AIN433 – Computer Vision Lab.

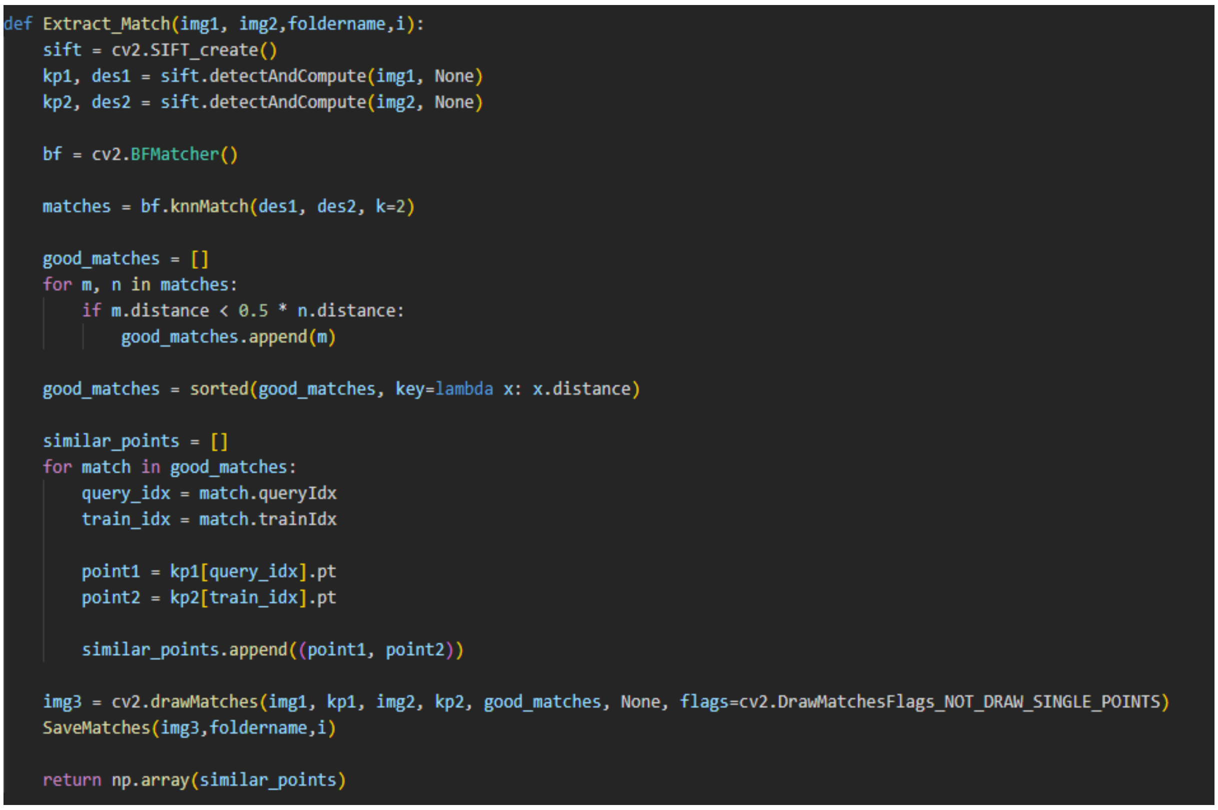
Programming Assignment 2

Burak Kurt

2200765010

PART 1: Feature Extraction:

In this part of assignment, SIFT algorithm is used to extract features from given images. SIFT algorithm needs grayscale images to extract features of it so images were read in grayscale format in the beginning. Then sift object was created using OpenCV function "SIFT_create". Next, sift object's "detectAndCompute" function is called to obtain features of images. Fig1 contains both feature extraction and matching function.



```python
def Extract_Match(img1, img2,foldername,i):
    sift = cv2.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)

    bf = cv2.BFMatcher()

    matches = bf.knnMatch(des1, des2, k=2)

    good_matches = []
    for m, n in matches:
        if m.distance < 0.5 * n.distance:
            good_matches.append(m)

    good_matches = sorted(good_matches, key=lambda x: x.distance)

    similar_points = []
    for match in good_matches:
        query_idx = match.queryIdx
        train_idx = match.trainIdx

        point1 = kp1[query_idx].pt
        point2 = kp2[train_idx].pt

        similar_points.append((point1, point2))

    img3 = cv2.drawMatches(img1, kp1, img2, kp2, good_matches, None, flags=cv2.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
    SaveMatches(img3,foldername,i)

    return np.array(similar_points)
```

