UNIVERSITY OF STAVANGER

Feature Descriptors

SIFT, SURF and FAST+BRIEF

A report presented for the project in the course

Image Processing and Computer Vision (ELE510)

MD Safin Sarker

B M Nafis Fuad

Abdullah Al Sajid

12 November 2023

Abstract

This project presents a comprehensive analysis of three keypoint detection and descriptor extraction algorithms—SIFT, SURF, and FAST+BRIEF—focusing on their performance across key factors such as scale sensitivity, feature detection capabilities, time efficiency, robustness, and versatility. SIFT demonstrates superior scale and rotation invariance through its scale-space representations, ensuring reliable feature matching. SURF prioritizes computational efficiency with integral images, maintaining good scale and rotation invariance. FAST, although efficient in feature detection, shows limitations in handling scale and rotation changes. Feature detection performance reveals competitive results between SIFT and SURF, with SIFT excelling in scenarios with significant scale. Time efficiency considerations highlight the computational expense of SIFT, contrasting with the speed-focused SURF and FAST. Each algorithm has its merits and drawbacks, necessitating careful consideration based on specific application requirements.

List of Figures

Table of Contents

1. Introduction

Feature descriptors are essential components in computer vision and image processing that provide a compact and distinctive representation of key points or regions in an image. These descriptors are used for various tasks, including object recognition, image matching, and feature tracking. Feature descriptors play a pivotal role in computer vision and image processing, enabling the robust representation of distinctive characteristics within images. These compact representations are crucial for tasks such as object recognition, image matching, and tracking.

Feature descriptors capture key information about local image regions, including gradient orientations, intensity comparisons, or texture patterns. They are designed to be invariant to various transformations, such as scale and rotation, making them versatile tools in computer vision applications. The selection of an appropriate feature descriptor depends on the specific task's requirements, including computational efficiency and the level of distinctiveness necessary for accurate and reliable feature matching and recognition.

1.1 Problem Definition

The scope in this project is to understand and describe key points in images using computer vision. There are two main tasks: finding these points (feature detectors) and providing more details about them (feature descriptors). The focus is on comparing different methods for finding and describing these image points, considering factors like effectiveness and results. The evaluation also looks at whether these methods can handle features at different sizes (scale invariance) and describe features, regardless of their orientation (rotational invariance).

2. Algorithms

Scale Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF) and Features from Accelerated Segment Test (FAST) with Binary Robust Independent Elementary Features (BRIEF) are discussed in this project report.

2.1 SIFT (Scale Invariant Feature Transform)

SIFT, or Scale-Invariant Feature Transform, is a computer vision algorithm used for locating the local features in an image, commonly known as the 'keypoints' of the image. These keypoints are scale and rotation invariant. This whole process can be divided into four parts,

Constructing a Scale Space: To make sure that features are scale-independent
Keypoint Localisation: Identifying the suitable features or keypoints
Orientation Assignment: Ensure the keypoints are rotation invariant
Keypoint Descriptor: Assign a unique fingerprint to each keypoint
And finally, these keypoints can be used for feature matching. [1]

The first step involves finding potential keypoints (interest points) in the image across multiple scales. To achieve scale invariance, the image is repeatedly blurred at different levels using Gaussian filters to create an image pyramid. At each scale level, the Difference of Gaussians (DoG) is computed by subtracting adjacent blurred images as shown in the figure 01.

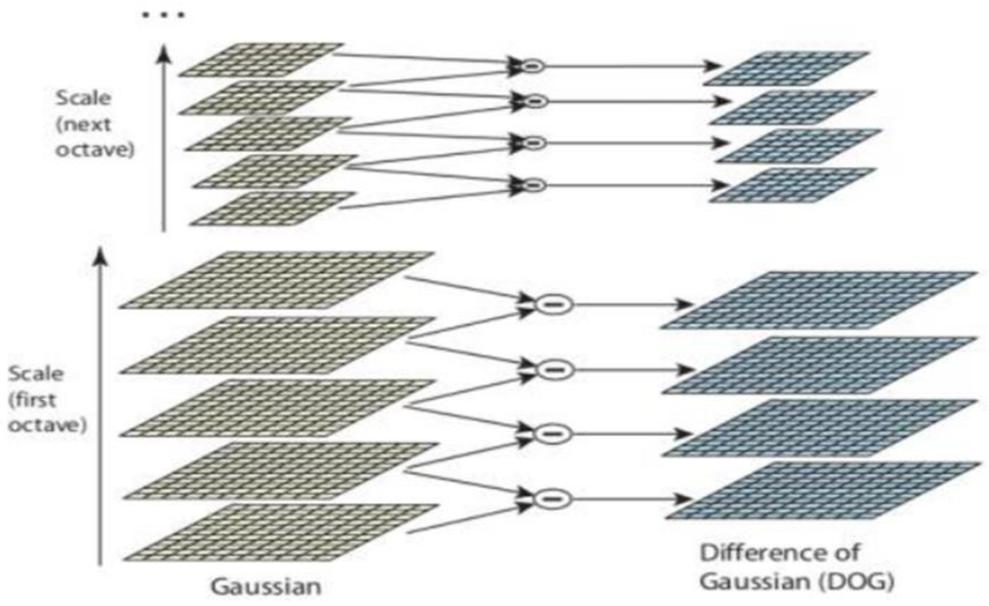


Figure 1: Difference of Gaussian [1]

The DoG represents the scale-normalized Laplacian of the image. Keypoints are detected by finding local maxima and minima in the DoG pyramid. These points correspond to potential keypoints at different scales as shown in figure 02.

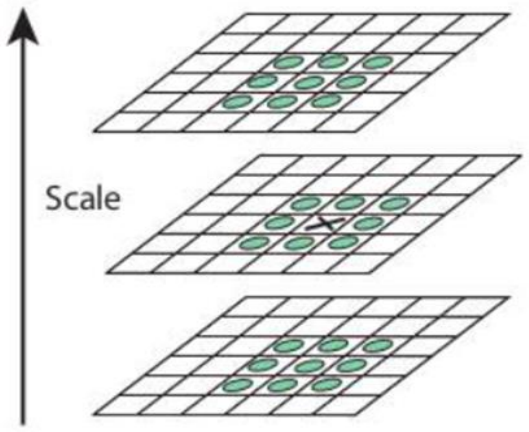


Figure 2: Finding Local Extrema in Scale Space [1]

After detecting potential keypoints, the algorithm performs a precise localization step to improve the accuracy of keypoint positions. It fits a 3D quadratic function to the DoG values in the scale-space to determine the keypoint's sub pixel location and scale. Keypoints with low contrast or those located near edges are discarded as they are less stable and reliable.

SIFT then assigns an orientation to each keypoint to make the descriptor's rotation invariant. A local gradient orientation histogram is created in the region around each keypoint. The dominant orientation in the histogram is chosen as the keypoint's orientation. For each keypoint, a descriptor is generated to represent its local appearance. A region around the keypoint is divided into subregions, and gradient histograms are computed for each subregion. These histograms are combined to form a single high-dimensional vector, known as the SIFT descriptor as shown in figure 03. The descriptor is normalized and can handle changes in illumination and contrast.

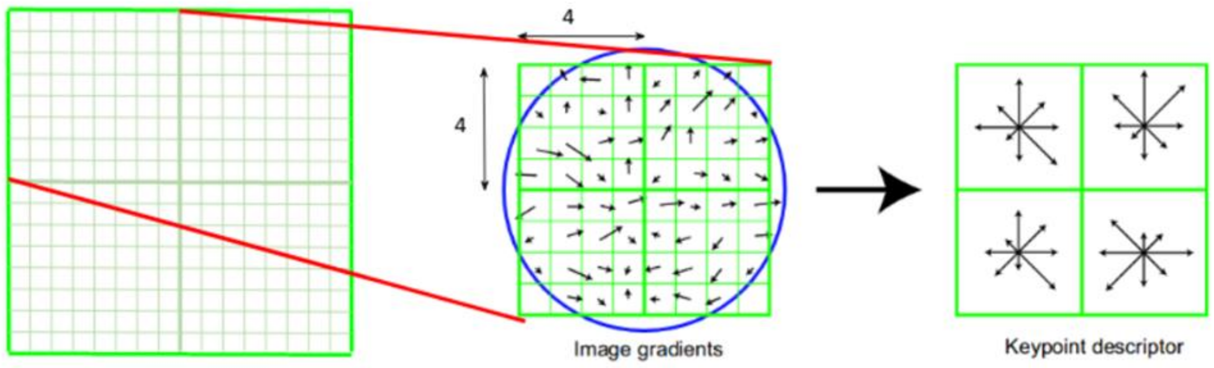


Figure 3: Generating Keypoint Descriptor [2]

2.2 SURF (Speed Up Robust Features)

SURF, introduced in 2006 by Bay, H., Tuytelaars, T., and Van Gool, L., represents a notable acceleration in keypoint detection and feature description compared to its predecessor, SIFT (Scale-Invariant Feature Transform).

Departing from SIFT's Laplacian of Gaussian (LoG) approximation with the Difference of Gaussian, SURF opts for a more efficient Box Filter approximation, facilitating parallel computations across various scales through the use of integral images.

