CMSC733: Project 1 - MyAutoPano

Khoi Viet Pham - Gnyana Teja Samudrala
Email: khoi@terpmail.umd.edu - sgteja@terpmail.umd.edu
Use 1 day late

I. Phase 1: Traditional Approach

In this section, we will present in details our approach to perform image stitching on multiple images. Here, we make the assumption that consecutive pair of images share at least an overlapping region. With this assumption, our algorithm is as follows: given a set of N images (indexed from $1 \rightarrow N$), we'll choose the image with index $\lfloor N/2 \rfloor$ as the anchor image. This means that we will perform perspective transformation on all remaining images so that we can transform them into the image plane of the anchor image. In the following subsections, we will illustrate figures of our intermediate results on each pair of consecutive images.

We mainly follow the steps listed in the instructions. The algorithm consists of 6 main steps: 1) corners detection 2) adaptive non-maximal suppression on detected corners 3) features extraction 4) features matching 5) homography estimation with RANSAC 6) warping and blending.

A. Corners Detection

The first step in the algorithm is to detect corners in all input images. Here, we use Harris Corner Detection algorithm to accomplish this task. All train/test/custom set images with detected corners are presented from figure 1 to figure 9.

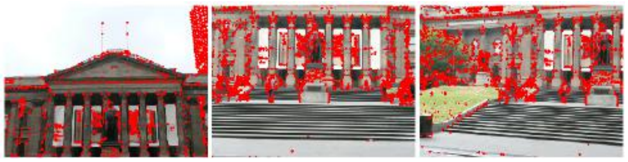
Fig. 1. Corners detected using Harris algorithm on Train/Set1 images.

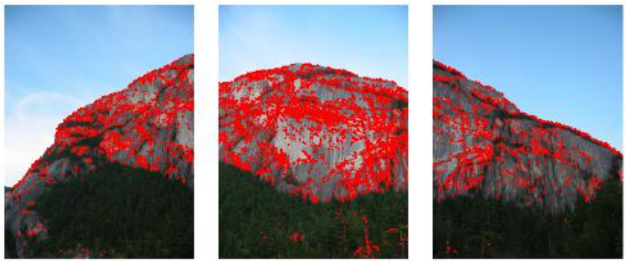
Fig. 2. Corners detected using Harris algorithm on Train/Set2 images.

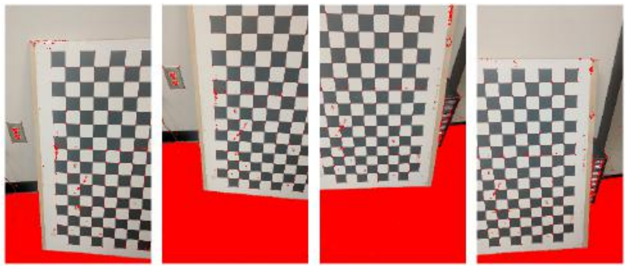
Fig. 4. Corners detected using Harris algorithm on Test/Set1 images.

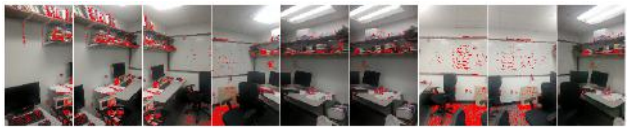
Fig. 5. Corners detected using Harris algorithm on Test/Set2 images.

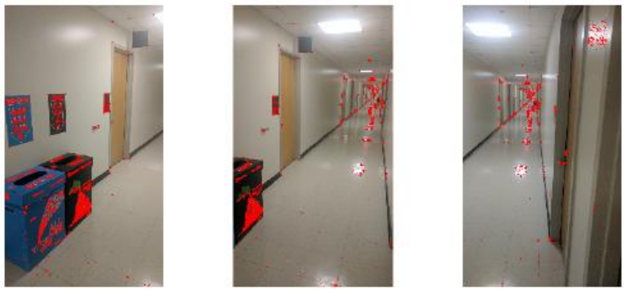
Fig. 6. Corners detected using Harris algorithm on Test/Set3 images.

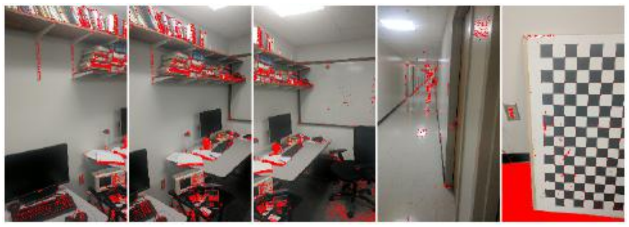
Fig. 7. Corners detected using Harris algorithm on Test/Set4 images.

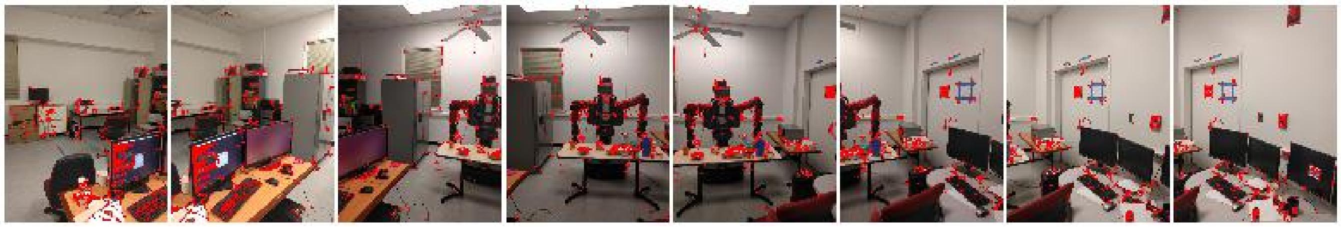
Fig. 3. Corners detected using Harris algorithm on Train/Set3 images.

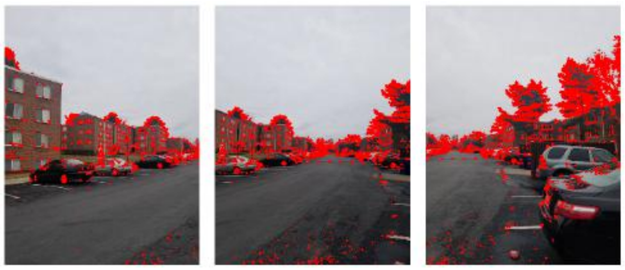
Fig. 8. Corners detected using Harris algorithm on CustomSet1 images.

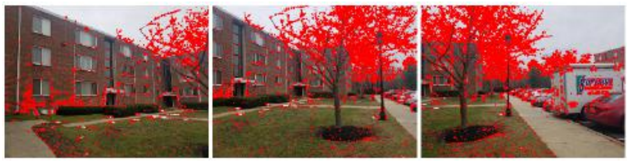
Fig. 9. Corners detected using Harris algorithm on CustomSet2 images.

B. Adaptive non-maximal suppression

By looking at the output of the previous step in figure 1, 2, 3, we can see that the number of detected corners is huge. There are a lot of redundant corners that we do not need to process at all. Therefore, in this step, we will apply adaptive non-maximal suppression (ANMS) in order to keep only those corners such that are equally distributed across the whole image.

It is worth noting that from the previous step, besides the detected corners, the Harris Corner Detection algorithm also produces a corner score map that describes how likely a pixel is a corner. Using this corner score map, ANMS will locate all local maximas (these are likely corners that we are interested) and try to separate them as far from each other as possible (so that these corners are equally distributed). In our code, we select the best 500 corners with ANMS.

The outputs of ANMS on all train/test/custom set images are presented from figure 10 to figure 18.

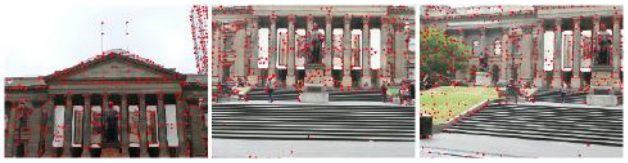
Fig. 10. Output of ANMS on Train/Set1 images.

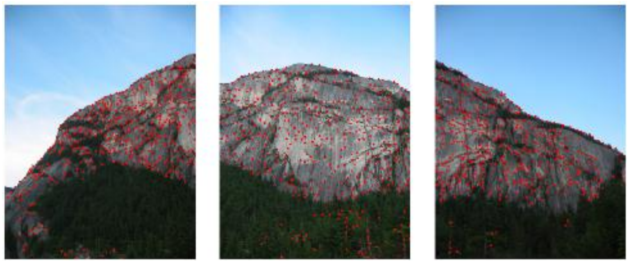
Fig. 11. Output of ANMS on Train/Set2 images.

