RBE549 Project 1
My AutoPano

Manideep Duggi
Masters in Robotics
Worcester Polytechnic Institute
Worcester, MA 01609
Email: mduggi@wpi.edu

Pavan Ganesh Pabbineedi
Masters in Robotics
Worcester Polytechnic Institute
Worcester, MA 01609
Email: ppabbineedi@wpi.edu

***Abstract*—MyAutoPano (Phase 1)**

I. Introduction

Implementation of **MyAutoPano (phase 1)** a traditional computer vision approach to create panoramic images from multiple overlapping photographs. The process that we followed involves corner detection using the **Shi-Tomasi method**, followed by **Adaptive Non-Maximal Suppression (ANMS)** for even corner distribution. Feature descriptors are created using image patches, which are then matched between images.Then we used **RANSAC** to remove incorrect matches and compute **homography** between images. The final step involves stitching and blending the aligned images to create a seamless panorama, with the system capable of handling multiple images while detecting cases with insufficient matching features.

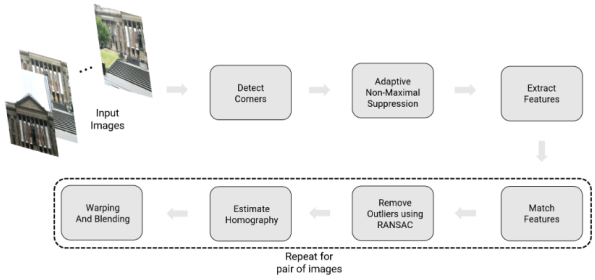

Fig. 1. Pipeline for the creating panorama

A. Corner Detection

The corner detection implementation utilizes the Shi-Tomasi method through OpenCV's **goodFeaturesToTrack** function, which finds the most prominent corners in the image or in the specified image region. The algorithm processes multiple **.jpg** images, first converting each image to grayscale for corner detection. The corner detector is configured to identify up to 1600 strongest corners with a quality threshold of 0.001 and ensuring a minimum Euclidean distance of 8 pixels between detected corners. Each detected corner is visually marked on the original image with a small red circle (radius 3 pixels).This approach ensures consistent and reliable corner detection across multiple images, which is crucial for the subsequent steps in the panorama stitching pipeline.

Fig. 2. Output after corner detection for an Image 1 from Set 1

Fig. 3. Output after corner detection for an Image 2 from Set 1

Adaptive Non-Maximal Suppression (ANMS)

Adaptive Non-Maximal Suppression (ANMS) is implemented to achieve a more uniform distribution of corner features across the image. The algorithm operates with two key parameters:

- The input corners detected by the Shi-Tomasi detector.
- N_{best}, set to 600 in the implementation, which determines the number of corners to retain.

The process begins by computing a suppression radius for each corner, representing the minimum Euclidean distance to a stronger corner. The steps are as follows:

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. 4. ANMS Algorithm

1. For each corner i, initialize its suppression radius $r[i]$ to a large value.

2. Use nested loops to compare each corner i with every other corner j. Compute the squared Euclidean distance (ED) between the two corners:

$$ED = (x_i - x_j)^2 + (y_i - y_j)^2$$

3. If the corner j has a stronger corner response than i (e.g., $R_j > R_i$) and the squared Euclidean distance is smaller than the current radius $r[i]$, update the radius:

$$r[i] = \min(r[i], ED)$$

4. This process ensures that $r[i]$ captures the distance to the nearest stronger corner for each corner i.

Once the suppression radii are computed for all corners, the corners are sorted in descending order based on their radii. Finally, the top N_{best} corners are selected:

$$\text{Top corners} = \text{Sort}(r[i], \text{desc})[: N_{\text{best}}]$$

This approach ensures that the selected corners are uniformly distributed across the image while preserving their relative strength.

