Panorama Stitching

22221290 Jiarun Liu

Environment

- Python 3.7+ required

```
cd code
pip install -r requirements.txt
```

Feature Detection, Descriptor and Matching

Details

Two methods are implemented in this part, one is SIFT and the other is descriptor formed by concatenated pixel values in a local window. We have both OpenCV implementation and self-written implementation.

The core function is designed as below:

```
def match(img1, img2, mode='sift', k=2, r=0.7, is_default=True, is_show=False):
    """
    img1: first image
    img2: second image
    mode: choose between 'sift' and 'cpv' for SIFT features & descriptors or
descriptors formed by concatenated pixel values
    k: param for knn
    r: ratio of inliers
    is_default: True for using OpenCV implementation, False for using self-
written one.
    is_show: show temp results
    """
    ...
    return kp1, kp2, matches    # key points and matching results
```

The k-nearest-neighbor algorithm is implemented in `knn()`. It can get the same result of OpenCV one but runs much slower.

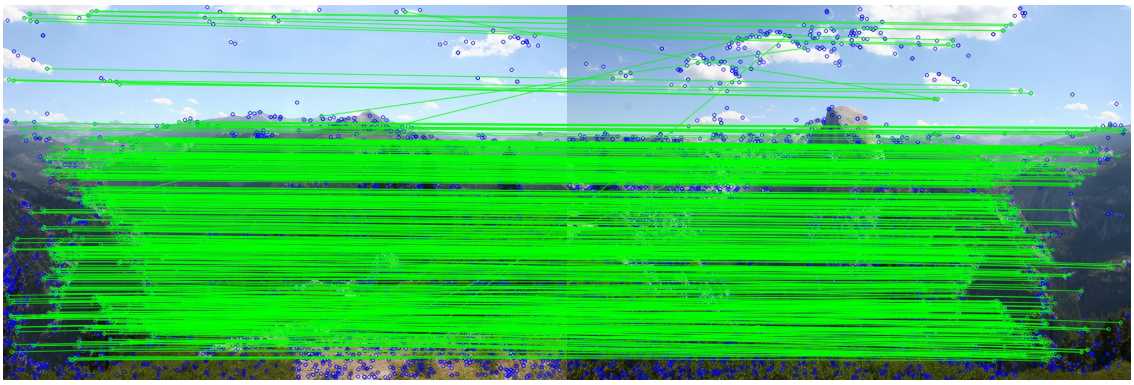
Running

```
cd code
python feature_matching.py
```

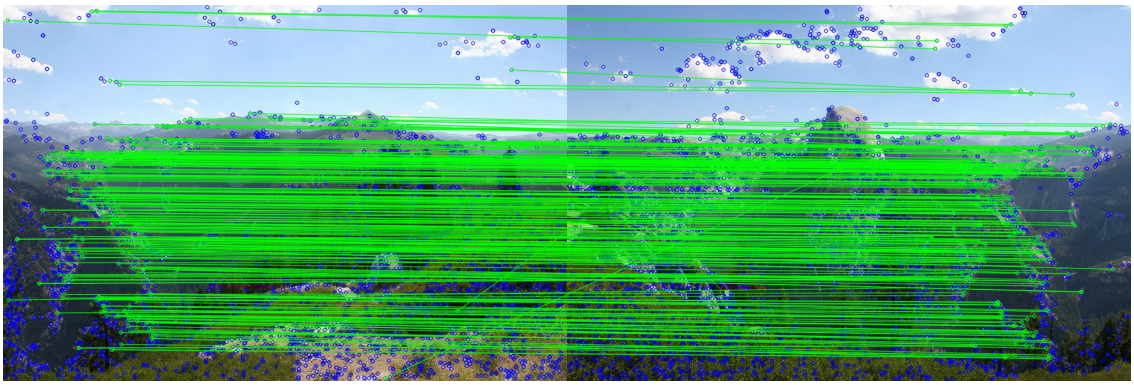
Please check `feature_matching.py` before running. Path to the images may need to be changed.

Results

- SIFT detector & descriptor



- SIFT detector & concatenated pixel values descriptor



Estimation of Homography

Details

OpenCV has a quick function for homography estimation which is `cv2.findHomography()`. A hand-written version for DLT and RANSAC algorithm to solve homography is also implemented.

The core function is designed as below:

```
def solve_homography(kp1, kp2, matches, threshold=5.0, max_iters=2000,
is_default=True)
    """
    kp1, kp2 and matches are results from match() function
    threshold: RANSAC filtering threshold
    max_iters: the initial iteration times for RANSAC
    is_default: True for using OpenCV implementation, False for using self-
written one.
    """
    ...
    return H    # a 3x3 homography matrix
```

An adapted version to decide the number of iterations according to our estimated ratio of outliers is also implemented in function `ransac_update_num_iters()`.

Running

```
cd code
python homography.py
```

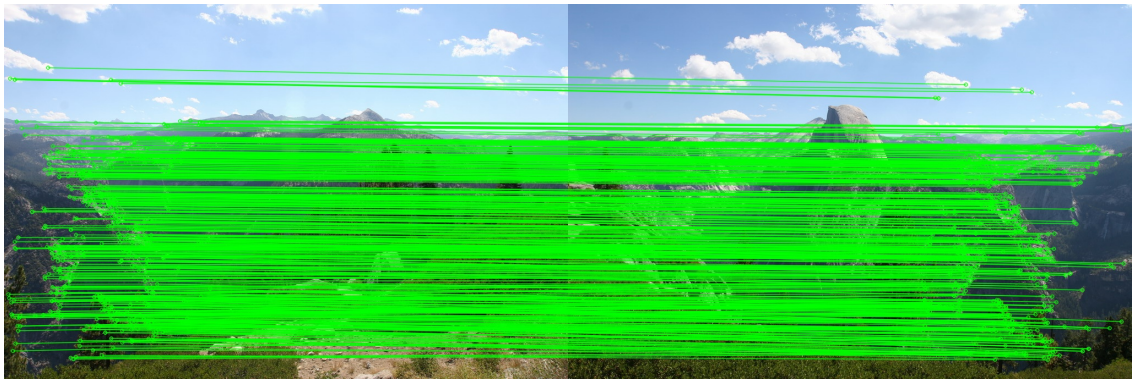
Please check `homography.py` before running. Path to the images may need to be changed.

Results

First we'll check the difference between OpenCV version and our solution:

```
# Result of cv2.findHomography()
[[ 1.24002276e+00 -1.31647610e-01 -2.01609885e+03]
 [ 2.03076531e-01  1.18323373e+00 -5.72358075e+02]
 [ 8.09221095e-05 -2.15928443e-06  1.00000000e+00]]
# Result of ransac()
[[ 1.26154433e+00 -1.20595906e-01 -2.06183878e+03]
 [ 2.10323160e-01  1.21232343e+00 -5.97800764e+02]
 [ 8.72400735e-05 -3.53953180e-06  1.00000000e+00]]
```

- SIFT detector & descriptor + RANSAC



- SIFT detector & concatenated pixel values descriptor + RANSAC

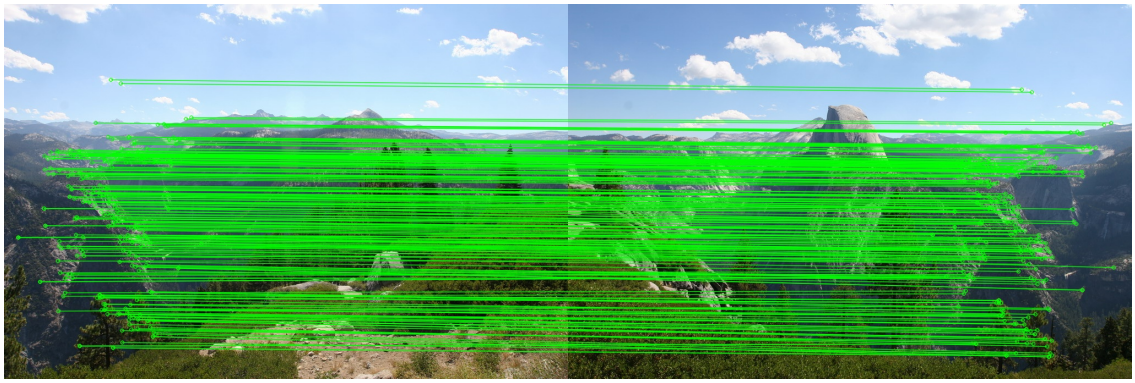


Image Stitching

Details

In image stitching part, we use linear blending and shifting method to create a panorama image. A blog has been referred to help me finishing this task: [Image Stitching_A Simplistic Tutorial - BitsMakeMeCrazy_Kushal Vyas's Blog](#).

During image warping, we first choose the center image and then apply matching and find homography matrix in the left part and the right part of the image set separately. Say, we have a homography matrix H . If the starting coordinate of each image is $(0, 0)$ and end point is (r, c) , we can get the new warped image dimension by $start_p = H \times (0, 0)$ uptil $end_p = H \times (r, c)$.

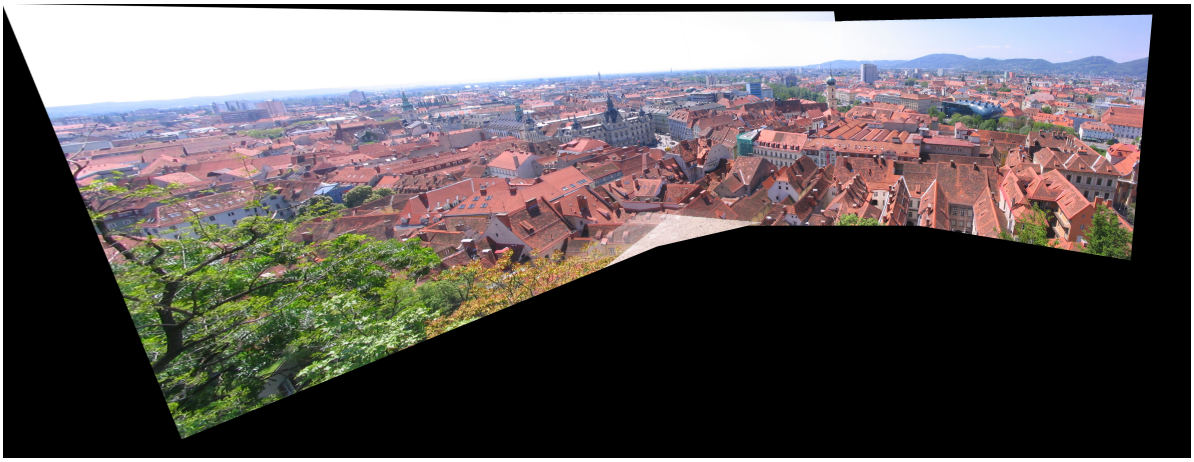
Check `panorama.py` for more details.

Running

```
cd code
python panorama.py
```

Please check `panorama.py` before running. Path to the images may need to be changed.

Results



Analysis and Discussion

The results above shows difference between different feature descriptors. Because of the direction and scale factor preservation, the SIFT descriptor is more robust and precise. Some how, all of these methods have some outliers. After applying RANSAC to these methods, we can get a better matching result.

	SIFT	Concatenated Pixel Values
Robustness	good	poor
Matching numbers	larger	smaller
Correctness	medium	medium

During homography fitting, a problem is to solve DLT problem by using SVD. According to the function:

$$\mathbf{A}_i\mathbf{h} = 0$$

It's easy to get the idea, but the accurate value of matrix $\mathbf{A}$ is the real problem. We find several solution on different reference books. Actually, this does not cause too much residual on results.

Another issue is the order of image when calculating homography matrix. At first we cannot get a correct result while image blending for there is no intersection part between two warped images. In the end we find that it's due to the wrong order while passing to `solve_homography()` function, and we get the inverse matrix of homography matrix.