COMPUTER VISION LAB 4 REPORT - bkoech

We first capture an image and do a principal component analysis on it. PCA is used to extract feature descriptors - the important features on an image. Once you have these features we can use them to compare it with other images.

When we are extracting features from images, there are two main steps:

1. Keypoint detection - We need to identify the main important locations in the image
2. Feature description - We use this to encode information about the key points into a numerical vector

Both of them are collectively known as Feature descriptors. A feature descriptor is the vector associated with each keypoint, containing information about local patterns, textures, and gradients

Examples of Feature descriptors we will use in this lab include:

1. SIFT - Scale Invariant Feature Transform.
2. ORB - Oriented First, Rotated Brief.
3. BRIEF - Binary Robust Independent Elementary Features.

We use these feature descriptors to compare and match images.

Once we have extracted the keypoints and the descriptors, we do a principal component analysis, to reduce their dimensionality before forming a single feature vector. We then use Principal component Analysis to reduce the dimensionality of the feature descriptors extracted by the methods above because they can have high dimensionality which can be very computationally expensive. We also choose the maximum 40 principal components. And finally we flatten these components into a single feature vector for easy comparison with other images.

Since we are comparing and matching images we need some sort of scoring function to tell us how good our 'model' or algorithm is doing. For this particular assignment we are using Euclidean distance and cosine similarity score.

Euclidean distance:

- It measures the distance between two points, since we have feature vectors for both images, it compares them index by index. If the euclidean distance between two images is large, it means that the images are not similar. Euclidean distance is sensitive to a small difference in pixels, which can be caused by lighting, rotation, or even scaling

Cosine similarity score:

- This is the measure of the cosine of the angle between two vectors, it focuses only on the direction of the vectors. If the vectors point in the same direction, the cosine similarity score will be very high, indicating that the features are similar.
- It is, unlike Euclidean distance, less variant to lighting, or scale since it only considers the orientation of the feature vector and not their size.

SCALE INVARIANT FEATURE TRANSFORM

These feature descriptor is:

- Scale invariant - it detects objects irregardless of the size, we use Gaussian Blur to smooth the images at multiple scales
- Rotation invariant - it detects objects irregardless of the rotation applied
- Illumination invariant - resistant to changes in lighting

When I applied SIFT feature detection on my images, I was able to identify **88** key points for the reference image and for the matching image we were able to detect **132** key points.

When I matched the two images with the top 40 principal components, using Euclidean distance and the cosine similarity score of:

- Euclidean Distance: 1.2836928362958133e-05
- Cosine Similarity Score: 0.29015591740608215

If I use a 80 principal components, this is the score I get:

- Euclidean Distance: 1.5197741959127598e-05
- Cosine Similarity Score: 0.2490941882133484

From the scores above, you can see that increasing the number of features even worsens the algorithm because we are increasing the features in each image thereby introducing more noise and diversity, the finer details in each of the images.

ORIENTED FIRST, ROTATED BRIEF

This algorithm uses FAST(Features from Accelerated Segment Test), a corner detection algorithm, used to identify key points in an image, after detecting these key points, we use BRIEF (Binary Robust Independent Elementary Features), to generate binary descriptors for the key points. We then finally use Orientation (Rotation invariance), to the Brief descriptors by calculating the orientation of the key points.

When I applied ORB to my images, I was able to identify **229** key points for the reference image and **385** key points for the matching image

When I matched with 40 principal components, using the Euclidean distance and the cosine similarity score of:

- Euclidean Distance: 9.325885519825322e-14
- Cosine Similarity Score: 0.28140262495784174

When I used 80 principal components, I identified **229** features for the key points and **385** for the matching image.

This is the score I get the same score as the one with 40. This means that most of the features are captured in the first 40 features and even increasing the features doesn't introduce any new features that can help improve the performance.

- Euclidean Distance: 9.325885519825322e-14
- Cosine Similarity Score: 0.28140262495784174

BINARY ROBUST INDEPENDENT ELEMENTARY FEATURES

This algorithm is considered fast in feature matching. It identifies points in the image where there is high contrast or a distinctive feature. It is by design easy to track these features and match them in different images. BRIEF works by selecting pairs of points within a small image patch around a keypoint. For each pair of points, it compares their intensity values (e.g., whether one is brighter than the other) and encodes the result as a binary bit (1 for brighter, 0 for darker). These bits form a binary descriptor, which is then used for matching keypoints.

