Computer Vision HW#2

311605006 劉劭暐

Additional package:
pip install tqdm
pip install ipywidgets

1. SIFT

```python
def SIFT_detection(img):
    SIFT_Detector = cv2.SIFT_create()
    kp, des = SIFT_Detector.detectAndCompute(img, None)

    return kp, des
```

Call sift function of cv, obtain keypoints and descriptors

2. KNN feature matching

```python
def featureMatching(kp0, kp1, des0, des1, threshold = 0.75):
    matches_1to2 = []
    matches_2to1 = []
    best_match = []

    for i in range(len(des0)):
        distances = [euclidean_distance(des0[i], des1[j]) for j in range(len(des1))]
        sorted_indices = np.argsort(distances)
        first_nearest = sorted_indices[0]
        second_nearest = sorted_indices[1]

        if distances[first_nearest] < threshold * distances[second_nearest]:
            matches_1to2.append(i)
            matches_2to1.append(first_nearest)
            best_match.append(list(kp0[i].pt + kp1[first_nearest].pt))


    best_match = np.asarray(best_match)

    return best_match
```

First I calculate the distance of descriptors between img1 and img2, then I sort them in ascending order. By this method, the first two index is the nearest img2 descriptors' index of descriptor i.

After that we apply Lowe's ratio test for eliminating bad match.

3. Homography

```
def Homography(matches):
    rows = []
    for i in range(matches.shape[0]):
        p1 = np.append(matches[i][0:2], 1)
        p2 = np.append(matches[i][2:4], 1)
        row1 = [0, 0, 0, p1[0], p1[1], p1[2], -p2[1]*p1[0], -p2[1]*p1[1], -p2[1]*p1[2]]
        row2 = [p1[0], p1[1], p1[2], 0, 0, 0, -p2[0]*p1[0], -p2[0]*p1[1], -p2[0]*p1[2]]
        rows.append(row1)
        rows.append(row2)
    rows = np.array(rows)
    U, s, V = np.linalg.svd(rows)
    H = V[-1].reshape(3, 3)
    H = H/H[2, 2]

    return H
```

Construct A matrix and use SVD to find H.
Here we reshape the last row of V to H.

4. RANSAC

```python
def RANSAC(matches, threshold = 0.3, iters = 3000):
    most_inliers = 0

    for i in range(iters):
        # STEP1, randomly select 4 data points
        points = Randompoints(matches)
        # STEP2, find Homography matrix
        H = Homography(points)

        # Implement rank check
        if np.linalg.matrix_rank(H) < 3:
            continue

        # Calculate error between p2, p2'
        matches_error = matches_errorCal(matches, H)

        # Get H that fit the most
        index = np.where(matches_error < threshold)[0]
        S = matches[index]
        num_inliers = len(S)
        if num_inliers > most_inliers:
            largest_S = S
            most_inliers = num_inliers
            best_H = H

    return largest_S, best_H
```

First we randomly select 4 pairs of matched points and find Homography of each pairs of matched points. Note that Homography should be full rank.
Second we calculate the error between estimated points and real points, if the error is small enough, add the match pairs to S.
If S has most inliers, best homography will be set to current H.
Run the loop a sufficient number of times, then return best homography.
5, Stitch image

```python
translation_mat = np.array([[1, 0, -x_min], [0, 1, -y_min], [0, 0, 1]])
H = np.dot(translation_mat, H)
```

```
warped_l = cv2.warpPerspective(src=left, M=H, dsize=size)
```

```
warped_r = cv2.warpPerspective(src=right, M=translation_mat, dsize=size)
```

Warp left and right image

```
for i in tqdm(range(warped_r.shape[0])):
    for j in range(warped_r.shape[1]):
        pixel_l = warped_l[i, j, :]
        pixel_r = warped_r[i, j, :]

        if not np.array_equal(pixel_l, black) and np.array_equal(pixel_r, black):
            warped_l[i, j, :] = pixel_l
        elif np.array_equal(pixel_l, black) and not np.array_equal(pixel_r, black):
            warped_l[i, j, :] = pixel_r
        elif not np.array_equal(pixel_l, black) and not np.array_equal(pixel_r, black):
            opt = [pixel_l, pixel_r]
            warped_l[i, j, :] = (pixel_l + pixel_r)/2#opt[np.argmax([np.linalg.norm(pixel_
        else:
            pass

stitch_image = warped_l[:warped_r.shape[0], :warped_r.shape[1], :]
```

Stitch image and store results in warped_l

Results: