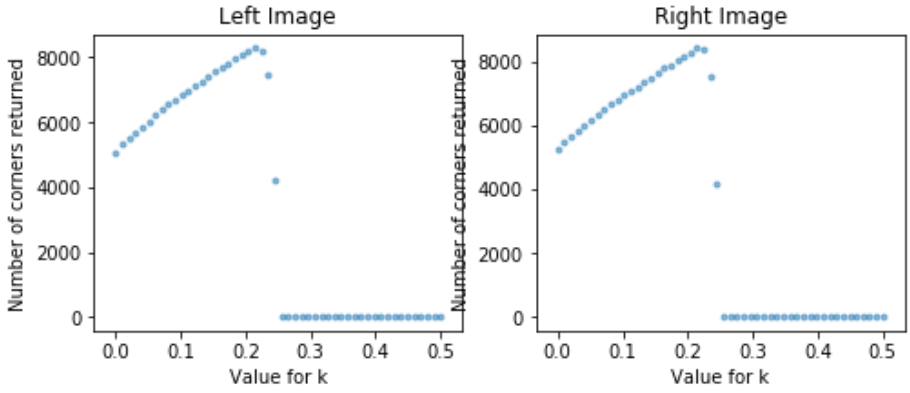
Dogan Aykas
6227725

Computer Vision Image Stitching Assignment

The assignment focuses on the task of stitching two images through several tasks mentioned in the steps stage. The report is organized under each step, describing the implementations and depicting some graphs and pictures.

Extracting keypoints with Harris Corner Detection

After uploading the two images with the help of the python library opencv, it is necessary to convert the images to grayscale in order to retrieve the corners assigned by Harris Corner algorithm. The imported function corner_harris calculates the matrix M that calculates the derivatives in each direction with the grayvalues. After the matrix is calculated, the images are filtered with the pixels that have a high corner likelihood. The default parameter for alpha for corner_harris is 0.05. A sensitivity analysis of the number of corners returned from this function is included in the code. Below are the graphs of the correlation of k and number of corners returned. For both images, the k-value that returned the most number of corners is 0.214.



After the retrieval of the indices of harris corners, it is needed to convert these indices to keypoints that are in the shape of a vector.

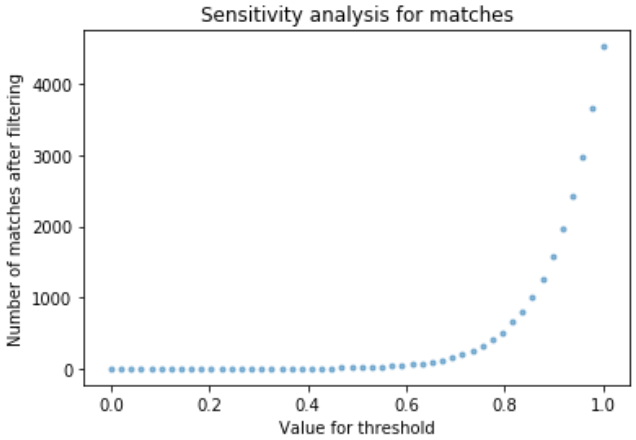
Computing SIFT Descriptors

Although we have returned the keypoints, there is no way to compare the keypoints in each image since these keypoints have only information about positions of the corner pixels. In order to compare the keypoints in two images, the descriptors for each keypoint is calculated.

Computing Distances Between Descriptors

The steps in the assignment suggest the normalization of the descriptor values before being compared. So, after normalizing each descriptor array, euclidean distance of a descriptor in one image with all the descriptors in the other image is calculated. After sorting the distances are calculated, a match is created if an only if the ratio of the nearest neighbor over second nearest neighbor is below a chosen threshold (0.8). Below is the correlation between the choice of k on the number of matches after the filtering occurs.

Dogan Aykas
6227725



RANSAC Implementation

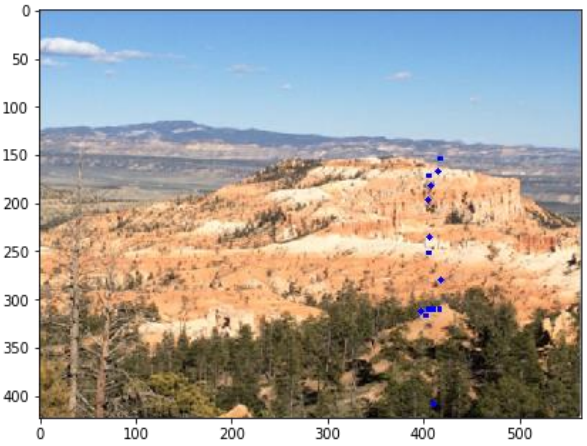
The result of this RANSAC implementation changes each time the code is run. I will stick to the parameters I got on my final running of the code.

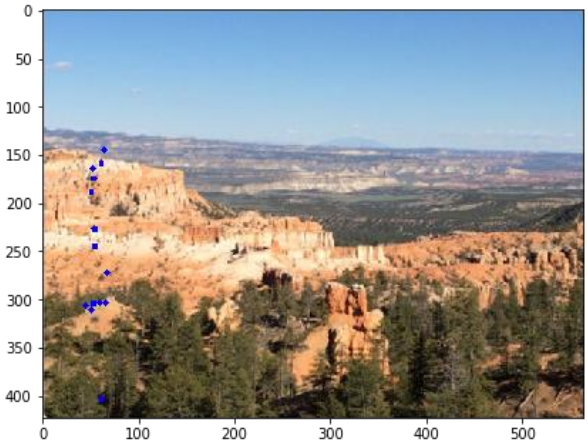
After the retrieval of matches between two images, we need to figure out a transformation that transforms the positions of keypoints in one image to the other one. Since the assignment hints to use an affine transformation, I used cv2.getAffineTransform function to calculate the desired transformation matrix, with dimensions 3*2. The problem is that there are 531 matches, and the affineTransformation calculates a transformation of 3 points in one image to their matches in the other image. Since it could be a non-useful transformation if we only consider 3 initialization keypoints, implementing a RANSAC implementation is required to return the best transformation after a number of iterations.

The implemented RANSAC function require 4 arguments: m1, m2, I and err. The variables m1 and m2 are the indices of matches in both images, "I" is the number of iterations for the RANSAC algorithm and err is the d value in order to calculate the number of inliers and outliers of each transformation. From experimentation through this assignment, I have had the best results using 2 for err and 10000 for number of iterations. The algorithm basically takes 3 sample matches, create the corresponding affine transform. Then the number of inliers and outliers is calculated with respect to the value err. These series of operations is repeated I times where the algorithm returns the best transformation that has the most number of inliers out of all matches. The returned 3*2 shaped affine transformation for the images is: $\begin{matrix} 0.84 & -0.02 & 365.76 \\ 0.06 & 0.99 & 7.41 \end{matrix}$

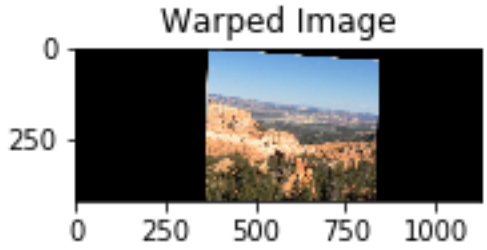
Out of the 531 matches fed into the RANSAC algorithm, the calculated affine transformation has a number of 14 inliers while the remaining 517 matches are outliers. The average residuals for the 14 inliers is 0.025 as calculated in the code, which is a logical result in the sense that the threshold distance to be an inlier is 2. Below is the representation of two images with these 14 inliers displayed with color blue.

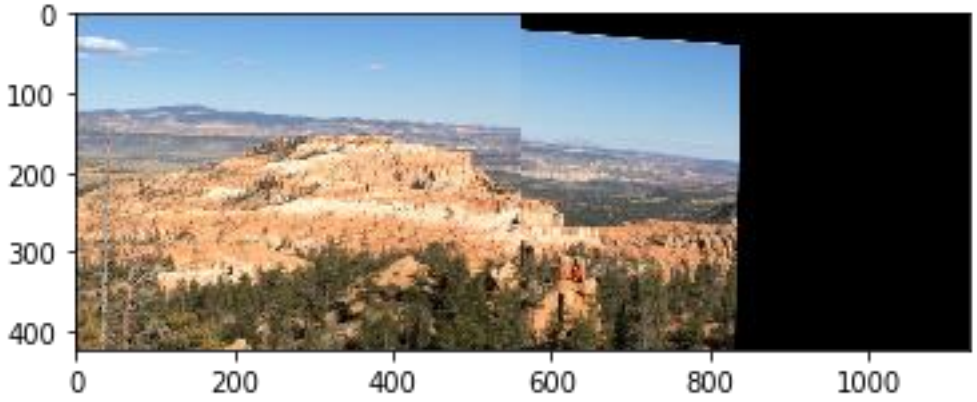
Dogan Aykas
6227725





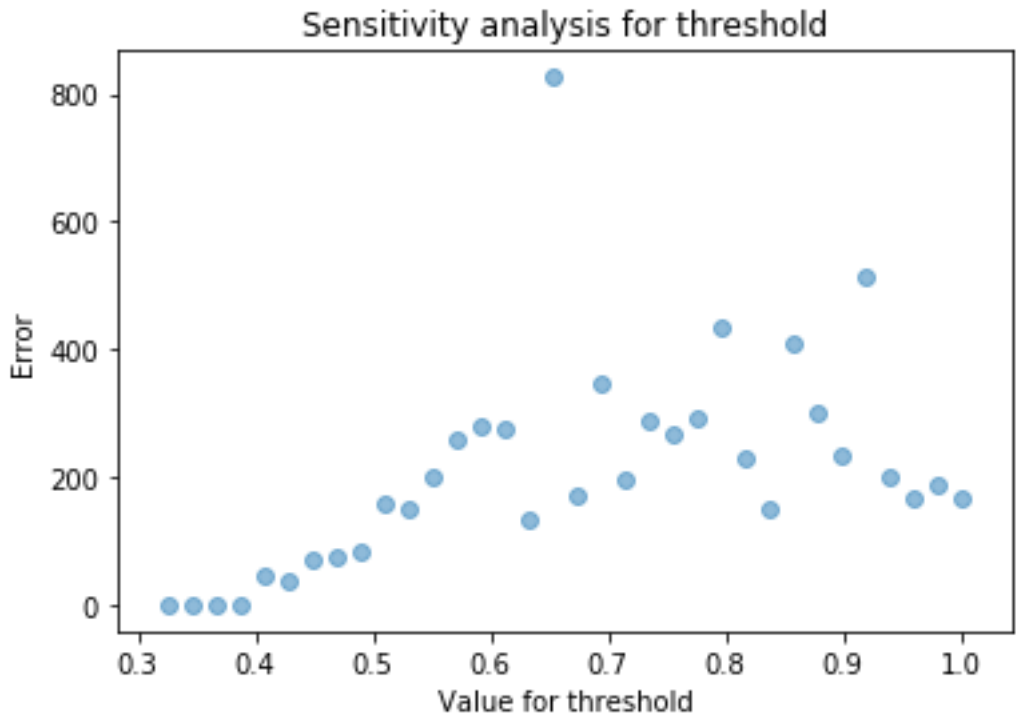
The next step is the stitch these two images with respect to the calculated affine transformation using warpAffine function. After the necessary operations below is the transformation that transforms the right image to left. The second picture depicts the stitched images.





The computed affine transformation has an average error/distance of 327.41 as show in the code. The value will change after running each script since RANSAC samples its starting matches. In “Computing Distances Between Descriptors” part, the correlation between the choice for the threshold and number of matches were shown. For sensitivity analysis, we need to observe the change of that threshold to the accuracy, which we defined as the euclidean distance between keypoints after acquiring RANSAC implementation. Below is the graph of this sensitivity analysis.

Dogan Aykas
6227725



For smaller values than 0.3, there were not enough matches for the RANSAC function to run so there is no transformation returned for those values. As it is depicted in the graph, the values between 0.3 and 0.4 have really small errors although they should not be considered as the code returns so few matches that does not help the RANSAC algorithm during its iterations. It can be seen that the values between 0.7 and 0.9 can be considered. There is a considerable amount of variance since this sensitivity analysis requires multiple iterations each time, thus I needed to limit the number of iterations for computational complexity.