RBE/CS549 Computer Vision

Project 1- AutoPano

Anagha Dangle - Mihir Kulkarni
ardangle@wpi.edu - mmkulkarni@wpi.edu
Using 2 Late days

***Abstract*—The assignment is divided into two phases:**
Phase 1: Traditional approach- In this phase of assignment, traditional approach to Panorama stitching is implemented. The detailed process for the same is explained further in the document.
Phase 2: Deep learning approach- Two approaches of supervised as well as unsupervised are used in the scope of the project. This part is used for finding the homography for further processing which is explained in detail further.

***Index Terms*—Adaptive Non-maximal Suppression, RANSAC, Homography matrix, Spatial Transformer Network, TensorDLT.**

I. Phase 1: Traditional approach

In this assignment, a simplified version of Panorama stitching was to be implemented. The basic task is to stitch multiple images to form a Panorama using warping and Homography technique. The following steps are applied on every individual two images and then combined for the multiple images in the set. This approach has six main basic steps:
1. Corner detection
2. Adaptive Non-Maximal Suppression
3. Extract the features i.e. Create a feature descriptor
4. Match the features from both the images
5. RANSAC for removing the outliers and getting the homography matrix
6. Warping, stitching and blending the generated images
Each step of the algorithm is detailed in following section.

A. Detect corners

The first step to panorama stitching is to find the strongest corners in the image. For this we have implemented cv2.goodFeaturesToTrack which is the part of OpenCV and an implementation of Shi-Tomashi method of identifying strongest corners. It also has the functionality of using Harris Corner detector is set True, can be used. A grayscale input image is given to the function, with other parameters being quality level and euclidean distance between the corners. It is observed by us that changing the quality of corners has a rastic effect on the further steps of matching features. The value was iteratively adjusted to be optimal for the set of the images given. However, given a new set of images this tuning would further be more cumbersome. Also the goodFeaturesToTrack already uses the ANMS in itself as seen in the paper by Shi-Tomashi so the step of doing ANMS is likely to not make much difference.

B. Adaptive Non-Maximal Suppression

The ANMS algorithm finds the N_best corners of the grayscale image. The euclidean distance between the points is used to find the best corner between the cluster of points and uniformly distribute them. As seen from the figure the corners are evenly distributed. This will help to get good and even matches for further stitching. The best corners to be selected are decided by sorting the corners in descending order based on their calculated Euclidean distance. What we observed was that number of best corners has a non-linear effect i.e. increasing or decreasing the number of corners does not necessarily have the same effect on the matches.

Fig. 1: Detected corners (Train set 1): Iteration 2

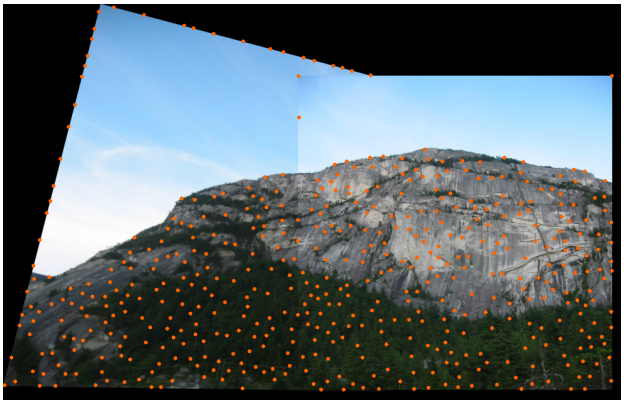

Fig. 2: Detected corners (Train set 2): Iteration 2

C. Feature descriptor

The first step we do here is to pad the input image with the patch size defined. This is done to avoid boundary and indexing conflicts. Later, the image is divided into patches of 400 size around a selected corner on which a Gaussian blur is also applied. This 40x40 patch is sub-sampled to get a 8x8 which in turn is resized to 64x1 vector with encoded features. We also apply mean and standardize the vector to remove any bias.

D. Feature matching

Each corner point now has a 64 feature vector associated with it. In this step we find one-to one correspondence between the images. We find the SSD between the points in image 1 and image 2. Then after sorting, the ratio between the first and second lowest distance is calculated and if it is below the threshold we provided, we accept the matched pair or else reject it. Instead of using the SSD, other distances can also be explored such as Euclidean and others.

Fig. 3: Match features (Train set 1)

E. RANSAC

This is one of the most important part of the entire process. Depending on the thresholds of RANSAC the number of pairs selected can vary. The matching pairs that we get from the feature matching have some outliers i.e. incorrect pairs as well. We need to filter out these pairs to get proper homography matrix and stitching.

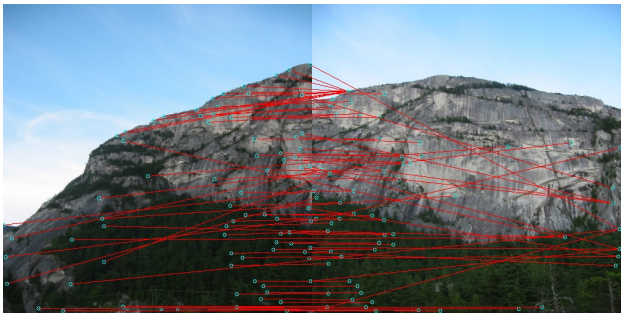

Fig. 4: Match features (Train set 2)

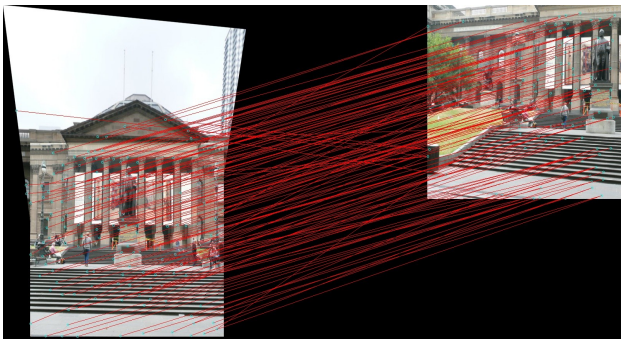

Fig. 5: Match features (Train set 1): Iteration 2

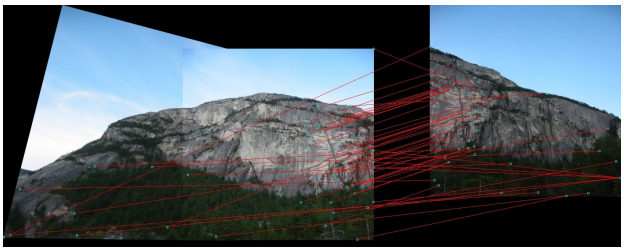

Fig. 6: Match features (Train set 1): Iteration 2

The RANSAC algorithm runs for 3000 iterations and selects 4 points randomly. These four points are selected from the feature vector1 and feature vector2. The exact homography between these two is calculated. Then we compute SSD between estimated points after transformation and the actual points in the image. The largest set of inliers is then selected through the algorithm after repeating 3000 times. The final homography is then calculated by using this set of inliers.