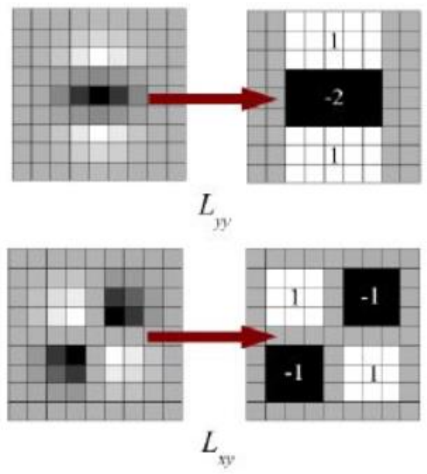


Figure 4: Box Filter Approximation [3]

SURF optimizes scale space analysis by efficiently applying filters of various sizes directly on the original image, in parallel. The strategy involves increasing filter sizes and sampling intervals for each new octave, with non-maximum suppression aiding in the localization of interest points across scales.

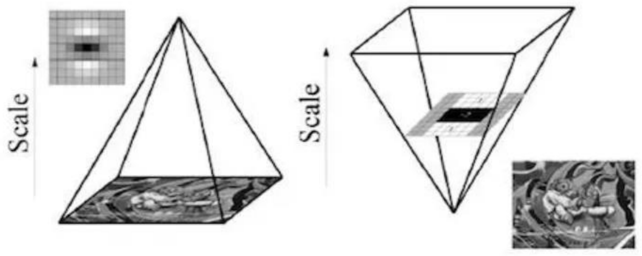


Figure 5: Scaling as Image Pyramid [4]

Instead of iteratively reducing the image size (left), the use of integral images allows the up-scaling of the filter at constant cost (right).

For orientation assignment, SURF employs wavelet responses in horizontal and vertical directions within a local neighborhood. The dominant orientation is estimated by summing responses in a sliding orientation window.

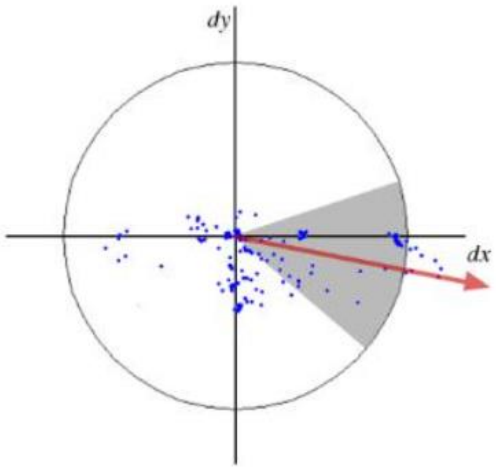


Figure 6: Orientation Alignment [3]

2.3 FAST+BRIEF

2.3.1 FAST (Features from Accelerated Segment Test)

Features from accelerated segment test (FAST) is a widely-used corner detection algorithm, which can be used as a feature detector.

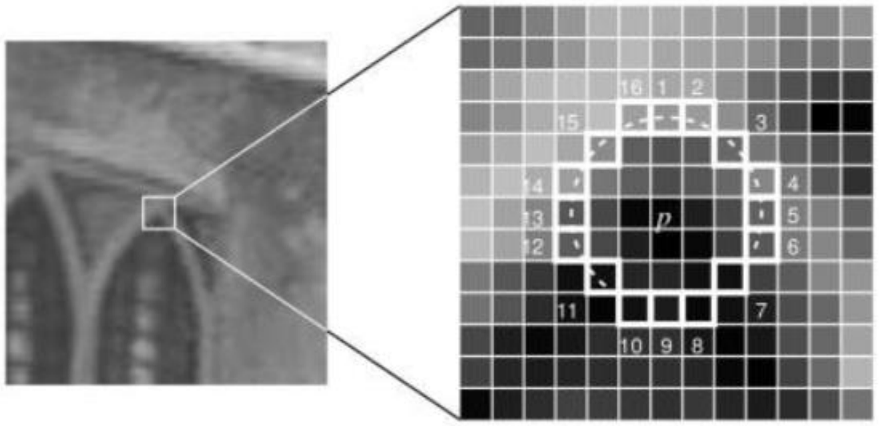


Figure 7: Selecting surrounding pixel around the pixel under test [5]

The FAST algorithm looks at pixels in a picture to find interesting points or corners. It does this by checking if a pixel, let's call it "p," is much brighter or much darker than its surrounding pixels. As shown in figure 07, a circle of 16 pixels around p, if at least 12 of them are significantly brighter or darker than p, then p is considered a corner.

To speed things up, there's a quick test that only looks at four pixels in specific positions within the circle and excludes the other non-corners, in this case pixels 1, 5, 9 and 13 are considered. If three out of these four pixels satisfy the threshold criterion, then the algorithm selects that pixel as point of interest.
The FAST corner detection method stands out from others, and an interesting discovery is that it struggles with computer-generated images perfectly aligned to the x- and y-axes. Surprisingly, FAST fails to detect corners in such images, which are sharp and clear. This happens because FAST looks for a circle of contrasting pixels around a corner, and for a corner to be detected, this circle must include both edges of the corner. [6]

2.3.2 BRIEF (Binary Robust Independent Elementary Features)

BRIEF (Binary Robust Independent Elementary Features) is a feature descriptor used in computer vision. Unlike some other methods, BRIEF doesn't detect features by itself; it relies on a separate feature detection algorithm for that.

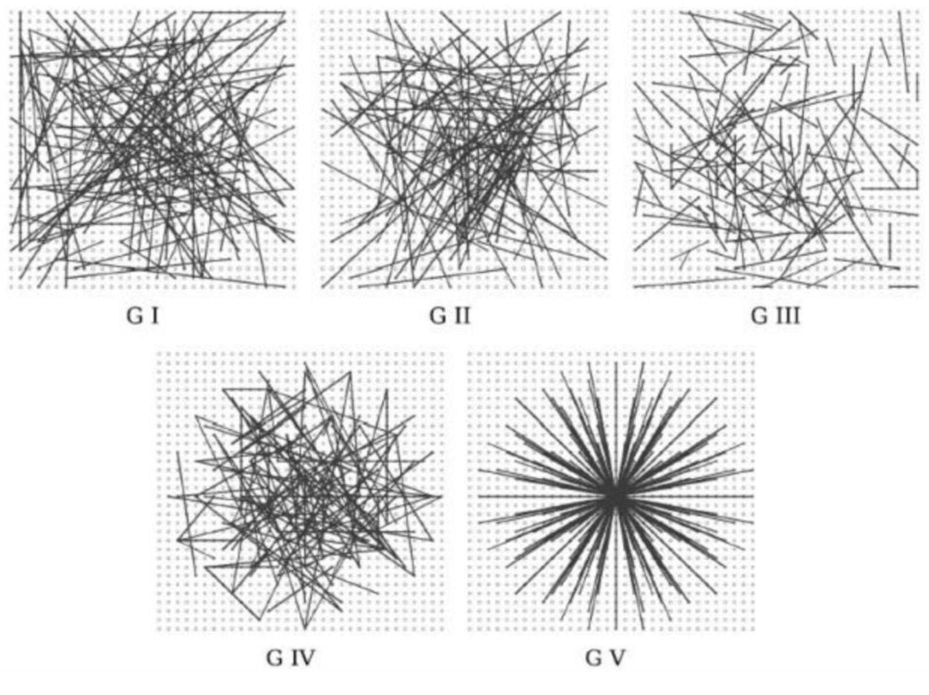


Figure 8: Different approaches to choosing the test locations [7]

Before applying BRIEF, the image is pre-smoothed to make it less sensitive to noise and more stable. A Gaussian kernel is used for this purpose. The neighborhood around a feature is then transformed into a binary feature vector, which consists of 128 to 512 bits.

To create this binary vector, a set of n test pairs is chosen based on specific sampling geometries. These pairs undergo a binary test that compares the intensity differences between the selected pixels. The binary test outputs 0 or 1, depending on which pixel has the higher intensity.

The results of these binary tests for all n pixel pairs are combined into a binary string, forming the feature vector or descriptor. BRIEF doesn't consider orientation information, making it not rotationally invariant. However, its binary string structure enables rapid feature comparison using hamming distance, involving operations like XOR and bit counting on the binary strings of two features.

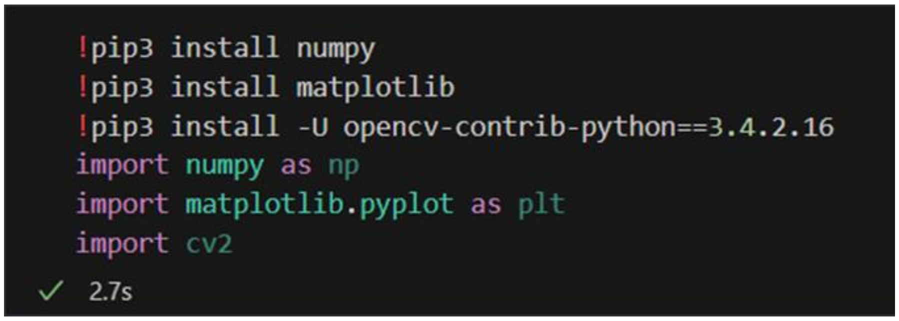
3. Implementation

To ensure the successful execution of the project code, the following components are required:

NumPy: For scientific computing and data manipulation.

Matplotlib: For data visualization and plotting.

