CS172 Computer Vision I:
Project1: local feature extraction

Ziyun Zeng
Student ID 2019533086
zengzy1@shanghaitech.edu.cn

Abstract

This report is about local feature extraction project. Author gives its own way to choose matched points in two pictures and calculate the homography matrix. Furthermore, some library functions are used to help stitch two pictures better.

1. Introduction

Panoramic pictures can help people have a more intuitive understanding of the scene of a place. As a university student, author makes some experiments on picture stitch starting from the surrounding landscape.

More details about the experiments will be shown in this report.

2. Algorithm Implementation

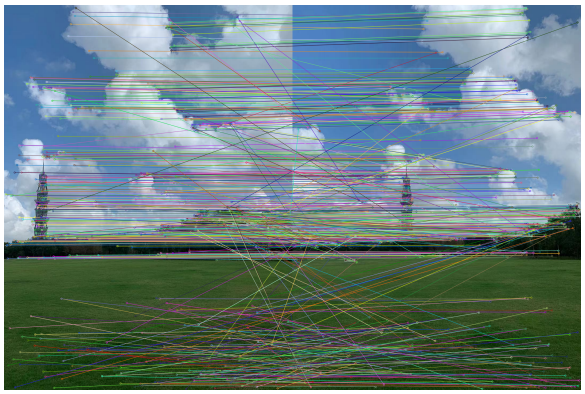
There are five main processes in the algorithm to finish the local feature extraction project.

2.1. Sift

Use SIFT to find out all the key points and feature vectors of every pictures. This part use several library functions, like $cv2.xfeatures2d.SIFT\_create()$, and detectandcompute in stitch.py realizes such functions.

2.2. Match

Find pairs of similar points in two pictures. First use rho in stitch.py to calculate the similarity of every pairs of key points, and I keep those pairs who have a similarity which is larger 97 percent. Other choosing methods are also added into matches in stitch.py to improve the correctness of point pairs. Method 1 is to remove all points which are similar to more than two points in another picture simultaneously. Method 2 is to make sure that the most similar point to point A should be point A's most similar point.



This pair of picture does not apply Method 1 and Method 2 mentioned above. It is clearly to see that many key points on the grass area have a big similarity but they are not the corresponding points. The reason why they match together is that such point are too general, so they may look like many points in another picture.

After applying these two method to choosing match pairs, we can feel intuitively that these match pairs are much better than formal ones.

2.3. RANSAC

Get some homography matrices via homography in stitch.py and use RANSAC algorithm in RANSAC in stitch.py to find the most suitable homography matrix.

2.4. Transform

Use the library function cv2.warpPerspective() to transform a picture with the support of homography matrix gotten from RANSAC in stitch.py

2.5. Stitch

Cover picture 2 on picture 1 which had been transformed before.

3. Code

This part is about how to run my code. First, put 4 pictures needed to be stitched in the same folder as stitch.py. In the main function of stitch.py, a list named all picture should contains all pictures needed to be stitched, and we should keep the order from right to left to make sure that the code will run correctly.

The result of stitching right 1 and right 2 will be kept in result1.jpg, the result of stitching right 3 and result1.jpg will be kept in result2.jpg, and the final result will be kept in result.

4. Mathematical Formula

4.1. $\rho(\mathbf{u}, \mathbf{v})$

Definition: $\rho(\mathbf{u}, \mathbf{v})$ is the similarity of $\mathbf{u}$ and $\mathbf{v}$.

$\mathbf{u}$ is one of the vector in picture 1's feature vector list and $\mathbf{v}$ is one of the vector in picture 2's feature vector list.

Because the $\mathbf{u}$ and $\mathbf{v}$ a vector in $\mathbf{R}^{128}$, the cosine angle between two vectors can express the similarity of these two vectors.

$$\rho(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\|_2 \cdot \|\mathbf{v}\|_2}$$

4.2. Homography Matrix

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \sim \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

After several transform, we can get a new equation.

$$\begin{bmatrix} \mathrm{x}_1^\top & 0 & -x_1\mathrm{x}_1' & -y_1\mathrm{x}_1' & -x_1' \\ 0 & \mathrm{x}_1^\top & -x_1\mathrm{y}_1' & -y_1\mathrm{y}_1' & -y_1' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

In the same way, we can get similar equation on another 3 pairs of points. Then we get solve the homography matrix by numpy.linalg.solve().

5. Result

First have a look at the results of 3 pairs of pictures.

These pictures are in ./photo/pair1 or pair2 or pair3 for each group respectively.

6. Problems and Solutions

There are several problems found .

6.1. Picture Size

The pictures taken by our mobile phone have a very high definition, which have about 100 thousand key points and feature vectors. So if applying SIFT algorithm directly to the original picture, it may take

us a lot of time when matching key points due to the $O(n^2)$ complexity.

My solution is to resize the picture from 1440*1080 to 640*480 so less key points will be detected by SIFT algorithm, and it can accelerate the speed of the process to a great extent.

6.2. Picture Deformation

Because of the sequence of stitching pictures, the landscape on the right is blurred after stitching the whole panorama.

My solution is to make the landscape on the right farther than the one on the left, so the deformation after stitching won't be so obvious.

There is a foreseeable method that is to change the sequence of stitching and final location of the panorama from left-side to the middle, which needs to make some improvements on the algorithm.

6.3. Obvious Splice Seam

The main reason for the appearance of an obvious splice seam is that cameras are able to automatic parameter adjustment to take a better picture. So parameters like shutter time, aperture and so on may be different for every specific shoot.

There are two foreseeable methods. Method 1 is to apply linear interpolation along the nearby the region of splice seam. Method 2 is to use Poisson Fusion, which is better than Method 1 and can make a more nature panorama.