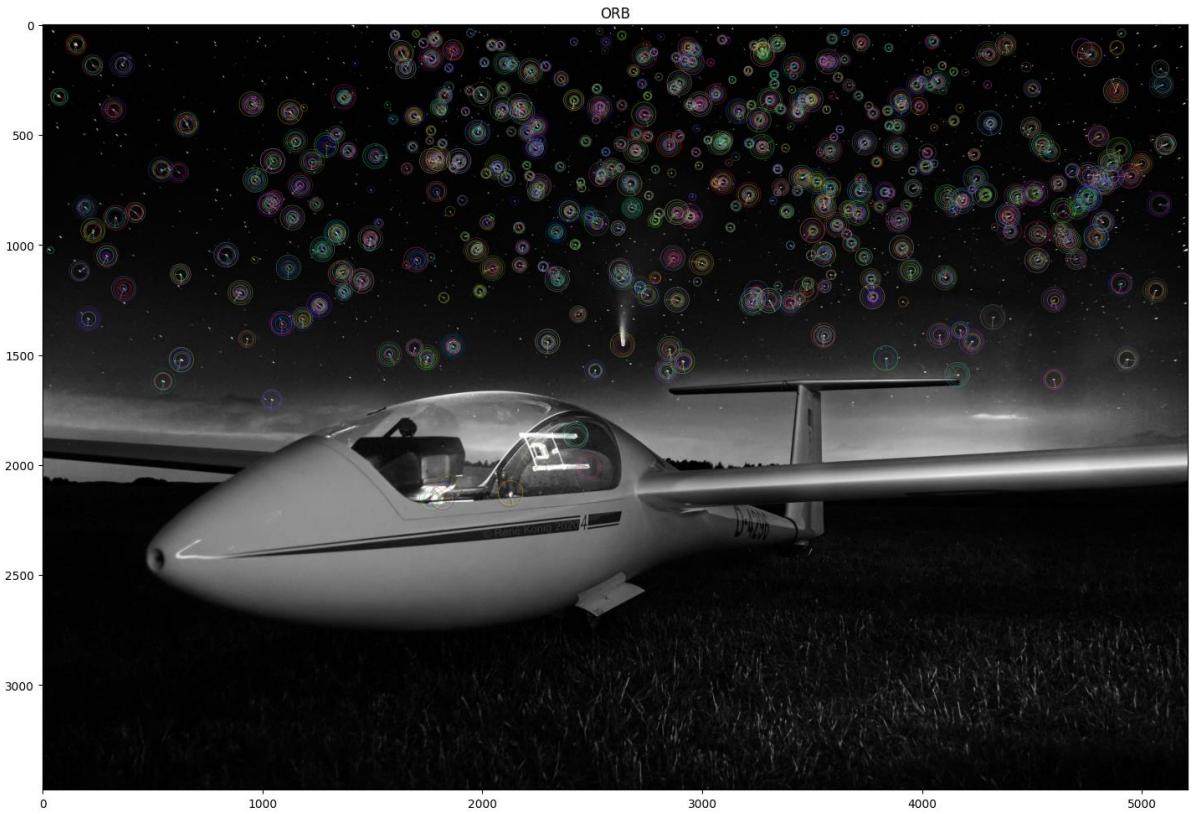

Figure 23: Comet Neowise ORB

Figure 23 shows that ORB is also detecting a lot of the high-contrast stars, but some of the lower contrast ones as well as the grass blades in the foreground are ruled out by the Harris corner measure.