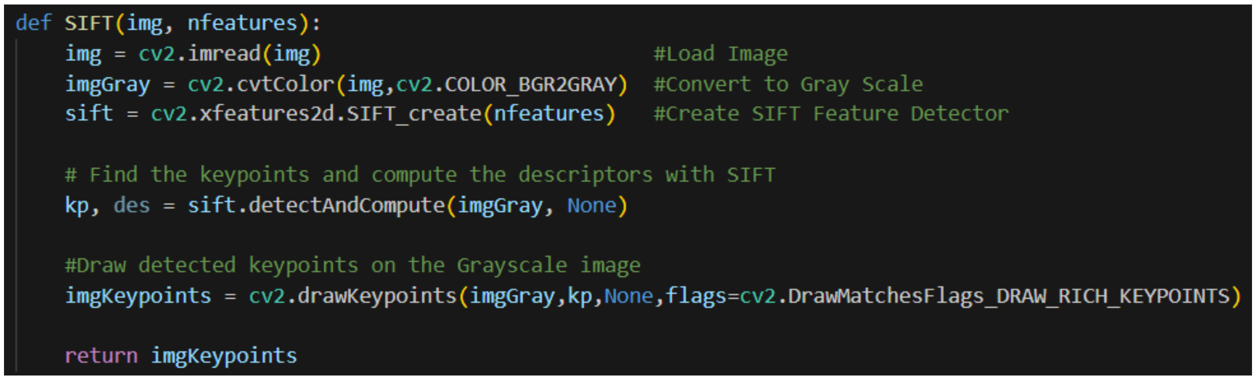
OpenCV-Contrib-Python 3.4.2.16: For image and video processing



```
!pip3 install numpy
!pip3 install matplotlib
!pip3 install -U opencv-contrib-python==3.4.2.16
import numpy as np
import matplotlib.pyplot as plt
import cv2
✓ 2.7s
```

Figure 9: Setting up the environment

3.1 SIFT

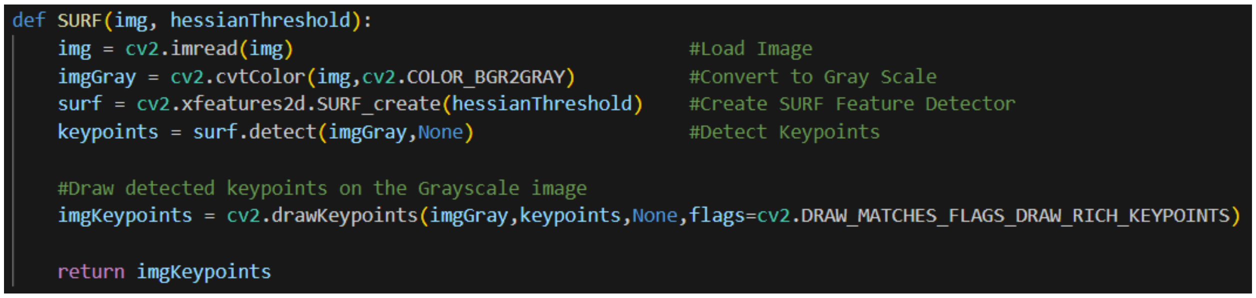


```python
def SIFT(img, nfeatures):
    img = cv2.imread(img)                          #Load Image
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY) #Convert to Gray Scale
    sift = cv2.xfeatures2d.SIFT_create(nfeatures)  #Create SIFT Feature Detector

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

    #Draw detected keypoints on the Grayscale image
    imgKeypoints = cv2.drawKeypoints(imgGray,kp,None,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)

    return imgKeypoints
```

Figure 10: SIFT Algorithm

Figure 10 shows the implementation of SIFT in python using cv2. The code first loads an input image and then converts the loaded image to grayscale. The code creates a SIFT feature detector using cv2.xfeatures2d.SIFT_create(nfeatures), where nfeatures is the number of desired keypoints to detect. This detector is used to find keypoints (kp) and compute their descriptors (des) in the grayscale image. The code then draws the detected keypoints on the grayscale image using cv2.drawKeypoints(). Finally, it returns the image with those keypoints visualized. This implementation is based on the functionalities available in OpenCV [1].

3.2 SURF

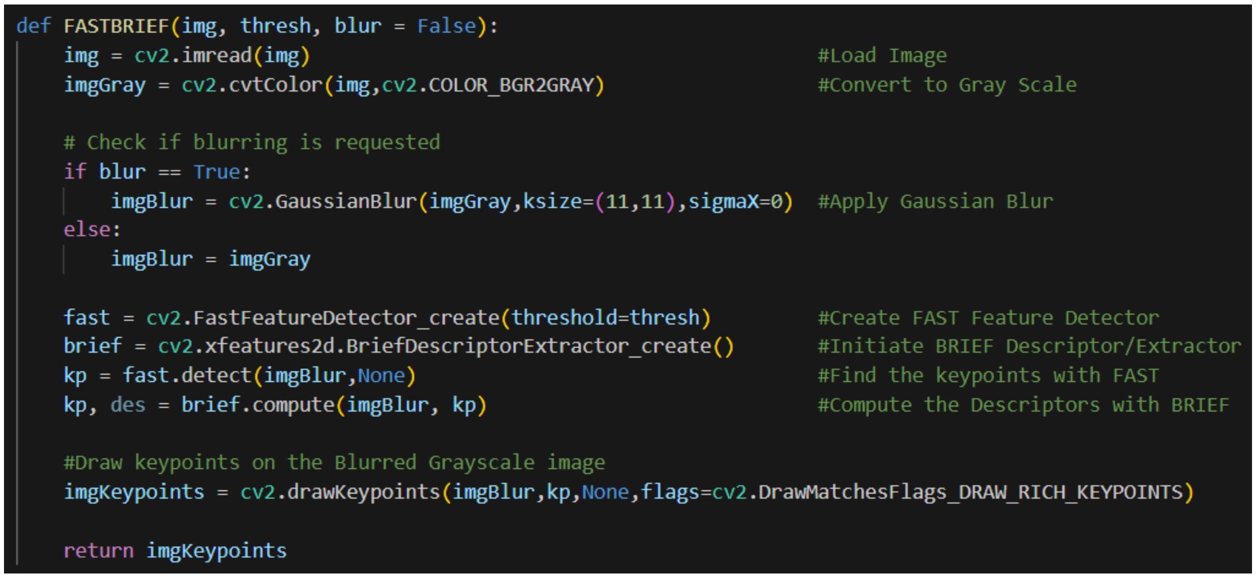


```python
def SURF(img, hessianThreshold):
    img = cv2.imread(img)                              #Load Image
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)     #Convert to Gray Scale
    surf = cv2.xfeatures2d.SURF_create(hessianThreshold) #Create SURF Feature Detector
    keypoints = surf.detect(imgGray,None)              #Detect Keypoints

    #Draw detected keypoints on the Grayscale image
    imgKeypoints = cv2.drawKeypoints(imgGray,keypoints,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    return imgKeypoints
```

Figure 11: SURF Algorithm

Figure 11 shows the SURF Implementation. In SURF function it takes two parameters for image path and the hessianThreshold that influences keypoint detection. OpenCV's imread function to read the image specified by the img parameter. The image is converted from the BGR color space to grayscale using the cvtColor function.Then An instance of the SURF feature detector is created using cv2.xfeatures2d.SURF_create. The hessian threshold parameter is passed to control the sensitivity of keypoint detection. The detection method of the SURF object is called on the grayscale image (imgGray) to find keypoints. The flags parameter specifies how the keypoints should be drawn. cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS indicates that rich keypoints with orientation and size information should be drawn. This implementation is based on the functionalities available in OpenCV [3].

3.3 FAST+BRIEF



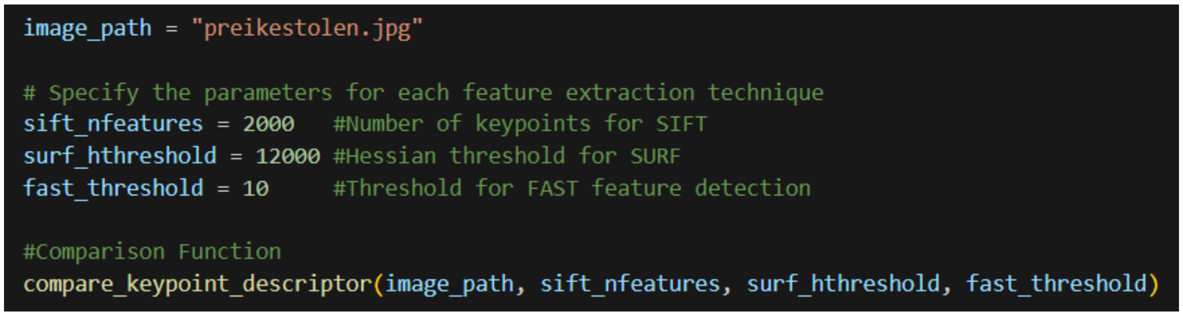
```
def FASTBRIEF(img, thresh, blur = False):
    img = cv2.imread(img)                                        #Load Image
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)               #Convert to Gray Scale

    # Check if blurring is requested
    if blur == True:
        imgBlur = cv2.GaussianBlur(imgGray,ksize=(11,11),sigmaX=0)  #Apply Gaussian Blur
    else:
        imgBlur = imgGray

    fast = cv2.FastFeatureDetector_create(threshold=thresh)      #Create FAST Feature Detector
    brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()    #Initiate BRIEF Descriptor/Extractor
    kp = fast.detect(imgBlur,None)                               #Find the keypoints with FAST
    kp, des = brief.compute(imgBlur, kp)                         #Compute the Descriptors with BRIEF

    #Draw keypoints on the Blurred Grayscale image
    imgKeypoints = cv2.drawKeypoints(imgBlur,kp,None,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)

    return imgKeypoints
```

Figure 12: FAST-BRIEF Algorithm

This figure 12 shows the FAST+BRIEF implementation. FASTBRIEF function takes three parameters: image file path, Threshold value for the FAST detector, and Threshold value for the FAST detector. The code reads the image and converts it to grayscale. If the blur parameter is set to True, Gaussian blur is applied to the grayscale image using **cv2.GaussianBlur**. After that instance for the FAST detector and BRIEF descriptor extractor is created with **cv2.FastFeatureDetector_create()** and **cv2.xfeatures2d.BriefDescriptorExtractor_create().** The FAST detector is used to find key points and BRIEF descriptors are computed for the detected key points. in the preprocessed image. The key points are drawn on the preprocessed image, and the resulting image **imgKeypoints** is returned by the function. This implementation is based on the functionalities available in OpenCV [8].

4. Testing Analysis and Results

4.1 Case 01 Comparison for Different Algorithms



```python
image_path = "preikestolen.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 2000   #Number of keypoints for SIFT
surf_hthreshold = 12000 #Hessian threshold for SURF
fast_threshold = 10     #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold, fast_threshold)
```

Figure 13: Code Snippet of Case 01 Parameters

Figure 13 shows the parameters for case 01. We put the Preikestolen picture and set SIFT features 2000, hessian threshold 12000 for SURF and 10 for FAST.

From figure 14 illustrates the characteristics and descriptors of Scale-Invariant Feature Transform (SIFT). In this depiction, SIFT effectively detects individuals at smaller scales, including the identification of shadows in those scales. However, as the scale increases, the algorithm begins to identify stones. Notably, the orientation of a key point is aligned with the direction of the highest gradient, where the intensity experiences the most significant changes.

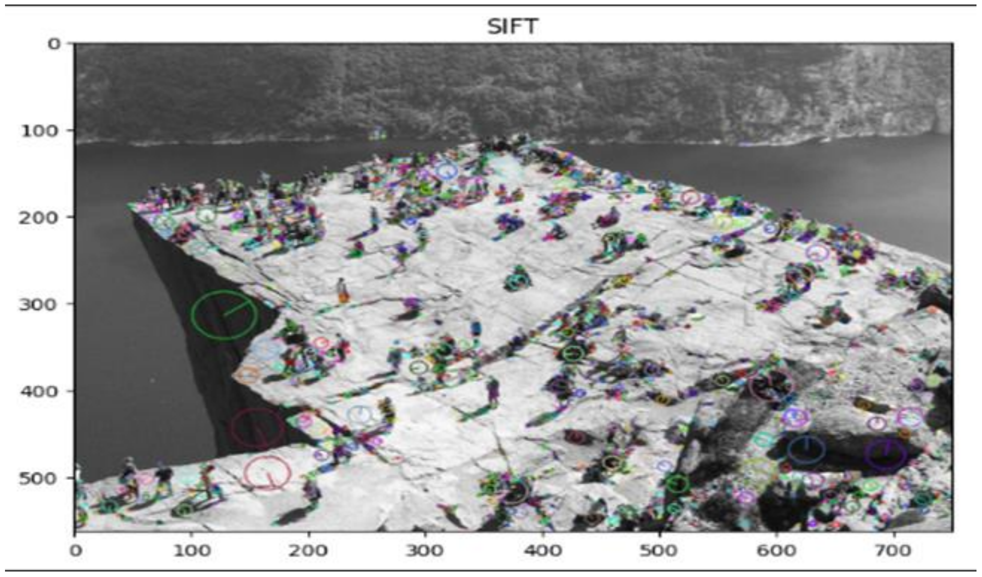


Figure 14: SIFT algorithm on Case 01

For instance, the larger key points associated with the adjacent mountain are oriented towards the illuminated portion, highlighting the algorithm's capability to discern features in well-lit areas. This observation underscores the versatility of SIFT in recognizing various objects and structures across different scales and lighting conditions.

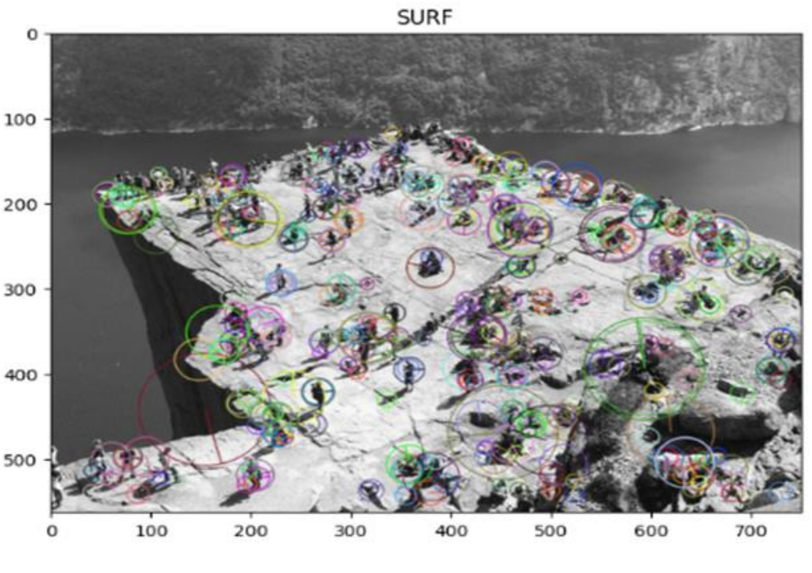


Figure 15: SURF Algorithm on Case 01

SURF outperforms SIFT by detecting a greater number of key points. In the case of SURF, all individuals are successfully identified at both small and medium scales. However, certain individuals on the front side of the plain surface may remain undetected. Notably, at larger scales, SURF excels in identifying features such as stones and mountainous regions.

A noteworthy example is the green circle keypoint on the right side of the mountain. This keypoint is indicative of SURF's ability to assign an orientation vector directed towards the nearest area of intensity changes. This emphasizes the robustness of SURF in capturing a diverse range of features, even in challenging scenarios, and its proficiency in handling varying scales and complex landscapes.

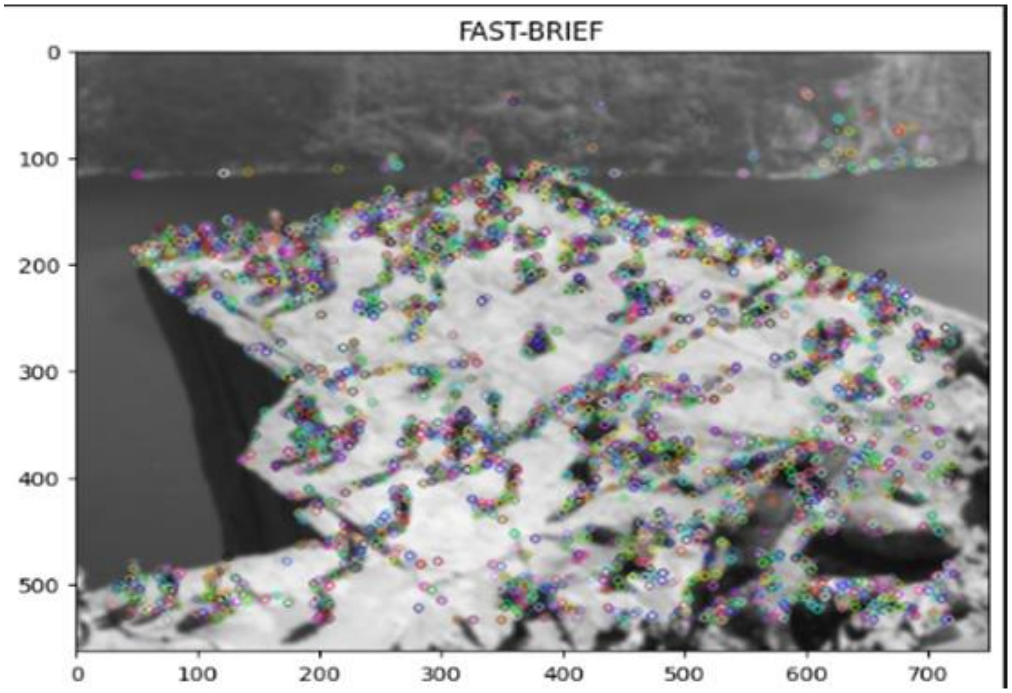
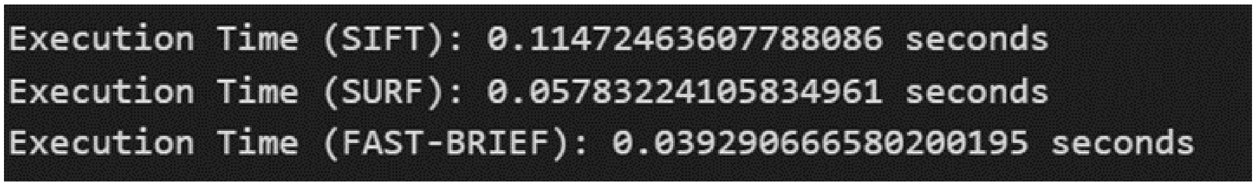


Figure 16: FAST-BRIEF Algorithm on Case 01

FAST+BRIEF excels in detecting corners of edges, with a notable concentration on the right-side and left -side corner of the plain surface. The application of the Gaussian blur technique enhances the algorithm's ability to pinpoint corners along parallel edges. Interestingly, keypoints are also detected on the opposite side of the fjord, representing corners in that region. This distinctive approach of FAST+BRIEF makes it particularly adept at identifying edge corners and highlights its adaptability to diverse landscapes, including scenarios with blurred or parallel edges.

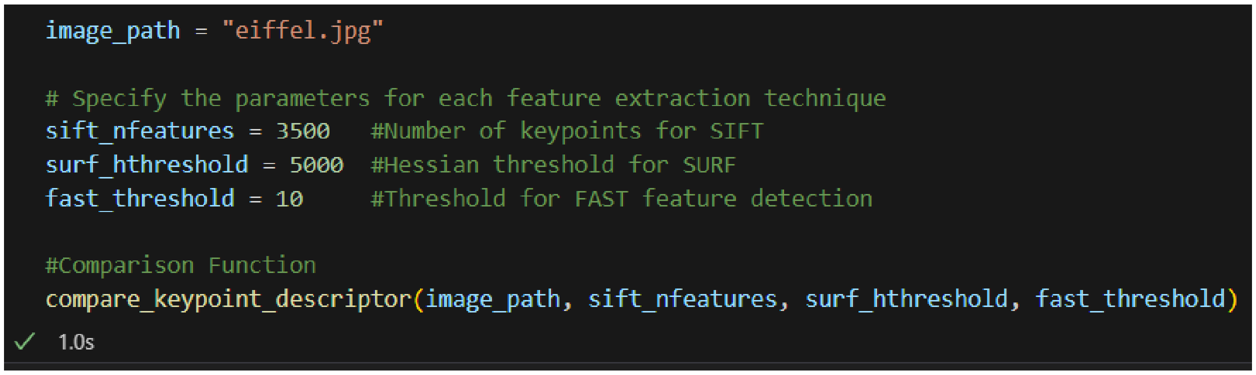


```
Execution Time (SIFT): 0.11472463607788086 seconds
Execution Time (SURF): 0.05783224105834961 seconds
Execution Time (FAST-BRIEF): 0.03929066658020195 seconds
```

Figure 17: Execution time of Different Algorithms for Case 01

As we can see from figure 17, here SIFT takes 0.115 seconds, SURF 0.058 seconds, and FAST-BRIEF 0.039 seconds for execution, showcasing varying processing speeds with FAST-BRIEF being the fastest among the three feature extraction methods.

4.2 Case 02 Comparison for Different Algorithms



```
image_path = "eiffel.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 3500   #Number of keypoints for SIFT
surf_hthreshold = 5000  #Hessian threshold for SURF
fast_threshold = 10     #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold, fast_threshold)
✓ 1.0s
```

Figure 18: Code Snippet of Case 02 Parameters

For Case 02, we are using a picture of the Eiffel Tower. Parameter values for different algorithms have been set to 3500 (SIFT), 5000 (SURF) and 10 (FAST-BRIEF) as shown in fig 18.

By analyzing the picture from figure 19, we can see that SIFT is detecting a lot of small-scale features in the image from the Eiffel tower and also other objects around the tower like trees, buildings, people etc. except the sky and the open road in front. The sky and the road have been detected in the higher scale features. If we look at the orientation vector, we can see that it is pointing towards the strongest gradient in its neighboring area.

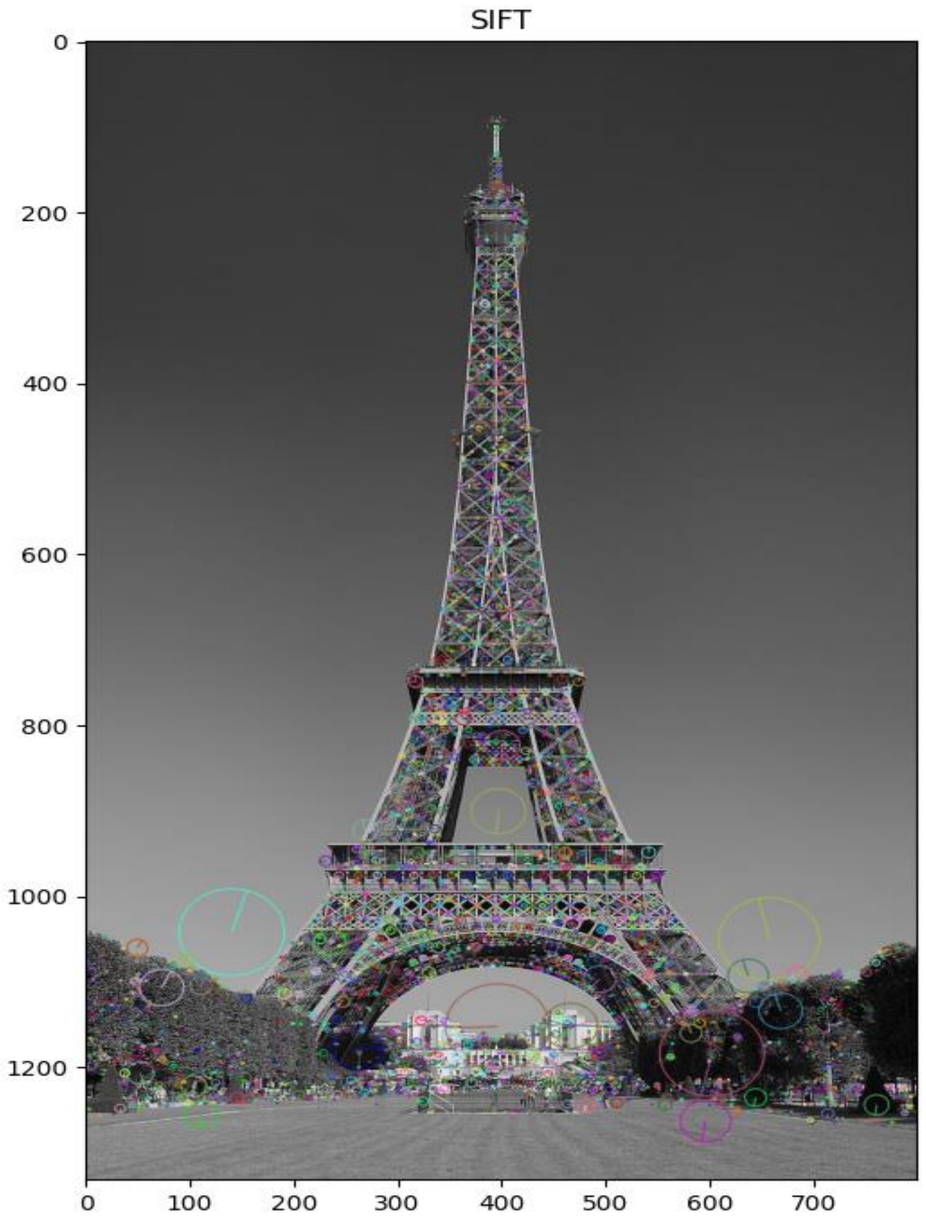


Figure 19: SIFT Algorithm on Case 02

SURF is detecting keypoints mainly from the Eiffel tower in small scale features and the bottom part of the tower behind the trees were detected in higher scale features as shown in figure 20. SURF didn't detect any part from the sky or the road. Detection of the tower looks more precise in SURF than SIFT and it also takes less execution time than SIFT.

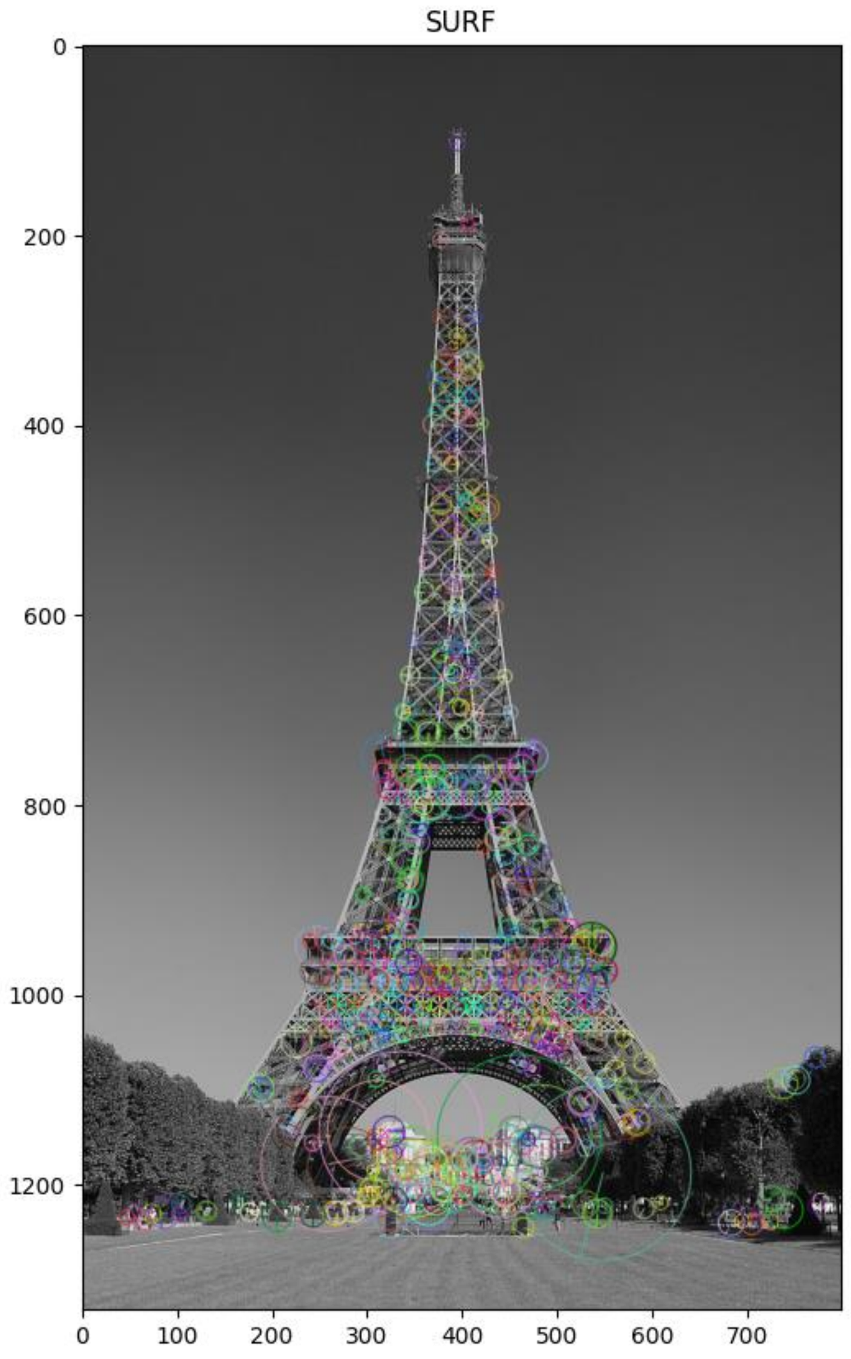


Figure 20: SURF Algorithm on Case 02

From figure 21, we can see that FAST-BRIEF blurs the image and detects the tower and other objects around the tower in small scale features. The sky and the road were not detected as keypoints in FAST-BRIEF.

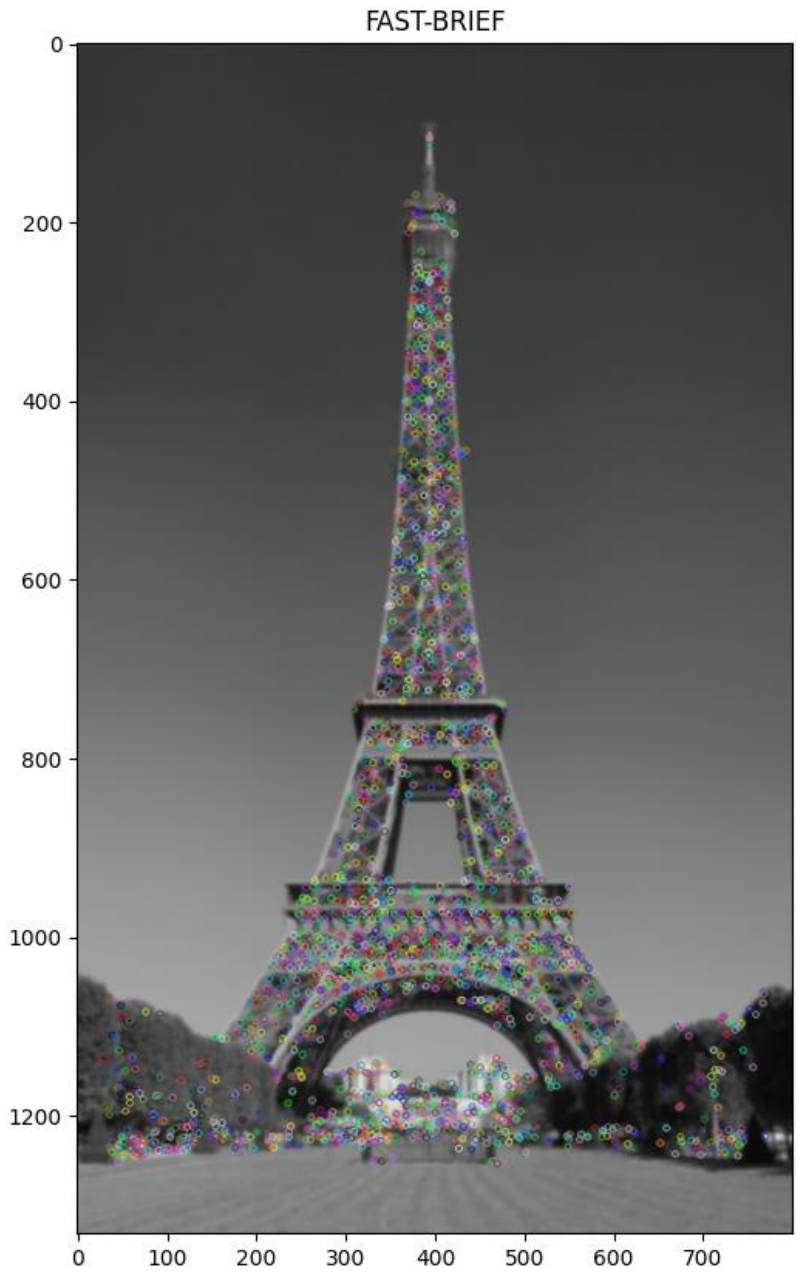
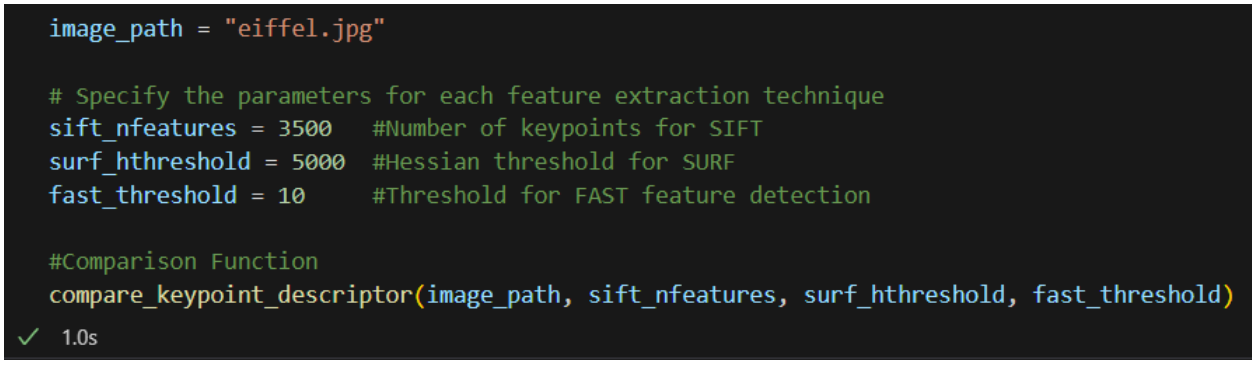


Figure 21: FAST-BRIEF Algorithm on Case 02

Among all the algorithms FAST-BRIEF has the lowest execution time which has been shown in figure 22.



```
image_path = "eiffel.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 3500   #Number of keypoints for SIFT
surf_hthreshold = 5000  #Hessian threshold for SURF
fast_threshold = 10     #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold, fast_threshold)
✓ 1.0s
```

Figure 22: Execution time of Different Algorithms for Case 02

4.3 Case 03 Comparison for Different Algorithms

Case 3 is a photograph of a zebra with its reflection on the water. The number of features for the SIFT is set to 800, and the threshold for SURF and FAST is set to 5000 and 10 respectively.

