CMSC733: Project 1 -MyAutoPano

Chayan Kumar Patodi
UID : 116327428
Email: ckp1804@terpmail.umd.edu
Using 2 Late Days

Saket Seshadri Gudimetla Hanumath
UID : 116332293
Email: saketsgh@terpmail.umd.edu
Using 2 Late Days

***Abstract*—The purpose of this project is to stitch two or more images in order to create one seamless panorama image. In this project we will be working on Homography estimation, which is one of the most significant concepts in Computer Vision. It holds many applications in the fields of Image Processing. This project has two phases. Phase 1 the traditional approach, where given input images, we detect corners, estimate homography and stitch the images. Phase 2 is implementing two deep learning approaches to estimate the homography between the two images. We use supervised and unsupervised approaches to do the same.**

I. Phase 1 : Traditional Approach

In this section we work on a traditional approach to achieve our goal of panorama stitching. We present the details of how to perform image stitching on multiple images. To begin with, we make sure that the consecutive pair of images that we are working with share at least an overlapping region. The whole pipeline can be divided into 5 steps. Sequentially, these are corner detection, Adaptive Non-maximal Suppression(ANMS),Generation of Feature Descriptors , Feature Matching, RANSAC for outlier rejection, and finally stitching the images. The whole pipeline is explained in Figure 1.

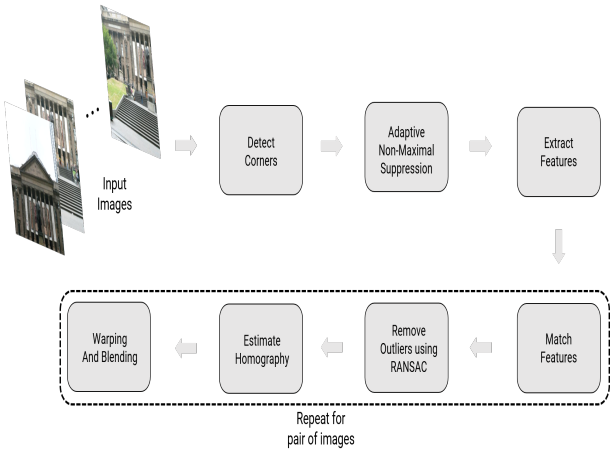


Fig. 1. Panorama Stitching using traditional method.

A. Corner Detection

The very first step in the given pipeline is extracting corners in the given input images. We have used Corner Harris **(cv2.CornerHarris)** for this task.The output of this step for all the sets is presented from figure 2 to 10.

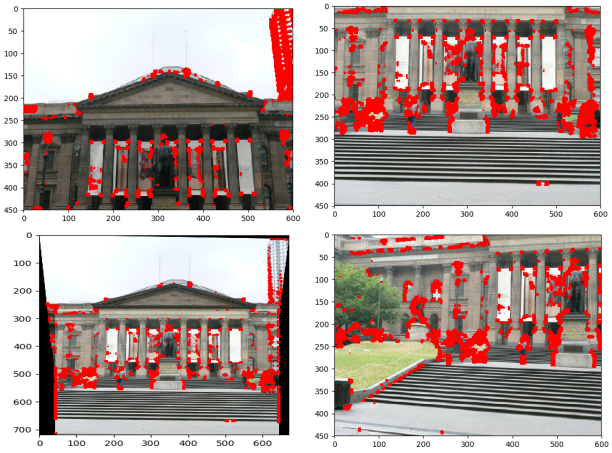


Fig. 2. Detected Corners in Train Set 1.We read the images in sequence and stitch two images first and then repeat the whole pipeline with the stitched image and next input image and thus the corners detected are in the stitched images.

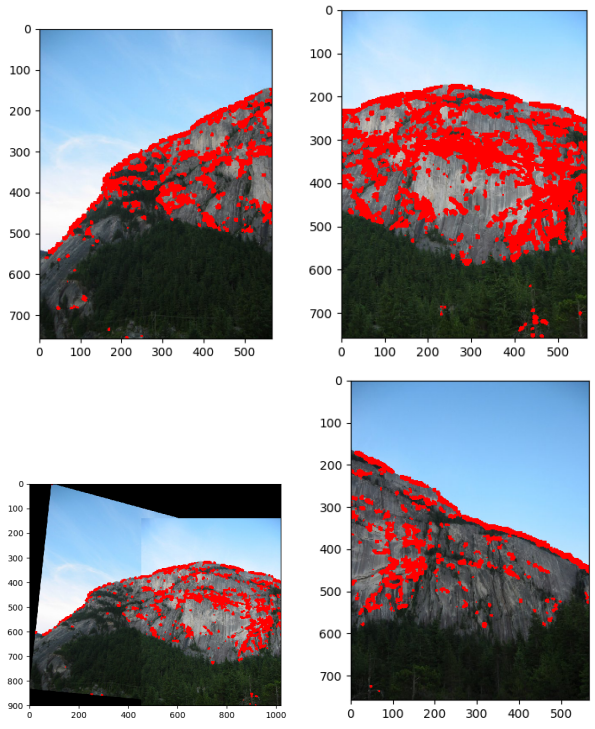


Fig. 3. Detected Corners in Train Set 2

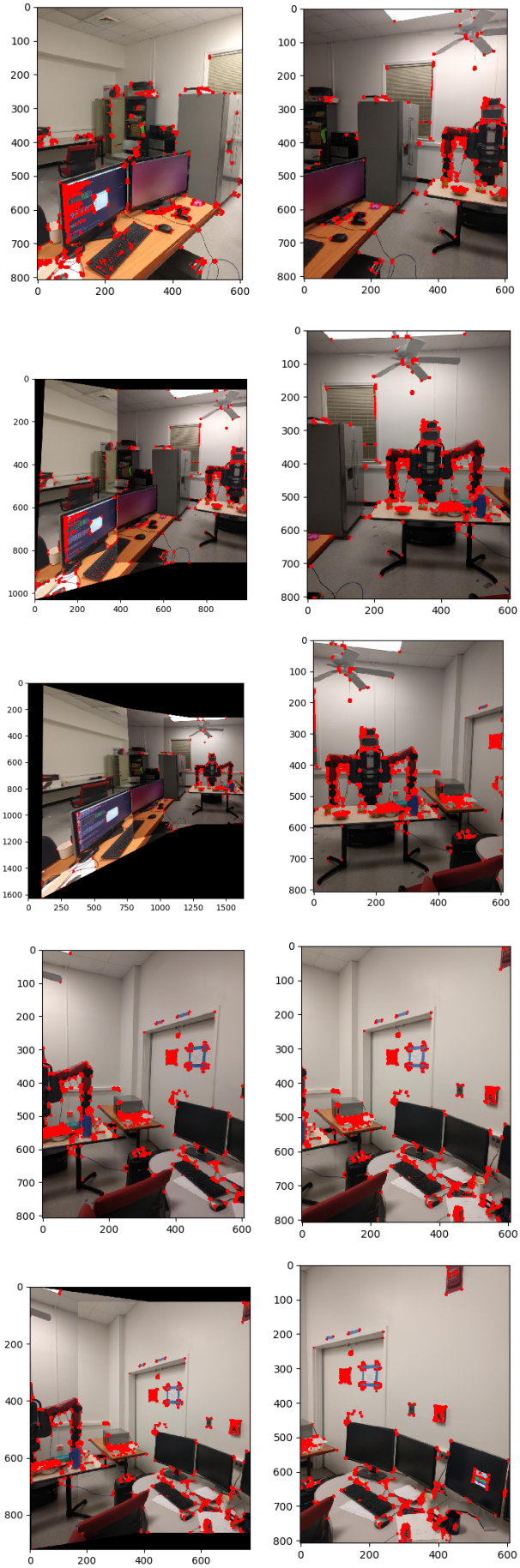


Fig. 4. Detected Corners in Train Set 3

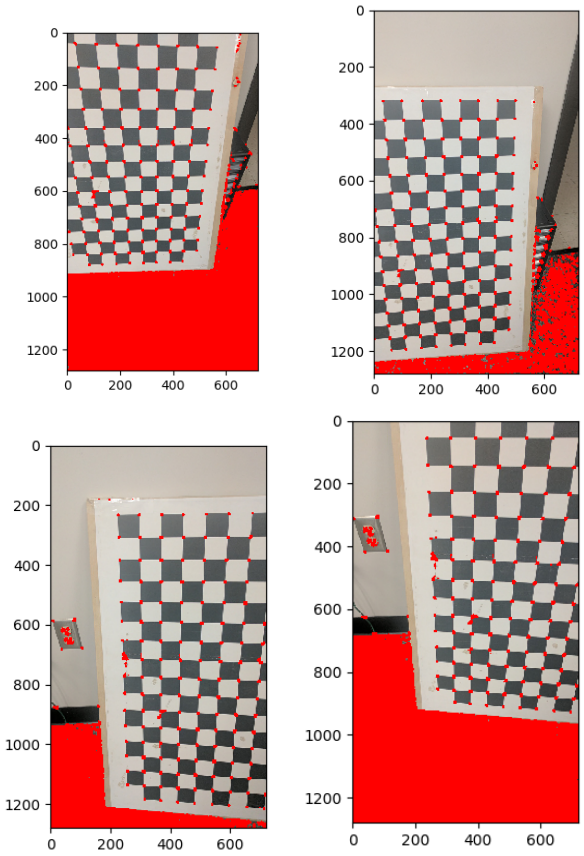
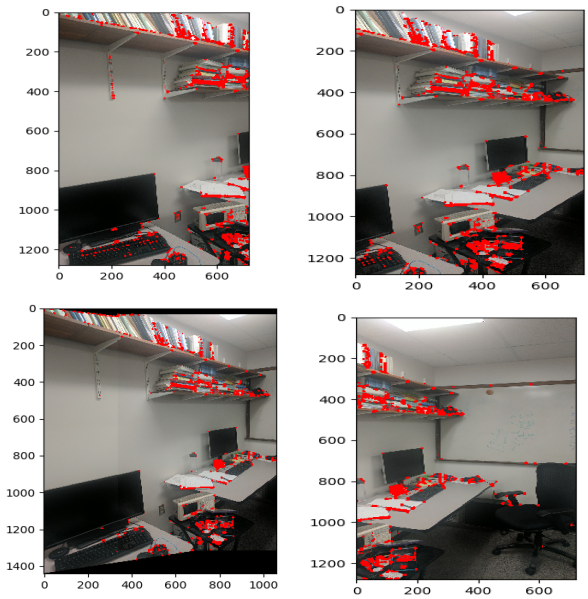


Fig. 5. Detected Corners in Test Set 1



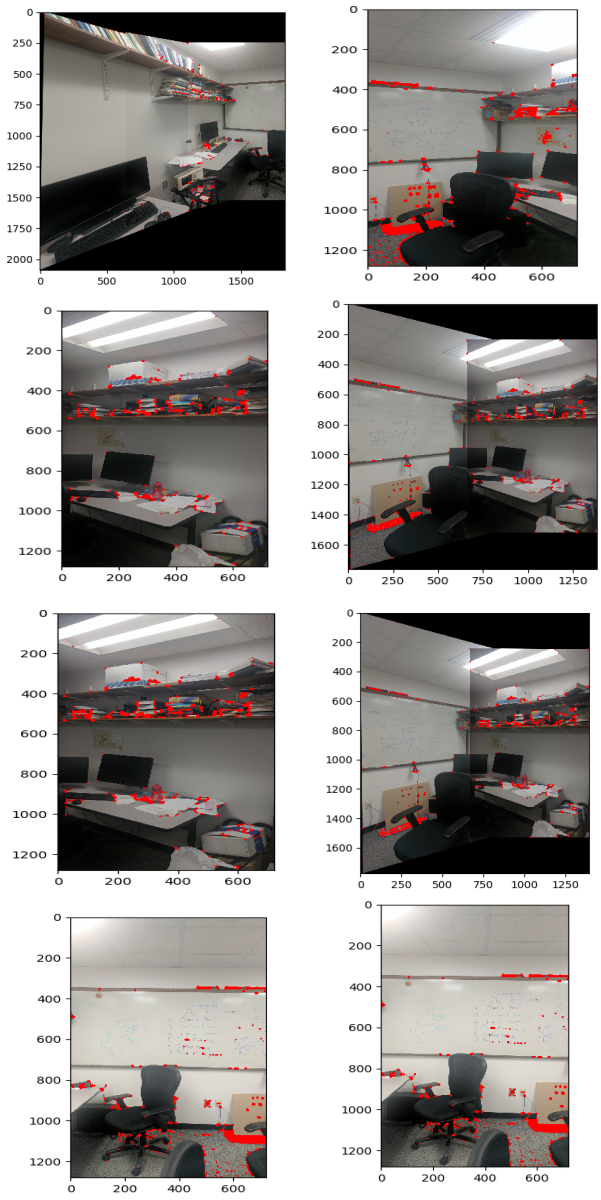
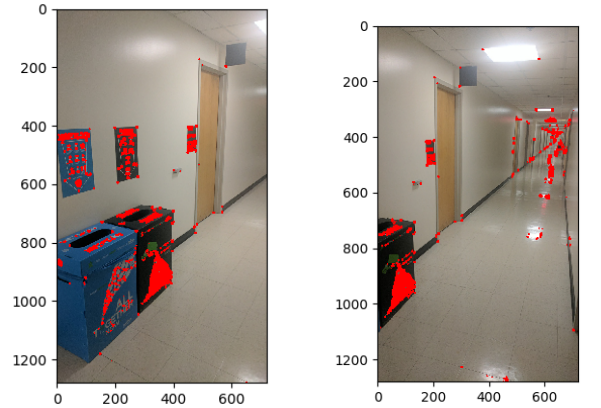


Fig. 6. Detected Corners in Test Set 2



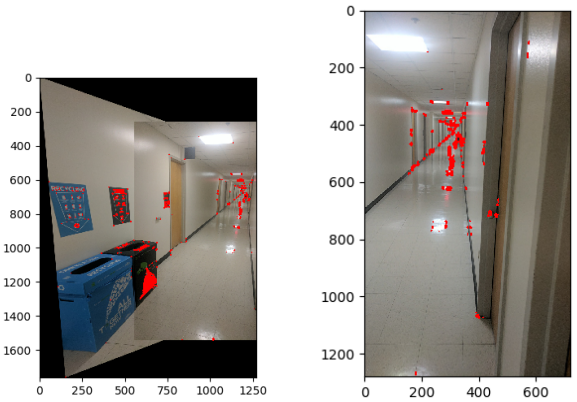


Fig. 7. Detected Corners in Test Set 3

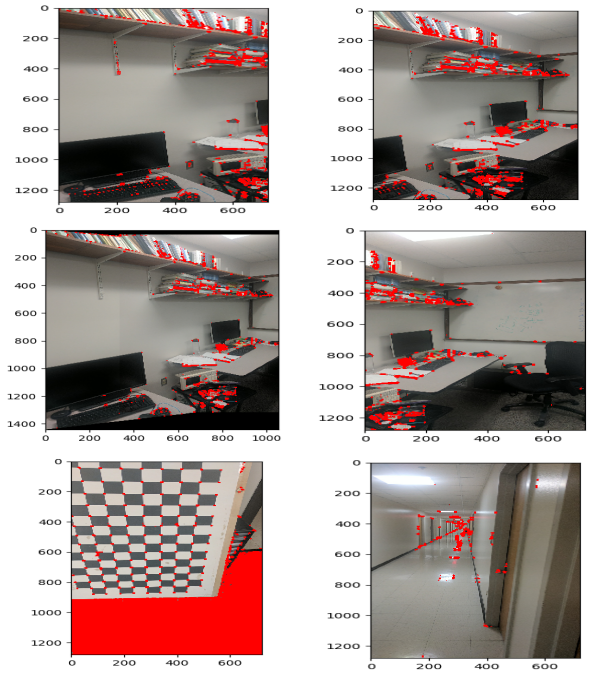
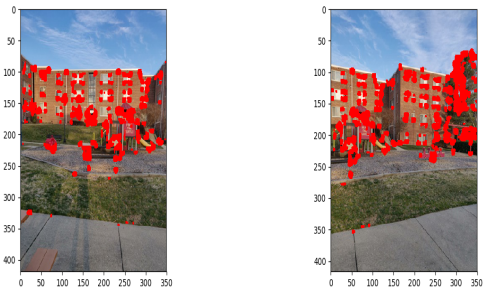


Fig. 8. Detected Corners in Test Set 4



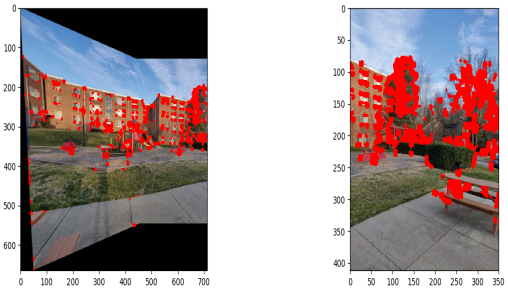
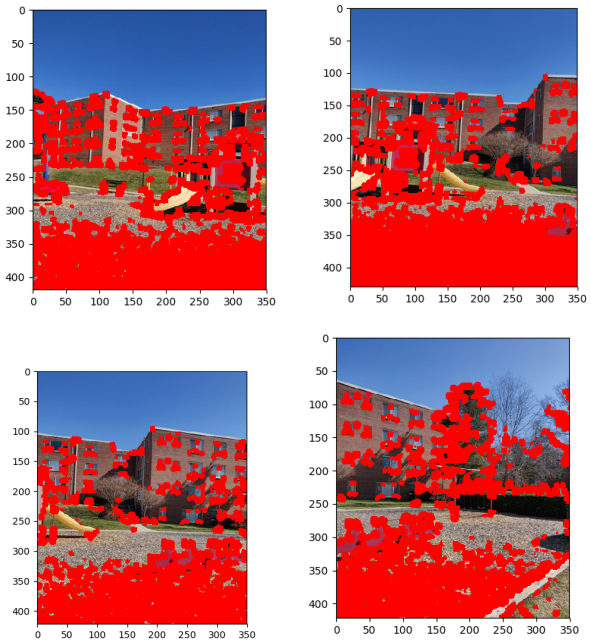


Fig. 9. Detected Corners in Custom Set 1



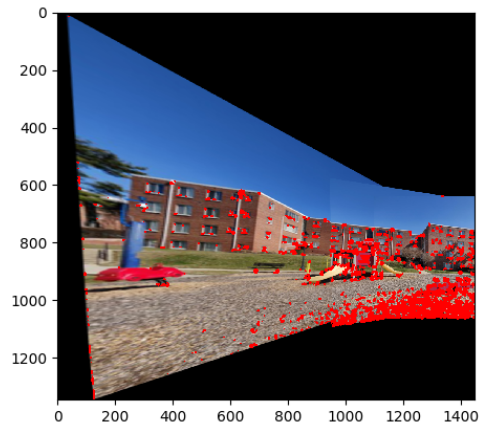


Fig. 10. Detected Corners in Custom Set 2

B. *Adaptive Non-maximal Suppression*

The second step in the pipeline is to filter out the redundant corners. Observing the output of corner harris we can see that the number of detected corners is huge. The objective of this step is to detect corners such that they are equally distributed across the image in order to avoid wrong or incorrect warping. We want to choose only the Nbest corners after ANMS, which is the objective of this step. The algorithm for implementing ANMS is given in figure 11 below. The outputs of this step are presented from figure 12 to 20.

Input : Corner score Image (C_{img} obtained using `cornermetric`), N_{best} (Number of best corners needed)
Output: (x_i, y_i) for $i = 1 : N_{best}$

Find all local maxima using `imregionalmax` on C_{img};
Find (x, y) co-ordinates of all local maxima;
$((x, y)$ for a local maxima are inverted row and column indices i.e., If we have local maxima at $[i, j]$ then $x = j$ and $y = i$ for that local maxima);

Initialize $r_i = \infty$ for $i = [1 : N_{strong}]$

for $i = [1 : N_{strong}]$ **do**
  for $j = [1 : N_{strong}]$ **do**
    if $(C_{img}(y_j, x_j) > C_{img}(y_i, x_i))$ **then**
      $\text{ED} = (x_j - x_i)^2 + (y_j - y_i)^2$
    end
    if $ED < r_i$ **then**
      $r_i = \text{ED}$
    end
  end
end

Sort r_i in descending order and pick top N_{best} points

Fig. 11. ANMS algorithm

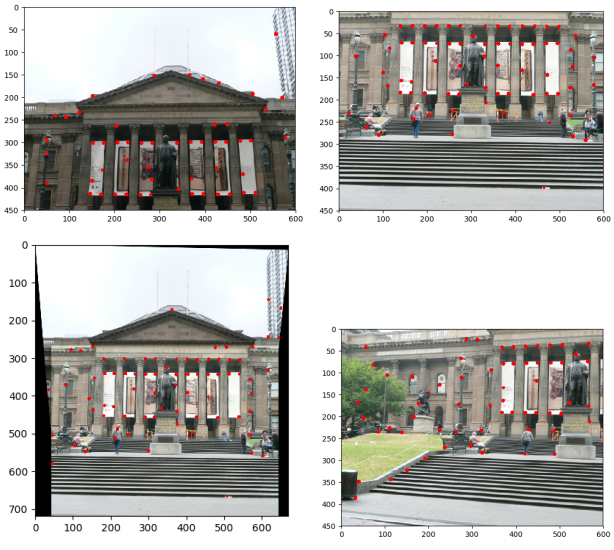


Fig. 12. Detected Corners after ANMS for Train Set 1

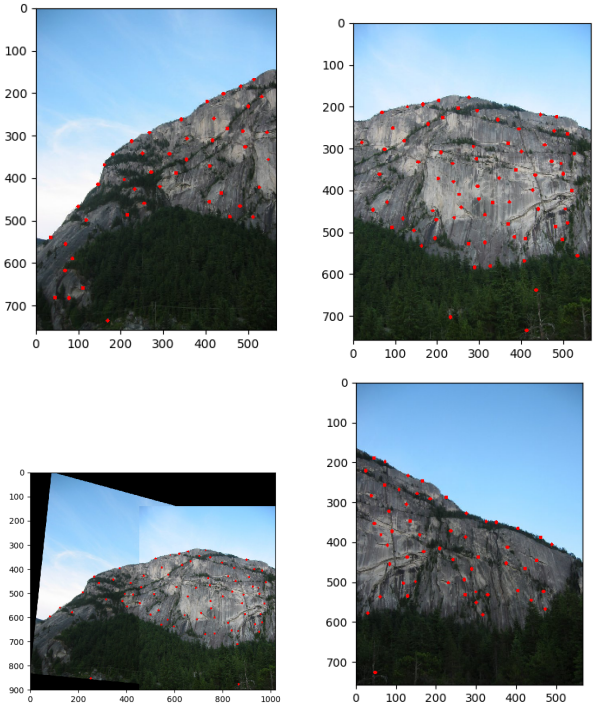


Fig. 13. Detected Corners after ANMS for Train Set 2.

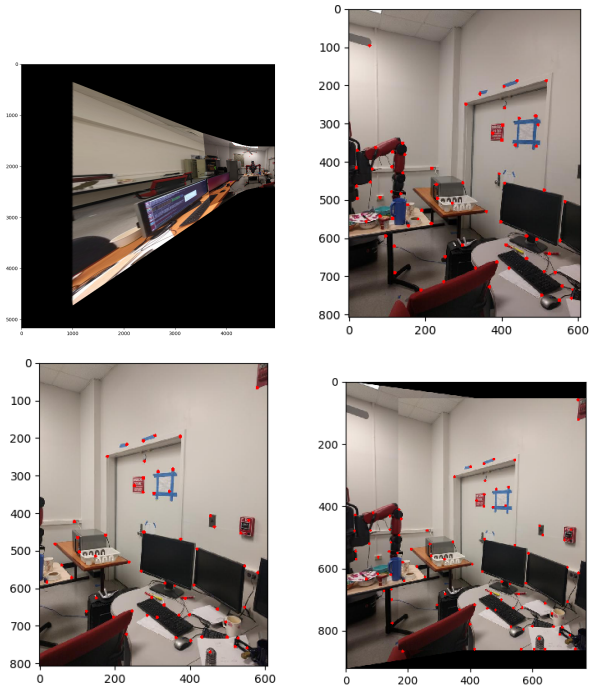
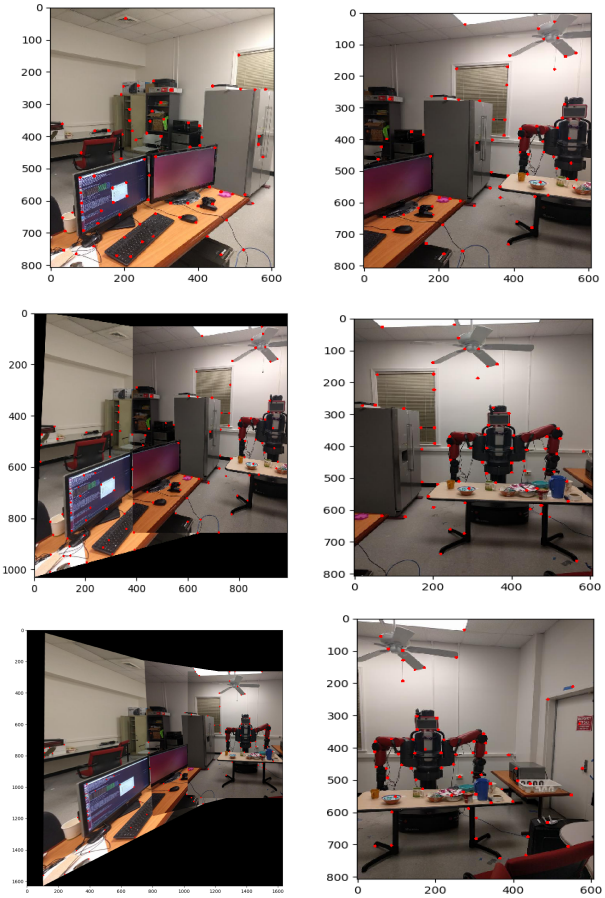


Fig. 14. Detected Corners after ANMS for Train Set 3.



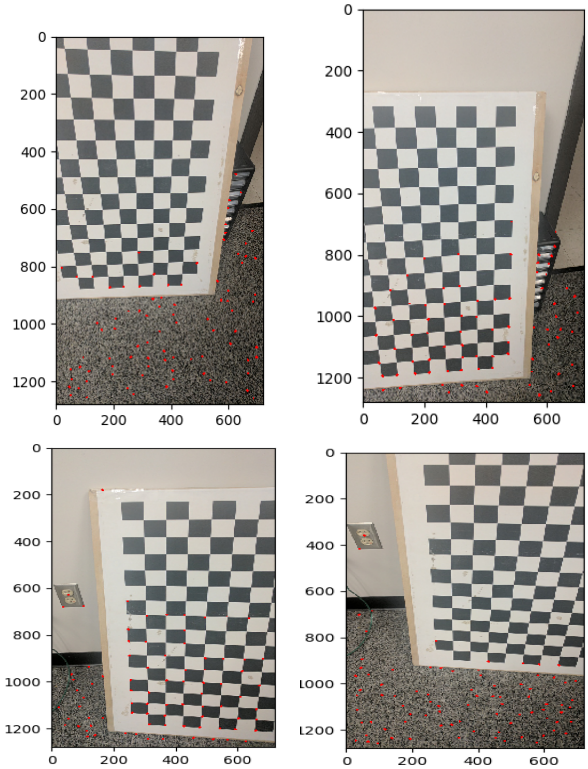


Fig. 15. Detected Corners after ANMS for Test Set 1

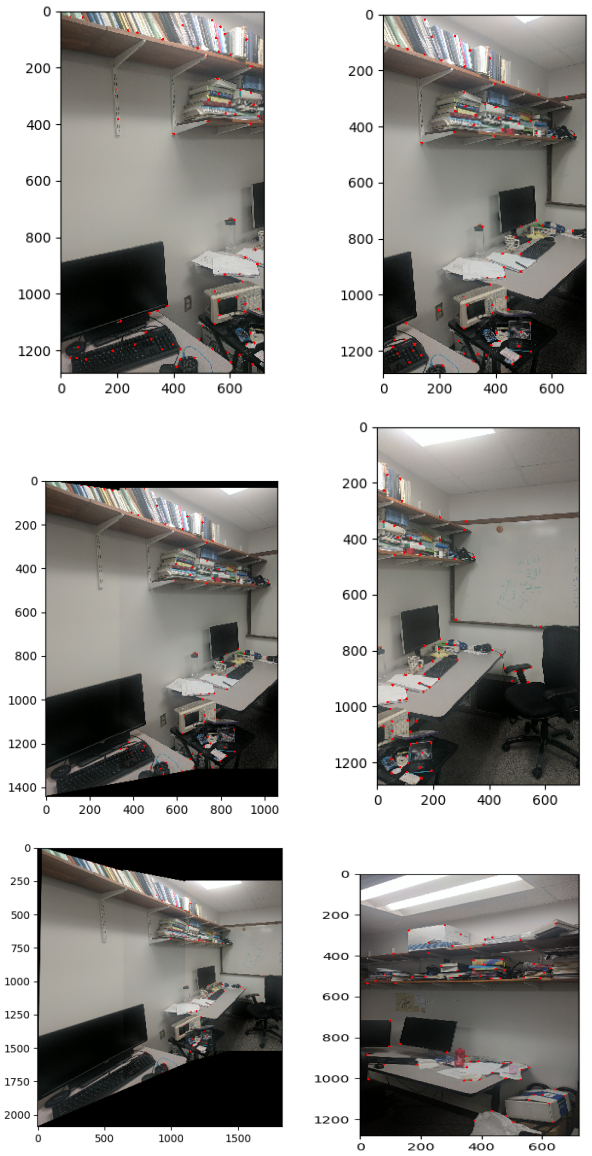
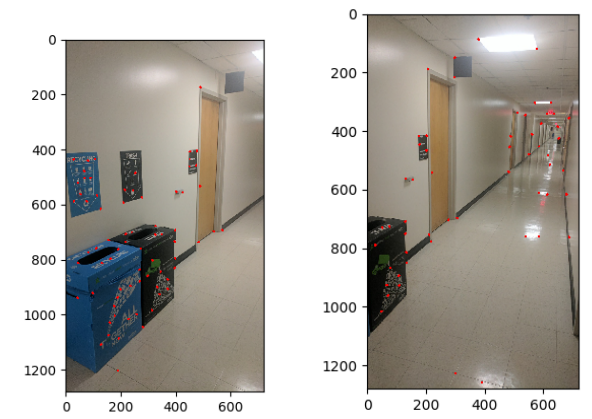


Fig. 16. Detected Corners after ANMS for Test Set 2



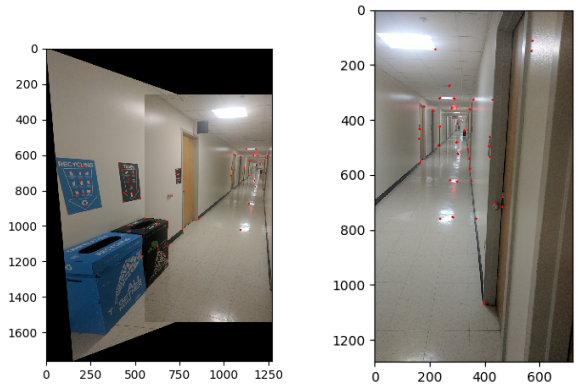


Fig. 17. Detected Corners after ANMS for Test Set 3.

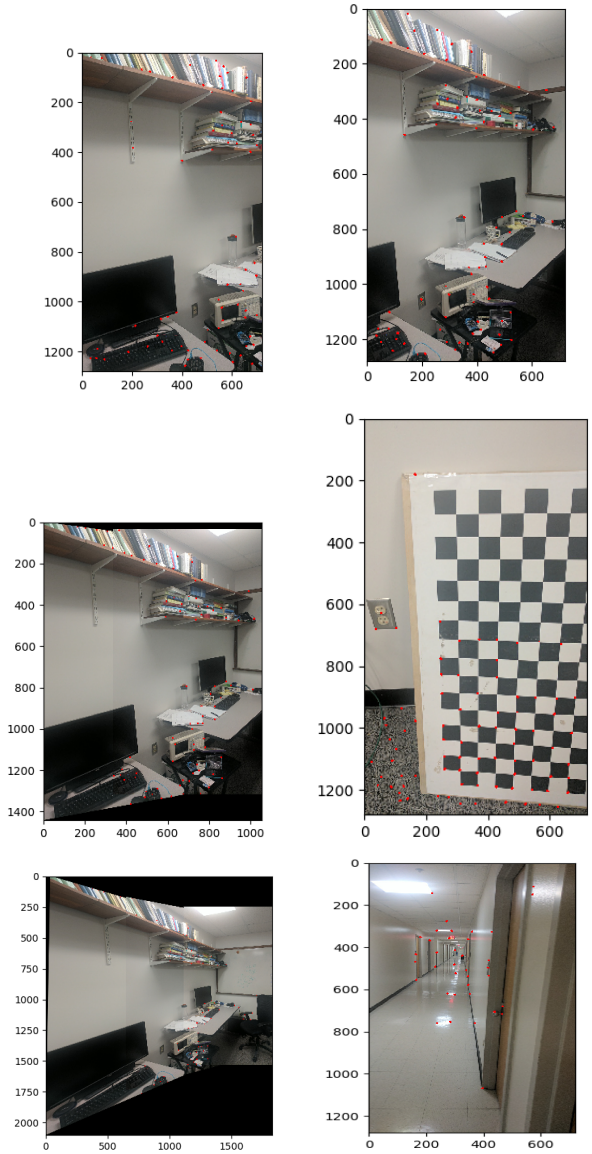


Fig. 18. Detected Corners after ANMS for Test Set 4.

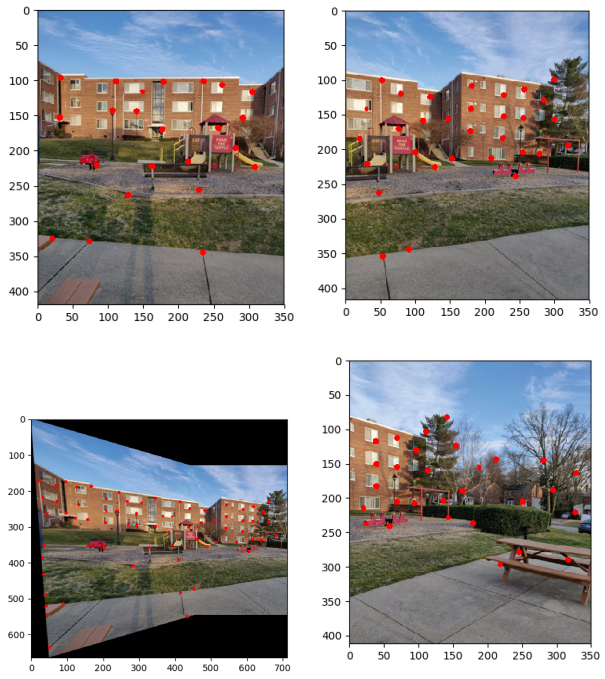


Fig. 19. Detected Corners after ANMS for Custom Set 1.

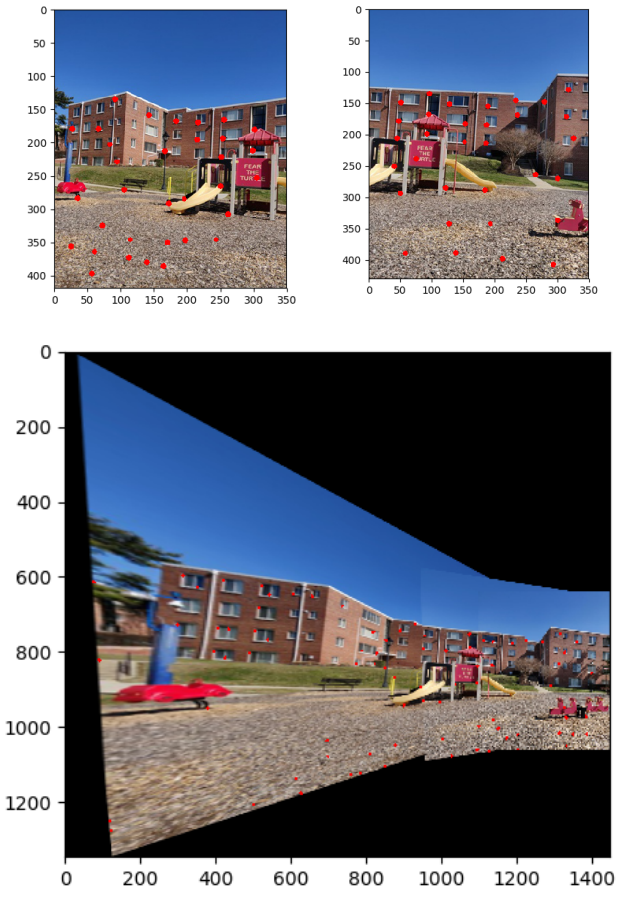


Fig. 20. Detected Corners after ANMS for Custom Set 2.

C. Feature Descriptor

In order to stitch the images, we need to match the corners on one image to the same corners in another image. To do so, we need to produce a feature vector that encodes all local information around it. To do so, we take a patch of size 40 X 40, centered around the feature point. Now pre-processing this patch and reshaping it to obtain a vector of size 64X1. We now standardize this vector to remove bias and achieve some amount of illumination invariance.The output of this step for a few input images is presented in figure 21.

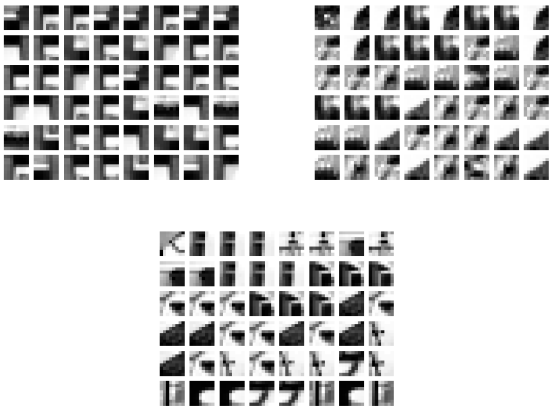
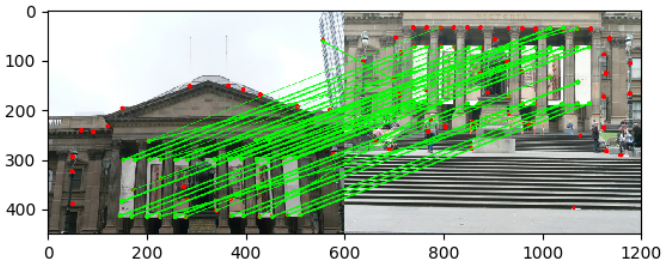


Fig. 21. Vector Features before reshaping

D. Feature Matching

Once we have encoded key point feature vector, we match the feature points. This is also know as finding correspondences in between two images. We compute match pairs in two different images of interest by calculating the sum of square difference of each feature vector in the first image to each such vector in the second image and save them in sorted lists. We then take the ratio of the best match to the second best match and if it is below 0.90 (user chosen threshold), we reject it. We find the confident feature correspondences and these points will be used to estimate the transformation between the two images. The output of this step is presented from figure 22 to 30.



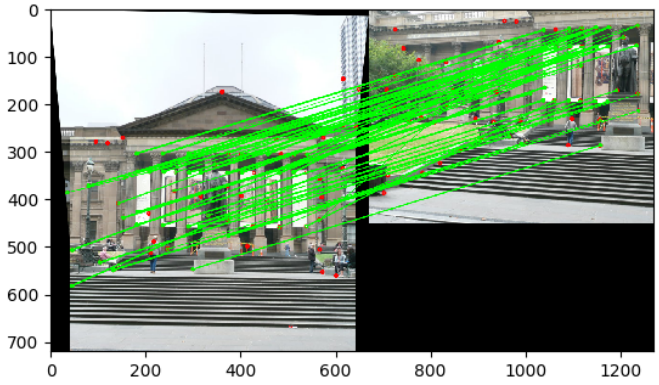


Fig. 22. Feature Matching for Train Set 1

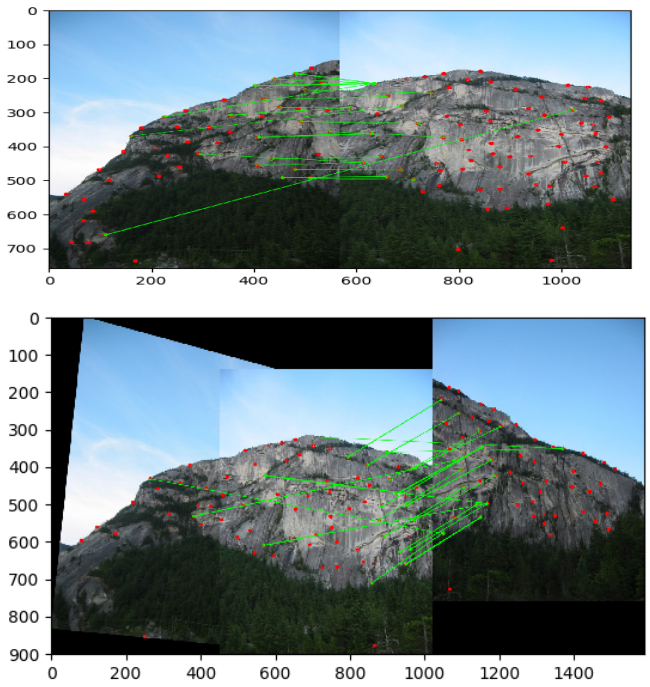
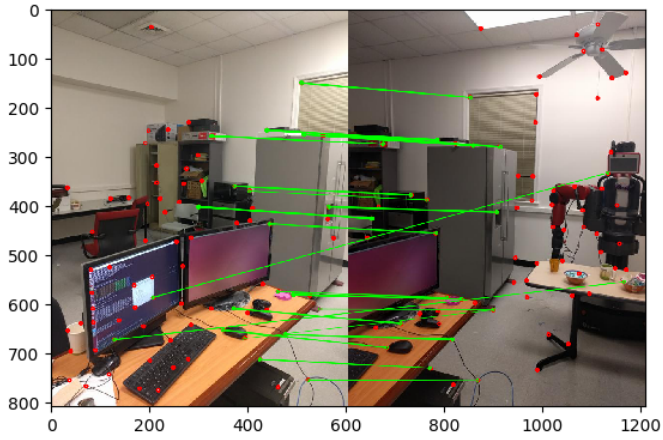
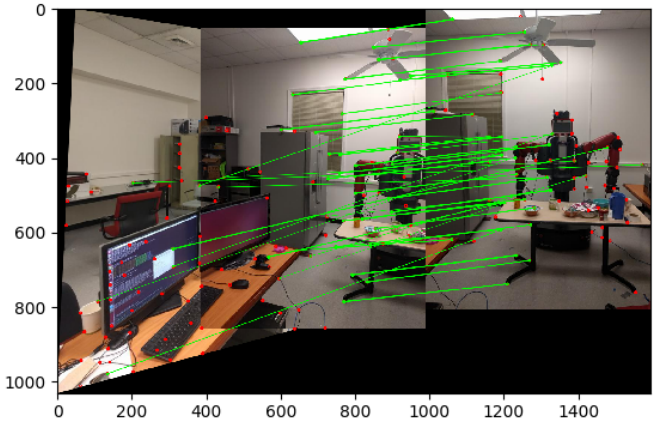
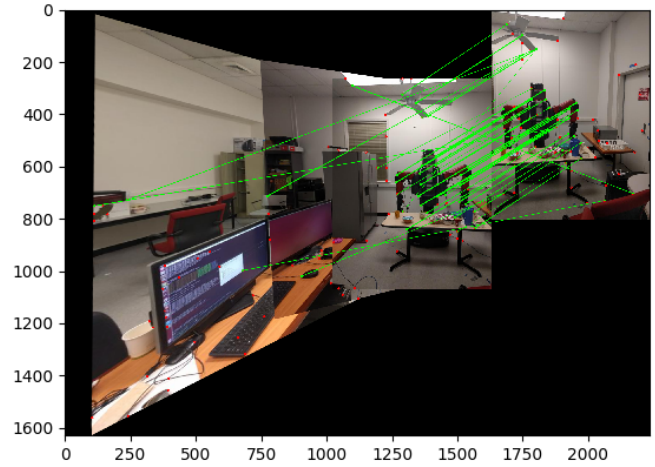
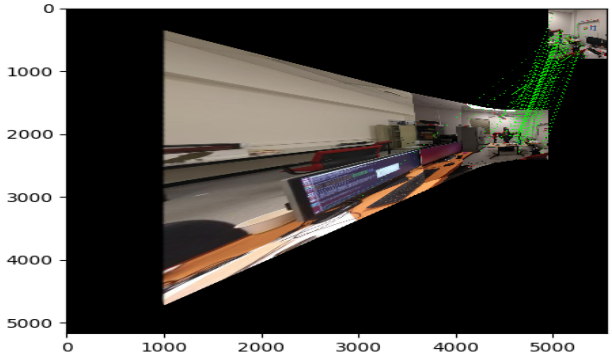


Fig. 23. Feature Matching for Train Set 2









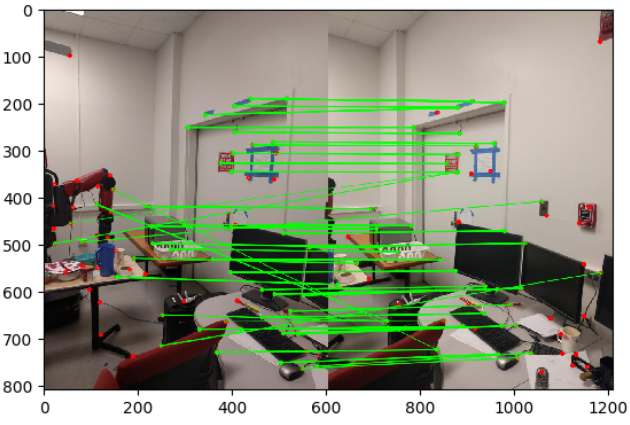


Fig. 24. Feature Matching for Train Set 3

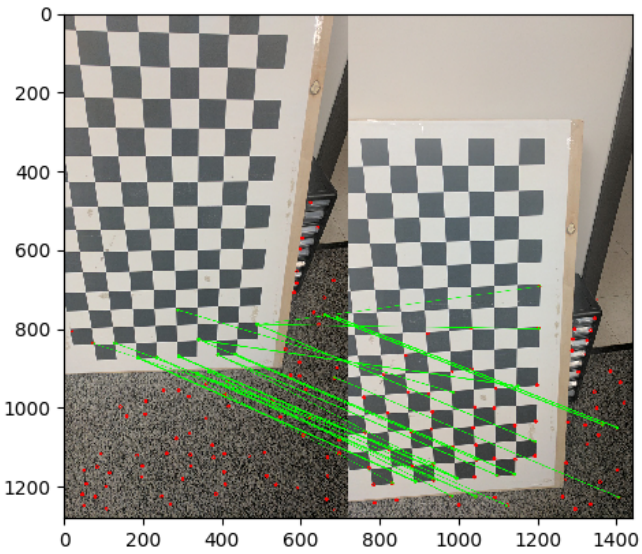
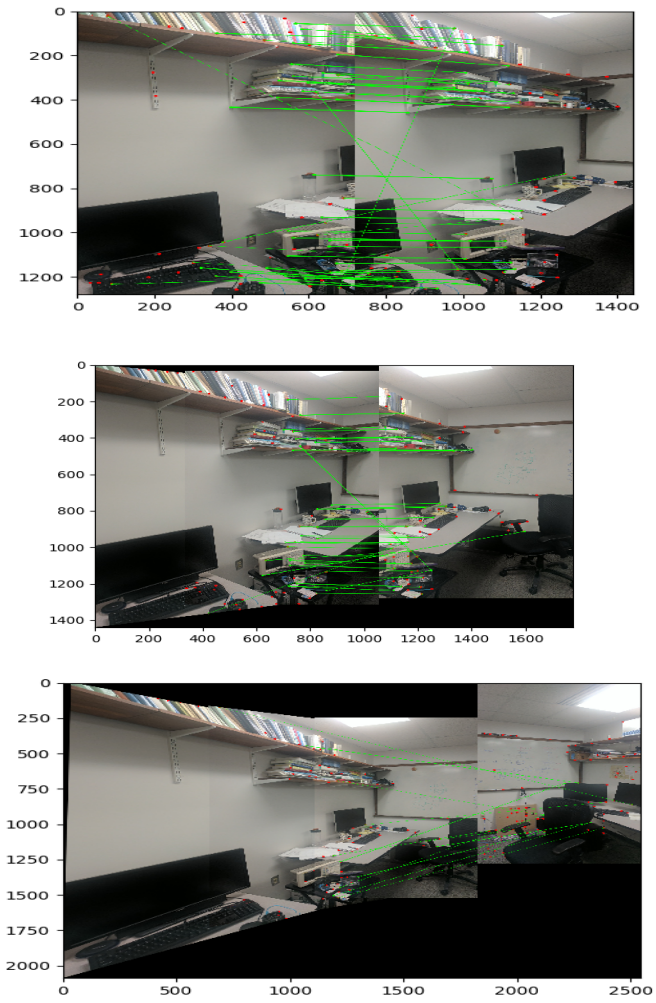
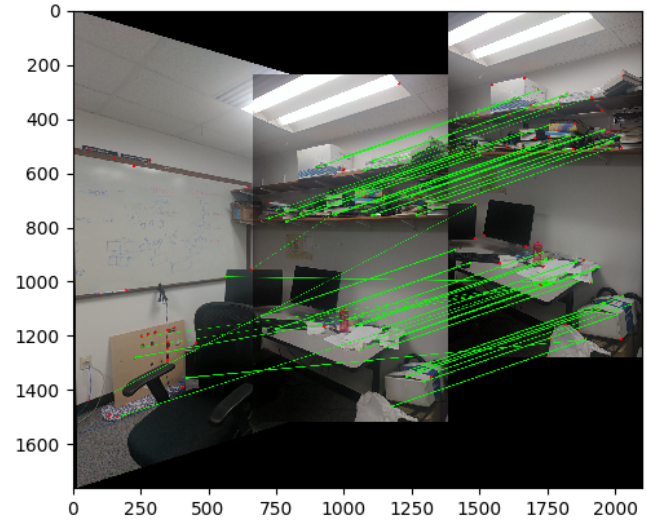


Fig. 25. Feature Matching for Test Set 1





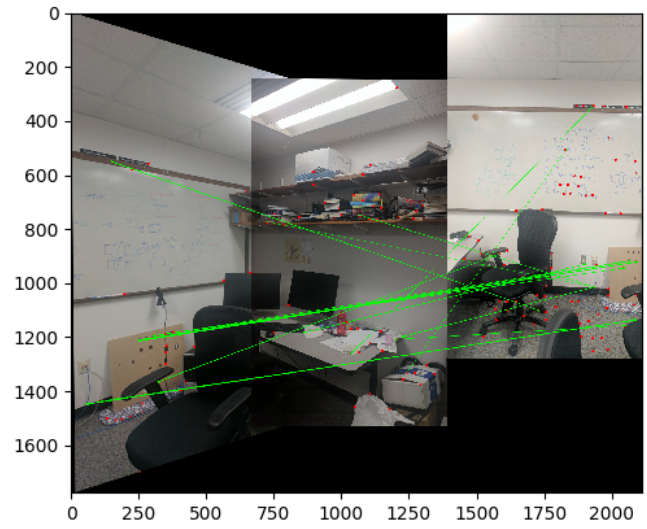


Fig. 26. Feature Matching for Test set 2

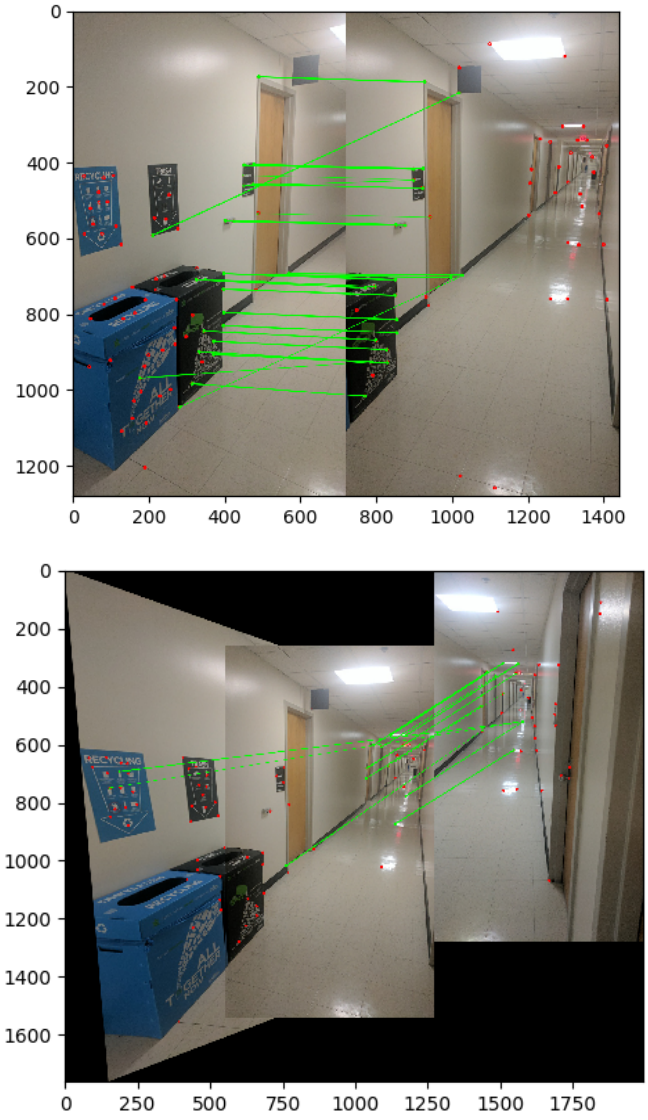
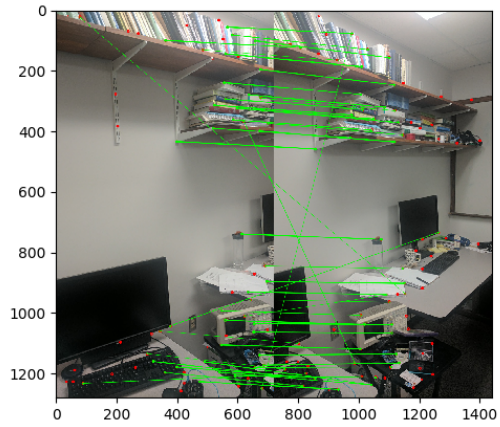


Fig. 27. Feature Matching for Test Set 3



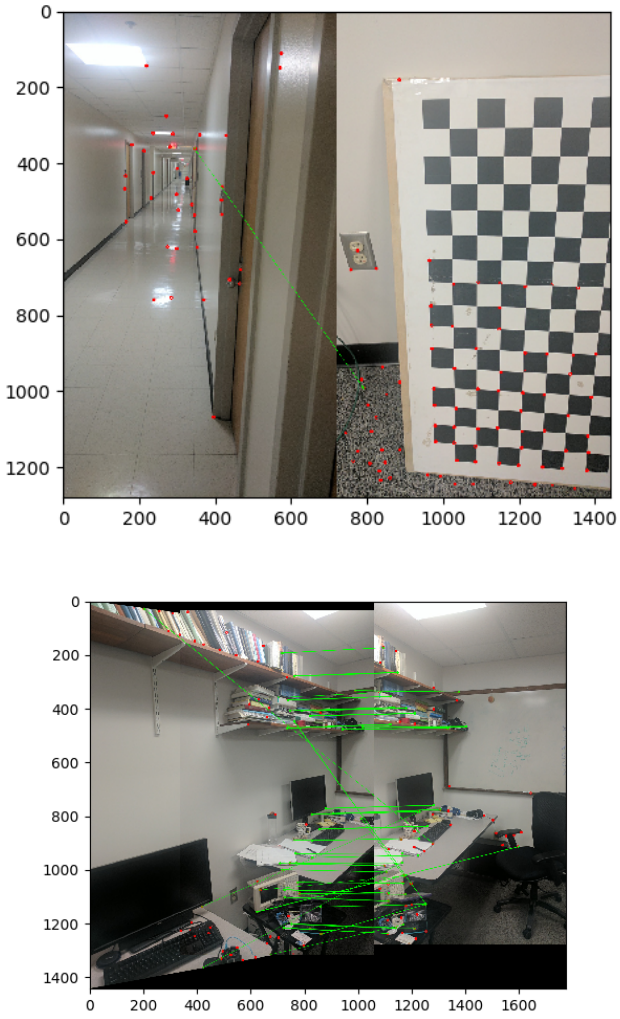
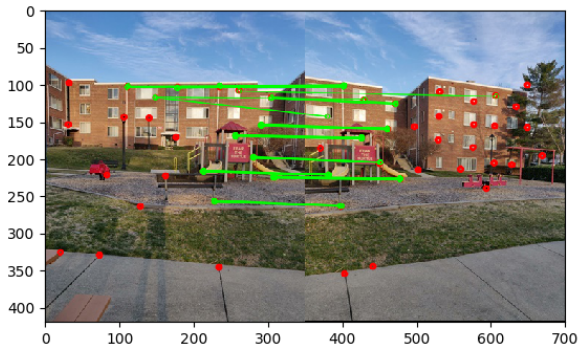
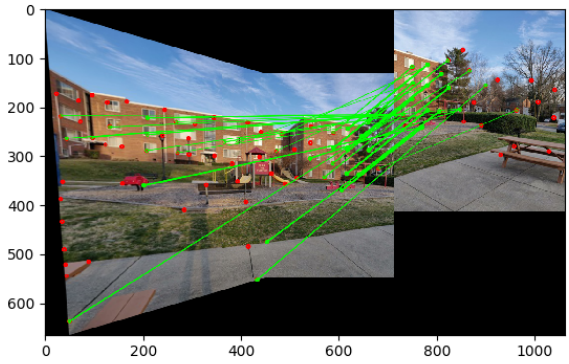


Fig. 28. Feature Matching for Test Set 4





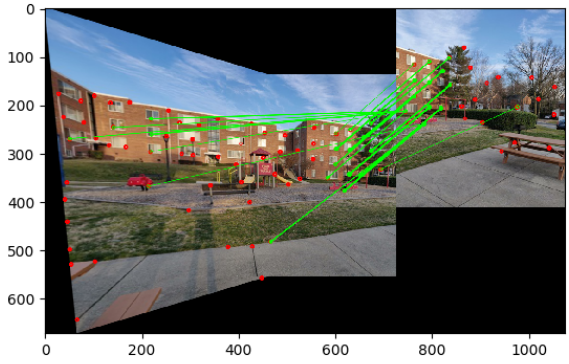
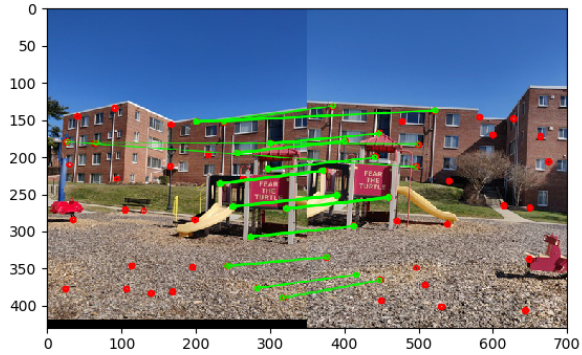
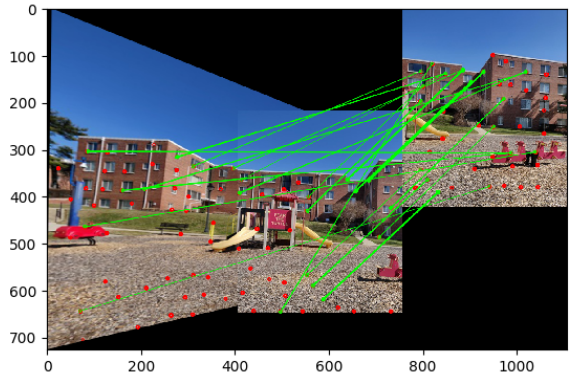


Fig. 29. Feature Matching for Custom Set 1.





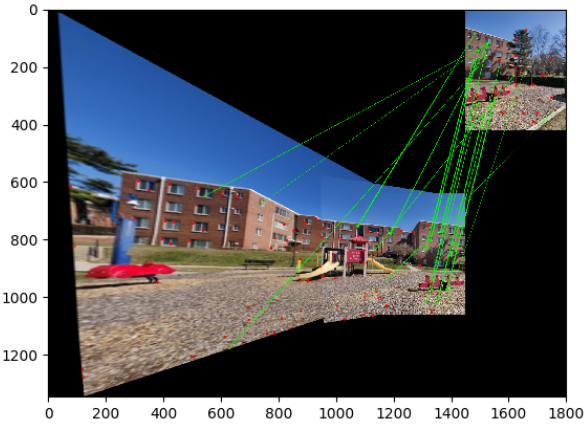


Fig. 30. Feature Matching for Custom Set 2.

E. Random Sample Concensus (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). To remove incorrect matches, we will use a robust method called Random Sample Consensus or RANSAC. We follow the following steps, select four feature pairs at random from both the images, compute homography between these pairs, compute inliers whose sum of square difference is less than 1000 (our threshold) and then we repeat this steps for 700000 iterations, keeping the largest set of inliers. We again compute homography with this set of inliers to achieve the best results. If the the number of inliers after RANSAC is less than 30 percent of the matched features before RANSAC, we reject that pair of image since we need a good amount of matches for warping.

The output for RANSAC is shown figure 31 to 39.

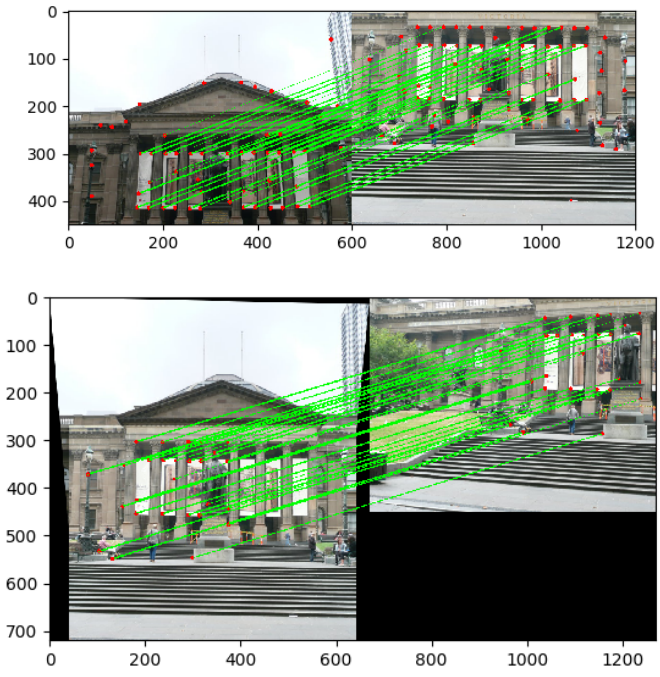


Fig. 31. Feature Matching after RANSAC for Train Set 1

F. Warping and Blending Images

Now that we have computed the homography matrix ,we perform perspective transform **(cv2.perspectiveTransform)** and then apply translation for the images to overlap. We then use Warp Perspective **(cv2.warpPerspective)** to stitch the images.

(We read the images in sequence and stitch two images first and then repeat the whole pipeline with the stitched image and next input image.)

When blending these images, there are inconsistencies between pixels from different input images due to different exposure/white balance settings or photometric distortions or

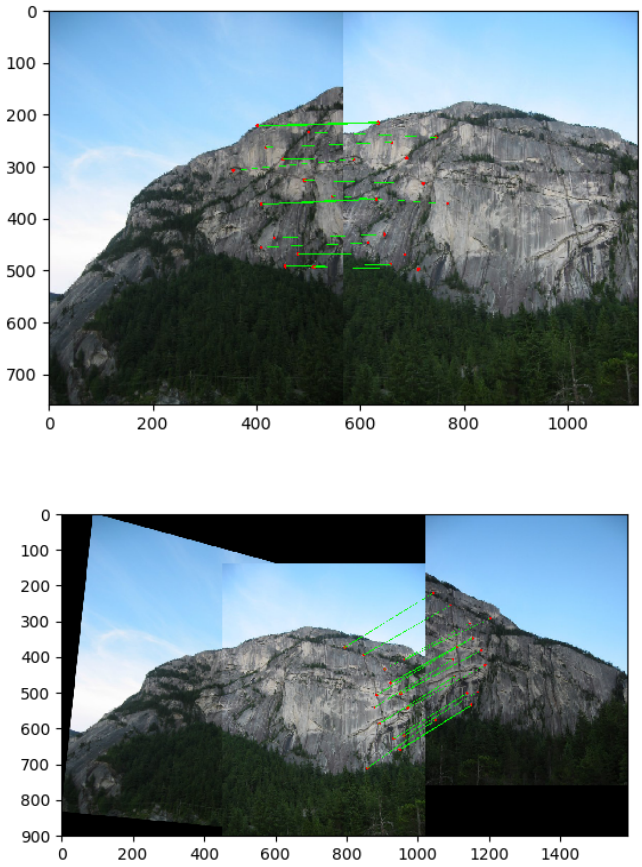


Fig. 32. Feature Matching after RANSAC for Train Set 2

vignetting. This can be resolved by Poisson blending or any other form of blending where you average out the pixels, so that the panorama doesn't look like overlap of three multiple images, but just one single image.

The final outputs are presented from figure 40 to 48.

G. Issues and their Solution

In some cases, two consecutive input images do not share a lot of correspondences, which makes it difficult for our algorithm to find a homography matrix between them. Since our one assumption is that the input images come in sequence (e.g. from left to right), if we have to drop one image because of not having enough matches, our algorithm will also have to drop all the previous images. These issues can be observed in Test Set 1 (fig.34) and Train Set 3(fig.33). For example, we have N = 7 images, and image 2 and 3 do not share enough correspondences. In this case, we have to drop image 1 and 2 (breaking the chain from 1 - 7 into 3 - 7) and only stitching the images from 3 to 7.

H. Results and Conclusion

In this phase we created a panorama using traditional approach. Our algorithm give us very good results. For further improvements, we can consider blending techniques such as

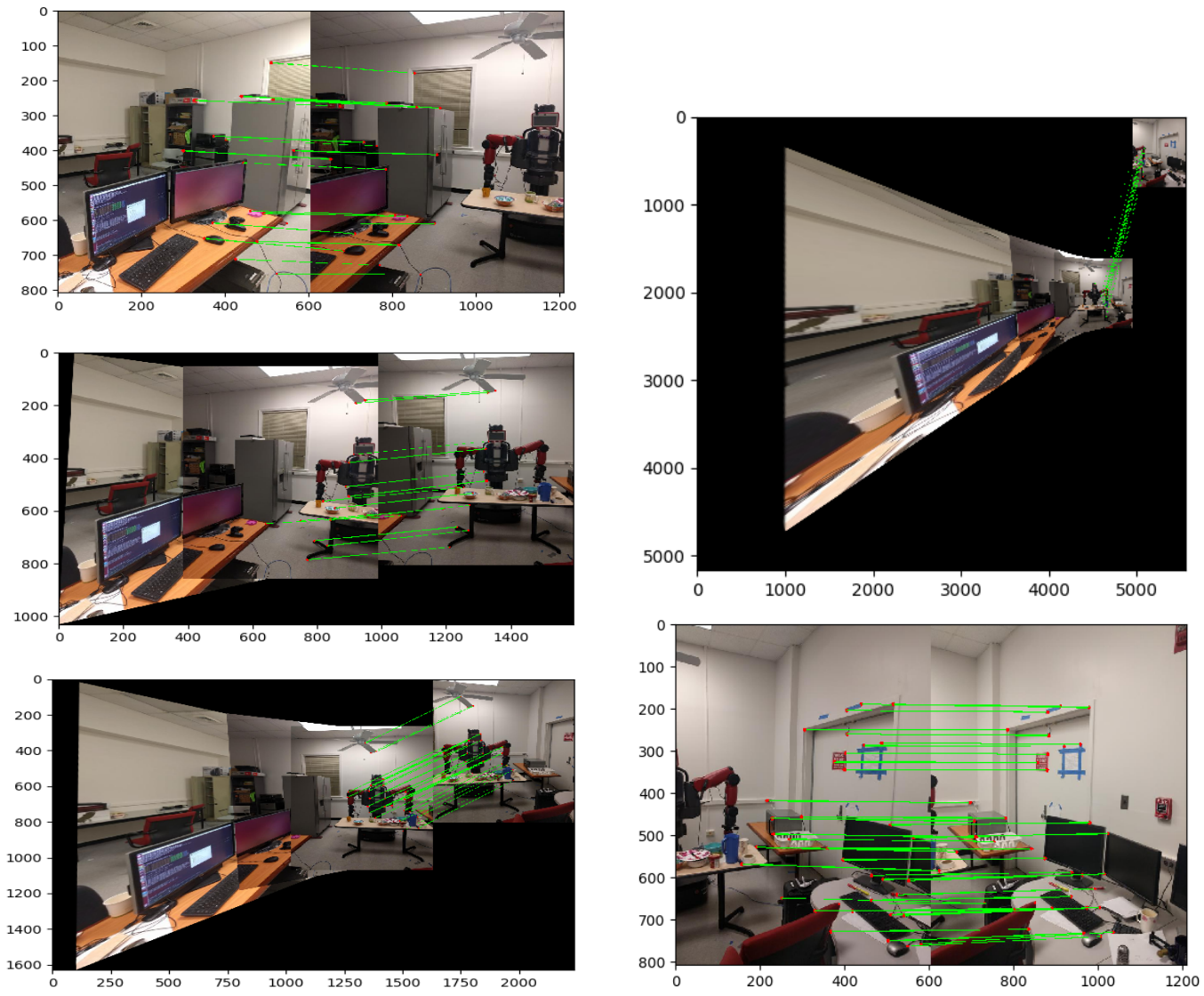


Fig. 33. Feature Matching after RANSAC for Train Set 3

Poisson or Laplacian or just averaging out the pixel values of the two images at the same location.

The stitching algorithm fails for images with less overlapping area or no matches at all. Still, the algorithm is robust enough to stitch all the sets, and can be made more accurate by tweaking the parameters involved. The parameters we choose right now are arbitrary and works well with most of the sets.

For certain cases , such as Test4 , the matched points are less and after RANSAC it reduces even further , since there's no overlapping in those regions. So we get no stitched image over there. Thus, our algorithm is robust enough to handle such cases and reject such images during the stitching process.

II. Phase 2 : Deep Learning Approach

The following section explains homography estimation using deep learning methods. We have implemented two techniques in this section namely-"supervised" and "unsupervised". In the initial stages both the techniques require

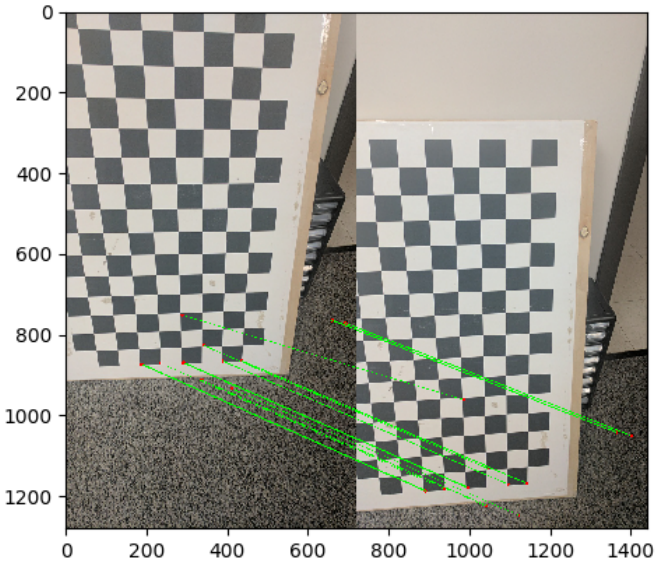
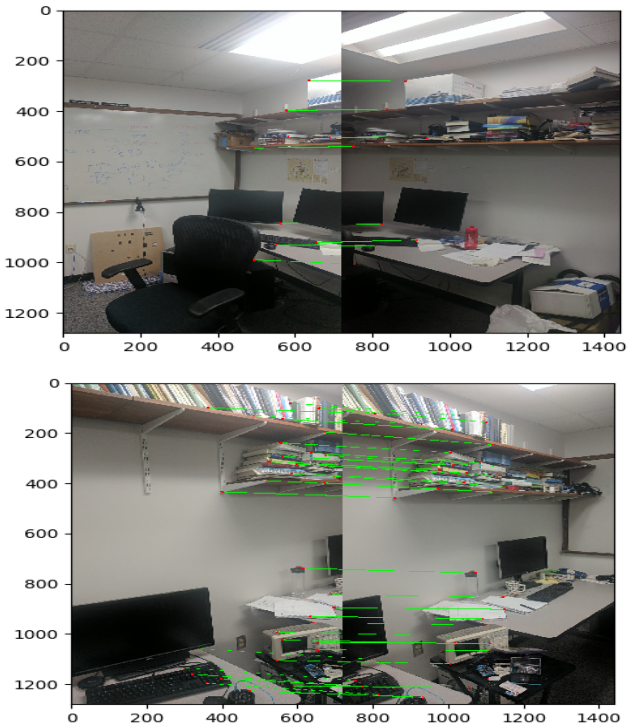


Fig. 34. Feature Matching after RANSAC for Test Set 1



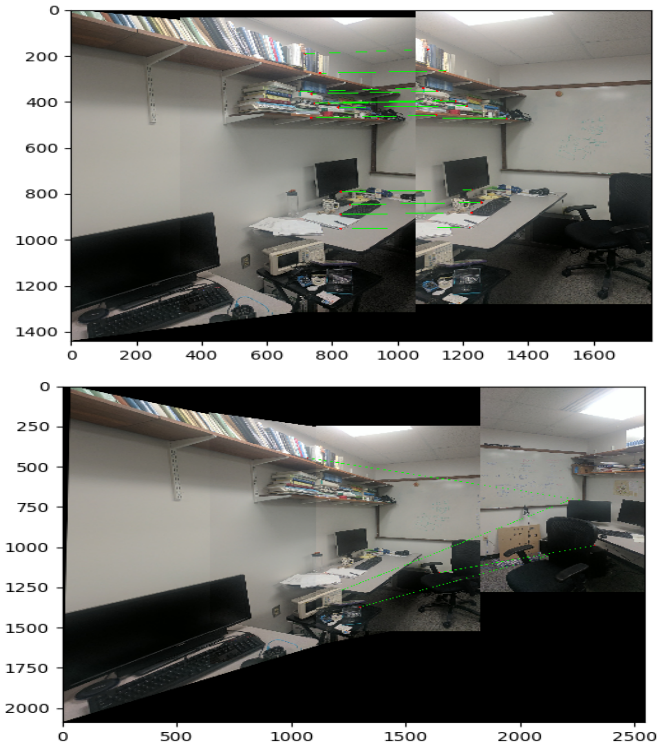


Fig. 35. Feature Matching after RANSAC for Test Set 2

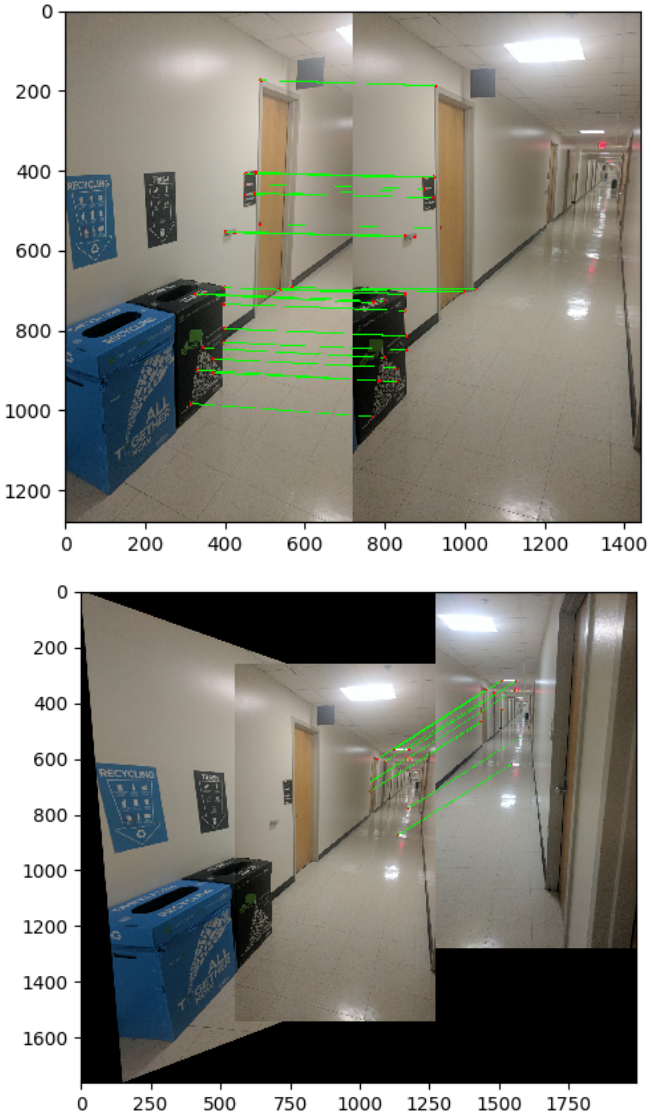


Fig. 36. Feature Matching after RANSAC for Test Set 3

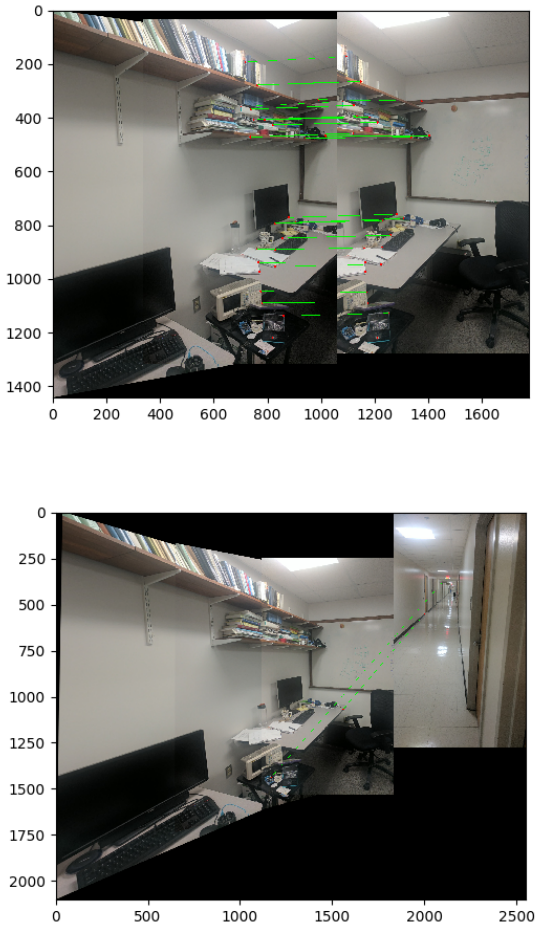
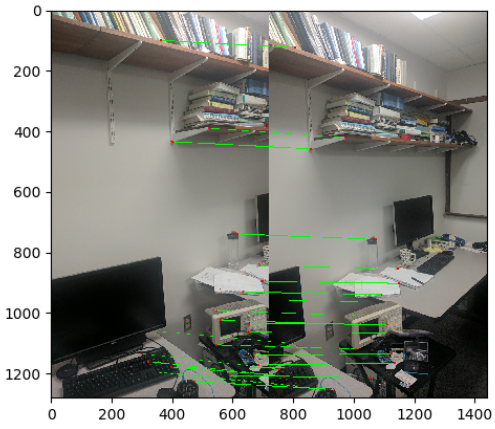
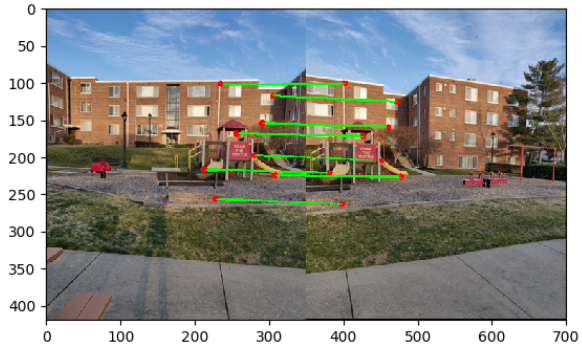


Fig. 37. Feature Matching after RANSAC for Test Set 4





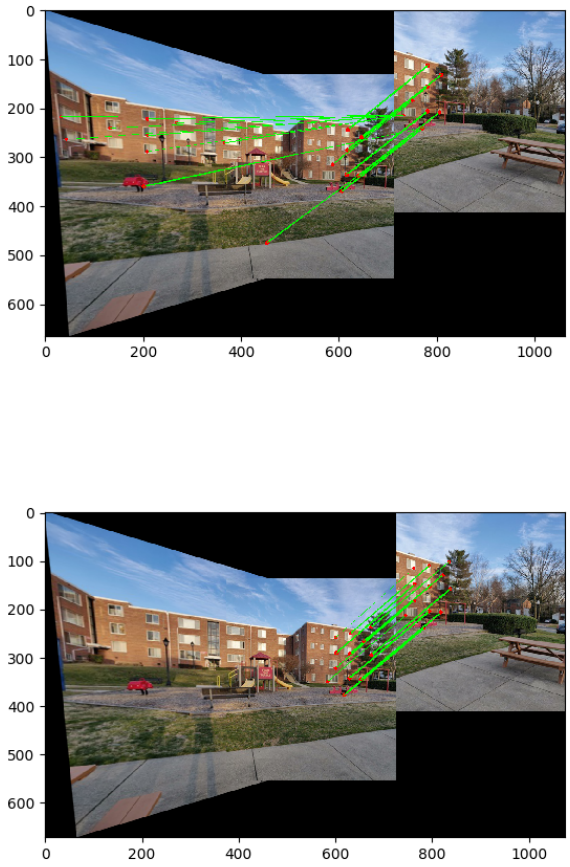
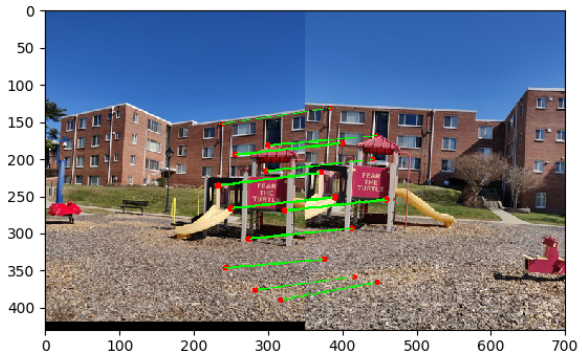


Fig. 38. Feature Matching after RANSAC for custom Set 1



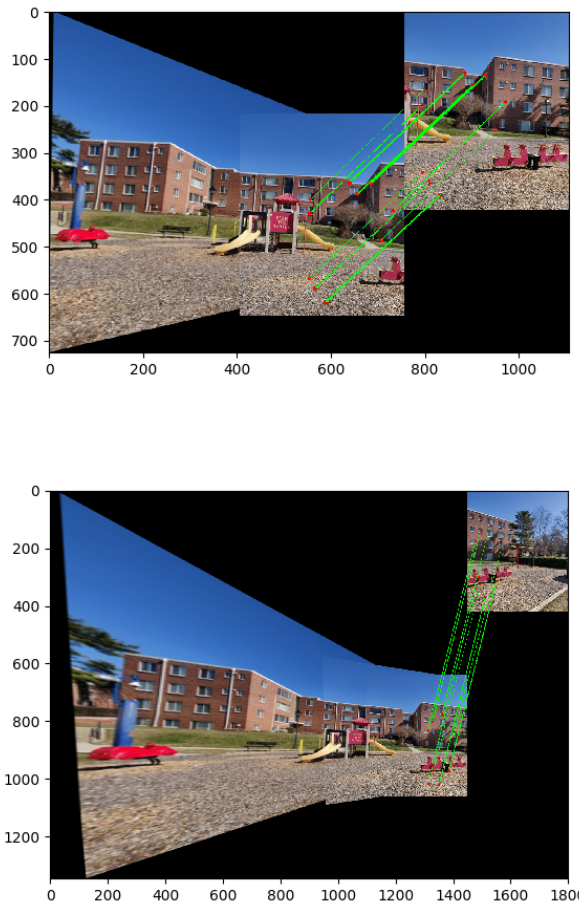


Fig. 39. Feature Matching after RANSAC for Custom Set 2

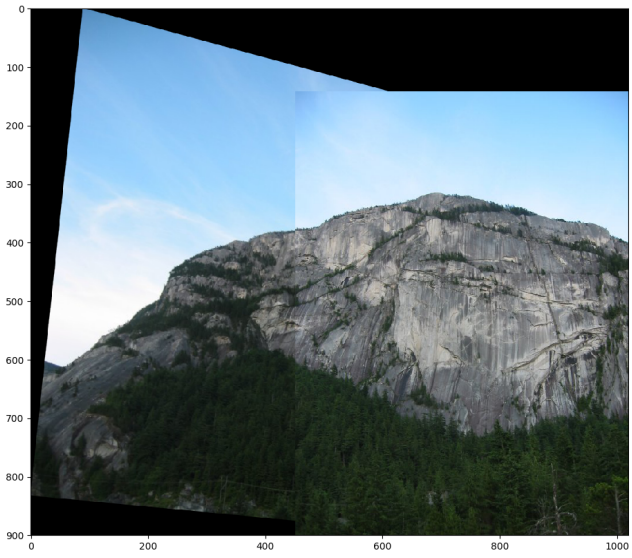
synthetic data-set which we explain how to generate in the following subsection.

A. Synthetic Data Generation

The deep learning methods require synthetic data-set which is generated as follows.

We take an image, standardize it, convert it to gray-scale and then resize it to dimensions (320, 240). Then we randomly generate a patch in the image of size (128, 128) as mentioned in the research paper [1], let's call it patch A. Now this patch's four corners are randomly perturbed(by a factor of 32, as per [1]) to generate a distorted version of the patch, let's call it patch B. The above mentioned patches can be seen in the figure 49 which shows patch A in blue color and the perturbed patch B in green. Now, the next step is compute homography between the two patches. We found it using cv.getPerspectiveTransform() function of opencv. Once we have the homography from patch A to B we compute inverse of it using np.linalg.inv() function to find the corresponding homography from B to A. This homography matrix is then used to warp the entire image containing the two patches to

Fig. 40. **Results for Train Set 1**



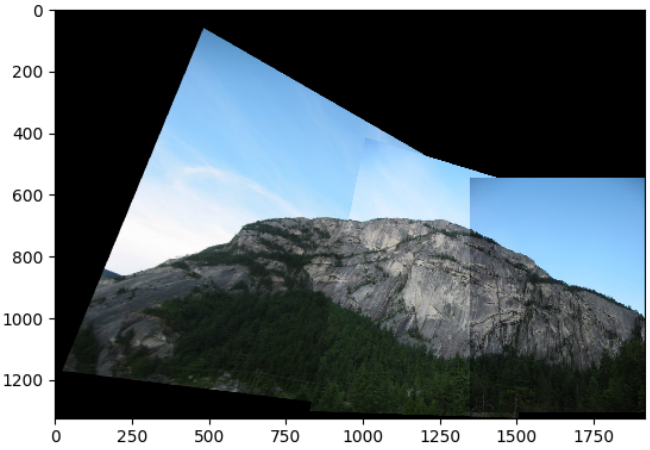
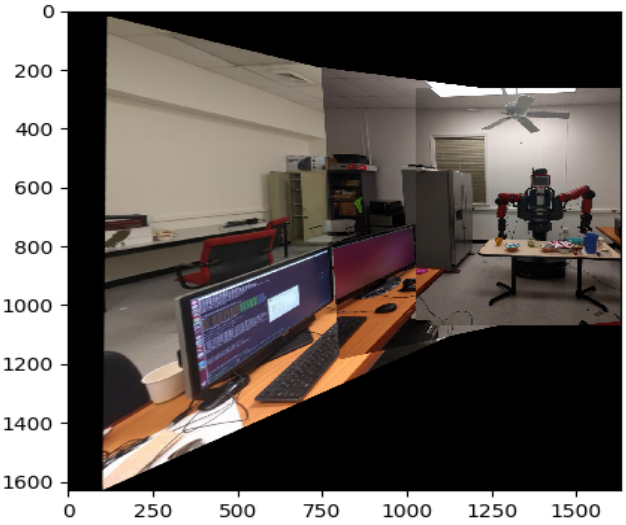


Fig. 41. **Results for Train Set 2**



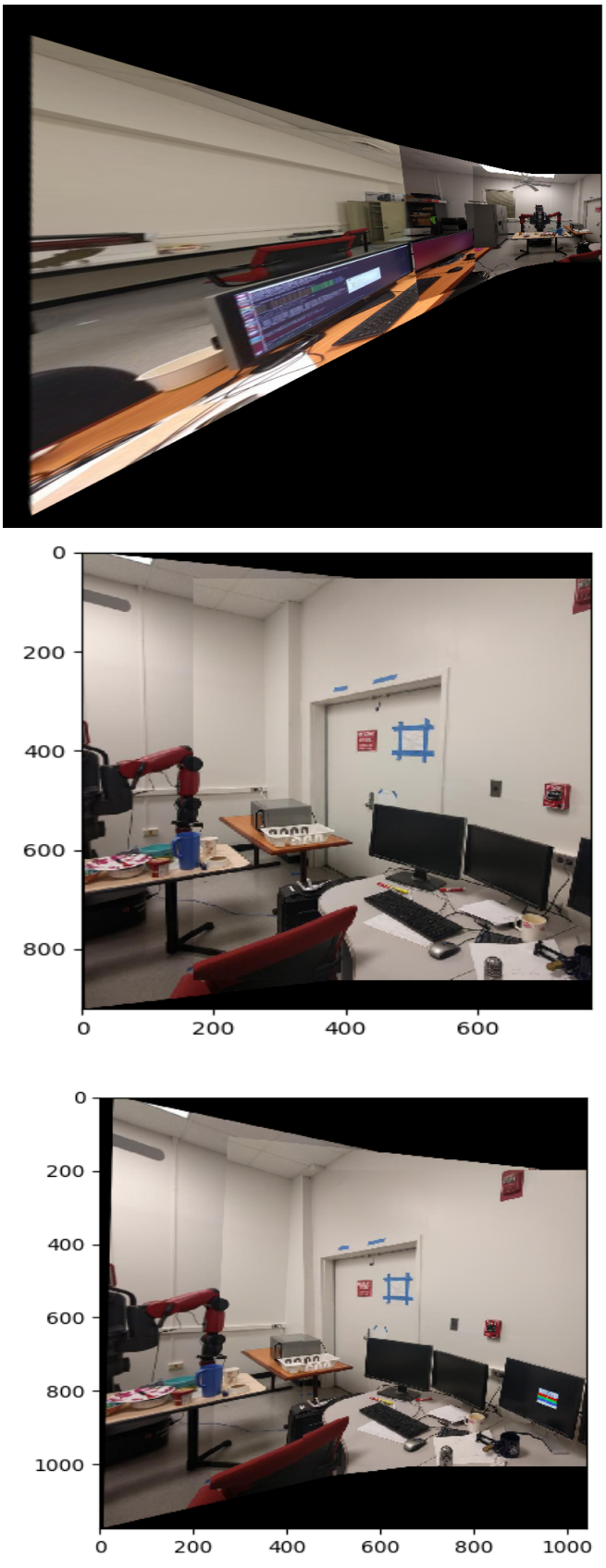


Fig. 42. **Results for Train Set 3**

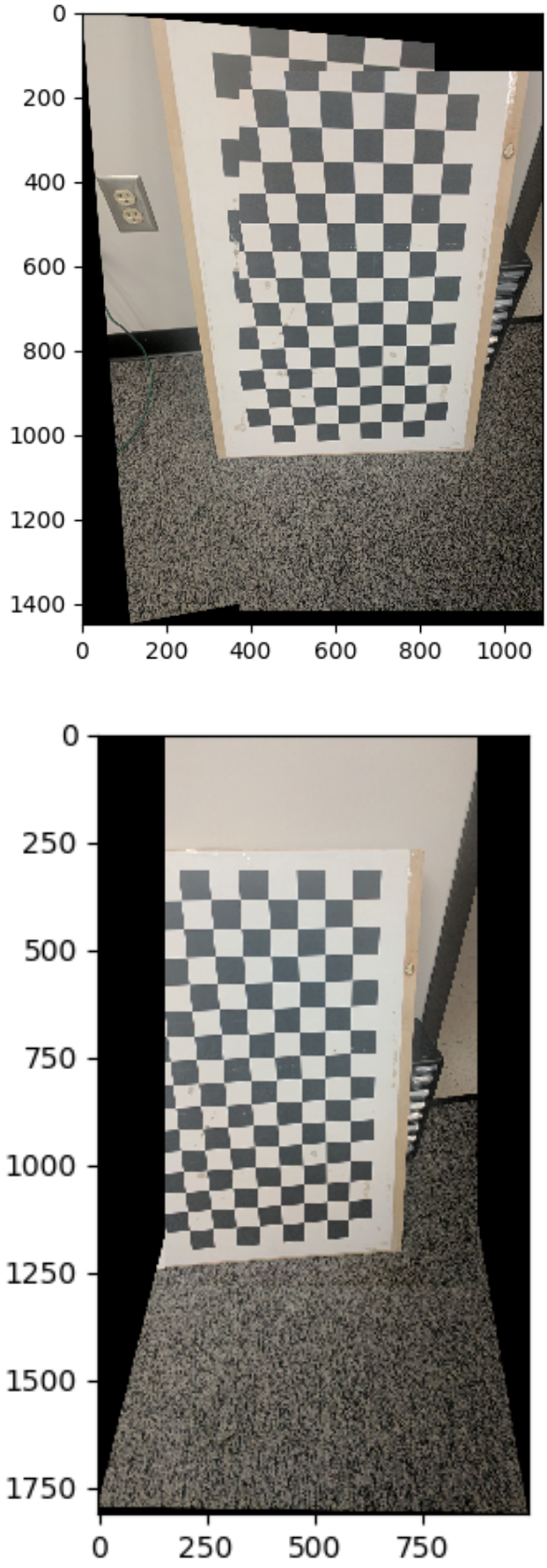
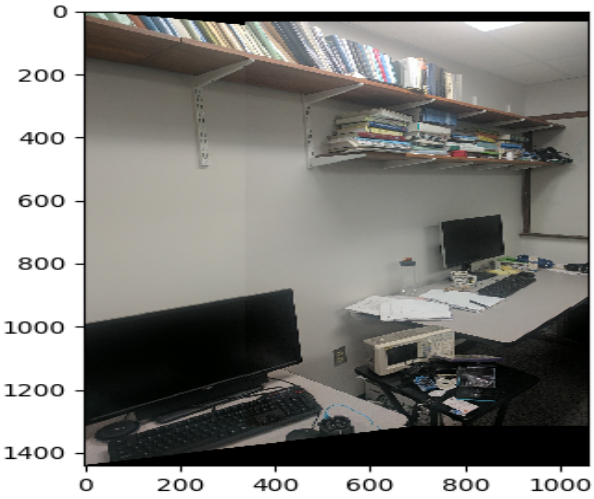


Fig. 43. **Results for Test Set 1**



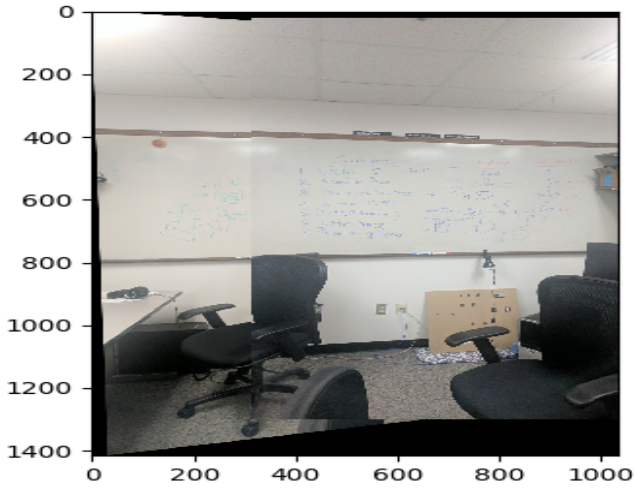
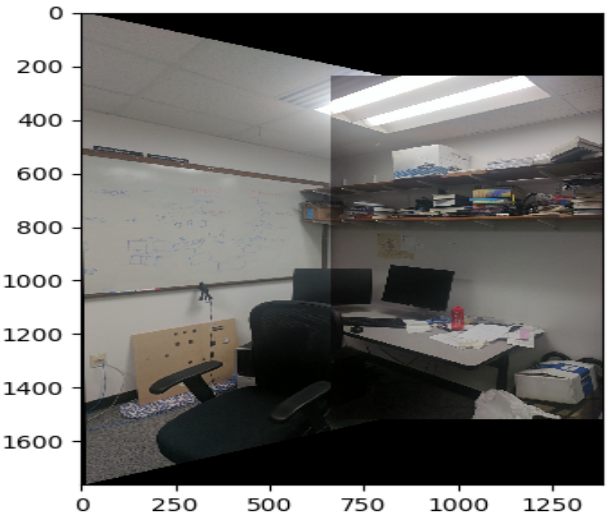


Fig. 44. **Results for Test Set 2**



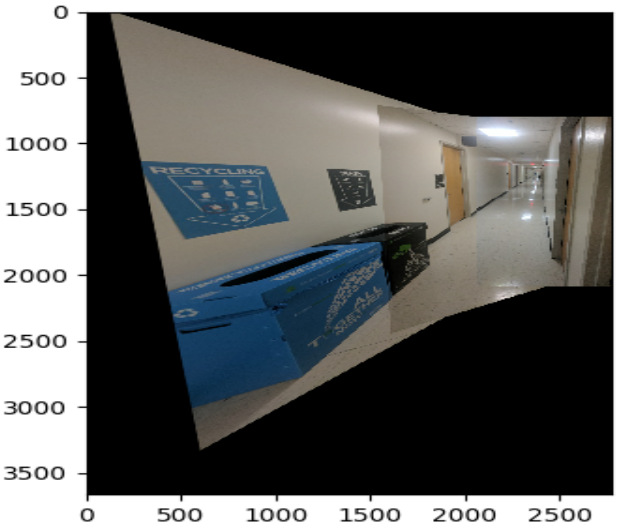


Fig. 45. **Results for Test Set 3**

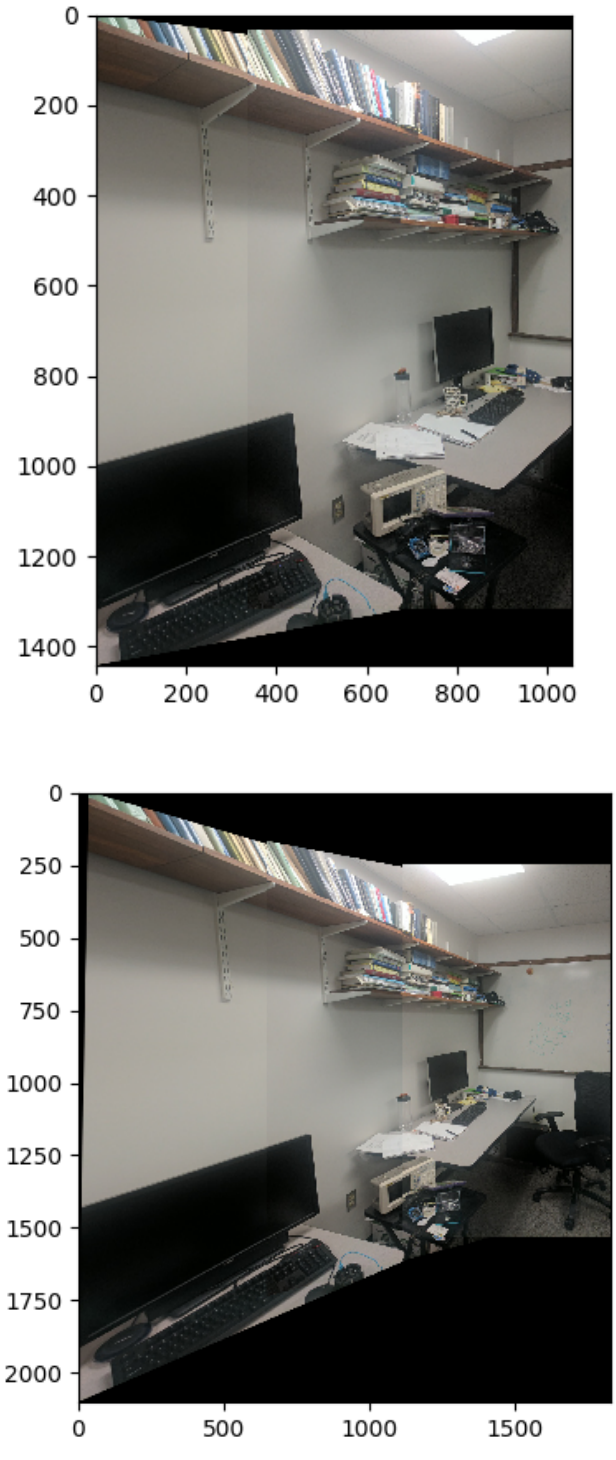


Fig. 46. **Results for Test Set 4**

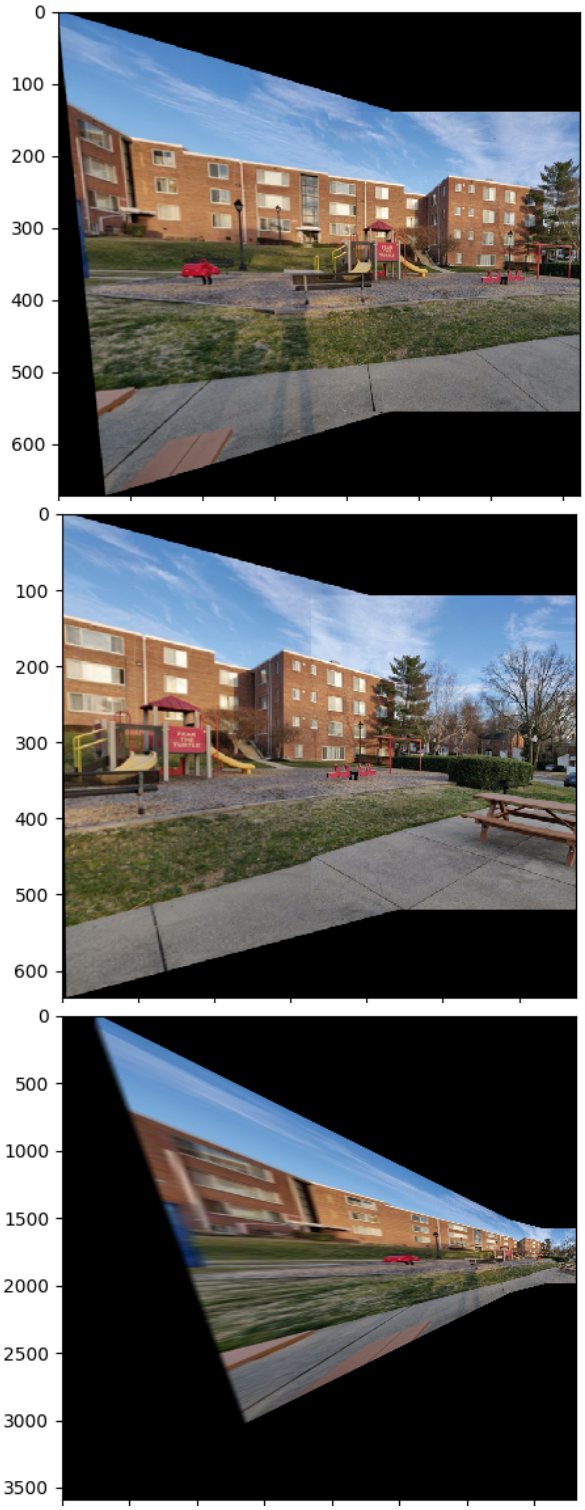


Fig. 47. **Results for custom Set 1**

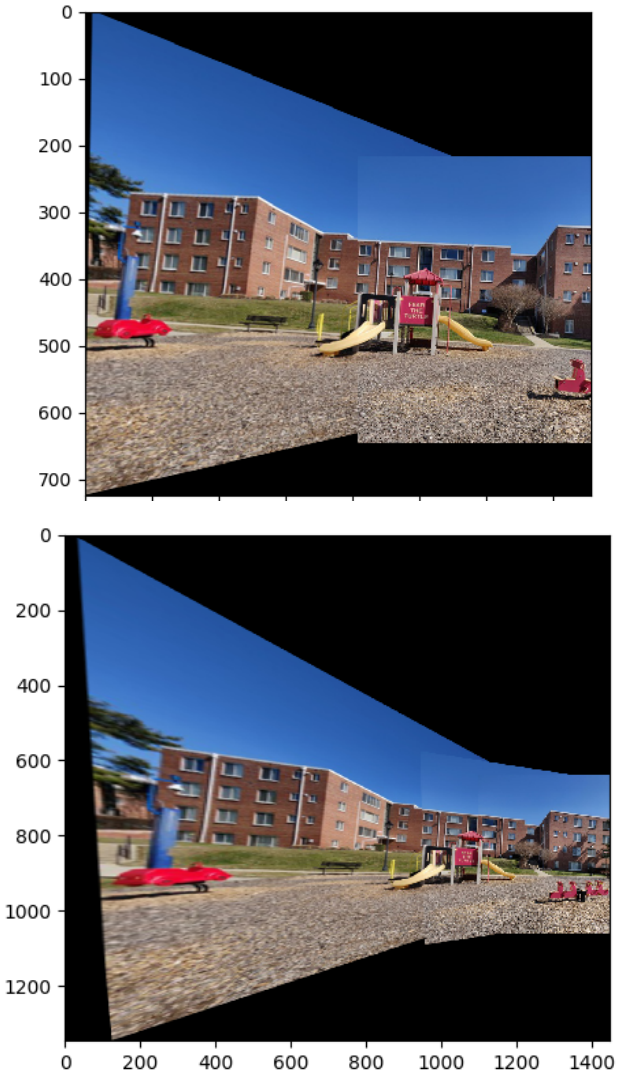


Fig. 48. **Results for custom Set 2**

form a warped image. The final step is to take the warped image and crop the original coordinates of patch A to get a transformed version of patch A. The the final two patches can be seen in figure 49. These two patches emulate two images that have been taken from different angles and contain decent correspondences and/or similar objects of interest. These two patches are then fed to the two models as input images of shape (128, 128, 2).

1) Data-set for Supervised: The above procedure represents patch generation for one image. We have employed an on-the-go data generation procedure where we generate the data-set dynamically during training with a size of 50,000 images per epoch in case of supervised learning. This is done by generating 10 random pairs of patches per image and we feed 10 such images per iteration thus generating 100 images in one iteration. In total there are 500 iterations per epoch hence generating 50k images per epoch. We employed such

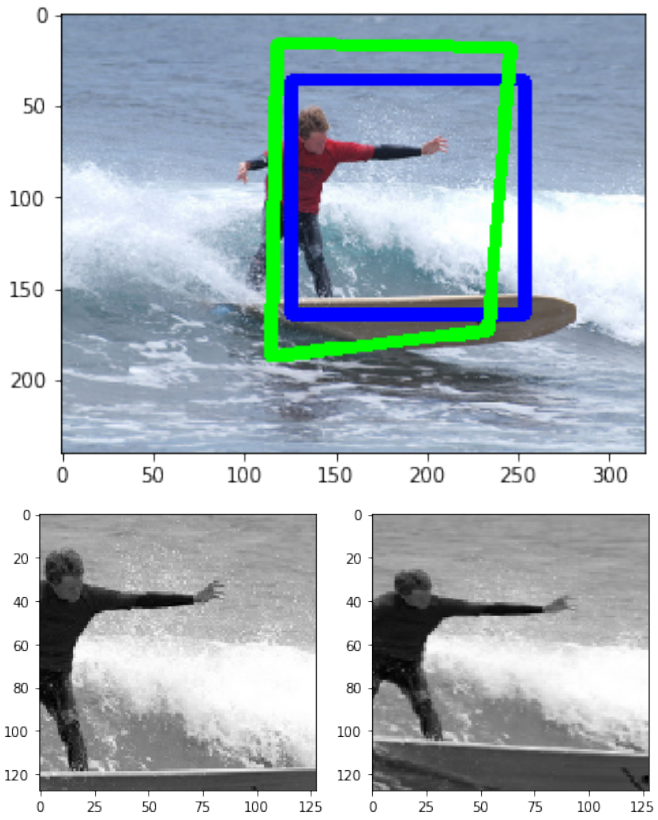


Fig. 49. Synthetic Data Generation

a method to avoid any over-fitting that can be caused by creating data-set before hand and using them in training.
The same scheme was used on the validation set as well in order to compare the performance of the model on other types of images as well and decide how good was the fit.

2) Data-set for Unsupervised: We again use the on-the-go generation method for this method but have limited our training samples to 5000 since it was mentioned in the paper [2] that the unsupervised approach requires comparatively less data to train itself. In this case we generate only one pair of patches per input image. We have used a batch size of 32 while training our unsupervised model.

III. CNN ARCHITECTURE

The architecture we used for both supervised and unsupervised models can be seen in the figure 50 and 55. The architecture is inspired from paper [1]. It consists of 8 convolutional layers consisting of filter sizes 64 for the first 4 and 128 for the latter. This is followed by a dense layer of 1024 units. A dropout layer with probability 0.5 was added after the dense layer in order to account for any overfitting. The final layer consists of a dense layer with 8 units. This gives us the H_{4pt} matrix.

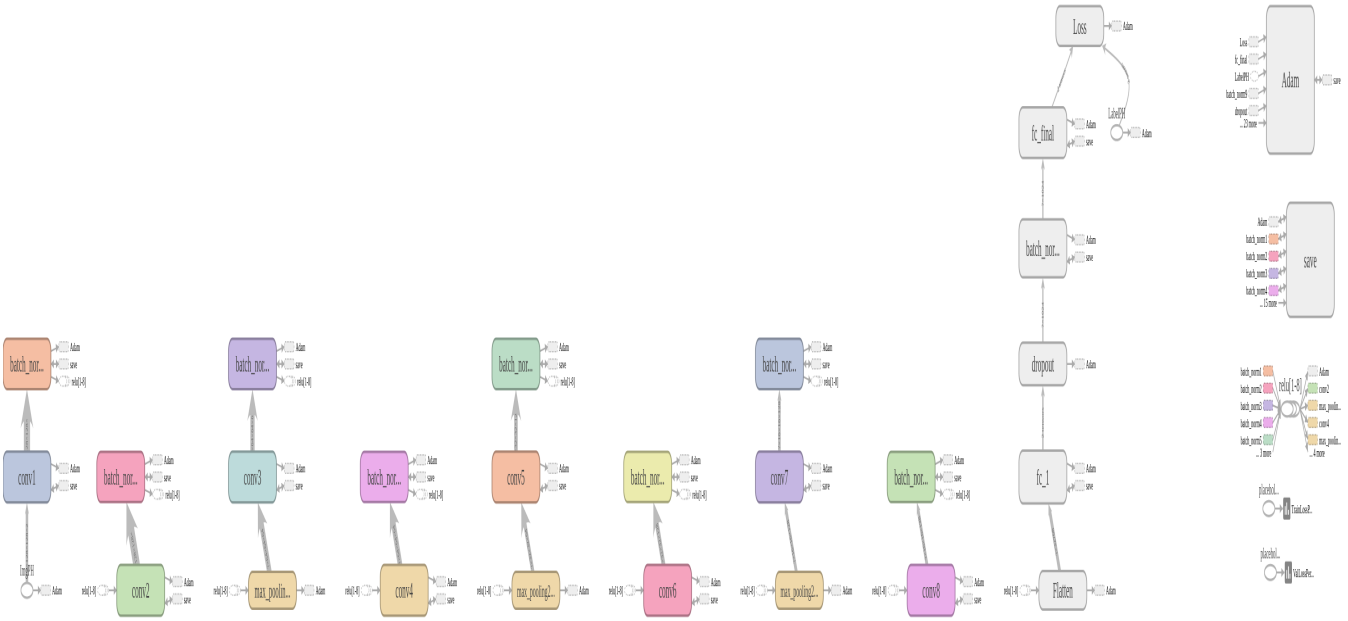


Fig. 50. Tensorflow Graph for Supervised Approach.

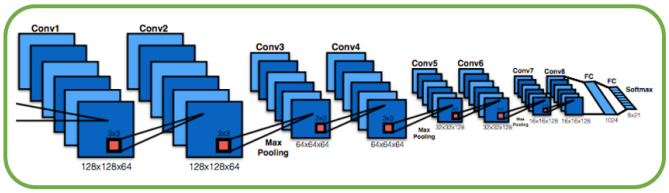


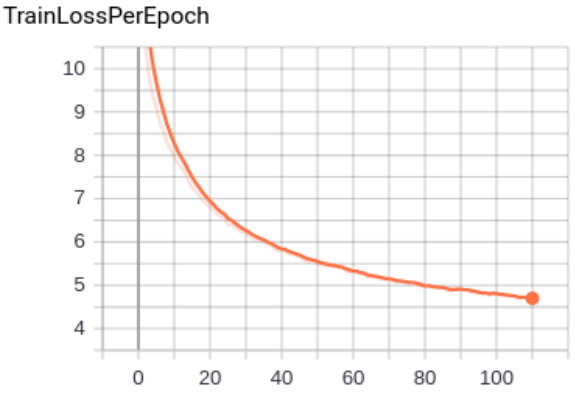
Fig. 51. Overview of the supervised deep learning system for homography estimation. Bottom: Architecture of the network which mimics the network given in paper [[1]].

IV. Supervised Method

The supervised model is trained with the H_{4pt} values as labels which are nothing but the amount of perturbation caused between patches B and A. The loss function which is optimised is the L2 loss between predicted homography and the H_{4pt} values that have been added as groundtruth labels. We used Adam optimiser with a learning rate of 1e-4. Image pre-processing turned out to be an important step as without the scaling of images and H_{4pt} values our loss values were higher and the output was taking too many epochs to reduce and thus eventually converge. By scaling the pixel values between [0,1] and the H_{4pt} values between [-1,1] we observed faster convergence and less obnoxious loss values.

The plots of training and validation losses can be seen in figure 52 . It is evident from the plots that the training and validation loss values are very much similar and comparable thus showing that the model is a good fit. The L2 loss was found to be 4.7 at the end of 110 epochs.

We trained the network for 110 epochs and 55.5k iterations which took us nearly 16 hours to train on a GTX 1070-laptop GPU. Upon observing the plots one can infer that it can trained even further to achieve convergence at some point. Due to limited time and computing capability we decided to halt at 110 since the loss value was acceptable and the output as seen in the figure 53 shows very good results. In the figure 53 green patches represent the ground truth and red patches represent the predicted perturbation. As evident from the outputs the error between predicted and actual perturbations are very minute.



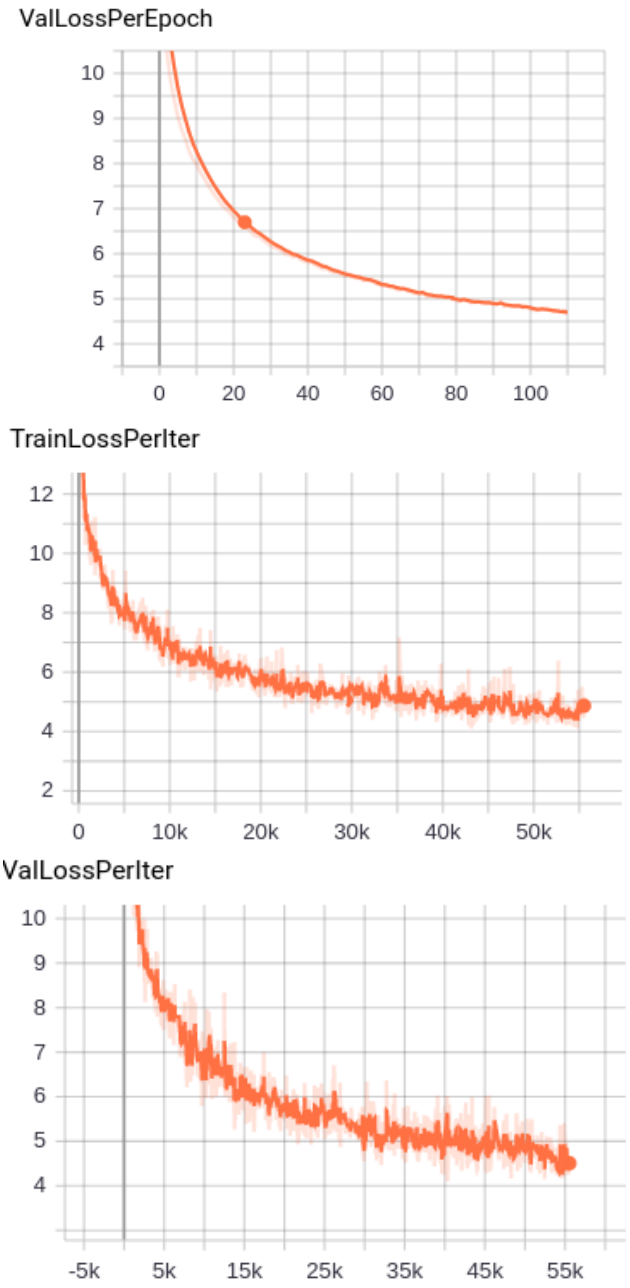


Fig. 52. Train Loss and Validation Loss

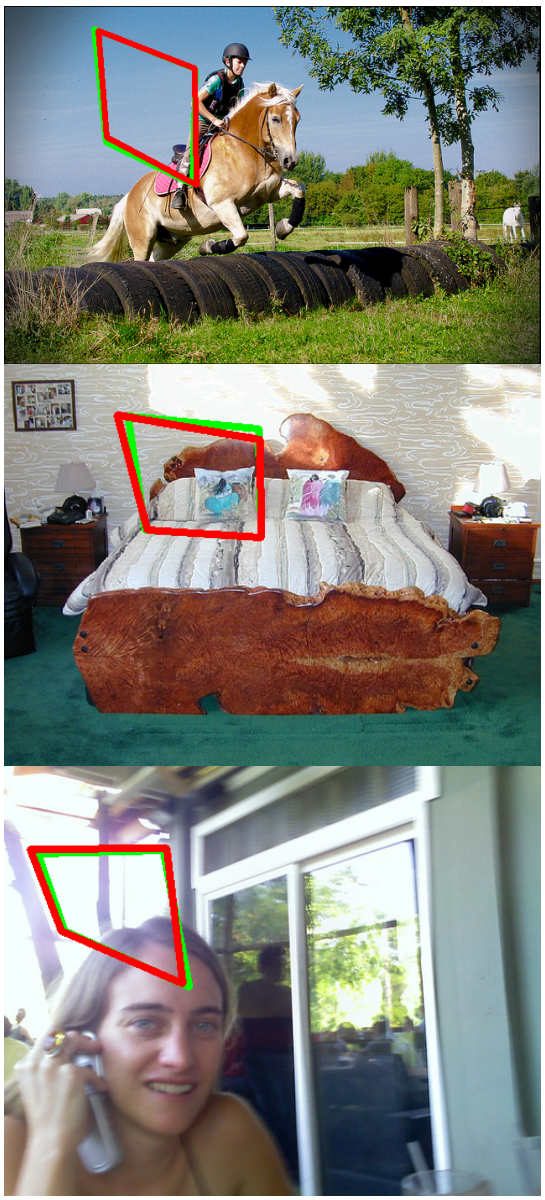


Fig. 53. Green patches represent the ground truth and red patches represent the predicted perturbation.

V. Unsupervised Method

For the unsupervised method we used Adam optimiser with learning rate 1e-5 a batch size of 32. As explained before we have generated data on the go with 5000 samples as our limit. The loss function we had optimised was the L1 loss. We trained the model for 22 epochs after which we observed convergence of training loss at 0.0112. For pre-processing we only performed image scaling and did not scale the H_{4pt} values.

For implementing the TensorDLT we have referred to the author's code (of the paper[2]), reference of which has been provided in [3] of references. The plot of training and validation are shown in figure 56.

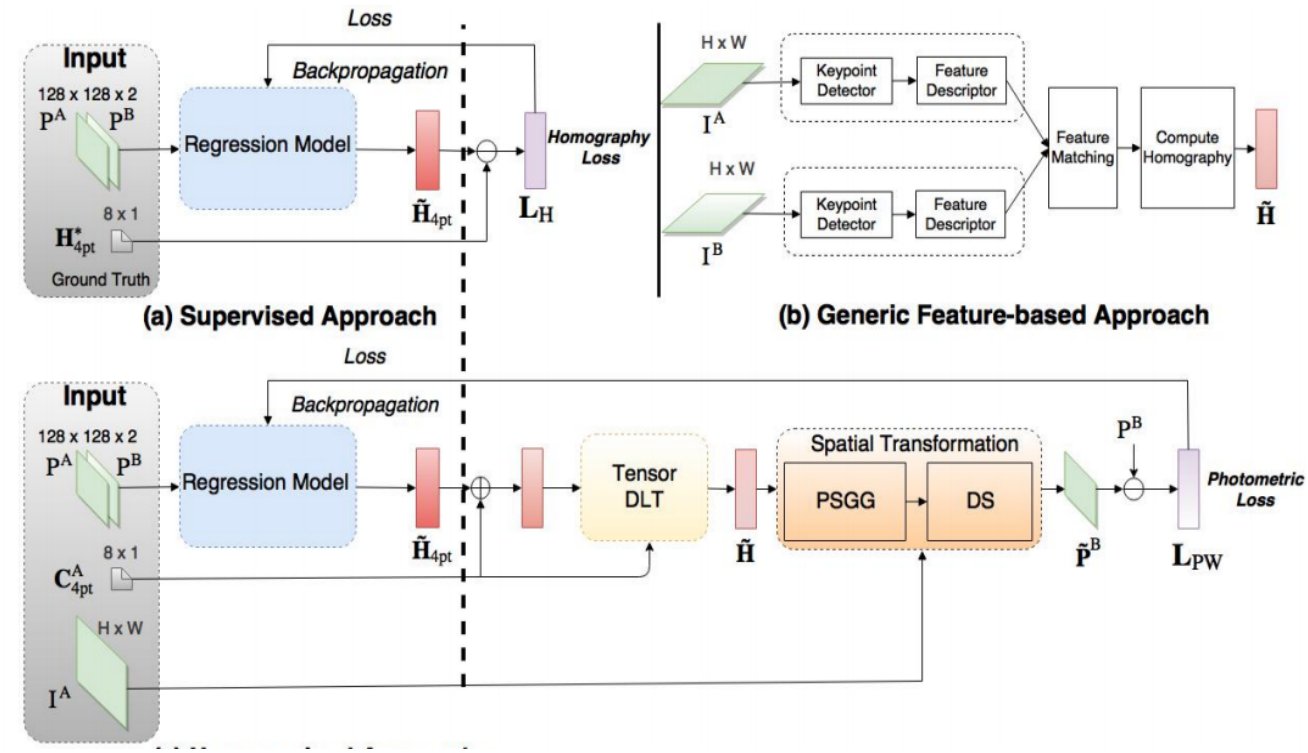


Fig. 54. : Overview of homography estimation methods; (a) Benchmark supervised deep learning approach; (b) Feature-based methods; and (c) Our unsupervised method. DLT: direct linear transform; PSGG: parameterized sampling grid generator;DS: differentiable sampling.

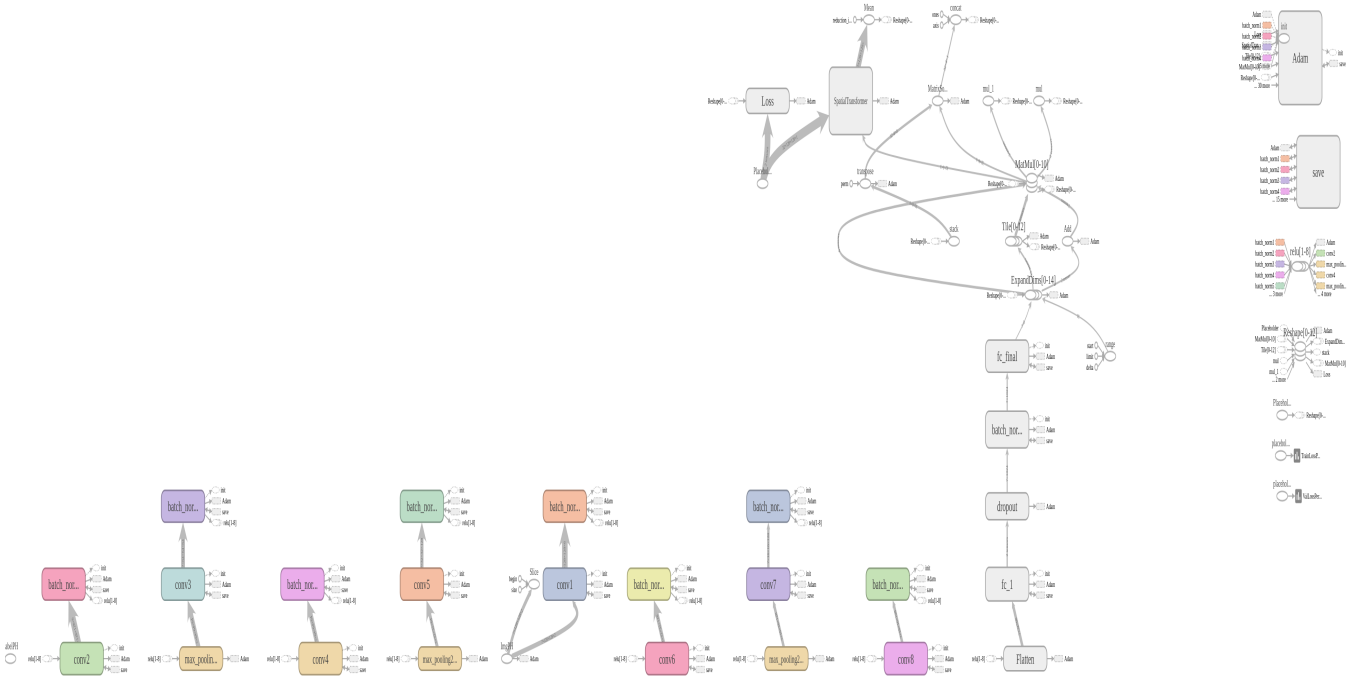


Fig. 55. Tensorflow Graph for Unsupervised Approach.

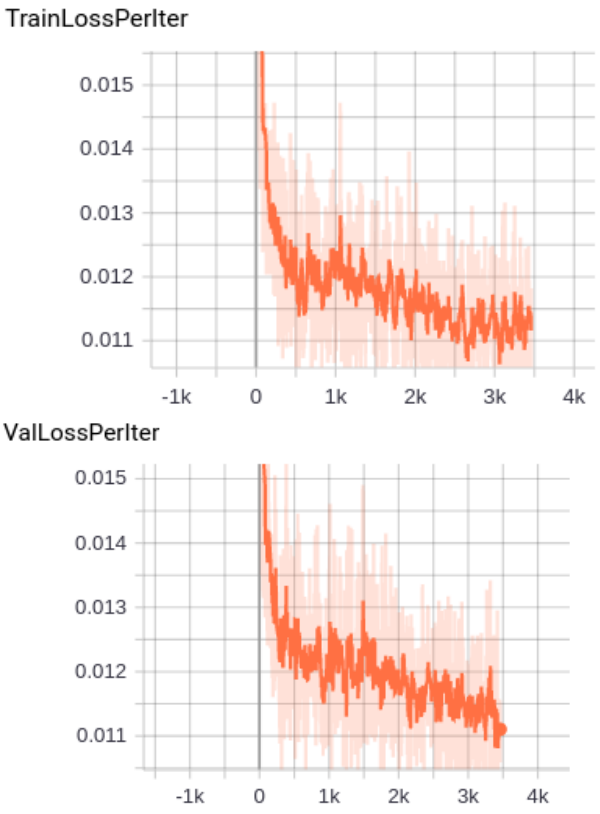


Fig. 56. Training Loss and Validation Loss for Unsupervised Approach.

VI. Conclusion and final thoughts

We have implemented the traditional approach to panorama stitching as well as used deep learning methods to estimate homography. It is evident from the results that the traditional approach works well in most cases but it is difficult to obtain ideal outputs in all cases. For e.g the traditional feature based methods prove to be inaccurate when they fail to detect sufficient keypoints, or produce incorrect keypoint correspondences due to illumination changes.

Deep Learning approach are more robust compared to traditional approaches when it comes to illumination variation. Supervised learning methods has promising results but can be limited in scope since it requires ground truth labels. Unsupervised approach does not have that drawback.

We were not able to satisfactorily generate output for our implementation of tensorDLT since we were out of time. Thus, in order to understand the process thoroughly we referred to the author's implementation and trained our model.

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

[3] https://github.com/tynguyen/unsupervisedDeepHomographyRAL2018

[4] Gaussian Derivative : J. Shi and C. Tomasi, "Good features to track," Cornell University, Tech. Rep., 1993.

[5] C. G. Harris, M. Stephens et al., "A combined corner and edge detector." in Alvey vision conference, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[6] https://cmsc733.github.io/2020/proj/p1/