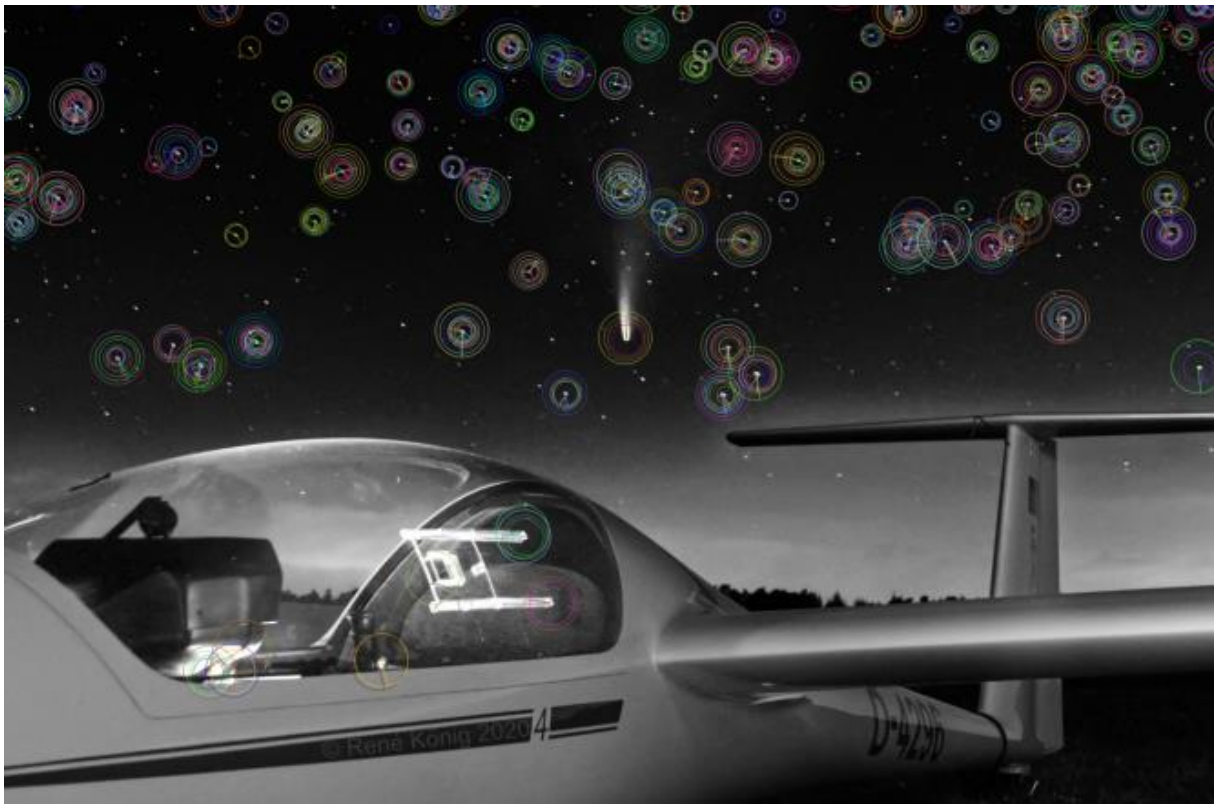

Figure 24: Comet Neowise ORB (Zoomed In)

Figure 24 shows that the orientation vector of the features points to the center of intensity, clearly visible in case of the bright tail of the comet or the illuminated edges of the window frame in the canopy. The orientation vectors of stars which are lower on in the sky primarily point downwards, towards the illuminated horizon or glider, while for stars higher in the sky point towards the nearest stars or star clusters, depending on the scale of the feature.

5 Discussion

In this project we have implemented and examined three methods for detection and description of features: SIFT, FAST + BRIEF and ORB and observed their behaviour in regard to scale and rotation of features as well as time to detect and describe the features.

5.1 Conclusion

While the adequately named FAST + BRIEF method is the fastest of the discussed methods to detect and describe features, it is neither scale nor rotationally invariant and therefore does not provide information on the scale and rotation of a feature.
ORB solves both these problems and is still faster than SIFT. ORB´s results have not always been coherent and include a lot of multi-detections of features, which could however be caused by insufficient parameter tuning.
SIFT is by far the slowest of the compared methods, though it´s results have been more coherent than ORB´s. It should be noted that SIFT works quite differently, compared to ORB and FAST + BRIEF, which work more similarly, and therefore provides different results. The features detected and described in an image by different algorithms can therefore not always be expected to match up.