Fig. 7: RANSAC (Train set 1)

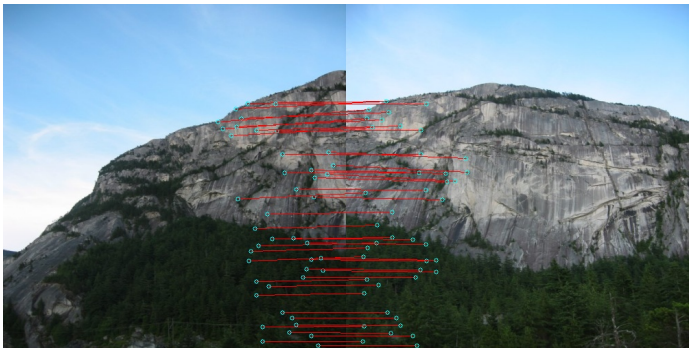

Fig. 8: RANSAC (Train set 2)

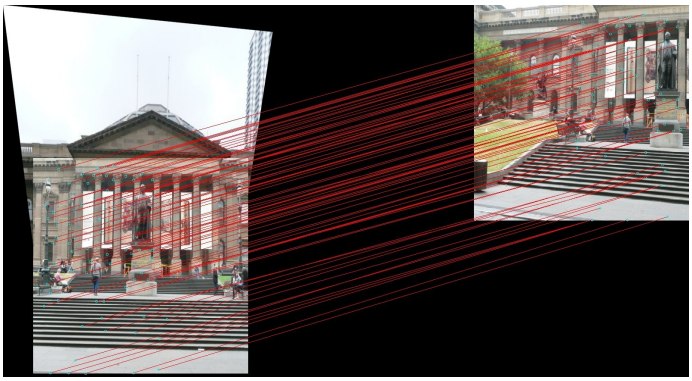

Fig. 9: RANSAC (Train set 1): Iteration 2

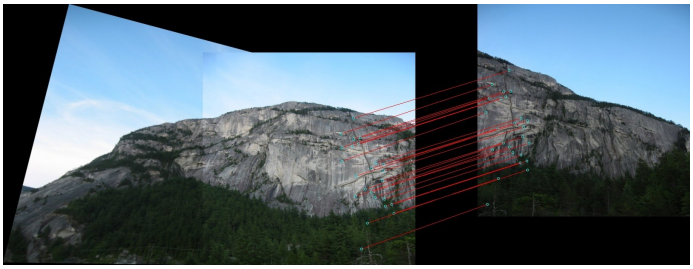

Fig. 10: RANSAC (Train set 2): Iteration 2

F. Stitch images

So for stitching the images we a use a very basic method for N multiple images. We compute homography for images sequentially, and keep on adding it on the newly created image. We warp the second image and add it on the first image. Then this image acts as the image holder and then we add third image on the image created in the previous iteration. This cycle goes on till the end of images is reached. The problem with this approach is that the panorama is not created for more than five images and also depending on the set of images this number varies. Tuning some of the parameters gives better results but it is still trial-and-error process. The problems with this approach is the original image gets warp to an extent where stitching on the image holder is not possible.

Another problem with this approach is it considers all the images sequentially so if the test set has random images and there are no features detected it throws an error. As the number of images start increasing the warped distortion increases and thus adding more images to the same place holder becomes a problem. The blending of the images is also not uniform as it is just overlapped on the previous image and not properly blended.

We studied different warping and blending techniques like cylindrical warping, perspective warping, Laplace blending, Poisson blending, etc. however we were not able to implement them due to time constraint. According to our observation, these techniques would perform far better than the crude algorithm we implemented.

Fig. 11: Panorama (Train set 1)

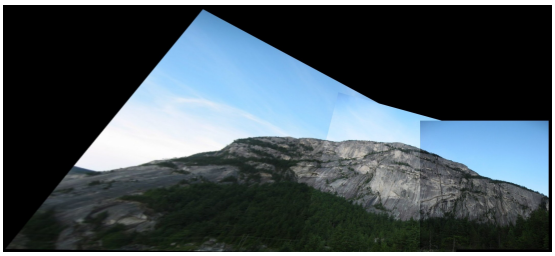

Fig. 12: Panorama (Train set 2)

G. Improvements that can be made and observations

We observed that changing the percent of inliers affects the RANSAC calcultaions and in-turn the stitching. So to improve the accuracy for stitching we increased the probability of outliers in every iteration. We also decreased the number of corners that are generated iteratively. When the images go beyond a threshold number of 3 this process is triggered. This ensures that the best matches are retained and the homography matrix calculated is proper.

We also wanted to test another approach where we separate the number of images in three parts and stitch them together at the end and also dynamically select matches from only specific region of the images. This would probably lessen the problems while stitching and blending. However, due to time constraint this approach was not implemented.

H. Results

Following are sample results for Test Set1 and Custom Set1. Other results are shown in the end of the report. The results from all the panoramas are shown in Fig (33).

Fig. 13: Detected corners (Test Set 1)

