Image Matching Techniques in OpenCV-Python

OpenCV in Python offers a variety of image matching and feature detection algorithms. While the "best" algorithm can vary depending on your specific use case and requirements, here are some of the commonly used and effective ones.

- **ORB (Oriented FAST and Rotated BRIEF):**

  ORB is a popular and efficient feature detection and matching algorithm. It is suitable for tasks that involve detecting and matching keypoints between images. It combines speed with good performance and can handle scale and rotation changes.

- **SIFT (Scale-Invariant Feature Transform):**

  SIFT is a robust algorithm for keypoint detection and matching. It is known for its scale and rotation invariance, making it suitable for various computer vision tasks. Note that SIFT is patented, and depending on your use case, you may need to consider licensing issues.

- **SURF (Speeded-Up Robust Features):**

  SURF is another feature detection and matching algorithm that offers speed improvements compared to SIFT while maintaining robustness to scale and rotation changes.

- **FLANN (Fast Library for Approximate Nearest Neighbors):**

  FLANN is not a feature detection algorithm itself but rather a library that provides efficient methods for approximate nearest neighbor searches. It can be used in combination with feature detectors like SIFT or SURF to speed up the matching process.

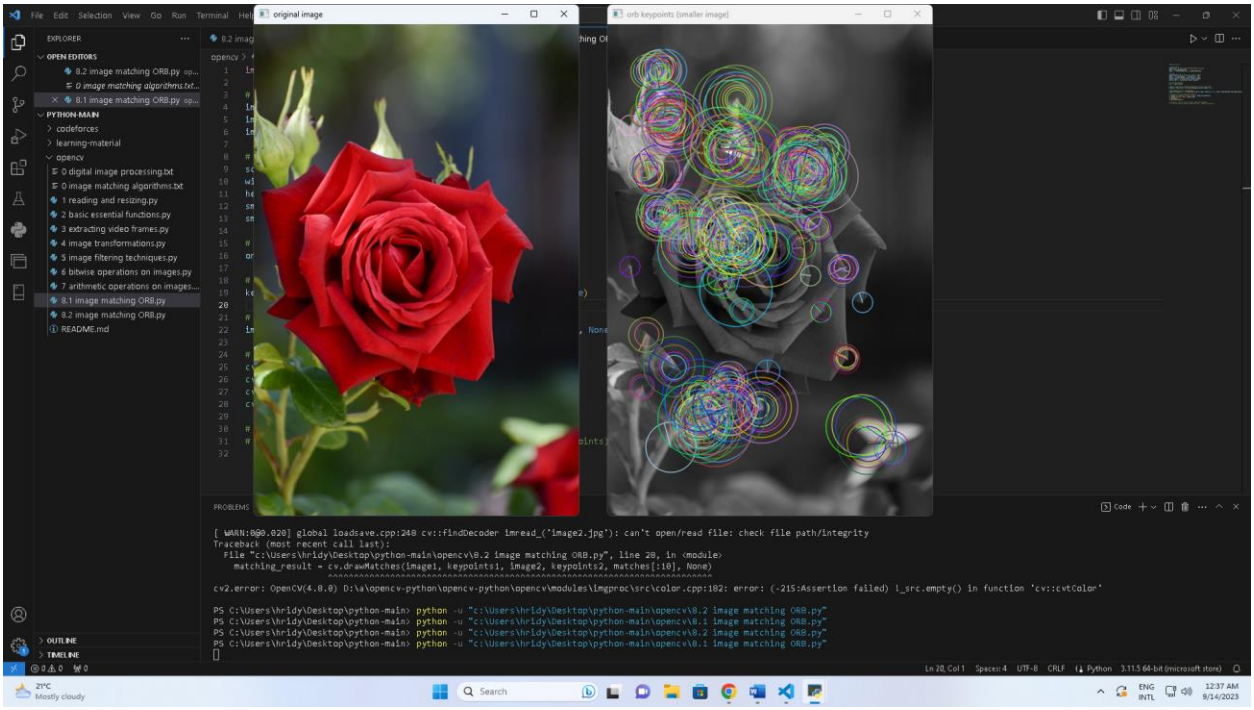
- **Brute-Force Matcher:**
  OpenCV also provides a simple brute-force matcher that can be used with feature descriptors like SIFT and SURF. It compares all features from one image to all features in another image, making it easy to implement but potentially slower for large datasets.
- **AKAZE (Accelerated-KAZE):**
  AKAZE is a feature detection and description algorithm that is an improved version of KAZE. It's designed to be faster and more robust to various image transformations.

ORB KEYPOINT DETECTION IN A SINGLE IMAGE

```
import cv2 as cv

# image loading
image_path = 'C:/Users/hridy/Desktop/images/rose1.jpg'
img = cv.imread(image_path)
image = cv.imread(image_path, cv.IMREAD_GRAYSCALE)

# resize
scale_percent = 50  # adjust this value as needed
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
smaller_image = cv.resize(image, (width, height))
smaller_img = cv.resize(img, (width, height))

# create an ORB object
orb = cv.ORB_create()

# detect keypoints and compute descriptors for the resized image
keypoints, descriptors = orb.detectAndCompute(smaller_image, None)

# draw keypoints on the resized image
image_with_keypoints = cv.drawKeypoints(smaller_image, keypoints, None,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

# display the resized image with keypoints
cv.imshow('orb keypoints (smaller image)', image_with_keypoints)
cv.imshow('original image', smaller_img)
cv.waitKey(0)
```

```
cv.destroyAllWindows()

# optionally, you can save the smaller image with keypoints
# cv.imwrite('smaller_image_with_keypoints.jpg', image_with_keypoints)
```

OUTPUT

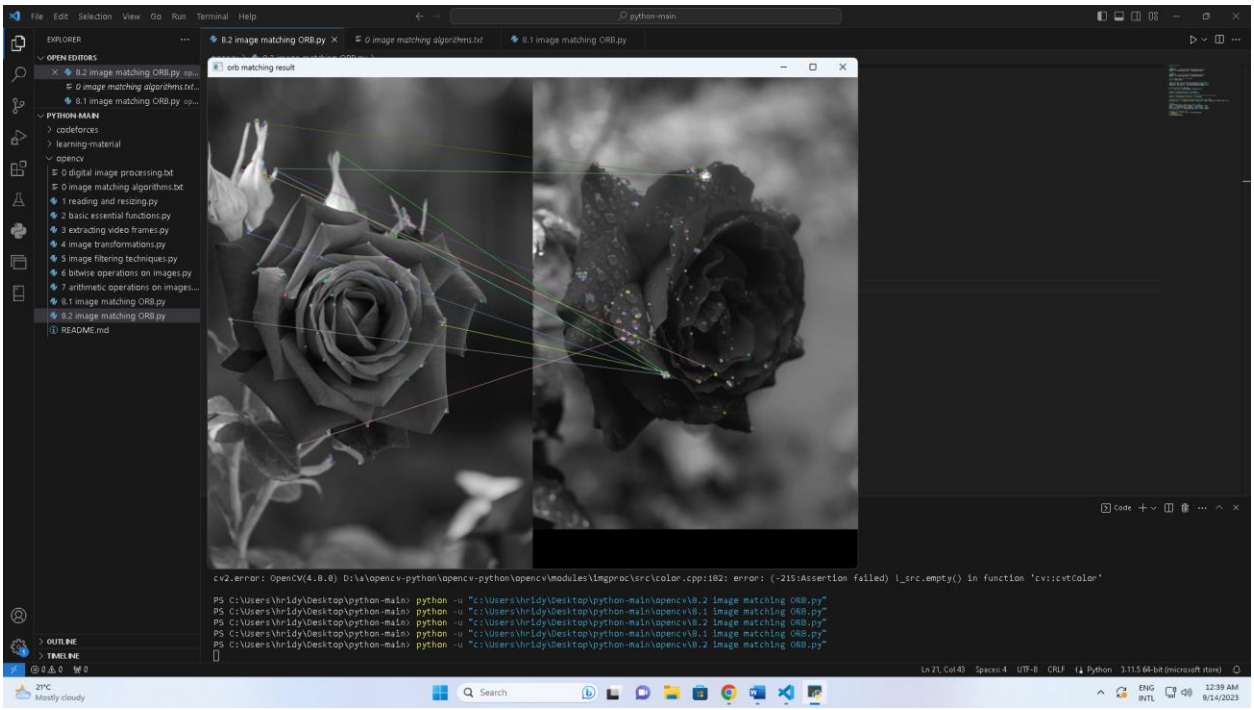


ORB KEYPOINT COMPARISON IN TWO IMAGES

```
import cv2 as cv

# image1
image1_path = 'C:/Users/hridy/Desktop/images/rose1.jpg'
image1 = cv.imread(image1_path, cv.IMREAD_GRAYSCALE)

# image2
image2_path = 'C:/Users/hridy/Desktop/images/rose2.jpg'
image2 = cv.imread(image2_path, cv.IMREAD_GRAYSCALE)

# create ORB objects for both images
orb = cv.ORB_create()

# detect keypoints and compute descriptors for both images
keypoints1, descriptors1 = orb.detectAndCompute(image1, None)
keypoints2, descriptors2 = orb.detectAndCompute(image2, None)
```

```python
# create a brute-force matcher
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

# match descriptors between the two images
matches = bf.match(descriptors1, descriptors2)

# sort the matches by their distance (lower distance means better match)
matches = sorted(matches, key=lambda x: x.distance)

# draw the first 10 matches (you can adjust the number as needed)
matching_result = cv.drawMatches(image1, keypoints1, image2, keypoints2,
matches[:10], None)

# resize
scale_percent = 50  # adjust this value as needed
width = int(matching_result.shape[1] * scale_percent / 100)
height = int(matching_result.shape[0] * scale_percent / 100)
final_comparison = cv.resize(matching_result, (width, height))

# display the matching result
cv.imshow('orb matching result', final_comparison)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT

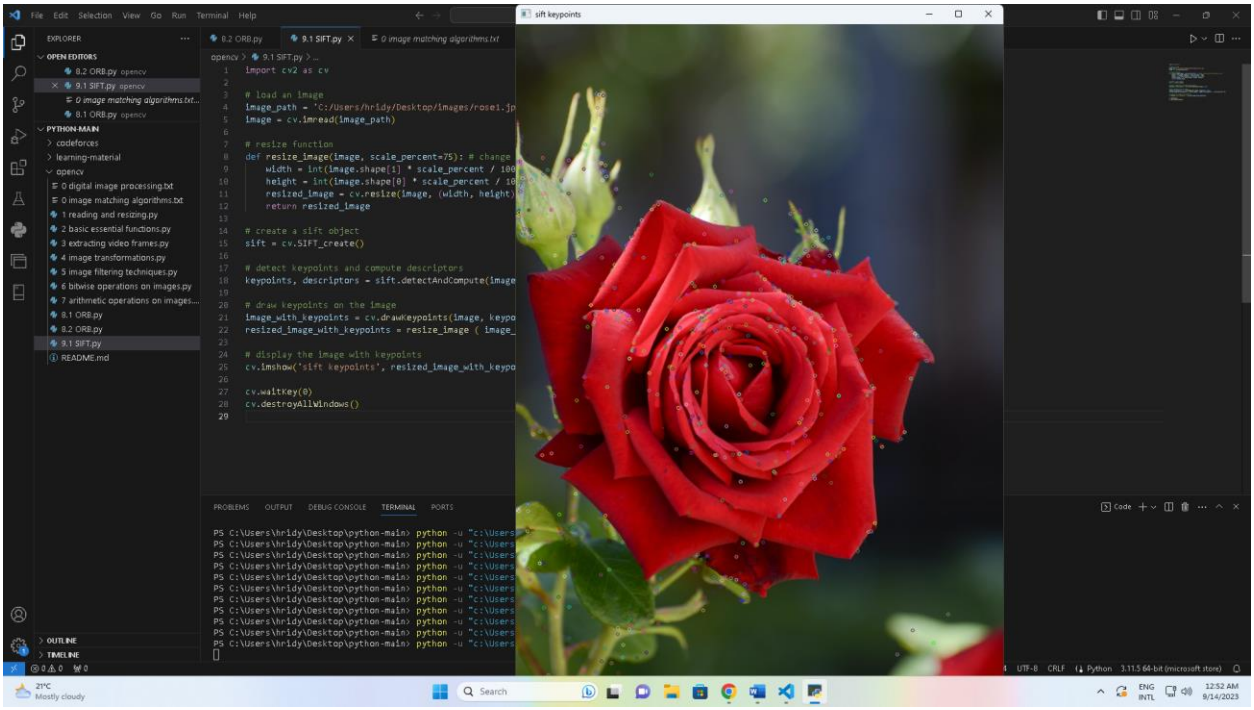


SIFT KEYPOINT DETECTION IN ONE IMAGE

```
import cv2 as cv

# load an image
image_path = 'C:/Users/hridy/Desktop/images/rose1.jpg'
image = cv.imread(image_path)

# resize function
def resize_image(image, scale_percent=75): # change scale percent according to
use
    width = int(image.shape[1] * scale_percent / 100)
    height = int(image.shape[0] * scale_percent / 100)
    resized_image = cv.resize(image, (width, height))
    return resized_image

# create a sift object
sift = cv.SIFT_create()

# detect keypoints and compute descriptors
```

```
keypoints, descriptors = sift.detectAndCompute(image, None)

# draw keypoints on the image
image_with_keypoints = cv.drawKeypoints(image, keypoints, None)
resized_image_with_keypoints = resize_image ( image_with_keypoints ,
scale_percent= 75)

# display the image with keypoints
cv.imshow('sift keypoints', resized_image_with_keypoints)

cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT

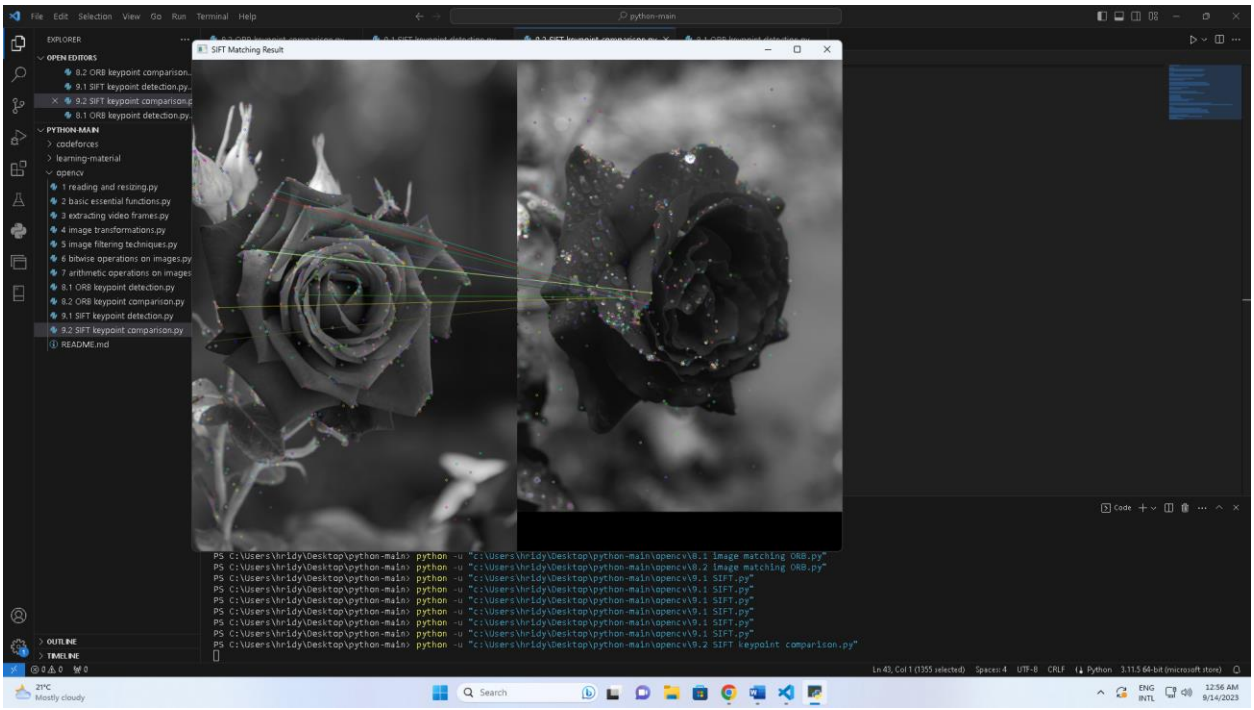


SIFT KEYPOINT COMPARISON WITH TWO IMAGES

```
import cv2 as cv

# image1
image1_path = 'C:/Users/hridy/Desktop/images/rose1.jpg'
image1 = cv.imread(image1_path, cv.IMREAD_GRAYSCALE)
```

```
# image2
image2_path = 'C:/Users/hridy/Desktop/images/rose2.jpg'
image2 = cv.imread(image2_path, cv.IMREAD_GRAYSCALE)