5.2 Limitations and Future Work

This project only focused on understanding some methods for feature detection and description without performing further tasks, like keypoint mapping, that use those features and their descriptors. Their real-world usefulness is therefore difficult to judge.
Only the primary parameter of each algorithm was tuned to match a given image. However, the algorithms have further parameters that could be adjusted to a given image to get better results.

5.3 Learning Experience

The work on this project has helped us understand the difference between feature detection and description. The research and experimental work have taught us how different algorithms for feature detection and description work, which results they provide and what their advantages and disadvantages are.

References

[1] D. Tyagi, “Introduction To Feature Detection And Matching,” Medium, 03 January 2019. [Online]. Available: https://medium.com/data-breach/introduction-to-feature-detection-and-matching-65e27179885d. [Accessed 11 November 2022].

[2] D. Tyagi, “Introduction to FAST (Features from Accelerated Segment Test),” Medium, 02 January 2019. [Online]. Available: https://medium.com/data-breach/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65. [Accessed 12 November 2022].

[3] OpenCV, “Introduction to SIFT (Scale-Invariant Feature Transform),” Doxygen, 12 November 2022. [Online]. Available: https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html. [Accessed 12 November 2022].

[4] D. Tyagi, “Introduction to SIFT(Scale Invariant Feature Transform),” Medium, 16 March 2019. [Online]. Available: https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40. [Accessed 12 November 2022].

[5] E. a. D. T. Rosten, “Machine Learning for High-Speed Corner Detection,” *Computer Vision –ECCV 2006,* pp. 430--443, 2006.

[6] OpenCV, “FAST Algorithm for Corner Detection,” Doxygen, 11 November 2022. [Online]. Available: https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html. [Accessed 11 November 2022].

[7] D. Tyagi, “Introduction to BRIEF(Binary Robust Independent Elementary Features),” Medium, 19 March 2019. [Online]. Available: https://medium.com/data-breach/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6. [Accessed 12 November 2022].

[8] M. Calonder, V. Lepetit, C. Strecha and P. Fua, “BRIEF: Binary Robust Independent Elementary Features,” in *Computer Vision - ECCV 2010, 11th European Conference on Computer Vision*, Heraklion, 2010.

[9] OpenCV, “ORB (Oriented FAST and Rotated BRIEF),” Doxygen, 12 November

2022. [Online]. Available: https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html. [Accessed 12 November 2022].

[10] D. Tyagi, “Introduction to ORB (Oriented FAST and Rotated BRIEF),” Medium, 01 January 2019. [Online]. Available: https://medium.com/data-breach/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf. [Accessed 12 November 2022].

[11] OpenCV, “BRIEF (Binary Robust Independent Elementary Features),” Doxygen, 12 November 2022. [Online]. Available: https://docs.opencv.org/3.4/dc/d7d/tutorial_py_brief.html. [Accessed 12 November 2022].