Fig. 5. Key Points after ANMS for Image 1 from Set 1

Fig. 6. Key Points after ANMS for Image 2 from Set 1

Patch Extraction and Processing

The feature descriptor extracts a 41×41 pixel patch centered around each corner point. This patch size is selected to capture sufficient local context around the feature point while remaining small enough to maintain local distinctiveness. The implementation processes these patches through the following steps:

1. Gaussian Blur

- A 3×3 Gaussian filter is applied to the patch to reduce noise sensitivity.
- This step enhances robustness to small image variations.

2. Subsampling

- The 41×41 patch is reduced to 8×8 using bilinear interpolation.
- This step decreases computational complexity while retaining essential information.
- The resulting patch is transformed into a compact 64-dimensional feature vector.

3. Standardization

- The feature vector is normalized to have zero mean and unit variance.
- This normalization ensures invariance to illumination changes.
- It also reduces the impact of contrast differences between images.

These processing steps ensure that the feature descriptor is compact, robust, and distinctive, making it suitable for various computer vision tasks.

Feature Matching

Feature matching in our implementation follows a systematic approach to find corresponding points between image pairs. The process begins with the `match_features` function, which takes two sets of feature descriptors (`desc1` and `desc2`) and their corresponding corners, along with a `ratio_threshold` parameter set to 0.8.

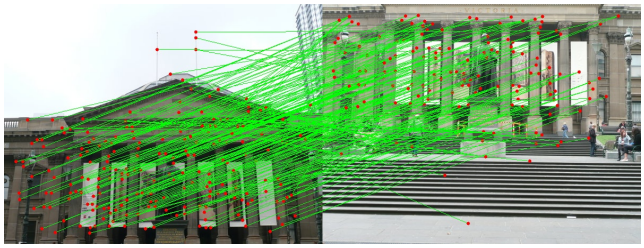

Fig. 7. Output after feature Matching

1. Sum of Squared Differences (SSD)

- For each descriptor in the first image, the function computes the Sum of Squared Differences (SSD) with all descriptors in the second image.
- This effectively measures the similarity between feature vectors.
- The distances are stored and sorted to find the two best matches [**?**].

2. Lowe's Ratio Test

- A match is considered valid only if the ratio of the best match distance to the second-best match distance is less than the `ratio_threshold`.
- This test helps filter out ambiguous matches where a feature point might have multiple similar matches in the second image.
- The `ratio_threshold` of 0.8 provides a good balance between match quantity and quality:
  - A lower value yields fewer but more reliable matches.
  - A higher value gives more matches but potentially includes more false positives [**?**].

3. Storing Matches

- The matches are stored as pairs of indices (i, j), where i represents the index of the feature in the first image and j represents the matching feature's index in the second image.
- These matched pairs are then used as input for the RANSAC algorithm to estimate the homography matrix between the images.

Impact on Panorama Quality

The effectiveness of the feature matching significantly impacts the quality of the final panorama, as poor matches can lead to incorrect homography.

Feature Matching

RANSAC works like a detective, trying to find the best way to transform one image to match another. It follows these steps:

1) Randomly selects four pairs of matched points from the two images.
2) Calculate how these points would transform between images, known as a **homography**.
3) Checks if other matched points agree with this transformation. If a point agrees, it is called an **inlier**.
4) Repeat this process multiple times, keeping track of which transformation results in the highest number of inliers.
5) Finally, we use all the good matches (inliers) to calculate the **final transformation** that is used to stitch the images together.

Our RANSAC implementation uses a threshold of 3.0 pixels for inlier detection and runs for a maximum of 2000 iterations, providing an optimal balance between accurate homography estimation and computational efficiency in the image stitching process.