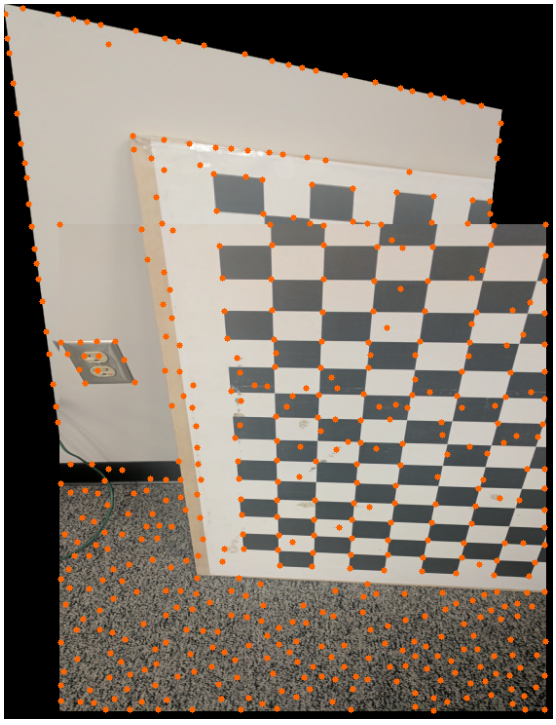

Fig. 14: Detected corners (Test Set 1): Iteration 2

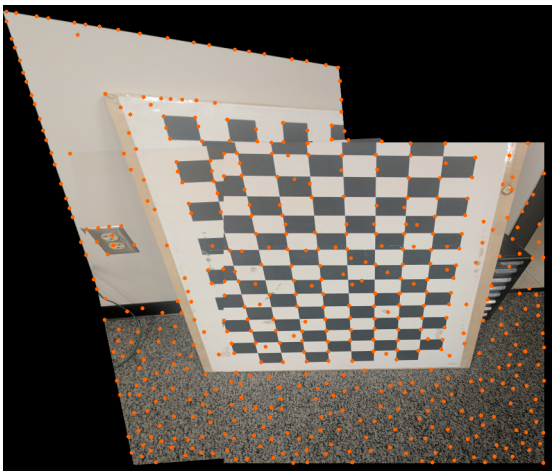

Fig. 15: Detected corners (Test Set 1): Iteration 3

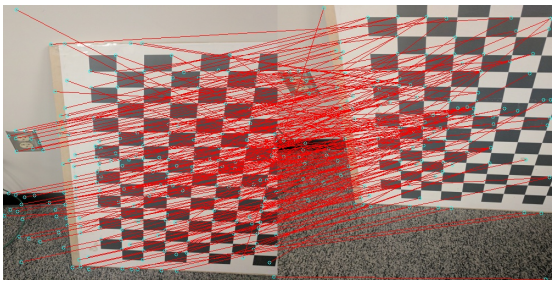

Fig. 16: Matched features (Test Set 1)

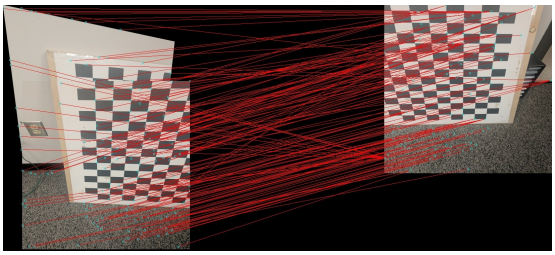

Fig. 17: Matched features (Test Set 1): Iteration 2

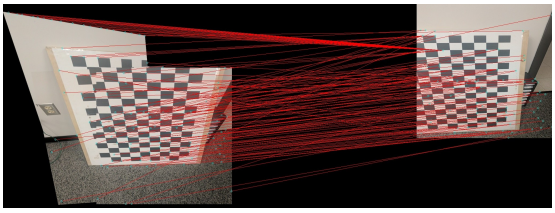

Fig. 18: Matched features (Test Set 1): Iteration 3

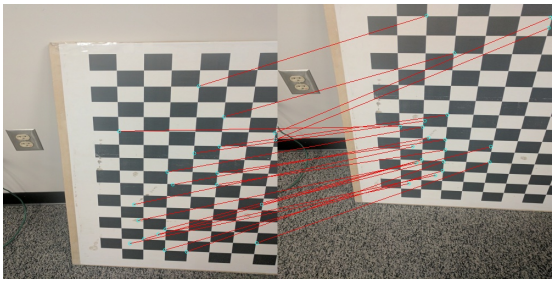

Fig. 19: RANSAC (Test Set 1)

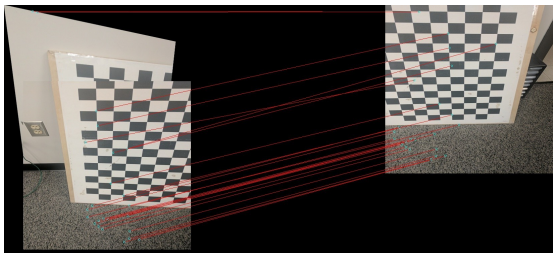

Fig. 20: RANSAC (Test Set 1): Iteration 2

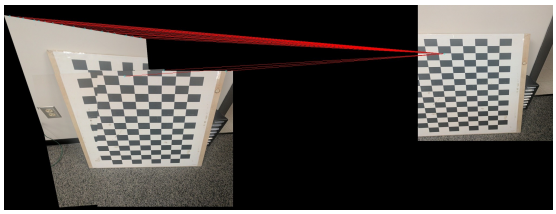

Fig. 21: RANSAC (Test Set 1): Iteration 3

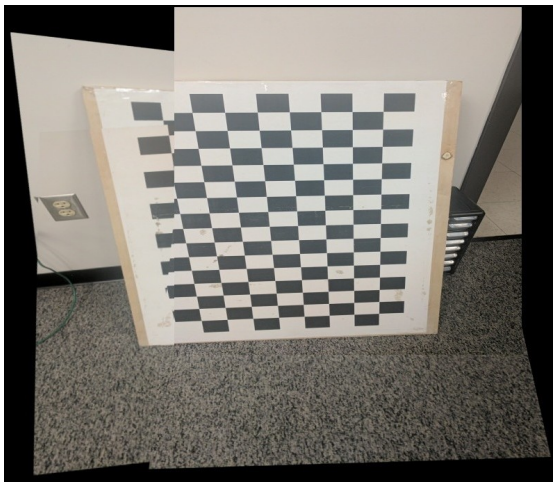

Fig. 22: Result (Test Set 1)

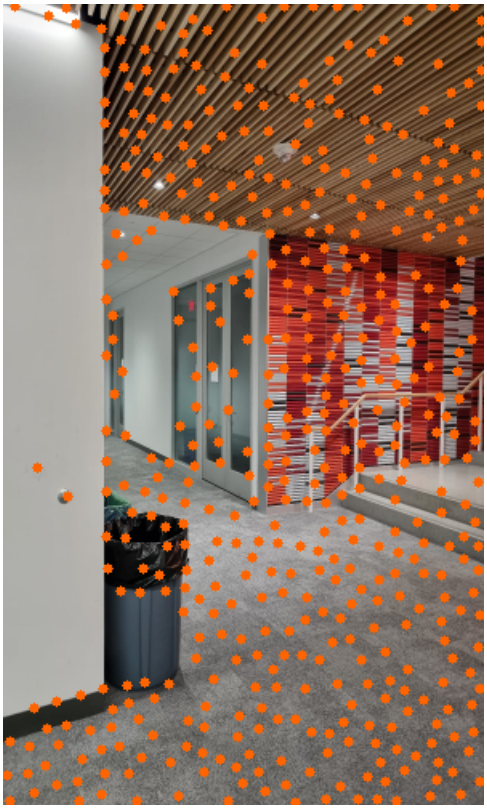

Fig. 23: Detected corners (Custom Set 1)