Figure 1: Feature Extraction and Matching Implementation

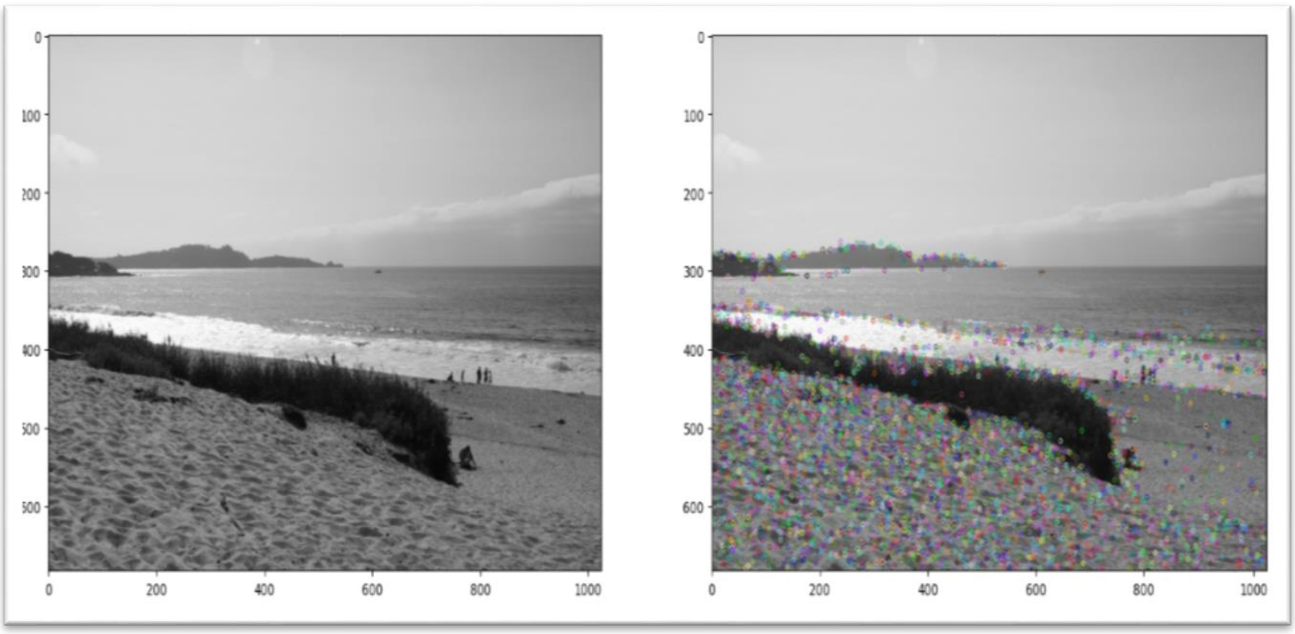


Figure 2: Featur

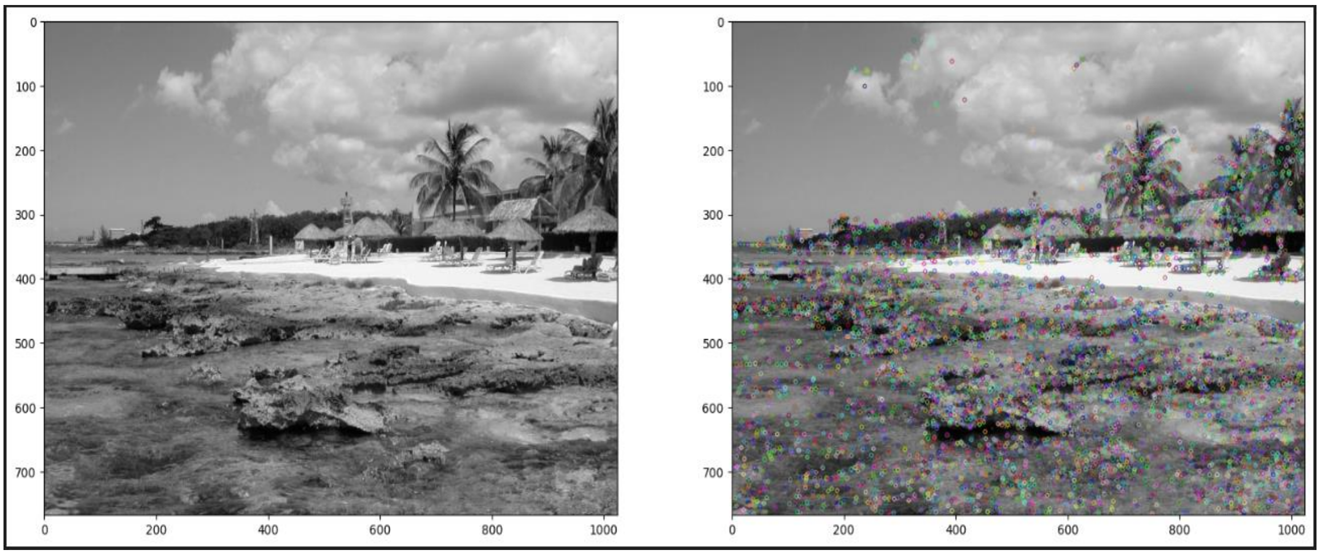


Figure 3: Feature Extraction Example

PART 2: Feature Matching:

After feature extraction of two input images where images are sequential, a BFMatcher object was created and used to find the two best matches for each feature from first image in the second image. This was done using the knnMatch method which performs k-nearest neighbor matching. Then, function filters these matches to retain only the “good” matches. A match is considered good if the distance of the first match is less than 75% of the second match. Next, good matches were sorted in ascending order. Finally, for each good match, the function retrieves the corresponding points in the two images. These points are stored in the “similar_points” list. Corresponding points were connected with line after drawMatches function of OpenCV was called. That merged images with lines were stored in matches folder.

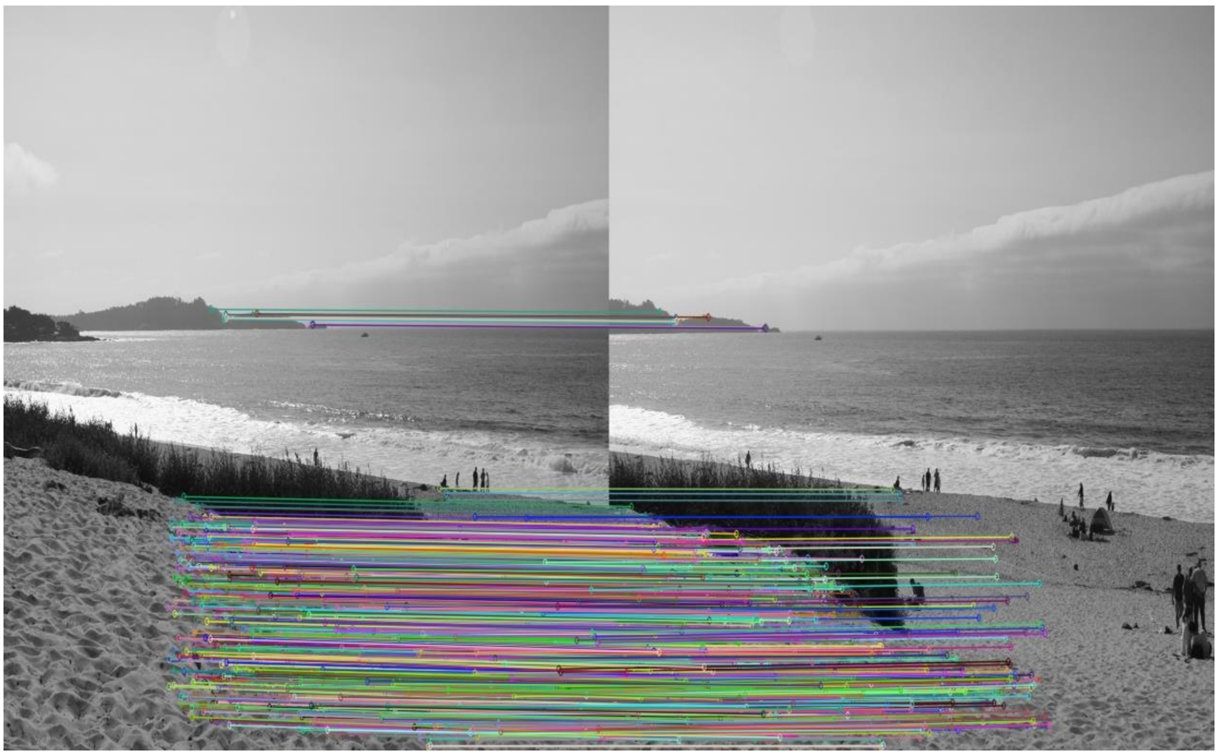


Figure 4: Feature Matching Example

