BBM418 Computer Vision Lab.
Assignment 2 - Image Stitching with Keypoint Descriptors

Fatma Nur Demirbas,
21727116
Department of Computer Engineering
Hacettepe University
Ankara, Turkey
b21727116@cs.hacettepe.edu.tr

Overview

In this assignment, it is aimed to obtain a final panorama image that will combine the subimages provided using key point identification methods (SIFT/SURF and ORB) and include all the scenes in the subimages.

1 Implementation

The names of all directories in the dataset directory are first retrieved. The first and second photos from the first panoramic directory are read and blended after that. If there aren't enough matches to generate the homography matrix (I need to use this error handling because it can't find more than 4 good matches in pano to find matches with the BF matcher algorithm using the key points found by the SIFT/SURF ORB, an error message and the merge function are returned, and the next panorama images (index) are retrieved. In my first plan, I blended the first two photos if there is no difficulty. The merged image then started merging with subsequent images (img12+img3 = img123:::). If there were more than 4 good matches, the pictures would be combined and the next picture would be taken. But I got a lot of chars and distortions in this plan. Then, a time-consuming but better way to get a better image than the first one: Blending each image with the next one, a series of pairs containing subimages was created. This continues until the length of the array reaches 1. When it is 1, it saves the picture. (img1+img2 = img12; img2+img3 = img23; …… ; img12+img23=img123:::).

The left and right images are first transformed to gray scale in the merge procedure. Then SIFT/SURF and ORB are utilized to extract features. Key points on each image are plotted after obtaining key points and their descriptors. Following that, a brute-force matcher (BF matcher) is employed to kknMatch features. It uses a distance calculation to match the descriptor of one feature in the first set with all other features in the second set. And the one that is the closest is returned. All matches are sorted, and the top 50 are saved as good matches. Following that, excellent matching spots are plotted.

Their corresponding coordinates for left and right images are produced after getting satisfactory matches: Two empty numpy arrays with two columns (x,y) and rows with the number of good matches are formed. After that, each match is iterated. The descriptor index of each matching is obtained for each image using queryIdx and trainIdx. Then, using a keypoint with the same index as the descriptor index, the coordinates of suitable matching points are obtained.

findHomographyMatrix is called to get homography matrix after collecting satisfactory matching coordinates in left and right images. 4 random integer values are determined using the RANSAC technique via random.sample. Then, using these random integer values as index, four coordinates for the right and left picture are obtained. Then, using the equation A:h = 0, a manual matrix 'A' is generated to find the homography matrix. Because is the eigen-decomposition of A^T:A , we may deduce that h should be the eigenvector of A^T:A with an eigenvalue of zero (or, in the case of noise, the eigenvalue closest to zero), which is the smallest.[3]

The homography matrix is obtained in vector form as a result of this process. As a result, it is reformed into a 3x3 shape. The error between well matched points on the right picture and good matched points on the left image, multiplied by the homography matrix, is then calculated: On the left image, a numpy matrix of well-matched points is transposed (::x2 > 2x::). The 'ones' row is introduced so that matrix multiplication can be done. All coordinates are divided by their third element after being translated, therefore the third element can be fixed as 1 ([w x;w y;w] > [x; y; 1]). Then, to match the shape of the modified coordinates matrix that we acquired, the 'ones' column is added to the numpy matrix of excellent matched points on the right picture.[3]

After that, these matrices are utilized to determine the distance between each vector (corresponding coordinates on the left and right images) (Euclidian distance is used when calculating.). After obtaining all error values for each coordinate, these values are added together to provide an error value for the appropriate homography matrix. If the total error sum is less than the preceding sums, it is assigned as the minimal error sum and the homography matrix is updated to the current one. To get the best-fitting homography matrix, this technique is done N times (in my implementation, this N is 50.)

After obtaining the optimal fitting homography matrix, the warpPerspective function is used to apply the homography matrix to the right picture and blend the left and right images. To begin, a blank picture area is produced. Its dimensions are (height of left picture + width of right image) × (height of left image + width of right image). The left picture is then moved to the far left of that empty image area. Then, in nested loops, each potential coordinate value for the right picture is iterated. The inverse of the homography matrix is multiplied by each coordinate (if we multiplied by non-inverse homography matrix, then the right image will be transformed to the left: out of the free space coordinates and it will overlap with left image).After the coordinate has been changed, it is split by its third element, allowing us to x the third element as 1 ([w x;w y;w] > [x; y; 1]). The altered coordinate is next tested to see if it extends beyond the free image space's boundaries. If not, the color value for the corresponding position on free image space is assigned to that translated coordinate. Finally, dark pixels in the right-hand half of the combined picture are erased.

2 Result by Dataset

2.1 SIFT/SURF Feature Extraction

Patent required for SURF feature extraction. I saw it when I downgraded Python versions from the internet, but even though I downgraded, I couldn't work for SURF.

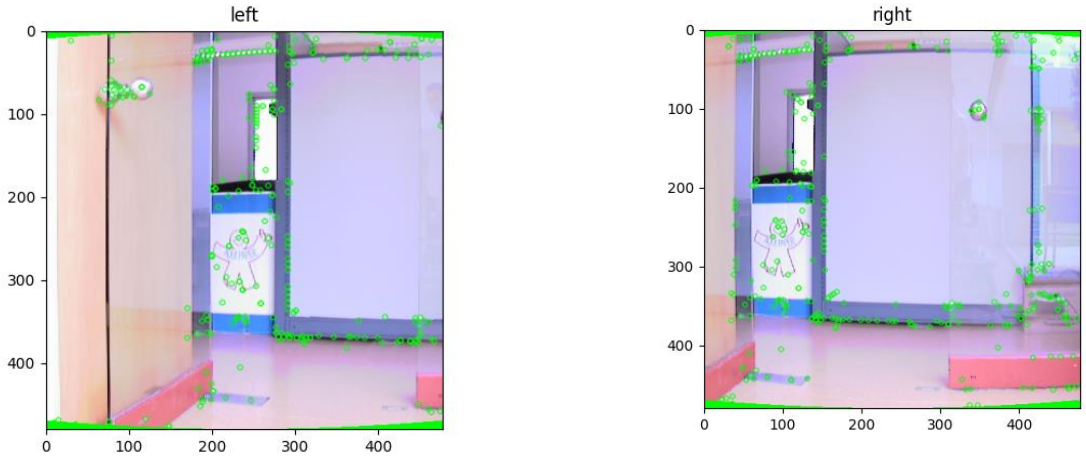

Figure 1: Pano1 – Feature Points of cyl_image05 and cyl_image06

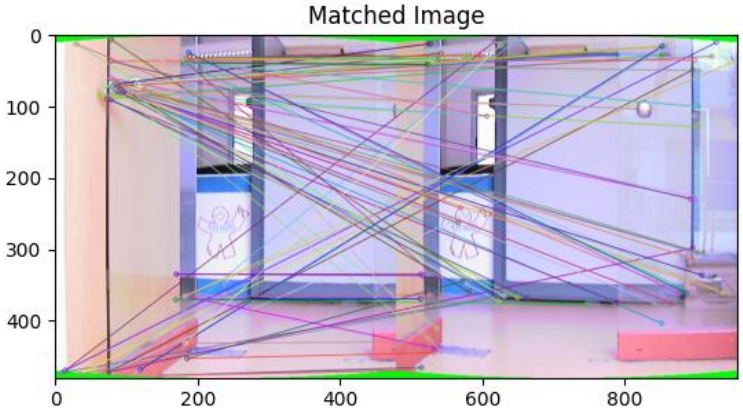

Figure 2: Pano1 - Feature Point Matching of cyl_image05 and cyl_image06

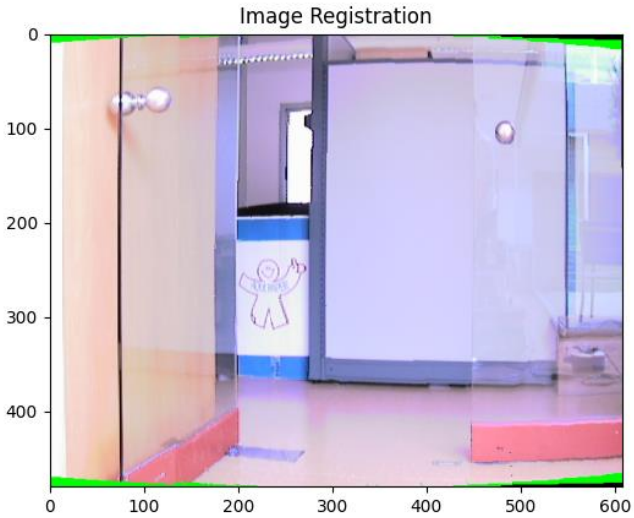

Figure 3: Pano1 – Subimages1 of cyl_image05 and cyl_image06

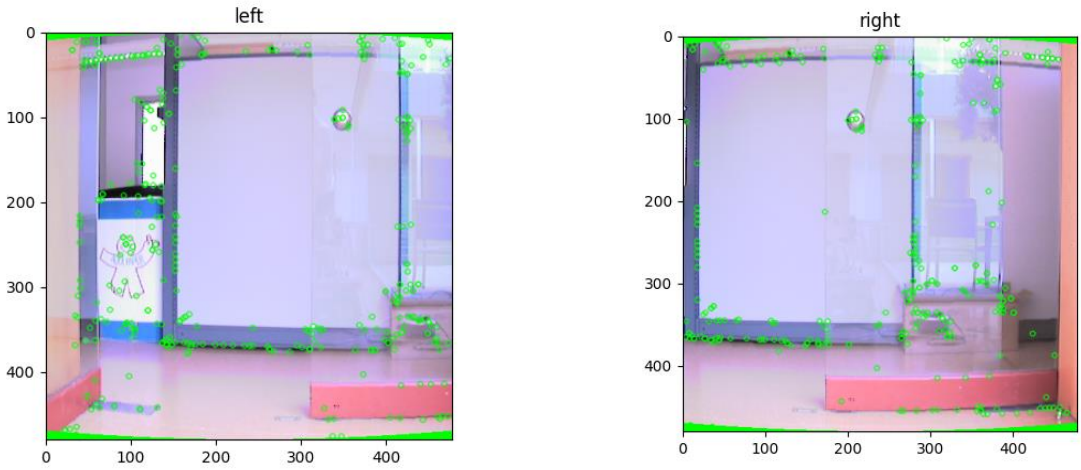

Figure 4: Pano1 – Feature Points of cyl_image06 and cyl

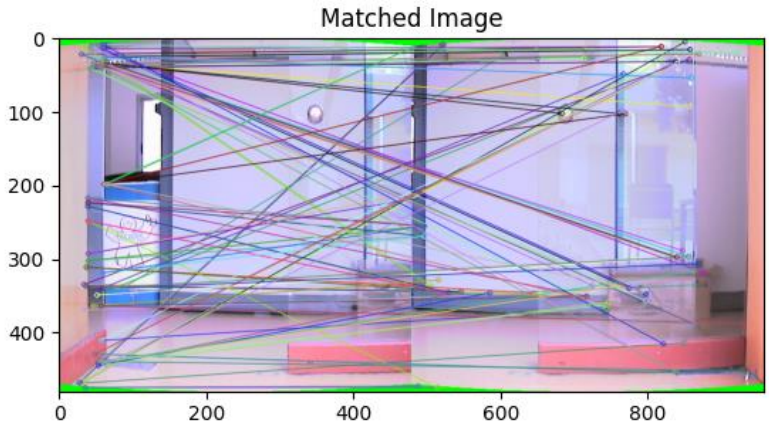

_image07

Figure 5: Pano1 - Feature Point Matching of cyl_image06 and cyl_image07

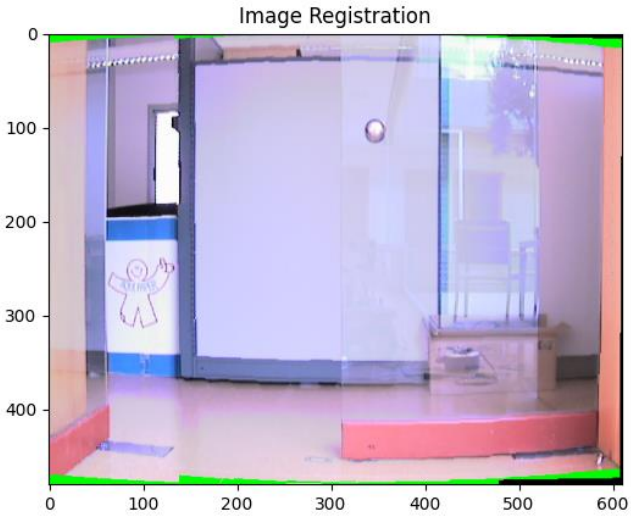

Figure 6: Pano1 – Subimages2 of cyl_image06 and cyl_image07

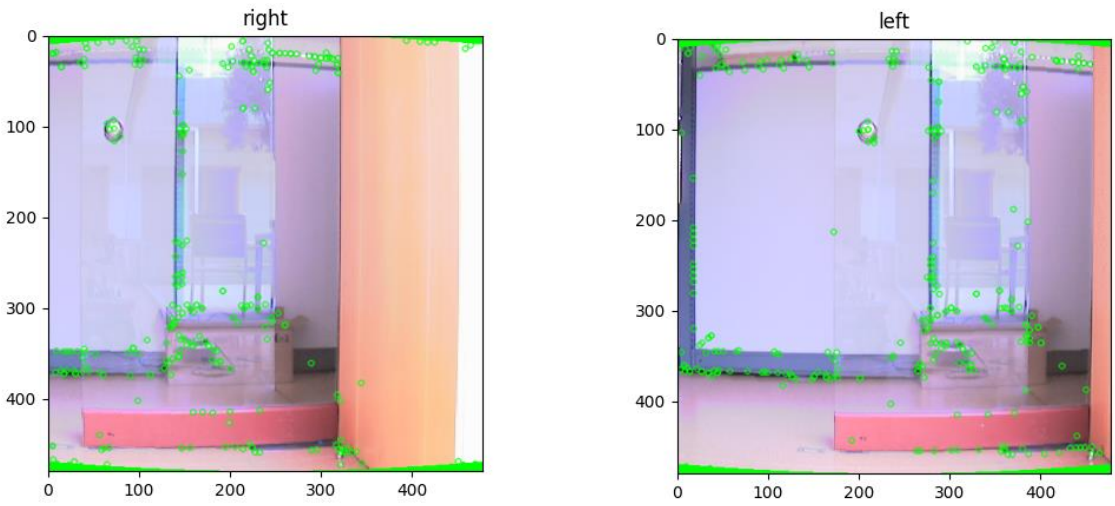

Figure 7: Pano1 – Feature Points of cyl_image07 and cyl_image08

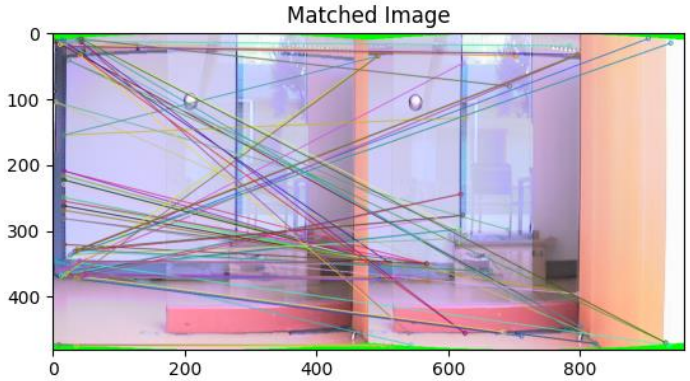

Figure 8: Pano1 - Feature Point Matching of cyl_image07 and cyl_image08

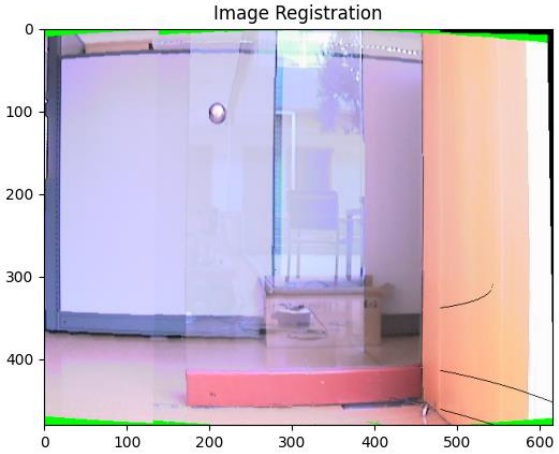

Figure 9: Pano1 – Subimages3 of cyl_image07 and cyl_image08

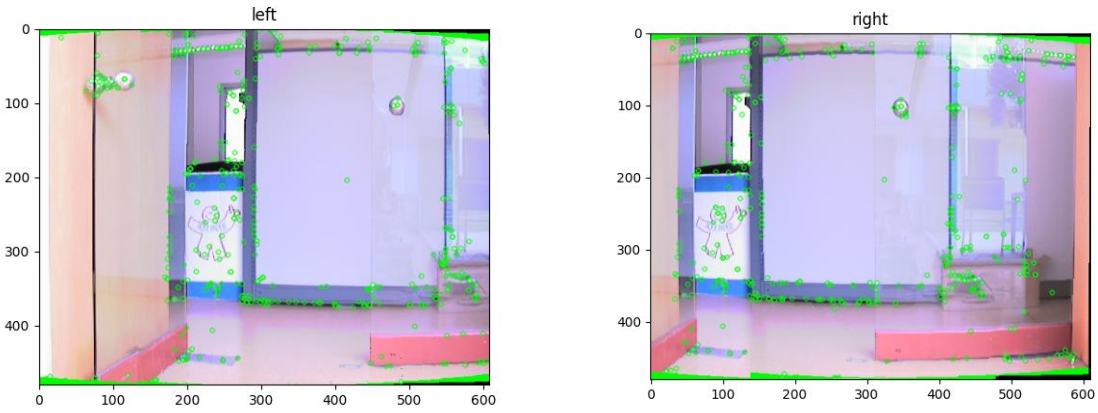

Figure 10: Pano1 – Feature Points of Subimages1 and Subimages2

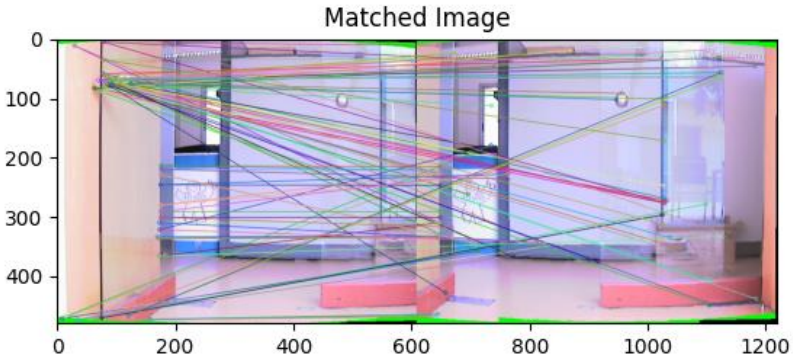

Figure 11: Pano1 - Feature Point Matching of Subimages1 and Subimages2

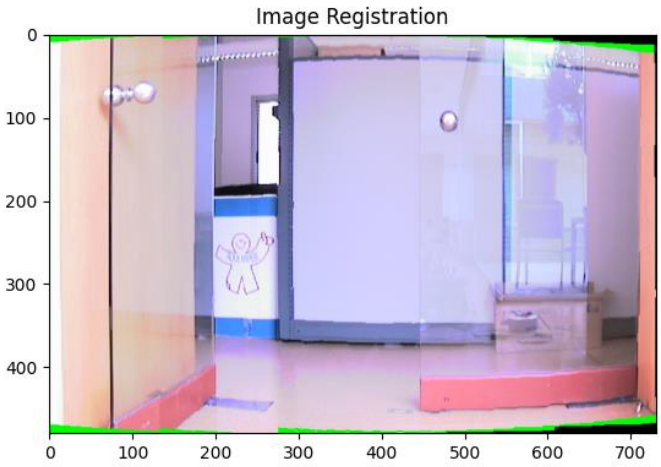

Figure 12: Pano1 – Subimages-1 of Subimages1 and Subimages2

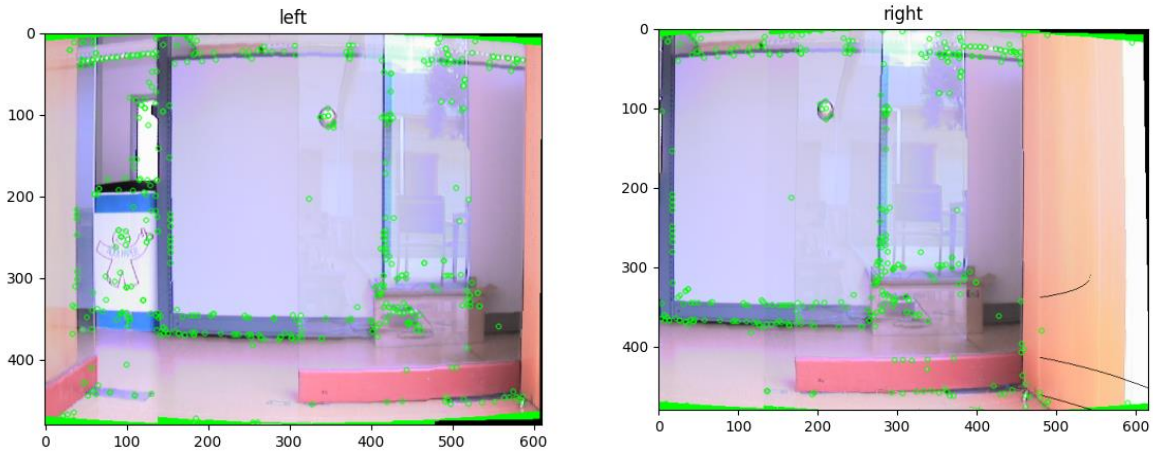

Figure 13: Pano1 – Feature Points of Subimages2 and Subimages3

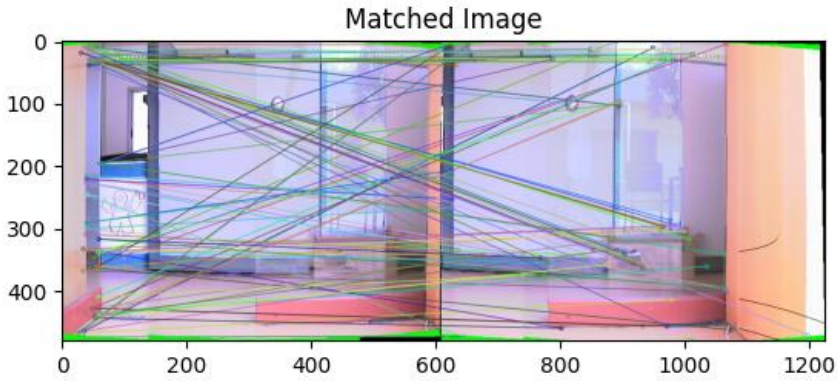

Figure 14: Pano1 - Feature Point Matching of Subimages2 and Subimages3

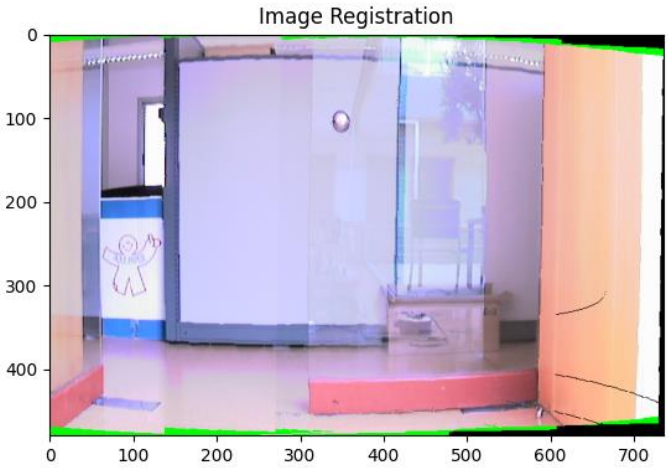

Figure 15: Pano1 – Subimages-2 of Subimages2 and Subimages3

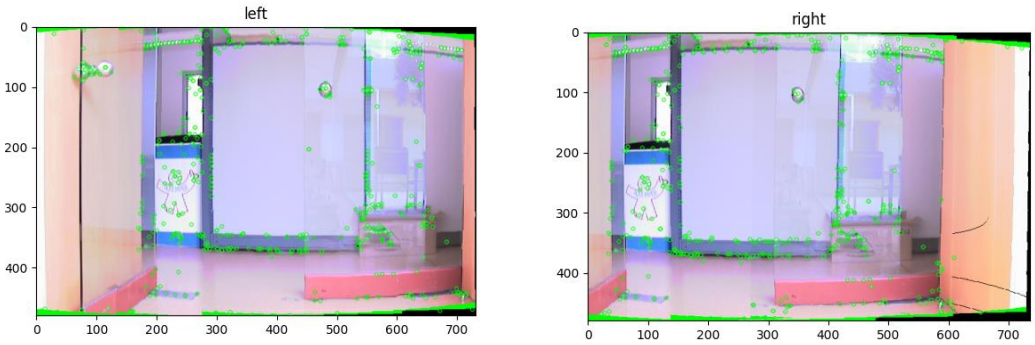

Figure 16: Pano1 – Feature Points of Subimages-1 and Subimages-2

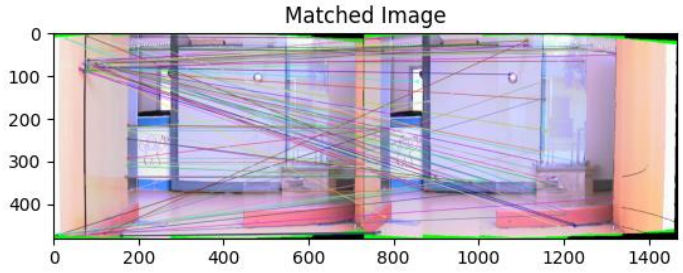

Figure 17: Pano1 - Feature Point Matching of Subimages-1 and Subimages-2

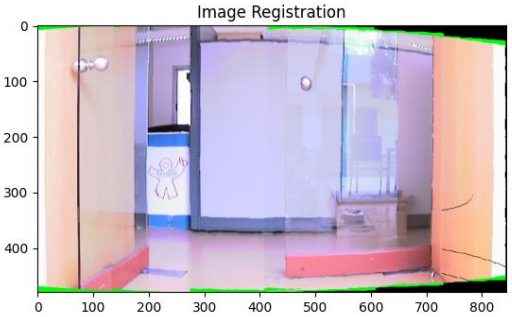

Figure 18: Pano1 – Subimages-2 of Subimages-1 and Subimages-2

Figure 19: Pano1 – My Constructed Panorama Image

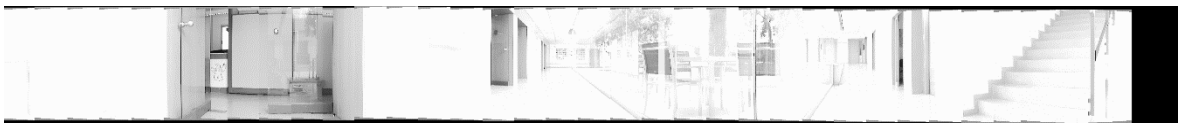

Figure 20: Pano1 - Ground Truth Panorama

2.2 ORB Feature Extraction

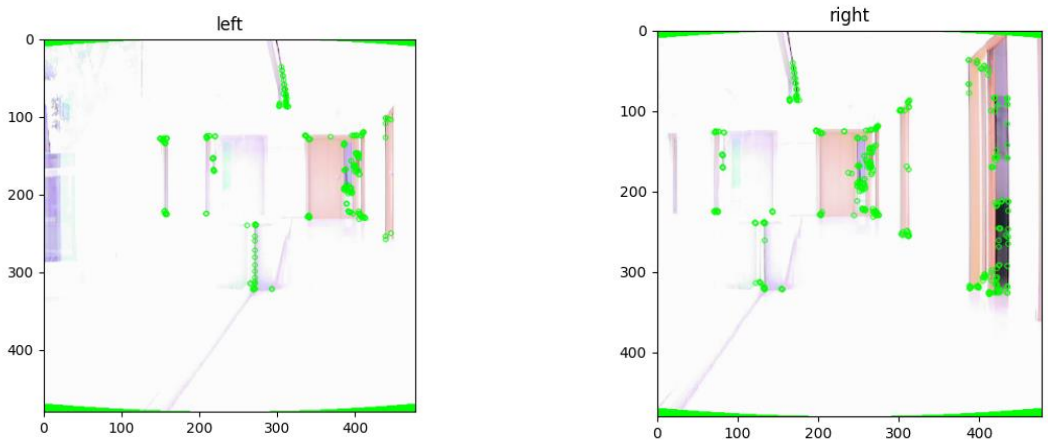

Figure 1: Pano2 – Feature Points of cyl_image24 and cyl_image25

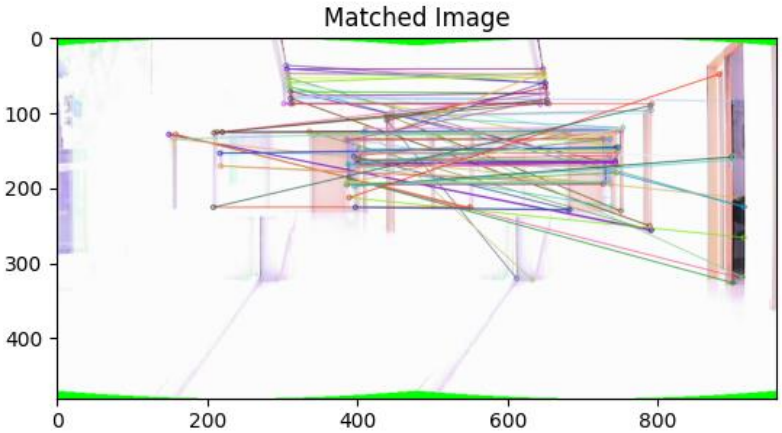

Figure 2: Pano2 - Feature Point Matching of cyl_image24 and cyl_image25

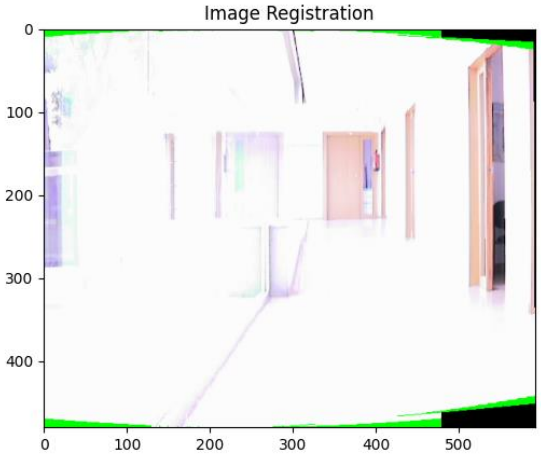

Figure 3: Pano2 – Subimages1 of cyl_image24 and cyl_image25

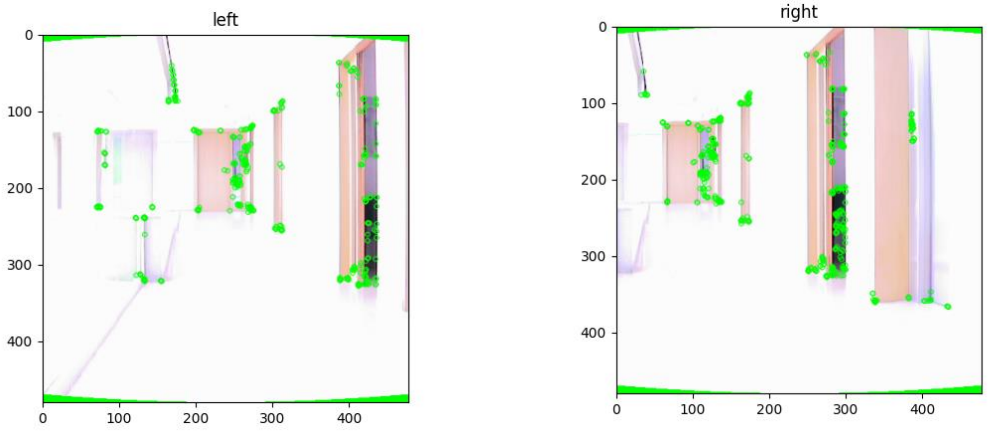

Figure 4: Pano2 – Feature Points of cyl_image25 and cyl_image26

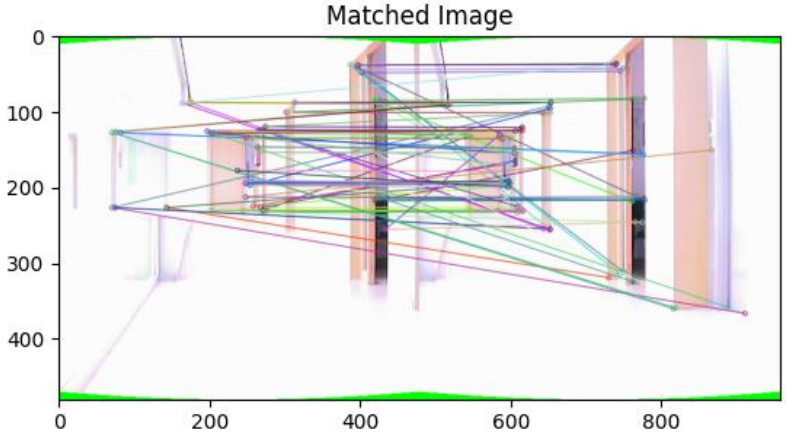

Figure 5: Pano2- Feature Point Matching of cyl_image25 and cyl_image26

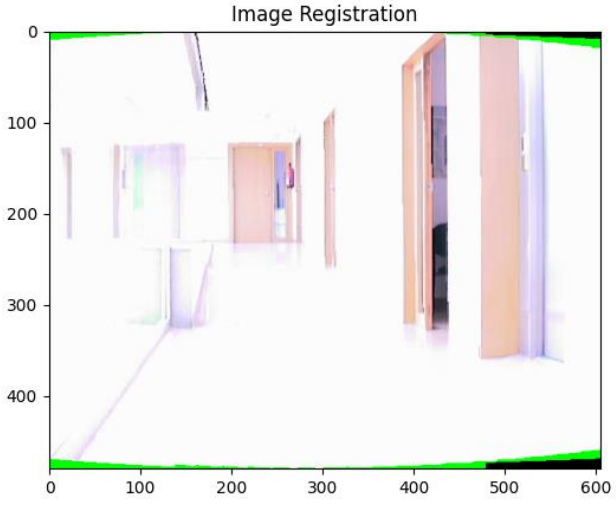

Figure 6: Pano2 – Subimages2 of cyl_image25 and cyl_image26

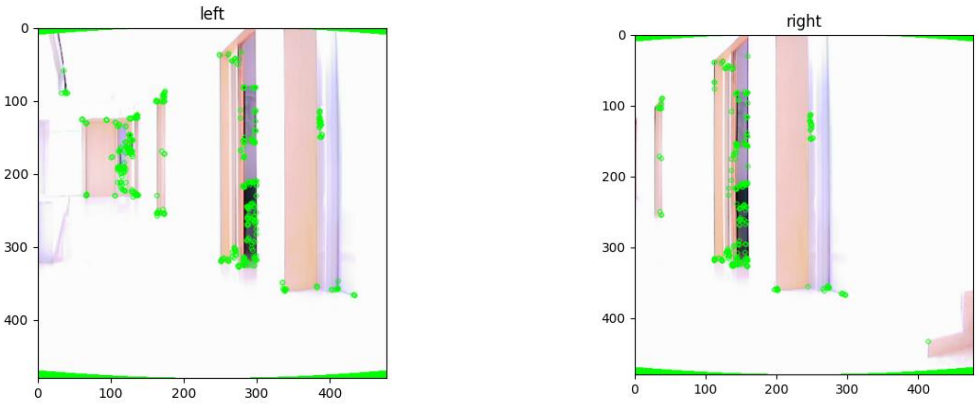

Figure 7: Pano2 – Feature Points of cyl_image26 and cyl_image27

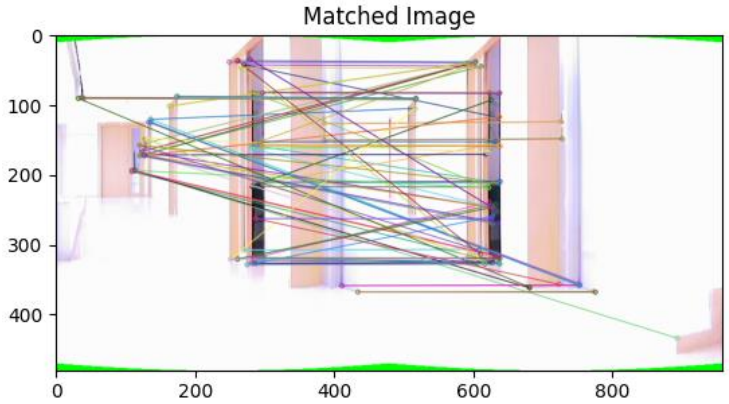

Figure 8: Pano - Feature Point Matching of cyl_image26 and cyl_image27

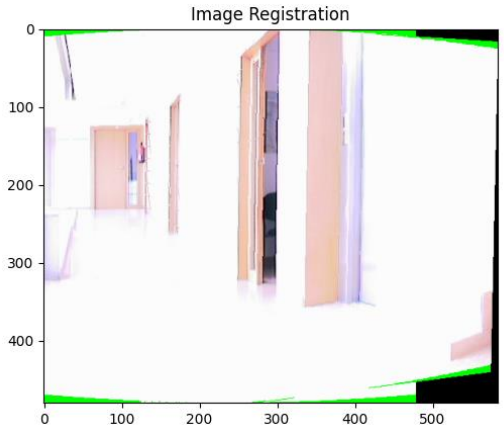

Figure 9: Pano2 – Subimages3 of cyl_image26 and cyl_image27

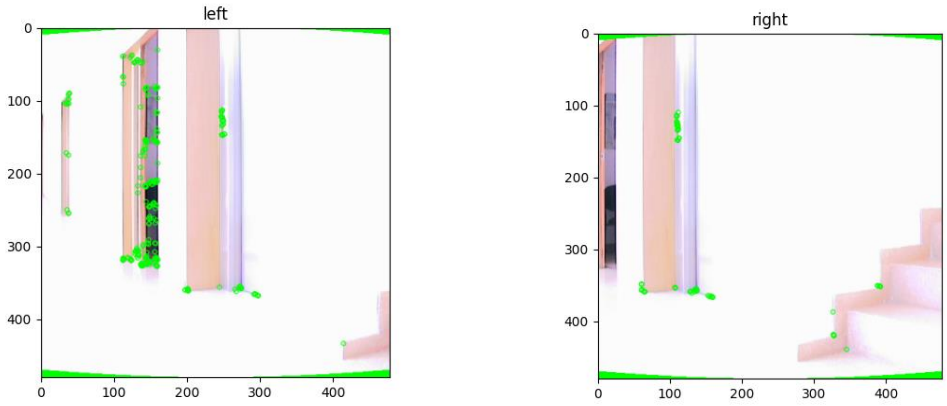

Figure 10: Pano2 – Feature Points of cyl_image27 and cyl_image28

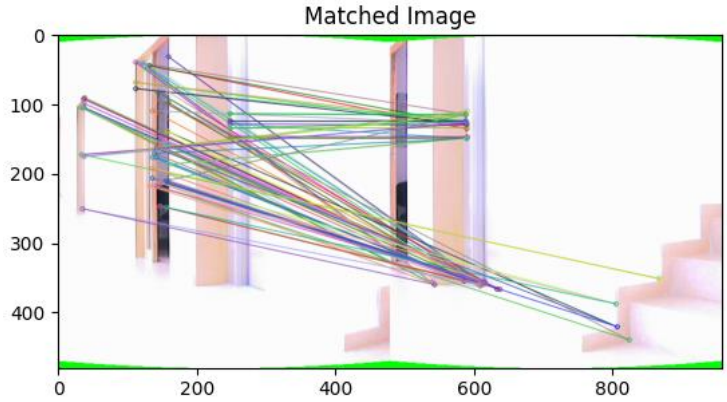

Figure 11: Pano2 - Feature Point Matching of cyl_image27 and cyl_image28

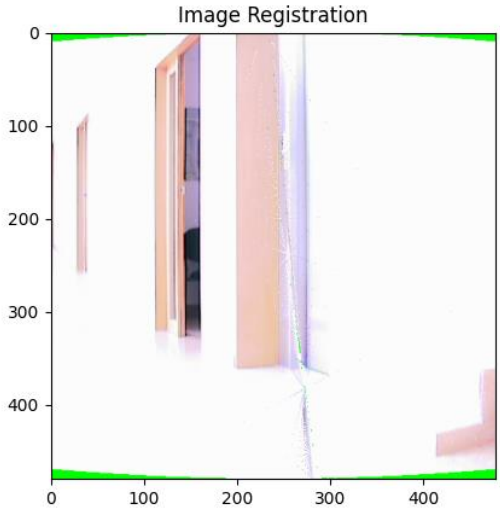

Figure 12: Pano2 – Subimages4 of cyl_image27 and cyl_image28

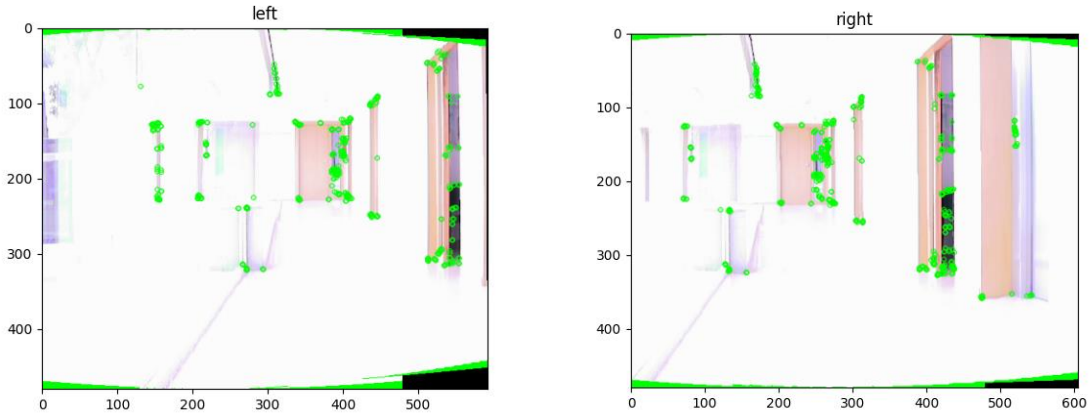

Figure 13: Pano2 – Feature Points of Subimages1 and Subimages2

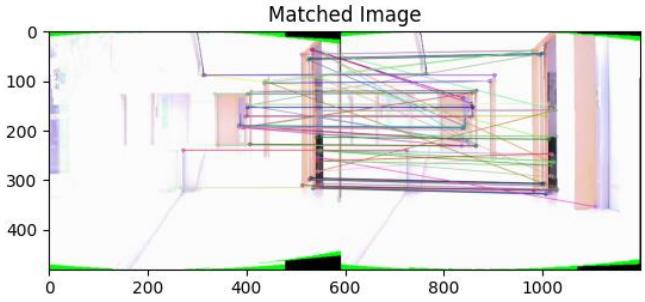

Figure 14: Pano2 - Feature Point Matching of Subimages1 and Subimages2

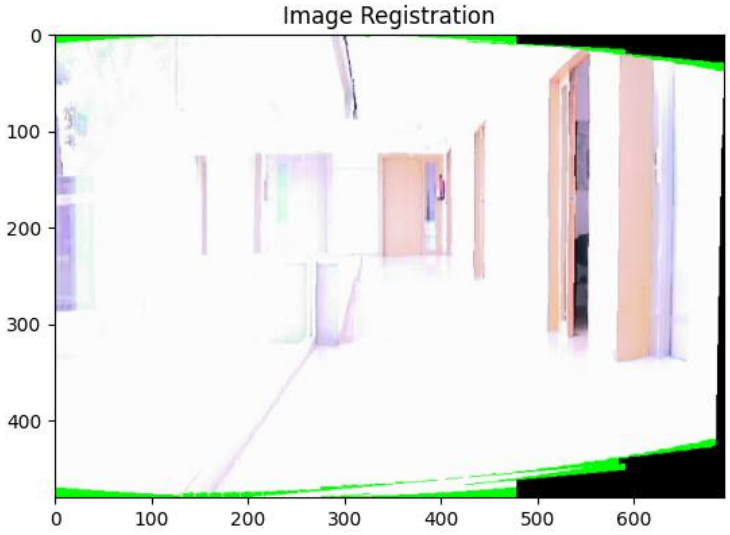

Figure 15: Pano2 – Subimages-1 of Subimages1 and Subimages2

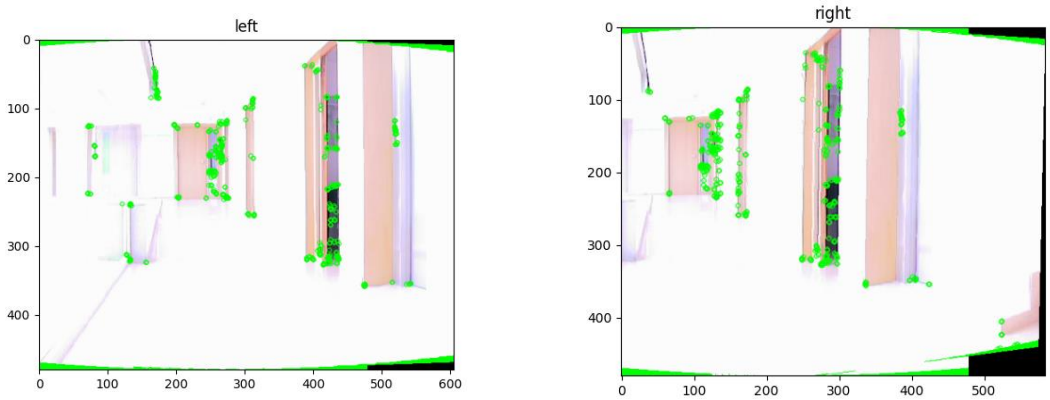

Figure 16: Pano2 – Feature Points of Subimages2 and Subimages3

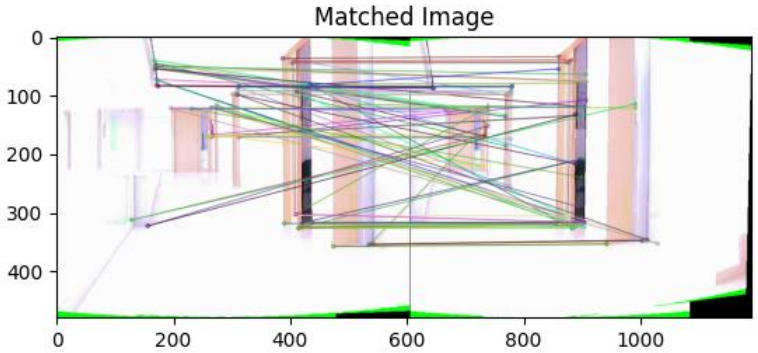

Figure 17: Pano2 - Feature Point Matching of Subimages2 and Subimages3

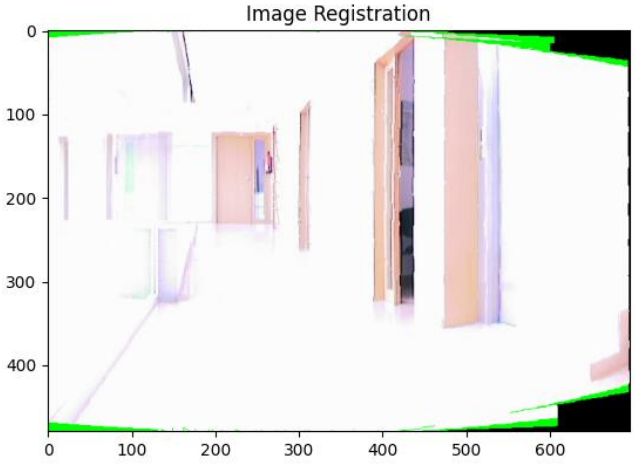

Figure 18: Pano2 – Subimages-2 of Subimages2 and Subimages3

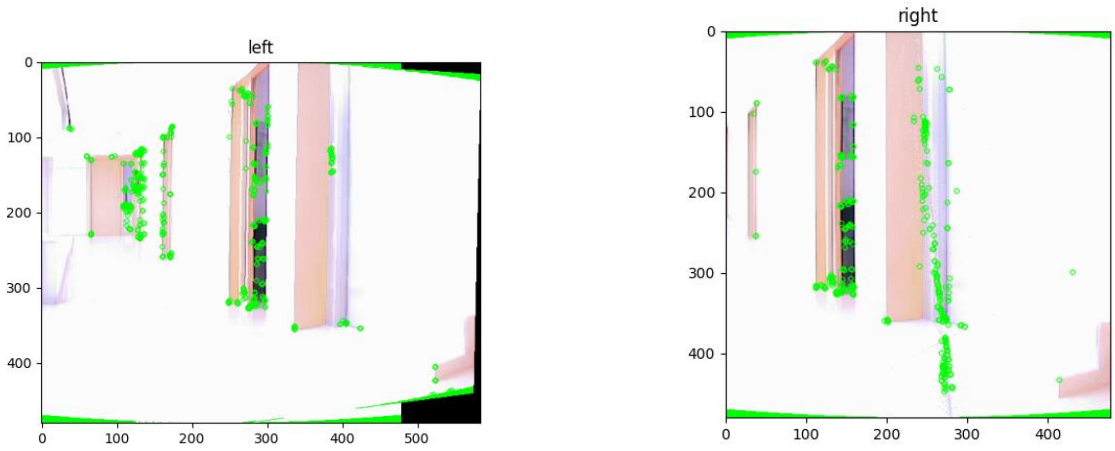

Figure 19: Pano2 – Feature Points of Subimages3 and Subimages4

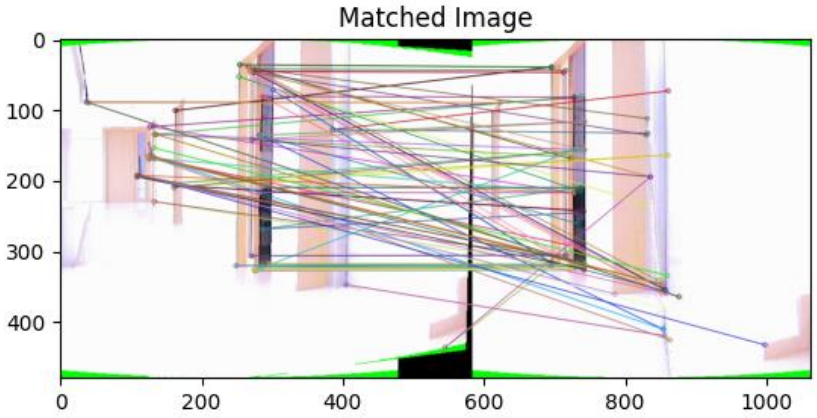

Figure 20: Pano2 - Feature Point Matching of Subimages3 and Subimages4

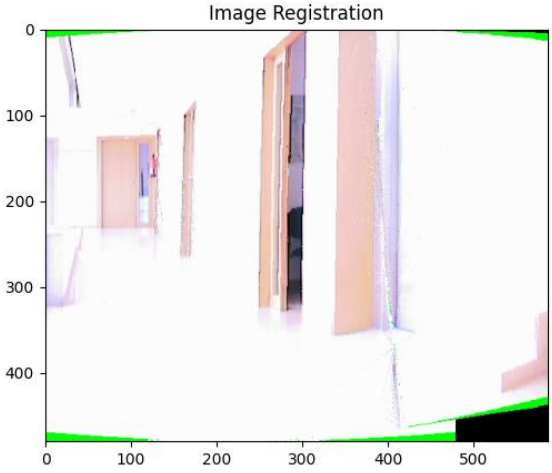

Figure 21: Pano2 – Subimages-3 of Subimages3 and Subimages4

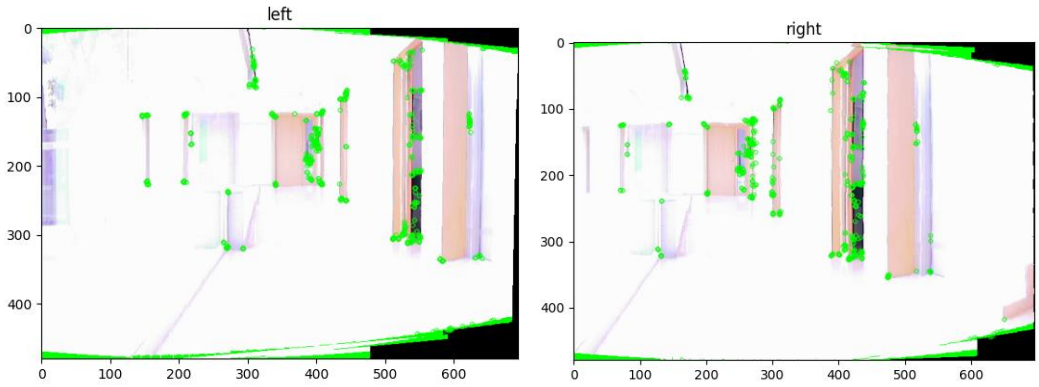

Figure 22: Pano2 – Feature Points of Subimages-1 and Subimages-2

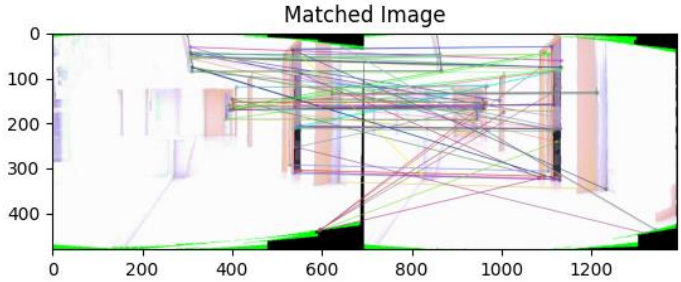

Figure 23: Pano2 - Feature Point Matching of Subimages-1 and Subimages-2

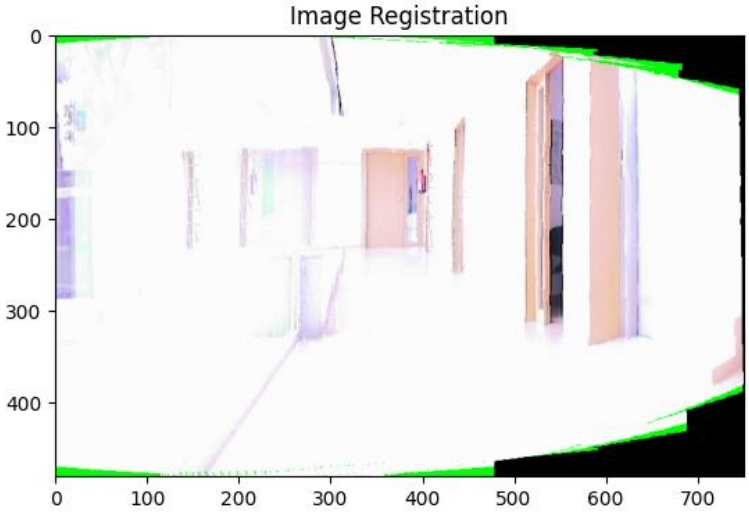

Figure 24: Pano2 – Subimages-1.1 of Subimages-1 and Subimages-2

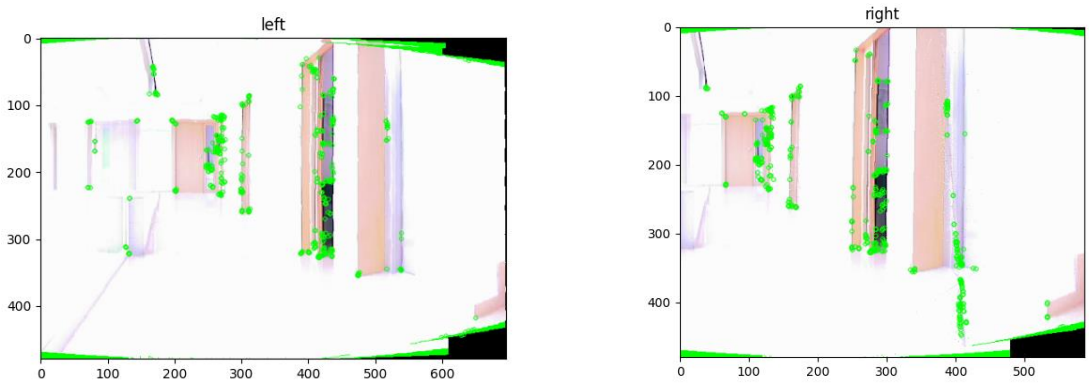

Figure 25: Pano2 – Feature Points of Subimages-2 and Subimages-3

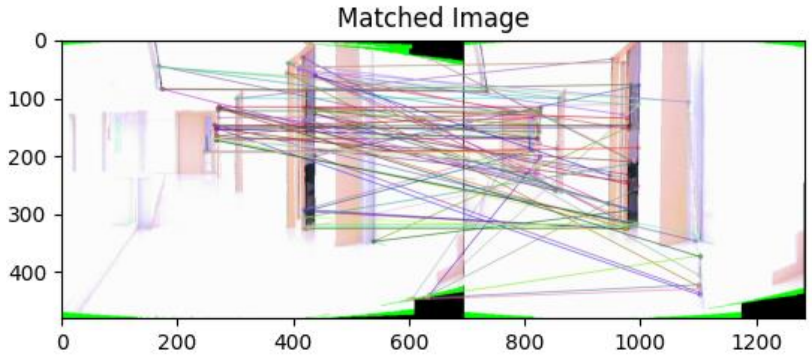

Figure 26: Pano2 - Feature Point Matching of Subimages-2 and Subimages-3

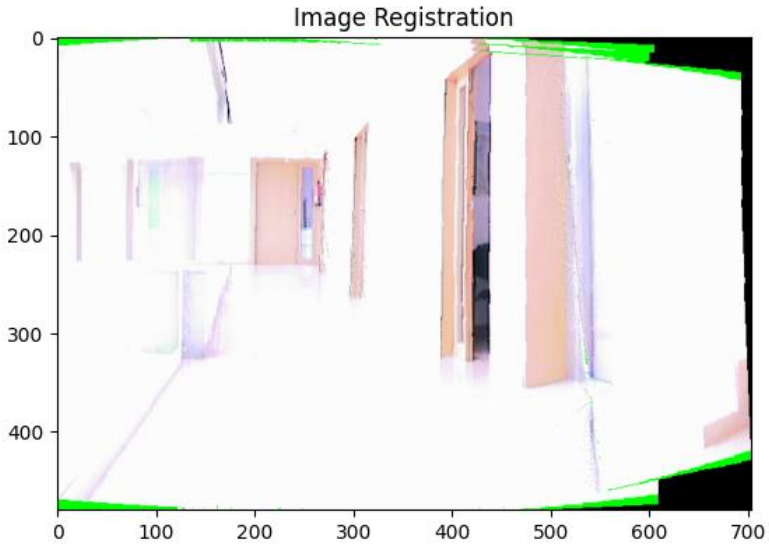

Figure 27: Pano2 – Subimages-1.2 of Subimages-2 and Subimages-3

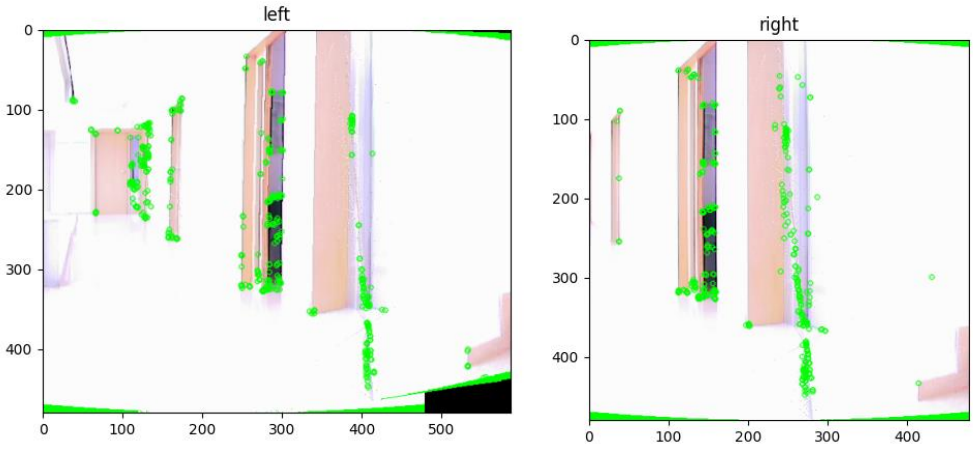

Figure 28: Pano2 – Feature Points of Subimages-1.1 and Subimages-1.2

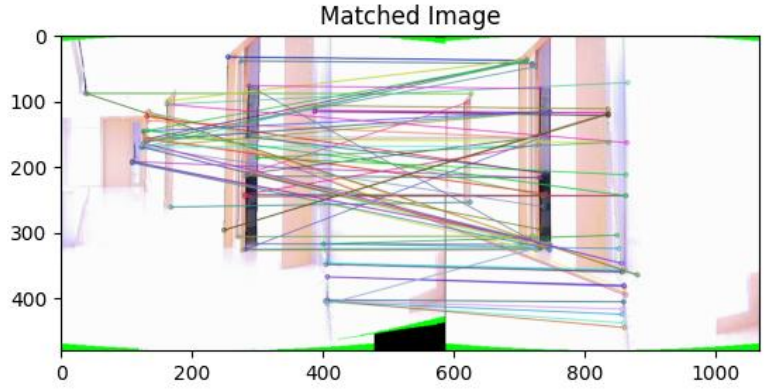

Figure 29: Pano2 - Feature Point Matching of Subimages-1.1 and Subimages-1.2

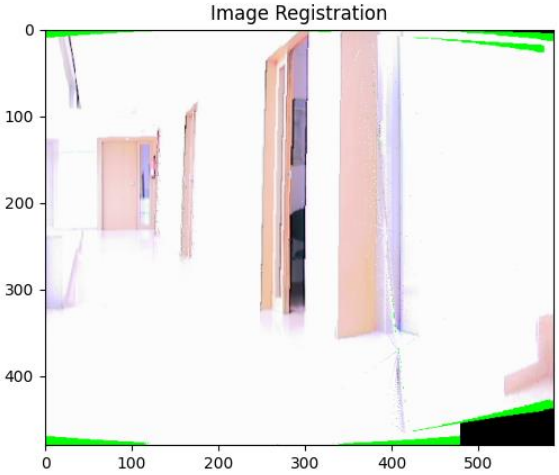

Figure 30: Pano2 – Subimages-1.1.1 of Subimages-1.1 and Subimages-1.2

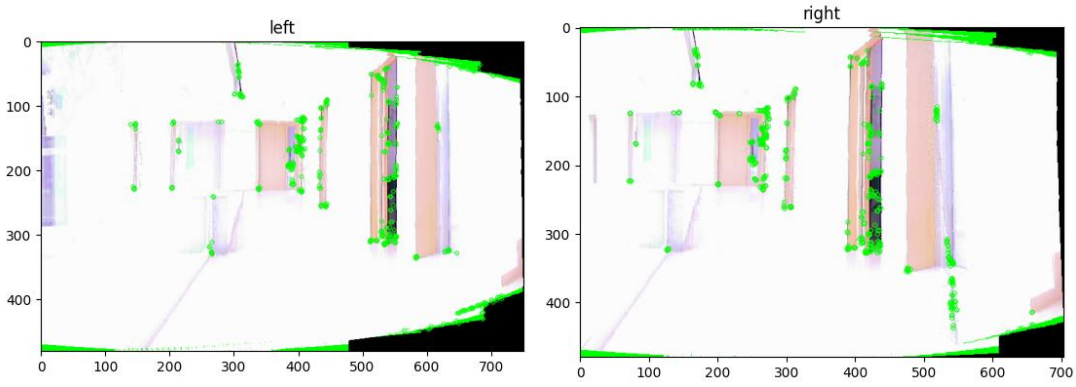

Figure 31: Pano2 – Feature Points of Subimages-1.2 and Subimages-1.3

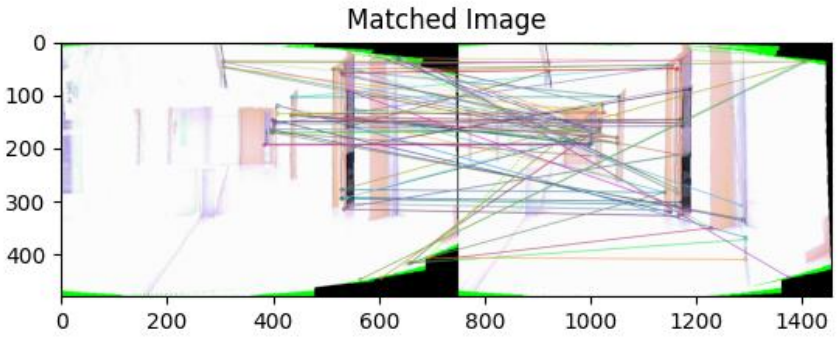

Figure 32: Pano2 - Feature Point Matching of Subimages-1.2 and Subimages-1.3

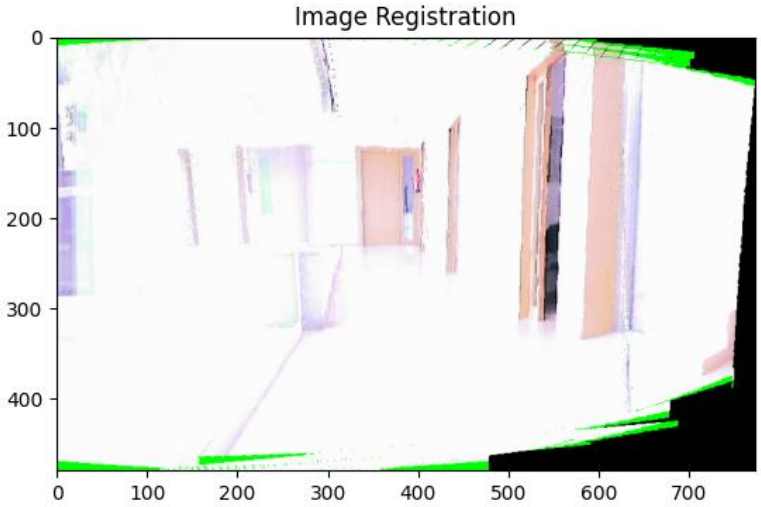

Figure 33: Pano2 – Subimages1.1.2 of Subimages-1.2 and Subimages-1.3

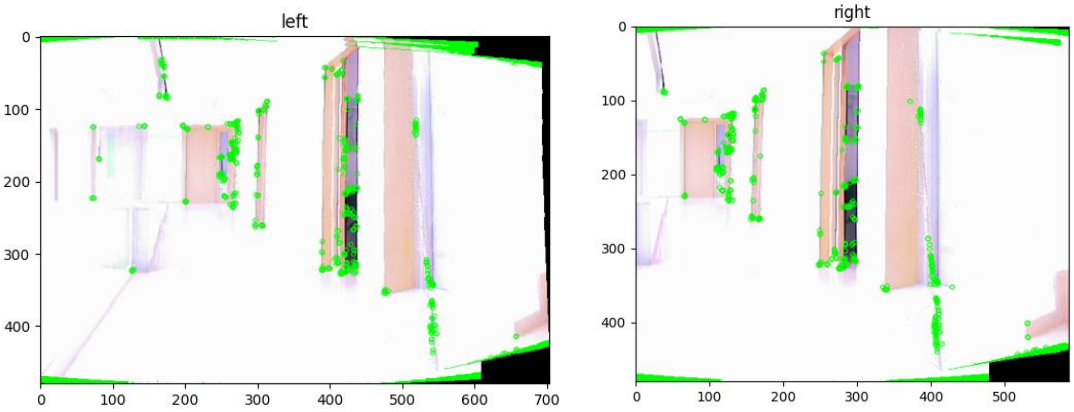

Figure 34: Pano2 – Feature Points of Subimages-1.3 and Subimages-1.4

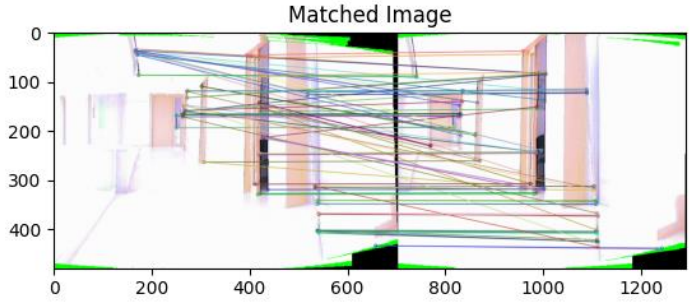

Figure 35: Pano2 - Feature Point Matching of Subimages-1.3 and Subimages-1.4

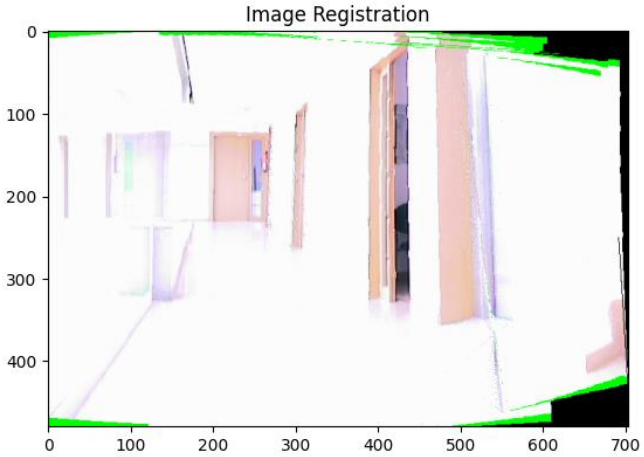

Figure 36: Pano2 – Subimages1.1.2 of Subimages-1.3 and Subimages-1.4

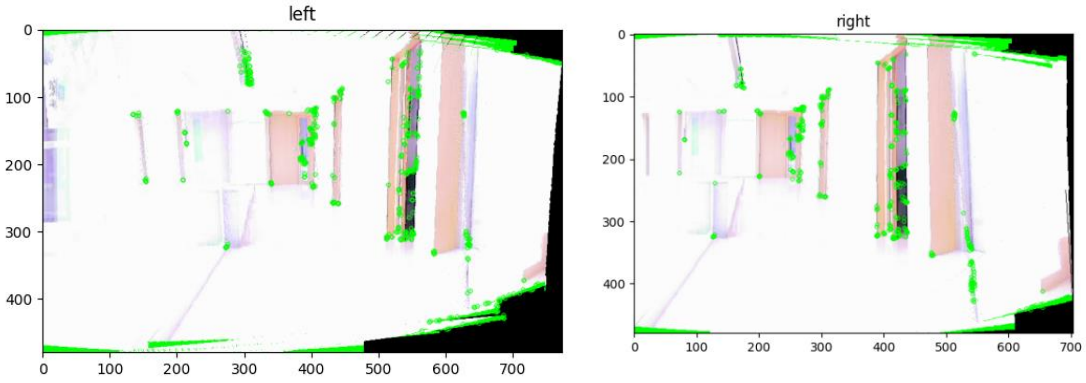

Figure 37: Pano2 – Feature Points of Subimages1.1.1 and Subimages1.1.2

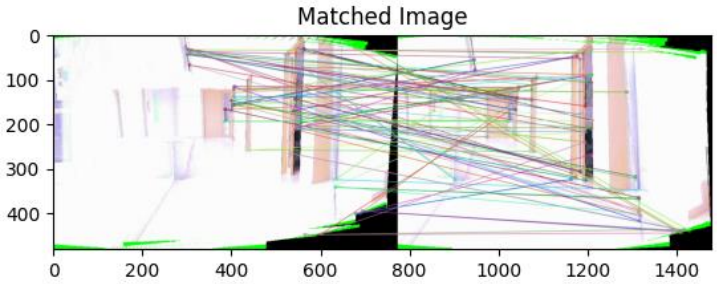

Figure 38: Pano2 - Feature Point Matching of Subimages1.1.1 and Subimages1.1.2

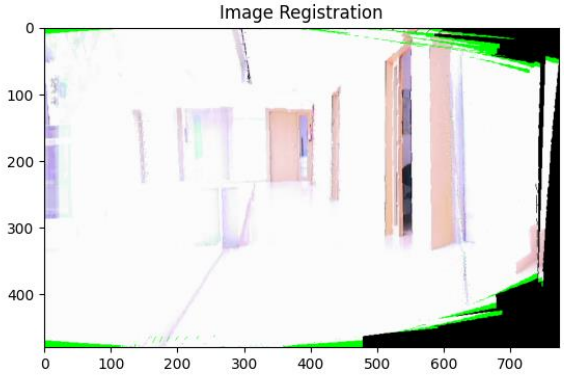

Figure 39: Pano2 – Subimages(Panorama) of Subimages1.1.1 and Subimages1.1.2

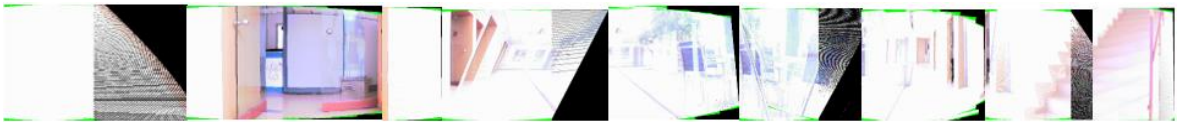

Figure 40: Pano1 – My Constructed Panorama Image

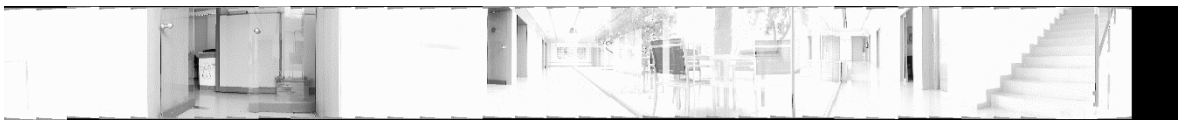

Figure 41: Pano1 - Ground Truth Panorama

3 Conclusion

Table 1: Runtime and Accuracy Comparison of the Description Methods (SIFT/SURF and ORB)

Comparison Result		
Description Methods	Runtime	Accuracy
SIFT/SURF	12.4 min	~ 62%
ORB	9.8 min	~ 45%

According to the results I have obtained, ORB is faster than SIFT, but not as successful as SIFT.

Although the functions I wrote without using a ready-made library did not give the desired picture, it gave subimages that can be considered successful. The shifts and smears may have resulted from areas that were overlooked in the merge and homography sections, since the dataset has somewhat abstract colors.

References

[1] https://www.analyticsvidhya.com/blog/2019/10/detailed-guide-powerful-sift-technique-image-matching-python

[2] https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

[3] https://en.wikipedia.org/wiki/Random_sample_consensus

[4] https://upcommons.upc.edu/bitstream/handle/2117/111334/cSalembier97.pdf