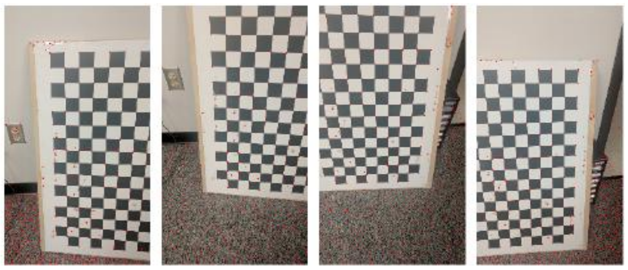
Fig. 13. Output of ANMS on Test/Set1 images.

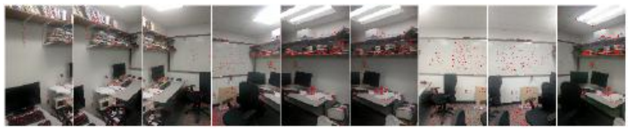
Fig. 14. Output of ANMS on Test/Set2 images.

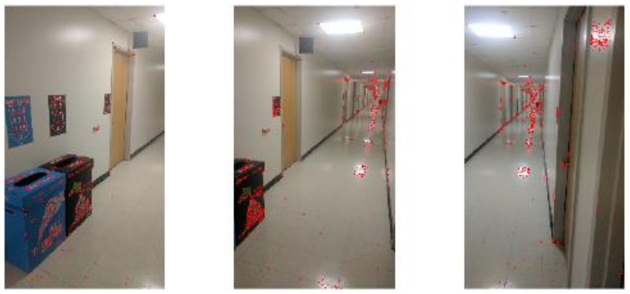
Fig. 15. Output of ANMS on Test/Set3 images.

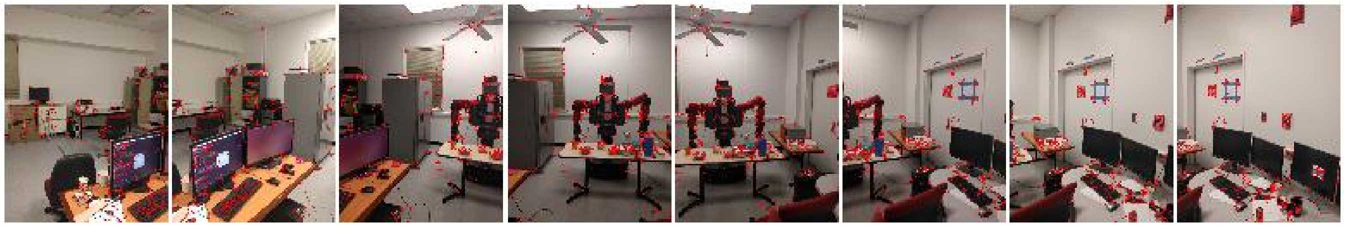
Fig. 12. Output of ANMS on Train/Set3 images.

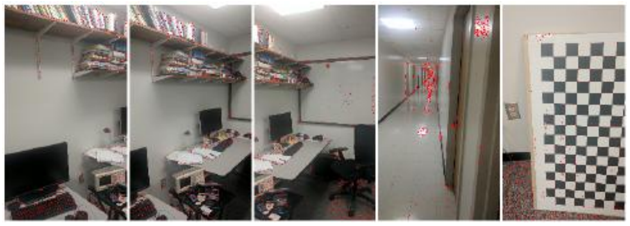
Fig. 16. Output of ANMS on Test/Set4 images.

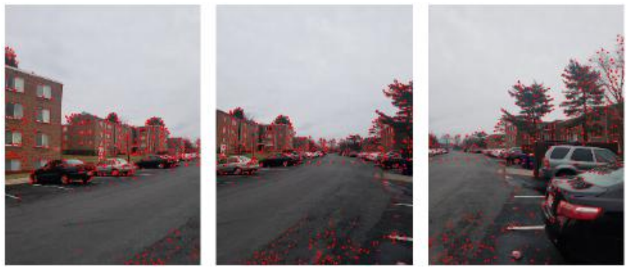
Fig. 17. Output of ANMS on CustomSet1 images.

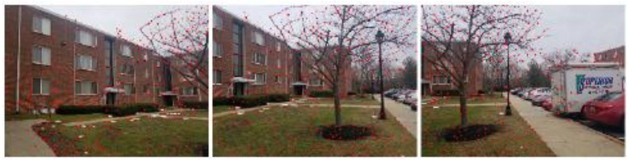
Fig. 18. Output of ANMS on CustomSet2 images.

C. Features Extraction

In order to stitch images together, we need to match the corners (keypoints) on this image with the corners (keypoints) in the other image. To perform matching, first, for each corner, we need to produce a feature vector that encodes all local information around it. We follow exactly the instruction to perform this task.

Because 500 corners in each image (result we got from ANMS) is too many to display in one single figure, we decide to randomly select only 100 corners among all input images and display their feature vectors.

These feature vectors on all train/test/custom set images are presented from figure 19 to figure 27.

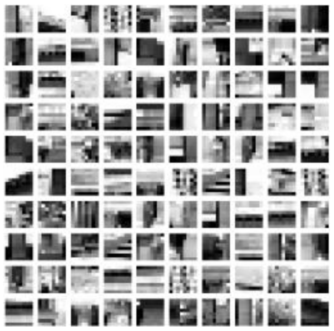
Fig. 19. 100 random feature vectors from Train/Set1 images.

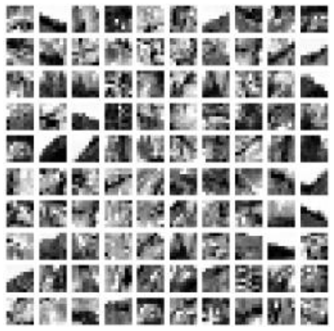
Fig. 20. 100 random feature vectors from Train/Set2 images.

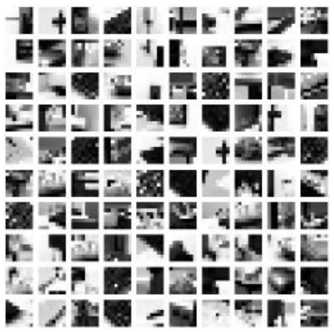
Fig. 21. 100 random feature vectors from Train/Set3 images.

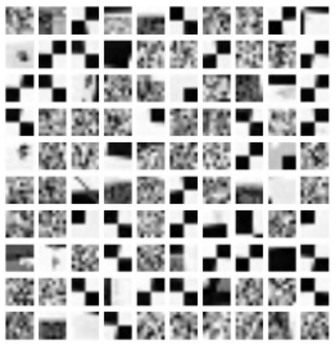
Fig. 22. 100 random feature vectors from Test/Set1 images.

Fig. 23. 100 random feature vectors from Test/Set2 images.

Fig. 24. 100 random feature vectors from Test/Set3 images.

Fig. 25. 100 random feature vectors from Test/Set4 images.

Fig. 26. 100 random feature vectors from CustomSet1 images.

Fig. 27. 100 random feature vectors from CustomSet2 images.

D. Features Matching

We are now able to match corners between two consecutive images. We also follow exactly the instruction for this task. We use Gaussian with kernel equals 3 in order to blur the 40×40 window around each corner. For each corner in image A, we look for its best and second best match in image B (in terms of L2 distance between the feature vectors). If the ratio between the best and second best match is lower than 0.70, we select the best match. Otherwise, we reject it (this might not be a good way to reject). Also after getting all the matching points, if the count is less than a certain threshold value (10 wroked well) they are not considered. By doing this we can avoid the images which are out of the sync in the panorama.

Matching results of each pair of consecutive images in the train/test/custom set are displayed from figure 28 to figure36.

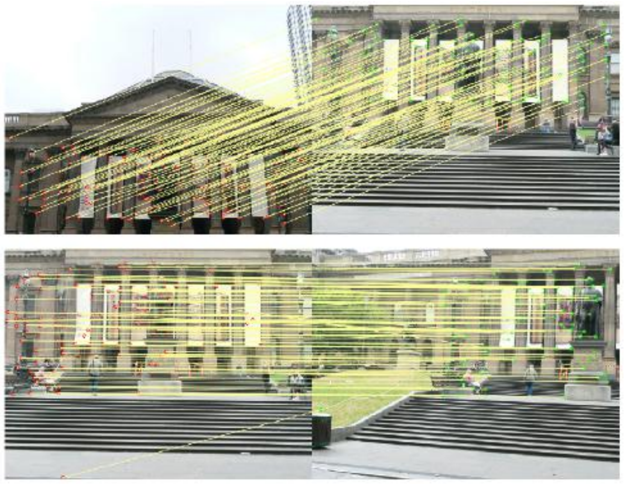

Fig. 28. Matching results on Train/Set1 images.