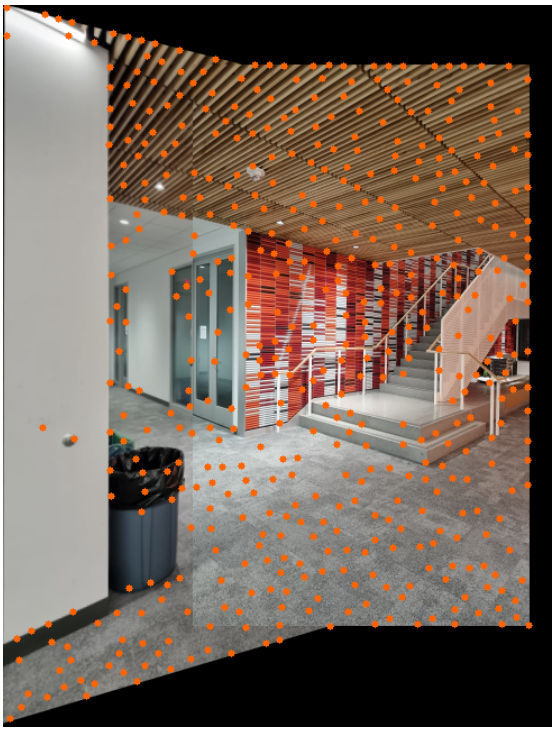

Fig. 24: Detected corners (Custom Set 1): Iteration 2

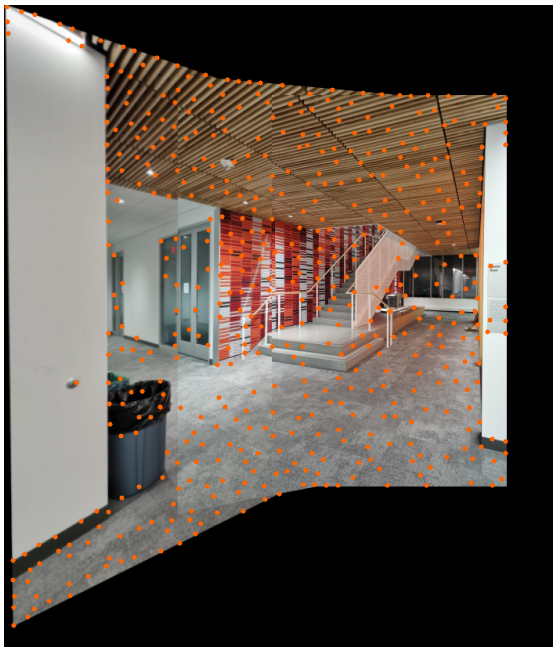

Fig. 25: Detected corners (Custom Set 1): Iteration 3

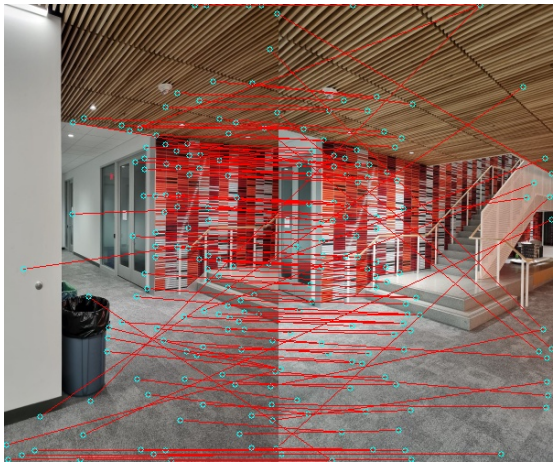

Fig. 26: Matched features (Custom Set 1)

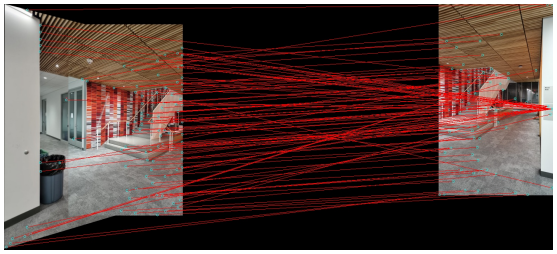

Fig. 27: Matched features (Custom Set 1): Iteration 2

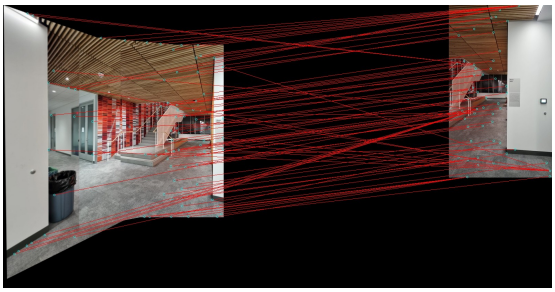

Fig. 28: Matched features (Custom Set 1): Iteration 3

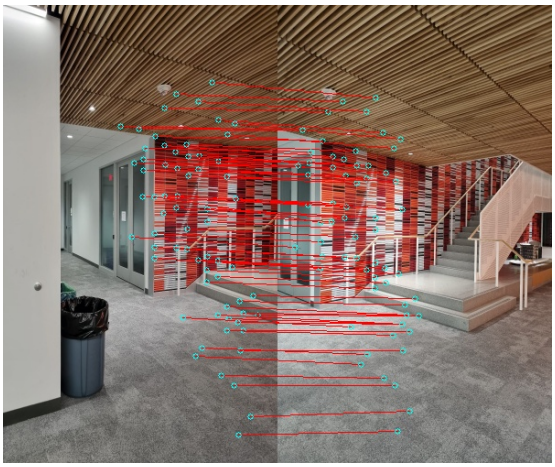

Fig. 29: RANSAC (Custom Set 1)

