4B-BAUTISTA-MP 3

Manualy installing openCV repo that support SURF  
  
!apt-get install -y cmake libopencv-dev build-essential git pkg-config libgtk-3-dev \

libavcodec-dev libavformat-dev libswscale-dev libtbb2 libtbb-dev libjpeg-dev \

libpng-dev libtiff-dev libdc1394-22-dev libv4l-dev v4l-utils \

libxvidcore-dev libx264-dev libxine2-dev gstreamer1.0-tools \

libgstreamer-plugins-base1.0-dev libgstreamer-plugins-good1.0-dev \

libtesseract-dev libopenblas-dev liblapacke-dev checkinstall

: openCV repository that support the oldversion of SURF Algorithm .

Repo Cloning

!git clone https://github.com/opencv/opencv.git

!git clone <https://github.com/opencv/opencv_contrib.git>  
: Github repository that will be used in the activity.

Installing eneything that are needed for OpenCV to run SURF Algorithm

%cd opencv

!mkdir build

%cd build

!cmake -D CMAKE\_BUILD\_TYPE=RELEASE \

-D CMAKE\_INSTALL\_PREFIX=/usr/local \

-D OPENCV\_ENABLE\_NONFREE=ON \

-D OPENCV\_EXTRA\_MODULES\_PATH=../../opencv\_contrib/modules \

-D BUILD\_EXAMPLES=ON ..

!make -j8

!make install

: install the requirement for the process of SURF Algo and the location of the installation .

Restart Runtime  
  
import os

os.kill(os.getpid(), 9)  
  
; Kill the operation to enable the process and the installation of the SURF algorithm.

**Machine Problem No. 3: Feature Extraction and Object Detection**

The objective of this machine problem is to implement and compare the three feature extraction methods (**SIFT**, **SURF**, and **ORB**) in a single task. You will use these methods for feature matching between two images, then perform image alignment using **Homography** to warp one image onto the other.

**Problem Description:**

You are tasked with loading two images and performing the following steps:

1. Extract keypoints and descriptors from both images using **SIFT**, **SURF**, and **ORB**.
2. Perform feature matching between the two images using both **Brute-Force Matcher** and **FLANN Matcher**.
3. Use the matched keypoints to calculate a **Homography matrix** and align the two images.
4. Compare the performance of SIFT, SURF, and ORB in terms of feature matching accuracy and speed.

**Step 1: Load Images**

* Load two images of your choice that depict the same scene or object but from different angles.

# Load two images of the same scene/object but from different angles

img1 = cv2.imread('img1.jpg', cv2.IMREAD\_GRAYSCALE)

img2 = cv2.imread('img2.jpg', cv2.IMREAD\_GRAYSCALE)

# Display the loaded images

plt.subplot(1, 2, 1), plt.imshow(img1, cmap='gray'), plt.title('Image 1')

plt.subplot(1, 2, 2), plt.imshow(img2, cmap='gray'), plt.title('Image 2')

plt.show()

**Step 2: Extract Keypoints and Descriptors Using SIFT, SURF, and ORB**

* Apply the **SIFT** algorithm to detect keypoints and compute descriptors for both images.

# Initialize SIFT

sift = cv2.SIFT\_create()

# Detect keypoints and descriptors

kp1\_sift, des1\_sift = sift.detectAndCompute(img1, None)

kp2\_sift, des2\_sift = sift.detectAndCompute(img2, None)

# Draw keypoints

img1\_sift = cv2.drawKeypoints(img1, kp1\_sift, None)

img2\_sift = cv2.drawKeypoints(img2, kp2\_sift, None)

# Save and display images

cv2.imwrite('sift\_keypoints1.jpg', img1\_sift)

cv2.imwrite('sift\_keypoints2.jpg', img2\_sift)

plt.subplot(1, 2, 1), plt.imshow(img1\_sift), plt.title('SIFT Keypoints Image 1')

plt.subplot(1, 2, 2), plt.imshow(img2\_sift), plt.title('SIFT Keypoints Image 2')

plt.show()

* Apply the **SURF** algorithm to do the same.

image\_path = 'img2.jpg'

image = Image.open(image\_path)

image\_np = np.array(image)

#Convert to grayscale for keypoint detection.

gray\_image = cv2.cvtColor(image\_np, cv2.COLOR\_RGB2GRAY)

#Initialize the SURF.

surf = cv2.xfeatures2d.SURF\_create()

#Detect keypoints and compute descriptors.

keypoints, descriptors = surf.detectAndCompute(gray\_image, None)

#Draw the detected keypoints.

image\_with\_keypoints = cv2.drawKeypoints(image\_np, keypoints, None, (255, 0, 0), flags=cv2.DRAW\_MATCHES\_FLAGS\_DRAW\_RICH\_KEYPOINTS)

#Display Original Image.

plt.figure(figsize=(10, 6))

plt.subplot(1, 2, 1)

plt.imshow(image\_np)

plt.title(r'$\bf{Original\ Image}$')

#Display SURF Keypoints.

plt.subplot(1, 2, 2)

plt.imshow(image\_with\_keypoints)

plt.title(r'$\bf{SURF\ Keypoints}$')

plt.tight\_layout()

plt.show()

* Finally, apply **ORB** to extract keypoints and descriptors.

# Initialize ORB

orb = cv2.ORB\_create()

# Detect keypoints and descriptors

kp1\_orb, des1\_orb = orb.detectAndCompute(img1, None)

kp2\_orb, des2\_orb = orb.detectAndCompute(img2, None)

# Draw keypoints

img1\_orb = cv2.drawKeypoints(img1, kp1\_orb, None)

img2\_orb = cv2.drawKeypoints(img2, kp2\_orb, None)

# Save and display images

cv2.imwrite('orb\_keypoints1.jpg', img1\_orb)

cv2.imwrite('orb\_keypoints2.jpg', img2\_orb)

plt.subplot(1, 2, 1), plt.imshow(img1\_orb), plt.title('ORB Keypoints Image 1')

plt.subplot(1, 2, 2), plt.imshow(img2\_orb), plt.title('ORB Keypoints Image 2')

plt.show()

**Step 3: Feature Matching with Brute-Force and FLANN**

* Match the descriptors between the two images using **Brute-Force Matcher**.

# Brute-Force matcher for SIFT

bf = cv2.BFMatcher(cv2.NORM\_L2, crossCheck=True)

matches\_sift = bf.match(des1\_sift, des2\_sift)

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

# Draw matches

img\_matches\_sift = cv2.drawMatches(img1, kp1\_sift, img2, kp2\_sift, matches\_sift[:50], None)

# Save and display matches

cv2.imwrite('sift\_bf\_match.jpg', img\_matches\_sift)

plt.imshow(img\_matches\_sift)

plt.title('SIFT BF Matcher')

plt.show()

* Repeat the process using the **FLANN Matcher**.

# FLANN matcher for SIFT

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\_sift\_flann = flann.knnMatch(des1\_sift, des2\_sift, k=2)

# Apply ratio test

good\_matches = []

for m, n in matches\_sift\_flann:

    if m.distance < 0.75 \* n.distance:

        good\_matches.append(m)

# Draw matches

img\_matches\_sift\_flann = cv2.drawMatches(img1, kp1\_sift, img2, kp2\_sift, good\_matches[:50], None)

# Save and display matches

cv2.imwrite('sift\_flann\_match.jpg', img\_matches\_sift\_flann)

plt.imshow(img\_matches\_sift\_flann)

plt.title('SIFT FLANN Matcher')

plt.show()

**Step 4: Image Alignment Using Homography**

* Use the matched keypoints from **SIFT** (or any other method) to compute a **homography matrix**.
* Use this matrix to warp one image onto the other.
* Display and save the aligned and warped images.

# Extract location of good matches

src\_pts = np.float32([kp1\_sift[m.queryIdx].pt for m in good\_matches]).reshape(-1, 1, 2)

dst\_pts = np.float32([kp2\_sift[m.trainIdx].pt for m in good\_matches]).reshape(-1, 1, 2)

# Compute Homography matrix

H, mask = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC, 5.0)

# Warp image 1 onto image 2

height, width = img2.shape

aligned\_image = cv2.warpPerspective(img1, H, (width, height))

# Save and display aligned image

cv2.imwrite('aligned\_image.jpg', aligned\_image)

plt.imshow(aligned\_image, cmap='gray')

plt.title('Aligned Image')

plt.show()

**Step 5: Performance Analysis**

1. **Compare the Results**:
   * Analyze the performance of **SIFT**, **SURF**, and **ORB** in terms of keypoint detection accuracy, number of keypoints detected, and speed.
   * Comment on the effectiveness of **Brute-Force Matcher** versus **FLANN Matcher** for feature matching.

import time

# Function to measure keypoint extraction time

def measure\_keypoint\_extraction(detector, img, name):

    start\_time = time.time()

    keypoints, descriptors = detector.detectAndCompute(img, None)

    end\_time = time.time()

    print(f"{name} Keypoints: {len(keypoints)}")

    print(f"{name} Keypoint Extraction Time: {end\_time - start\_time:.4f} seconds")

    return keypoints, descriptors

# Function to measure matching time

def measure\_matching(matcher, descriptors1, descriptors2, name):

    start\_time = time.time()

    matches = matcher.match(descriptors1, descriptors2)

    end\_time = time.time()

    print(f"{name} Matches: {len(matches)}")

    print(f"{name} Matching Time: {end\_time - start\_time:.4f} seconds")

    return matches

# Load images

img1 = cv2.imread('img1.jpg', cv2.IMREAD\_GRAYSCALE)

img2 = cv2.imread('img2.jpg', cv2.IMREAD\_GRAYSCALE)

# Initialize detectors

sift = cv2.SIFT\_create()

surf = cv2.xfeatures2d.SURF\_create()  # Ensure OpenCV supports SURF

orb = cv2.ORB\_create()

# Brute-Force Matchers

bf\_sift = cv2.BFMatcher(cv2.NORM\_L2, crossCheck=True)

bf\_orb = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)

# FLANN parameters

FLANN\_INDEX\_KDTREE = 1

index\_params = dict(algorithm=FLANN\_INDEX\_KDTREE, trees=5)

search\_params = dict(checks=50)

flann\_sift = cv2.FlannBasedMatcher(index\_params, search\_params)

# SIFT Analysis

print("\nSIFT Analysis")

kp\_sift\_1, des\_sift\_1 = measure\_keypoint\_extraction(sift, img1, "SIFT (Image 1)")

kp\_sift\_2, des\_sift\_2 = measure\_keypoint\_extraction(sift, img2, "SIFT (Image 2)")

matches\_sift\_bf = measure\_matching(bf\_sift, des\_sift\_1, des\_sift\_2, "SIFT Brute-Force Matcher")

# SURF Analysis

print("\nSURF Analysis")

kp\_surf\_1, des\_surf\_1 = measure\_keypoint\_extraction(surf, img1, "SURF (Image 1)")

kp\_surf\_2, des\_surf\_2 = measure\_keypoint\_extraction(surf, img2, "SURF (Image 2)")

matches\_surf\_bf = measure\_matching(bf\_sift, des\_surf\_1, des\_surf\_2, "SURF Brute-Force Matcher")

# ORB Analysis

print("\nORB Analysis")

kp\_orb\_1, des\_orb\_1 = measure\_keypoint\_extraction(orb, img1, "ORB (Image 1)")

kp\_orb\_2, des\_orb\_2 = measure\_keypoint\_extraction(orb, img2, "ORB (Image 2)")

matches\_orb\_bf = measure\_matching(bf\_orb, des\_orb\_1, des\_orb\_2, "ORB Brute-Force Matcher")

# SIFT FLANN Matching

print("\nSIFT FLANN Matching Analysis")

start\_time = time.time()

matches\_sift\_flann = flann\_sift.knnMatch(des\_sift\_1, des\_sift\_2, k=2)

good\_matches\_sift = []

for m, n in matches\_sift\_flann:

    if m.distance < 0.7 \* n.distance:

        good\_matches\_sift.append(m)

end\_time = time.time()

print(f"SIFT FLANN Matches: {len(good\_matches\_sift)}")

print(f"SIFT FLANN Matching Time: {end\_time - start\_time:.4f} seconds")

# Performance Summary

print("\nPerformance Summary:")

print(f"SIFT Keypoints (Image 1): {len(kp\_sift\_1)}, (Image 2): {len(kp\_sift\_2)}")

print(f"SURF Keypoints (Image 1): {len(kp\_surf\_1)}, (Image 2): {len(kp\_surf\_2)}")

print(f"ORB Keypoints (Image 1): {len(kp\_orb\_1)}, (Image 2): {len(kp\_orb\_2)}")

print(f"SIFT BF Matches: {len(matches\_sift\_bf)}, SURF BF Matches: {len(matches\_surf\_bf)}, ORB BF Matches: {len(matches\_orb\_bf)}")

print(f"SIFT FLANN Matches: {len(good\_matches\_sift)}")

Result :  
  
SIFT Analysis

SIFT (Image 1) Keypoints: 398

SIFT (Image 1) Keypoint Extraction Time: 0.1153 seconds

SIFT (Image 2) Keypoints: 926

SIFT (Image 2) Keypoint Extraction Time: 0.0797 seconds

SIFT Brute-Force Matcher Matches: 109

SIFT Brute-Force Matcher Matching Time: 0.0229 seconds

SURF Analysis

SURF (Image 1) Keypoints: 570

SURF (Image 1) Keypoint Extraction Time: 0.1946 seconds

SURF (Image 2) Keypoints: 1709

SURF (Image 2) Keypoint Extraction Time: 0.2741 seconds

SURF Brute-Force Matcher Matches: 142

SURF Brute-Force Matcher Matching Time: 0.0334 seconds

ORB Analysis

ORB (Image 1) Keypoints: 477

ORB (Image 1) Keypoint Extraction Time: 0.0067 seconds

ORB (Image 2) Keypoints: 500

ORB (Image 2) Keypoint Extraction Time: 0.0065 seconds

ORB Brute-Force Matcher Matches: 99

ORB Brute-Force Matcher Matching Time: 0.0101 seconds

SIFT FLANN Matching Analysis

SIFT FLANN Matches: 0

SIFT FLANN Matching Time: 0.0177 seconds

Performance Summary:

SIFT Keypoints (Image 1): 398, (Image 2): 926

SURF Keypoints (Image 1): 570, (Image 2): 1709

ORB Keypoints (Image 1): 477, (Image 2): 500

SIFT BF Matches: 109, SURF BF Matches: 142, ORB BF Matches: 99

SIFT FLANN Matches: 0