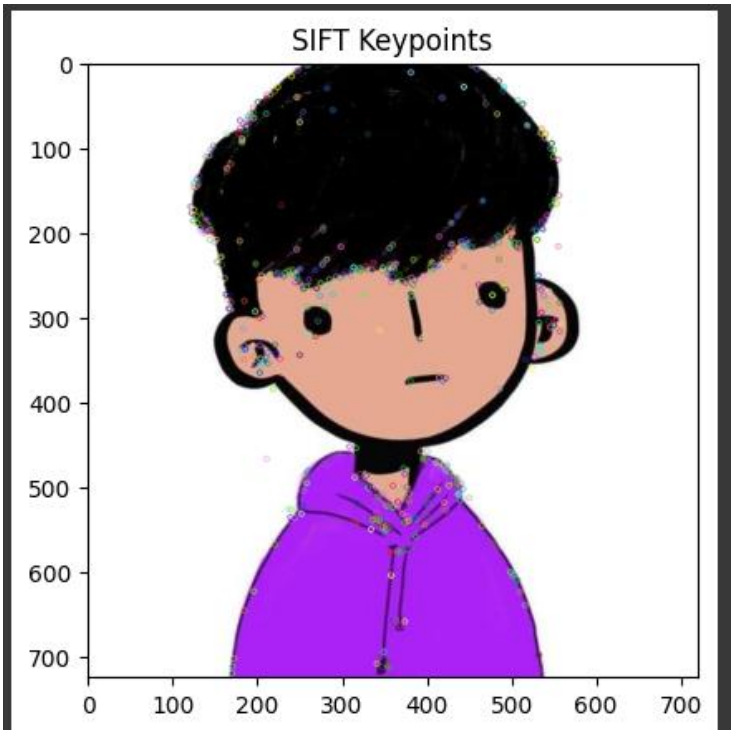
```
Suggested code may be subject to a license | PkParadox88/30days_of_openCV | Aliktk/copy_move_forgery_detection
import cv2
import matplotlib.pyplot as plt

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

#Initialize SIFT detector
sift = cv2.SIFT_create()

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

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

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

This code starts by importing two libraries: cv2 (OpenCV) for image processing and matplotlib.pyplot for displaying images. First, it loads an image named sample.jpg using cv2.imread() and then converts this image to grayscale with cv2.cvtColor(), since the SIFT feature detection algorithm (used in the next step) requires a grayscale input. Next, it initializes a SIFT detector with cv2.SIFT_create(), which is a tool for identifying unique points of interest, known as "keypoints," in an image. These keypoints are often features like edges or corners that can help with recognizing or matching objects. Using sift.detectAndCompute(), it detects these keypoints and calculates their corresponding "descriptors" (feature vectors that describe each keypoint). The code then uses cv2.drawKeypoints() to draw circles or other markers on the original image at each keypoint location, highlighting where SIFT has found features. Finally, it displays the resulting image with marked keypoints using matplotlib.pyplot. The image is converted to RGB format for correct color representation before displaying, and a title is added to the plot window to label it as "SIFT Keypoints."

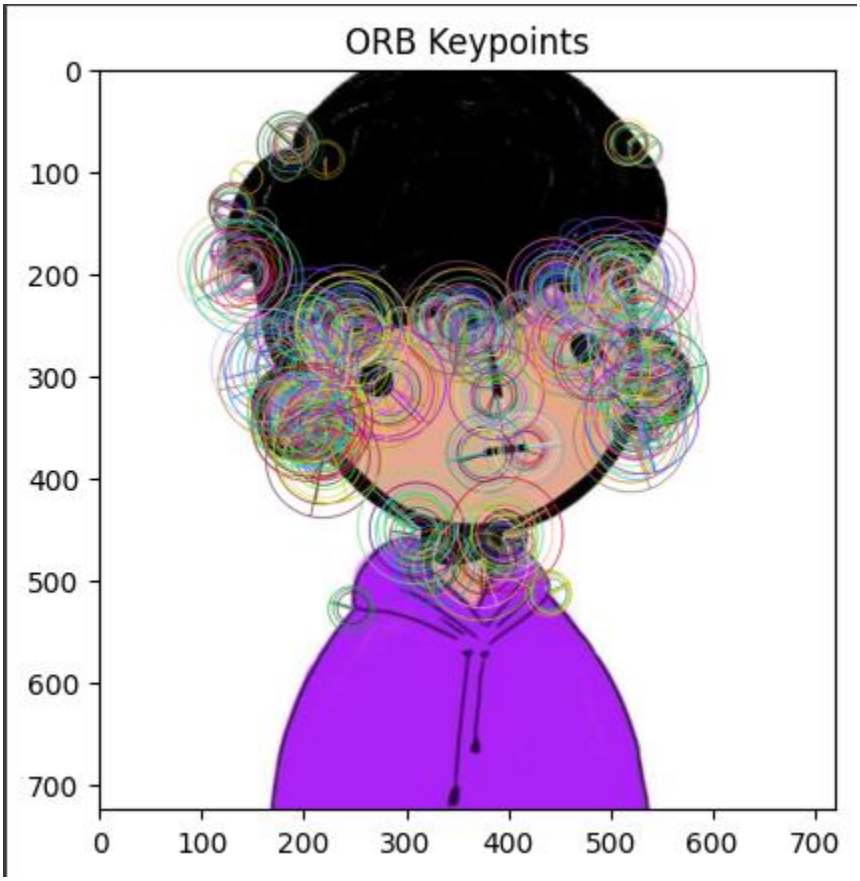
Result:



```
Suggested code may be subject to a license | Aliktk/copy_move_forgery_detection
import cv2
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread("sample.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, flags=cv2.DrawMatchesFlags_DRAW_RICH_KEYPOINTS)

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

This code uses the ORB (Oriented FAST and Rotated BRIEF) method to find and display keypoints in an image. It begins by importing OpenCV for image processing and Matplotlib for showing images. The image named sample.jpg is loaded and converted to grayscale for keypoint detection. An ORB detector is created to identify keypoints and their descriptors in the image. The detected keypoints are drawn on the original image, highlighting the interesting features. Finally, the image with keypoints is converted to RGB format for proper color display and shown using Matplotlib with the title "ORB Keypoints."

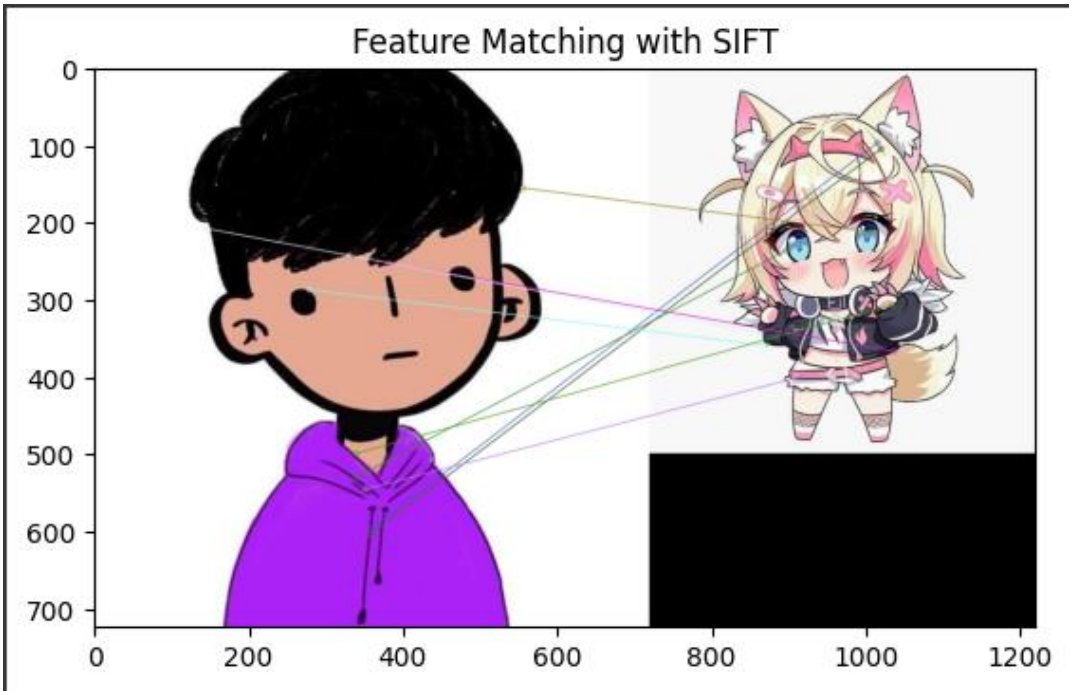
Result:



```
Suggested code may be subject to a license | sitamgithub-MSIT/MindWave | drowning-in-codes/cv_learn
# Load the images
image1 = cv2.imread('sample.jpg')
image2 = cv2.imread('sample2.jpg')

#Initialize SIFT detector
sift = cv2.SIFT_create()

#Detect keypoints and descriptors
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
matches = sorted(matches, key=lambda x: x.distance)

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

#Display the image with matches
plt.imshow(cv2.cvtColor(image_matches, cv2.COLOR_BGR2RGB))
plt.title('Feature Matching with SIFT')
plt.show()
```

