161180038

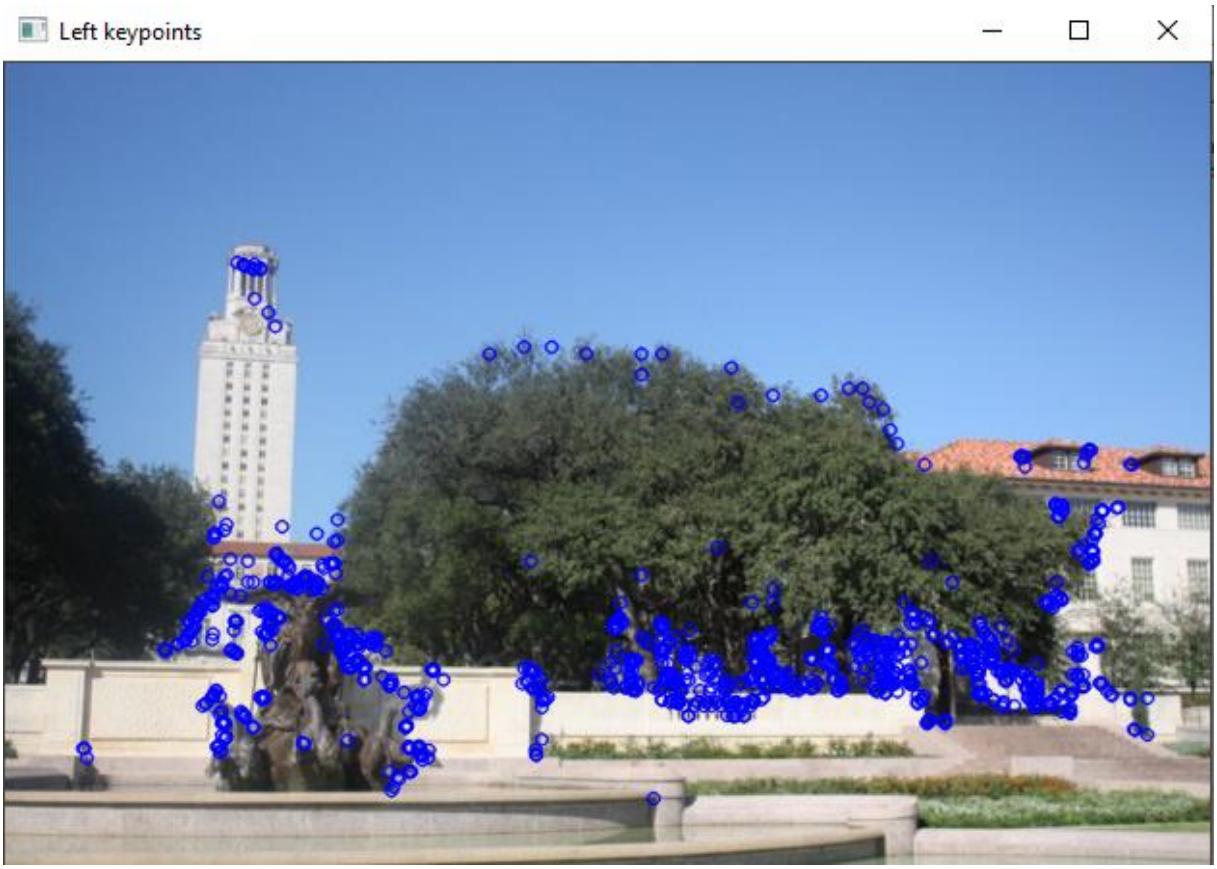
Hilal Ilgaz

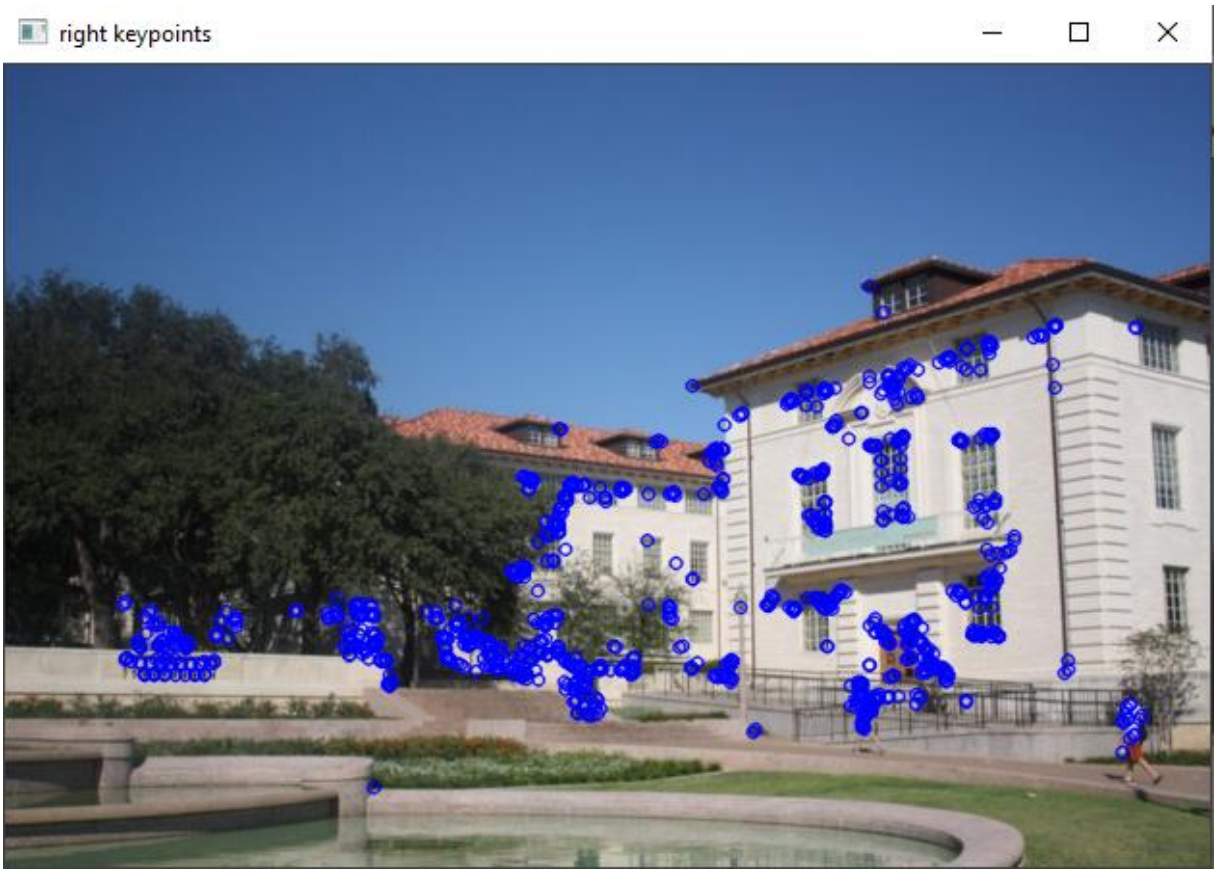
CENG471 Assignment 3

The Report

If you want to capture a large scene and the camera used to capture this scene has a resolution of 640x480, then a panoramic view cannot be achieved under these conditions. Image stitching is used because panoramic images cannot be obtained. In other words, image stitching is basically capturing multiple images of a large scene and superimposing these images with the help of various algorithms to obtain a single image of the big scene. My goal is stiching two images which is captured in same place but different directions.

First of all, I used ORB for detecting and computing keypoints and descriptors. ORB means Oriented FAST and Rotated BRIEF. This algorithm was developed by Ethan Rublee, Vincent Rabaud, Kurt Konolige, Gary Bradski. The purpose of the development of this algorithm is to use it instead of SURF and SIFT algorithms. Because SURF and SIFT algorithms are patented and not free to use. ORB can perform like SIFT for feature detection. ORB's aim is FAST keypoint detector and BRIEF descriptor [1]. I created an ORB model using ORB_create(nFeatures=1000) method. nFeatures is used for detecting more features. Default value is 500. After that I detected and computed keypoints and descriptors using detectAndCompute() method for both sides of the image. For every keypoint which is detected, I have a descriptor. These descriptors are arrays for definition of the keypoints. drawKeypoints() method used for drawing keypoints. For two sides of the image, I used twice this method.

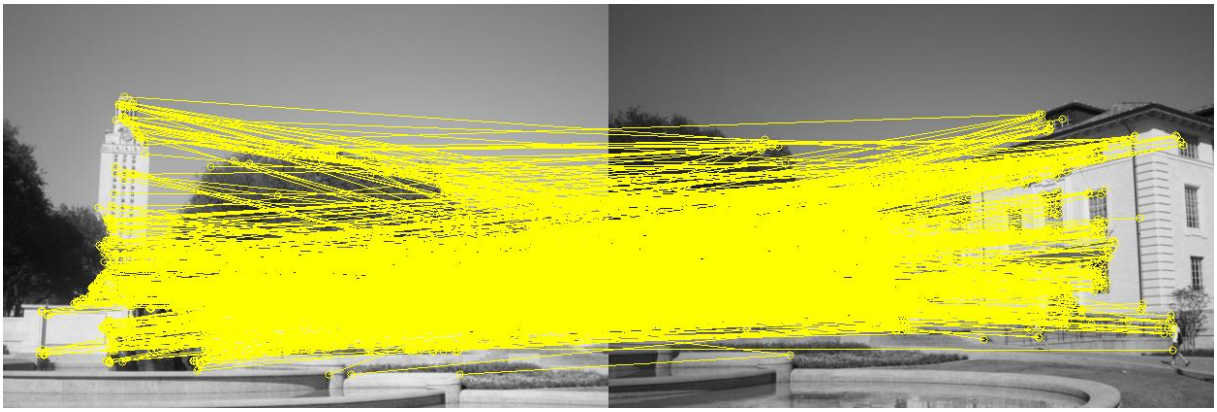

Left keypoints


right keypoints

```
Descriptor for the first left keypoint:
[139  18 116 247  99  61 184  88 181  90 149 183 253  93 191 111 255 145
 223  39 164  71 110 194  87 219 114 143  54 252  37 247]

Descriptor for the first right keypoint:
[  2  80   8 155 109 137 177  76  30  29  84 163 172 223 107  73 202  25
 163 193  66  22  66 241  68  17 250 171  20  28  29 155]
```

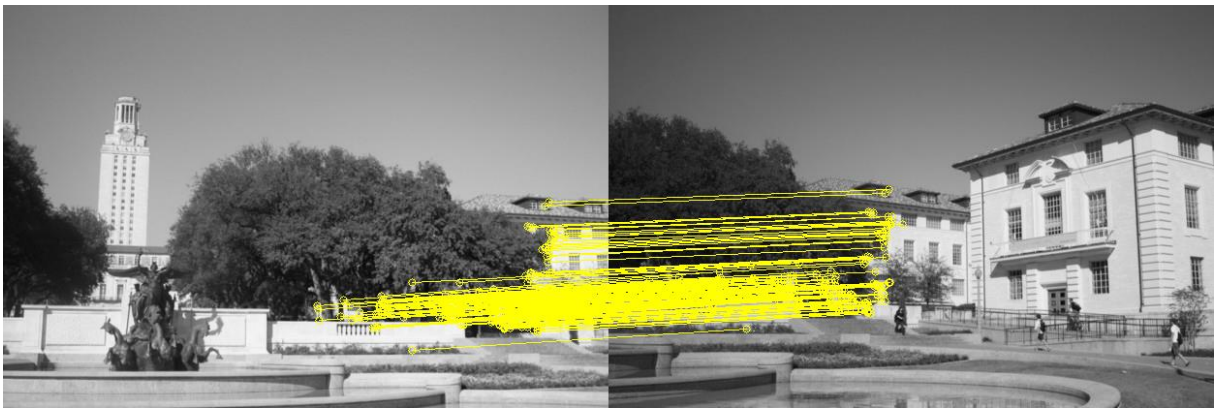
Next step is matching keypoints between the images. For matching, descriptors from the two sides of image should be compared and those matching points are sorted by their distance. For this process, I used BFMatcher. Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation and the closest one is returned. Create() method has a normType parameter. This parameter is optional but uses for specifying the distance. I used NORM_HAMMING. This is Hamming distance. I wanted to work with best matches. I used knnMatch() method. This method returns k best matches where k is specified by the user [2]. In my code k is 2. After this process, next thing is drawing matches. I did not use a ready-to-use method fort hat. I write my own draw_matches() method. This method takes 5 input parameters. Those are images, keypoints and matches. My algorithm is creating a black image after taking shape of images. The black image will become my output image. So, it has to be equal size with input images. So, I set the number of rows and columns. First image is created by me is black, but it has to be colorful. I created three channels and stacked them to my output image. For drawing all matches on the output image, I used queryIdx and trainIdx methods. The other methods are used for drawing circles and lines. queryIdx is query descriptor index in the first image. trainIdx is train descriptor index in the second image. With using draw_matches() method I drew all matches. For understandable result I also drew 30 matches. The results are below.



All matches between two images' keypoints

30 matches between two images' keypoints.

My threshold value is 0.6 for eliminating bad matches. The result is below.



The last step is warping images on one image. I also wrote my own method for that. matchimages() method take 3 input parameters. These are images and homography value. I create some lists for representing coordinates of the reference image. perspectiveTransform() method is used for calculating homography matrix. For warping second image, I used warpPerspective() method. This method's inputs are second image, output image's width and height and transformation matrix. For applying this method I set minimum number of matches which is 15. After that I changed keypoint type to float. The reason is using these for findHomography() method. findHomography() method takes RANSAC as an input. According to this bad matches' percentage higher than good matches' percentage, result can be accurate [3]. The stitched image is below.

References

1. Rublee E. Rabaud V. (2011) *ORB: an efficient alternative to SIFT or SURF*. Web: http://www.willowgarage.com/sites/default/files/orb_final.pdf
2. *Basics of Brute Force Matcher*. Web: https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html
3. *Feature Matching + Homography to find Objects*. Web: https://docs.opencv.org/master/d1/de0/tutorial_py_feature_homography.html