RBE 549: Project 1
My AutoPano

Deepak Harshal Nagle
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: dnagle@wpi.edu
Telephone: (774) 519-8335
Using 2 late dates

Irakli Grigolia
Computer Science
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: igrigolia@wpi.edu
Telephone: (508) 373-3402
Using 2 late dates

***Abstract*—Abstract—This report presents our submission of Project 1 - Autopano, a tool that can stitch together multiple images to create a seamless panorama. The report outlines the solutions for both Phase 1 and Phase 2 of the Panorama Problem. Phase 1 utilizes classical computer vision methods such as Homography Matrices to achieve stitching and Phase 2 employs deep learning algorithms. The Homography Net employs supervised and unsupervised neural networks to accurately stitch two or more images. This project aims to provide an efficient, automated solution to the image stitching problem and demonstrates the effectiveness of deep learning techniques.**

PHASE 1: CLASSICAL APPROACH

The traditional approach to creating a panoramic image involves several key steps:

1. Corner Detection
2. Adaptive Non-Maximal Suppression (ANMS)
3. Feature Descriptors Computation
4. Feature Matching
5. RANSAC for Outlier Rejection and Robust Homography Estimation
6. Stitching and Blending

A. Corner Detection

The first step in creating a panoramic image is to detect features, such as corners, in the images that will be stitched together. Three techniques have been used for this purpose: OpenCV's cornerHarris and goodFeaturestoTrack. The cornerHarris computes the corner strength for each pixel, while goodFeaturestoTrack provides corner point locations in descending order of their probability of being a corner. Further, goodFeaturestoTrack can also provide the corners in a uniformly distributed manner using a non-maximal suppression algorithm. It gives us a choice of corner detection algorithms between the Shi-Tomasi Algorithm and the Corner Harris algorithm itself. Considering these factors, "goodFeaturestoTrack" is more robust, and thus, was implemented by us for detecting the corners. Though using "goodFeaturestoTrack" would ensure that uniformly distributed corners, we decided to implement our own ANMS algorithm (see the next section) for non-maximal suppression. Thus, we decided to pass a smaller value for the minDistance argument in the "goodFeaturestoTrack".

Fig. 1. Corner Detection Example

B. Adaptive Non-Maximal Suppression

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. 2. ANMS algorithm

The Adaptive Non-Maximal Suppression (ANMS) algorithm is used to find the N strongest equally distributed corners

in an image. This is done to suppress corners that are too close to one another and to achieve an evenly distributed set of corners. The algorithm involves considering corners that have higher corner strength than a given corner and finding, among them, the one which is closest to the given corner. Similarly, performing the same process for all the corners, sorting the final list of closest corners is sorted in descending order out of which the Nbest corners as output are selected. The pseudo-code for ANMS is provided:

Fig. 3. ANMS Example

C. Feature Descriptors

The feature extraction step involves describing each corner point found in the previous step with a feature vector. This is done by taking a 41x41 patch around the corner point and applying Gaussian blur. For the corner points which are near the edges of the image, take the patch such that it will terminate at the edge. The blurred patch is then subsampled to an 8x8 matrix. The matrix is then flattened and standardized to remove bias and illumination effects. The resulting 64x1 vector is used as a feature descriptor to characterize each corner and is useful for matching corners in images. This step helps to address the problems of scale, rotation, and illumination.

D. Feature Matching

The process of feature matching involves finding the correspondence between features detected in two different images. This is done by computing the sum of squared differences (SSD) between each feature vector in the first image and the feature vectors in the second image. A threshold is set on the ratio of the lowest and second-lowest SSD for each feature point in the first image to determine if the pair is a match. If the ratio is less than the threshold, the pair is accepted, or else rejected. The number of matches between the two images is also considered, where we define a minimum of 10 matches as required for the images to be considered stitchable. The result of the feature-matching process is displayed by drawing lines between the matched points.

Fig. 4. Feature Matching Example

E. RANSAC for Outlier Rejection and Robust Homography Estimation

The RANSAC outlier rejection algorithm is incorporated into the pipeline to avoid noisy and incorrect feature pairs and obtain a better Homography. RANSAC (Random Sample Consensus) algorithm is used to obtain a robust Homography (H) matrix estimation. The algorithm pairs four corners from Image-I to four corners in Image-II randomly and computes the exact Homography matrix H. Now, another set of four corners is taken from Image-I and this ”H” is used to predict corresponding points in image-II. The SSD between the estimated points and the actual points in Image-II is calculated. If the SSD is under a given threshold, we add the pairs to a list of inliers, and the process is continued to other points in Image-II to obtain the final list of inliers. Finally, this whole process is repeated for all the points from Image-I and the largest set of inliers and the corresponding H matrix are kept.

F. Stitching and Blending

Texture map is generated by first convolving all the 120 filters with the input gray-scale image to get a stack of 120 output kernels. Thus, each point on the original image now has 120 different values (cluster centers) assigned to it. Using KMeans clustering, we generate the Texture map where each pixel is now represented by a Texton ID (out of 0 to 63, both inclusive) assigned to it. Similar to this, we perform Color gradient based on the colored 3-layer (RGB) input to group together the similarities in colors of the image. For this color map, we perform the KMeans clustering to obtain 16 different cluster values. Finally, we perform KMeans clustering on the grayscale image to obtain the brightness map with 16 clusters.

The process of stitching two images can be broken down into two main steps: obtaining the homography matrix and performing the actual stitching.

Obtaining the homography matrix involves several sub-steps:

1) Converting the images to grayscale 2) Detecting keypoints in the images using the ”Good Features to Track” method 3)Applying the ”Adaptive Non-Maximal Suppression” technique to reduce the number of keypoints 4) Computing feature descriptors for each keypoint using a method such as ”Scale-Invariant Feature Transform” (SIFT) 5) Performing feature matching to find corresponding keypoints in the two images 6) Filtering the matches using techniques such as the ”Random Sample Consensus” (RANSAC) method to eliminate

outliers Finally, computing the homography matrix using the remaining matches

The actual stitching of the images is performed using the computed homography matrix. This involves:

1) Warping one image using the homography matrix to align it with the other image

2) Computing the bounding box of the warped image to determine the size of the final stitched image

3) Translating the warped image so that its origin is at the top-left corner of the bounding box

4) Blending the two images together by alpha-blending or feathering the boundary between them

5) Returning the final stitched image along with the origin offsets of the warped image in the final stitched image.

All the example plots are provided at the end of the report

I. Phase 2: Deep Learning Approach

In this stage, we've used deep learning approach to estimate the homography between two images—one being the original picture and the other being a warped copy of the original image.

A. Data Generation

The data utilized in this study was obtained from the MSCOCO dataset, which includes color images. Given the vast size of the dataset, a subset was selected and divided into Train, Validation, and Test datasets, consisting of 5000, 1000, and 1000 images, respectively. The images were initially contained in a zip file and were then extracted, transformed into grayscale, and stored in a folder named "Images/Original".

A random patch of 128 x 128 pixels was extracted from each grayscale image and a perturbation with a value of r = 32 was applied. Both the original and perturbed patches were combined and stored. Also untouched and warped patches were saved as well in another folder named "Data/Train/Original and Data/Train/Warped respectively. The difference between the original and perturbed corners was calculated as H4Pt and used as the label for the data. The same process was carried out for the Validation and Test datasets.

In supervised deep learning, the weights of parameters and biases are adjusted based on the error in the previous epoch. The loss metric used in this architecture is the L2 Loss (Mean Squared Error) between the predicted 4-point homography (ˆH4pt) and the ground truth 4-point homography vector (H4pt). In terms of parameter and hyperparameter tuning, the L2 loss function is used with the Adam optimizer. The learning rate used was set to 0.05, but when it was decreased from 0.05 to 0.005, the loss was significantly reduced. The number of epochs also affects the loss, and as the number of epochs increased, the loss significantly decreased over time. The batch size was set to 128, but it was noticed that a low batch size of 64 resulted in a lower loss compared to a batch size of 128, as a lower batch size allows for more corrections to be made in the weights using diverse batches. Even lower batch size of 32 or 16 might have been better but due to time constraints we were not able to test this hypothesis.

B. Supervised Approach:

Utilizing the data that we previously generated, neural net implementation begins by adopting the network architecture from the HomographyNet paper. This design, similar to Oxford's VGG Net, incorporates 3x3 convolutional blocks with BatchNorm and ReLUs . Our convolutional neural network (ConvNet) is implemented using PyTorch's nn module. It has four convolutional layers (conv1, conv2, conv3, conv4), each followed by batch normalization, ReLU activation, and max pooling. The output from the convolutional layers is then flattened and passed through two fully connected layers (fc1 and fc2) before the final output is produced. The network also contains a dropout layer (dropout) with a dropout rate of 0.5, which helps to prevent overfitting. The forward method defines the forward pass of the network, which takes an input tensor and passes it through the defined layers.

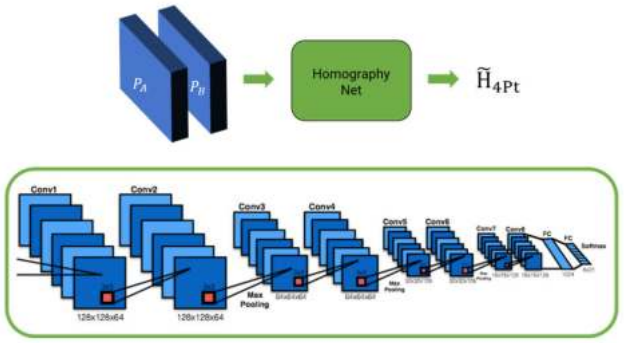

Fig. 5. Supervised Network

Fig. 6. Patch Generation Example

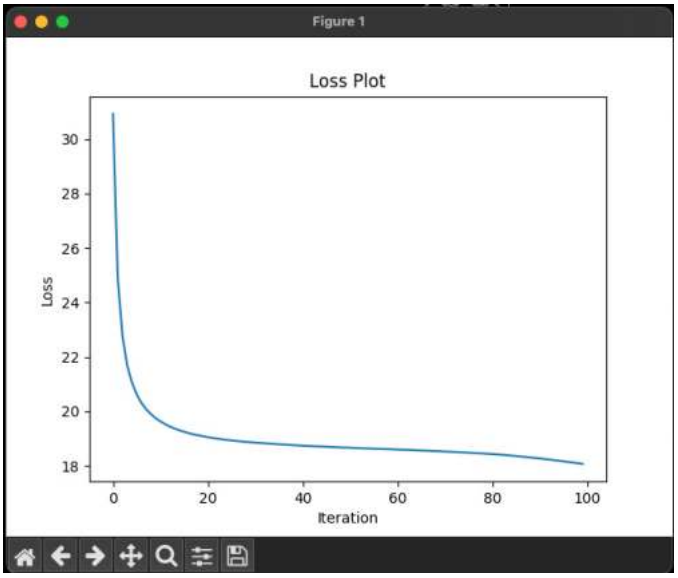

Fig. 7. Train Loss Plot

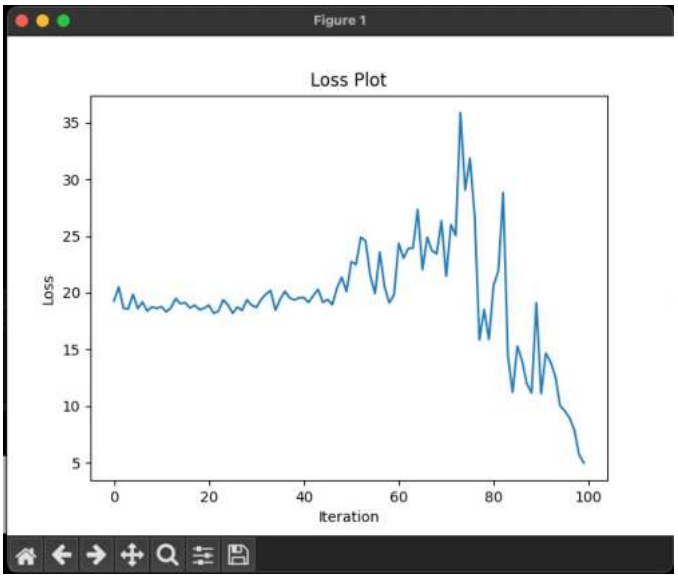

Fig. 8. Validation Loss Plot

C. Unsupervised Approach:

The initial CNN network of the Unsupervised model is similar to the supervised learning part. In addition we have two more components called the Direct Linear Transform (DLT) and Spatial Transformer Network (STN)

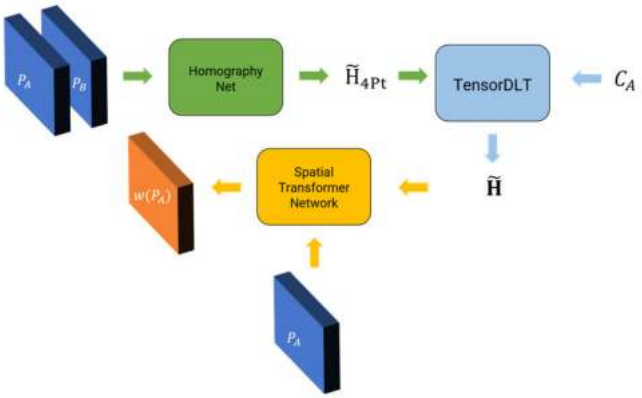

Fig. 9. Unsupervised Network

The Direct Linear Transform (DLT) algorithm takes as input the output from a CNN network, represented as H4Pt, and the corner coordinates of Patch A. Utilizing this information, the algorithm predicts the corners of Patch B. The Patch A and Patch B corner information is used to calculate the Homography matrix (3x3) using a method discussed in the paper.

Spatial Transformer Networks: The homography matrix, generated by the TensorDLT network, is inputted into this network along with the original image. The STN then learns to apply the homography, distorting the original image to form a new patch. Essentially, the network is trained to perform the openCV cv2.warpPerspective() method in a manner that allows differentiation.

Photometric Loss: To evaluate the output of the Spatial Transformer Network, the L1 photometric loss is utilized. This compares the newly created patch with the ground truth patch. If there's a discrepancy, the loss is backpropagated through the network.

We attempted to implement the Direct Linear Transform (DLT) and Spatial Transformer algorithms, however, were unable to fully complete and successfully run the models. There were several factors that could have contributed to the implementation difficulties. One possibility is that the input data was not properly formatted or preprocessed, leading to incorrect results. Another issue could have been a lack of understanding of the mathematical concepts involved in DLT and Spatial Transformer, leading to incorrect implementation. In addition, we had a hard time training network on cluster. It appears that initial script given was not quite right and due to that we were getting "Partition TimeLimit" error and were unable to train for few days until we figured out what was wrong with it. We tried fixing what we could but due to time constraints and some problems we needed to address for phase 1, we were unable to implement all of them.

D. Conclusion

For this project, we attempted to stitch images into a panorama through two methods: a conventional computer vision technique and a deep learning approach. During Phase

1, we focused on utilizing a feature-based method to determine the optimal Homography Matrix for blending the images. In Phase 2, we utilized implemented supervised Neural Network to enhance the accuracy of Homography matrix estimation. Although we encountered some difficulties in Phase 2, we did our best to overcome them. Despite not achieving the desired results, we acquired valuable lessons and consider the project to be a success.

II. Classical Approach Panaroma Generations

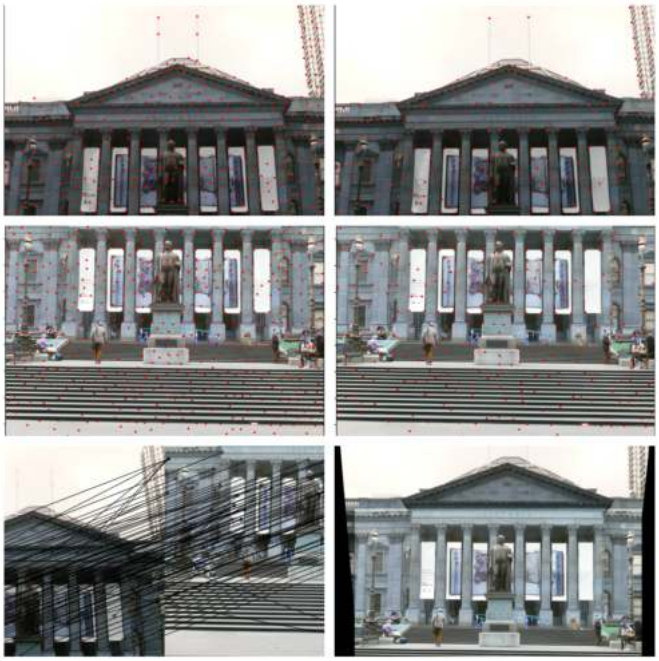

Fig. 11. Train Set 1: Image 1 and 2 stitched

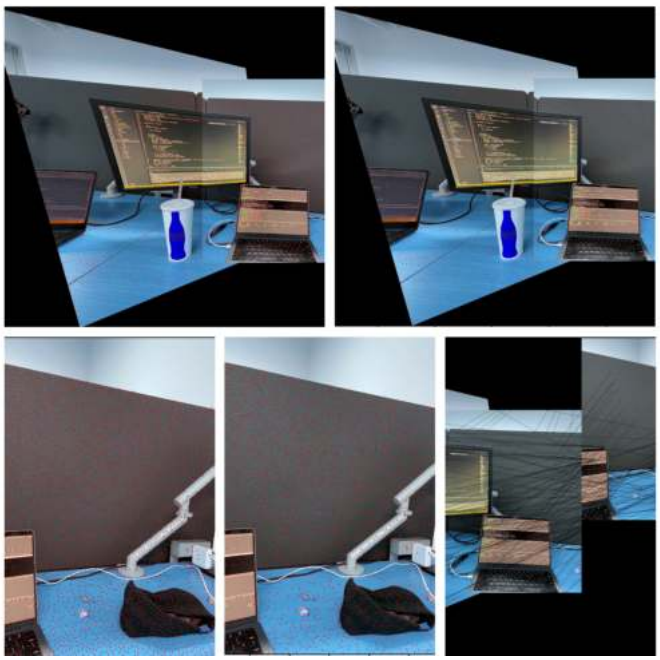

Fig. 10. Train Set 3: Image 1 and 2 stitched

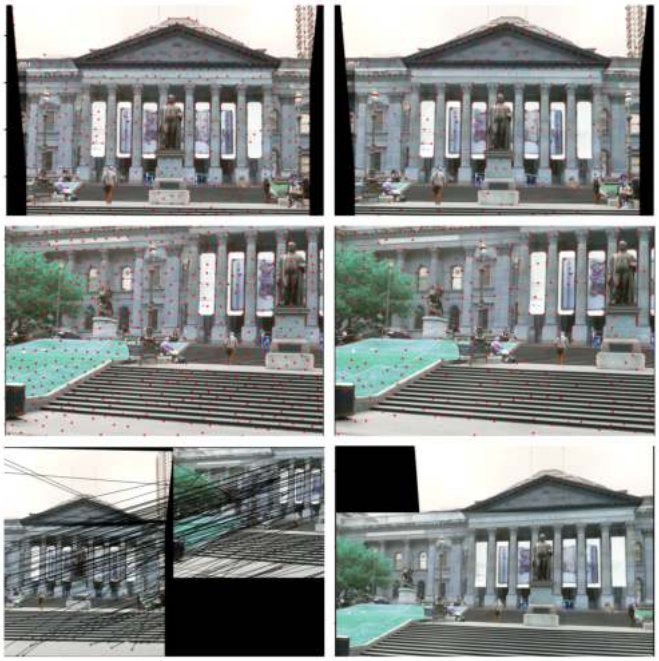

Fig. 12. Train Set 1: Image 1, 2, and 3 stitched

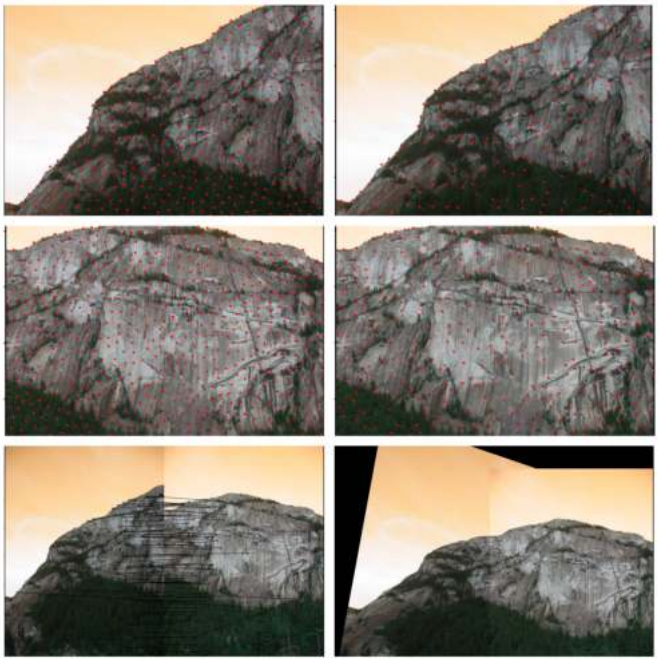

Fig. 13. Train Set 2: Image 1 and 2 stitched

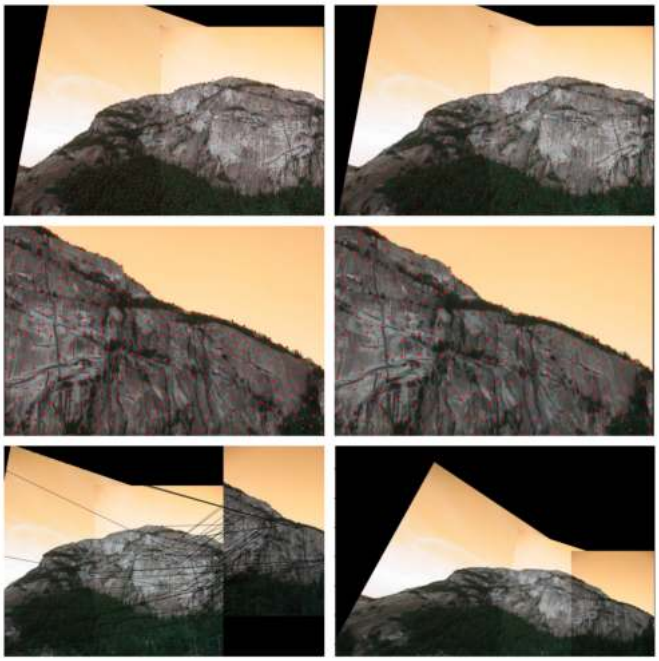

Fig. 14. Train Set 2: Image 1, 2, and 3 stitched

Fig. 15. Custom Set 1: Image 1 and 2 stitched

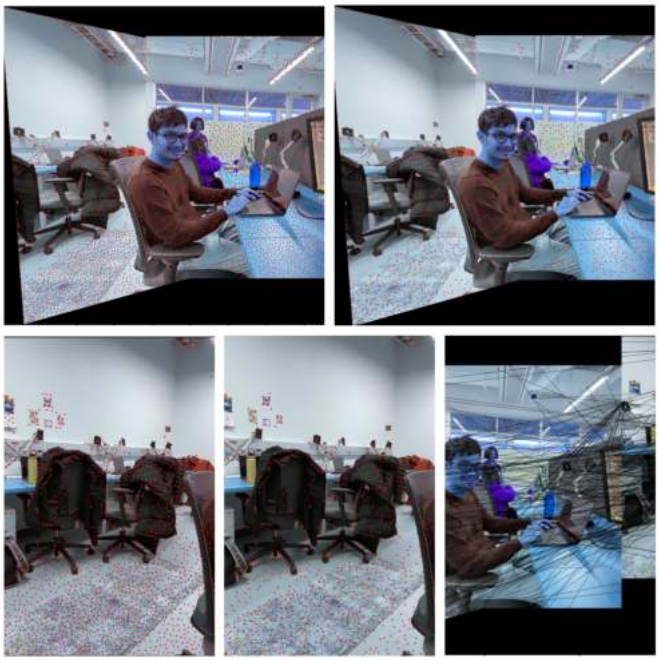

Fig. 16. Custom Set 1: Image 1, 2, and 3 stitched

Fig. 17. Custom Set 2: Image 1 and 2 stitched

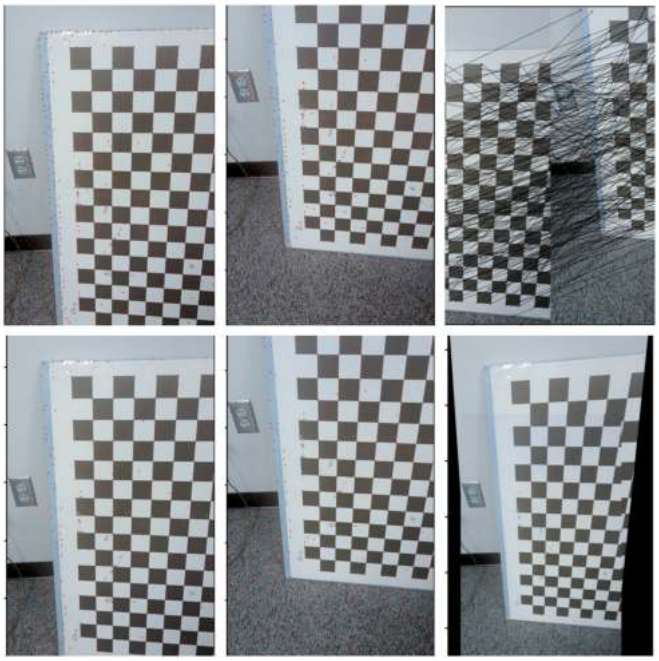

Fig. 19. Test Set 1: Image 1 and 2 stitched

Fig. 18. Custom Set 2: Image 1, 2, and 3 stitched

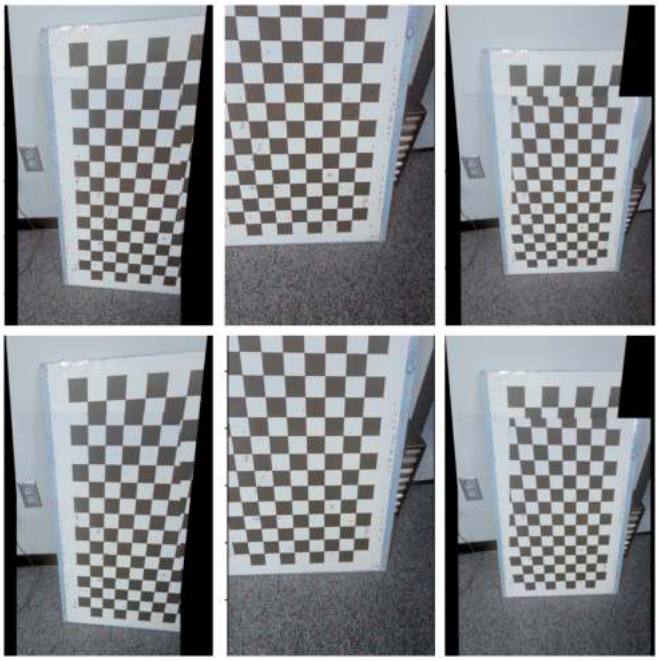

Fig. 20. Test Set 1: Image 1, 2, and 3 stitched

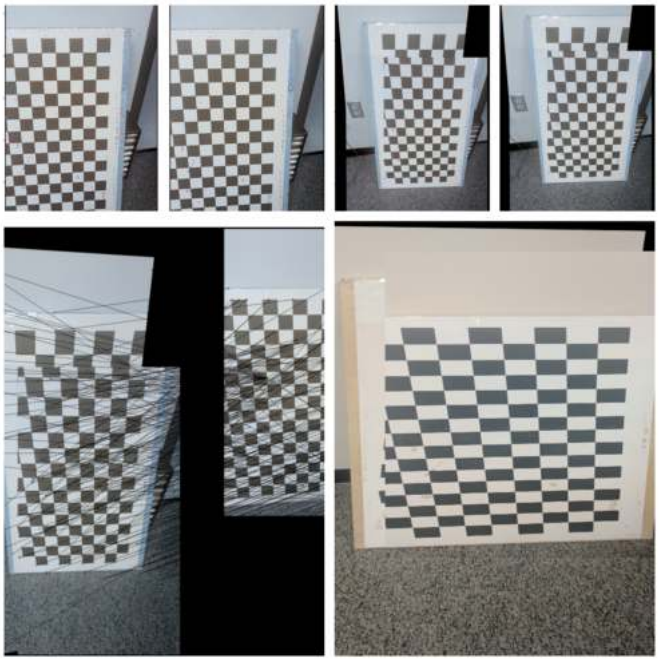

Fig. 21. Test Set 1: Image 1, 2, 3, and 4 stitched

Fig. 22. Test Set 2: Image 1 and 2 stitched

Fig. 23. Test Set 2: Image 1, 2, and 3 stitched

Fig. 24. Test Set 3: Image 1 and 2 stitched

Fig. 25. Test Set 3: Image 1, 2, and 3 stitched

Fig. 26. Test Set 3: Image 1 and 2 stitched

Fig. 27. Test Set 3: Image 1, 2, and 3 stitched

REFERENCES

[1]
1 Daniel DeTone, Tomasz Malisiewicz, Andrew Rabinovich.Deep Image Homography Estimation (2016)
2 Ty Nguyen, Steven W. Chen, Shreyas S. Shivakumar, Camillo J. Taylor, Vijay Kumar. Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model (2018)
3 Eric Brachmann1, Alexander Krull1, Sebastian Nowozin, Jamie Shotton, Frank Michel1, Stefan Gumhold1, Carsten Rother1. DSAC - Differentiable RANSAC for Camera Localization (2018)