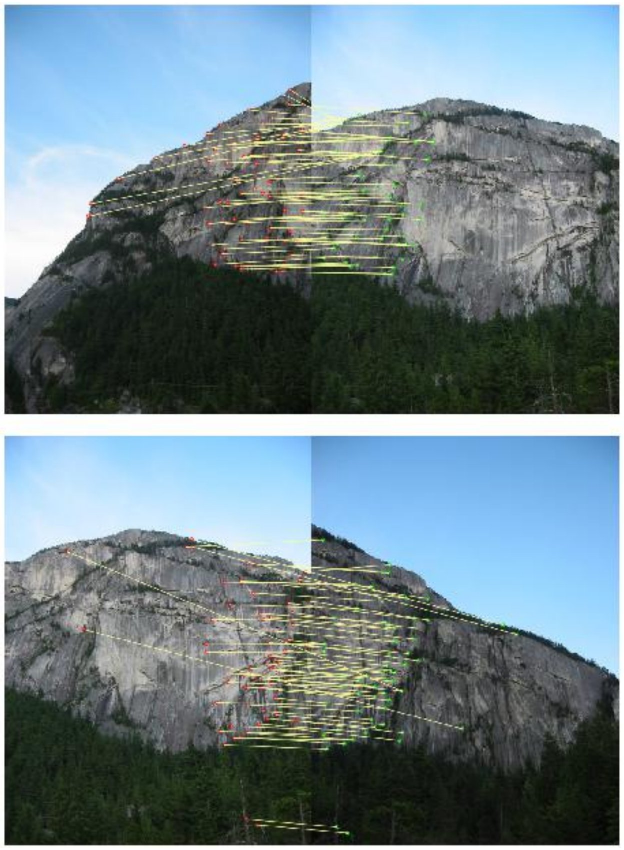

Fig. 29. Matching results on Train/Set2 images.

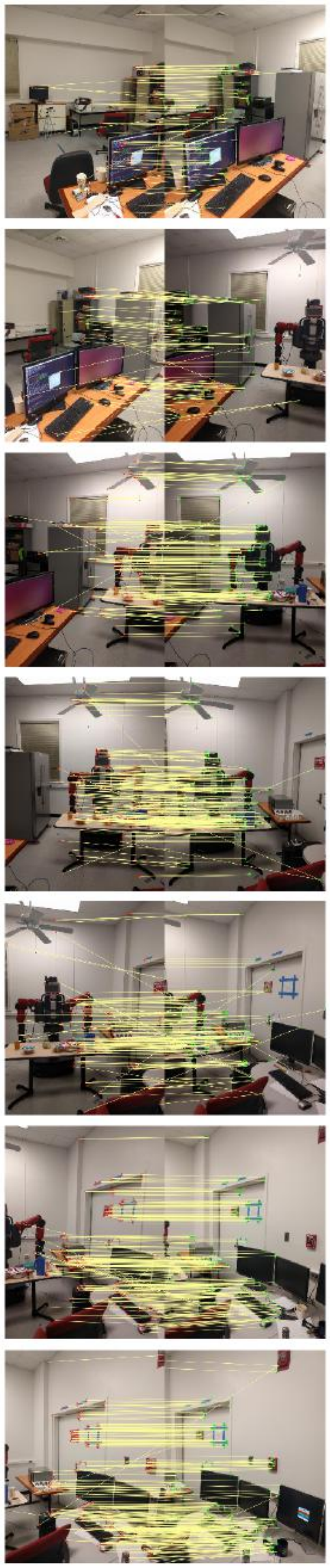

Fig. 30. Matching results on Train/Set3 images.

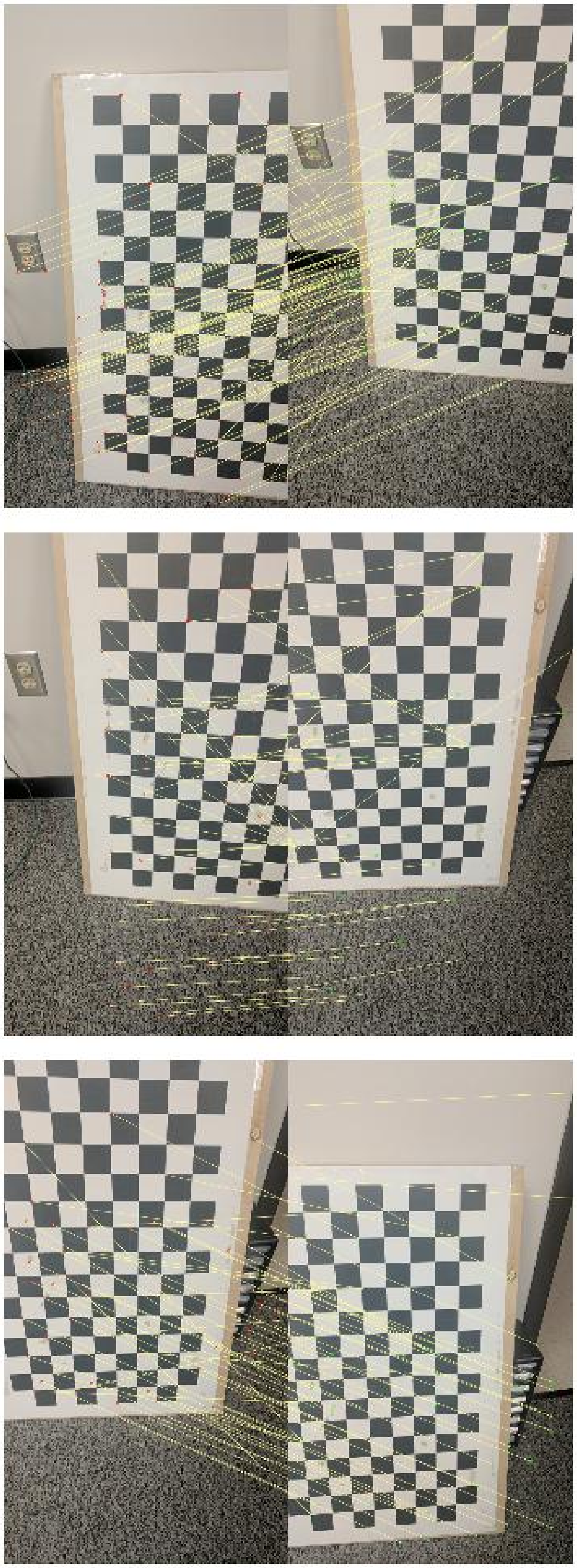
Fig. 31. Matching results on Test/Set1 images.

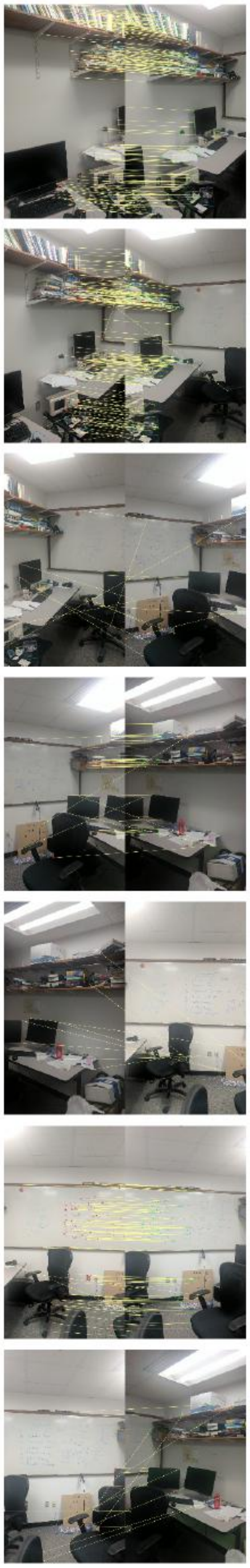
Fig. 32. Matching results on Test/Set2 images.

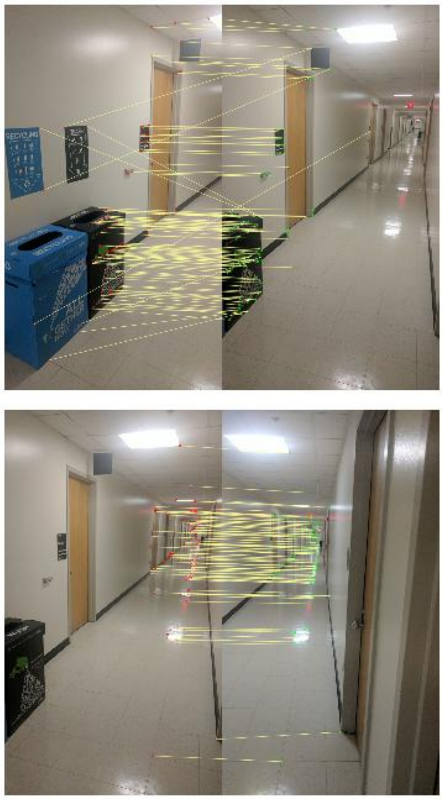

Fig. 33. Matching results on Test/Set3 images.

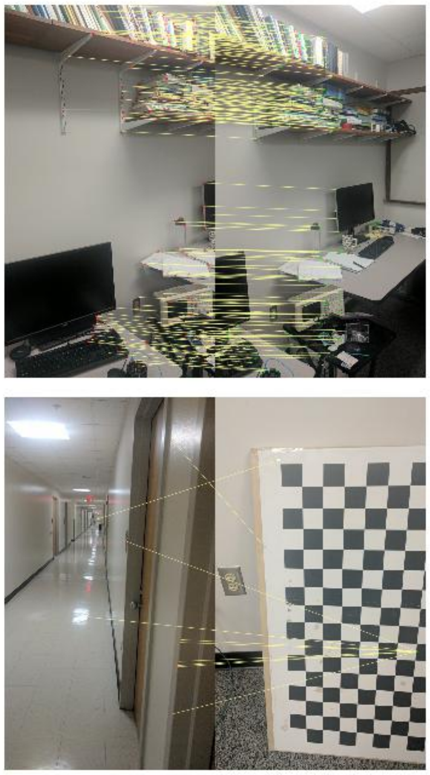

Fig. 34. Matching results on Test/Set4 images.

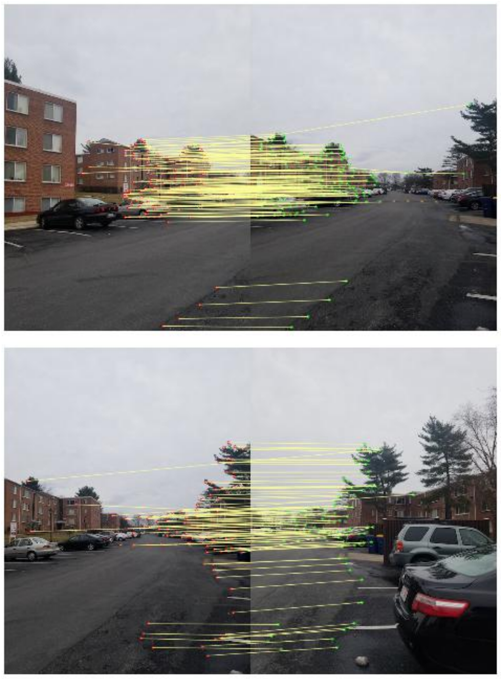

Fig. 35. Matching results on CustomSet1 images.

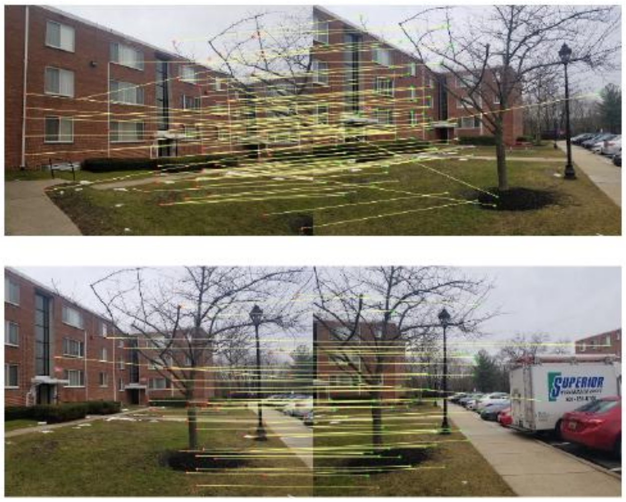

Fig. 36. Matching results on CustomSet2 images.

E. Homography Estimation with RANSAC

Looking at the output from the previous step, we can see that there are a lot of outliers (features matched together but are not correspondences). In this step, we can use Random Sample Consensus (RANSAC) to remove those outliers. After those outliers have been removed, we can compute the homography matrix from the inliers. Here, we also implement ourselves the code to compute the homography matrix instead of using *cv2.getPrespectiveTransform*.

The results after applying RANSAC on the train/test/custom set images are presented from figure 37 to figure 45.

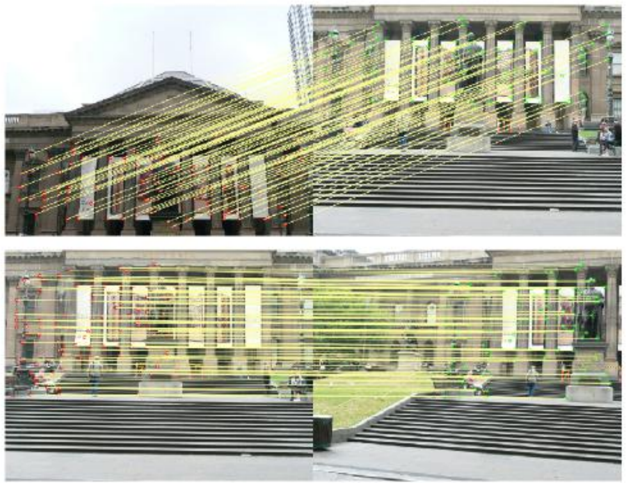
Fig. 37. RANSAC results on Train/Set1 images.

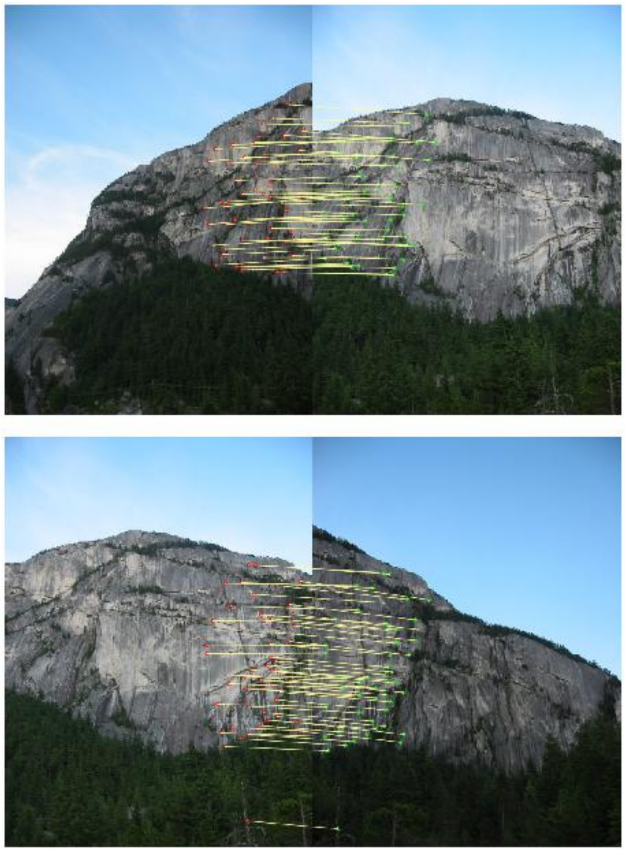
Fig. 38. RANSAC results on Train/Set2 images.

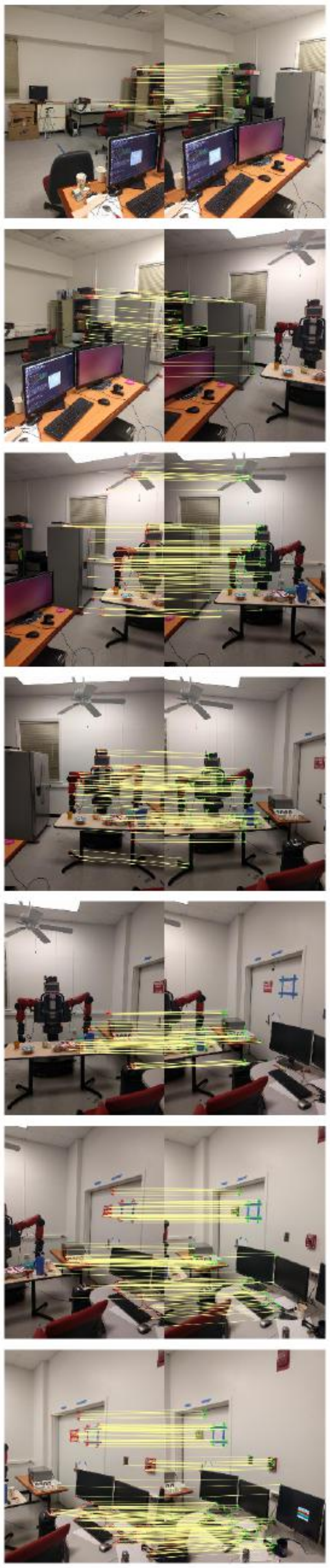
Fig. 39. RANSAC results on Train/Set3 images.

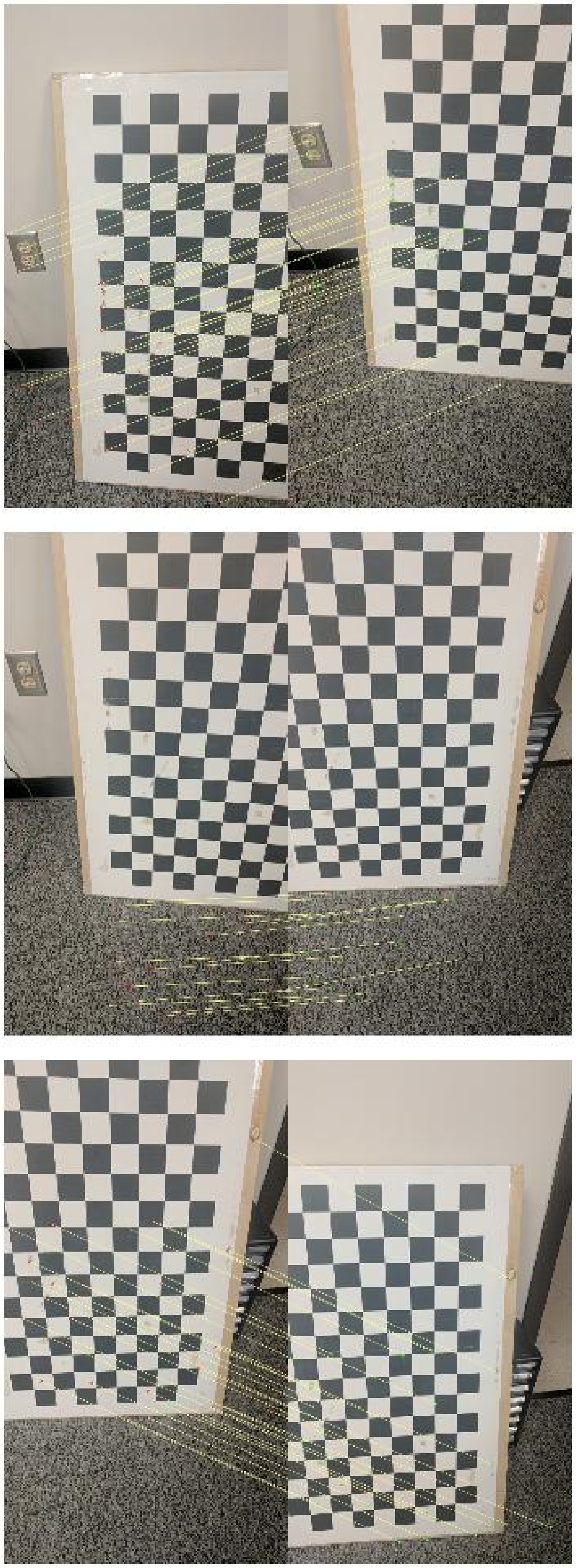

Fig. 40. RANSAC results on Test/Set1 images.

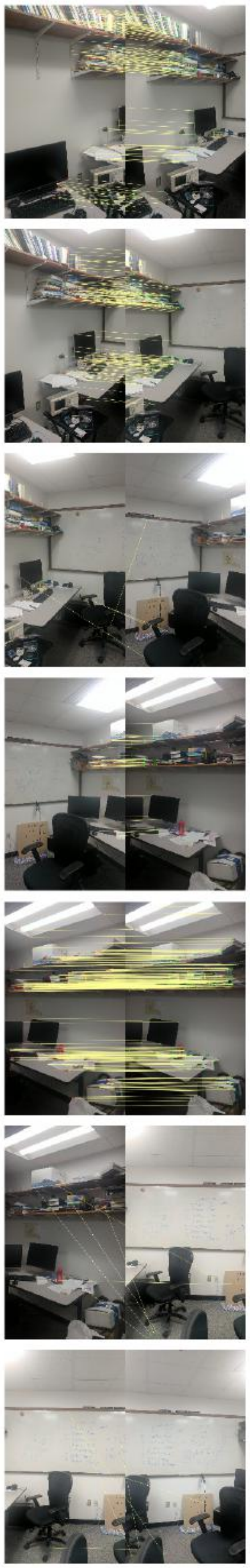

Fig. 41. RANSAC results on Test/Set2 images.

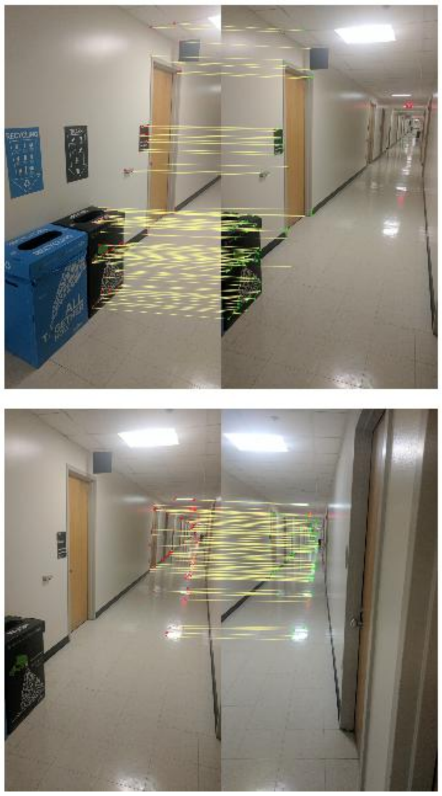

Fig. 42. RANSAC results on Test/Set3 images.

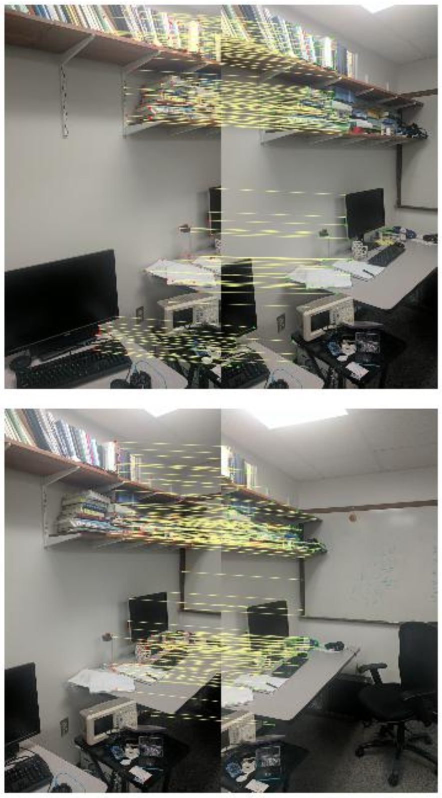

Fig. 43. RANSAC results on Test/Set4 images.

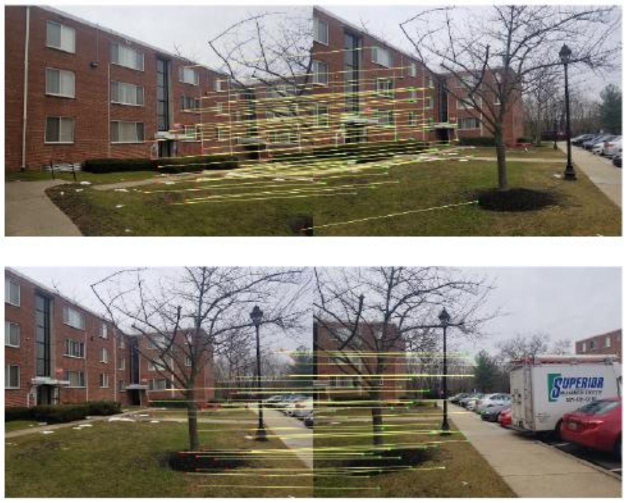

Fig. 45. RANSAC results on CustomSet2 images.

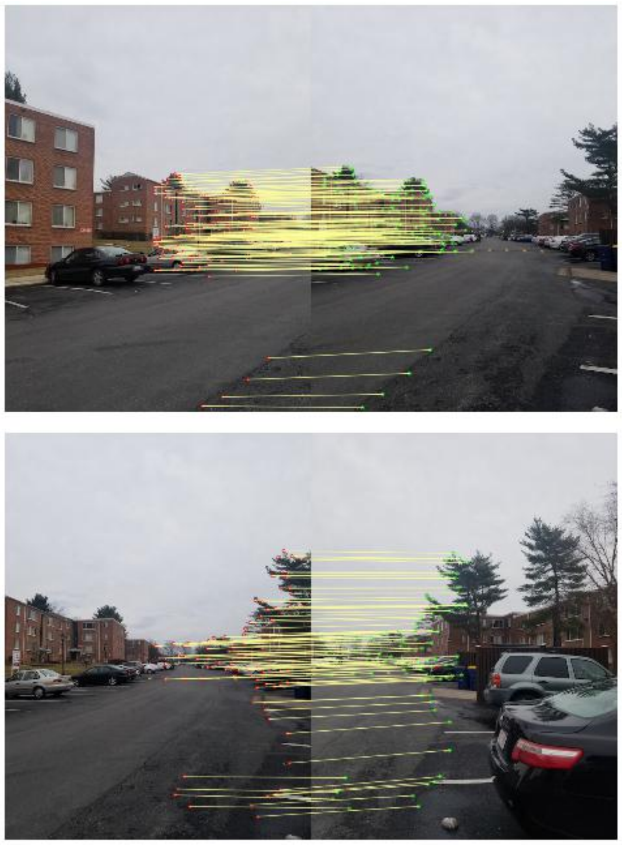

Fig. 44. RANSAC results on CustomSet1 images.

F. Warping and blending

Now that we have computed the homography matrix to perform the perspective transform between all pair of consecutive images, we go forward to warp and blend them together.

Suppose $H_{i,i+1}$ is the homography matrix to warp image i into image $i+1$. Then in order to find the homography matrix to warp image i into image k ($i < k$), we compute it as follows:

$$H_{i,k} = \prod_{j=i}^{k-1} H_{j,j+1}. \quad (1)$$

To find the homography matrix to warp in the reverse direction, we simply invert the computed homography matrix. That is $H_{i+1,i} = H_{i,i+1}^{-1}$.

With the equation above, we easily find the matrix to warp all images into the anchor image (recap: the anchor image is the one with index $\lfloor N/2 \rfloor$).

However, there are 2 issues with our algorithm:

1) **Numerical issue:** when the number of images is large (e.g. $N > 10$), we will have to multiply a lot of homography matrices together, which would easily result in numerical issues (*float64* datatype in Python is unable to represent very small or very large float number).
2) **Homography matrix does not exist:** in some cases, 2 consecutive images do not share a lot of correspondences, which makes us unable to find a reliable homography matrix between them. Because we assume that the input images come in sequence (e.g. from left to right, or from right to left, or from top to bottom, etc.), if we have to drop one image because not having enough correspondences, we will also have to drop all previous images.
   For example, we have $N = 10$ images, and image 3 and 4 do not share enough correspondences. In this case, we have to drop image 3 and also image 2 and image 1 (breaking the chain from $1 \to 10$ into $1 \to 3$ and $4 \to 10$ and only keep the latter).

We don't have enough time to try any fancy blending algorithms. Ours is just a simple approach: blending by placing each image on top of each other (there's no average operation or image compositing).

The results of warping and blending of all images in the train/test/custom set are presented from figure. 46 to figure. 54. For Train/Set3, we have to drop image *1.jpg* due to the homography matrix between image 1 and 2 is not reliable (create extremely bad distortion). Also the images from test set 2, the matching between images 3 and 4 is not found exactly due to the symmetry of the environment. In the Test Set 4, it was successful in neglecting the last two images to get a neat panorama.

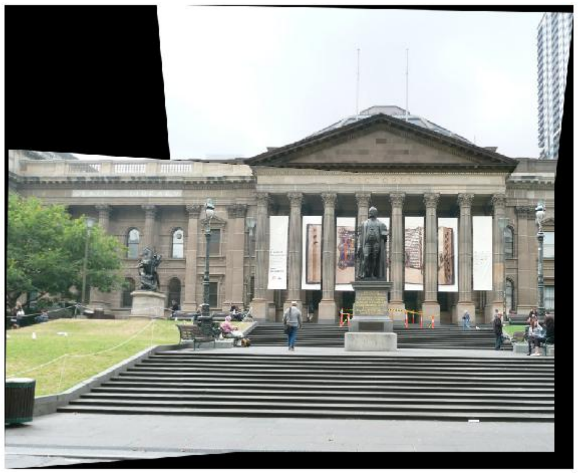

Fig. 46. Warping and blending result on Train/Set1 images.

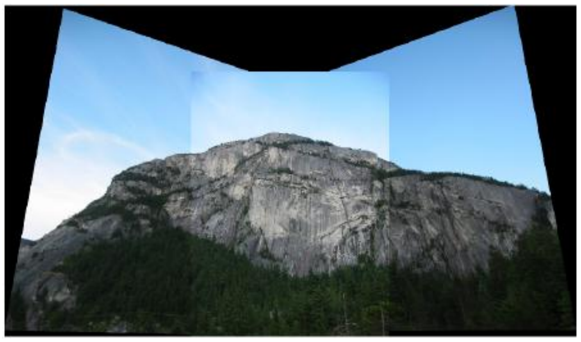

Fig. 47. Warping and blending result on Train/Set2 images.

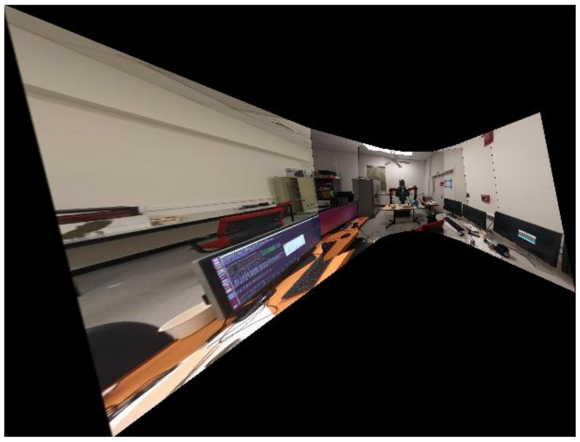

Fig. 48. Warping and blending result on Train/Set3 images.

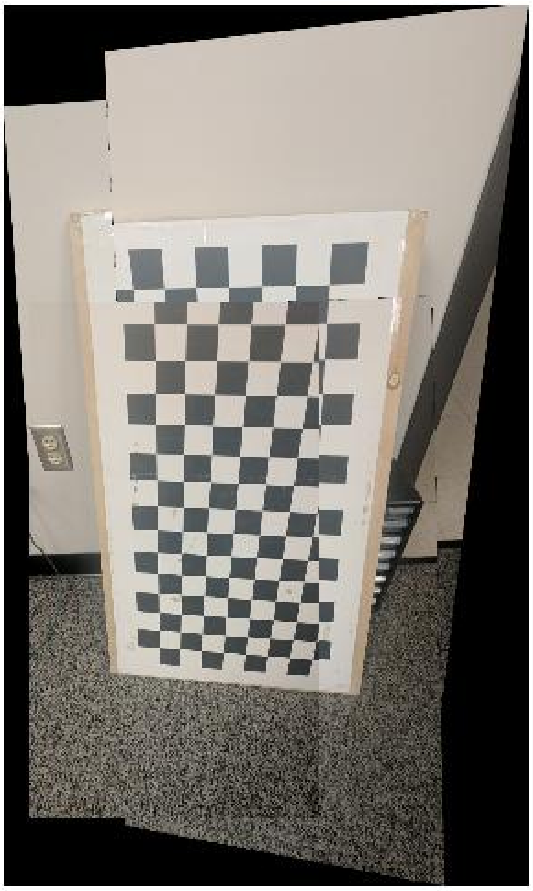
Fig. 49. Warping and blending result on Test/Set1 images.

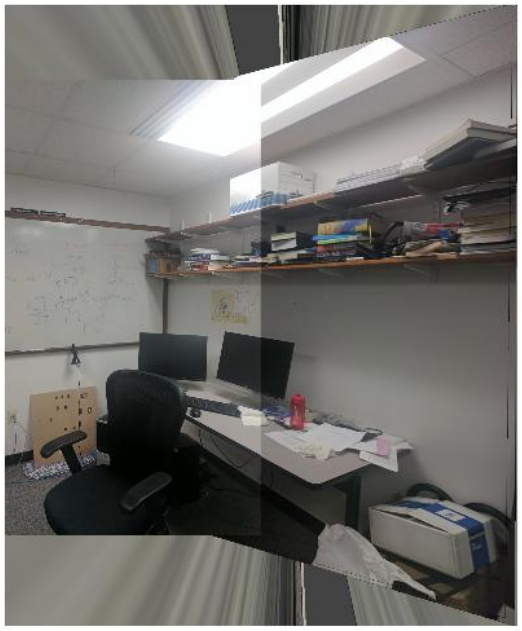
Fig. 50. Warping and blending result on Test/Set2 images.

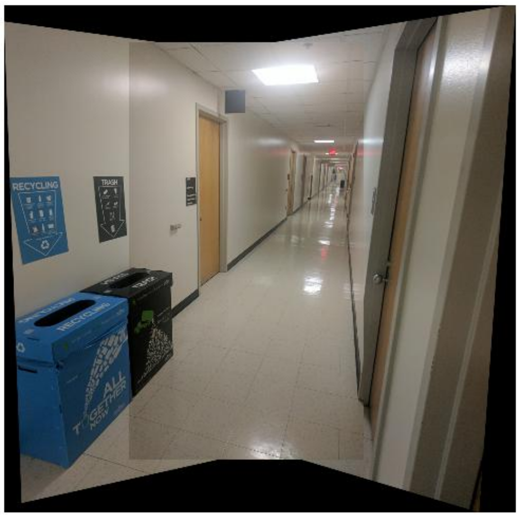

Fig. 51. Warping and blending result on Test/Set3 images.

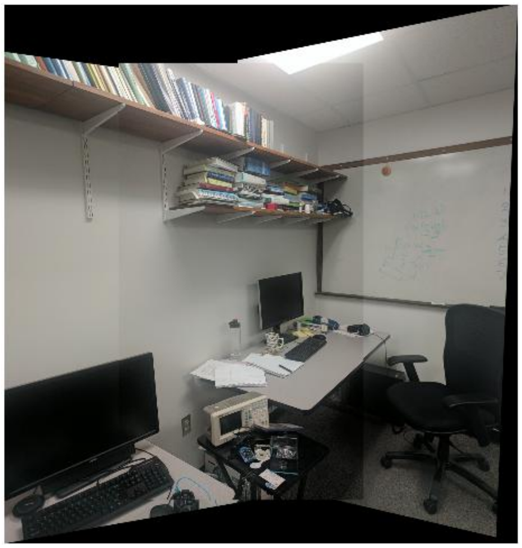
Fig. 52. Warping and blending result on Test/Set4 images.

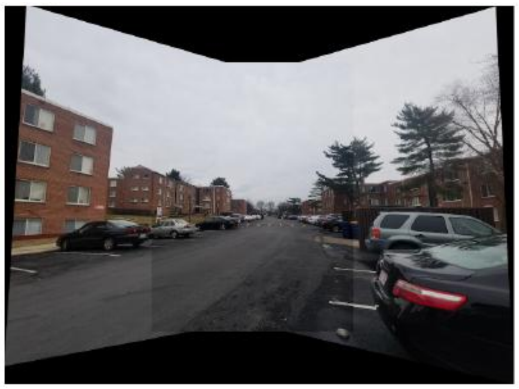
Fig. 53. Warping and blending result on CustomSet1 images.

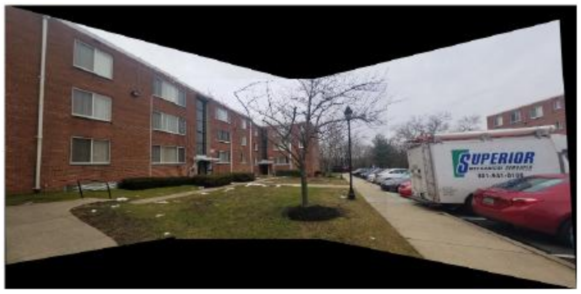

Fig. 54. Warping and blending result on CustomSet2 images.

II. Phase 2: Deep Learning Approach

In this section, we present our implemented deep learning approach to perform homography estimation and stitch images. Here we train our network both in a supervised and unsupervised way.

A. Data Generation

We follow the instruction on the web page and from [1] to generate synthetic data to train our model. We resize the input image to 320×240, convert it to grayscale, choose the same image patch size (128×128), and amount of perturbation ($\rho = 32$) as in [1]. We show examples of our synthetic data in figure 55.

We employ 2 approaches to feed data into our training procedure as follows:

1) *Static dataset:* For each image in the train/val dataset, we randomly generate 10 synthetic train/val instances and save them to local files. Each instance consists of an image pair (patch A, patch B) and the H_{4Pt} values. Since there are 5000 images in the train set and 1000 images in the validation set, our generated dataset has 50000 examples for training and 10000 examples for validation. In our training procedure, we only need to read from these saved files and feed them into our network.
2) *Dynamically generated dataset:* Create a static dataset as above might potentitally lead to our network overfit to the static dataset. Therefore, our second approach is to generate new synthetic data along the way while training. With this, our network is likely to receive new input examples at all time, which would prevent it from overfitting. This is very similar to data augmentation.

B. Network Architecture

We use the same network architecture for both our supervised and unsupvervised model. The architecture is shown in figure 56. Our architecture is a shallower version of the one presented in [1]. Input to the network is the pair of patch A and patch B, and the output is the 4-point homography H_{4Pt}.

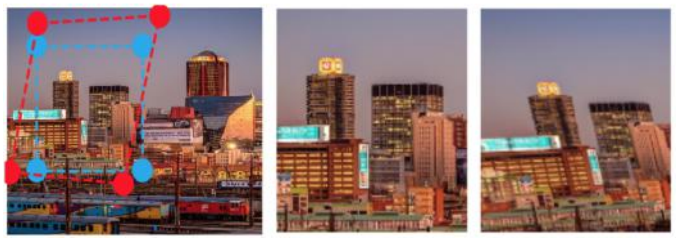

Fig. 55. Example of our synthetic data: (left) original image with patch A and its corner perturbations (middle) patch A (right) patch B.

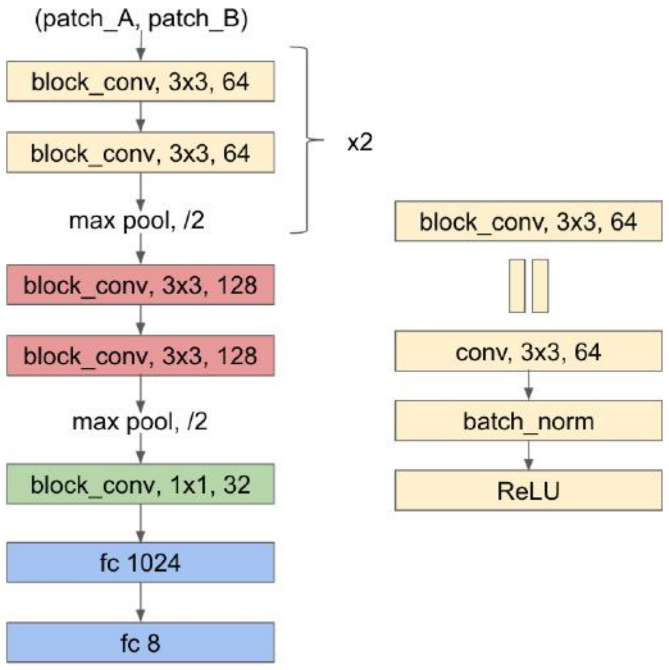

Fig. 56. (Left) Our network architecture (Right) Explanation of a block_conv layer: it consists of the 3×3 convolutional layer followed by a batch normalization and ReLU layer.

C. Supervised Model

We use Adam optimizer with learning rate $1e-4$ and batch size 64 to train the network. For the supervised model trained on the static dataset, as it experiences overfitting easily, we add a dropout layer with probability 0.5 after the FC-1024 layer. For the supervised model trained on dynamically generated data, it hardly achieves overfitting. We also notice that scaling image values into the range $[0, 1]$ (by dividing image pixel value by 255), and the H_{4Pt} values into the range $[-1, 1]$ (by dividing corner perturbation by 32) help the network converge better.

We present a plot of the training vs. validation loss in figure 57. Here, the loss is the mean squared error between the groundtruth and the predicted H_{4Pt} values after we have scaled them into the range $[-1, 1]$. Therefore, to scale them back into pixel value, we must apply the following formula: let x be the MSE loss, the MSE loss on the pixel scale is $32\sqrt{x}$.

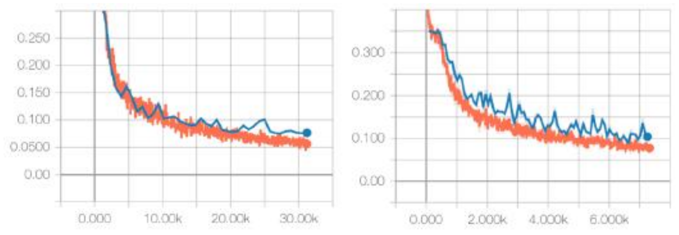

Fig. 57. Plot of train loss (red) vs. validation loss (blue) of our supervised model trained on (left) static dataset (right) dynamically generated data.

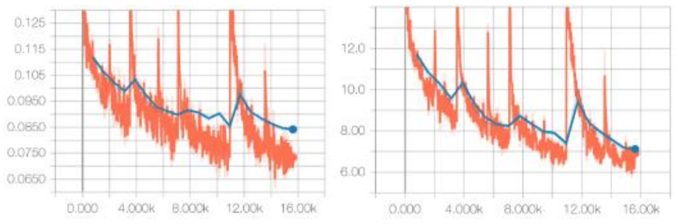

Fig. 58. Plot of train (red) vs. validation (blue) loss of our unsupervised model on the static dataset. (Left) Photometric loss (Right) H_{4Pt} loss.

Due to limitations in time and computing resources, we could not train our model any further even though the loss value is still gradually decreasing. We will later present our supervised model performance in the last subsection.

D. Unsupervised Model

We also use Adam optimizer with learning rate $1e-4$ and batch size 64 to train our unsupervised model. Here, we only train the model on the static dataset due to time limitation. We also scale image pixel value into $[0, 1]$ but we did not scale the H_{4Pt} values.

We present a plot of the training vs. validation photometric loss and H_{4Pt} loss in figure 58. Notice that our model only learns to minimize the photometric loss. We only show the H_{4Pt} loss here to demonstrate the capability of the model to estimate homography during training. We will present our unsupervised model performance in the next subsection.

E. Performance on Train/Val/Test set

We present the performance in terms of mean corner pixel error of our models on the train, validation, and secret test set in table I. The way we calculate the mean corner error is the same as in [1]. In the table, *Static* means the model has been trained on our statically created dataset, and *Dynamic* means the model has been trained using dynamically generated data (as explained in section II.A).

We also record the forward pass run-time of our network architecture to be around 0.0024 second on the GeFore GTX 1080 Ti GPU.

From the performance table, we can see that our models generalize really well on the test set. The mean corner error on the train and test set are not too much different. Moreover, judging from the training loss plots in figure 57 and 58 there's still a lot of room for improvement. However, as we

Model name	Train EPE	Val EPE	Test EPE
Supervised + Static	10.55px	10.58px	10.54px
Supervised + Dynamic	12.64px	12.63px	12.64px
Unsupervised + Static	10.12px	11.09px	10.95px

TABLE I
AVERAGE EPE RESULTS OF OUR MODELS ON THE TRAIN/VALIDATION/TEST SET.)

mentioned in the last subsections, due to limitation in time and computation resources, we could not continue to train our models further. The reason that our *Supervised + Dynamic* model performs worse than the others is that we have trained it for only around 6000 iterations (you can see the number of training iterations in figure 57), whereas we train *Supervised + Static* for more than 30000 iterations. We believe that *Supervised + Dynamic* will perform better than *Supervised + Static* if we train it more.

F. Comparision between Classical and Deep Learning Method for Homography Estimation

In this section, we show some examples to compare between traditional and deep learning method for estimating homography between 2 images.

For deep learning method, in order to estimate the homography between 2 images A and B, we will do as follows:

1) Resize A and B to $(320, 240)$ and convert them to grayscale.
2) Obtain 5 random crops of size $(128, 128)$ in A, and obtain 5 random crops at those same locations of size $(128, 128)$ in B (always make sure that in those 5 crops, there is a crop at the center of the image since this crop is likely to give more accurate estimates).
3) For each pair of random crops at the same location, forward them through our supervised/unsupervised model to get the homography. The final homography is the average between all computed homography matrices.

Examples to compare between traditional and deep learning method are shown in figure 59.

Even though we have very great results with the deep learning model shown in figure 59, our proposed method above to estimate the homography between 2 arbitrary images are not good. Our proposed method is only good with synthetic data when the 2 input patches share visual similarities. On the other hand, given 2 arbitrary images, it is difficult to select two patches with similar visual appearance to input to the network. We have tried generating random crops across the whole two images. This is also the reason we are unable to produce panoramas using supervised and unsupervised approach.

REFERENCES

[1] D. DeTone, T. Malisiewicz, and A. Rabinovich, *Deep Image Homography Estimation*.

[2] T. Nguyen, S. W. Chen, S. S. Shivakumar, C. J. Taylor, V. Kumar, *Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model*.

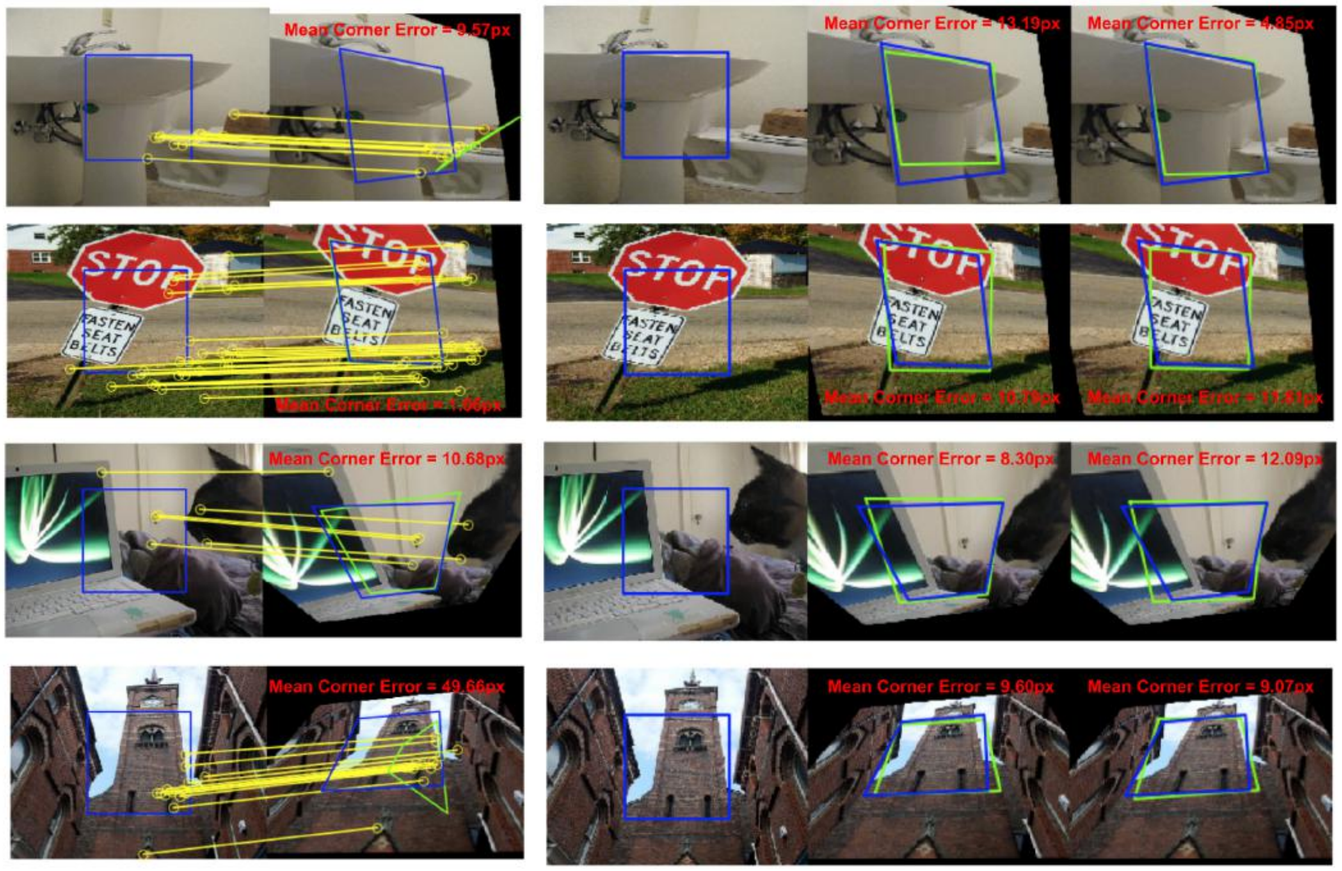
Mean Corner Error = 9.57px
Mean Corner Error = 13.19px
Mean Corner Error = 4.85px
Mean Corner Error = 1.06px
Mean Corner Error = 10.79px
Mean Corner Error = 11.81px
Mean Corner Error = 10.68px
Mean Corner Error = 8.30px
Mean Corner Error = 12.09px
Mean Corner Error = 49.66px
Mean Corner Error = 9.60px
Mean Corner Error = 9.07px

Fig. 59. Traditional vs Deep Learning Homography Estimation Method. Blue is the groundtruth box and green is our transformed box in the warped image using the estimated homography. Left column shows the results of using traditional method. Right column, in the center shows the results of the supervised model. Right column, on the right shows the results of using the unsupervised model. Row 1: traditional method fails due to the detected keypoints lie too close and are almost colinear, while deep learning method achieves very good results. Row 2: traditional method produces an almost perfect result, whereas deep learning shows acceptable prediction. Row 3: traditional and deep learning method produce similar results on this example. Row 4: traditional method again fails due to the detected keypoints are too close and almost colinear, whereas deep learning method does not have this problem.