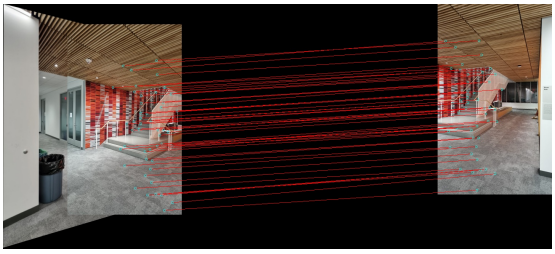

Fig. 30: RANSAC (Custom Set 1): Iteration 2

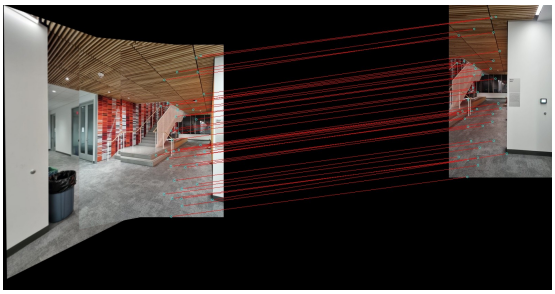

Fig. 31: RANSAC (Custom Set 1): Iteration 3

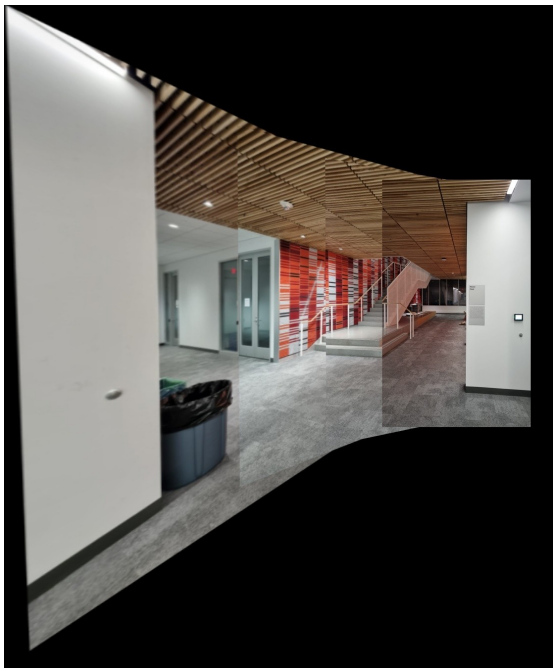

Fig. 32: Result (Custom Set 1)

Fig. 33: Results

II. Phase 2: Deep Learning Approach

In this section, the same task is to be implemented using 2 Deep Learning models, Supervised and Unsupervised. The dataset used for both the models is a small subset of the MSCOCO dataset containing 5000 images for training and 1000 images for validation. Both the approaches are explained in detail below.

A. Data generation

We first generate the dataset required for training. In brief, a random image from the dataset (of 5000 images) is selected. Then, a random patch from the images is selected, warped into a random patch, and a perspective transform (H matrix) between the original patch and the warped patch is calculated. Using this matrix, and the corner points of Patch A, we generate the corner points of Patch B. We do this for 7500 random images. Finally, the original image, corner points of Patch A and B, and the homography matrix for all 7500 images are stored into a pickle file.

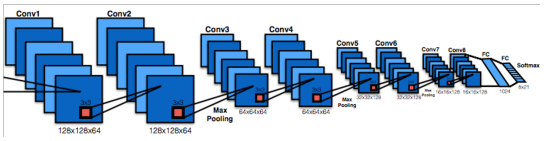

Fig. 34: Model Architecture

B. Supervised Model

We have implemented the same model architecture that is mentioned in their paper [1]. This model consists of 8 convolution layers and 2 fully connected layers with a softmax calculated at the end. The input to the model is a stack of 2 grayscale images (PatchA and PatchB) of size 128 x 128 each. Whereas, the output of the model is a vector of size 8 x 1 i.e. the H4pt indicating the difference of 4 corners of the 2 patches as shown in figure x. The architecture of this model[1] is shown in figure 34.The Loss function used here is the standard L2 loss in which, the output of the model (H4pt) is compared with the ground truth labels. Here, the labels are the difference of 4 corners of the 2 original patches. The model parameters used and the loss are stated in table I. The checkpoints for each epoch are obtained and saved to use for testing. The plot of training loss over epoch is shown in figure 35. ADAM Optimizer has been used with a learning rate of 1e-4.

C. Unsupervised Model

In the unsupervised approach, we utilized the same CNN architecture as the supervised model. The H4pt, i.e. the model output, is passed on to the TensorDLT function. The TensorDLT can be considered similar to the OpenCV function getPerspectiveTransform. It also takes the corners of the original Patch A as an input along with the H4pt. The output of TensorDLT is the 3 x 3 Homography matrix which is then given as an input to the Spatial Transformer Network

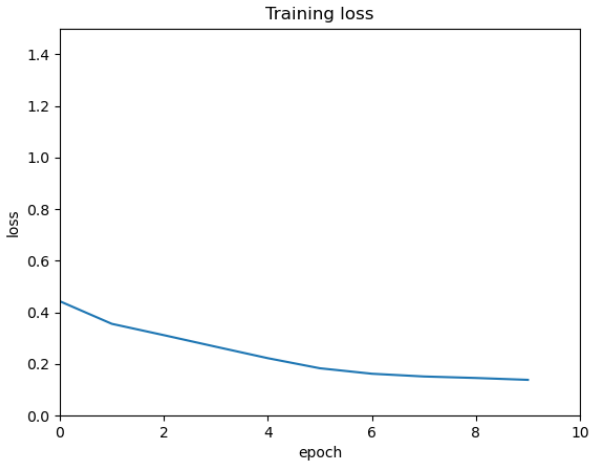

Fig. 35: Loss plot of Supervised

function. We have used the warp_perspective function from the Kornia library for this part. The main differentiating aspect of unsupervised model from supervised is the loss function. We do not utilize the labels to calculate the loss. Instead, we are using a photometric loss function that uses a general warping function to warp the Patch A and compare it with Patch B. The Optimizer used is ADAM with a learning rate of 1e-4 same as above. The loss plot over epochs is shown is figure 36.

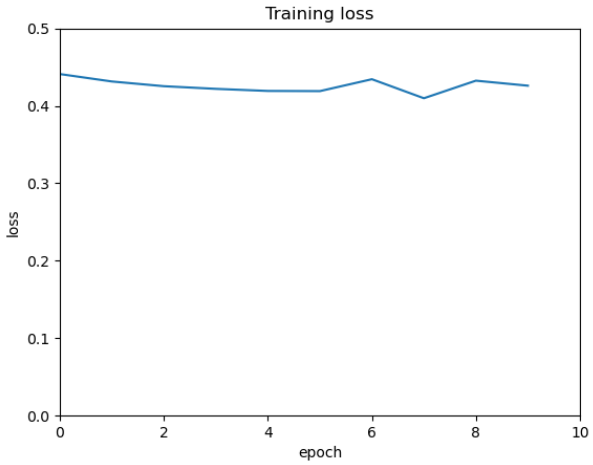

Fig. 36: Loss plot of Unsupervised

Model Type	Epochs	Batch Size	Learning Rate	Loss
Supervised	10	16	1e-4	0.0136
Unsupervised	10	64	1e-4	0.427

TABLE I: Results.

Fig. 37: Results of Supervised Model

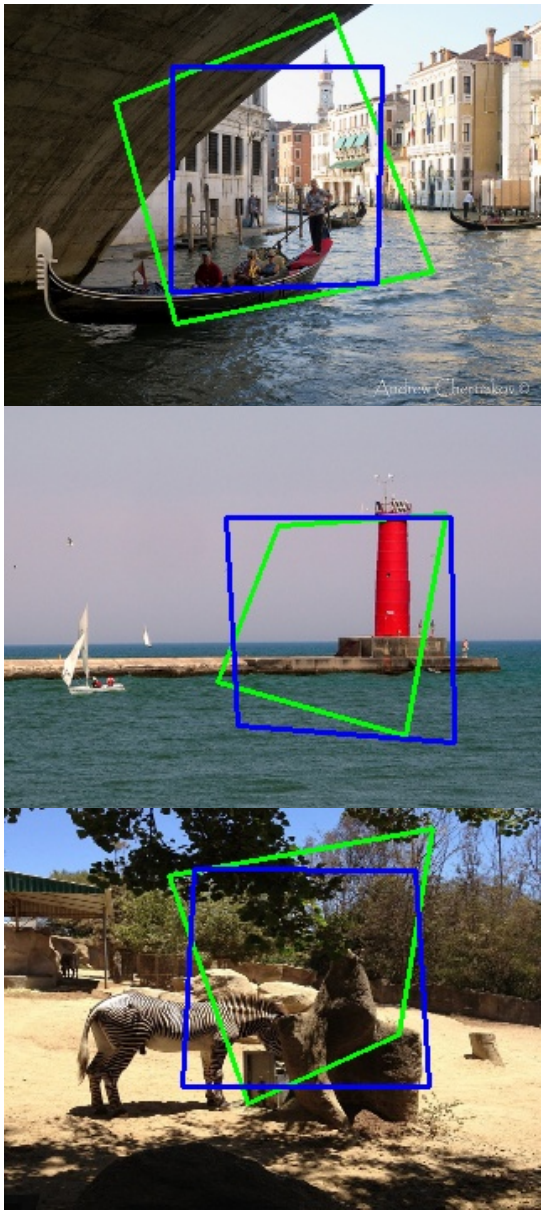

Fig. 38: Results of Unsupervised Model

D. Conclusion

1) Supervised: Our conclusion for the supervised part is that maybe due to generation of random patches used in the creation of the dataset, the model behaves differently for different images. The model seems to be overfitting after a certain point. We were not able to figure out the exact reason as the loss was not constant over the iterating epochs. The results for some images seem to be close while for others they are completely unrelated.

2) Unsupervised: The Loss more or less remains constant for every epoch due to some reason. This results in a bad accuracy of the model as we can see from the pictures in figure 38. The predicted homography is not close to the ground truth. We tried to change parameters, however the results remained same.

The final transformations i.e. the warping of the images was not completed, however once the model results are obtained that part can be also completed.

REFERENCES

[1] DeTone, Daniel, Tomasz Malisiewicz, and Andrew Rabinovich. "Deep image homography estimation." arXiv preprint arXiv:1606.03798 (2016).
[2] Nguyen, Ty, et al. "Unsupervised deep homography: A fast and robust homography estimation model." IEEE Robotics and Automation Letters 3.3 (2018): 2346-2353.
[3] https://pytorch.org/tutorials/intermediate/spatial_transformer_tutorial.html

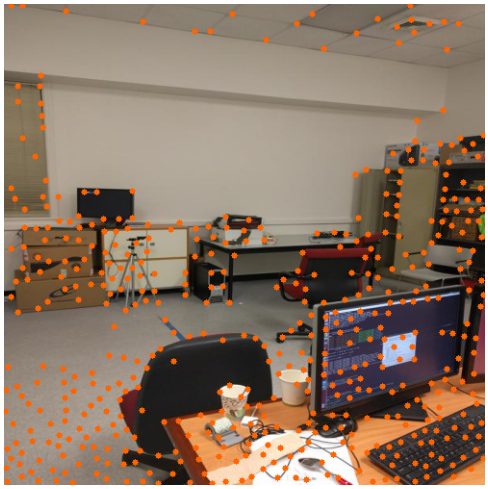

Fig. 39: Detected corners (Train Set 3)

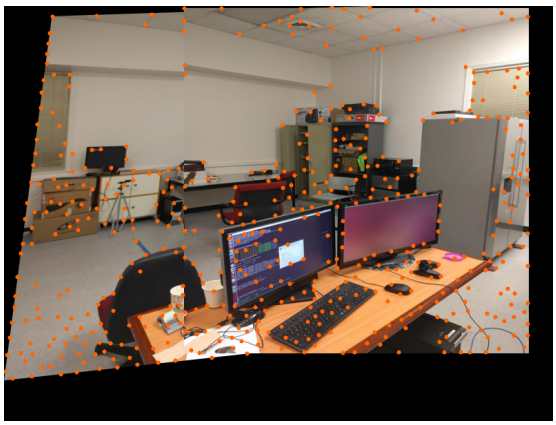

Fig. 40: Detected corners (Train Set 3): Iteration 2

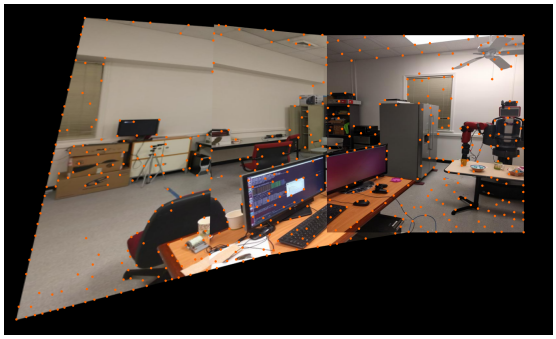

Fig. 41: Detected corners (Train Set 3): Iteration 3

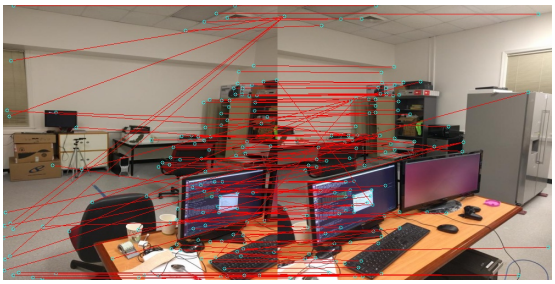

Fig. 42: Matched features (Train Set 3)

Fig. 43: Matched features (Train Set 3): Iteration 2

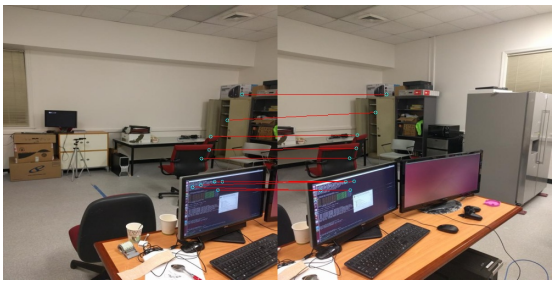

Fig. 44: RANSAC (Train Set 3)