Appendix

A1: Complete Code and images for Different Cases (.zip file)

```
!pip install opencv-contrib-python --user
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```
def
SIFT(img,show=True,nfeatures=500,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINT
S):
    # Initiate SIFT detector/descriptor
    sift = cv2.SIFT_create(nfeatures)

    # Find the keypoints and compute the descriptors with SIFT
    kp, des = sift.detectAndCompute(img, None)

    if show == True:
        imgWkeypoints = cv2.drawKeypoints(img,kp,None,flags=flags)
        cv2.imshow("SIFT",imgWkeypoints)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return kp, des
```

```
def
FASTBRIEF(img,thresh=10,show=True,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOIN
TS):
    # Initiate FAST detector
    fast = cv2.FastFeatureDetector_create(threshold=thresh)
    # Initiate BRIEF descriptor/extractor
    brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()

    # find the keypoints with FAST
    kp = fast.detect(img,None)
    # compute the descriptors with BRIEF
    kp, des = brief.compute(img, kp)

    if show == True:
        imgWkeypoints = cv2.drawKeypoints(img,kp,None,flags=flags)
        cv2.imshow("FAST + BRIEF",imgWkeypoints)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return kp, des
```

```
def
ORB(img,show=True,nfeatures=500,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS
):
    # Initiate ORB detector/descriptor
    orb = cv2.ORB_create(nfeatures)

    # Find the keypoints and compute the descriptors with ORB
    kp, des = orb.detectAndCompute(img, None)

    if show == True:
        imgWkeypoints = cv2.drawKeypoints(img,kp,None,flags=flags)
        cv2.imshow("ORB",imgWkeypoints)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return kp, des
```

```
def compare(imgNames,imgPath='./images/', rotate=False, angle='random',
            blurFAST=False,FASTthresh=1,nfeaturesORB=64,nfeaturesSIFT=16,show=
False):
    #Initialise lists to store results for multiple images
    imgs = []
    rotimgs = []
    fastbrief_kps = []
    fastbrief_dess = []
    fastbrief_rotkps = []
    fastbrief_rotdess = []
    orb_kps = []
    orb_dess = []
    orb_rotkps = []
    orb_rotdess = []
    sift_kps = []
    sift_dess = []
    sift_rotkps = []
    sift_rotdess = []

    #for-loop to handle multiple images (with the same parameters)
    for ind, imgName in enumerate(imgNames):

        img = cv2.imread(imgPath+imgName,cv2.IMREAD_GRAYSCALE)

        #blur images befor applying FAST if the edges are too sharp
        if blurFAST == True:
            img_forFAST = cv2.GaussianBlur(img,ksize=(11,11),sigmaX=0)
```

```
        else:
            img_forFAST = img

        #find keypoints and calculate their descriptors using different
algorithms
        fastbrief_kp, fastbrief_des = FASTBRIEF(img_forFAST,thresh =
FASTthresh,show=show)
        orb_kp, orb_des = ORB(img,nfeatures = nfeaturesORB,show=show)
        sift_kp, sift_des = SIFT(img,nfeatures = nfeaturesSIFT,show=show)

        #append results to respective list
        imgs.append(img)

        fastbrief_kps.append(fastbrief_kp)
        fastbrief_dess.append(fastbrief_des)

        orb_kps.append(orb_kp)
        orb_dess.append(orb_des)

        sift_kps.append(sift_kp)
        sift_dess.append(sift_des)

        #perform same operations on rotated image, if desired
        if rotate == True:
            #rotate image
            h, w = img.shape[:2]
            cx, cy = (w // 2, h // 2)
            if angle == 'random':
                angle = np.random.randint(10,350)
            rotationM = cv2.getRotationMatrix2D([cx,cy],angle,1)
            rotimg = cv2.warpAffine(img,rotationM,(w,h))
            rotimg_forFAST = cv2.warpAffine(img_forFAST,rotationM,(w,h))
            rotimgs.append(rotimg)

            #find keypoints and calculate their descriptors using different
algorithms
            fastbrief_rotkp, fastbrief_rotdes =
FASTBRIEF(rotimg_forFAST,thresh = FASTthresh,show=show)
            orb_rotkp, orb_rotdes = ORB(rotimg,nfeatures =
nfeaturesORB,show=show)
            sift_rotkp, sift_rotdes = SIFT(rotimg,nfeatures =
nfeaturesSIFT,show=show)

            #append results to respective list
            fastbrief_rotkps.append(fastbrief_rotkp)
            fastbrief_rotdess.append(fastbrief_rotdes)
```

```
            orb_rotkps.append(orb_rotkp)
            orb_rotdess.append(orb_rotdes)

            sift_rotkps.append(sift_rotkp)
            sift_rotdess.append(sift_rotdes)

    if rotate == True:
        return imgs, rotimgs, fastbrief_kps, fastbrief_dess, fastbrief_rotkps,
fastbrief_rotdess, orb_kps, orb_dess, orb_rotkps, orb_rotdess, sift_kps,
sift_dess, sift_rotkps, sift_rotdess
    else:
        return imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess,
sift_kps, sift_dess
```

```
def plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,
                    SideBySide=False,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYP
OINTS):
    #for-loop to handle multiple images (with the same parameters)
    for ind, img in enumerate(imgs):
        plt.figure(figsize=(60,60))
        ImWKeypoints_fastbrief =
cv2.drawKeypoints(img,fastbrief_kps[ind],outImage=None,flags=flags)
        if SideBySide == True:
            plt.subplot(131)
        else:
            plt.subplot(311)
        plt.imshow(ImWKeypoints_fastbrief)
        plt.title('FAST + BRIEF')
        ImWKeypoints_orb =
cv2.drawKeypoints(img,orb_kps[ind],outImage=None,flags=flags)
        if SideBySide == True:
            plt.subplot(132)
        else:
            plt.subplot(312)
        plt.imshow(ImWKeypoints_orb)
        plt.title('ORB')
        ImWKeypoints_sift =
cv2.drawKeypoints(img,sift_kps[ind],outImage=None,flags=flags)
        if SideBySide == True:
            plt.subplot(133)
        else:
            plt.subplot(313)
        plt.imshow(ImWKeypoints_sift)
        plt.title('SIFT')
        plt.show()
```

```
imgNames = ['chessboard.png']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,

      blurFAST = True,

      FASTthresh = 1,

      nfeaturesORB = 64,

      nfeaturesSIFT = 16)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
print('FAST + BRIEF descriptor feature 0:\n',fastbrief_dess[0][0],', \n1st
byte in binary:',bin(fastbrief_dess[0][0][0]))
print('\nFAST + BRIEF descriptor feature 4:\n',fastbrief_dess[0][4],', \n1st
byte in binary:',bin(fastbrief_dess[0][4][0]))
print('\nFAST + BRIEF descriptor feature 0 == feature
4?:\n',fastbrief_dess[0][0]==fastbrief_dess[0][4])

for ind, imgName in enumerate(imgNames):
    plt.figure(figsize=(60,60))
    ImWKeypoints_fastbrief =
cv2.drawKeypoints(imgs[ind],[fastbrief_kps[ind][:][0],fastbrief_kps[ind][:][4]
],outImage=None,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)
    plt.subplot(311)
    plt.imshow(ImWKeypoints_fastbrief)
    plt.xlim(350,1200)
    plt.ylim(350,400)
    plt.title('FAST + BRIEF')
```

```
img = cv2.imread('./images/chessboard.png',cv2.IMREAD_GRAYSCALE)
img_blurred = cv2.GaussianBlur(img,ksize=(11,11),sigmaX=0)
print('FAST + BRIEF:')
%timeit FASTBRIEF(img_blurred,thresh=1,show=False)
print('\nORB:')
%timeit ORB(img,nfeatures=64,show=False)
print('\nSIFT:')
%timeit SIFT(img,nfeatures=16,show=False)
```

```
imgNames = ['luca_tomasetti.jpeg']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=70,nfeaturesORB=8,nfeaturesSIFT=64)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['CampusUllandhaug.jpg']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=80,nfeaturesORB=320,nfeaturesSIFT=640)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['Campus Ullandhaug mot Hafrsfjord.jpg']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=80,nfeaturesORB=640,nfeaturesSIFT=1280)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['2022-06-05 (4).png','2022-06-05 (7).png']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=30,nfeaturesORB=200,nfeaturesSIFT=100)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['297755main_GPN-2001-000009_full.jpg']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=30,nfeaturesORB=100,nfeaturesSIFT=100)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['2022-10-06 (27).png','2022-10-06 (28).png']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=30,nfeaturesORB=100,nfeaturesSIFT=100)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['pexels-photo-4388096.webp']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=80,nfeaturesORB=64,nfeaturesSIFT=128)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['Car-Light-Trails-4.jpg']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=80,nfeaturesORB=300,nfeaturesSIFT=300)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=True)
```

```
imgNames = ['LS4.jpg']
imgs, fastbrief_kps, fastbrief_dess, orb_kps, orb_dess, sift_kps, sift_dess =
compare(imgNames,FASTthresh=50,nfeaturesORB=3000,nfeaturesSIFT=3000)
plot_comparison(imgs,fastbrief_kps,orb_kps,sift_kps,SideBySide=False)
```