Computer Vision HW3 Group 16

— 0856622 余家宏、309551067 吳子涵、309551122 曹芳驊

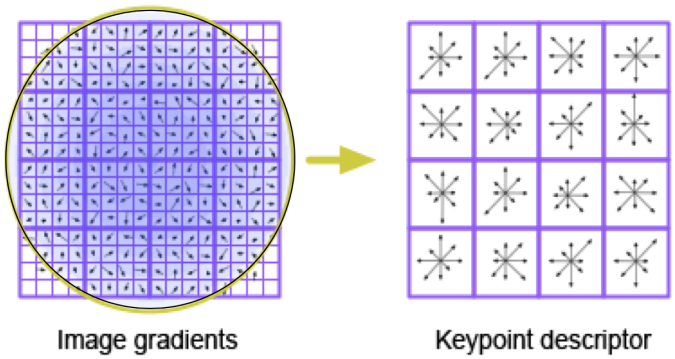
Introduction

- Image stitching is the process of combining multiple images with overlapping field to produce a panorama image.
- Before blending the images, we need to align them first. Using the keypoints and descriptors calculated by SIFT, we would know the correspondences between the two images. Finally, we use RANSAC algorithm to estimate a homography matrix and apply warping transformation to stitch the two images.

Implementation Procedure

Step 1: Interest point detection and feature description

First, we use OpenCV built-in SIFT function to detect interest points in both images. SIFT can generate a 128 dimentions description for each interest point.
(16 cells * 8 directions = 128 dimentional description)




```
# Step1: get sift keypoints and descriptions
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(img1_gray, None)
kp2, des2 = sift.detectAndCompute(img2_gray, None)
```

Step 2: Feature matching by SIFT feature

There are many ways to define the feature distance between two features. For example, L2 distance $||f_1 - f_2||$ is the simplest way. However, because L2 distance may give good scores for ambiguous incorrect matches, we apply ratio distance $\frac{||f_1 - f_2||}{||f_1 - f_2'||}$ in our implementation instead.

f_1 is a feature of image 1, f_2 is the best SSD match to f_1 of image 2, and f_2' is the second best SSD match to f_1 of image 2. If the ratio distance is smaller than a preset threshold, f_2 will be considered as the match feature point of f_1.

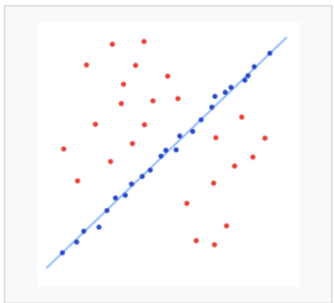
```
def find_match(des1, des2):
    ret = []
    dist_thresh = 0.6
    for i, des_1 in enumerate(des1):
        best = [1e6, 1e6]
        best_idx = [-1, -1]
        for j, des_2 in enumerate(des2):
            dist = ssd(des_1, des_2)
            if dist < best[0]:
                best[0], best[1] = dist, best[0]
                best_idx[0], best_idx[1] = j, best_idx[0]
            elif dist < best[1]:
                best[1] = dist
                best_idx[1] = j
        if L2_dist(des_1, des2[best_idx[0]])/L2_dist(des_1, des2[best_idx[1]])\
           < dist_thresh:
            ret.append([i, best_idx[0]])

    return ret
```

Step 3: RANSAC to find homography matrix H

Random sample consensus (RANSAC) is an iterative method to estimate parameters of a mathematical model from a set of observed data that contains outliers, when outliers are to be accorded no influence on the values of the estimates. RACSAC can be split into 3 steps.

1. Random sample the number of points.
2. Solve for model parameters using those samples.
3. Score by the ratio of inliers with a preset threshold.

A data set with many outliers for which a line has to be fitted.



Fitted line with RANSAC; outliers have no influence on the result.

RANSAC is an iterative method that we need to repeat step 1 to step 3 to find the best model with high confidence.
To determine the number of iterations, we use the formula below.

$$N = \frac{log(1-p)}{log(1-(1-e)^s)}$$

N : The number of iterations.
p : The probability that there's only inliers in one random sample.
e : Outlier ratio.
S : The number of sample points.

