جامعة الأمير مقرن بن عبد العزيز
Prince Mugrin University

Lab 7

SIFT Detector and Features Matching with OpenCV

Learning Objectives:

- Understand how SIFT works and why it is useful.
- Detect keypoints and compute descriptors using SIFT.
- Perform feature matching between two images..
- Use SIFT for applications like object recognition and image stitching
- Apply other features detections algorithms (SURF, FAST, BRIEF)

Introduction

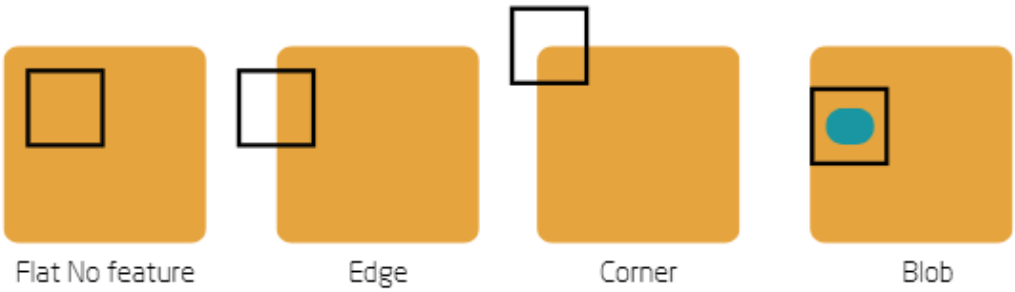
Feature detection and matching are fundamental techniques in computer vision used to identify corresponding points in different images. These techniques enable tasks such as **object recognition, image stitching, 3D reconstruction, augmented reality, and autonomous navigation**.
Among the most powerful feature detection methods is the **Scale-Invariant Feature Transform (SIFT)** algorithm, which extracts unique keypoints and descriptors that remain consistent across different scales, rotations, and lighting conditions.

1️⃣ Understanding SIFT (Scale-Invariant Feature Transform)

📌 Why Is SIFT Important?

SIFT is a widely used method for detecting and describing local features in images. Unlike traditional edge detection methods, SIFT keypoints are:

- **Scale-invariant:** They remain consistent regardless of the image size.
- **Rotation-invariant:** They can be detected in different orientations.
- **Illumination-robust:** They work well under varying lighting conditions.

