RBE/CS 549 Computer Vision

P1 - AutoPano

Uthiralakshmi Sivaraman
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
usivaraman@wpi.edu
Using 2 late days

Noopur Koshta
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, USA
nkoshta@wpi.edu
Using 1 late day

***Abstract*—In this project, we presented two different ways of stitching two or more images to create a seamless panorama image. First part explores the traditional approach to find a homography matrix between a set of two images. Second part describes the implementation of a supervised and an unsupervised deep learning approach of estimating homography between synthetically generated data.**

***Index Terms*—Corner Detection, ANMS, Feature Descriptor, Feature Matching, RANSAC, Homography**

I. PHASE 1 : TRADITIONAL APPROACH

In this approach, initially the corners are detected for each image that needs to be stitched together. Once the corners are detected, adaptive non-maximal suppression method is method is used to find local maxima of corners. Features are extracted from each image and then matched.Matched features clubbed with random sample consensus, and the outliers are removed. Homography is estimated and then the images are blended together. Three sample images and their expected panorama is shown in figure 2.

A. Corner Detection

The first step in computing a homography matrix between two images is to detect the corner points. We have detected the corners using both Harris and Shi-Tomashi method.

Harris Corner Detection: The *cv2.cornerHarris* function from OpenCV library uses Harris method for corner detection. The output from this function is the corner strength for each pixel, which is used for adaptive non-maximal suppression.

Shi-Tomashi Corners Detection: The cv2.goodFeaturesToTrack function from OpenCV library uses Shi-Tomashi method along with a non-maximal suppression algorithm to obtain uniform corner points. Hence, the output form the function are corner co-ordinates and not corner strengths. Therefore, we could not use the output from *cv2.goodFeaturesToTrack* for the adaptive non-maximal suppression.

B. Adaptive Non-Maximal Suppression (ANMS)

In this step we find the Nbest corners in the image. This done because in a real image, a corner is never perfectly sharp and it might get a lot of hits in the previous step. The steps involved in ANMS are as follows:

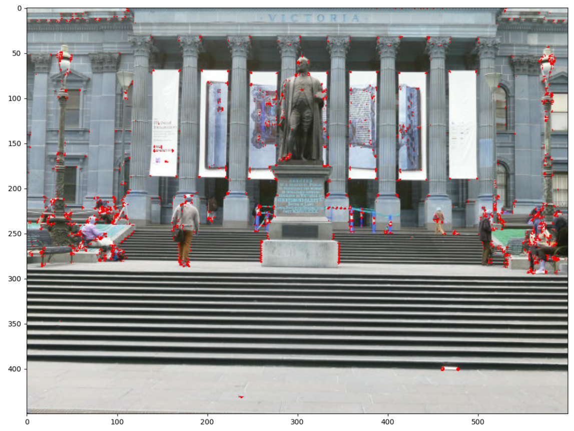


Fig. 1. Harris Corner Output

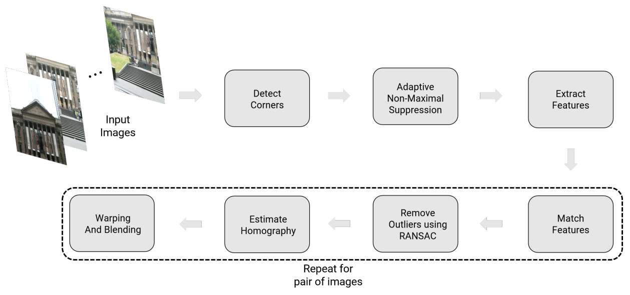


Fig. 2. Overview of panaroma stitching using traditional method

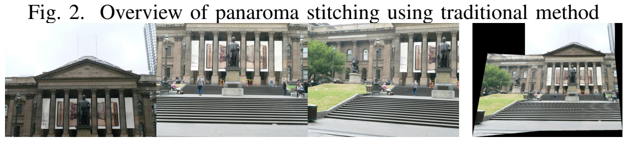


Fig. 3. First three images: Input to the panorama stitching algorithm, last image: output of the panorama stitching algorithm

```
Input  : Corner score Image ($C_{img}$ obtained using cornermetric), $N_{best}$
         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
  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. 4. ANMS algorithm

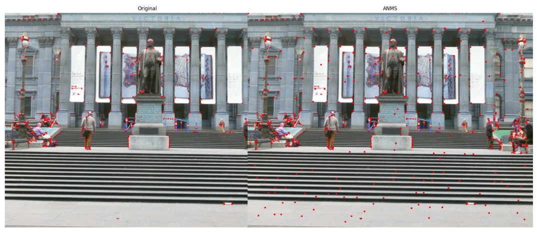


Fig. 5. ANMS 1

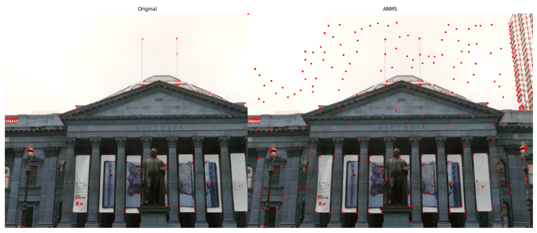


Fig. 6. ANMS 2

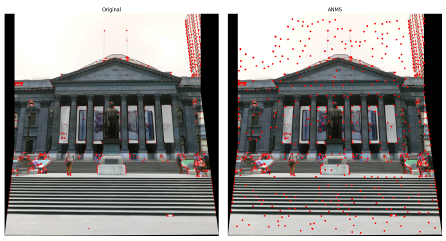


Fig. 7. ANMS 3

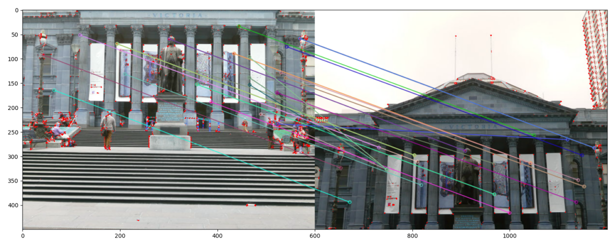


Fig. 8. Feature Matching 1

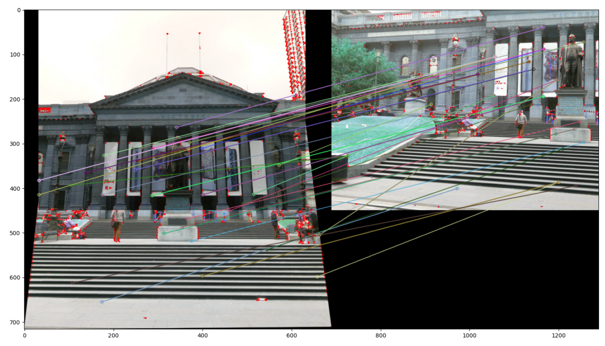


Fig. 9. Feature Matching 2

C. Feature Descriptor

After we get the corner points, we need a descriptor to describe the feature for each point. To obtain that, a patch of size 41 × 41 centered at each corner point is used. This patch is then blurred and sub-sampled to a dimension of 8×8, which is then flattened to obtain a 64 × 1 vector. Then we standardized the vector to have zero mean and variance of 1. Standardization is used to remove bias and to achieve some amount of illumination invariance.

D. Feature Matching

Now that we have a feature descriptor for each corner point, we can find the point matches from two images. For a corner point from image 1, we computed the sum of square differences between all points in image 2. Then we found a best match as the point with lowest distance and a second best match with second lowest distance. If the ratio of lowest distance and second lowest distance is less than a particular value, we accept the matched pair, else we reject it.

E. RANSAC for outlier rejection and to estimate Robust Homography

We know that all matches we get are not important which are called outliers. If we don't remove these outliers they

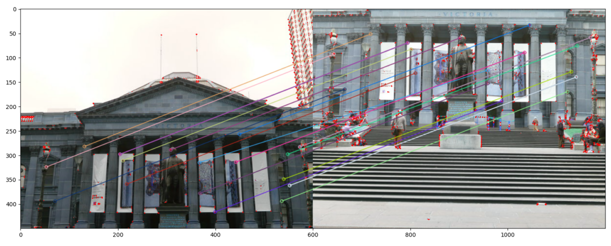


Fig. 10. RANSAC 1

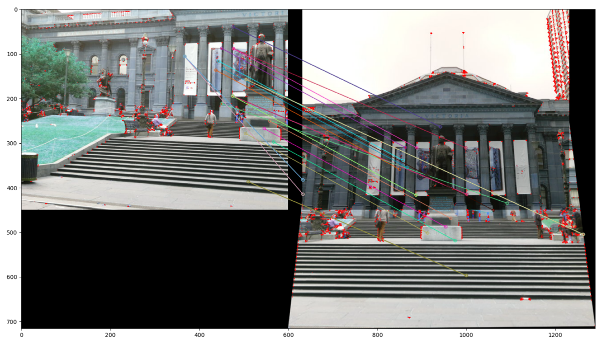


Fig. 11. RANSAC 2

will cause problem in stitching. We also compute Homography matrix after eliminating outliers. Homography is the transformation between 2 images. Homography has rotational and translation components in 2D space. Thus we finally find Homography matrix of inliers of both the match pairs. We keep on iterating till we get good inliers from the match pairs.

*1) **For Outliers Rejection:**:* In Outliers rejection we compute Homography matrix and multiply its inverse with the image to get the second image reference to first image, We compute predicted match with the target match and threshold the outliers.

*2) **To Compute Homography:**:* We compute Homography again on the inliners after we reject them earlier on basis of threshold. We pass this homography matrix to get the warp perspective with the image 1.

F. Stitching and Blending Images

After obtaining homography between the images, we perform warping and stitching. The stitching is performed in a sequential manner i.e., the images are selected in order from left to right and/or top to bottom. This is done in the following way:

1) : Compute homography between N 1 and Nth image. Here N = 2, 3..

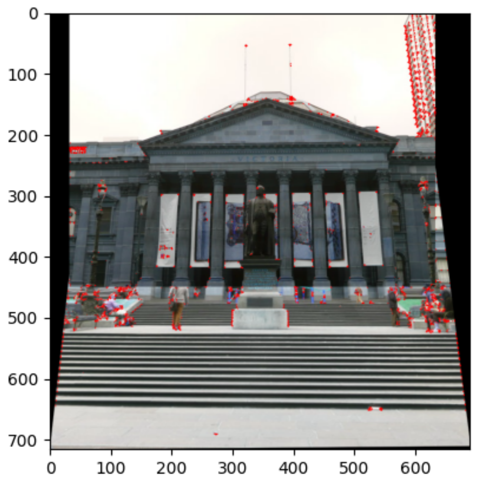


Fig. 12. Stitched Image

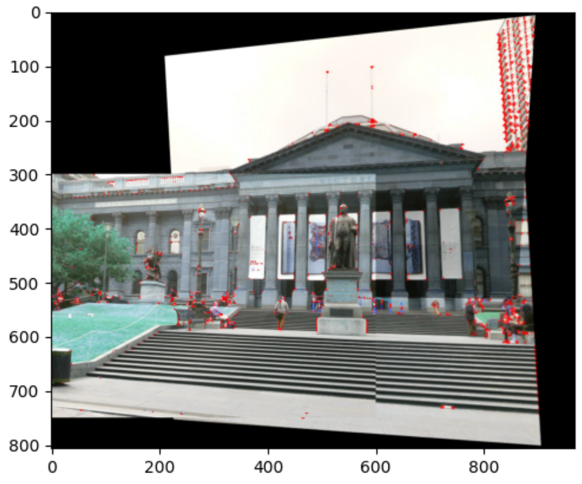


Fig. 13. Panaroma Output for Train Set 1

2) : Apply this homography to the 4 corner points of the N 1 th image and obtain the minimum X and Y translation of the image. Then, remove this offset from the homography matrix such that top-left corner of the N 1 th image is (0, 0) and warp the N 1 image using this new homography matrix.

3) : A bigger frame is then constructed, which will contain the stitched images for the panorama, by adding the sizes of the warped(N 1) image and the Nth image and place these images in this frame.

4) : Use the pano image created in the previous step and the N + 1th image to the repeat the steps 1-3.

G. Conclusion

Through Classical method , we were able to stitch 2 or 3 images but did not work well for 4-5 images. One of the reasons I found why there were some discrepancies were because the number of features matches did not cover the entire image range. In the example of Museum Picture, the feature matching for stairs were almost not present an hence as you can see from the final panaroma, the stairs alone are not aligned. One of the key observations I found is that if we increase the number of iterations, RANSAC works best. We have increased number of iterations from 1000 till 8000. Also, match ratio for feature matching of 0.2 and distance threshold of 15 for RANSAC works well for Museum case.

II. PHASE 2 : DEEP LEARNING APPROACH

In this method we implement the entire pipeline of traditional Panorama stitching using deep learning. Deep learning here is used to compute the homography matrix between several images. Robust and fast Homography estimation is required applications, especially in robotics, to estimate pose between multiple aerial images for collaborative autonomous exploration and monitoring. Both supervised and unsupervised approach have been implemented to compute the homography. The empirical results presented in the paper demonstrate that compared to traditional approaches, the unsupervised algorithm achieves faster inference speed, while maintaining comparable or better accuracy and robustness to illumination variation. Furthermore, the unsupervised method has superior adaptability and performance compared to the corresponding supervised deep learning method.

A. Data Generation

In order to train the networks we create synthetic dataset. For the supervised approach we generate synthetic image data set with corresponding labels. Here labels are our ground truth which is only used while training the supervised network. We generate the synthetic data set from MSCOCO dataset which contains a lot of objects in natural images. Since the convolution doesn't accept image sizes of arbitrary shape we crop two patches from the same image namely patch A and patch B. In order to to get a complete patch of dimensions 128x128, the left top corner of a patch can only be placed in a limited area which is shown in figure 15. The first patch, patch A is a square patch of the dimension 128x128 as shown in figure 32. Then, in order to create a different patch(or in a sense image) the corners of the first patch are perturbed i.e each corner point's location xi=(xi ,yi) is changed by a small amount in the range $[-\rho, \rho]$ where ρ is 16. This perturbation is saved as ground truth for our supervised model. The perturbed patch is not a perfect square of dimension 128x128 (dimensions accepted by our network) and thus, to get a perfect square patch, we inverse warp the original image, it uses the homography matrix. This homography matrix is computed using *cv2.getPerspectiveTransform* between corner point of patch A and corner points of patch B (which is equal to sum of each corner point of patch A and its respective perturbation) The following data fields have been saved for supervised approach :

- Patch1 and Patch2 stack
- H˜ 4pt ground truth

For the Unsupervised part data fields saved are as followed:

- Patch1 and Patch2 stack
- Corner points of patch in the first image
- Reference to original image from which the patch was cropped

B. Supervised Approach

In the paper the task of computing homography is formulated as a classification problem and regression problem and here we have implemented the regression network. The only difference is that in the classification we discretize the output into 8 points and 21 bins (8x21 output) while in the regression one the output is 4 point parameterization of homography.

Architecture: This network uses 3x3 convolutional blocks with Batch-Normalization and ReLU activations, and are architecturally similar to Oxford's VGG Net [3]. The network uses eight convolutional layers with a max pooling layer, of pool size = 2 × 2, stride = 2, after every two convolutions. The first four convolutional layers have 64 filters per layer and the next four convolutional layers use 128 filters. These convolutional layers are followed by two fully-connected Dense layers. Dropout with a probability of 0.4 is applied after the last convolutional layer and the first fully-connected layer, to avoid overfitting. The first Dense layer has 1024 units and the final Dense layer outputs 8 units. The model block diagram is shown in fig.

Training parameters: We used SGD optimizer with a learning rate of 0.0005 and momentum of 0.9 to train our model for 50 epochs with 5000 images. We calculate euclidean L2 Loss- Mean Squared Loss as suggested in the official implementation and use mean absolute error as our training performance metric.

C. Unsupervised Approach

Supervised model is more biased since it is dependent on the ground truth values. It is very exciting as well as complex to use deep learning to learn the Homography matrix and warped images. In this section, we deal with unsupervised method. We implemented the TensorDLT algorithm as suggested in the paper 'Unsupervised Deep Homography: A Fast and Robust Homography Estimation Model' This Converts H4pt predicted to Homogeneous matrix. Also Spatial Transformation layer is implemented to compute Photo-metric loss which is the L1 loss. We back propagates to optimize the weights. This model is not biased like the supervised model.

Architecture of the unsupervised network consists of mainly three parts:

1) HomographyNet -: This network is same as Supervised network and the output of this network is 4-point Homography.

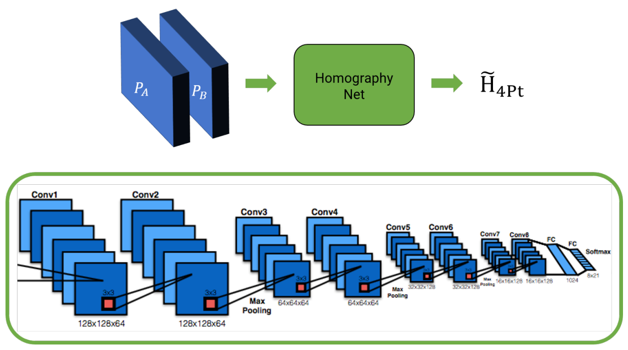


Fig. 14. Extracted random patch PA from original image IA is shown as dashed blue box. Corners of the patch PA denoted by CA are are shown as blue circles.

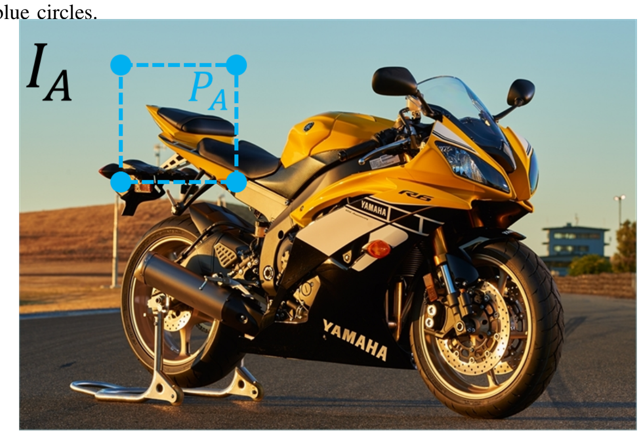


Fig. 15. Extracted random patch PA from original image IA is shown as dashed blue box. Corners of the patch PA denoted by CA are are shown as blue circles.

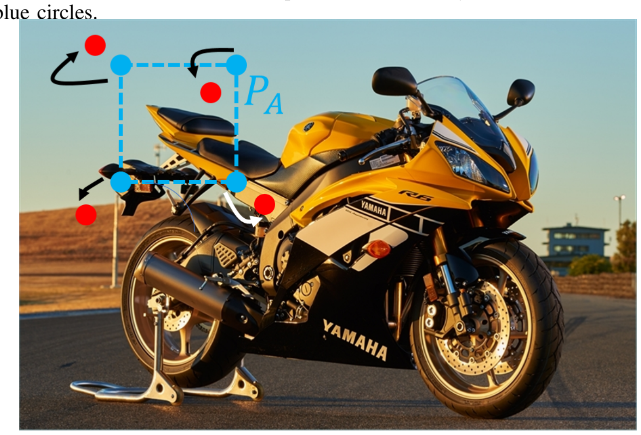


Fig. 16. Random perturbation applied to corners of the patch PA denoted by CA are shown as blue circles to obtain corners of patch PB denoted by CB are shown as red circles.

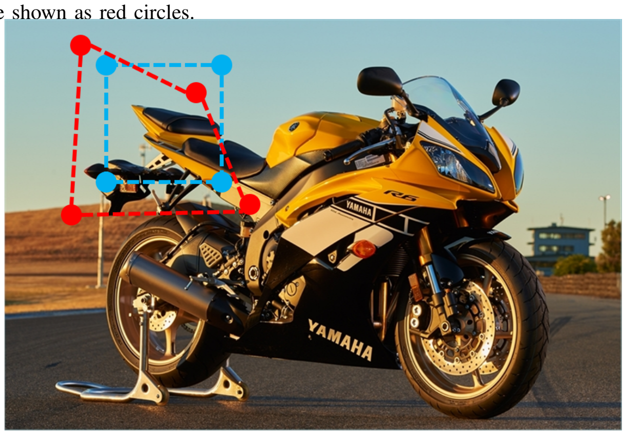


Fig. 17. Red dashed lines show the patch formed by perturbed point corners

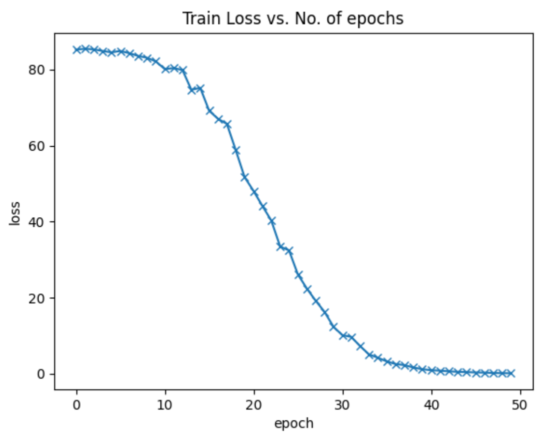


Fig. 18. Training Loss vs Number of Epochs for Supervised Network

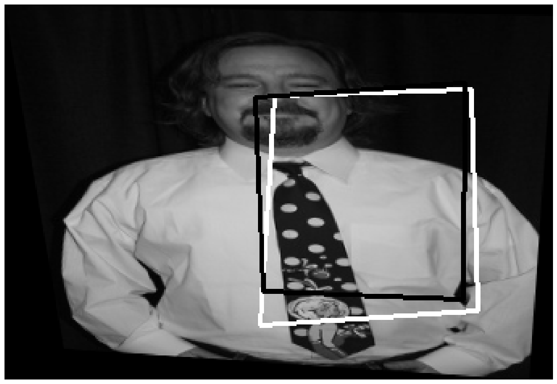


Fig. 19. Predicted Patch vs Original Patch Test Image 1

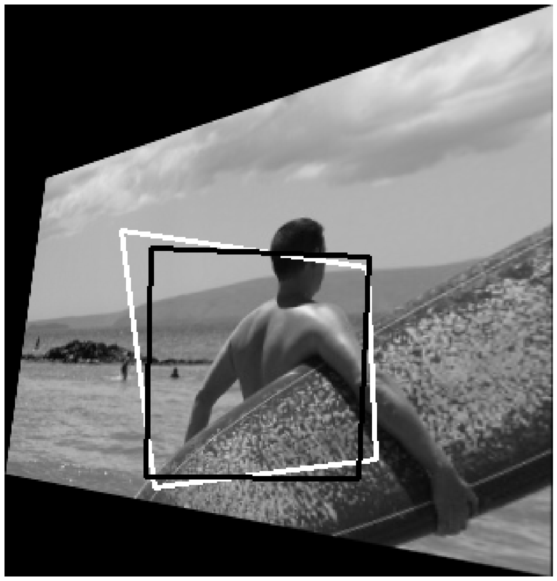


Fig. 20. Predicted Patch vs Original Patch Test Image 2

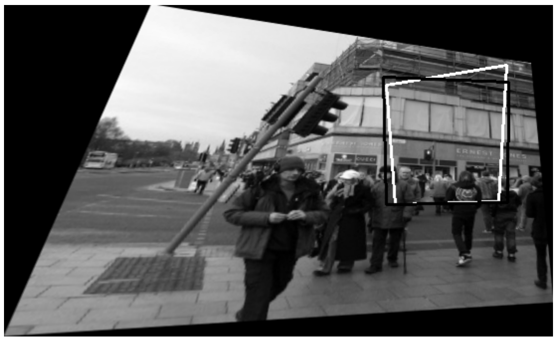


Fig. 21. Predicted Patch vs Original Patch Test Image 3

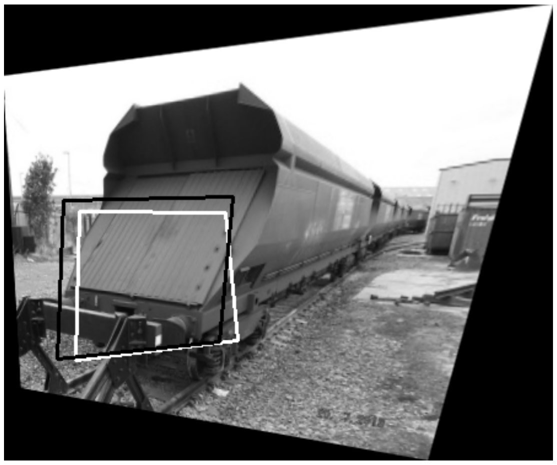


Fig. 22. Predicted Patch vs Original Patch Test Image 4

2) TensorDLT -: This layer of network is used to implement DLT to get the 3x3 Homography Matrix in a differentiable way. For implementing this part, We have used the official paper and found the H matrix based on PyTorch tensor operations rather than using numpy array computation.

3) Spatial Transformer Layer -: The next layer applies the 3x3 homography estimate H output by the Tensor DLT to get warped images. These warped images are necessary in computing the photometric loss function i.e. L1 loss subtracting

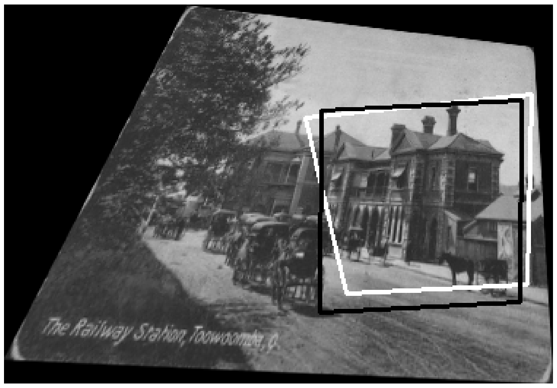


Fig. 23. Predicted Patch vs Original Patch Test Image 5

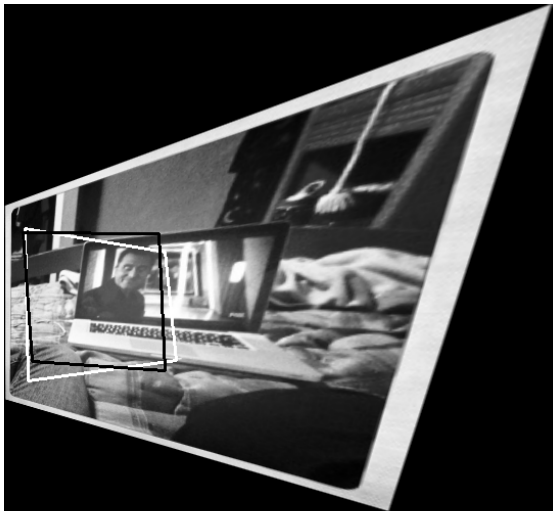


Fig. 24. Predicted Patch vs Original Patch Test Image 6

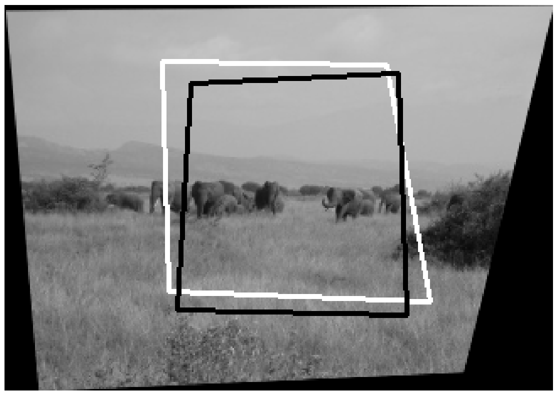


Fig. 25. Predicted Patch vs Original Patch Test Image 7

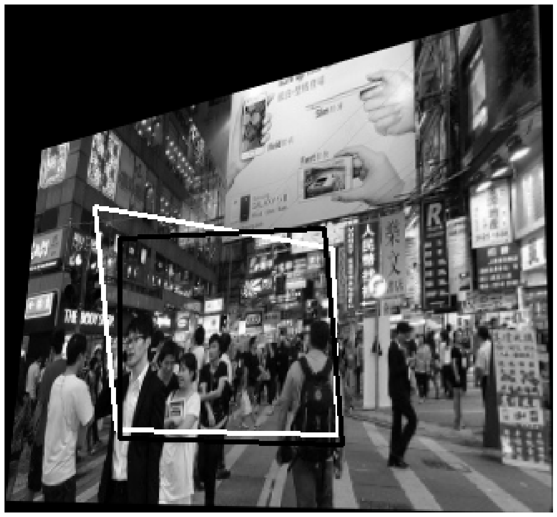


Fig. 26. Predicted Patch vs Original Patch Test Image 8

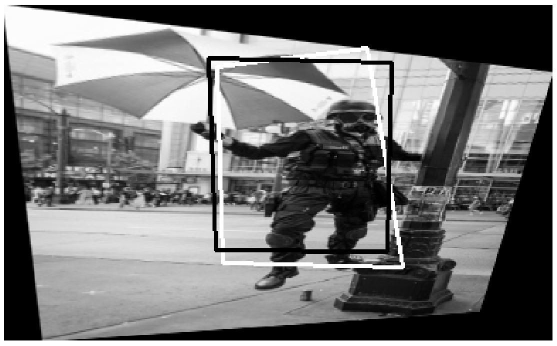


Fig. 27. Predicted Patch vs Original Patch Test Image 9

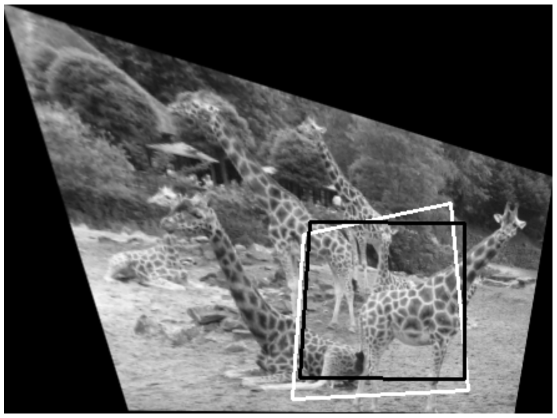


Fig. 28. Predicted Patch vs Original Patch Test Image 10

the predicted warped image with the patch warped image. We have used Kornia for finding warped patch given Hmatrix and Original Image.

D. Conclusion

The supervised model for Homography estimation was able to well generalise for test cases. The loss valye started from around 85 reduced to 0.32 over 50 epochs when trained with SGD optimizer for leaning rate of 0.0005 and momentum of 0.9.

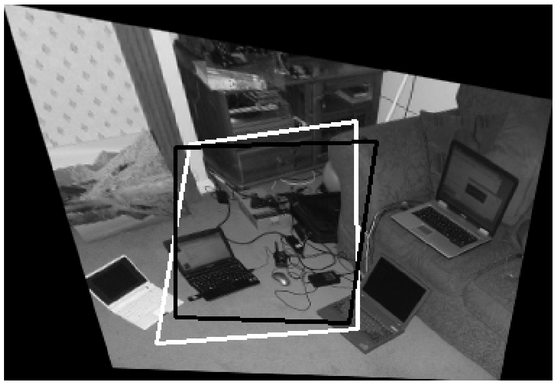


Fig. 29. Predicted Patch vs Original Patch Test Image 11

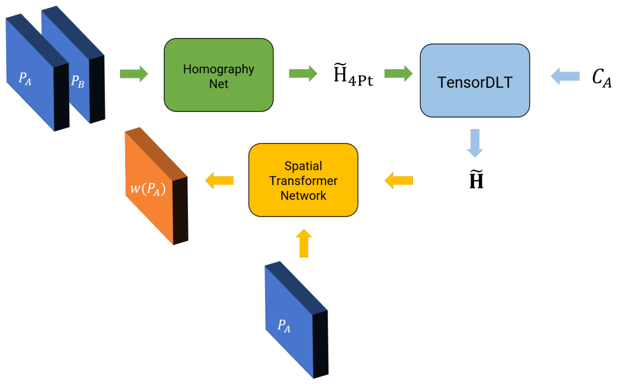


Fig. 30. Overview of the unsupervised deep learning system for homography estimation

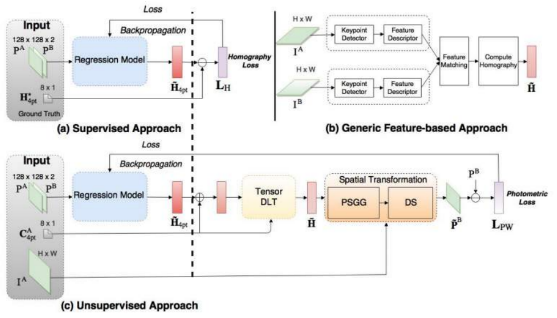


Fig. 31. Overview of homography estimation methods; (a) supervised deep learning approach; (b) Feature-based methods; and (c) unsupervised method

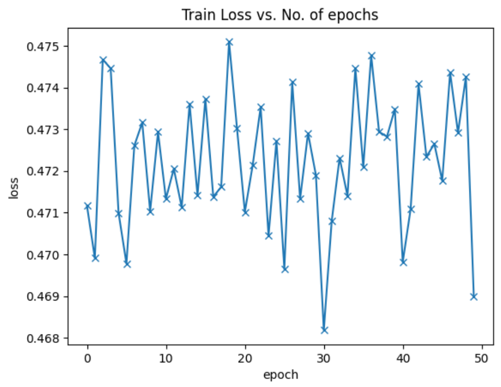


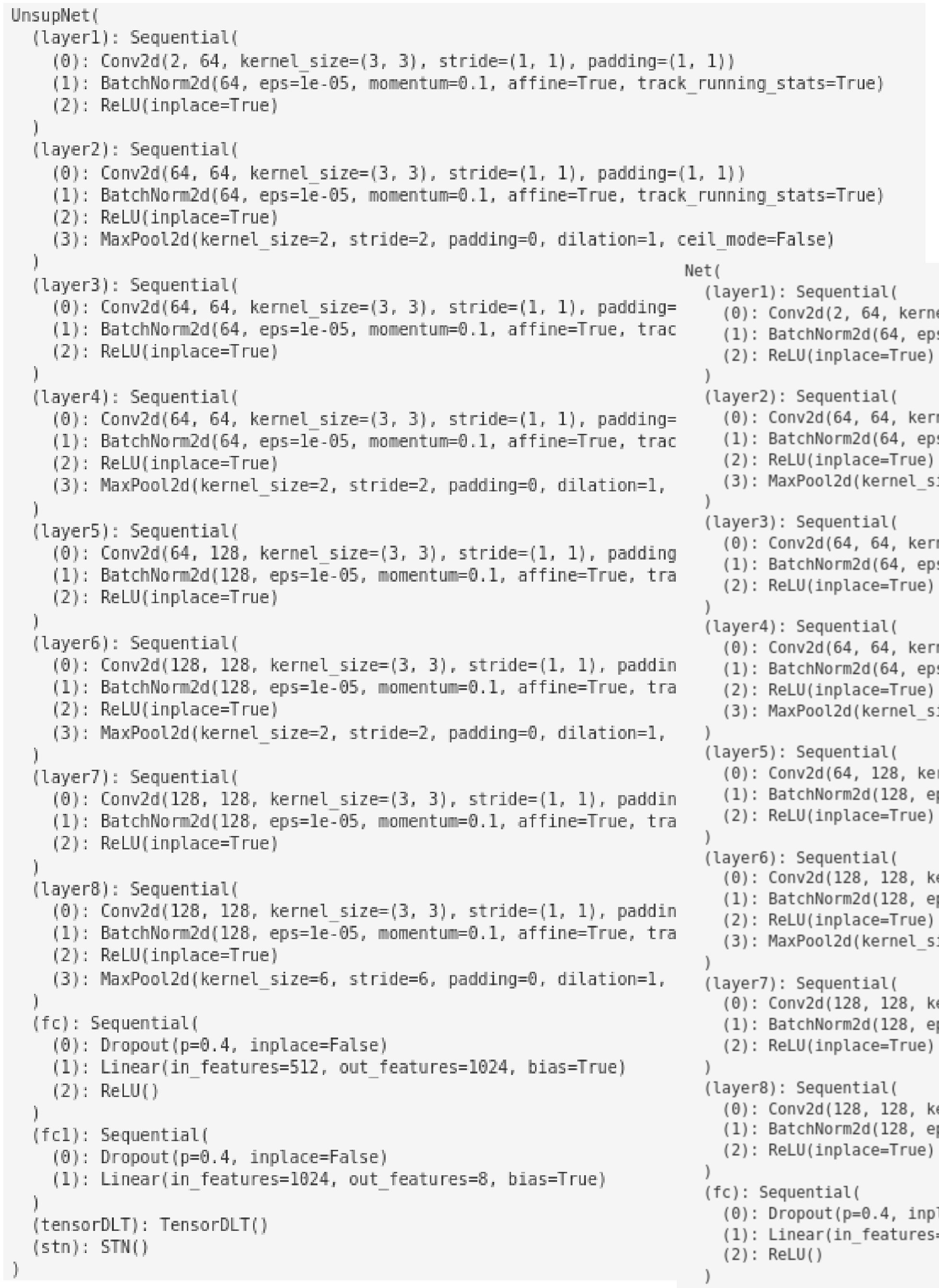
Fig. 32. Training Loss vs Epochs for Unsupervised

The Unsupervised model gave L1 loss value in 0.47 range but there was not much reduction after training over 50 Epochs. Further work would be train for more number of epochs and test the unsupervised model trained. Due to time constraints, we were unable to test the unsupervised model.

References

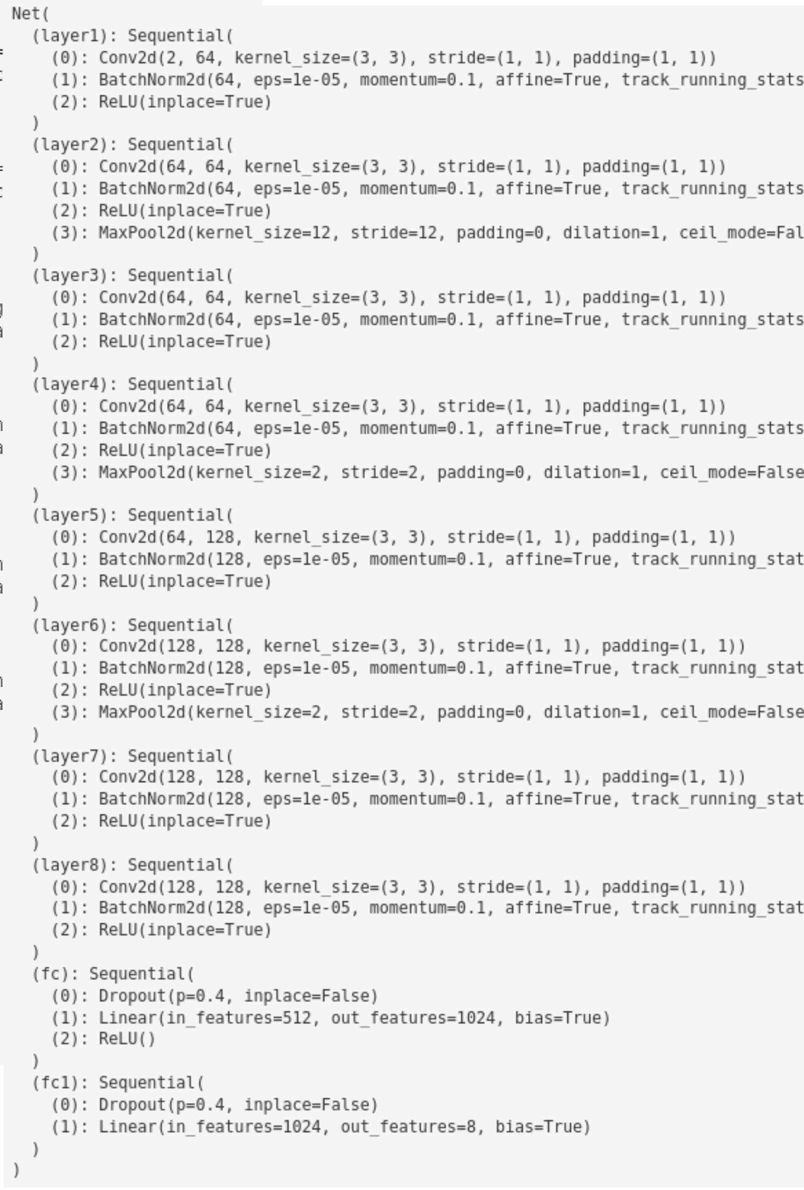
[1] https://medium.com/@sergioalves94/deep-learning-in-pytorch-with-cifar-10-dataset-858b504a6b54.

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



```
UnsupNet(
  (layer1): Sequential(
    (0): Conv2d(2, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (layer3): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
    (2): ReLU(inplace=True)
  )
  (layer4): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, trac
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
  )
  (layer5): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
    (2): ReLU(inplace=True)
  )
  (layer6): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), paddin
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
  )
  (layer7): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), paddin
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
    (2): ReLU(inplace=True)
  )
  (layer8): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), paddin
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, tra
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=6, stride=6, padding=0, dilation=1,
  )
  (fc): Sequential(
    (0): Dropout(p=0.4, inplace=False)
    (1): Linear(in_features=512, out_features=1024, bias=True)
    (2): ReLU()
  )
  (fc1): Sequential(
    (0): Dropout(p=0.4, inplace=False)
    (1): Linear(in_features=1024, out_features=8, bias=True)
  )
  (tensorDLT): TensorDLT()
  (stn): STN()
)
```

Fig. 33. Model Architecture Unsupervised



```
Net(
  (layer1): Sequential(
    (0): Conv2d(2, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (2): ReLU(inplace=True)
  )
  (layer2): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=12, stride=12, padding=0, dilation=1, ceil_mode=Fal
  )
  (layer3): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (2): ReLU(inplace=True)
  )
  (layer4): Sequential(
    (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
  )
  (layer5): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
    (2): ReLU(inplace=True)
  )
  (layer6): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False
  )
  (layer7): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
    (2): ReLU(inplace=True)
  )
  (layer8): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stat
    (2): ReLU(inplace=True)
  )
  (fc): Sequential(
    (0): Dropout(p=0.4, inplace=False)
    (1): Linear(in_features=512, out_features=1024, bias=True)
    (2): ReLU()
  )
  (fc1): Sequential(
    (0): Dropout(p=0.4, inplace=False)
    (1): Linear(in_features=1024, out_features=8, bias=True)
  )
)
```

Fig. 34. Model Architecture Supervised

[3] Nguyen, Ty, et al. "Unsupervised deep homography: A fast and robust homography estimation model." IEEE Robotics and Automation Letters 3.3 (2018): 2346-2353.

[4] https://kornia.readthedocs.io/en/latest/geometry.html

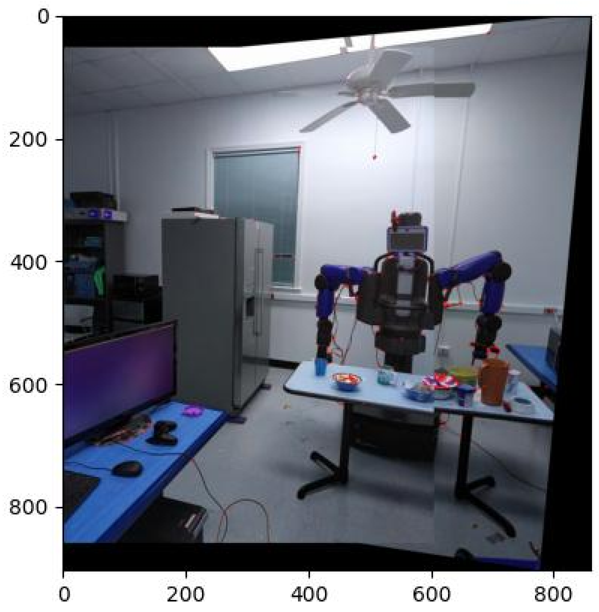


Fig. 35. Panorama Output for Train Set

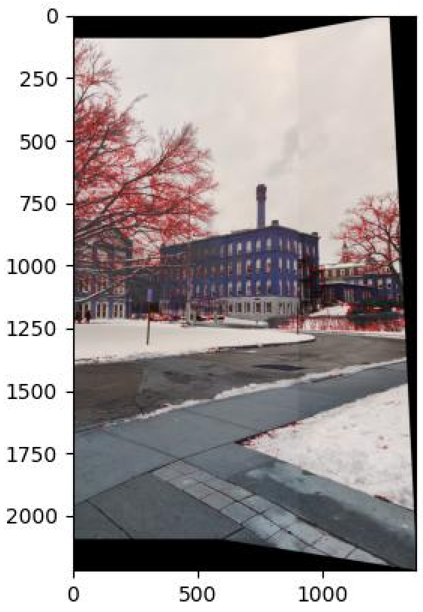


Fig. 36. Panorama Output for Custom Set taken from WPI Campus