CS 484, Spring 2022
Homework Assignment 2: Local Features

Melike Demirci – 21702346

For this homework I have used Python with following libraries

- Numpy(1.22.2)
- opencv-python(4.5.5.62)
- opencv-contrib-python(4.5.5.64)

Loading Data

As the first step, source images have been read as a grayscale images from the data folder. File names are acquired from the txt file whose name is asked from the user. After loading the images, direction of the stitching is also asked from the user.

After saving images in an array, it is passed to stitch function which is as following:

```
def stitch(images, direction):
    if direction == 1:
        images = images[::-1]

    result = images[0]
    for i in range(1, len(images)):
        result = stitchTwo(result, images[i])
        rows, cols = np.where(result[:, :] != 0)
        max_col = max(cols)-5
        result = result[:, :max_col]
    return result
```

Figure 1: Stitch Function

In this function, images are read one by one from the beginning or the end according to the user input for the direction. Two images are given as an input to stitchTwo function and output is the stitched image. In a loop, all the images are stitched together.

Detecting and Describing Local Features

In order to detect local features in images, I have used SIFT Class, which is used for extracting key points and computing descriptors using the Scale Invariant Feature Transform(SIFT) algorithm by D. Lowe [1].

As it can be seen from the below figure, detectAndCompute function of the SIFT library is used. This function takes the image and return the corresponding key points and descriptions. After acquiring the descriptions of the key points, they are matched according to some criterion.

```
def stitchTwo(img1, img2):

    # Find key points and SIFT descriptors
    sift = cv2.xfeatures2d.SIFT_create()
    kp1, d1 = sift.detectAndCompute(img1, None)
    kp2, d2 = sift.detectAndCompute(img2, None)

    # Match keypoints
    good_points, good_matches = match(d1, d2)
    mat = cv2.drawMatchesKnn(img1, kp1, img2, kp2, good_matches, None, flags=2)
    global drawCount
    cv2.imwrite('matches/matching' + str(drawCount) + '.jpg', mat)
    drawCount += 1
    print(len(good_points))
    if len(good_points) > 5:
        image1_kp = np.float32([kp1[i].pt for (_, i) in good_points])
        image2_kp = np.float32([kp2[i].pt for (i, _) in good_points])
        H, status = cv2.findHomography(image2_kp, image1_kp, cv2.RANSAC)

    warped = cv2.warpPerspective(img2, H, (img1.shape[1] + img2.shape[1], img1.shape[0]))
    result = blend(warped, img1, img2)

    return result
```

Figure 2: stitchTwo Function

Feature Matching

For the feature matching, BFMatcher library of CV2 is used. For each descriptor in the first set, this matcher finds the closest descriptor in the second set by trying each one [2]. In order to implement Euclidian distance, cv2.NORM_L1 is given as a parameter to the function. By using knnMatch function, k best matches can be acquired. K has been set to 2 so that ratio test, which is explained by D. Lowe in his paper, could be applied [3]. I have tried different ratios; when it is higher than 0.8, number of good points increases and it may have bad effect to stitching. But when it is too low, number of good points decreases a lot, which is also ineffective. 0.70 is chosen as a final ratio.

```python
def match(d1, d2):
    matcher = cv2.BFMatcher(cv2.NORM_L1)
    raw_matches = matcher.knnMatch(d1, d2, k=2)
    good_points = []
    good_matches= []
    for m1, m2 in raw_matches:
        if m1.distance < 0.70 * m2.distance:
            good_points.append((m1.trainIdx, m1.queryIdx))
            good_matches.append([m1])

    return good_points, good_matches
```

Figure 3: match Function

After finding matched key points, they are drawn on the images and saved in the matched folder. One of these examples can be seen in Figure 4.

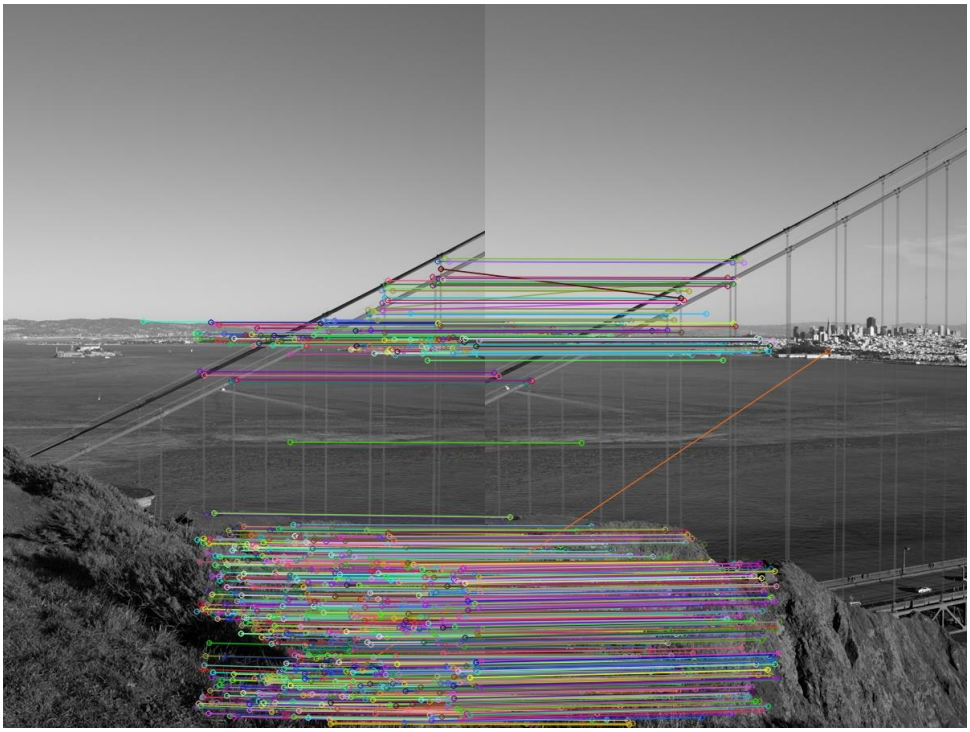


Figure 4: Example of the Matched Points of Two Images

In order to print matches, drawMatchesKnn function of the cv2 has been used [4]. As it can be seen from the Figure 2, additionally to the distance criteria of the descriptions; number of descriptions for every matched images were also considered. At least 5 matches should exist in pair of images.

Image Registration

After finding the corresponding points in the reference image and the target image, the geometric relationship between two input images has been discovered by using RANSAC algorithm which is implemented in findHomography function of cv2 [5]. Output of this function is used as a transformation matrix for the perspective warping.

Blending

Blending performed as following

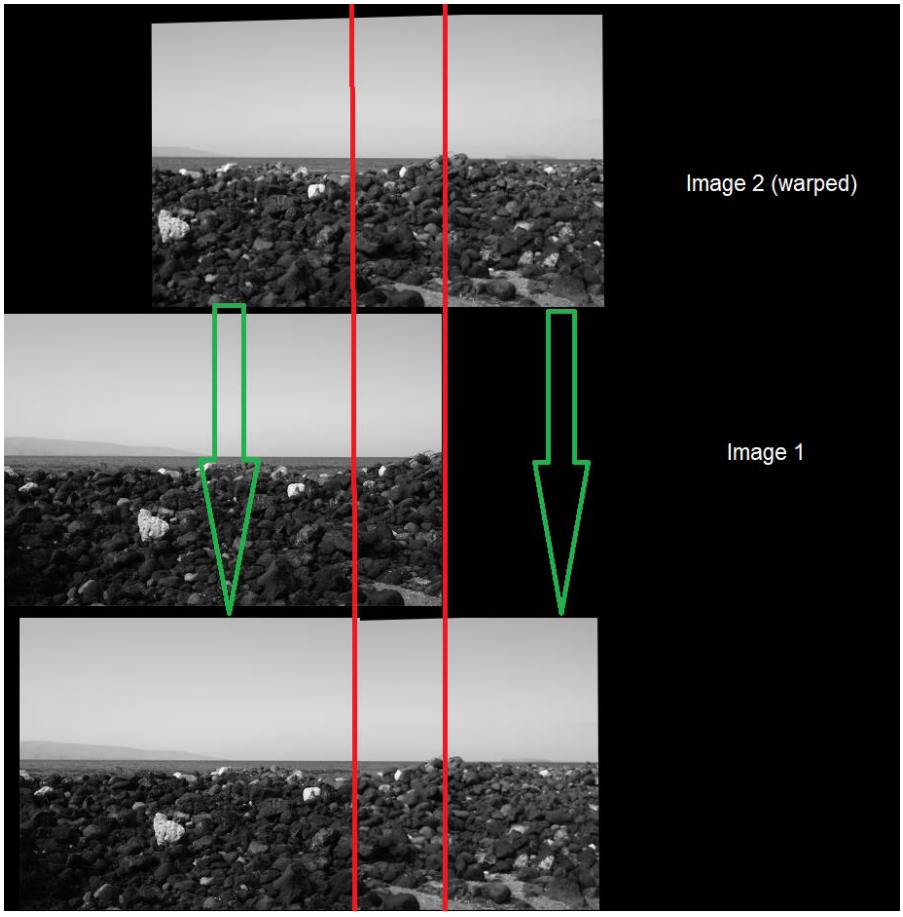


Figure 5: Blending

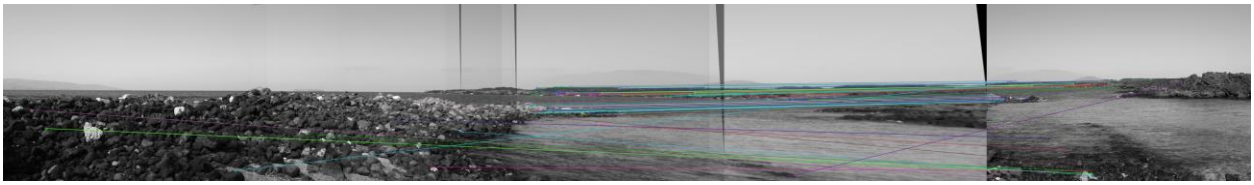
Some window width was set, which is showed on the Figure 5 with the red lines. Pixels inside these red lines is averaged from the two images. Left side of the image is directly taken from image 1 and the left side is taken from the second image which is warped.

Final Results

Figure 6: Result of Fishbowl Images

Figure 7: Result of Golden Gate Images

As it can be seen from the final results, the stitching implementation worked well on the golden gate images. However, for the fishbowl images, some errors occurred through the end of stitching. There might be some implementation errors causing this wrong warping such as under filtering of the matched key points. Fishbowl images were harder to stitch due to the similar feature occurrence in different parts of the images. This can be seen more clear in the following figure;



For example, the long green line is one of the erroneous matches. It may cause due to the similar appearance of the rocks.

REFERENCES

[1] https://docs.opencv.org/3.4/d7/d60/classcv_1_1SIFT.html

[2] https://docs.opencv.org/4.x/d3/da1/classcv_1_1BFMatcher.html

[3] https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[4] https://docs.opencv.org/4.x/d4/d5d/group__features2d__draw.html

[5] https://docs.opencv.org/3.4/d1/de0/tutorial_py_feature_homography.html