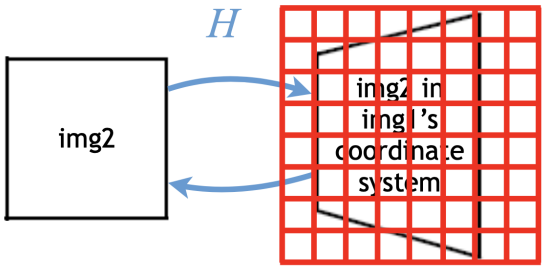
```python
def ransac(kp1, kp2, matches):
    n_sample = 8
    outlier_ratio = 0.05
    n_iter = int(np.log(1 - 0.99) / np.log(1 - (1-outlier_ratio)**n_sample))
    best_homo = None
    best = 1e6
    kp1 = np.asarray(kp1)
    kp2 = np.asarray(kp2)
    kp1 = kp1[matches[:, 0]]
    kp2 = kp2[matches[:, 1]]
    for _ in range(n_iter):
        # random sample matches feature points
        samples = random.sample(range(len(kp1)), n_sample)
        samples = np.asarray(samples)
        # calculate homography
        tmp_homo = homography(kp1[samples], kp2[samples])
        # calculate in/out liers
        outl_cnt = count_out_liers(tmp_homo, kp1, kp2)
        print(outl_cnt)
        if outl_cnt < best:
            best = outl_cnt
            best_homo = tmp_homo

    return best_homo
```

Using RANSAC, we will obtain the best homography matrix which has the least outliers after transfering the feature points.

Step 4: Warp image to create a panoramic image

After we retrieve the transformation matrix between two images by RANSAC, we can perspectively transform one of the images to the other's coordinate system.

In our implementation, we try warping the right-side image of the panorama to the coordinate system of the left-side image.



Create transformation result

We simply do the forward transfromation from the coordinate in image-left, and then do the nearest-neighbor interpolation to find the correspondent pixel in image-right. Because of the later stitch, we set the the output size as H x Wl+Wr.

Iterate through every pixel in the output image to obtain the result.

```
def WarPerspective(src, H, outshape):
    assert type(src) is np.ndarray, 'src is not np.ndarray'
    dst = np.zeros((outshape[0], outshape[1], src.shape[2]), dtype = np.uint8)
    for i in range(outshape[1]):
        for j in range(outshape[0]):
            p_trans = np.dot(H, [i,j,1])
            p_trans /= p_trans[2]
            x,y = int(p_trans[0]),int(p_trans[1])
            if x >= 0 and y >= 0 and x < src.shape[1] and y < src.shape[0]:
                dst[j,i] = src[y,x]
    cv2.imwrite('output/perspective.png', dst)
    return dst
```

Stitch with linear blending

In order to get the transformation result, we need to stitch with the left-side image.

We first decide the border of overlapping region of two images, and then use linear blending (below formula) for the overlap part to calculate the pixel value.

$$g(x) = (1 - \alpha)f_0(x) + \alpha f_1(x)$$

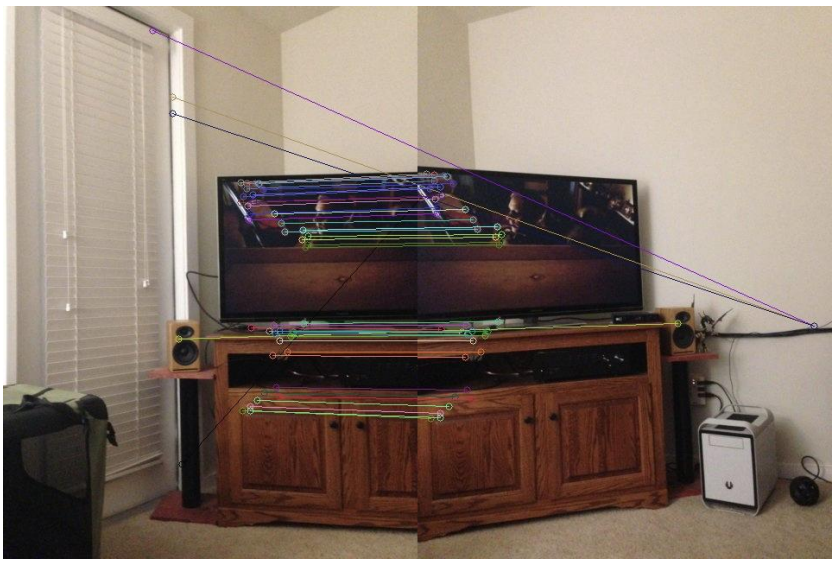
α = distance to left border / overlap region width.