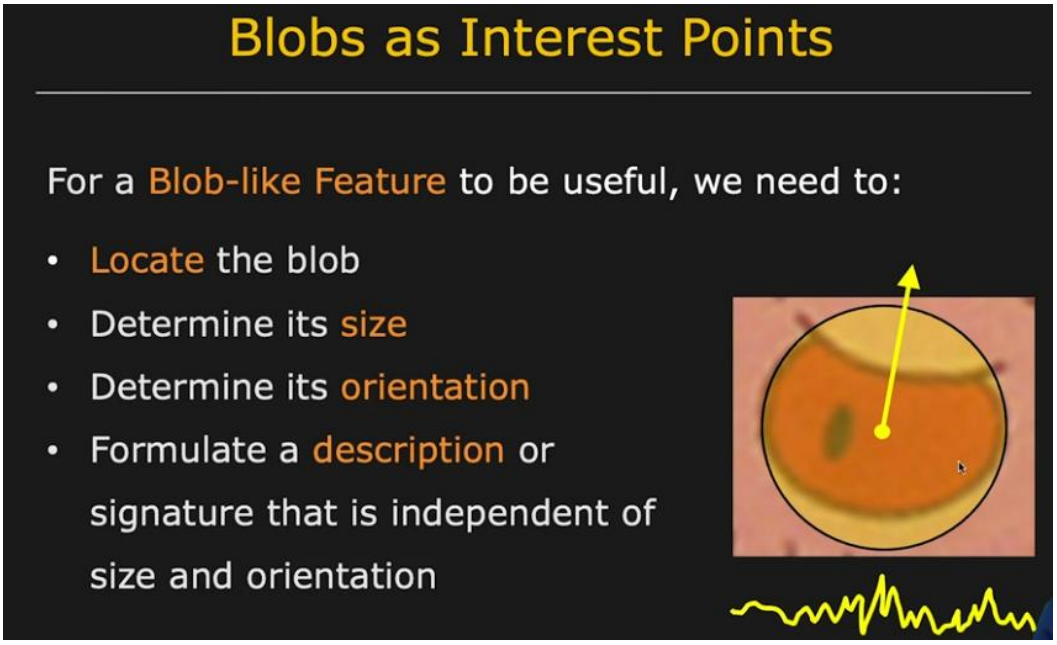


The difference between (Edge, Corner, Blob

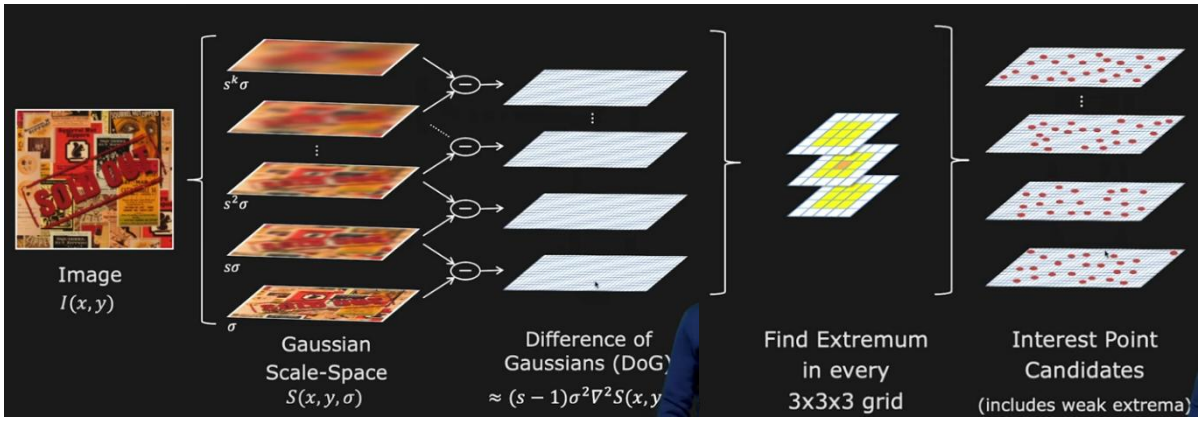
Before we start please open “sift.ipynb” file and focus with me

📌 How Does SIFT Work?