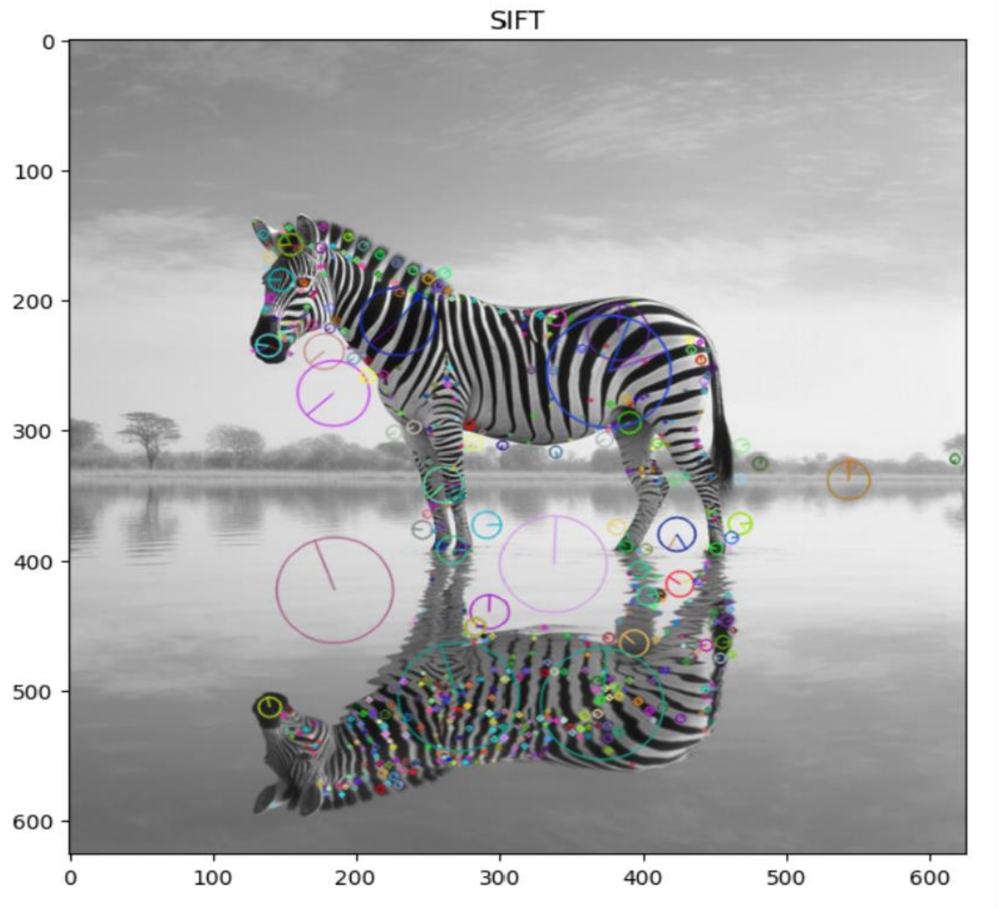


Figure 23: SIFT Algorithm on Case 03

It appears that in figure 23, the SIFT algorithm is initially identifying key points in the reflection of the zebra before detecting similar key points on the actual zebra. Additionally, SIFT seems to be finding key points in the water and trees present in the background of the image. This behavior suggests that SIFT is sensitive to various features within the image, including reflections and elements in the background, contributing to a comprehensive identification of distinctive points throughout the entire scene.

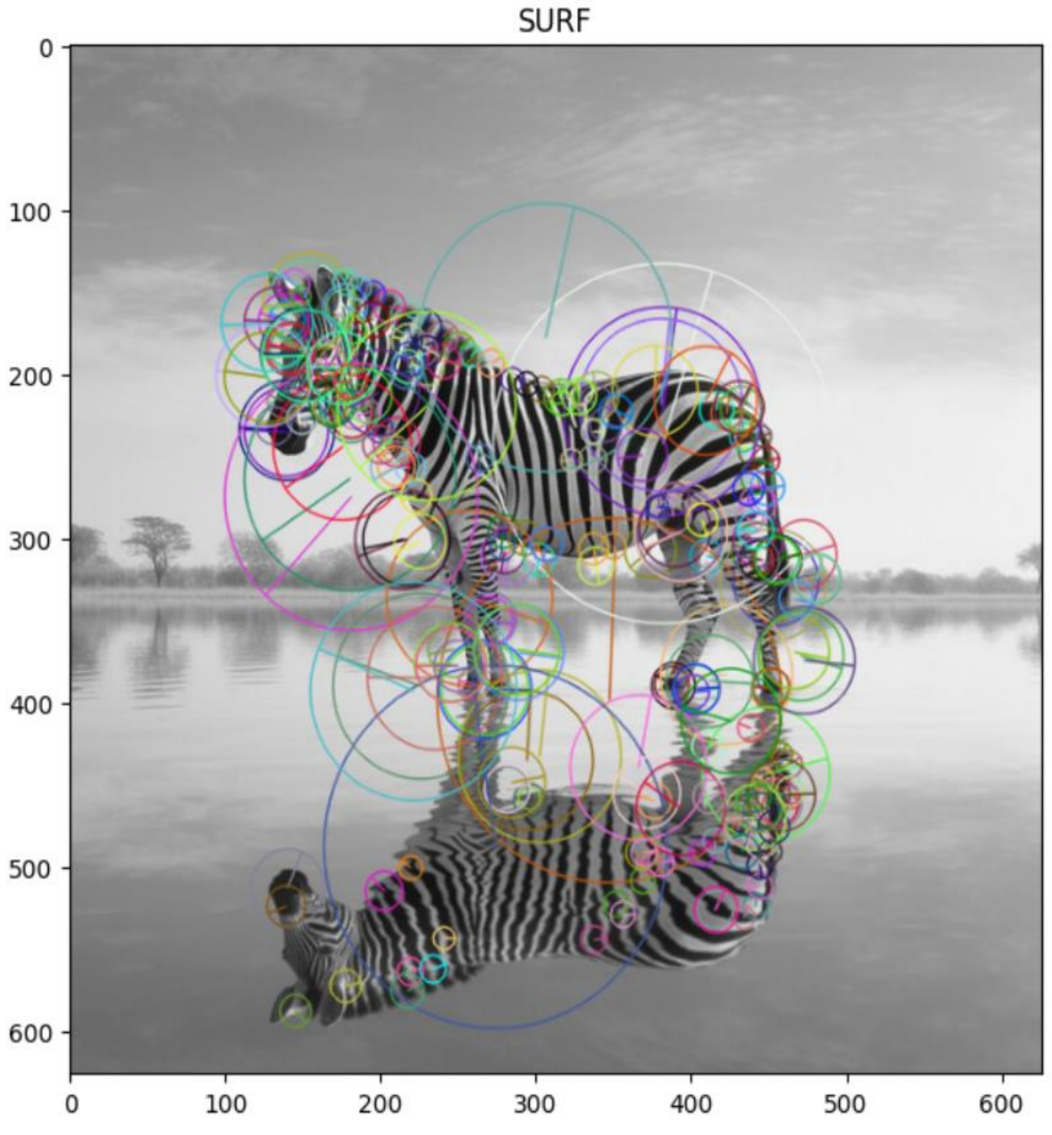


Figure 24: SURF Algorithm on Case 03

SURF easily detects key points in the actual zebra, with significantly fewer key points identified in the reflection. Moreover, some key points, especially the one detected later, include portions of both the real zebra and its reflection in the image. This behavior indicates that SURF is adept at recognizing features in the primary subject (the real zebra) while maintaining sensitivity to the presence of reflections, occasionally resulting in key points that span both aspects of the image.

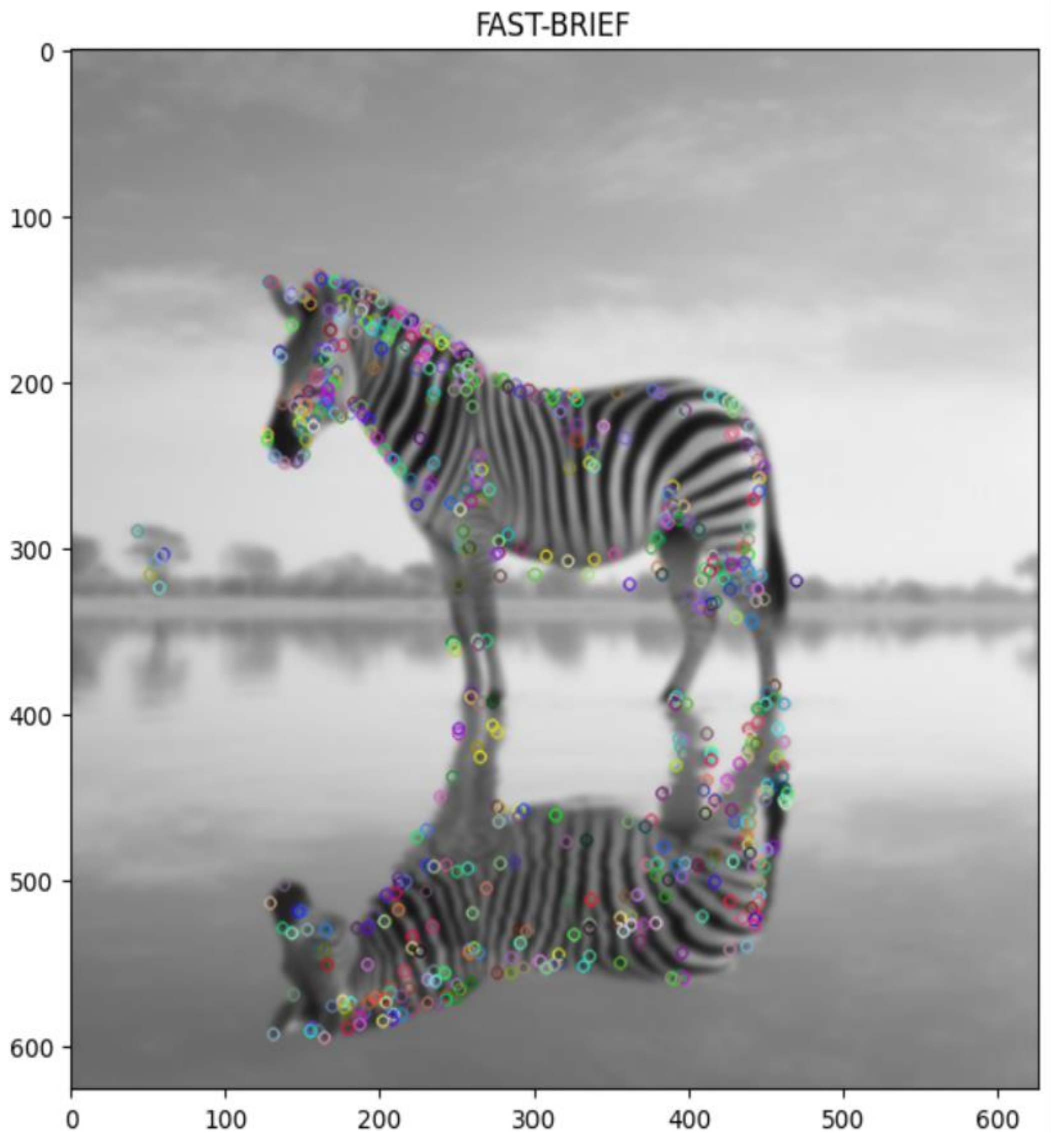


Figure 25: FAST-BRIEF Algorithm on Case 03

In figure 25, using Fast-Brief demonstrates symmetric detection of key points, with nearly all points identified in the actual image being similarly detected in the reflection. This symmetric behavior suggests that Fast-Brief is robust in recognizing corresponding features between the real image and its reflection.