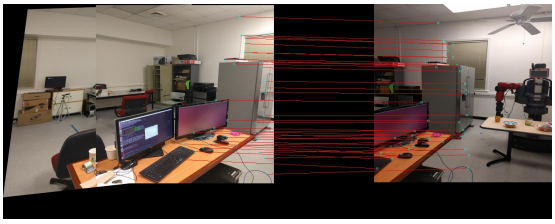

Fig. 45: RANSAC (Train Set 3): Iteration 2

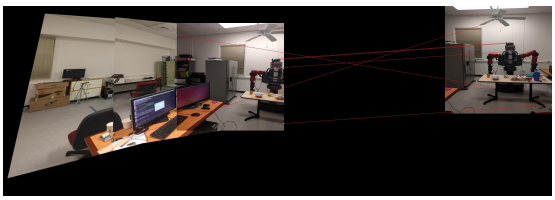

Fig. 46: RANSAC (Train Set 3): Iteration 3

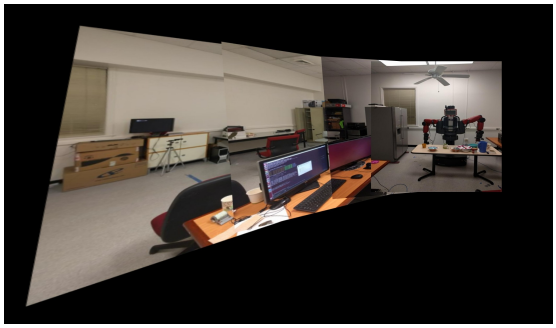

Fig. 47: Result (Train Set 3)

Fig. 48: Detected corners (Custom Set 2)

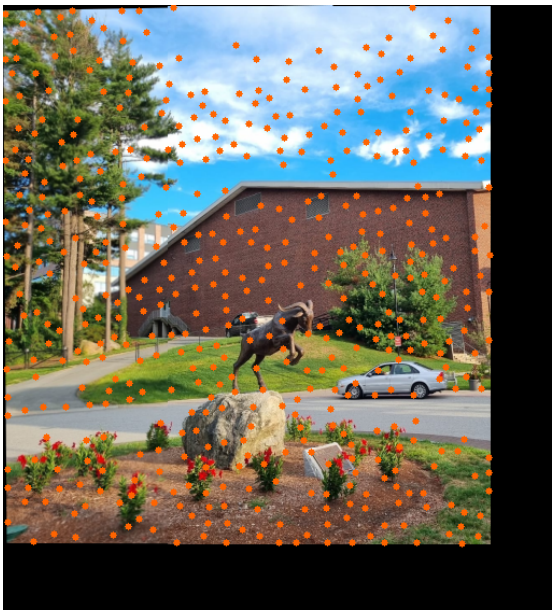

Fig. 49: Detected corners (Custom Set 2): Iteration 2

Fig. 50: Detected corners (Custom Set 2): Iteration 3

Fig. 51: Matched features (Custom Set 2)

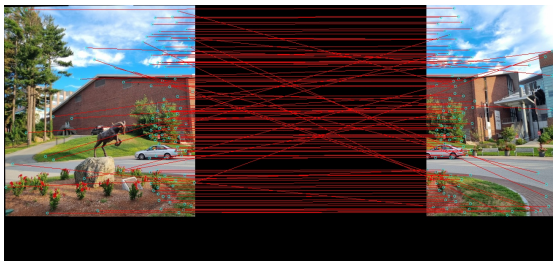

Fig. 52: Matched features (Custom Set 2): Iteration 2

Fig. 53: Matched features (Custom Set 2): Iteration 3

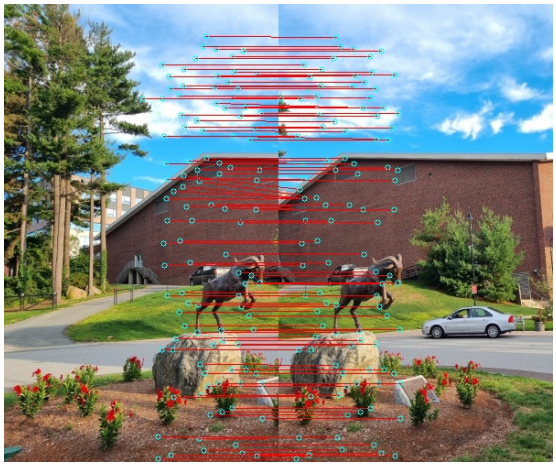

Fig. 54: RANSAC (Custom Set 2)

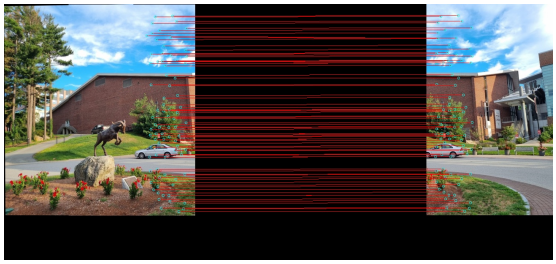

Fig. 55: RANSAC (Custom Set 2): Iteration 2

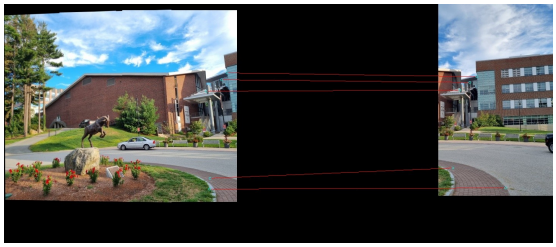

Fig. 56: RANSAC (Custom Set 2): Iteration 3

Fig. 57: Result (Custom Set 2)

Fig. 58: Detected corners (Test Set 3)

Fig. 59: Detected corners (Test Set 3): Iteration 2

Fig. 60: Matched features (Test Set 3)

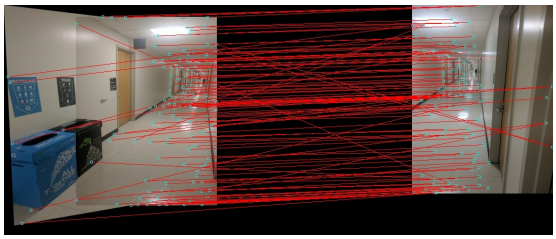

Fig. 61: Matched features (Test Set 3): Iteration 2

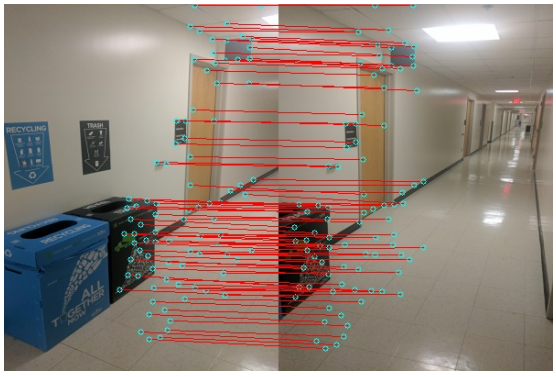

Fig. 62: RANSAC (Test Set 3)