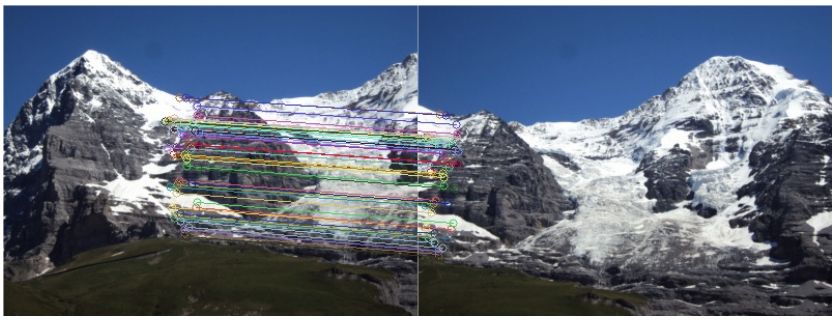
```
def stitch(img_left, img_right, H):
    result = WarPerspective(img_right, H = H, outshape = (img_right.shape[0],
                            img_right.shape[1]+img_left.shape[1]))
    border = img_left.shape[1]
    for i in range(img_left.shape[0]):
        for j in range(img_left.shape[1]):
            if any( val != 0 for val in result[i,j]):
                border = min(border, j)
    total_overlap = float(img_left.shape[1] - border)
    for i in range(img_left.shape[0]):
        for j in range(img_left.shape[1]):
            if any( val != 0 for val in result[i,j]):
                alpha = float(img_left.shape[1]-j)/total_overlap
                result[i,j] = (result[i,j]*(1-alpha) + img_left[i,j]*alpha).astype(int)
            else:
                result[i,j] = img_left[i,j]

    return result
```

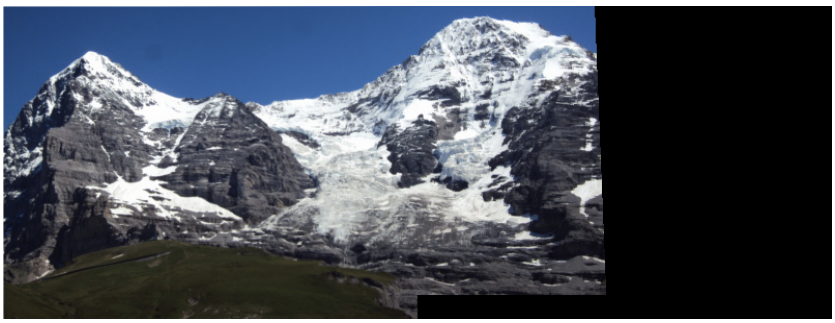
Experimental Results

For RANSAC, we let $p = 0.99$, $e = 0.05$, $S = 8$ in the formula mentioned above.

Sample Images



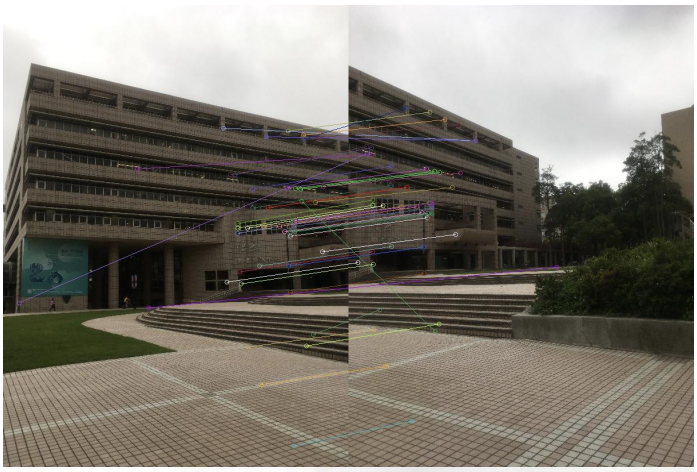





예 원
빌 리 지


예 원
빌 리 지

Our Images



Discussion

- While we were doing experiment with our own images, we've noticed that it took a very long time to process. After analyzing the reason, we found that the images we took were in a very high resolution, which cause SIFT generating too many feature points. Because the feature point matching algorithm takes $O(N^2)$ in time complexity, the whole process will stuck. To solve this problem, we forced the process to go on after finding enough matches, for example, 60 matches were enough for image stitching.
- For our own images, using linear blending will cause some blur shadow in the overlap region because of the high resolution. We can use pyramid blending or just find the midline of the region, then only do blending withing a smaller region on the both sides to tackle this problem.

Conclusion

In this project, we implement an algorithm of image stitching. First, we use SIFT to obtain feature points and desciptions, then we use ratio distance to match feature points in two different images and calculate the homography matrix by RANSAC. In the end, we stitch two images together to generate a panoramic image.

Work assignment plan between team members

- Coding: 余家宏、曹芳驊
- Experiment: 余家宏、吳子涵、曹芳驊
- Report: 余家宏、吳子涵、曹芳驊