Republic of the Philippines
Laguna State Polytechnic University
Province of Laguna

Exercises No. 2			
Topic:	**Module 2.0: Feature Extraction and Object Detection**	**Week No.**	6-7
Course Code:	**CSST106**	**Term:**	1st Semester
Course Title:	**Perception and Computer Vision**	**Academic Year:**	2024-2025
Student Name		**Section**	
Due date		**Points**	

Instructions:
Complete the following tasks using OpenCV and relevant libraries in Python (e.g., OpenCV, scikit-image). Each task should be implemented in a separate code block. Submit your Python script or notebook with all cells executed, outputs displayed, and brief explanations describing your approach, observations, and results.

SIFT Feature Extraction

SIFT (Scale-Invariant Feature Transform) detects important points, called **keypoints**, in an image. These keypoints represent distinct and unique features, such as corners or edges, that can be identified even if the image is resized, rotated, or transformed. SIFT generates a **descriptor** for each keypoint, which helps in matching these points across images.
The code first loads the image, converts it to grayscale (because many feature detectors work better on grayscale images), and then uses the SIFT algorithm to detect keypoints. The keypoints are visualized on the image.

Key Points:

- **Keypoints** are important image features.
- **Descriptors** are used to describe and match these keypoints.

Task 1: SIFT Feature Extraction

1. Load an image of your choice.
2. Use the **SIFT (Scale-Invariant Feature Transform)** algorithm to detect and compute keypoints and descriptors in the image.
3. Visualize the keypoints on the image and display the result.

Sample Code:

```
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('your_image.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize SIFT detector
sift = cv2.SIFT_create()

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

# Draw keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the image with keypoints
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.title('SIFT Keypoints')
plt.show()
```

SURF Feature Extraction

SURF (Speeded-Up Robust Features) is similar to SIFT but is optimized for speed. SURF focuses on finding features faster, making it useful for real-time applications. It also detects keypoints and generates descriptors but uses a different mathematical approach to SIFT.

In the code, SURF is used to detect keypoints in a grayscale image, and the keypoints are visualized similarly to SIFT. The performance of SURF is usually faster than SIFT, but it might miss certain keypoints that SIFT would detect.

Key Points:

- SURF is faster than SIFT.
- It can be a good choice for real-time applications.

Task 2: SURF Feature Extraction

1. Load a different image (or the same one).
2. Apply the **SURF (Speeded-Up Robust Features)** algorithm to detect and compute keypoints and descriptors.
3. Visualize and display the keypoints.

Sample Code:

```
# Load the image
image = cv2.imread('your_image.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize SURF detector (You might need OpenCV-contrib for SURF)
surf = cv2.xfeatures2d.SURF_create()

# Detect keypoints and descriptors
keypoints, descriptors = surf.detectAndCompute(gray_image, None)

# Draw keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the image with keypoints
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.title('SURF Keypoints')
plt.show()
```

ORB Feature Extraction

ORB (Oriented FAST and Rotated BRIEF) is a feature detection algorithm that is both fast and computationally less expensive than SIFT and SURF. It is ideal for real-time applications, particularly in mobile devices. ORB combines two methods: **FAST** (Features from Accelerated Segment Test) to detect keypoints and **BRIEF** (Binary Robust Independent Elementary Features) to compute descriptors.

The code uses ORB to detect keypoints and display them on the image. Unlike SIFT and SURF, ORB is more focused on speed and efficiency, which makes it suitable for applications that need to process images quickly.

Key Points:

- ORB is a fast alternative to SIFT and SURF.
- It's suitable for real-time and resource-constrained environments.

Task 3: ORB Feature Extraction (20 points)

1. Apply the **ORB (Oriented FAST and Rotated BRIEF)** algorithm to detect keypoints and compute descriptors on another image.
2. Visualize and display the keypoints.

Sample Code:

```
# Load the image
image = cv2.imread('your_image.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Initialize ORB detector
orb = cv2.ORB_create()

# Detect keypoints and descriptors
keypoints, descriptors = orb.detectAndCompute(gray_image, None)

# Draw keypoints on the image
image_with_keypoints = cv2.drawKeypoints(image, keypoints, None)

# Display the image with keypoints
plt.imshow(cv2.cvtColor(image_with_keypoints, cv2.COLOR_BGR2RGB))
plt.title('ORB Keypoints')
plt.show()
```

Feature Matching using SIFT

In this exercise, **feature matching** is used to find similar points between two images. After detecting keypoints using SIFT, the algorithm uses a **Brute-Force Matcher** to find matching keypoints between two images. The matcher compares the descriptors of the keypoints and finds pairs that are similar.

