CMSC 733 Project 1 Report
My Auto-Pano

Darshan Shah
M. Eng Robotics
University of Maryland
College Park, Maryland 20740
Email: dshah003@umd.edu

Mayank Pathak
M. Eng Robotics
University of Maryland
College Park, Maryland 20740
Email: pathak10@umd.edu

I. Phase 1: Traditional Approach

In this section, we shall look into the implementation of automatic panorama creation using traditional approaches of Image processing and computer vision. The idea to is to find similar matching regions within images, transform the second image to match the reference image and finally blend them together to form one continuous image.

The steps taken to achieve a panorama is as follows: 1) Pre-processing 2) Harris Corner detection and Adaptive Non-maximal Suppression 3) Feature Descriptor and Matching 5) RANSAC outlier rejection 6) Blending Images together. Each of these sections are explained in the following sections.

A. Corner Detection

Corners are considered to be the best interest points and are proved to be effective in abstracting the features of an Image. In order to detect corners, We have implemented Harris Corner detection algorithm. To do so, we first convert both the images into gray-scale and apply the `cv2.goodFeaturesToTrack()` function. This function calculates corner quality measure at every source in the image pixel using Harris Corner detection algorithm and further applies a non maximum suppression to get feature points which are evenly distributed throughout the image. Figure 1 shows the output generated after this process. The number of corners to be detected for this image was restricted to 60 corners.

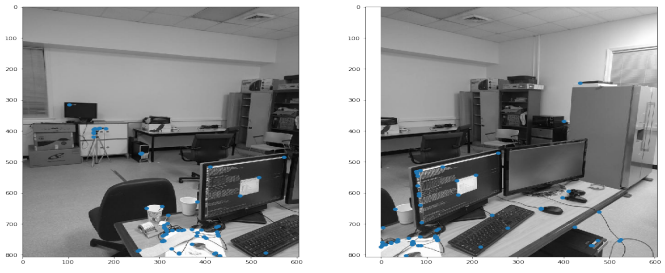


Fig. 1. The points over the image indicate the output after implementing Harris Corner detection and ANMS

B. Feature Descriptor

Next, we create a feature descriptor vector which accurately captures the attributes of the corner detected in the previous step. This is implemented in the `GetDescriptor()` function. The function takes the image and respective key-points as input and outputs a feature vector of size (1,64). A patch of size 40 x 40 is extracted around each corner and a Gaussian blur filter is applied throughout the extracted patch. Once blurred, the patch size is shrunk to 8x8 and further flattened to get a vector of size 64. This vector is standardized so as to make it invariant to brightness and illumination. The result is now a feature descriptor of the given key-point.

C. Feature Matching

Feature matching is the process of finding feature correspondences between two similar images. Feature matching helps find similar regions which are later used to estimate translation and rotation of points of second image with respect to first image. The feature matching is implemented by first taking one feature vector from first image and then computing a sum of squared difference (SSD) with every other feature vector from the second image. The key-points having the least SSD value are said to be in maximum resemblance with each other. To make the computation more efficient, and reduce the number of calculations, We have created a Look-up table which gets populated every time the SSD of new set of vectors is computed. The ratio of the least 2 values corresponding to every feature vector is taken (lowest distance/second lowest distance). If this ratio is less than a threshold value of 0.6, the match is accepted. else, it's discarded. The idea is that an ambiguous or bad match will have a ratio close to 1 whereas unique matches will have a low ratio.

The final set of key-points and their correspondences are displayed using a custom drawMatches() function. The Figure shows the output of key-point matching function. It is clearly evident that there are several mismatches between the elements. These shall be eliminated by further implementing RANSAC on key-points.

D. RANSAC Outlier rejection

The Random Sample Concensus or RANSAC is a robot method which we used to remove incorrect matches. The

Fig. 2. Feature Matching: Similar points are matched between the two images.

Fig. 3. Final Output after the application of RANSAC outlier rejection algorithm.

RANSAC is implemented by using the inbuilt OpenCV package `cv2.findHomography()` which does the job of both performing the RANSAC and computing homography.

E. Image Blending

There are several methods of Stitching two images together. For this project, we have implemented a simple approach: Alpha blending. A blank numpy array is created whose size is equal to the resultant size of the 2 images. The alpha blending technique is implemented using the formula

$$I_{blend} = \alpha I_{left} + (1 - \alpha) I_{right}$$

The downsides of this method is the presence of a seam line. The images being blended have different brightness and illumination which makes the blending unnatural and distinct.

II. Phase 2: Deep Learning Approach

In this phase of the project, Homography estimation is done using two different machine learning approaches: Supervised Learning; and Unsupervised Learning. Deep learning approach uses Convolution neural network to map 4-point homography parameters between images and hence doesn't need to learn full (3×3) Homography matrix. Moreover, this approach skips local feature detection, hence faster than traditional approach.

A. Data Generation

One of the most crucial factors for implementing deep learning networks is the requirement of large amount of data for proper training the network.

Fig. 4. Output After Blending

To meet the requirement, the data is prepared as suggested by the instructors[1]. We generate almost unlimited number of labelled images by random selecting a patch from each image, applying random perturbation on each corners point, finding homography matrix for the random transformation, warping image with the found homography and then saving the images and homography matrix as their labels.

Illustration of these steps are shown in figure 5. The image patches of size 200×200 pixels are stacked upon each other and saved as an image of size 200×200×2. The Homography matrix for this image is saved as label with this image.

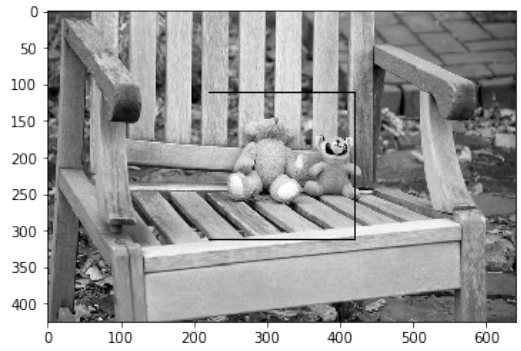
B. Supervised Learning

We implemented a Convolution Neural Network (CNN) for this part of project. The network closely follows the architecture of Oxford's VGG Net. The network contains 8 Convolution layers followed by 2 fully-connected layers. Batch normalization is applied after every convolution layer and a max pooling layer of size 2×2 with stride 2 is applied after every two convolution layers. The network takes input image of resized to 128×128×2 and contains 4 Convolution layers of filter size 64, followed by 4 layers of filter size 128. A layer with dropout probability 0.5 is added before the first fully-connected layer of 1024 units. The last fully-connected layer produces 8 real-valued numbers. The architecture of this network is illustrated in the Figure 6.

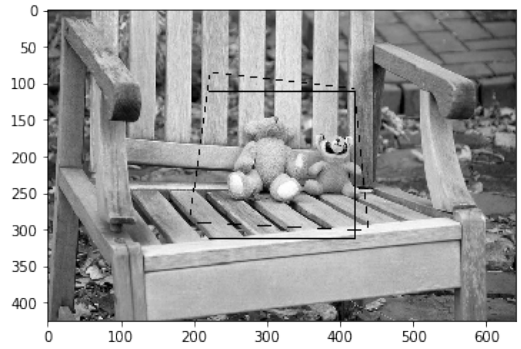
The predicted outputs are then compared with the true labels and the loss is calculated using the Euclidean (L2) loss. The test/train accuracy over epochs and loss value over epochs are then plotted in results.

C. Unsupervised Learning

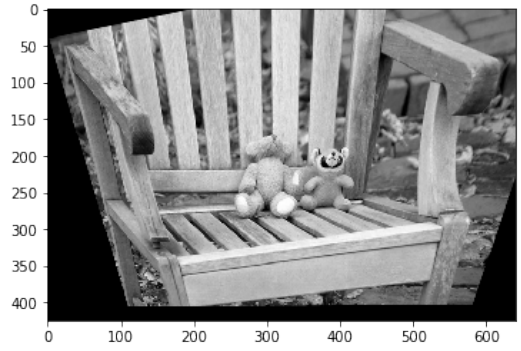
From the results obtained by supervised learning implementation, it can be observed that for images with significant illumination changes or images having large viewpoint difference, the network performance has vast scope of improvement[2]. Also, it requires ground truth labels for training. As stated in the instructions, an unsupervised learning algorithm is



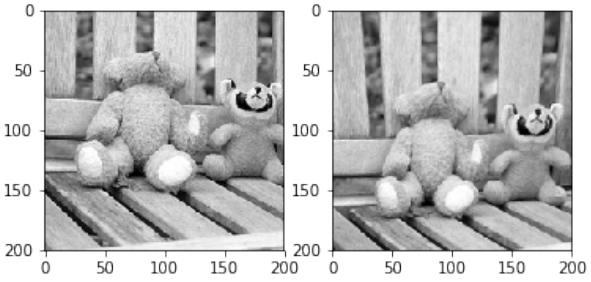
(a) sample training image with patch marked by rectangular lines



(b) Dashed box representing perturbed box



(c) skewed image warped with the homography obtained



(d) Patch P_a and Patch P_b

Fig. 5. Data preparation steps

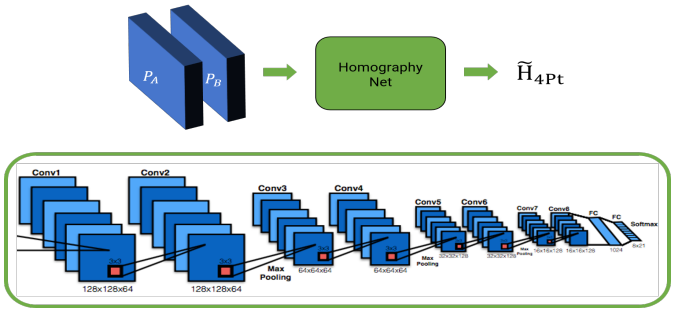


Fig. 6. Supervised HomographyNet Implementation

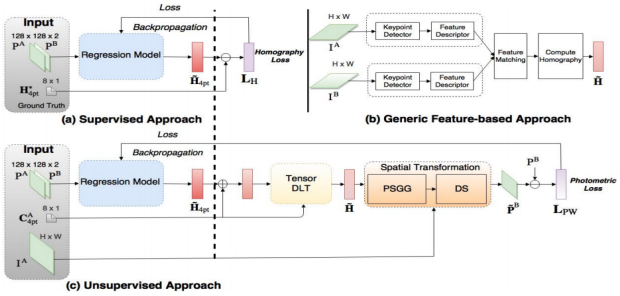


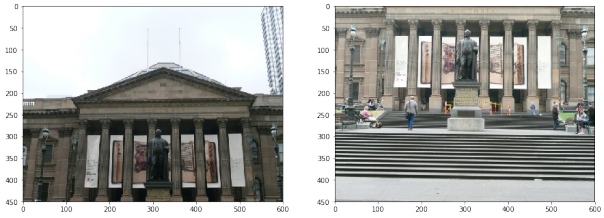
Fig. 7. Comparison of homography estimation methods.

implemented in this later part of this project and has been discussed in this section.

To start with, the same VGGNet architecture is implemented to get the $\tilde{H}_{4pt}$. The dashed line in the figure 7 shows the additions to the regression model that was applied to the supervised learning algorithm. A TensorDLT layer is added followed by a Spatial Transformation layer before calculating the pixel-wise photometric L1 loss. This loss is then backpropogated to the regression model to adjust weights.

1) TensorDLT: Tensor Direct Linear Transform layer is added to obtain the differential mapping from the $\tilde{H}_{4pt}$ to 3×3 homography parametrization matrix. This layer applies linear transformation to tensors, while preserving differential property to support backprop. This approach fails if the corresponding points are collinear.

2) Spatial Transformation Layer: This layer applies the obtained 3×3 homography estimation to the input image, creating a warped image to get the warped pixel coordinates and hence calculating loss. This layer is also differentiable.



(a) Actual Images

(b) Panorama Output using Traditional Approach

Fig. 8. Training Set 1

III. Results

This section shows us the final results obtained by the running the sample images on the traditional panorama algorithm.

A. Training Images

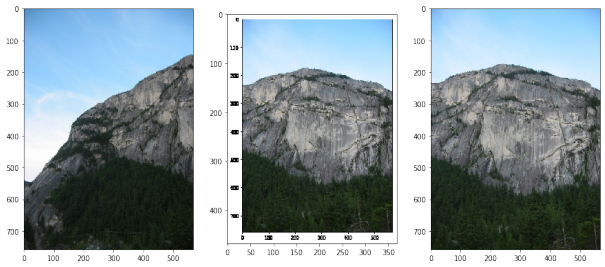
B. Testing Data

This Section Shows the Output from the given testing Data.

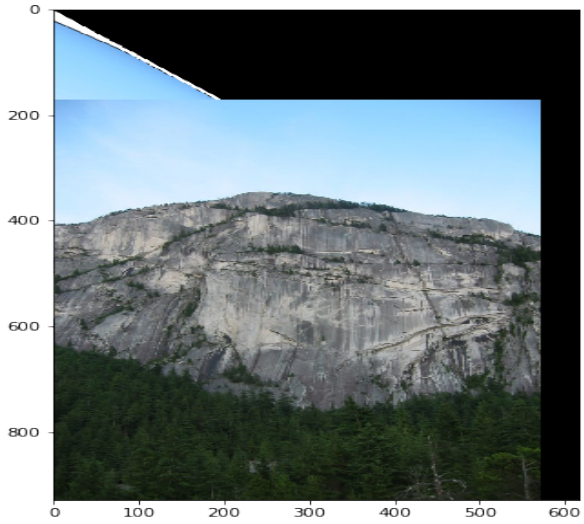
C. Phase2: Deep Network Approach

1) Supervised Learning: We used the starter code provided to implement this phase of our project. The network architecture, as explained earlier, was implemented in 'Network.py' file which returns the $\tilde{H}_{4pt}$ to the 'Train.py' file. However, we are facing some ValueError in tensorflow, and couldn't get it working. We Tried out best to debug this error, but were unsuccessful.

2) Unsupervised Learning: We implemented the Tensor-DLT algorithm as suggested by Ty Nguyen et al. in their paper 'Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model'. Also, the spatial transformation layer is implemented with photometric loss, but due to the

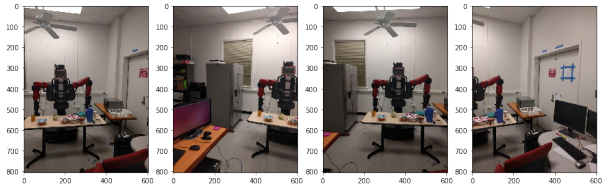


(a) Actual Images

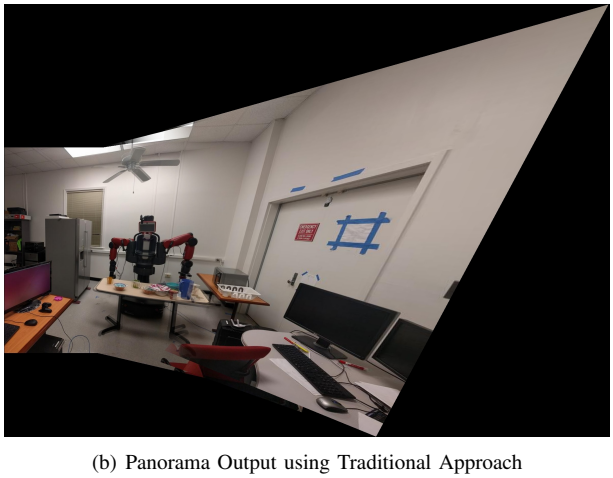


(b) Panorama Output using Traditional Approach

Fig. 9. Training Set 2

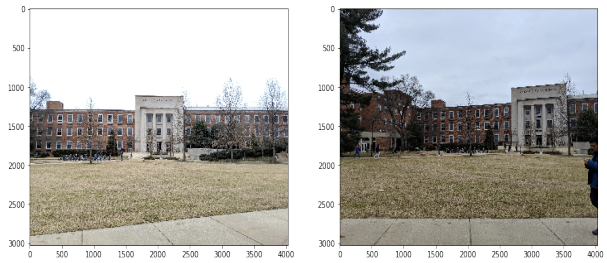


(a) Actual Images



(b) Panorama Output using Traditional Approach

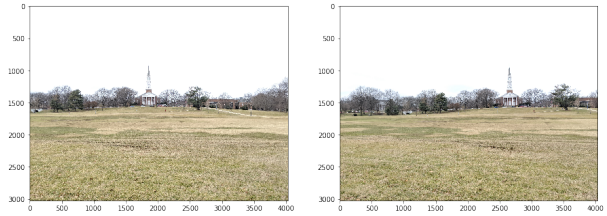
Fig. 10. Training Set 3



(a) Actual Images

(b) Panorama Output using Traditional Approach

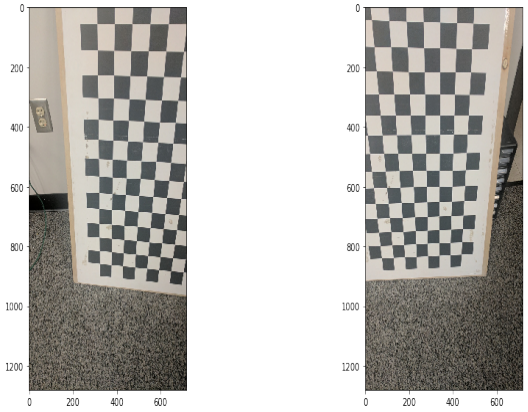
Fig. 11. Custom Training Set 1



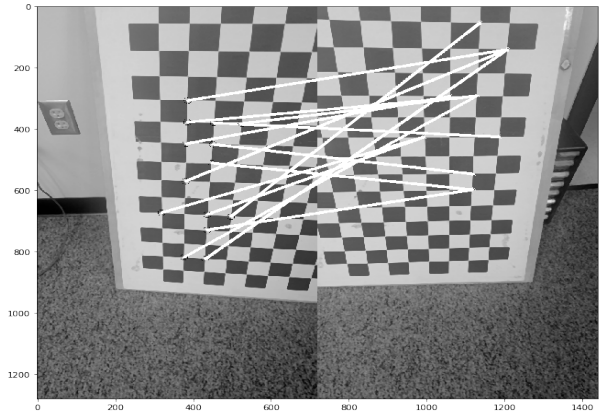
(a) Actual Images

(b) Panorama Output using Traditional Approach

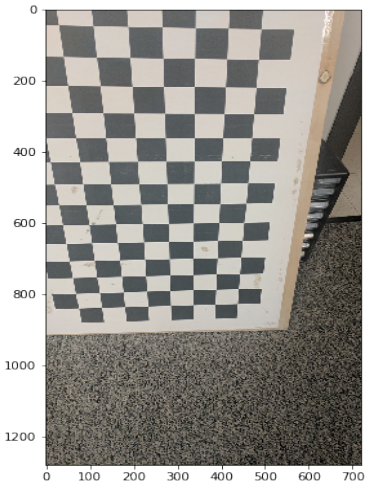
Fig. 12. Custom Training Set 2



(a) Input Images

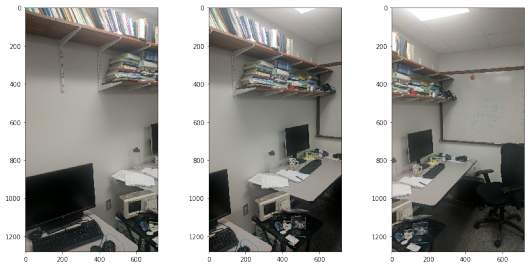


(b) Output after feature Matching: The given test images being symmetric in nature, there are being a lot mismatches in feature matching. leading to a inaccurate panorama creation

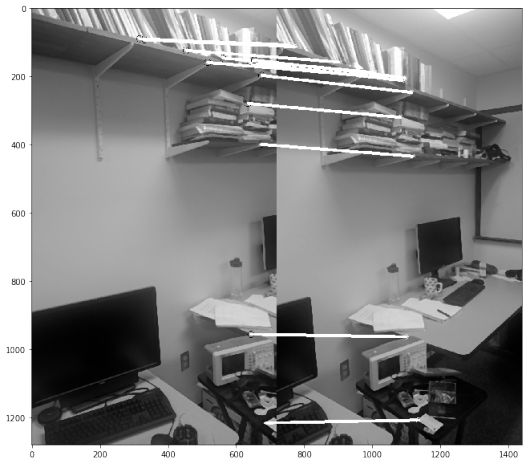


(c) Panorama Output.

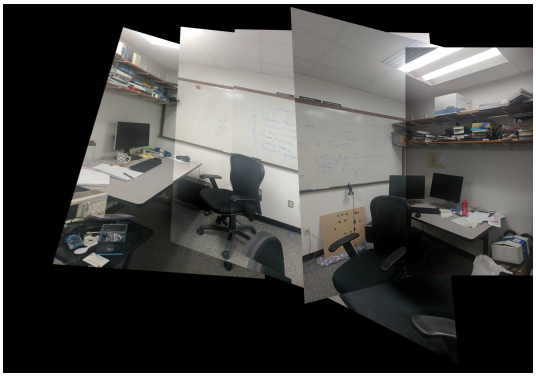
Fig. 13. Results from Test Set 1



(a) Input Images from Test Set 2



(b) Output after feature Matching and RANSAC



(c) Panorama Output for 8 images.

Fig. 14. Results from Test Set 3

above mentioned error in tensorflow, this part is not working as well.

Hence, as of now, the results for this phase are not included in this report. However, we are continuing working on debugging the error and will submit again if results are satisfying.

IV. Conclusion

From our experience with working on this project, We found the traditional methods to be more reliable and easy to begin with. But it appears that Deep learning methods might overcome some of the limitations provided by traditional methods such as Robustness to noise and need for tweaking/preprocessing images according to the scene being utilized. Meaning, same preprocessing steps might not help for images of all kinds. deep learning methods on the other hand, are trained using a diverse dataset of images, hence are more agile and robust in performance. On the downside, they are tedious to develop and require a lot of tweaking and testing before we arrive at the right parameters and hyper-parameters.

References

[1] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *CoRR*, abs/1606.03798, 2016.

[2] Ty Nguyen, Steven W. Chen, Shreyas S. Shivakumar, Camillo J. Taylor, and Vijay Kumar. Unsupervised deep homography: A fast and robust homography estimation model. *CoRR*, abs/1709.03966, 2017.