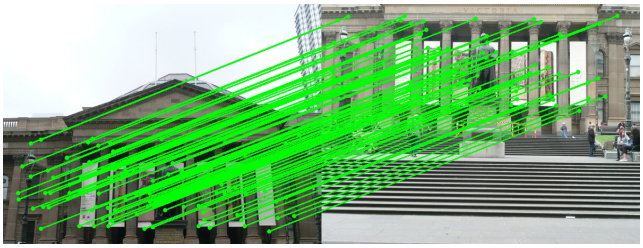

Fig. 8. RANSAC for the Image 1 and 2 form the Set 1

Stitching Implementation

The stitching process begins with the computation of a **homogenetic matrix** using matched feature points and RANSAC. The `stitch_images` function then uses this homography to warp the second image into the coordinate space of the first image. To handle memory constraints, the code processes images in chunks of 1000 pixels height, transforming each chunk using `cv2.warpPerspective` with the computed homography.

Blending Technique

The blending strategy employs a binary mask-based approach, where:

1) Binary masks are created for each warped image chunk.
2) Overlapping regions are identified using mask intersection.
3) Non-overlapping regions are directly copied from their respective images.
4) The overlap areas are blended using a simple weighted average (0.5 weight for each image).

Fig. 9. Stitched Image

OUTPUTS FOR THE GIVEN DATA SETS

Fig. 10. All Images in the given Set 1

Fig. 11. Stitched Image for the Set 1

Fig. 12. All Images in the given Set 2

Fig. 18. Dataset from the Custom Set 2

Fig. 13. Stitched Image 1 and 2 from the Set 2

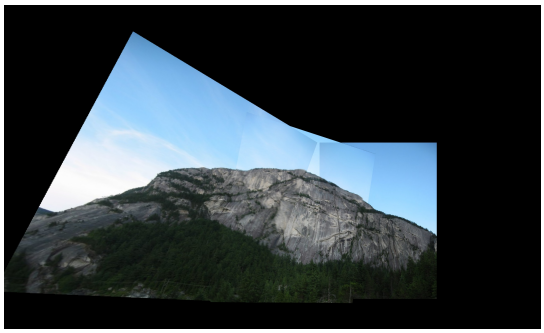

Fig. 14. Stitched Image from the Set 2

Fig. 19. Stitched Image 1 and 2 from the Custom Set 2

Fig. 20. Stitched Image from the Custom Set 2

Fig. 15. Dataset from the Custom Set 1

Fig. 16. Stitching Images from the Custom Set 1

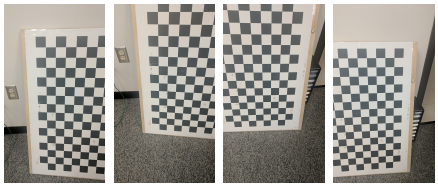

Fig. 21. Dataset from the Test Set 1

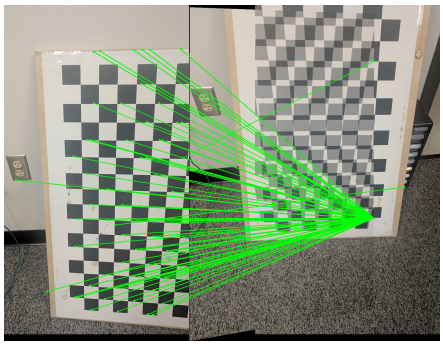

Fig. 22. Failed RANSAC for the Test Set 1

The repetitive black and white pattern of a chessboard can cause confusion in feature matching. Since the squares look alike, the algorithm might incorrectly match squares that are not at the same physical location.

Fig. 17. Final Stitched Image from the Custom Set 1 of 4 Images

Fig. 23. Dataset from the Test Set 3

Fig. 24. Stitched Image 1 and 2 from the Test Set 3

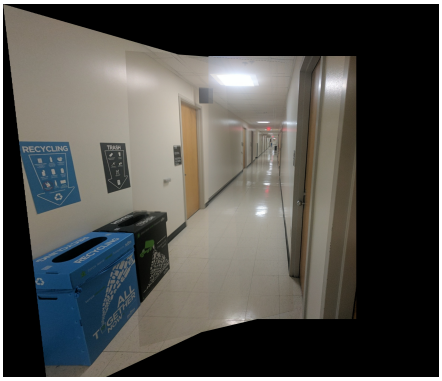

Fig. 25. Stitched Image from the Test Set 3

For indoor scenes with similar colors across images, increasing the number of ANMS keypoints to 1500 improves RANSAC's performance. It provides more unique feature matches, making it easier to compute an accurate homography transformation. This is especially useful when the scene has fewer visual differences or repetitive patterns, as more keypoints help identify small differences in overlapping regions.

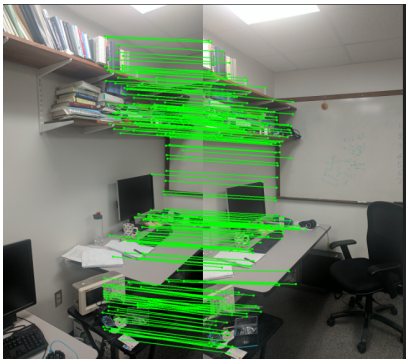

Fig. 26. Test set 4, Stitched Images 1 and 2

Fig. 27. Test set 4,Failed RANSAC because of the different features in the next image in the Test set 4

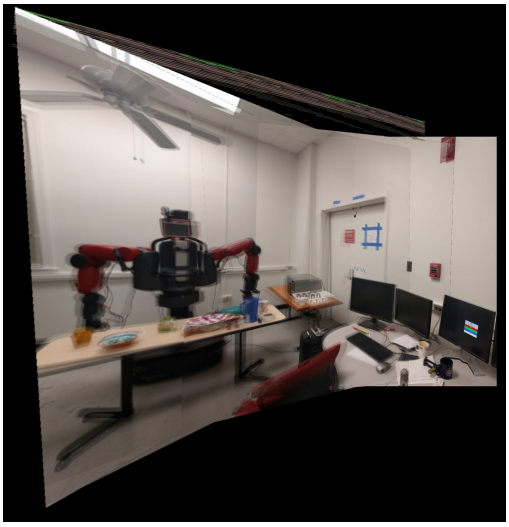

Fig. 28. Right Side stitch for Set 3

Fig. 29. Left Side stitch for Set 3

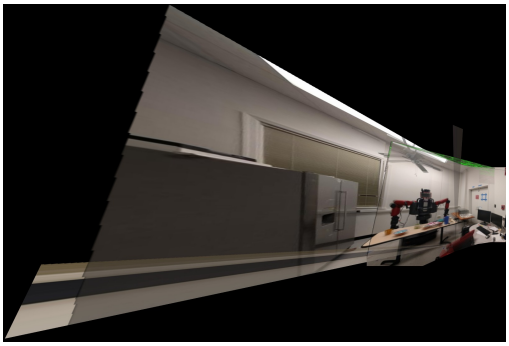

Fig. 30. stitching on both sides from the center Image of the data set

For large panoramic datasets with horizontal image sequences, we tested two stitching approaches.

The first approach used a **middle-out strategy**, where stitching started from the center images and moved outward in both directions at the same time. This created two separate panoramas: one for the left sequence and one for the right. Although both sequences had good alignment and blending, combining these two panoramas was difficult due to accumulated transformation errors and the challenge of matching features across larger images.

The second approach was a **sequential method**, where images were stitched one by one from one end to the other. However, this method led to significant error buildup and geometric distortions in the final panorama.

To improve the stitching process for large image sets, we increased the number of corner and ANMS keypoints and adjusted the RANSAC threshold for each stitch. These adjustments helped enhance feature matching and alignment, particularly for images with fewer distinctive features.

These results show the challenges of stitching large panoramic sequences and the need to carefully choose a strategy and optimize parameters based on the dataset.

II. PHASE 2: Deep Learning Approach

Using One Late Day for Project 1 Phase 2

A. Data Generation

Image Preprocessing

All images were resized to a uniform resolution of 480×480 pixels to ensure consistency.

A region of interest (ROI) was defined as the central area of the image, excluding a 160-pixel margin on all sides to avoid boundary artifacts during patch extraction.

Patch Extraction

A random point within the ROI was selected, and a 128×128 patch (denoted as P_A) was extracted from the original image.

Corner Perturbation

The four corner coordinates of P_A were perturbed by adding random displacements in the range $[-32, 32]$ pixels. This perturbation introduced synthetic geometric variations while ensuring the warped patch remained within valid image boundaries.

Homography Computation

The homography matrix H_{AB} between the original and perturbed corners was computed using `cv2.getPerspectiveTransform`. The inverse homography $H_{BA} = H_{AB}^{-1}$ (calculated via `np.linalg.inv`) was applied to warp the original image using `cv2.warpPerspective`, ensuring valid pixel extraction without black borders.

Warped Patch Extraction

The perturbed patch P_B was extracted from the warped image at the same coordinates as P_A in the original image.

Label Generation

Ground truth labels were derived as the displacement vectors between the original and perturbed corners:

$$H_{4Pt} = C_B - C_A$$

resulting in an 8-dimensional regression target.

Implementation Details

- **Dataset Scale:** For each of the 5,000 training images, 8 patches were generated, yielding 40,000 training samples and for each 1000 validating images, 20 patches were generated making 20,000 validating samples.
- **Storage:** The original patch (P_A), warped patch (P_B), perturbed corners, and original corners were saved. Patches were normalized to $[0, 1]$ and stored as $128 \times 128 \times 2$ depthwise stacks for model input.

Supervised and Unsupervised Learning Approaches for Homography Estimation

Network Design and Objectives

The training process aims to estimate the homography matrix by leveraging both supervised and unsupervised approaches. The core network architecture comprises convolutional neural networks (CNNs) for feature extraction, followed by regression layers predicting the corner displacements between original and perturbed image patches. Two training modes are explored:

- **Supervised Training:** Directly minimizes the loss between predicted and ground-truth 8-dimensional corner displacements.
- **Unsupervised Training:** Relies on photometric loss between the original image and the warped image derived using the estimated homography.

Supervised Model (SupModel)

The supervised model predicts the 8-dimensional corner displacement vector. The architecture of the model consists of:

- Eight convolutional layers with ReLU activation and batch normalization to extract features from the concatenated input patches.
- Max-pooling layers to reduce spatial dimensions while retaining key features.
- A fully connected layer that regresses the 8-dimensional output representing the displacements between corners.
- Dropout layers to prevent overfitting during training.

The input to this model is a $128 \times 128 \times 2$ tensor, which represents the concatenated original and warped patches, and the output is the predicted corner displacements $\hat{H}_{4Pt}$.

Supervised Model Training and Results

The supervised model was trained using the Mean Squared Error (MSE) loss function to minimize the difference between the predicted and ground-truth 8-dimensional corner displacements. The Adam optimizer was employed with a learning rate of 0.0001. The training was conducted over 40 epochs, during which the model progressively learned accurate displacement predictions.

At the end of training, the model achieved a validation loss of 439.40 and a training loss of 16.009. The best-performing model, based on the lowest validation loss, was selected for further testing. The End-Point Error (EPE) was calculated as

```
===== Model Summary =====
================================================================
Layer (type:depth-idx)                   Param #
================================================================
├─Conv2d: 1-1                            1,216
├─Conv2d: 1-2                            36,928
├─BatchNorm2d: 1-3                       128
├─BatchNorm2d: 1-4                       128
├─MaxPool2d: 1-5                         --
├─Conv2d: 1-6                            36,928
├─Conv2d: 1-7                            36,928
├─BatchNorm2d: 1-8                       128
├─BatchNorm2d: 1-9                       128
├─Conv2d: 1-10                           73,856
├─Conv2d: 1-11                           147,584
├─BatchNorm2d: 1-12                      256
├─BatchNorm2d: 1-13                      256
├─Conv2d: 1-14                           147,584
├─Conv2d: 1-15                           147,584
├─BatchNorm2d: 1-16                      256
├─BatchNorm2d: 1-17                      256
├─Dropout: 1-18                          --
├─Linear: 1-19                           33,555,456
├─Linear: 1-20                           8,200
================================================================
Total params: 34,193,800
Trainable params: 34,193,800
Non-trainable params: 0
================================================================
```

Fig. 31. Supervised Model Architecture

the average L2 loss for the train, validation, and test sets, with the following results:

- **Train Set:** Average L2 Loss = 58.1101, EPE = 7.62
- **Validation Set:** Average L2 Loss = 58.0664, EPE = 7.62
- **Test Set:** Average L2 Loss = 58.6462, EPE = 7.66

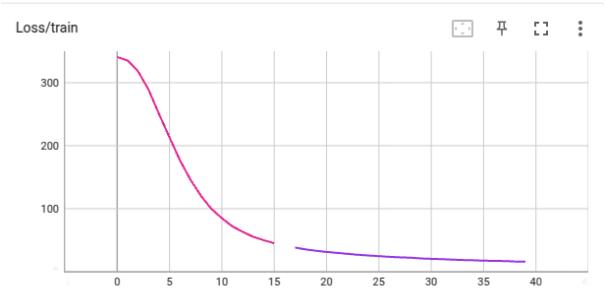

Fig. 32. Supervised Learning: Train Loss vs. Epochs

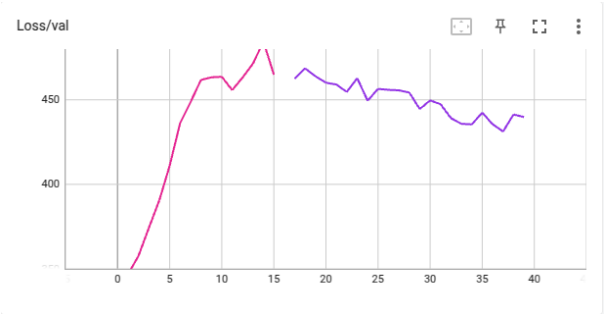

Fig. 33. Supervised Learning: Validation Loss vs. Epochs

Fig. 34. Supervised Learning-Based Homography vs. Ground Truth

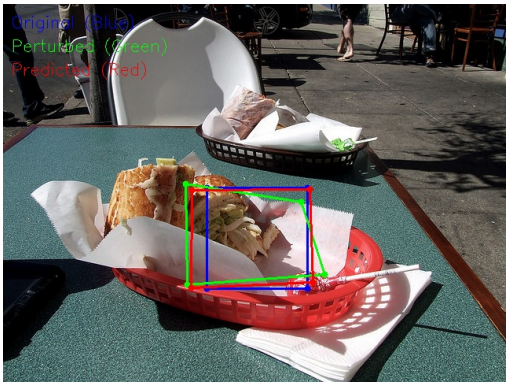

Fig. 35. Supervised Learning-Based Homography vs. Ground Truth

Fig. 36. Supervised Learning-Based Homography vs. Ground Truth

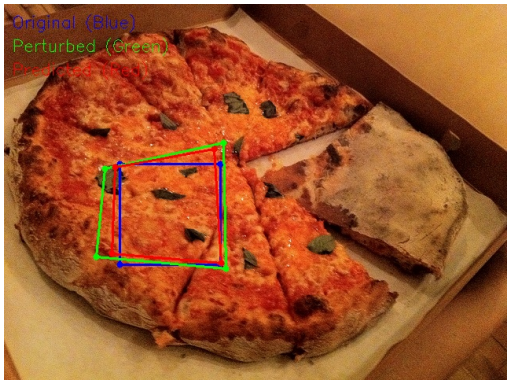

Fig. 37. Supervised Learning-Based Homography vs. Ground Truth

Unsupervised Model (UnSupModel)

The unsupervised model predicts the homography by incorporating:

- **Feature Extraction:** The same CNN architecture as the supervised model is used to predict corner displacements.
- **Tensor Direct Linear Transform (TensorDLT):** Converts corner displacements into a 3×3 homography matrix using a least-squares solution for $Ax = b$.
- **Warping Step:** The original image patch is warped using the estimated homography matrix H via `kornia`'s `warp_perspective` function.
- **Loss Function:** The photometric loss is computed as the pixel-wise L1 distance between the warped image and the target image.

```
===== Model Summary =====
==========================================================
Layer (type:depth-idx)                   Param #
==========================================================
├─SupModel: 1-1                         --
|    └─Conv2d: 2-1                      1,216
|    └─Conv2d: 2-2                      36,928
|    └─BatchNorm2d: 2-3                 128
|    └─BatchNorm2d: 2-4                 128
|    └─MaxPool2d: 2-5                   --
|    └─Conv2d: 2-6                      36,928
|    └─Conv2d: 2-7                      36,928
|    └─BatchNorm2d: 2-8                 128
|    └─BatchNorm2d: 2-9                 128
|    └─Conv2d: 2-10                     73,856
|    └─Conv2d: 2-11                     147,584
|    └─BatchNorm2d: 2-12                256
|    └─BatchNorm2d: 2-13                256
|    └─Conv2d: 2-14                     147,584
|    └─Conv2d: 2-15                     147,584
|    └─BatchNorm2d: 2-16                256
|    └─BatchNorm2d: 2-17                256
|    └─Dropout: 2-18                    --
|    └─Linear: 2-19                     33,555,456
|    └─Linear: 2-20                     8,200
├─TensorDLT: 1-2                        --
==========================================================
Total params: 34,193,800
Trainable params: 34,193,800
Non-trainable params: 0
==========================================================
```

Fig. 38. UnSupervised Model Architecture

A system of equations of the form $Ax = b$ is formulated, where:

$$A \cdot \mathbf{h} = \mathbf{b}$$

The matrix A is constructed using the four original corner coordinates (x_i, y_i) and the predicted perturbed corner coordinates (u_i, v_i). Each pair of corresponding points generates two linear equations:

This construction is repeated for all four corners, resulting in an 8×8 matrix A.

Least-Squares Solution for the Homography Matrix: To solve for the homography, TensorDLT employs the least-squares solution by computing the pseudoinverse of A using:

$$\mathbf{h} = A^{-1} \cdot \mathbf{b}$$

This is implemented in PyTorch using `torch.linalg.pinv()` for computing the pseudoinverse of A and obtaining the solution for H. The resulting homography matrix H is augmented to include $[0, 0, 1]$ in the bottom row, making it a 3×3 transformation matrix.

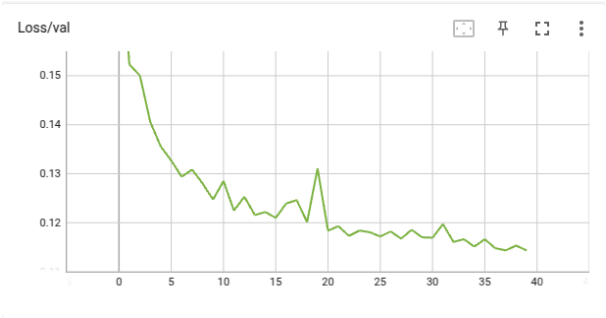

Fig. 39. Unsupervised Learning: Validation Loss vs. Epochs

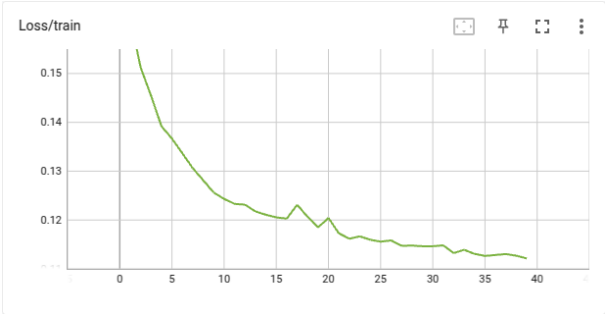

Fig. 40. Unsupervised Learning: Train Loss vs. Epochs

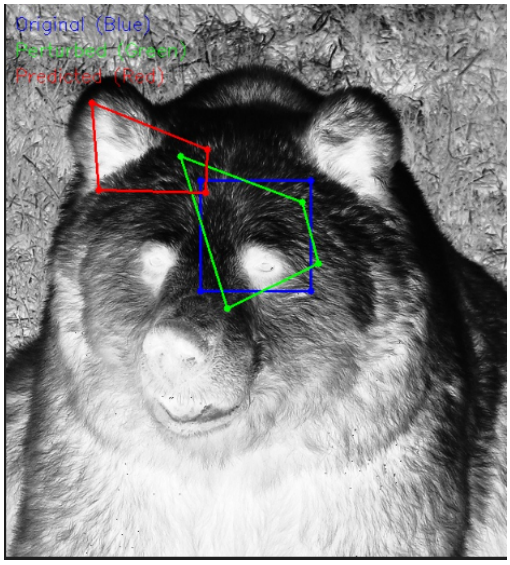

Fig. 41. Unsupervised Learning-Based Homography vs. Ground Truth

Fig. 41. highlights the performance of the unsupervised learning-based approach, which did not yield the expected results in our case. we tried to approach the problem using template matching, but the results were not satisfactory.

B. Panorama Stitching

The initial goal of Wrapper.py was to stitch multiple images into a seamless panorama. The script was designed to load a sequence of images, predict homographies between consecutive frames, and warp them to a common reference frame using cv2.warpPerspective. However, errors and challenges arose during execution, preventing the successful completion of the task as originally intended.

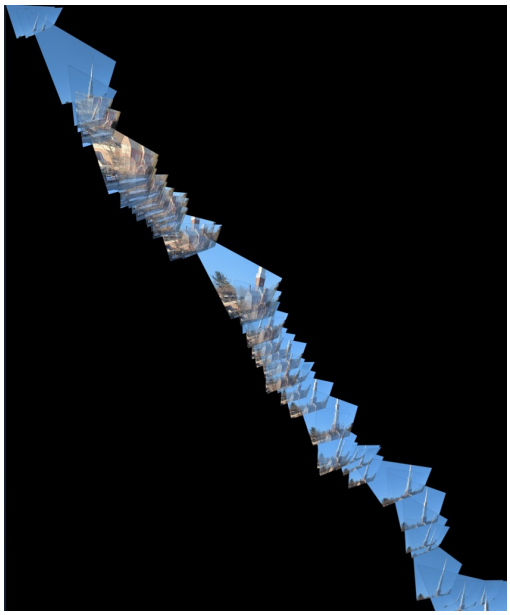

Fig. 42. Unsupervised Tower

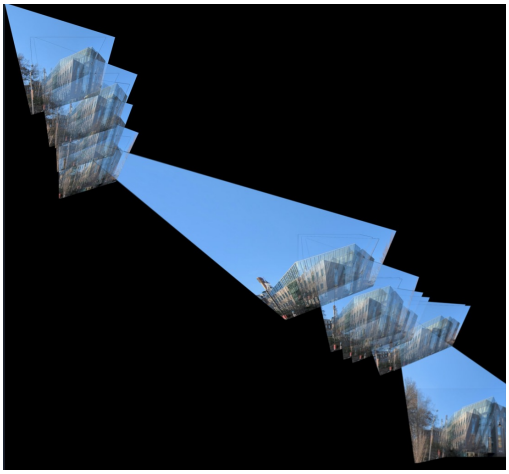

Fig. 43. Unsupervised UH