With Principal features being 40, the key points detected in the reference image is **177,** and for the matching image is **401,** its score was:

- **Euclidean Distance**: 1.2475893076498308e-13
- **Cosine Similarity Score**: 0.07906468456576787

When i increased the features to 80:

- **Euclidean Distance**: 1.2475893076498308e-13
- **Cosine Similarity Score**: 0.07906468456576787

You can see that it doesn't change ideally because most of the features are captured in the first 40 principal components

Based on the comparison of the three algorithms, we conclude that the BRIEF algorithm excels in **Euclidean distance** due to its binary descriptors producing smaller distances for similar keypoints. However, it performs poorly in **cosine similarity** because the lack of directional information in the binary patterns results in small differences causing a significant decline in similarity.

HOMOGRAPHY

This is a mathematical concept that describes the relationships between two images of the same scene, taken from different perspectives. It maps one point from one image to the same point but in a different image using projective transformation.

Homography is a 3 x 3 matrix. It is computed based on points correspondences between two images. The goal is to find the homography matrix that best maps points in one image to their corresponding points in the other image, compensating for changes in perspective, rotation, or scaling.

In this particular lab exercise, we manually defined corresponding points in one image, and in another image, then called the `cv2.findHomography()` while passing both points array and it will output our homography matrix. We could find these points using feature detection algorithms like listed above though.

We then take the first image and apply the homography matrix, to wrap the image to match the new perspective.

```python
warped_img = cv2.warpPerspective(img1, H, (width, height))
```

So to compare the two images, we first need to do object feature detection. We do this using the ORB algorithm. Here we get the key points and feature descriptors as mentioned above.

```python
orb = cv2.ORB_create()
```

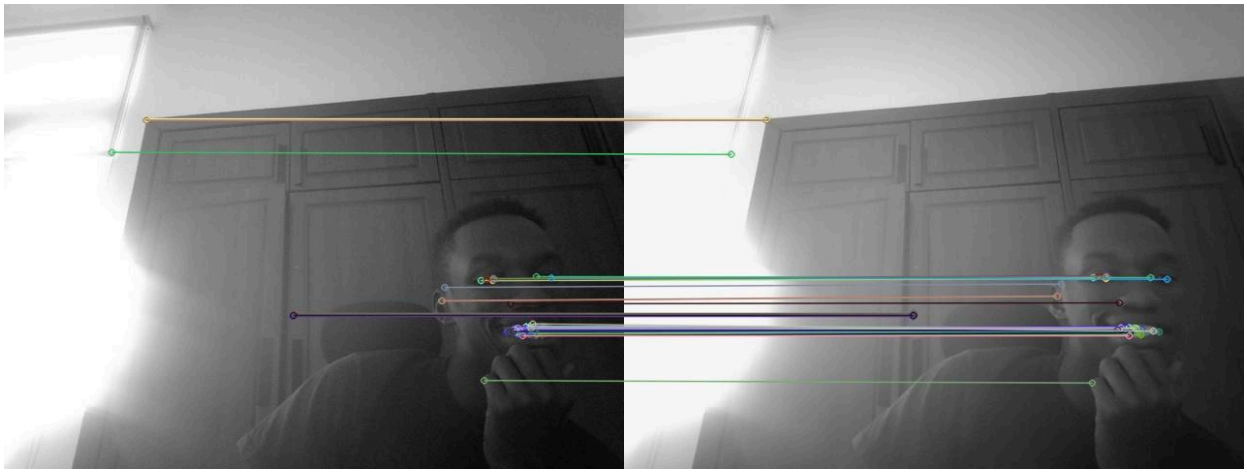
Now that we have the key features for both images, we need to establish the points that are similar in both images, we do this using BFMatcher

```python
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
matches = bf.match(des1, des2)
```

After establishing these matching points, we can draw them just to visualize and see what is happening in our code. We do this using `drawMatches()`

```python
img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches[:50], None,
flags=2)
cv2.imshow("Feature Matches", img_matches)
cv2.waitKey(0)
```

The output of the above code is here:



Have identified points that are similar and matching in the same images, we can the proceed to find the homography:

Python

```python
# Extract point coordinates
src_pts = np.float32([kp1[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
dst_pts = np.float32([kp2[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

# Compute Homography
H, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)

print("Homography Matrix:\n", H)
```