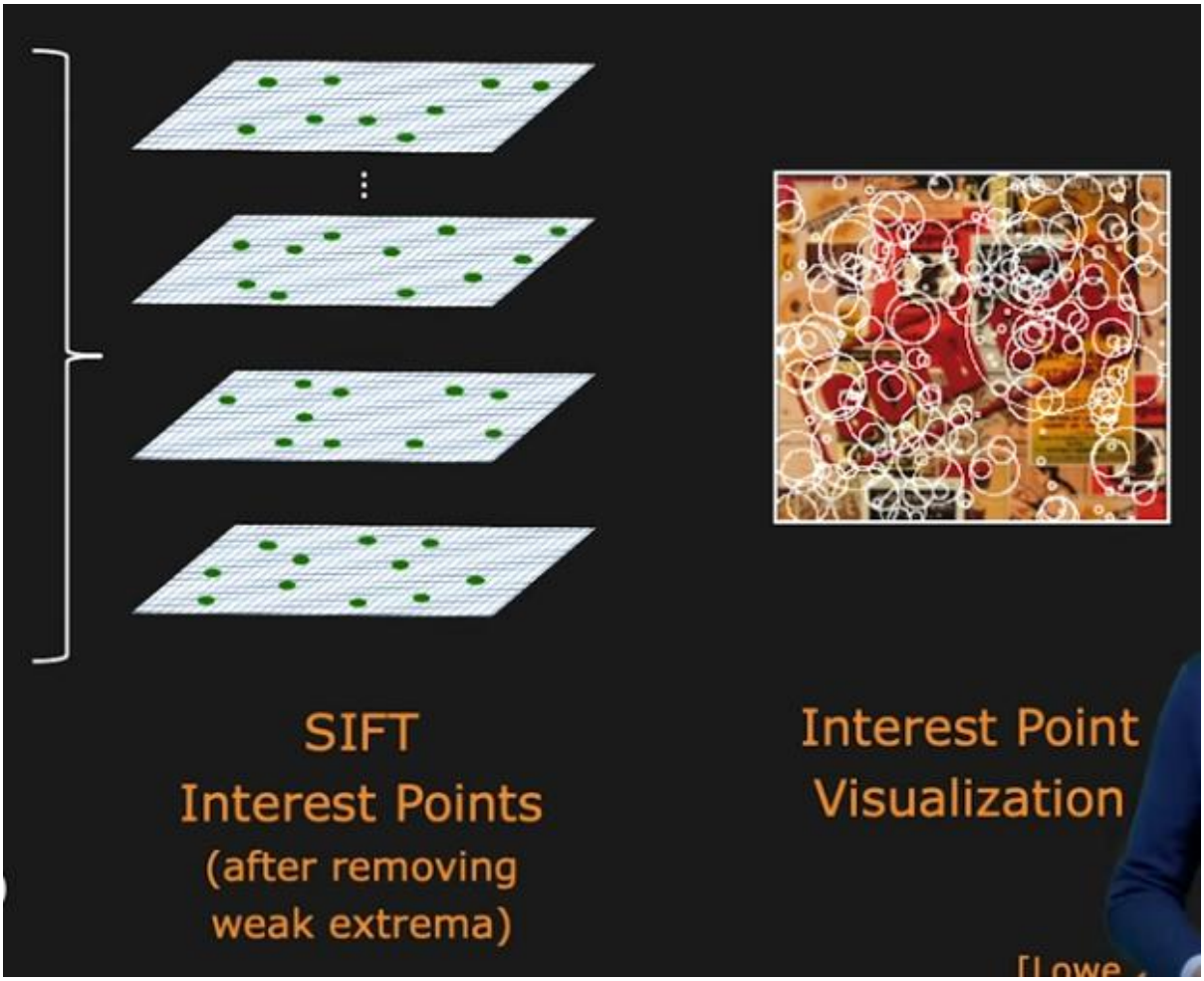
1. **Identify Distinctive Regions:**
   Determine areas of interest—keypoints, blobs, or unique features—that stand out from their surroundings. The goal is to select regions with distinctive, rich characteristics that remain identifiable under different conditions, as simple edges or corners are often insufficient.
2. **Scale-space Extrema Detection:**
   Construct a scale-space pyramid by convolving the image with Gaussian filters at multiple scales. Compute the Difference-of-Gaussians (DoG) between adjacent scales, then identify local extrema across both spatial and scale dimensions to pinpoint potential keypoints.
3. **Keypoint Localization:**
   Refine each candidate keypoint using a Taylor series expansion around its initial location to improve accuracy. Discard keypoints with low contrast or unstable edge responses, ensuring that only robust and reliable features are retained.
4. **Orientation Assignment**
   Calculate the local gradient magnitudes and orientations around each keypoint. Build an orientation histogram and assign one or more dominant orientations to each keypoint. This step guarantees rotation invariance, allowing features to be reliably matched regardless of image rotation.
5. **Keypoint Descriptor**
   Within the neighborhood of each keypoint, compute local gradients and weight them with a Gaussian window. Summarize these gradients into histograms to form a distinctive descriptor. Finally, normalize the descriptor to reduce the impact of illumination changes, enhancing its robustness.



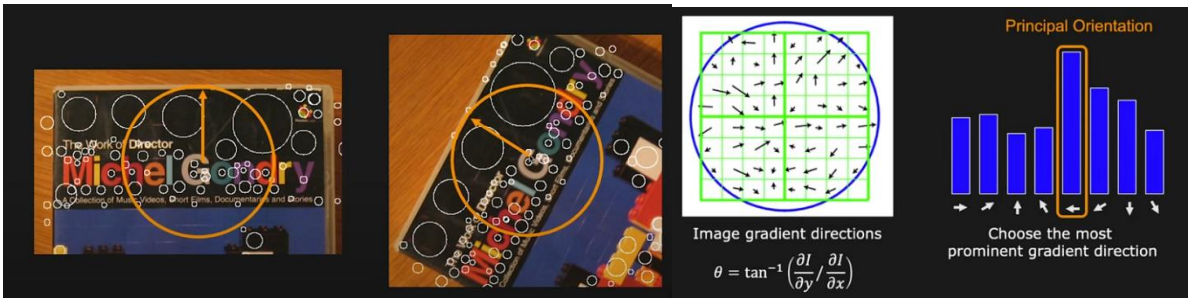
Step 1, what the kepoints need to be?



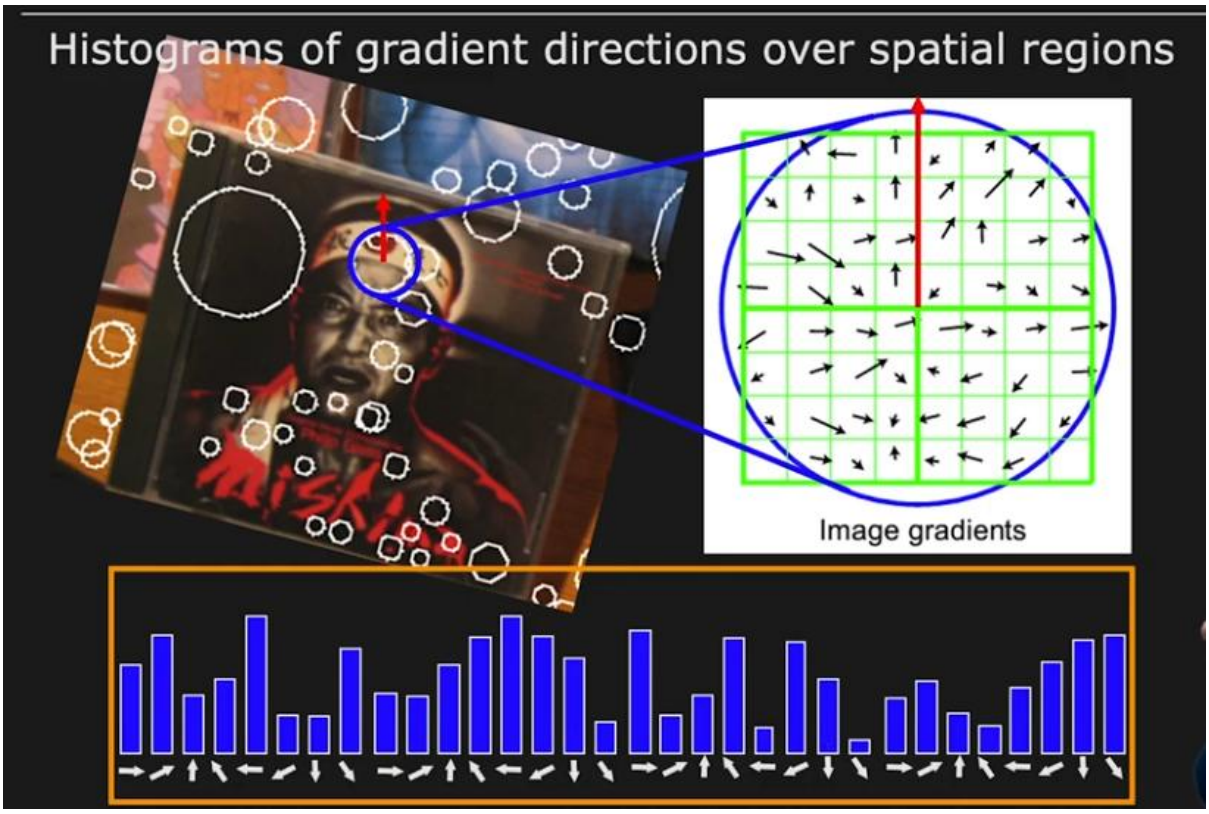
Step 2, detecting keypoints using Difference of Gaussians (DoG) method