# create SIFT objects for both images
sift = cv.SIFT_create()

# detect keypoints and compute descriptors for both images
keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# create a brute-force matcher
bf = cv.BFMatcher()

# match descriptors between the two images
matches = bf.knnMatch(descriptors1, descriptors2, k=2)

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

# Draw the first 10 matches (you can adjust the number as needed)
matching_result = cv.drawMatches(image1, keypoints1, image2, keypoints2,
good_matches[:10], None)

# Resize
scale_percent = 50  # adjust this value as needed
width = int(matching_result.shape[1] * scale_percent / 100)
height = int(matching_result.shape[0] * scale_percent / 100)
final_comparison = cv.resize(matching_result, (width, height))

# Display the matching result
cv.imshow('SIFT Matching Result', final_comparison)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT



SURF

Unable to implement SURF since it is patented and not free to use , as well as

Encountering the following errors

1. **module 'cv2' has no attribute 'SURF_create'**
2. **module 'cv2' has no attribute 'xfeatures2d'**

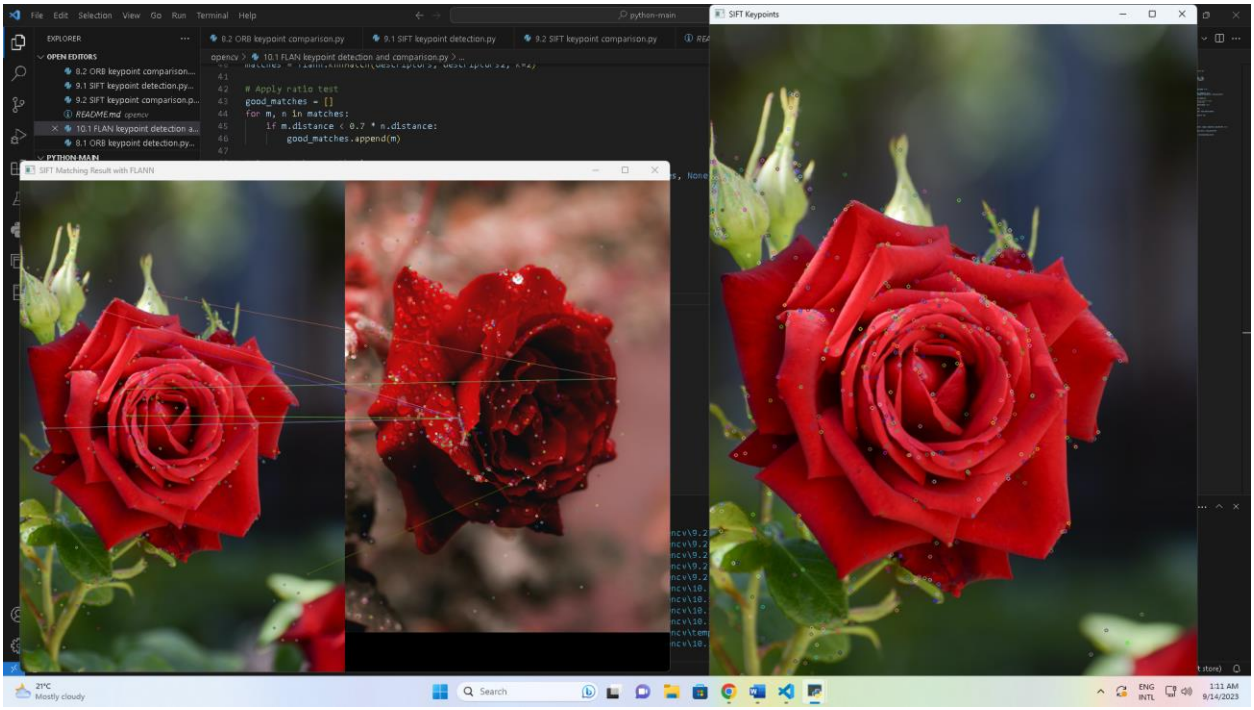
FLANN WITH SIFT

```
# FLANN WITH SIFT

import cv2 as cv

# Load an image
image_path = 'C:/Users/hridy/Desktop/images/rose1.jpg'
image = cv.imread(image_path)

# Resize function
def resize_image(image, scale_percent=50):
```

```
    width = int(image.shape[1] * scale_percent / 100)
    height = int(image.shape[0] * scale_percent / 100)
    resized_image = cv.resize(image, (width, height))
    return resized_image

# Create a SIFT object
sift = cv.SIFT_create()

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

# Draw keypoints on the image
image_with_keypoints = cv.drawKeypoints(image, keypoints, None)
resized_image_with_keypoints = resize_image(image_with_keypoints,
scale_percent=75)

# Display the image with SIFT keypoints
cv.imshow('SIFT Keypoints', resized_image_with_keypoints)

# Load another image for matching (you can load a different image)
image2_path = 'C:/Users/hridy/Desktop/images/rose2.jpg'
image2 = cv.imread(image2_path)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# Create FLANN-based matcher
flann_index_params = dict(algorithm=0, trees=5)
flann_search_params = dict(checks=50)
flann = cv.FlannBasedMatcher(flann_index_params, flann_search_params)

# Match descriptors using FLANN
matches = flann.knnMatch(descriptors, descriptors2, k=2)

# Apply ratio test
good_matches = []
for m, n in matches:
    if m.distance < 0.7 * n.distance:
        good_matches.append(m)

# Draw matches on the images
matching_result = cv.drawMatches(image, keypoints, image2, keypoints2,
good_matches, None)

# Resize the matching result
resized_matching_result = resize_image(matching_result, scale_percent=50)
```

```
# Display the matching result
cv.imshow('SIFT Matching Result with FLANN', resized_matching_result)

cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT

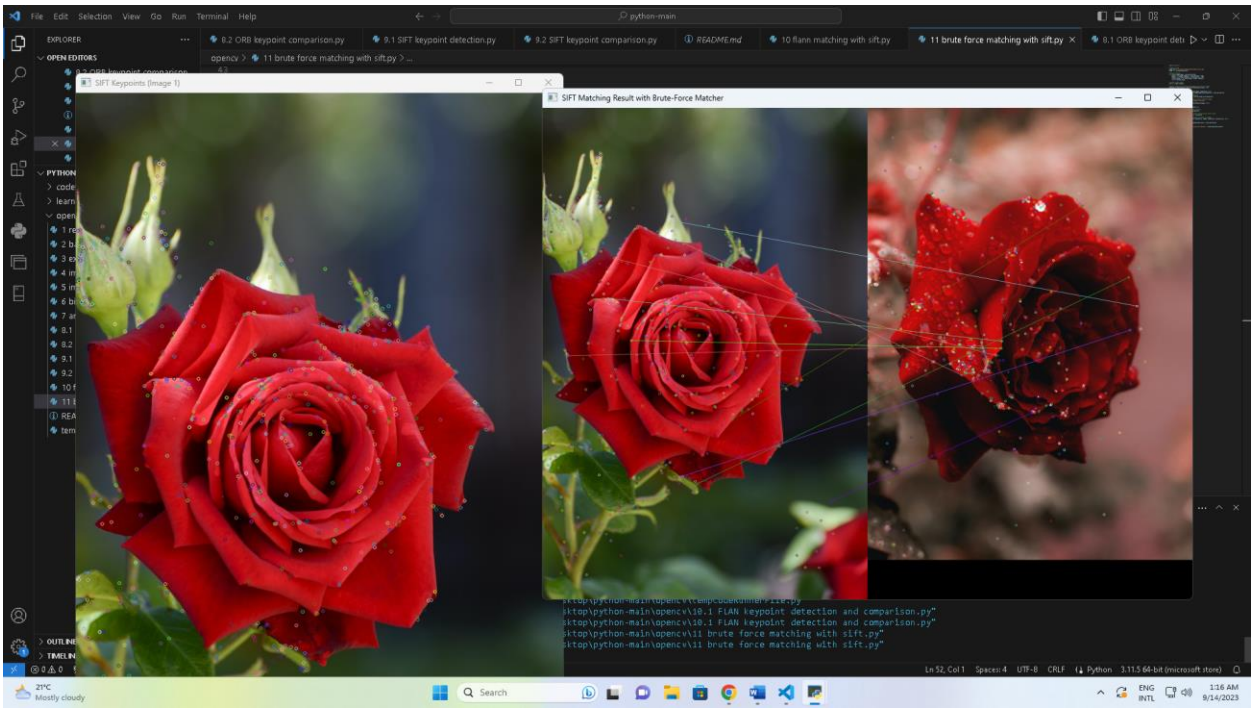


BRUTE FORCE WITH SIFT

```
import cv2 as cv

# Load an image
image_path = 'C:/Users/hridy/Desktop/images/rose1.jpg'
image = cv.imread(image_path)

# Resize function
def resize_image(image, scale_percent=50):
    width = int(image.shape[1] * scale_percent / 100)
    height = int(image.shape[0] * scale_percent / 100)
    resized_image = cv.resize(image, (width, height))
    return resized_image
```

```
# Create a SIFT object
sift = cv.SIFT_create()

# Detect keypoints and compute descriptors for the first image
keypoints1, descriptors1 = sift.detectAndCompute(image, None)

# Draw keypoints on the first image
image_with_keypoints1 = cv.drawKeypoints(image, keypoints1, None)
resized_image_with_keypoints1 = resize_image(image_with_keypoints1,
scale_percent=75)

# Display the first image with SIFT keypoints
cv.imshow('SIFT Keypoints (Image 1)', resized_image_with_keypoints1)

# Load another image for matching (you can load a different image)
image2_path = 'C:/Users/hridy/Desktop/images/rose2.jpg'
image2 = cv.imread(image2_path)
keypoints2, descriptors2 = sift.detectAndCompute(image2, None)

# Create a Brute-Force Matcher
bf = cv.BFMatcher(cv.NORM_L2, crossCheck=True)

# Match descriptors using the Brute-Force Matcher
matches = bf.match(descriptors1, descriptors2)

# Sort the matches by their distance (lower distance means better match)
matches = sorted(matches, key=lambda x: x.distance)

# Draw the first 10 matches (you can adjust the number as needed)
matching_result = cv.drawMatches(image, keypoints1, image2, keypoints2,
matches[:10], None)

# Resize the matching result
resized_matching_result = resize_image(matching_result, scale_percent=50)

# Display the matching result
cv.imshow('SIFT Matching Result with Brute-Force Matcher',
resized_matching_result)

cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT



AKAZE KEYPOINT DETECTION AND COMPARISON

```
import cv2 as cv

# Load two images for comparison
image1_path = 'C:/Users/hridy/Desktop/images/rose1.jpg'
image2_path = 'C:/Users/hridy/Desktop/images/rose2.jpg'

image1 = cv.imread(image1_path, cv.IMREAD_GRAYSCALE)
image2 = cv.imread(image2_path, cv.IMREAD_GRAYSCALE)

# Create an AKAZE object
akaze = cv.AKAZE_create()

# Detect keypoints and compute descriptors for both images
keypoints1, descriptors1 = akaze.detectAndCompute(image1, None)
keypoints2, descriptors2 = akaze.detectAndCompute(image2, None)

# Create a Brute-Force Matcher
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

# Match descriptors between the two images
```

```
matches = bf.match(descriptors1, descriptors2)

# Sort the matches by their distance (lower distance means better match)
matches = sorted(matches, key=lambda x: x.distance)

# Draw the first 10 matches (you can adjust the number as needed)
matching_result = cv.drawMatches(image1, keypoints1, image2, keypoints2,
matches[:10], None)

# Resize function
def resize_image(image, scale_percent=50):
    width = int(image.shape[1] * scale_percent / 100)
    height = int(image.shape[0] * scale_percent / 100)
    resized_image = cv.resize(image, (width, height))
    return resized_image

resized_matching_result = resize_image (matching_result)

# Display the matching result
cv.imshow('AKAZE Matching Result', resized_matching_result)
cv.waitKey(0)
cv.destroyAllWindows()
```

OUTPUT