5. Discussion

In this project, we implemented three keypoint detection and descriptor extraction algorithms: SIFT, SURF, and FAST+BRIEF. Our analysis focused on evaluating their performance across various factors such as scale sensitivity, feature detection capabilities, time efficiency, robustness, and versatility.

6. Conclusion:

Regarding Scale sensitivity SIFT excels in robustly handling scale and rotation variations through its scale-space representations, offering reliable feature matching. SURF prioritizes computational efficiency with integral images, maintaining good scale and rotation invariance. FASTis less invariant to scale and rotation changes. While in terms of feature detection performance SIFT and SURF exhibit competitive performance in feature detection, with SIFT often excelling in scenarios with substantial scale and FAST being a corner detection algorithm, is efficient in detecting features but may lack the distinctiveness and robustness of SIFT and SURF. However according to the time efficiency SIFT, though robust, is computationally expensive due to scale-space construction, whereas SURF prioritizes speed for efficient feature detection. FAST stands out here for speed but sacrifices some robustness compared to SIFT and SURF, particularly in handling scale and rotation variations. Each algorithm has its merits and drawbacks, and selecting the most appropriate one requires careful consideration of factors such as scale and rotation sensitivity, feature detection performance, time efficiency, and the nature of the application.

Reference

1. OpenCV, “Introduction to SIFT (Scale-Invariant Feature Transform),” [Online] Available: https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html [Accessed 12 November 2022].
2. Analytics Vidya, “SIFT Algorithm | How to Use SIFT for Image Matching in Python,” Aishwarya Singh [Online]. Available: https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python/ [Accessed 12 November 2022].
3. OpenCV, “Introduction to SURF (Speeded-Up Robust Features),” [Online] Available: https://docs.opencv.org/3.4/df/dd2/tutorial_py_surf_intro.html [Accessed 12 November 2022].
4. D. Tyagi, “Introduction to SURF (Speeded-Up Robust Features),” Medium, 20 March 2019. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-surf-speeded-up-robust-features-c7396d6e7c4e. [Accessed 12 November 2023].
5. OpenCV, “FAST Algorithm for Corner Detection,” [Online] Available: https://docs.opencv.org/4.x/df/d0c/tutorial_py_fast.html [Accessed 12 November 2022].
6. D. Tyagi, “Introduction to FAST (Features from Accelerated Segment Test),” Medium, 2 January 2019. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65. [Accessed 12 November 2023].
7. D. Tyagi, “Introduction to BRIEF (Binary Robust Independent Elementary Features),” Medium, 19 March 2019. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6. [Accessed 12 November 2023].
8. OpenCV, “BRIEF (Binary Robust Independent Elementary Features),” [Online] Available: https://docs.opencv.org/3.4/dc/d7d/tutorial_py_brief.html [Accessed 12 November 2022].

Appendix

A1. Full Code

```
#Setting up the enviroment
!pip3 install numpy
!pip3 install matplotlib
!pip3 install -U opencv-contrib-python==3.4.2.16
```

```
#Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import cv2
import time
```

```
def SIFT(img, nfeatures):
    img = cv2.imread(img)                                #Load Image
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)       #Convert to Gray Scale
    sift = cv2.xfeatures2d.SIFT_create(nfeatures)        #Create SIFT Feature Detector

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

    #Draw detected keypoints on the Grayscale image
    imgKeypoints =
cv2.drawKeypoints(imgGray,kp,None,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)

    return imgKeypoints
```

```
def SURF(img, hessianThreshold):
    img = cv2.imread(img)                                        #Load Image
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)               #Convert to Gray
Scale
    surf = cv2.xfeatures2d.SURF_create(hessianThreshold)         #Create SURF Feature
Detector
    keypoints = surf.detect(imgGray,None)                        #Detect Keypoints

    #Draw detected keypoints on the Grayscale image
    imgKeypoints =
cv2.drawKeypoints(imgGray,keypoints,None,flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_K
EYPOINTS)

    return imgKeypoints
```

```python
def FASTBRIEF(img, thresh, blur = False):
    img = cv2.imread(img)                                          #Load Image
    imgGray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)                 #Convert to
Gray Scale

    # Check if blurring is requested
    if blur == True:
        imgBlur = cv2.GaussianBlur(imgGray,ksize=(11,11),sigmaX=0)  #Apply
Gaussian Blur
    else:
        imgBlur = imgGray

    fast = cv2.FastFeatureDetector_create(threshold=thresh)         #Create FAST
Feature Detector
    brief = cv2.xfeatures2d.BriefDescriptorExtractor_create()       #Initiate
BRIEF Descriptor/Extractor
    kp = fast.detect(imgBlur,None)                                  #Find the
keypoints with FAST
    kp, des = brief.compute(imgBlur, kp)                            #Compute the
Descriptors with BRIEF

    #Draw keypoints on the Blurred Grayscale image
    imgKeypoints =
cv2.drawKeypoints(imgBlur,kp,None,flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)

    return imgKeypoints
```

```python
def compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold,
fast_threshold):
    #Measure execution time for processing an image
    #SIFT
    start_time = time.time()                                #Record Start Time
    img_sift = SIFT(image_path, sift_nfeatures)             #Extract SIFT
features from the image
    end_time = time.time()                                  #Record End Time
    execution_time = end_time - start_time                  #Calculate Execution
Time
    print(f"Execution Time (SIFT): {execution_time} seconds")

    #SURF
    start_time = time.time()                                #Record Start Time
    img_surf = SURF(image_path, surf_hthreshold)            #Extract SURF
features from the image
    end_time = time.time()                                  #Record End Time
    execution_time = end_time - start_time                  #Calculate Execution 
Time
    print(f"Execution Time (SURF): {execution_time} seconds")

    #FAST-BRIEF
    start_time = time.time()                                #Record Start Time
    img_fast = FASTBRIEF(image_path, fast_threshold, True)  #Extract FAST-BRIEF 
features from the image
    end_time = time.time()                                  #Record End Time
    execution_time = end_time - start_time                  #Calculate Execution 
Time
    print(f"Execution Time (FAST-BRIEF): {execution_time} seconds")

    #Plot the images
    plt.figure(figsize=(25, 10))

    plt.subplot(131)
    plt.imshow(cv2.cvtColor(img_sift, cv2.COLOR_BGR2RGB))   #Display the SIFT 
result
    plt.title('SIFT')

    plt.subplot(132)
    plt.imshow(cv2.cvtColor(img_surf, cv2.COLOR_BGR2RGB))   #Display the SURF 
result
    plt.title('SURF')

    plt.subplot(133)
    plt.imshow(cv2.cvtColor(img_fast, cv2.COLOR_BGR2RGB))   #Display the FAST-
BRIEF result
    plt.title('FAST-BRIEF')

    plt.show()
```

```
image_path = "preikestolen.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 2000   #Number of keypoints for SIFT
surf_hthreshold = 12000 #Hessian threshold for SURF
fast_threshold = 10     #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold, 
fast_threshold)
```

```python
image_path = "eiffel.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 3500   #Number of keypoints for SIFT
surf_hthreshold = 5000  #Hessian threshold for SURF
fast_threshold = 10     #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold,
fast_threshold)
```

```python
image_path = "zebra.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 800    #Number of keypoints for SIFT
surf_hthreshold = 5000  #Hessian threshold for SURF
fast_threshold = 10     #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold,
fast_threshold)
```

```python
image_path = "zebras.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 1000   #Number of keypoints for SIFT
surf_hthreshold = 4000  #Hessian threshold for SURF
fast_threshold = 18     #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold,
fast_threshold)
```

```python
image_path = "gregoryofnin.jpg"

# Specify the parameters for each feature extraction technique
sift_nfeatures = 1000   #Number of keypoints for SIFT
surf_hthreshold = 3000  #Hessian threshold for SURF
fast_threshold = 5      #Threshold for FAST feature detection

#Comparison Function
compare_keypoint_descriptor(image_path, sift_nfeatures, surf_hthreshold,
fast_threshold)
```