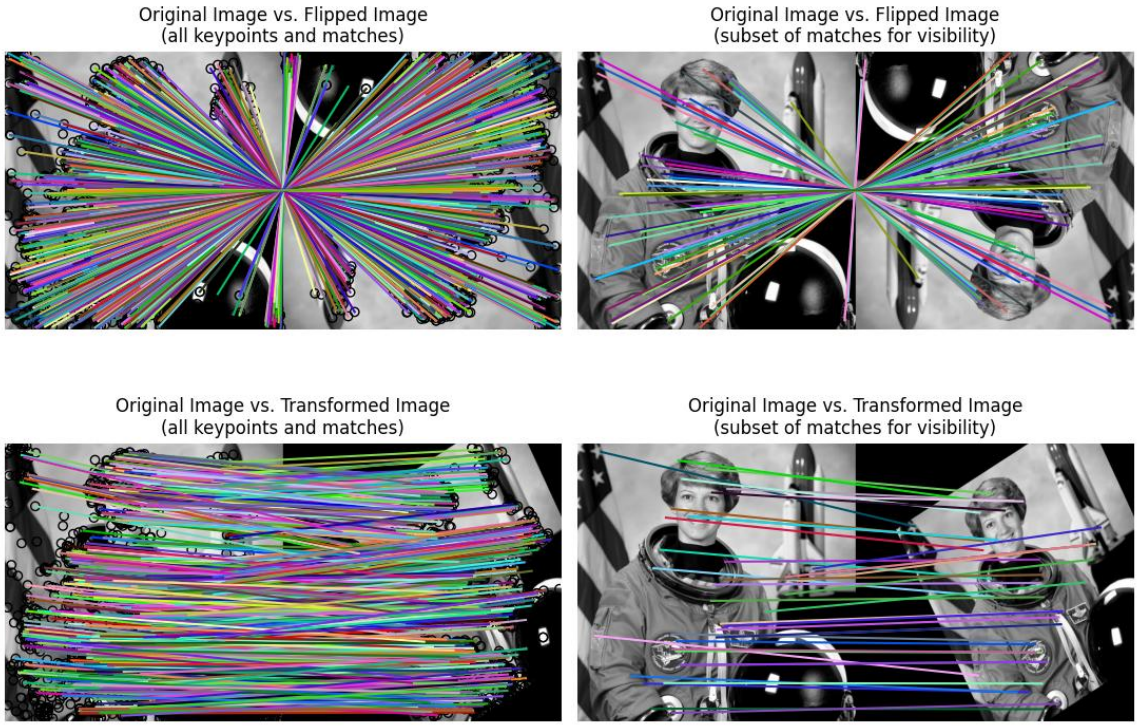
Step 3, keep strong keypoints only



Step 4, how to beat the orientation problem



Step 5, finally build the descriptor for each keypoint, which serves as the keypoint's ID and will help match keypoints between different images later on.



2️⃣ Detecting SIFT Features

🛠️ Task: Extract SIFT Keypoints from an Image

```
import cv2
import numpy as np

# Load an image
gray = cv2.imread('image.jpg', cv2.IMREAD_GRAYSCALE)

# Initialize SIFT detector
sift = cv2.SIFT_create()

# Detect keypoints and compute descriptors
keypoints, descriptors = sift.detectAndCompute(gray, None)

# Draw keypoints on the image
       img_keypoints = cv2.drawKeypoints(gray, keypoints, None,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

cv2.imshow('SIFT Keypoints', img_keypoints)
cv2.waitKey(0)
Cv2.destroyAllWindows()
```

3️⃣ Understanding Feature Matching

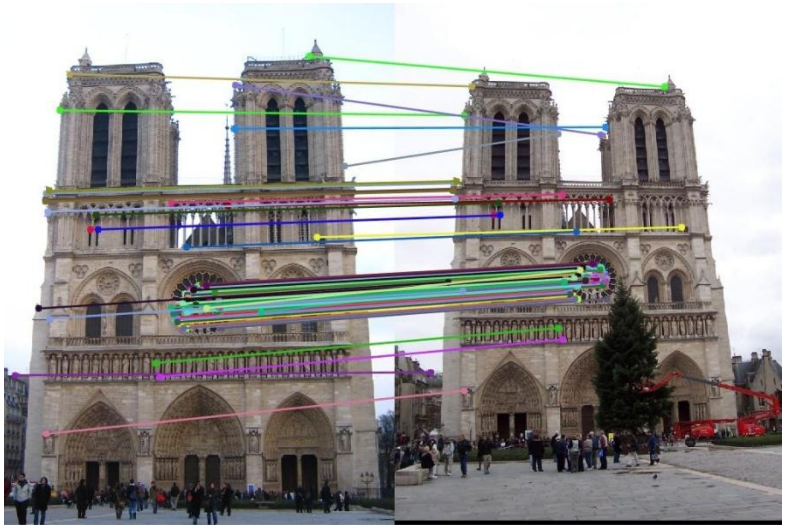
📌 Why Is Feature Matching Important?

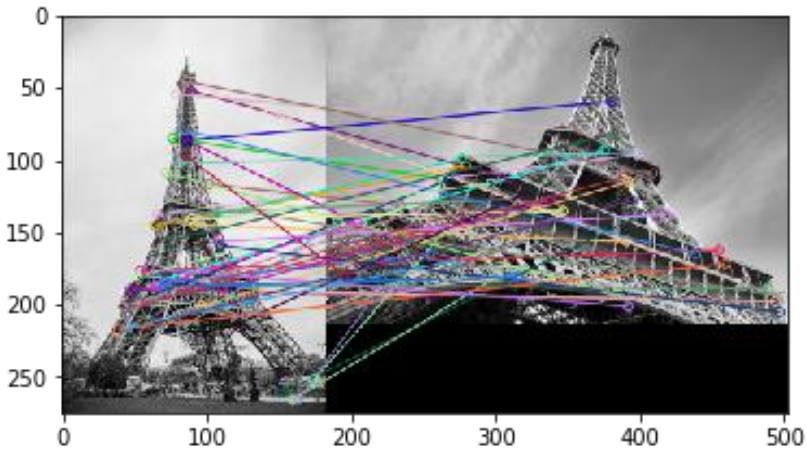
Feature matching allows us to compare images and recognize patterns despite changes in **scale, rotation, illumination, and perspective**. It enables:

- **Object recognition:** Identifying objects in different images.
- **Image stitching:** Combining multiple images to create panoramas.
- **3D reconstruction:** Estimating depth and structure from multiple images.
- **Augmented reality (AR):** Detecting real-world surfaces to overlay virtual content.
- **Autonomous navigation:** Enabling robots and self-driving cars to recognize landmarks and objects.

📌 How Does Feature Matching Work?

1. **Feature Detection:** Identify keypoints in an image (e.g., edges, corners, textures).
2. **Descriptor Computation:** Generate unique numerical representations of each keypoint.
3. **Feature Matching:** Compare descriptors between two images to find corresponding features.
4. **Filtering Matches:** Apply techniques like Lowe's ratio test to remove weak matches.