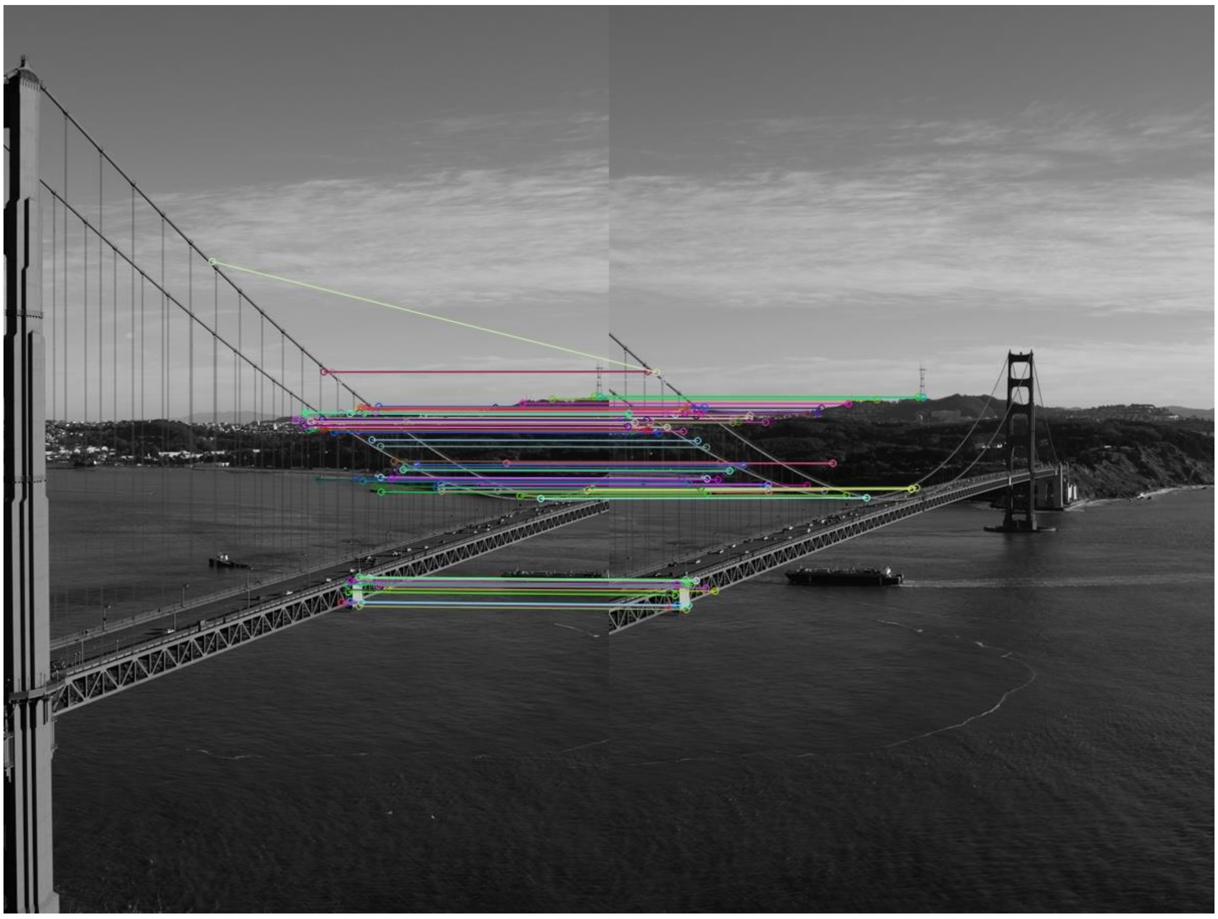


Figure 5: Feature Matching Example

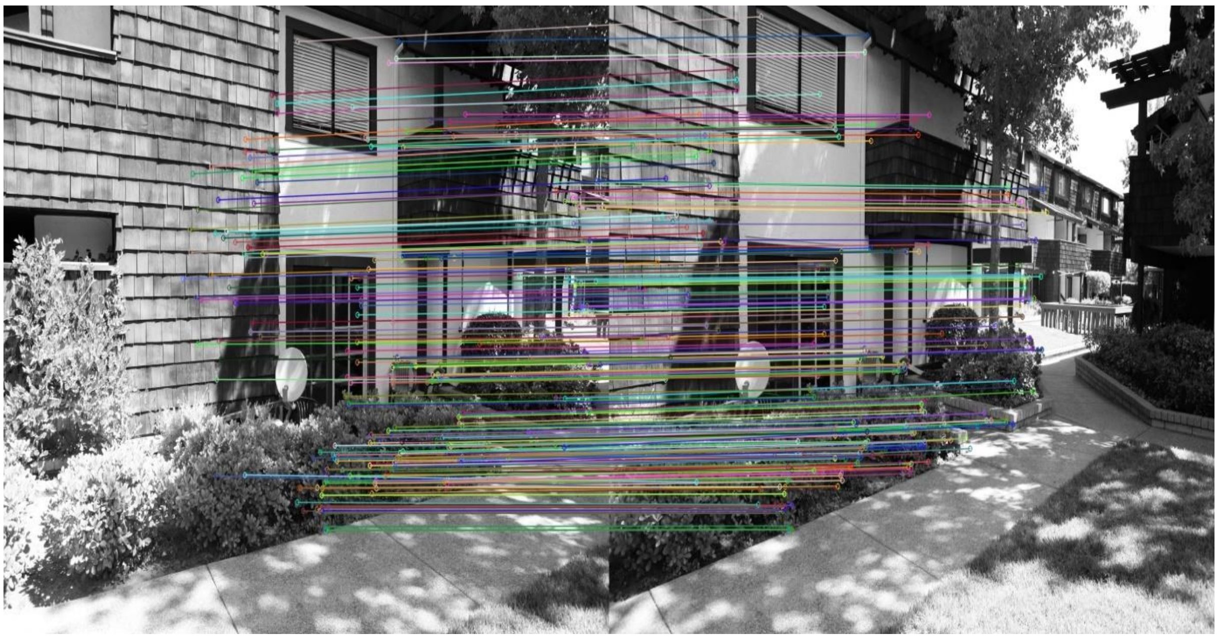
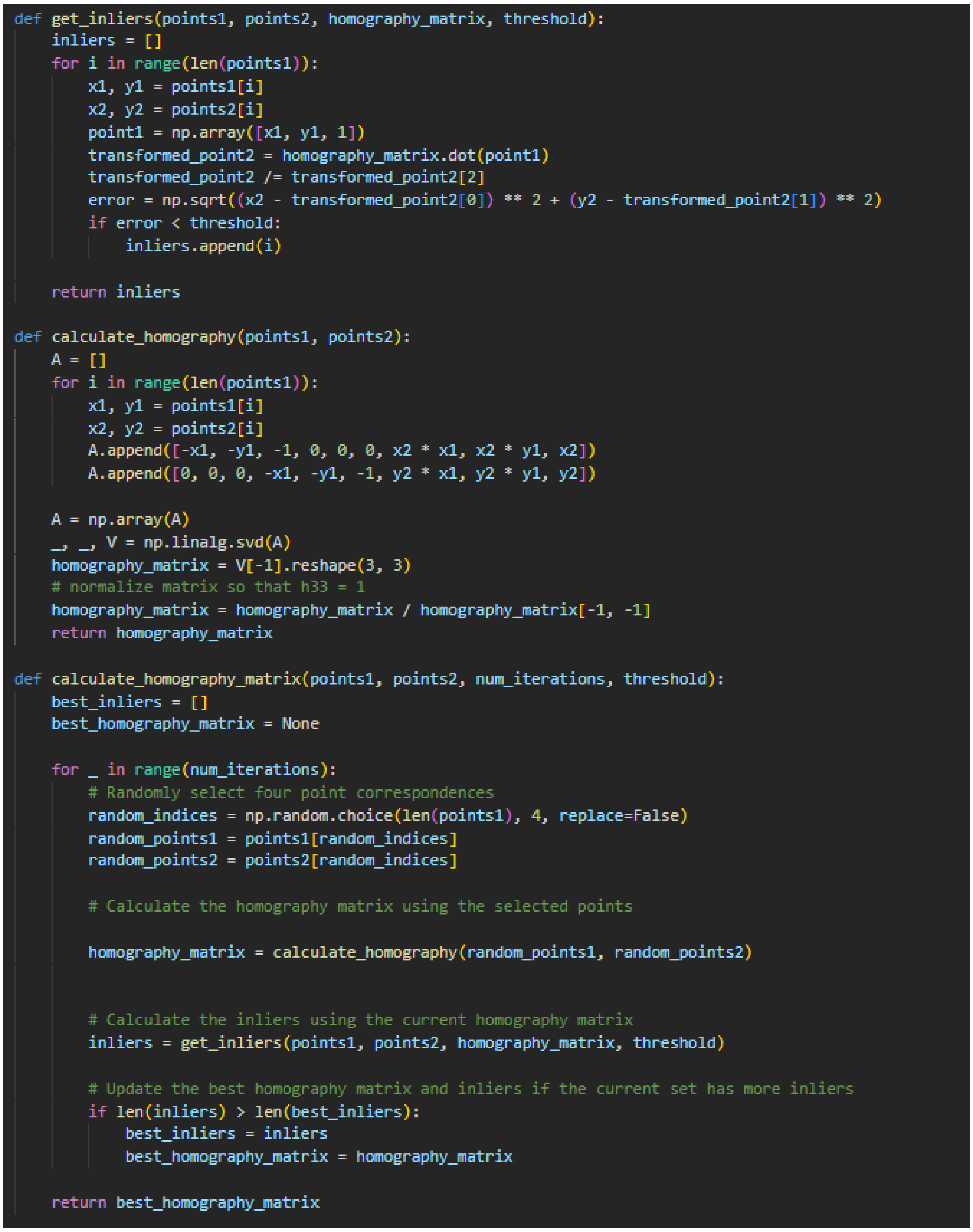


Figure 6: Feature Matching Example

PART 3: Calculating Homography Matrix:

A homography matrix is a 3x3 matrix that describes the transformation between two sets of corresponding points in different images. The RANSAC method is a robust algorithm that can estimate the homography matrix from a set of noisy and outlier-prone matches.



```python
def get_inliers(points1, points2, homography_matrix, threshold):
    inliers = []
    for i in range(len(points1)):
        x1, y1 = points1[i]
        x2, y2 = points2[i]
        point1 = np.array([x1, y1, 1])
        transformed_point2 = homography_matrix.dot(point1)
        transformed_point2 /= transformed_point2[2]
        error = np.sqrt((x2 - transformed_point2[0]) ** 2 + (y2 - transformed_point2[1]) ** 2)
        if error < threshold:
            inliers.append(i)

    return inliers

def calculate_homography(points1, points2):
    A = []
    for i in range(len(points1)):
        x1, y1 = points1[i]
        x2, y2 = points2[i]
        A.append([-x1, -y1, -1, 0, 0, 0, x2 * x1, x2 * y1, x2])
        A.append([0, 0, 0, -x1, -y1, -1, y2 * x1, y2 * y1, y2])

    A = np.array(A)
    _, _, V = np.linalg.svd(A)
    homography_matrix = V[-1].reshape(3, 3)
    # normalize matrix so that h33 = 1
    homography_matrix = homography_matrix / homography_matrix[-1, -1]
    return homography_matrix

def calculate_homography_matrix(points1, points2, num_iterations, threshold):
    best_inliers = []
    best_homography_matrix = None

    for _ in range(num_iterations):
        # Randomly select four point correspondences
        random_indices = np.random.choice(len(points1), 4, replace=False)
        random_points1 = points1[random_indices]
        random_points2 = points2[random_indices]

        # Calculate the homography matrix using the selected points

        homography_matrix = calculate_homography(random_points1, random_points2)


        # Calculate the inliers using the current homography matrix
        inliers = get_inliers(points1, points2, homography_matrix, threshold)

        # Update the best homography matrix and inliers if the current set has more inliers
        if len(inliers) > len(best_inliers):
            best_inliers = inliers
            best_homography_matrix = homography_matrix

    return best_homography_matrix
```

Figure 7: Implementation of Homography Matrix

The "get_inliers" function calculates the inliers of a given homography matrix. It takes as input two sets of points, a homography matrix, and a threshold for the error. It calculates the transformed point for each point in points1 using the homography matrix, and computes the Euclidean distance between the transformed point and the corresponding point in points2. If this distance is less than the threshold, the point is considered an inlier and its index is added to the list of inliers.

The “calculate_homography” function calculates a homography matrix from two sets of points. It constructs a matrix A from the point correspondences, and then computes the singular value decomposition (SVD) of A. The homography matrix is the last column of V, reshaped as a 3x3 matrix. The homography matrix is then normalized so that its bottom-right element is 1.

The “calculate_homography_matrix” function calculates the best homography matrix from two sets of points using the RANSAC algorithm. It takes as input the two sets of points, the number of iterations for the RANSAC algorithm, and a threshold for the error. In each iteration, it randomly selects four point correspondences, calculates the homography matrix using these points, and then calculates the inliers of this homography matrix. If the current homography matrix has more inliers than the best homography matrix found so far, it becomes the new best homography matrix. The function returns the best homography matrix found.