In the code, we load two images, detect their keypoints and descriptors using SIFT, and then use the matcher to draw lines between matching keypoints. The lines show which points in the first image correspond to points in the second image.

Key Points:

- Feature matching helps compare and find similarities between two images.
- The **Brute-Force Matcher** finds the closest matching descriptors.

Task 4: Feature Matching

1. Using the keypoints and descriptors obtained from the previous tasks (e.g., SIFT, SURF, or ORB), match the features between two different images using **Brute-Force Matching** or **FLANN (Fast Library for Approximate Nearest Neighbors)**.
2. Display the matched keypoints on both images.

Sample Code:

```
# Load two images
image1 = cv2.imread('image1.jpg', 0)
image2 = cv2.imread('image2.jpg', 0)

# Initialize SIFT detector
sift = cv2.SIFT_create()

# Find keypoints and descriptors with SIFT
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# Initialize the matcher
bf = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)

# Match descriptors
matches = bf.match(descriptors1, descriptors2)

# Sort matches by distance (best matches first)
matches = sorted(matches, key=lambda x: x.distance)

# Draw matches
image_matches = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches[:10], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

# Display the matches
plt.imshow(image_matches)
plt.title('Feature Matching with SIFT')
plt.show()
```

Real-World Applications (Image Stitching using Homography)

In this task, you use matched keypoints from two images to align or "stitch" them together. **Homography** is a mathematical transformation that maps points from one image to another, which is useful for aligning images taken from different angles or perspectives. This process is used in image stitching (e.g., creating panoramas), where you align and merge images to form a larger one.

The code uses the keypoints matched between two images and calculates the homography matrix. This matrix is then used to warp one image to align it with the other.

Key Points:

- **Homography** is used to align images.
- This is useful in applications like panoramic image creation or object recognition.

Task 5: Applications of Feature Matching

1. Apply feature matching to two images of the same scene taken from different angles or perspectives.
2. Use the matched features to align the images (e.g., using **homography** to warp one image onto another).

Sample Code:

```
# Load two images
image1 = cv2.imread('image1.jpg')
image2 = cv2.imread('image2.jpg')

# Convert to grayscale
gray1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
gray2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

# Detect keypoints and descriptors using SIFT
sift = cv2.SIFT_create()
keypoints1, descriptors1 = sift.detectAndCompute(gray1, None)
keypoints2, descriptors2 = sift.detectAndCompute(gray2, None)

# Match features using BFMatcher
bf = cv2.BFMatcher(cv2.NORM_L2)
matches = bf.knnMatch(descriptors1, descriptors2, k=2)
```

```python
# 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)

# Extract location of good matches
src_pts = np.float32(
    [keypoints1[m.queryIdx].pt for m in good_matches]
).reshape(-1, 1, 2)
dst_pts = np.float32(
    [keypoints2[m.trainIdx].pt for m in good_matches]
).reshape(-1, 1, 2)

# Find homography matrix
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

# Warp one image to align with the other
h, w, _ = image1.shape
result = cv2.warpPerspective(image1, M, (w, h))

# Display the result
plt.imshow(cv2.cvtColor(result, cv2.COLOR_BGR2RGB))
plt.title('Image Alignment using Homography')
plt.show()
```

Combining SIFT and ORB

By combining two feature extraction methods (SIFT and ORB), you can take advantage of the strengths of both. For example, SIFT is more accurate, but ORB is faster. By detecting keypoints using both methods, you can compare how they perform on different types of images and possibly combine their outputs for more robust feature detection and matching.

In the code, we extract keypoints from two images using both SIFT and ORB, and then you can use a matcher to compare and match the features detected by both methods.

Key Points:

- Combining methods can improve performance in some applications.
- SIFT is accurate, while ORB is fast, making them complementary in certain tasks.

Task 6: Combining Feature Extraction Methods (10 points)

1. Combine multiple feature extraction methods (e.g., SIFT + ORB) to extract features and match them between two images.
2. Display the combined result.

Sample Code:

```
# Use SIFT and ORB to extract features from two images

# Load two images
image1 = cv2.imread('image1.jpg', 0)
image2 = cv2.imread('image2.jpg', 0)

# SIFT detector
sift = cv2.SIFT_create()
keypoints1_sift, descriptors1_sift = sift.detectAndCompute(image1, None)
keypoints2_sift, descriptors2_sift = sift.detectAndCompute(image2, None)

# ORB detector
orb = cv2.ORB_create()
keypoints1_orb, descriptors1_orb = orb.detectAndCompute(image1, None)
keypoints2_orb, descriptors2_orb = orb.detectAndCompute(image2, None)

# You can match keypoints from both SIFT and ORB using a matcher of your choice.
```

Overall Understanding:

- **SIFT** is accurate for detecting and matching keypoints even in transformed images (scaled, rotated).
- **SURF** is faster than SIFT but still effective for keypoint detection.
- **ORB** is highly efficient and suited for real-time applications.
- **Feature Matching** is essential in comparing different images to find common objects or align them.
- **Homography** is used in aligning images, such as stitching images together to form a panorama.

These methods are foundational in computer vision tasks, such as object detection, image stitching, and image recognition.

Submission Instructions:

1. **Repository Setup**:
    - Create or navigate to the GitHub repository named CSST106-Perception and Computer Vision.
    - Inside the repository, create a folder specifically for this assignment (e.g., Exercise-2).

2. **Submission Format**:
    - **Processed Images**: Save and upload all images generated by your code (e.g., keypoint detection, feature matching, homography results) to the Exercise-2 folder in the repository.
    - **Code**: Save your Python scripts or Jupyter Notebook files containing your implementation and upload them to the same folder.
    - **Documentation**: Write a brief document (README.md or PDF) explaining your approach, observations, and results for each task. This document should provide enough detail for someone reviewing your work to understand your process.
    - **Folder Organization**:
        - Ensure that all content is properly labeled and structured for easy navigation.
        - You might have subfolders like:
            - images/ (for processed images)
            - code/ (for Python code or Jupyter notebooks)
            - documentation/ (for explanations, README.md, etc.)

3. **Filename Format**:
    - For all files (processed images, code, and documentation), use the following filename format: **[SECTION-BERNARDINO-EXER2]** 4D-BERNARDINO-EXER2

    Example filenames:
        - 4D-BERNARDINO-EXER2.py (for Python script)
        - 4D-BERNARDINO-EXER2.ipynb (for Jupyter Notebook)
        - 4D-BERNARDINO-EXER2-keypoints.jpg (for processed images)
        - 4D-BERNARDINO-EXER2-documentation.md (for documentation)

4. **Penalties**:
    - **Incorrect Filename**: If the filename format is not followed, there will be a 5-point deduction.
    - **Late Submission**: There will be a 5-point deduction per day for late submissions.
    - **Cheating/Plagiarism**: Any evidence of cheating or plagiarism will result in additional penalties according to the university's academic integrity policies.

By adhering to these submission guidelines and formatting, you will avoid penalties and ensure your work is well-received.

Rubric for Feature Extraction and Object Detection:

Criteria	Excellent (90-100%)	Good (75-89%)	Satisfactory (60-74%)	Needs Improvement (0-59%)
Task 1: SIFT Extraction and Explanation	Accurate keypoint detection, clear explanation of SIFT.	Minor issues in visualization or explanation	Basic extraction, unclear explanation	Incorrect extraction, no explanation
Task 2: SURF Extraction and Explanation	Accurate, efficient SURF detection, good comparison	Minor issues with SURF or comparison	Basic explanation	Incorrect or no explanation
Task 3: ORB Extraction and Explanation	Accurate, clear visualization and explanation of ORB	Minor issues	Basic explanation	Incorrect ORB extraction
Task 4: Feature Matching and Visualization	Clear matching of keypoints with insights	Minor visualization or matching issues	Minimal explanation	Incorrect or no matching
Task 5: Applications of Feature Matching	Demonstrates practical application, insightful explanation.	Minor issues in explanation	Basic explanation	No practical application
Task 6: Combining SIFT and ORB	Correct, creative solution, well-explained	Some creativity but lacks depth	Basic attempt	Incorrect or no solution

This exercise aligns with topics 2.1 and 2.2 by implementing feature extraction methods (SIFT, SURF, ORB) and applying them in real-world scenarios such as feature matching and object detection.