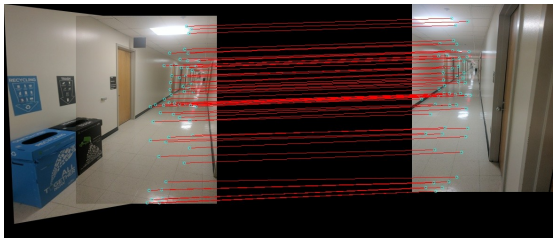

Fig. 63: RANSAC (Test Set 3): Iteration 2

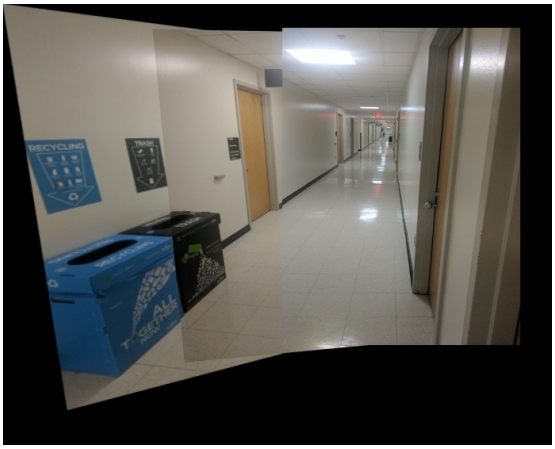

Fig. 64: Result (Test Set 3)

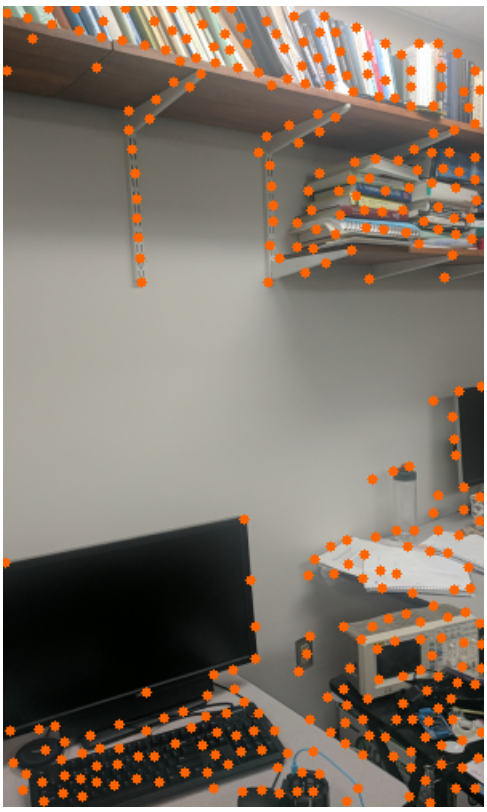

Fig. 65: Detected corners (Test Set 2)

Fig. 66: Detected corners (Test Set 2): Iteration 2

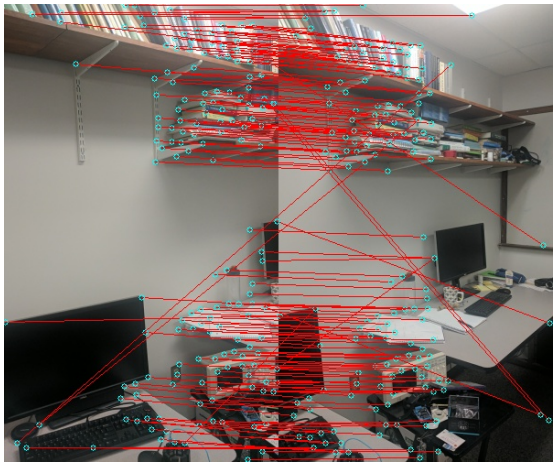

Fig. 67: Matched features (Test Set 2)

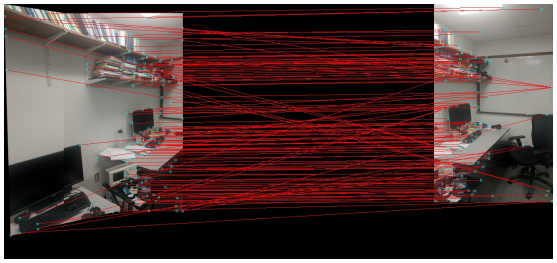

Fig. 68: Matched features (Test Set 2): Iteration 2

Fig. 69: RANSAC (Test Set 2)

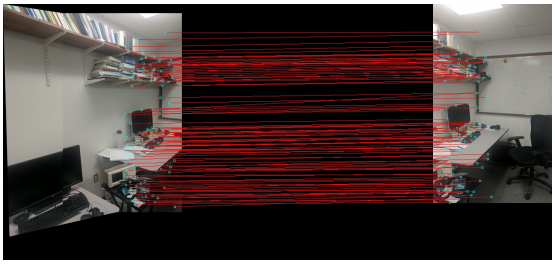

Fig. 70: RANSAC (Test Set 2): Iteration 2

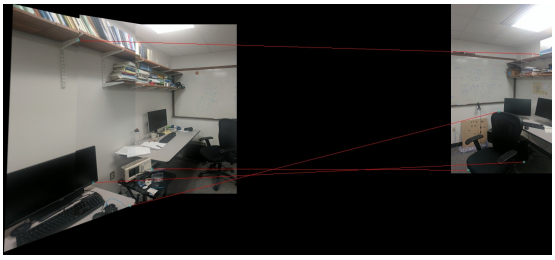

Fig. 71: RANSAC (Test Set 2): Iteration 3

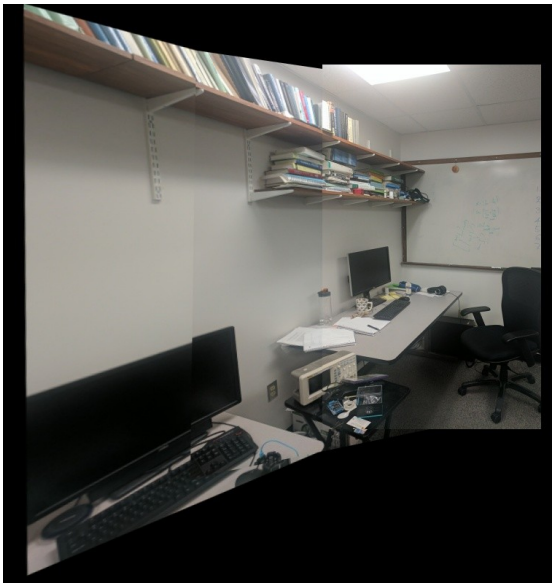

Fig. 72: Result (Test Set 2)