This code performs feature matching between two images using the SIFT (Scale-Invariant Feature Transform) algorithm. It begins by loading two images, sample.jpg and sample2.jpg. A SIFT detector is then initialized to find unique features, called keypoints, in both images and to compute their descriptors, which are used to describe these features. Next, a brute-force matcher is created to compare the descriptors from both images. It matches the features and sorts the matches based on their distance, meaning how similar the features are. The code then draws the first 10 best matches between the two images, highlighting where the features correspond. Finally, the matched image is displayed with Matplotlib, showing the result of the feature matching process.

Result:

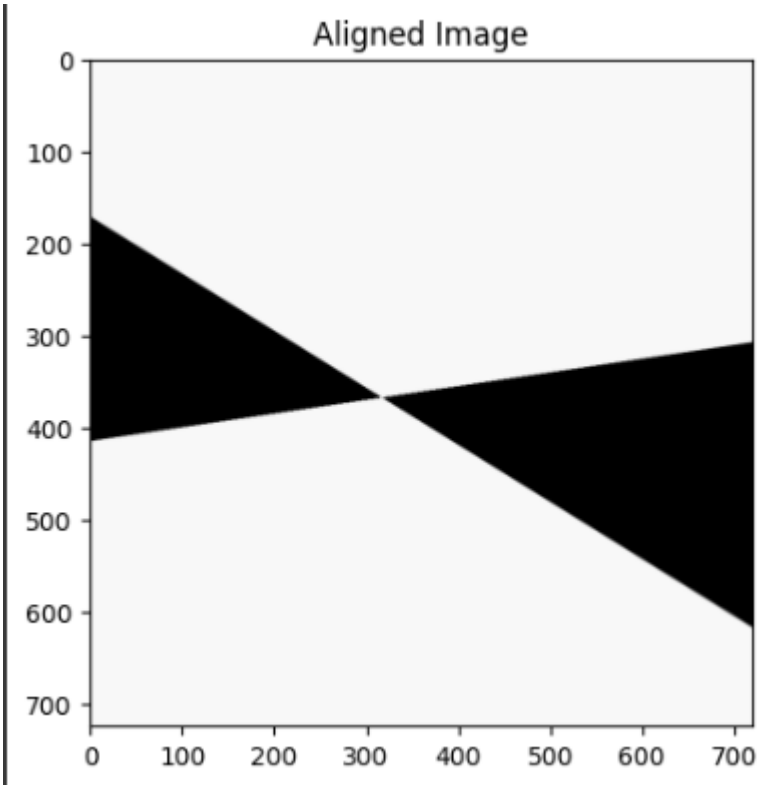


```python
import numpy as np

image1 = cv2.imread('sample.jpg')
image2 = cv2.imread('sample2.jpg')

#Convert to grayscale
gray_image1 = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)
gray_image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2GRAY)

#detect keypoints and descriptors using SIFT
sift = cv2.SIFT_create()
keypoints1, descriptors1 = sift.detectAndCompute(gray_image1, None)
keypoints2, descriptors2 = sift.detectAndCompute(gray_image2, None)

#Match features using BFMatcher
bf = cv2.BFMatcher()
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)

#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)

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

#Warp one image to aligh with the other
h, w = image1.shape[:2]
aligned_image = cv2.warpPerspective(image2, M, (w, h))

#Display the aligned images
plt.imshow(cv2.cvtColor(aligned_image, cv2.COLOR_BGR2RGB))
plt.title('Aligned Image')
plt.show()
```

This code performs image alignment by utilizing the SIFT (Scale-Invariant Feature Transform) algorithm for feature detection and matching. It begins by loading two images and converting them to grayscale for easier processing. Using SIFT, it detects keypoints and computes descriptors for both images, which represent their unique features. The code then employs a brute-force matcher to find matches between the descriptors of the two images, applying a ratio test to filter out poor matches. After extracting the locations of the good matches, it computes a homography matrix, which describes the transformation needed to align the second image with the first. Finally, it warps the

second image using this matrix to achieve alignment and displays the resulting aligned image.

Result:



```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load two images
image1 = cv2.imread('sample.jpg')  # Replace with your first image path
image2 = cv2.imread('sample2.jpg')  # Replace with your second image path

# Check if images are loaded correctly
if image1 is None or image2 is None:
    print("Error: One or both images not found.")
else:
    # SIFT detector
    sift = cv2.SIFT_create()
    keypoints1, descriptors1 = sift.detectAndCompute(image1, None)
    keypoints2, descriptors2 = 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)

    # Match keypoints from SIFT and ORB using a BFMatcher
    bf_sift = cv2.BFMatcher(cv2.NORM_L2, crossCheck=True)
    matches_sift = bf_sift.match(descriptors1, descriptors2)

    bf_orb = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches_orb = bf_orb.match(descriptors1_orb, descriptors2_orb)

    # Sort matches by distance
    matches_sift = sorted(matches_sift, key=lambda x: x.distance)
    matches_orb = sorted(matches_orb, key=lambda x: x.distance)

    # Draw the matches for SIFT
    image_matches_sift = cv2.drawMatches(image1, keypoints1, image2, keypoints2, matches_sift[:10], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

    # Draw the matches for ORB
    image_matches_orb = cv2.drawMatches(image1, keypoints1_orb, image2, keypoints2_orb, matches_orb[:10], None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)

    # Display the results
    plt.figure(figsize=(15, 10))

    plt.subplot(1, 2, 1)
    plt.imshow(cv2.cvtColor(image_matches_sift, cv2.COLOR_BGR2RGB))
    plt.title('SIFT Matches')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(cv2.cvtColor(image_matches_orb, cv2.COLOR_BGR2RGB))
    plt.title('ORB Matches')
    plt.axis('off')

    plt.show()
```

This code extracts features from two images using both the SIFT (Scale-Invariant Feature Transform) and ORB (Oriented FAST and Rotated BRIEF) algorithms, and then matches these features to visualize the correspondence between the two images. First, it imports the necessary libraries: OpenCV for image processing, NumPy for numerical operations, and Matplotlib for displaying images. The two images are loaded, and the code checks if they were loaded correctly. It then initializes the SIFT and ORB detectors, detecting keypoints and computing descriptors for both images with each method. After extracting the features, the code uses a brute-force matcher to find correspondences between the descriptors, applying the appropriate distance metric for each algorithm (L2 for SIFT and Hamming for ORB). The matches are sorted by distance to prioritize the best matches, and the top ten matches from both methods are drawn on the images to visualize the feature correspondences. Finally, the matched images are displayed side by side using Matplotlib, allowing for a direct comparison of the SIFT and ORB feature matching results.

Result: