NATIONAL UNIVERSITY OF MODERN LANGUAGES
ISLAMABAD

Computer Vision (Assignments)

Assignment: 02

Submitted to
Mr.Muhammad Waqar

Submitted By
Adeel Naeem
(BSAI-146)

Submission Date: May 20th ,2025

ASSIGMENT 2

1. SIFT (Scale-Invariant Feature Transform):
Detects and describes local features in images. It is scale and rotation invariant and highly robust to changes in illumination. SIFT identifies keypoints and computes descriptors for them using Difference of Gaussians and gradient histograms.

2. SURF (Speeded-Up Robust Features):
A faster alternative to SIFT that uses integral images and box filters to detect keypoints. It is also scale and rotation invariant, suitable for real-time applications, though it is patented.

3. ORB (Oriented FAST and Rotated BRIEF):
An efficient and free alternative to SIFT and SURF. It combines the FAST keypoint detector and the BRIEF descriptor with orientation compensation, providing good performance under scale and rotation with high speed.

4. BFMatcher (Brute Force Matcher):
Matches descriptors by computing distances between every possible pair. Simple and effective, but can be slow on large datasets.

5. FLANN (Fast Library for Approximate Nearest Neighbors):
A faster matcher for large datasets, using approximate methods like KD-Trees. It provides faster performance but can be slightly less accurate.

Performance Analysis

Evaluation under different conditions:

- **Lighting:**
  SIFT and ORB perform well under moderate lighting changes, but performance may degrade under extreme illumination.
- **Scale:**
  SIFT and SURF are scale-invariant and outperform ORB in recognizing resized objects.
- **Rotation:**
  All three techniques handle rotation well, with SIFT being the most stable.

Conclusion:
In real-time applications, **ORB with BFMatcher** provides a good balance between speed and accuracy.
For more robust detection, especially in cluttered scenes or transformed views, **SIFT with FLANN** is recommended for better performance despite being computationally heavier.

Code:

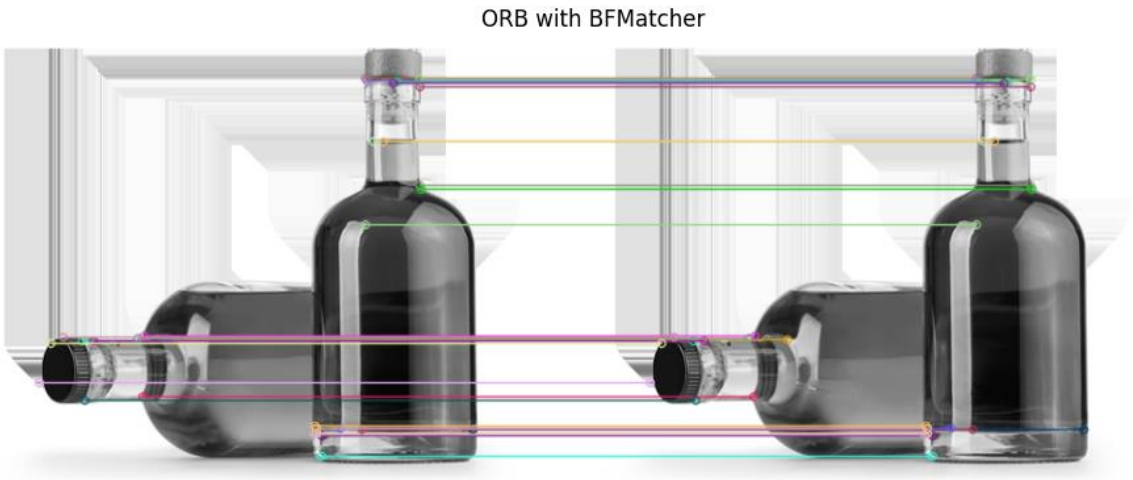
```
# ORB with BFMatcher (Free and Fast)
```

ASSIGMENT 2

```
import cv2

import numpy as np

import matplotlib.pyplot as plt



# Load the object and scene images

img_object = cv2.imread(r'C:\Users\Adeel\Desktop\Computer Vision\liquor-bottle-png-9.png', cv2.IMREAD_GRAYSCALE)

img_scene = cv2.imread(r'C:\Users\Adeel\Desktop\Computer Vision\liquor-bottle-png-9.png', cv2.IMREAD_GRAYSCALE)



# Initialize ORB detector

orb = cv2.ORB_create()



# Find keypoints and descriptors

kp1, des1 = orb.detectAndCompute(img_object, None)

kp2, des2 = orb.detectAndCompute(img_scene, None)



# Create BFMatcher and match descriptors

bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

matches = bf.match(des1, des2)



# Sort matches by distance

matches = sorted(matches, key=lambda x: x.distance)
```

```
# Draw matches

result = cv2.drawMatches(img_object, kp1, img_scene, kp2, matches[:30], None, flags=2)



# Show the result

plt.figure(figsize=(12,6))

plt.title("ORB with BFMatcher")

plt.imshow(result)

plt.axis('off')

plt.show()
```

Output:



Code :

```
# SIFT with FLANN Matcher (More Accurate, Heavier)

# Load the images

img_object = cv2.imread(r'C:\Users\Adeel\Desktop\Computer Vision\liquor-bottle-png-9.png', cv2.IMREAD_GRAYSCALE)

img_scene = cv2.imread(r'C:\Users\Adeel\Desktop\Computer Vision\liquor-bottle-png-9.png', cv2.IMREAD_GRAYSCALE)
```

ASSIGMENT 2

```
# Initialize SIFT detector

sift = cv2.SIFT_create()


# Detect keypoints and descriptors

kp1, des1 = sift.detectAndCompute(img_object, None)

kp2, des2 = sift.detectAndCompute(img_scene, None)


# FLANN parameters and matcher

FLANN_INDEX_KDTREE = 1

index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)

search_params = dict(checks=50)


flann = cv2.FlannBasedMatcher(index_params, search_params)

matches = flann.knnMatch(des1, des2, k=2)


# Apply Lowe's ratio test

good_matches = []

for m, n in matches:

    if m.distance < 0.75 * n.distance:

        good_matches.append(m)


# Draw matches

result = cv2.drawMatches(img_object, kp1, img_scene, kp2, good_matches, None, flags=2)
```

```
# Show the result
plt.figure(figsize=(12,6))
plt.title("SIFT with FLANN Matcher")
plt.imshow(result)
plt.axis('off')
plt.show()
```

Output: