Project 1- MyAutoPano
CMSC733

Yi-Chung Chen
Master of Engineering in Robotics
University of Maryland
College Park, MD
ychen921@umd.edu
Use 2 Late Days

Ji Liu
Master of Engineering in Robotics
University of Maryland
College Park, MD
liuji@umd.edu
* Not registered in this class

Shreyas Acharya
Master of Engineering in Robotics
University of Maryland
College Park, MD
shrey1s0@umd.edu
Use 2 Late Days

***Abstract*—This project report was composed of two parts. In Phase 1, we implement panorama stitching by using the traditional approach. In phase 2, two deep learning approaches was implemented to estimate the homography between two images. The details and the results will be shown in the following sections.**

I. Phaes 1: Traditional Approach

In this section, we will present the details of the implementation of the panorama stitching. We first find the corners in two images. Second, we implement ANMS (Adaptive Non-Maximal Suppression) to find the local maximum corners in images and extract the features. Then we can find the matching features between the two images. In addition, in order to find the best homography matrix for wrapping and stitching two images, we have implemented RANSAC to find the best homography and rejected outliers (features). At last, stitch two images. The overview of panorama stitching using traditional method is shown in figure 1.

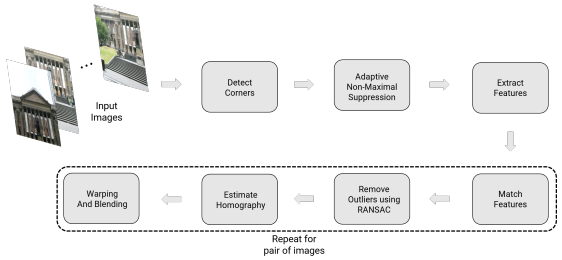

Fig. 1: Overview of Traditional Panorama Stitching

A. Corners Detection

In this project, we use Shi-Tomasi as our corner detector by using *cv2.goodFeaturesToTrack*. Shi-Tomasi corner detection is an algorithm that modified the Harris detection and has a better performance. The results of training and testing are shown in figures 2 to 10.

B. Adaptive Non-Maximal Suppression (ANMS)

In order to avoid crazy wrapping, the ANMS algorithm helps us find the strong corners. The ANMS will find the local maximum of corners in every region and give us the N_{best} best corners. The figures 11 to 19 show the results of our corners after filtering by ANMS algorithm.

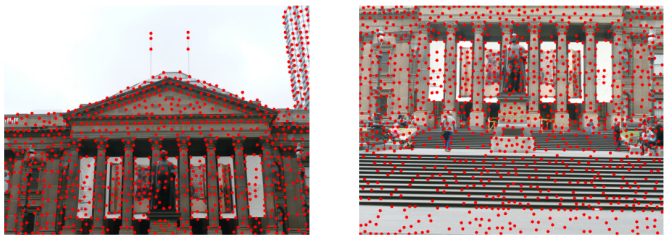

Fig. 2: Corners Detection in Train Set 1

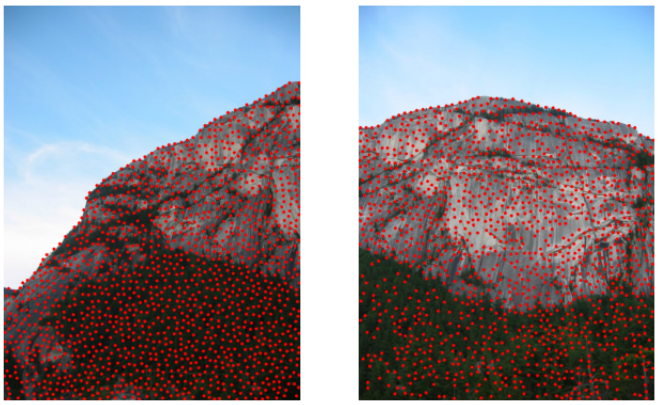

Fig. 3: Corners Detection in Train Set 2

C. Feature Descriptor

From the previous step, we have the x-y coordinates of every feature point. Now we need to encode the information of feature points by vectors. We first extract a patch of size 40x40 centered around the feature point, then apply Gaussian blur and sub-sample the blurred output to size 8x8. Finally, reshape the 8x8 patch to a 64x1 vector and standardize it to zero means. These vectors were called feature descriptors. The figures 20 to 28.

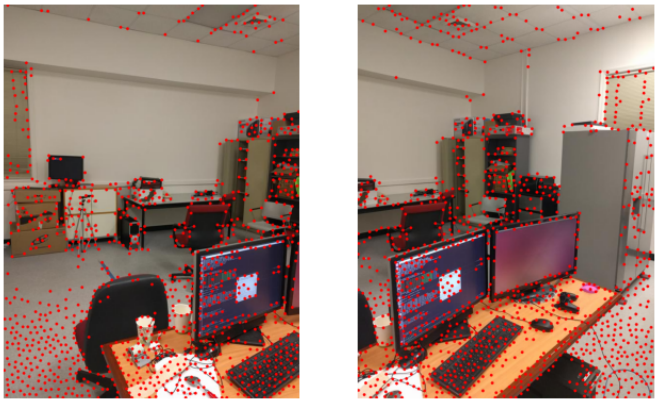

Fig. 4: Corners Detection in Train Set 3

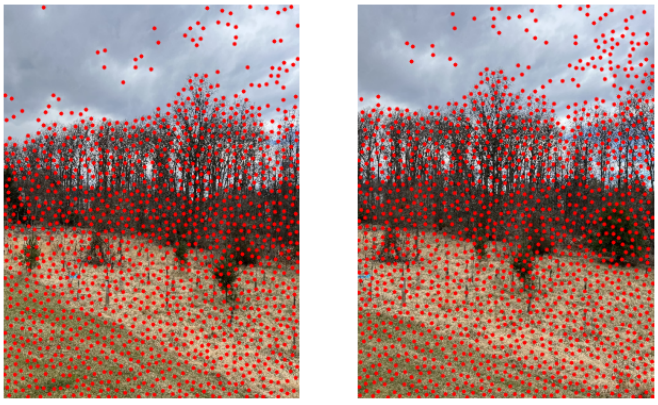

Fig. 5: Corners Detection in CostumSet 1

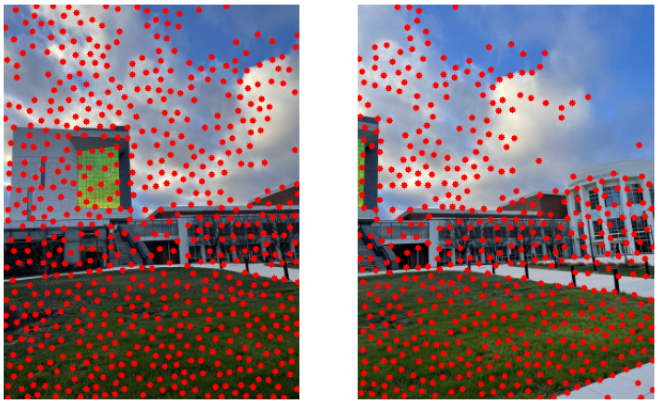

Fig. 6: Corners Detection in CostumSet 2

Fig. 20: Feature Descriptors in Train Set 1

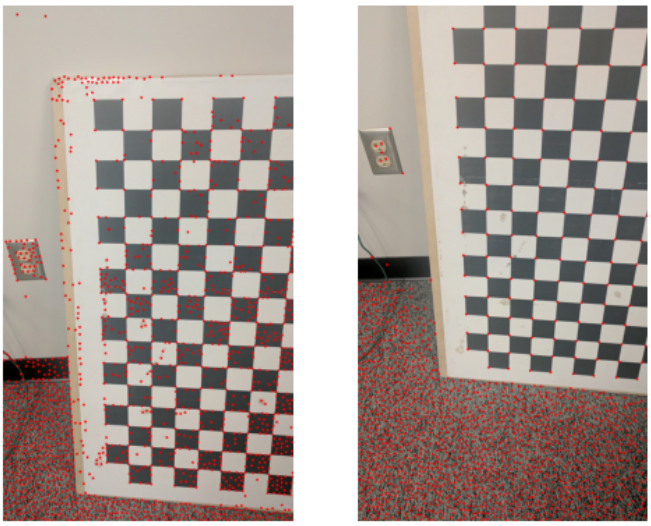

Fig. 7: Corners Detection in Test Set 1

Fig. 8: Corners Detection in Test Set2

Fig. 21: Feature Descriptors in Train Set 2

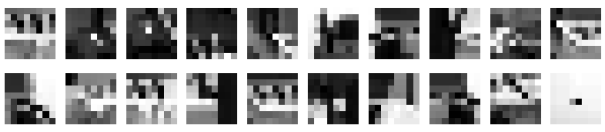

Fig. 22: Feature Descriptors in Train Set 3

Fig. 9: Corners Detection in Test Set1

Fig. 10: Corners Detection in Test Set4

Fig. 11: ANMS in Train Set 1

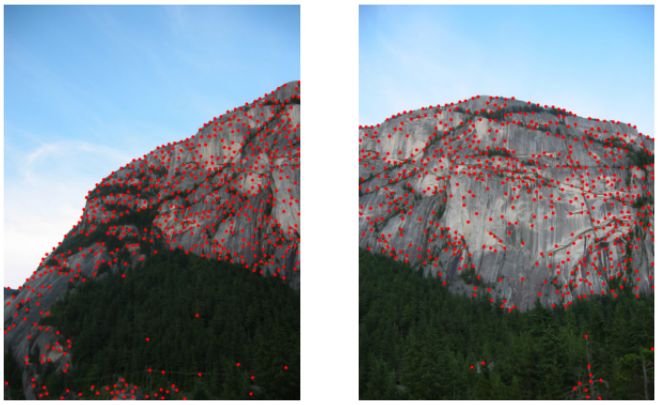

Fig. 12: ANMS in Train Set 2

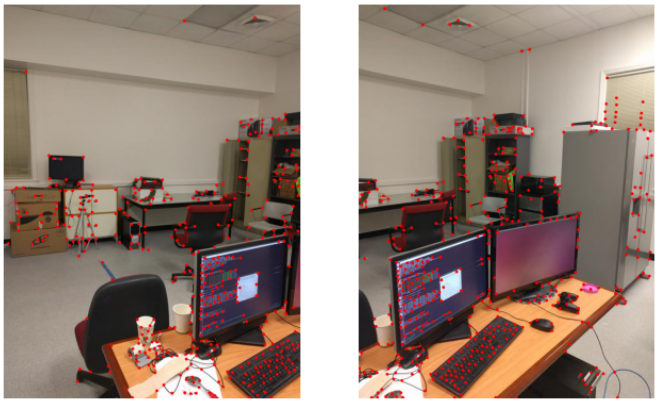

Fig. 13: ANMS in Train Set 3

Fig. 14: ANMS in CostumSet 1

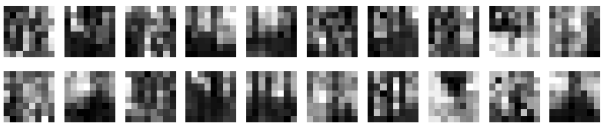

Fig. 23: Feature Descriptors in Custom Set 1

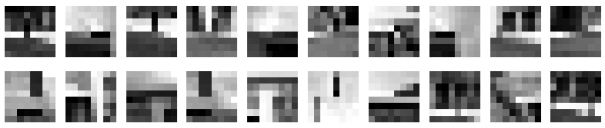