4️⃣ Feature Matching with SIFT

🛠️ Task: Match Features Between Two Images using BFMatcher

BFMatcher, short for Brute-Force Matcher, works by comparing every descriptor from one image to every descriptor from another image to find the best matches. Here's a breakdown of its process:

1. **Descriptor Comparison:**
   BFMatcher computes a distance (commonly Euclidean or L2 norm) between each descriptor in the first image and every descriptor in the second image.
2. **Finding Nearest Neighbors:**
   Using methods like `knnMatch`, it finds the top k closest descriptors (neighbors) for each descriptor, where k is specified by the user.
3. **Selecting Best Matches:**
   The matcher initially returns these nearest neighbors. However, not all matches are reliable, so further filtering is necessary.
4. **Lowe's Ratio Test:**
   To filter out ambiguous matches, the algorithm applies Lowe's ratio test. For each pair of matches (the best and the second-best), it checks if the best match's distance is significantly lower than the second-best (e.g., less than 75% of the second-best distance). If it is, the match is considered reliable.
5. **Result:**
   The final set of good matches represents keypoint pairs that are highly similar between the two images.

```
# Load two images
gray1 = cv2.imread('image1.jpg', cv2.IMREAD_GRAYSCALE)
gray2 = cv2.imread('image2.jpg', cv2.IMREAD_GRAYSCALE)

# Detect and compute features
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(gray1, None)
kp2, des2 = sift.detectAndCompute(gray2, None)

# Use BFMatcher for feature matching
bf = cv2.BFMatcher()
matches = bf.knnMatch(des1, des2, k=2)

# Apply ratio test (Lowe's ratio test)
good_matches = []
for m, n in matches:
    if m.distance < 0.75 * n.distance:
        good_matches.append(m)
```

```
# Draw matches
img_matches = cv2.drawMatches(gray1, kp1, gray2, kp2, good_matches, None,
flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

cv2.imshow('SIFT Feature Matching', img_matches)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

🛠️ Exercise 1: Extract SIFT Keypoints from an Image

Objective:

Extract and visualize SIFT keypoints from an image to understand how local features are detected.

Instructions:

1. *Load an image and convert it to grayscale.*
2. *Use SIFT to detect keypoints and compute descriptors. And modify its parameters*
3. *Draw the detected keypoints on the image and display it.*

🏄 Exercise 2: Extract SURF Keypoints from an Image

Objective:

Extract and visualize SURF (Speeded-Up Robust Features) keypoints from an image to understand how local features are detected efficiently.

Instructions:

1. Load an image and convert it to grayscale.
2. Use SURF to detect keypoints and compute descriptors. Modify its parameters (such as Hessian threshold) to observe its effect.
3. Draw the detected keypoints on the image and display it.

⚡ Exercise 3: Extract FAST Keypoints from an Image

Objective:

Extract and visualize FAST (Features from Accelerated Segment Test) keypoints from an image to understand how corner features are detected quickly.

Instructions:

1. Load an image and convert it to grayscale.
2. Use FAST to detect keypoints. Modify its parameters (such as threshold and non-max suppression) to observe its effect.
3. Draw the detected keypoints on the image and display it.

🔍 Exercise 4: Extract BRIEF Descriptors from an Image

Objective:

Extract and visualize BRIEF (Binary Robust Independent Elementary Features) descriptors from an image to understand how binary feature descriptors work.

Instructions:

1. Load an image and convert it to grayscale.
2. Use FAST to detect keypoints, as BRIEF does not detect keypoints on its own. Modify the FAST parameters if needed.
3. Compute BRIEF descriptors for the detected keypoints. Modify its parameters to observe the effect.
4. Draw the detected keypoints on the image and display it.

🛠️ Exercise 5: Match Features Between Two Images

Objective:

Match features between two images using SIFT and Brute Force Matcher to find correspondences.

Instructions:

1. *Load two images and convert them to grayscale.*
2. *Detect and compute SIFT features in both images.*
3. *Use different matcher (not BFMatcher) to find feature correspondences.*
4. *Filter matches using Lowe's ratio test.*
5. *Draw the matching keypoints between the two images and display them.*

References:

- **Feature Detection in OpenCV**
- SIFT **OpenCV**
- SURF OpenCV
- FAST OpenCV
- BRIEF OpenCV
- **Good Article**
- **Youtube**