P1 - AutoPano

Uday Sankar
usankar@wpi.edu
Using 5 Late days.

Gowri Shankar Sai Manikandan
gmanikandan@wpi.edu
Using 5 Late days.

Shaurya Parashar
sparashar@wpi.edu
Using 5 Late days.

***Abstract*—In this project, we intend to stitch two or more images and create a seamless panorama. This is to be achieved by two approaches - the first is to use classical computer vision techniques and the second to use deep learning techniques. Based on the approach taken, the project is split into two phases. In phase 1, features found using corner detection will be matched, in turn finding the Robust Homography. Then, the images can be warped and blended together. In phase 2, a supervised and an unsupervised networks are implemented for computing the Robust Homography. Finally, the results obtained from phase 2 are to be compared with those obtained in phase 1.**

I. PHASE 1: TRADITIONAL APPROACH

A. *Introduction*

The objective of this section is to implement a panorama stitching algorithm for multiple images using classical computer vision techniques. This can be achieved by using the following steps:

1) Corner detection.
2) Adaptive Non-Maximal Suppression (ANMS).
3) Feature Descriptors.
4) Feature Matching.
5) Random Sample Concensus (RANSAC).
6) Stitching/Blending Images together.

The pipeline involved in the traditional approach is shown in figure 1.

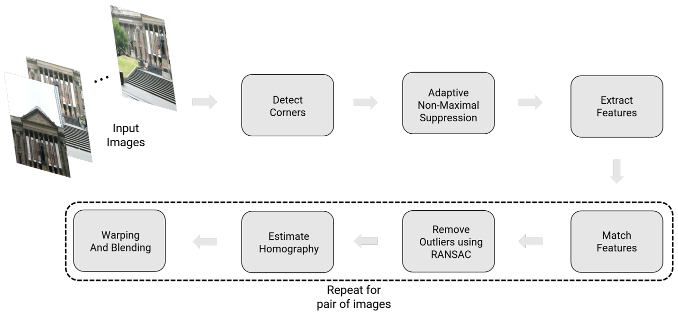

Fig. 1: Overview of Panorama Stitching using Traditional Approach.

B. *Corner Detection*

In order to get the features from the images so that they can matched, we start with the step of corner detection. For this project, we used two methods for this purpose - first one being Harris corner detection method and the second one being Shi-Tomasi corner detection method. For Harris corner detection, we used the cornerHarris() function and for Shi-Tomasi corner detection, we used the goodFeaturesToTrack() function. Both of these funcitons were imported from the OpenCV library.

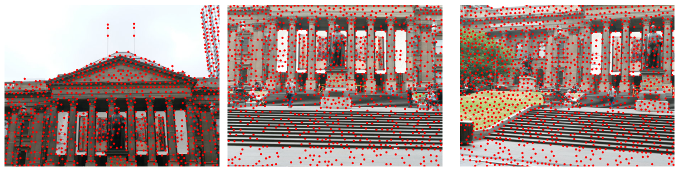

Fig. 2: Corner Detection for Train Set 1.

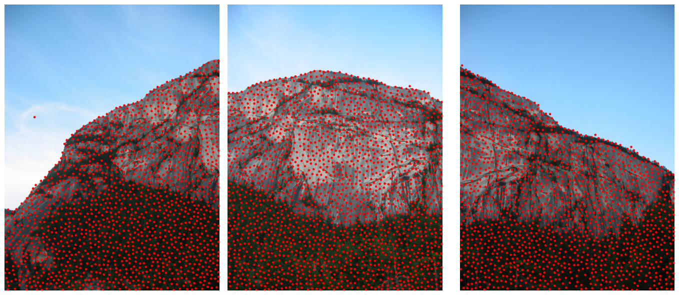

Fig. 3: Corner Detection for Train Set 2.

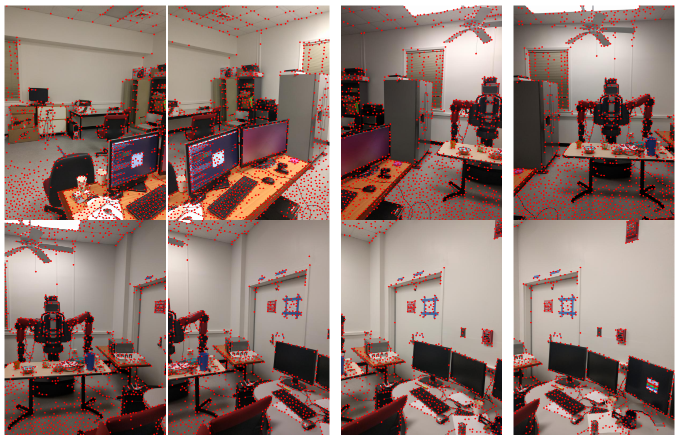

Fig. 4: Corner Detection for Train Set 3.

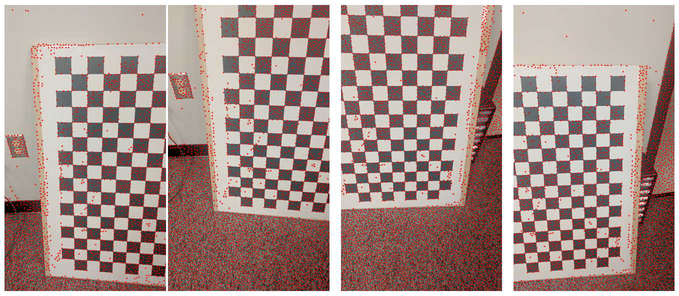

Fig. 5: Corner Detection for Test Set 1.

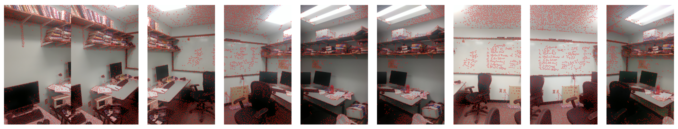
Fig. 6: Corner Detection for Test Set 2.

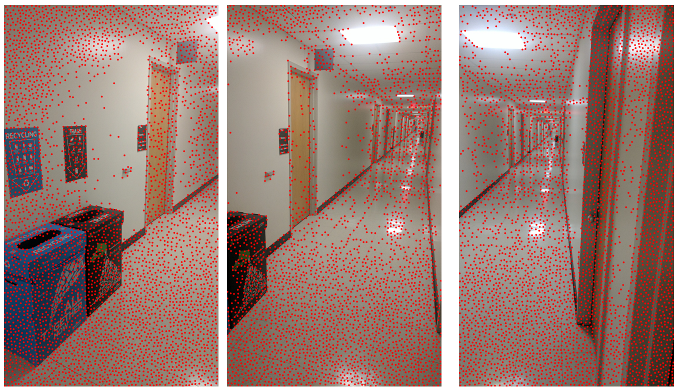
Fig. 7: Corner Detection for Test Set 3.

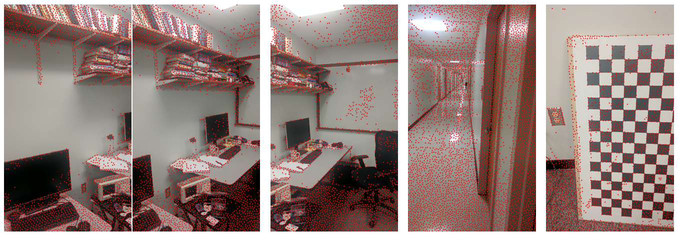
Fig. 8: Corner Detection for Test Set 4.

C. Adaptive Non-Maximal Suppression

From the features obtained during corner detection, the N_{best} features need to be selected for feature matching. This is done by using a technique called Adaptive Non-Maximal Suppression (ANMS). This step is necessary because, during corner detection, since corners in an image are not perfectly sharp, each corner can get multiple hits. ANMS basically makes sure that the corners detected equally distributed across the whole image. The pseudo-code for ANMS is shown in figure 8. In our code, an N_{best} value of 1000 was taken.

```
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
            ED = $(x_j - x_i)^2 + (y_j - y_i)^2$
        end
        if $ED < r_i$ then
            $r_i$ = ED
        end
    end
end
Sort $r_i$ in descending order and pick top $N_{best}$ points
```

Fig. 9: Pseudo-code for Adaptive Non-Maximal Suppression

The ANMS outputs based on the given pseudo-code are shown below.

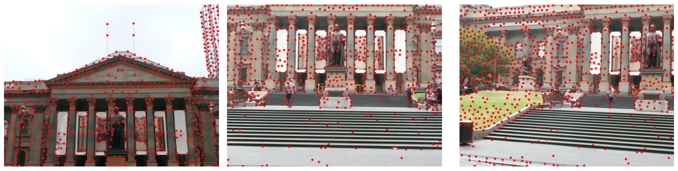
Fig. 10: ANMS for Train Set 1.

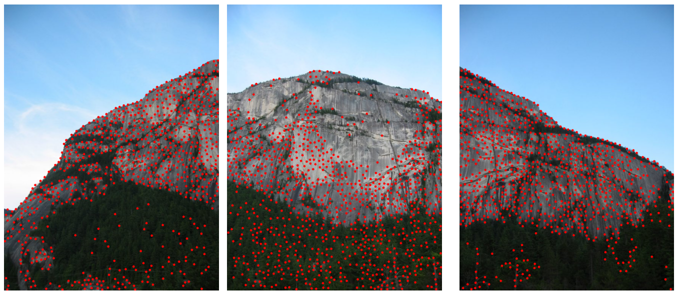
Fig. 11: ANMS for Train Set 2.

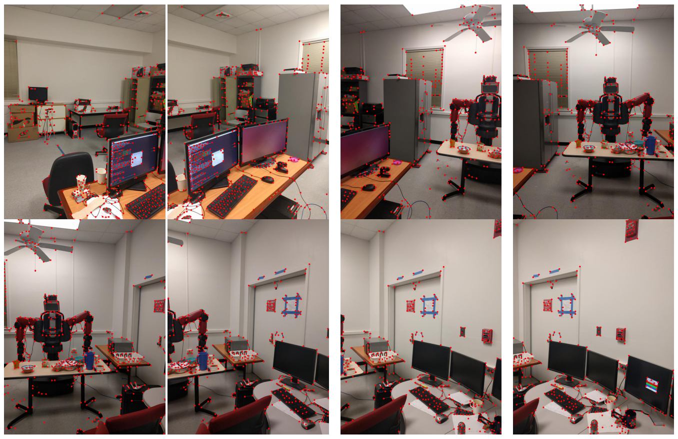
Fig. 12: ANMS for Train Set 3.

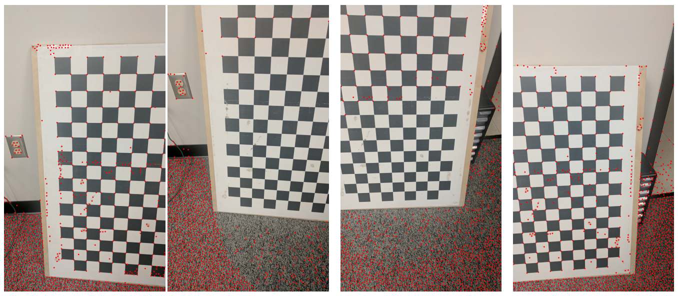
Fig. 13: ANMS for Test Set 1.

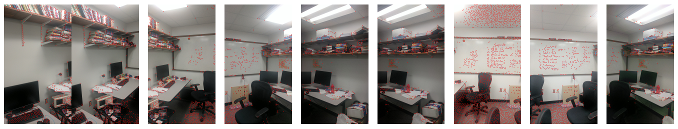
Fig. 14: ANMS for Test Set 2.

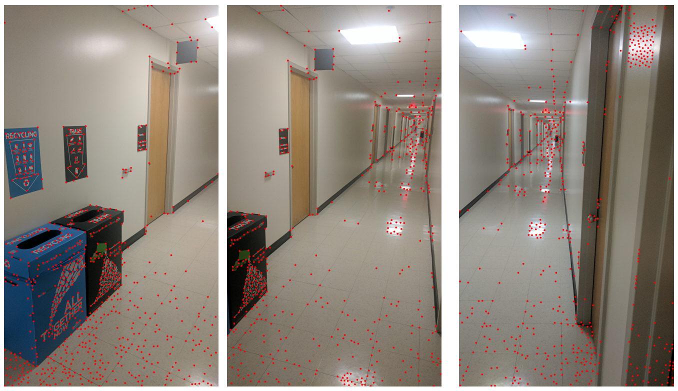
Fig. 15: ANMS for Test Set 3.

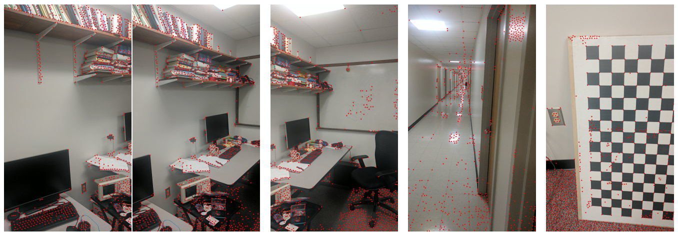
Fig. 16: ANMS for Test Set 4.

D. Feature Description

In this step, the goal is to describe each feature vector. First, we took a patch of size 41 x 41 around the feature point. Then we applied Gaussian blur and down-sampled the blurred patch to an 8 x 8 matrix. Then, we reshaped this matrix into a vector of 64 elements. The vector was finally standardized by making is mean 0 and variance 1. A sample patch taken for feature description is shown in figure 17.

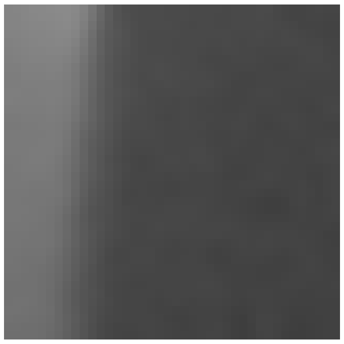
Fig. 17: A sample of a patch that is taken for feature description.

E. Feature Matching

Once we have the feature vectors, the features in two different images can be matched by computing the Sum-Square Distance (SSD) across each of these features. The ratio of the shortest and second shortest SSD is taken and compared with a certain comparison ratio which we took as 0.5. We also set a condition that if the matches found in this process is less than 30 matches, then it can be considered that there is no overlapping region in these images and panorama cannot be formed. An example for feature matching in action is shown in figure 18.

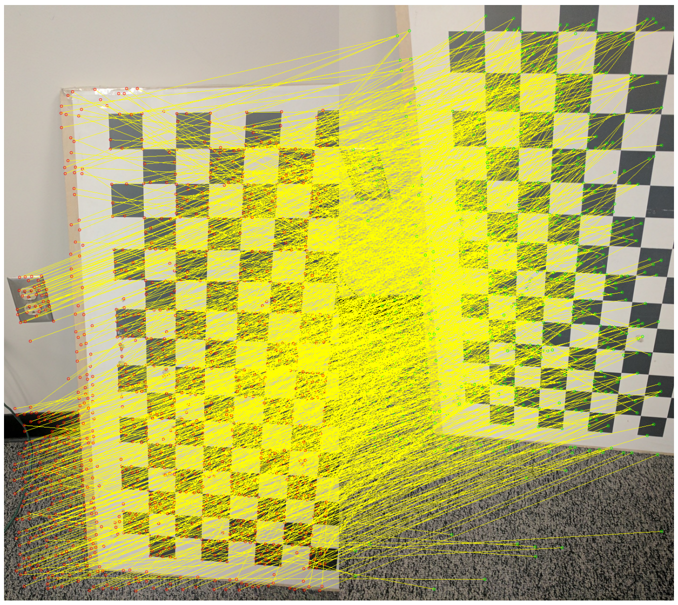
Fig. 18: Feature Matching.

F. RANSAC

If we closely look at the output of feature matching, we can see that there are a lot of incorrect matches. This will be major problem when we want to find the homography between these images. So, before proceeding to the next steps, we have to remove these outliers. This can be done by a method called **Random Sample Concensus (RANSAC)**. In RANSAC, initially four point pairs from the images are selected at random. Then the homography matrix is calculated between them using the formula in figure 19.

$$PH = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & x'_1 \\ 0 & 0 & 0 & -x_1 & -y_1 & -1 & x_1y'_1 & y_1y'_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x'_2 & y_2x'_2 & x'_2 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & x_2y'_2 & y_2y'_2 & y'_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x'_3 & y_3x'_3 & x'_3 \\ 0 & 0 & 0 & -x_3 & -y_3 & -1 & x_3y'_3 & y_3y'_3 & y'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x'_4 & y_4x'_4 & x'_4 \\ 0 & 0 & 0 & -x_4 & -y_4 & -1 & x_4y'_4 & y_4y'_4 & y'_4 \end{bmatrix} \begin{bmatrix} h1 \\ h2 \\ h3 \\ h4 \\ h5 \\ h6 \\ h7 \\ h8 \\ h9 \end{bmatrix} = 0$$

Fig. 19: Finding Homography.

Once the P matrix is obtained, the homography H can simply be calculated using Singular Value Decomposition.

$$SVD\ P = USV^T$$

The V vector in the above equation can be taken as the homography matrix of the two images. Now, this homography is used to estimate the random four points in image 2 with respect to image 1. The best set of inliers are found by repeating these steps from a number of iterations. Our code did this for 5000 iterations. Plotting the matching like feature matching but after RANSAC would look like figure 20.

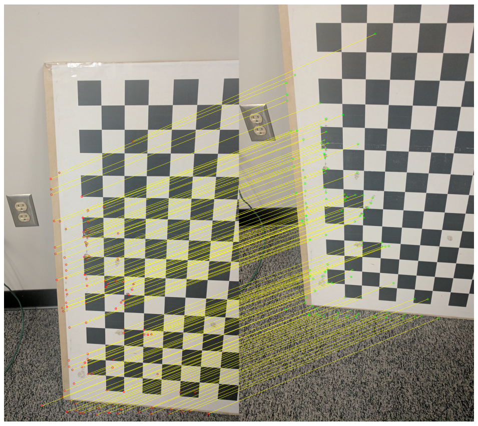

Fig. 20: RANSAC.

G. Stitching/Blending Images

Once the homography between the two images are obtained, they can formed into a single panorama by simply warping one image and then stitching it over the common area of second image. Since we were free to chose the way this can be done with multiple images, we chose the following way to approach this problem.

1) Find homography between image 1 and image 2.
2) Apply this homography on four corner points of image 1.
3) Calculate the minimum and maximum x and y translation of image.
4) Find the homography matrix for this translation and multiply with initial homography.
5) Warp image 1 with this new homography and then overlay image 2 on to image 1 forming the panorama.
6) Now this panorama becomes image 1 and next image in the input becomes image 2 and above steps are repeated.

A sample of panorama between two images being formed after warping and stitching is shown in figure 21.

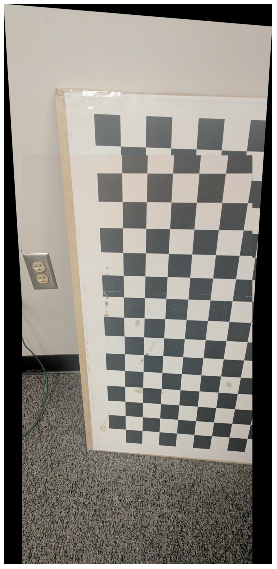

Fig. 21: Warping and Stitching to form panorama.

H. Results

Finally, the above mentioned approach was put to action on all the given sets of images. Given the time constraints in being able to debug the code, the results are not the greatest. But it is still amazing to see how powerful mathematical concepts can be when coupled with high computational power of present day computers. Even though, some results were not at all useful, some other results turned out to be impressive. For instance, aside some irregularities near the stair case, the panorama stitch for train set 1 is almost seamless. But at the same time our program struggles to give a proper output for the Train Set 3. We believe this is because our implementation finds it difficult to make correct matches of features when common features are heavily warped. For visualization, figure 24 shows the stitches that our code could come up with.

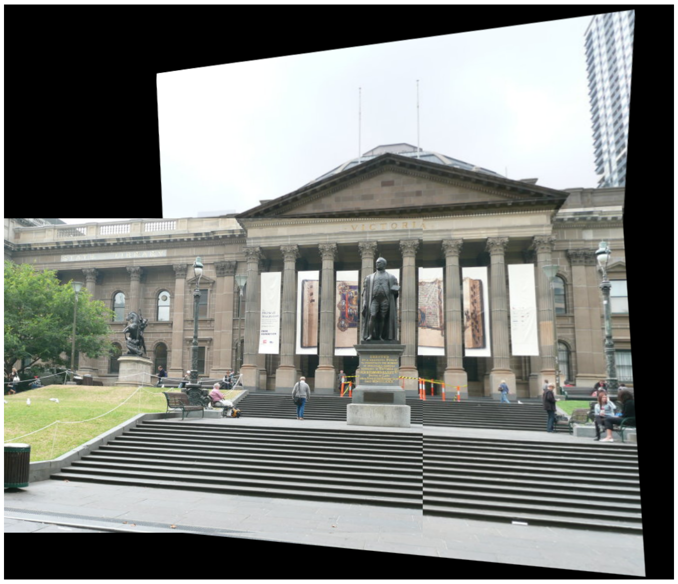

Fig. 22: Panorama Stitch of Train Set 1.

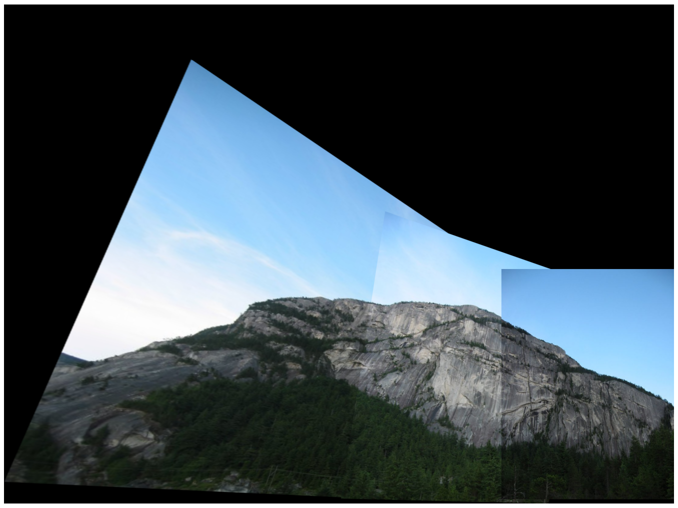

Fig. 23: Panorama Stitch of Train Set 2.

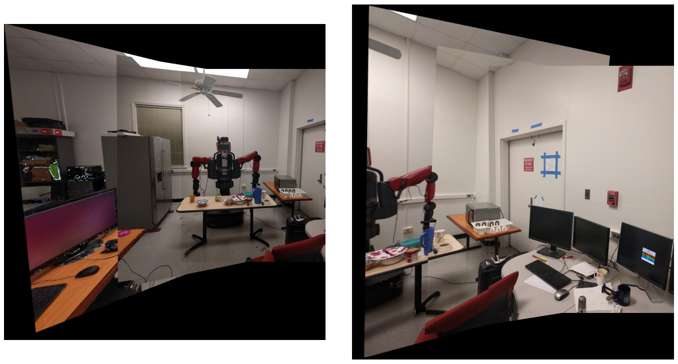

Fig. 24: Panorama Stitches of Train Set 3.

Now the results of panorama stitching for the test sets are shown below.

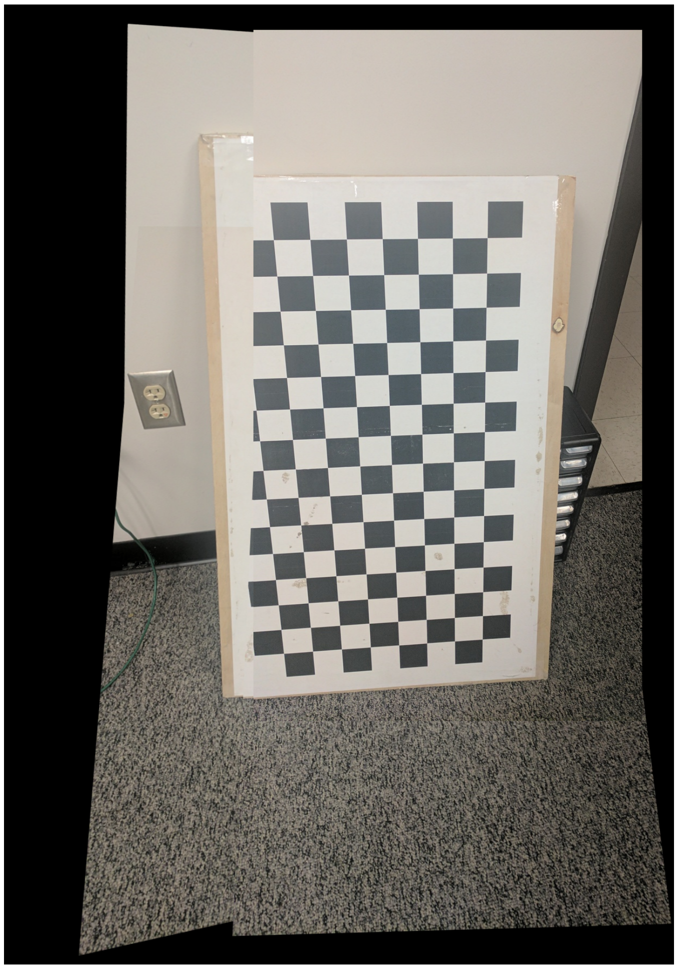

Fig. 25: Panorama Stitch of Test Set 1.

In this stitch, the checkerboard doesn't perfectly align. This is because in this particular case, unlike the other cases, the images are cyclic. The first and last images have common region of overlap as well. Since all the four images have been taken from a close distance, it is difficult to perfectly align them to get the expected output.

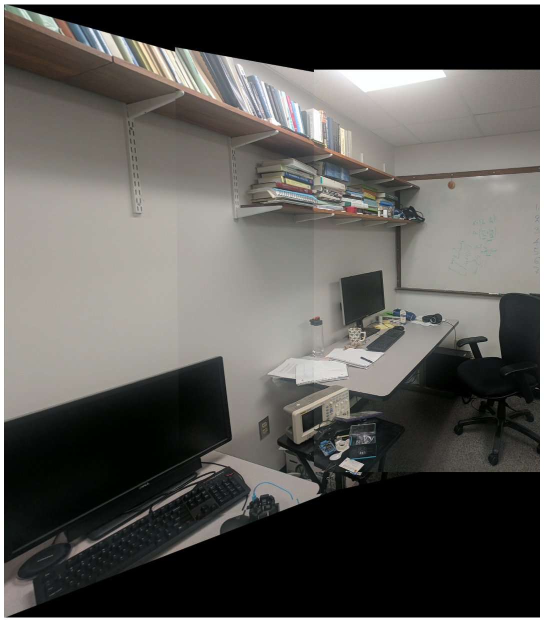

Fig. 26: Panorama Stitch of Test Set 2.

Test Set 2 might look similar to the case of Train set 3, but the reason that we got a incomplete output is that the fourth image does not have any usable common region with the either of the first three images. One way to solve this would be manually rearranging the images so as to get a series of images with each consecutive images having common regions. But that method is absolutely underwhelming. So we have to figure out a way for the program to automatically sort the images into a useful order. This can probably be done by using the theory of graphs, which is something to work on in the future.

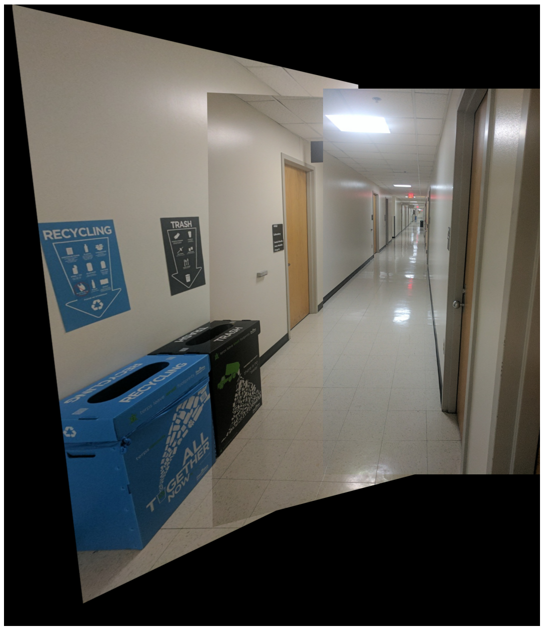

Fig. 27: Panorama Stitch of Test Set 3.

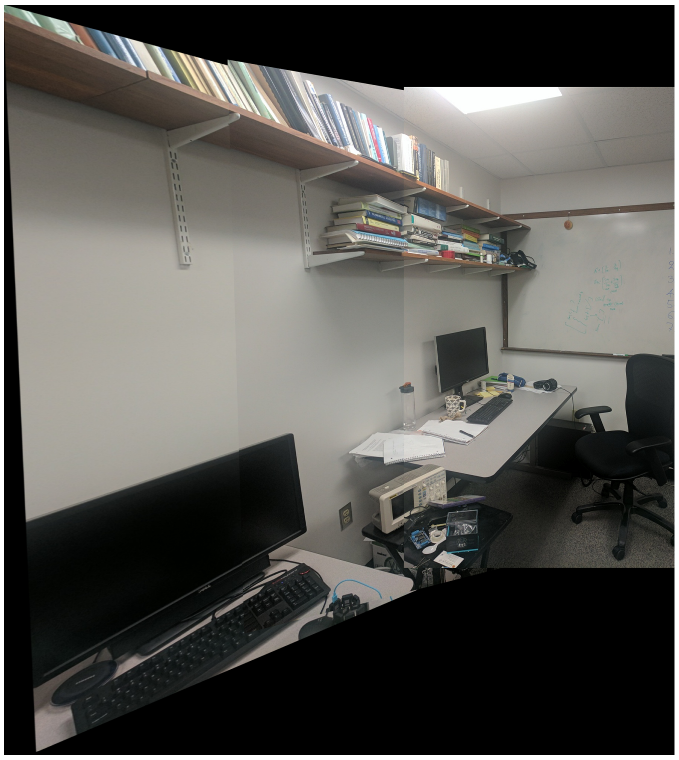

Fig. 28: Panorama Stitch of Test Set 4.

Finally, in test set 4, the fourth image does not even belong with the other three images, and in turn does not produce any usable matches with the other images. The output in figure 28 is the last panorama that was created before the program halted due to absence of necessary matches.

II. PHASE 2: DEEP LEARNING APPROACH

A. Data Generation

1. Random Patch was obtained from the image, while keeping in mind that the pixels remain within the image.

2. Random perturbation in a range of [-32,32] was added to the corners of this patch.

3. Then we warp the original image, using inverse of homography between the corner points of patch A and patch B. Thus, we generate the ground truth as the known homography between the patches as well. Figure 29 shows a sample of patch A, and it's corresponding patch B.

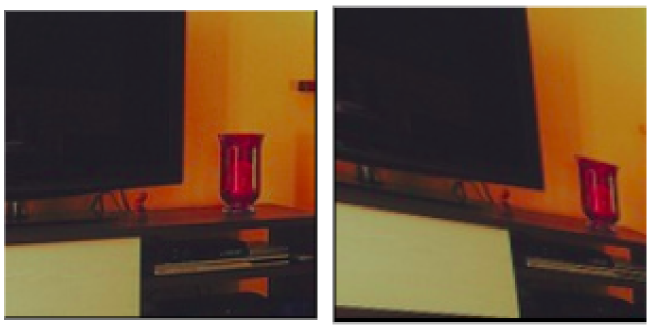

Fig. 29: Data Generation

B. Supervised Learning

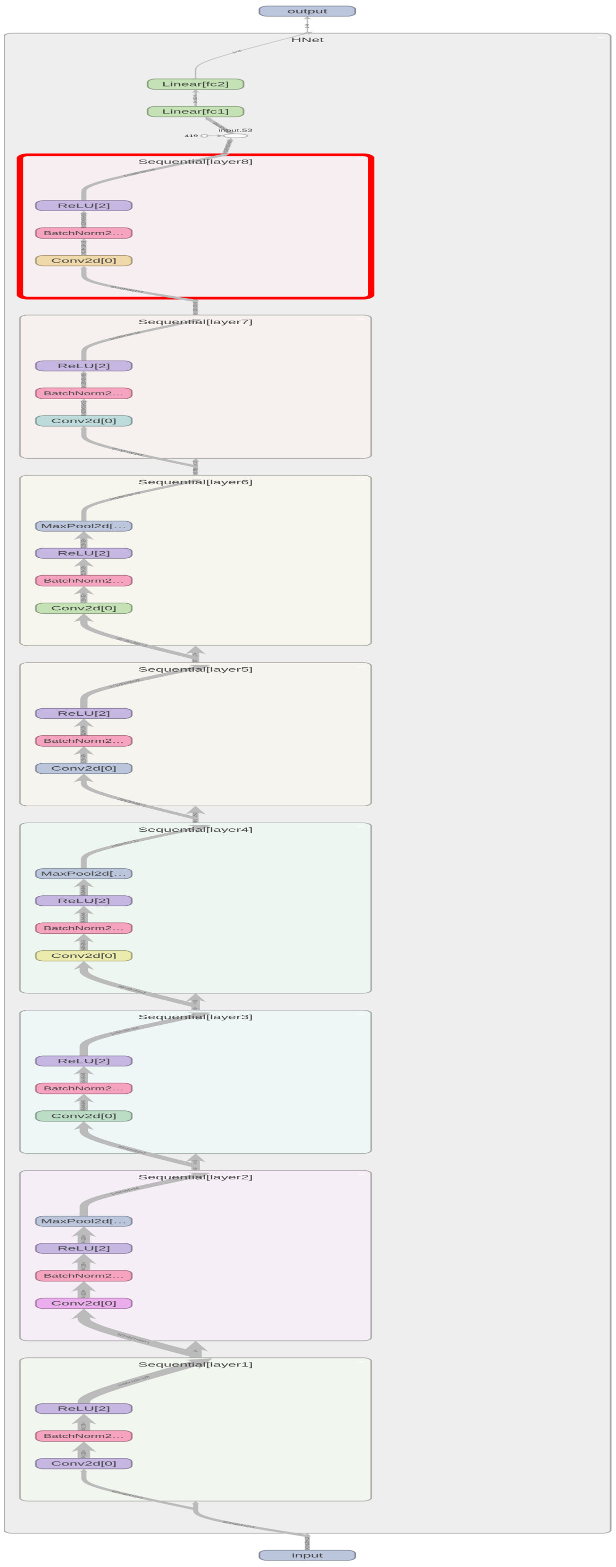

Fig. 30: Supervised Learning

1. Homography net model was developed for supervised learning, and its architecture can be seen Figure 30. The inputs for the network were the two patches generated, which

were then stacked channel wise. The homography between the patches was used as ground truth, to compute the loss function.

2. First Iteration - The first training iteration for training the supervised model had mini batch size as 64, stochastic gradient descent optimizer was used, while keeping the learning rate as 0.005. Figure 31, shows the loss curve obtained during the training and on validation set. We found that our model was getting overfit and was not performing well on the validation set.

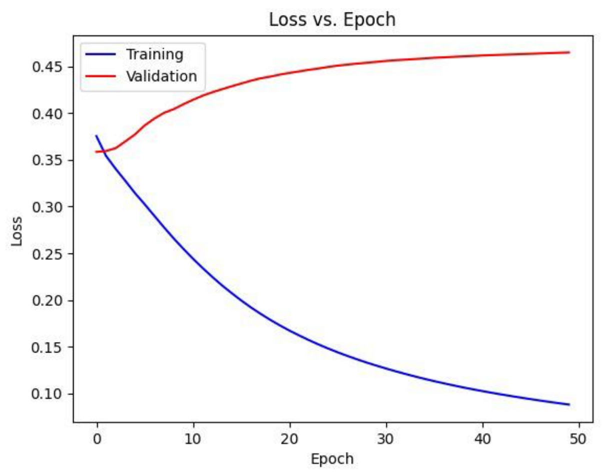

Fig. 31: Supervised Learning-Iteration 1

3. Second Iteration - We modified some parameters to reduce the overfit that our model was getting. Figure 32 shows the loss curve obtained during the training and on validation set. Batch size was kept to 2, and Adam optimizer was used with decreased learning rate of 0.001. In this iteration, our model went from overfit in the previous case, to dropping drastically in training loss. During testing, we found that the model was learning near identity matrix. We think it might be due to the decreased batch that we did to overcome overfit. Also, learning rate could've decreased as well. So, due to this, we weren't able to test our supervised model on generating homography, and warping images.

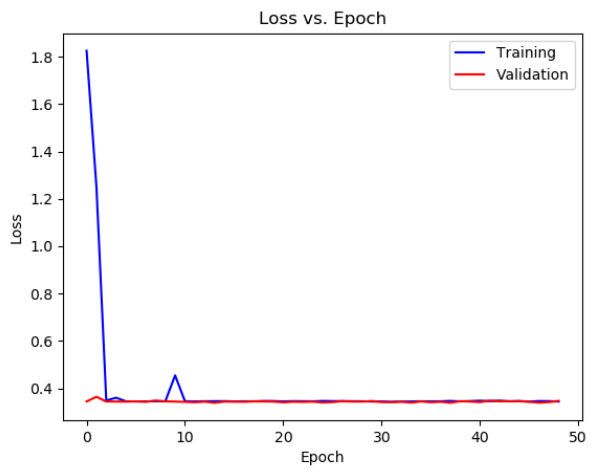

Fig. 32: Supervised Learning-Iteration 2

C. Unsupervised Learning

1. The unsupervised model (Fig. 33), contains the basic homography network that we had in supervised learning. It just takes in the patches stacked as input. Using the received 4 point homography, and corner points generated from the patch A of each input, we convert this 4 point homography into a 3*3 homography, and pass it through a Homography Warper in order to obtain the loss function.

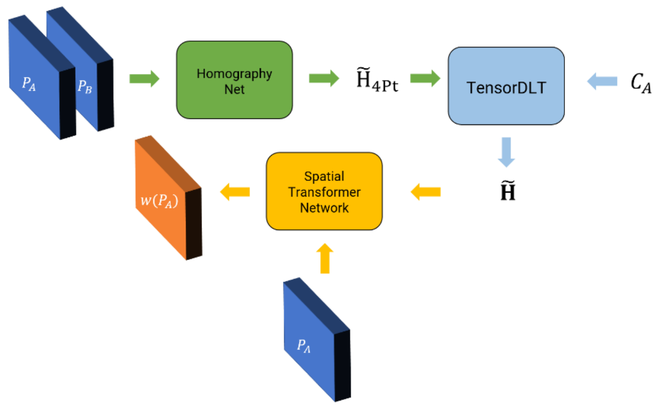

Fig. 33: Unsupervised Learning

2. The unsupervised model was trained for trained for 10 epochs, with mini-batch size as 16. Adam optimizer was taken with learning rate as 0.005. The reason we think may have been for improper convergence is the less number of epochs. The training code for unsupervised model didn't run on gpu, which we didn't expect. After hours of debugging too, we had to resort on training the model on cpu, which is why there is no convergence.The losses for the unsupervised model can be found below (Fig. 34).

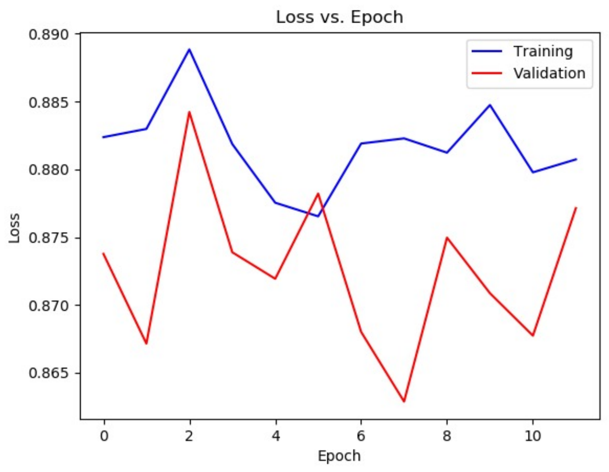

Fig. 34: Unsupervised Learning-losses