Fig. 24: Feature Descriptors in Custom Set 1

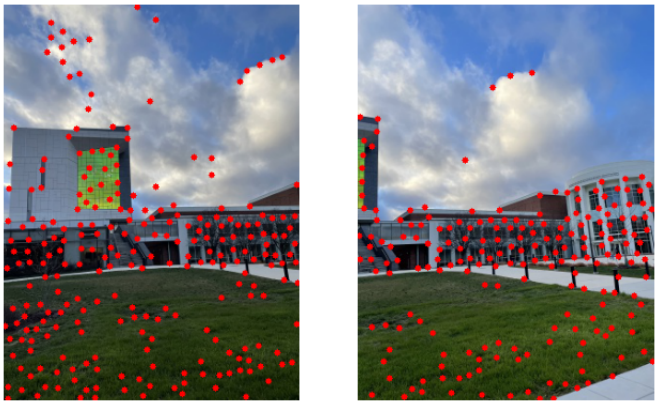

Fig. 15: ANMS in CostumSet 2

Fig. 16: ANMS in Test Set 1

Fig. 17: ANMS in Test Set2

Fig. 18: ANMS in Test Set1

Fig. 19: ANMS in Test Set4

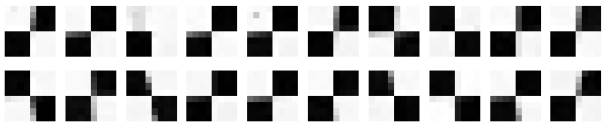

Fig. 25: Feature Descriptors in Test Set 1

Fig. 26: Feature Descriptors in Test Set 2

Fig. 27: Feature Descriptors in Test Set 3

Fig. 28: Feature Descriptors in Test Set 4

D. Feature Matching

After computing descriptors, we can use them to find the corresponding points between two images. We need to pick a point in the first image and then compute the sum of the square difference between all points in the second image. Last, store or reject the feature points that the ratio of the lowest distance to the second lowest distance is above or below a designed ratio number. Repeat this in all points in the first image and we can find the matching feature points.

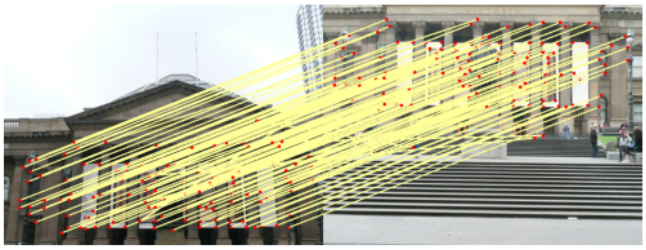

Fig. 29: Feature Matching in Train Set 1

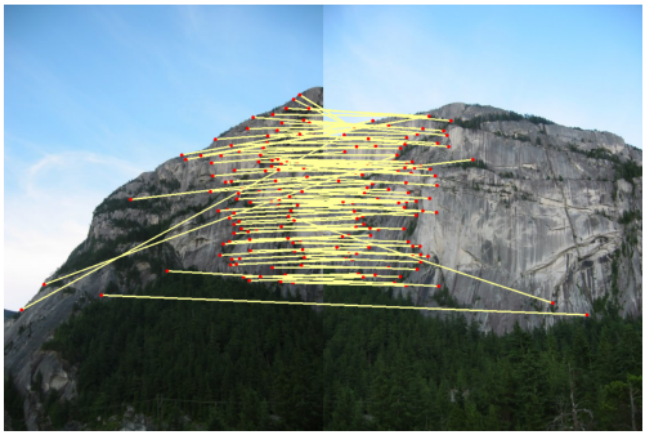

Fig. 30: Feature Matching in Train Set 2

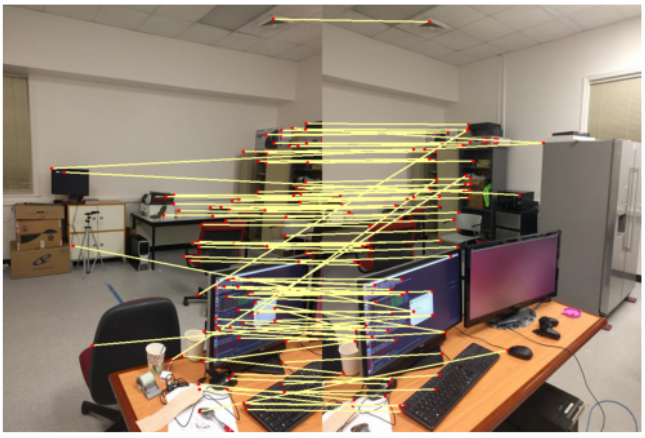

Fig. 31: Feature Matching in Train Set 3

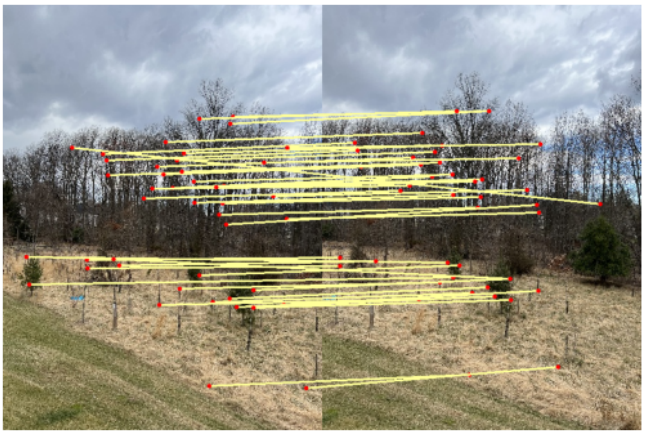

Fig. 32: Feature Matching in Custom Set 1

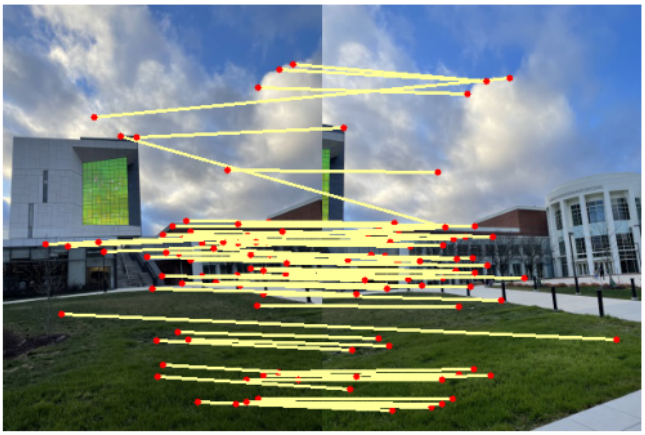

Fig. 33: Feature Matching in Custom Set 2

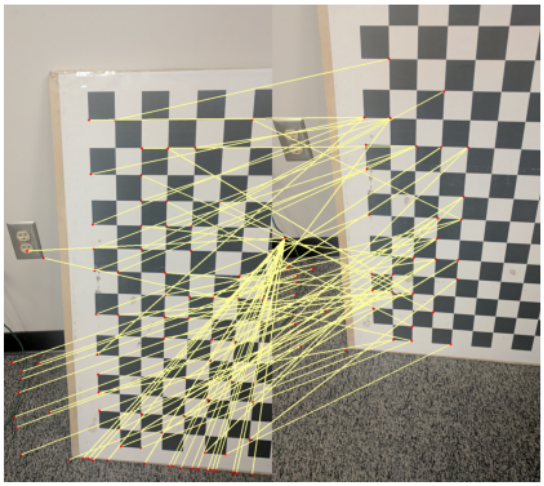

Fig. 34: Feature Matching in Test Set 1

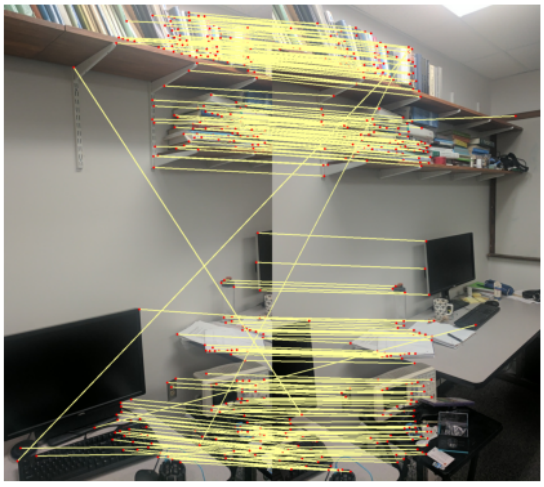

Fig. 35: Feature Matching in Test Set 2

Fig. 36: Feature Matching in Test Set 3

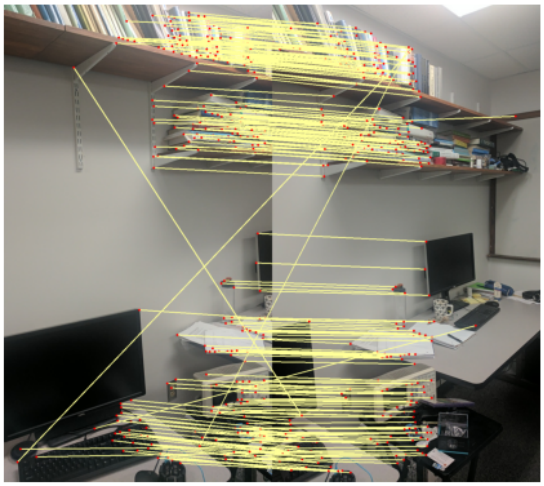

Fig. 37: Feature Matching in Test Set 4

E. RANSAC Homography

We can observe from feature-matching figures that not all feature correspondences are not matched. Therefore, we will utilize RANSAC to compute homography and find the homography that has the lowest error value after transforming the feature points in the image 1 to image 2 view position. You can find the detail of the RANSAC algorithm in [1]. The figures 38 to 46 show our result. Although we can see that RANSAC indeed removes some outliers, it can not completely remove all bad matching.

Fig. 38: Feature Matching by RANSAC in Train Set 1

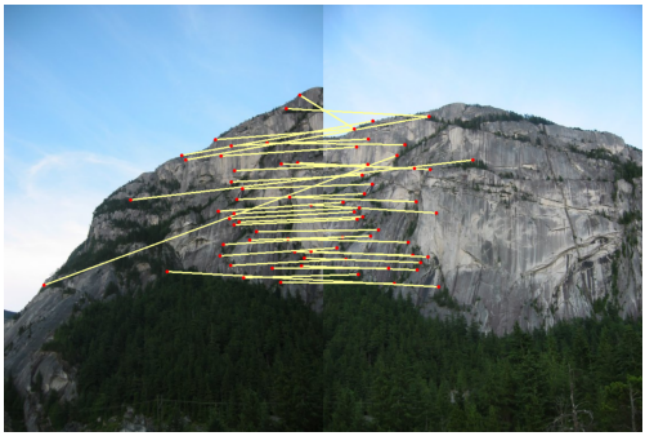

Fig. 39: Feature Matching by RANSAC in Train Set 2

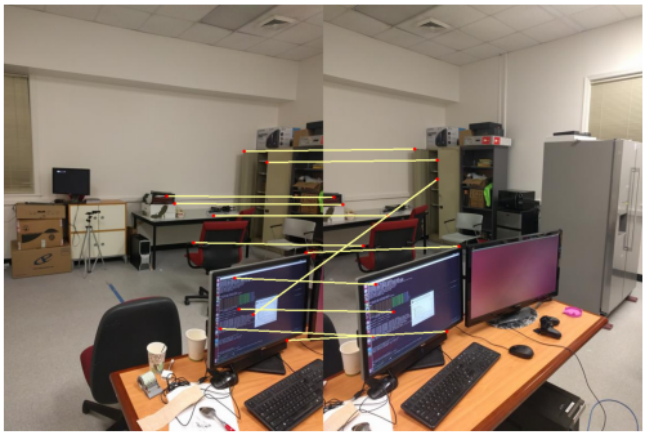

Fig. 40: Feature Matching by RANSAC in Train Set 3

Fig. 41: Feature Matching by RANSAC in Custom Set 1

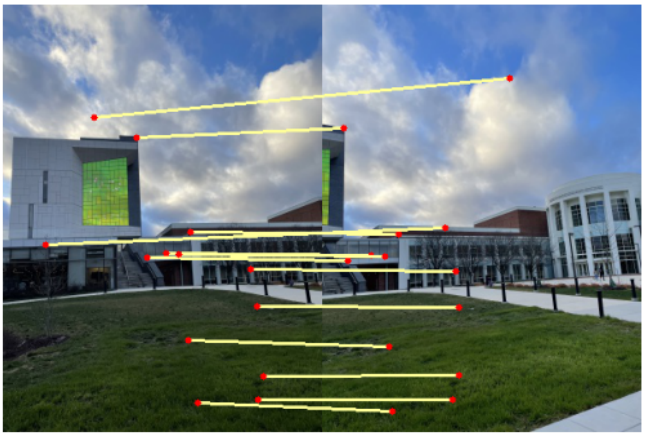

Fig. 42: Feature Matching by RANSAC in Custom Set 2

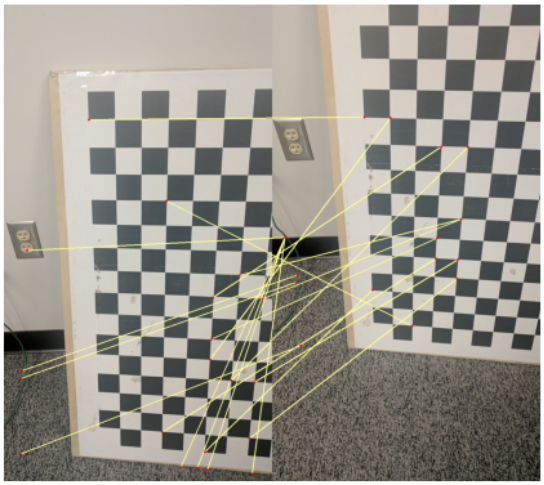

Fig. 43: Feature Matching by RANSAC in Test Set 1

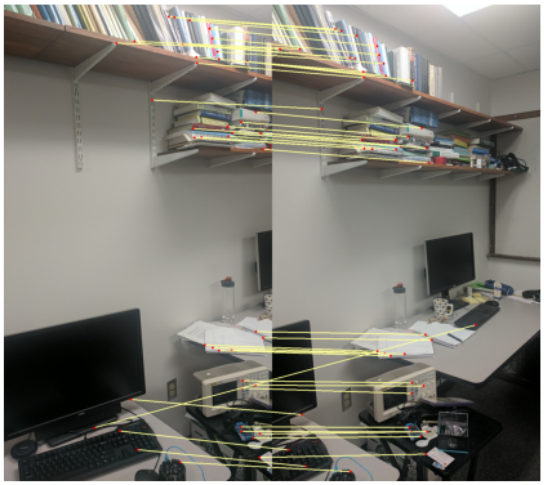

Fig. 44: Feature Matching by RANSAC in Test Set 2

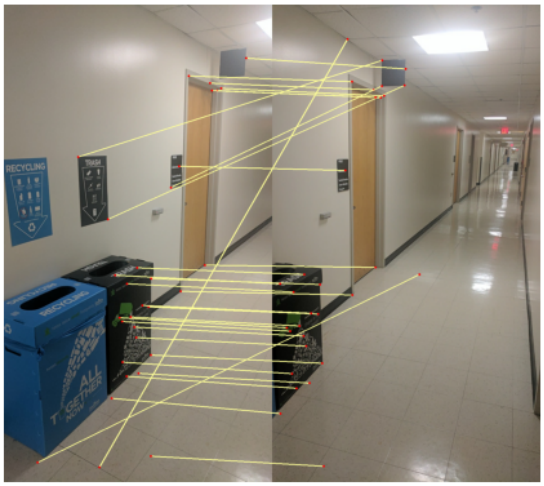

Fig. 45: Feature Matching by RANSAC in Test Set 3

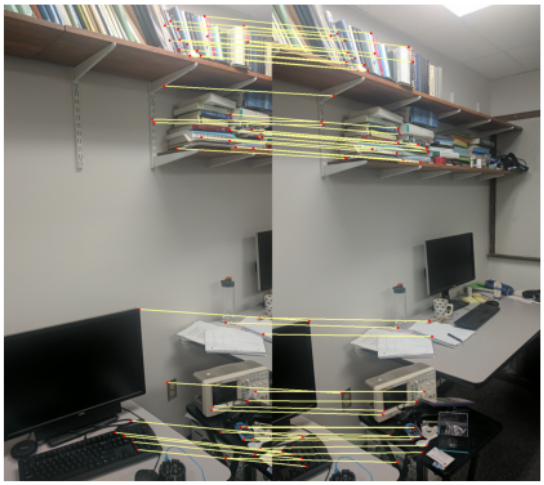

Fig. 46: Feature Matching by RANSAC in Test Set 4

F. Image Stitching

At last, we can stitch two images by wrapping the first image and stitching to the second image. If we have multiple images, we can do this repeatedly and stitch all the images of different view positions. However, we did not blend common regions in our project since we were running out of time. The final results are shown in figures 47 to 55.

G. Discussion

1) Stitching Multiple Images: In our experiment, when we doing panorama stitching up to 5 images, the images will be distorted severely. Due to the RANSAC algorithm randomly picking 4 points, the homography that we computed sometimes wraps the image crazy. In addition, our code sometimes can not stitch all images successfully and we infer that RANSAC gave us a bad homography that might cause this problem. Therefore, in order to successfully stitch numerous images, we will stitch the first and last two images synchronously and repeat this to the central image.

2) Rejected None or Very Less Overlapped Image.: We reject the no-overlap images by setting a threshold. If the number of matching points is below the threshold, the image will be rejected. However, the threshold is varied in different cases.

Fig. 47: Panorama Stitching in Set 1

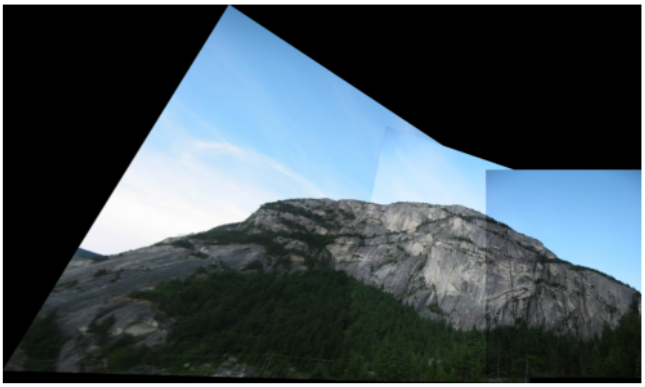

Fig. 48: Panorama Stitching in Set 2

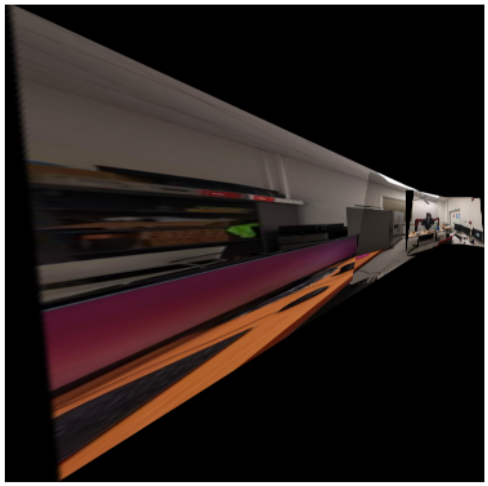

Fig. 49: Panorama Stitching in Set 3

Fig. 52: Panorama Stitching in Test Set 1

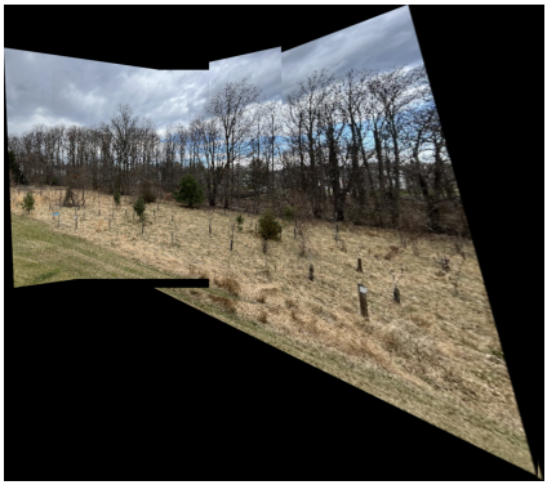

Fig. 50: Panorama Stitching in Custom Set1

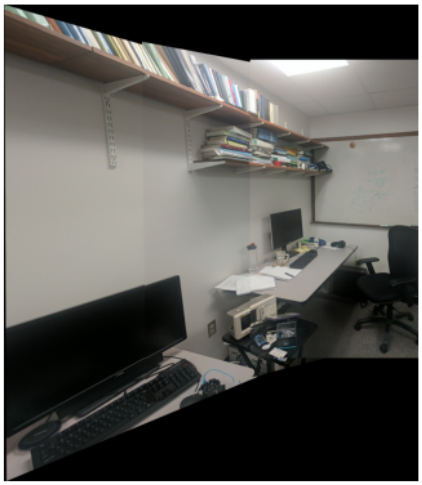

Fig. 53: Panorama Stitching in Test Set 2

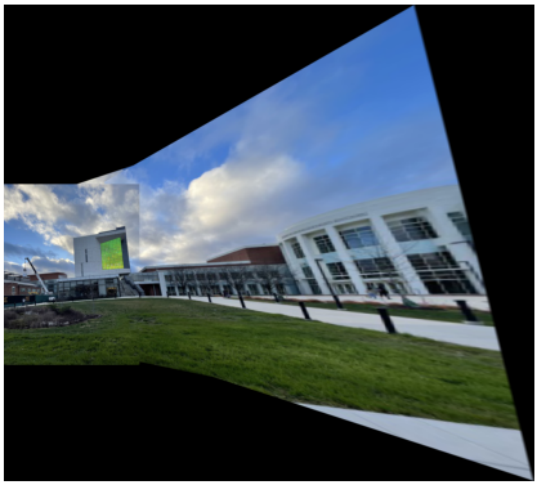

Fig. 51: Panorama Stitching in Custom Set 2

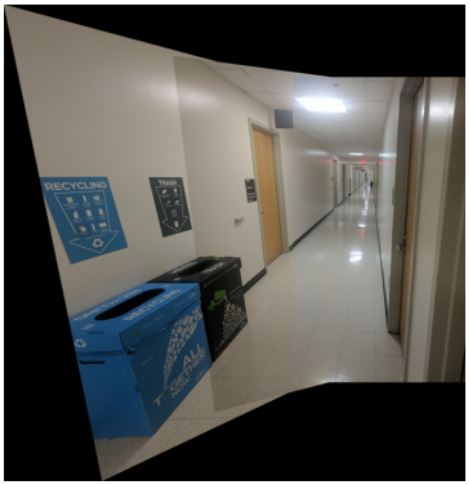

Fig. 54: Panorama Stitching in Test Set 3

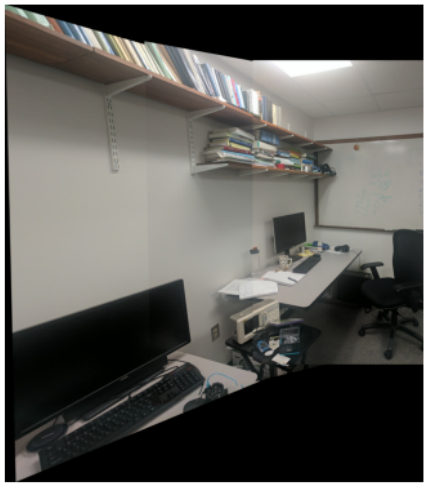

Fig. 55: Panorama Stitching in Test Set 4

II. Phase 2: Deep Learning Approach

In Phase 2, we implement two deep learning approaches to estimate the homography between two images. The model effectively combines corner detection, ANMS, feature extraction, feature matching, RANSAC and estimate homography all into one. We define a custom architecture, which consists of a VGG-19 based backbone for extracting features followed by multiple inception blocks and fully connected layers for regression task.

To train a deep neural network model to estimate the homography between pairs of images, we need a lot of images with known homographies between them. This is a very difficult task. Therefore we generate synthetic pair of images to train the network. We use a subset of MSCOCO dataset for generating known homographies of objects in natural settings.

A. Data Generation

We generate synthetic data to train our model. We first resize the input images to 320 x 240. Then we randomly generate a crop of the input image with size 128 x 128 and perform a perturbation of ρ = 32. We generate different type of data for our deep learning approach.

1) Supervised Approach: Once we generate a random cropped patch of 128 x 128, we randomly generate a new set of points by performing a perturbation of ρ = 32 to the corner points [2]. Moreover, we also ensure that he newly generated points are not collinear.Then we calculate the difference between the corner points and based on that and calculate the homography between them. Then we warp the original image by performing homography transformation. We extract the two patches from the original image and the warped image. The Data generator obtains two patches patches, one original crop and the warped patch obtained after applying homography transformation and a homograpy matrix that represents the transformation between the original patch and the warped patch with the former being considered as input and the latter as the output.

2) Unsupervised Approach: For the unsupervised approach, the data generator returns the cropped patches (P_A, P_B), and the original image as inputs and the warped image and difference of patch corners (H_{4Pt}) as the output [3].

B. Network Architecture

We have implemented the same architecture for supervised as well as unsupervised learning approaches. We have implemented a custom architecture with a VGG-19 backbone. It is a pre-trained model on the ImageNet dataset with pre-trained weights. We retrieve a specific layer to serve as the output layer of the VGG-19 base. We specify two input patches with the input shape of (128, 128, 3). Then we concatenate patches and pass them to the inception block. The inception block consists of multiple convolutional layers with custom kernel sizes and filer sizes that compute simultaneously and not sequentially. This is followed by a Max-Pooling layer with Batch normalization and fully connected layers. The network architecture is shown in Fig. 56, with a sample inception block in Fig. 57.

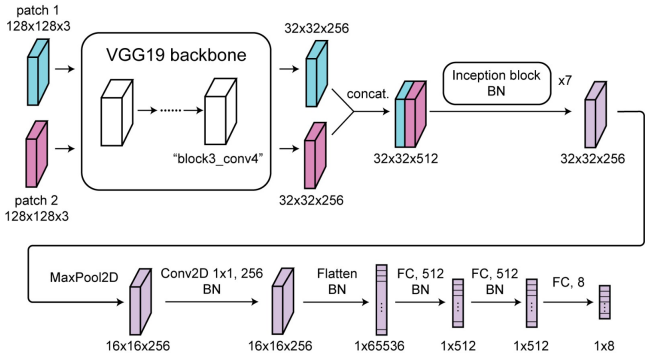

Fig. 56: Network Architecture

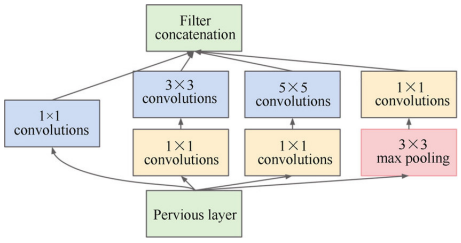

Fig. 57: Example of Inception model

1) Supervised Model: For the supervised model, we use Adam Optimizer and a learning rate of 1e-3 with a learning decay on plateau in loss metrics. We train the network on a smaller batch size of 8 and train the network for 150 epochs. The data generator reshuffles the dataset after every epoch to ensure randomness in data selection and avoid overfitting. We define a custom loss function that computes the mean squared error (MSE) between ground truth and the homography matrix. We test the supervised model on a custom test dataset. The hyperparameters of the supervised model are shown in Table I and the training & validation loss are shown in Fig. 58.

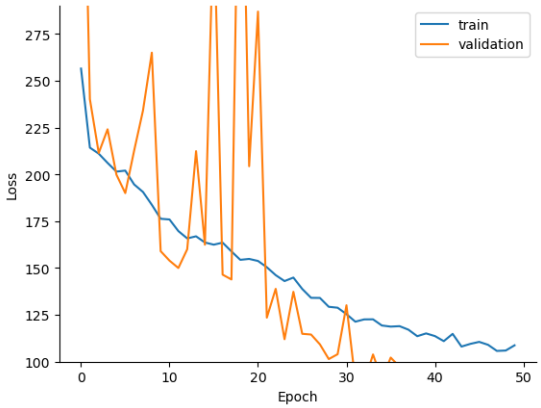

Fig. 58: Training and Validation Loss over 50 Epochs of Supervised Model

Batch Size	8
Optimizer	Adam
Epochs	150
Learning Rate	10^{-3}
Number of Parameters	37,779,808

TABLE I: Hyperparameters of Supervised Model

2) Unsupervised Model: For the Unsupervised model, we also use Adam Optimizer and a learning rate of 1e-3 with learning decay on plateau in loss metrics. Even for the unsupervised model, we train the network on a small batch size of 8 and 100 epochs. Here too, we define a custom loss function that computes the absolute difference between predicted and actual corner shift values in the x and y directions. The data generator reshuffles the dataset after every epoch to avoid data exhaustion. The hyperparameters of the unsupervised model are shown in Table II and the training & validation loss are shown in Fig. 59.

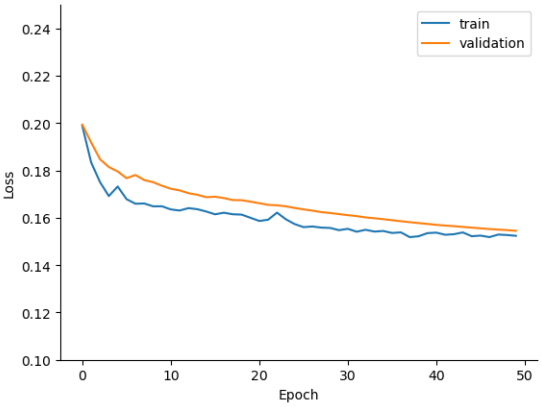

Fig. 59: Training and Validation Loss over 50 Epochs of Unsupervised Model

Batch Size	8
Optimizer	Adam
Epochs	100
Learning Rate	10^{-3}
Number of Parameters	37,779,808

TABLE II: Hyperparameters of Supervised Model

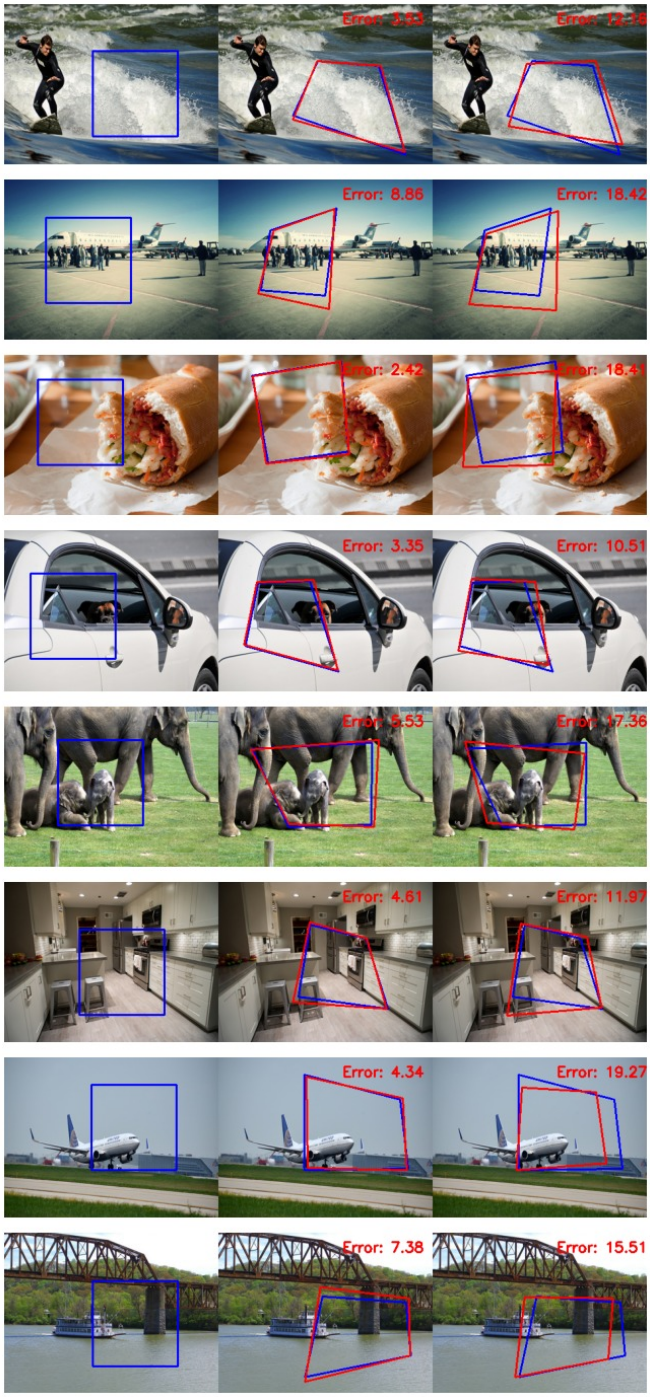

Fig. 60: Supervised (2nd Column) and Unsupervised (3rd Column) estimated homographies against synthetic ground truth (1st Column) from Train/Val/Test set

Model	Train EPE / std	Val. EPE / std	Test EPE / std
Supervised	5.160 / 2.486	5.149 / 2.574	5.159 / 2.356
Unsupervised	16.018 / 5.756	16.069 / 5.743	15.834 / 5.214

TABLE III: AVERAGE EPE OF OUR MODELS ON THE TRAIN/VALIDATION/TEST SET.

C. Discussion

In phase 2 of the project, we used a neural network, i.e., homography net, to infer the homography between artificially transformed image pairs. We implemented both the supervised and unsupervised training schemes, and we obtained comparable, if not better, performance with our supervised model. This is potentially due to our experimentation with the model architecture. We chose a pretrained model (VGG19) as our backbone, which provides useful built-in feature extraction. The subsequent stacking of inception blocks helps to further extract multi-resolutional features that are instrumental to discovering the homography. On average, our supervised model achieved an L2 loss of 5 pixels consistently across the training, validation, and test set. In addition, this error is significantly lower than that reported in [2]. Thus, our model architecture points to a potential revenue for improving the performance of the homography net.

During training, we found that two-stage training helps to achieve better performance, especially in the case of unsupervised training. Specifically, we first trained the model with $\rho = 16$, and once the model performance plateaus, we moved on to $\rho = 32$, the same parameter as chosen in [2]. If instead we trained the model directly with $\rho = 32$, the performance in general suffers. Thus, the training of the homography net benefits from a two-stage training schedule.

In implementing the training for the unsupervised model, we experienced some numerical instabilities in the loss calculation that resulted in "NaN" values being backpropagated through the network. We addressed this issue by implementing extra checks for these edge cases, and force the TensorFlow to run in eager mode such that we can exert more control in terms of program flow. Finally, implementing the differential TensorDLT and Spatial Transformer was slightly challenging as we needed to ensure the successful compilation of the complete TensorFlow computation graph. Nevertheless, our implementation allowed us to successfully train the model in the unsupervised fashion. We present EPE results on Train, Val, and Test sets in Table III and can observe that our supervised model has a higher performance than the unsupervised model. Figure 60 shows the supervised and unsupervised homographies against synthetic ground truth from Train, Val, and Test sets.

III. Conclusion

We have successfully implemented both phases of the project, solving for homography through traditional feature matching and a modern data-driven approach. Our unique implementation of homography net architecture provides a potential route for improved inference accuracy using convolutional neural networks for homography extraction.

References

[1] Wikipedia contributors. Random sample consensus. https://en.wikipedia.org/wiki/Random_sample_consensus.

[2] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *arXiv preprint arXiv:1606.03798*, 2016.

[3] Ty Nguyen, Steven W Chen, Shreyas S Shivakumar, Camillo Jose Taylor, and Vijay Kumar. Unsupervised deep homography: A fast and robust homography estimation model. *IEEE Robotics and Automation Letters*, 3(3):2346–2353, 2018.