My AutoPano!

Sarthak Mehta
Department of Robotics Engineering
Worcester Polytechnic Institute
smehta1@wpi.edu

Prasham Soni
Department of Robotics Engineering
Worcester Polytechnic Institute
psoni@wpi.edu

***Abstract*—This report presents the findings and methodologies of the My AutoPano project, which aims to seamlessly stitch two or more overlapping images into a single panoramic image while ensuring smooth transitions and alignment. For Phase1, classical methods were employed to achieve this objective, leveraging traditional feature detection and transformation techniques. For phase 2, deep learning approaches have been introduced to enhance the accuracy and robustness of the stitching process.**

I. Phase1

Each image is assumed to have an overlapping area of approximately 30% to 50%. The following steps were undertaken to accomplish this:

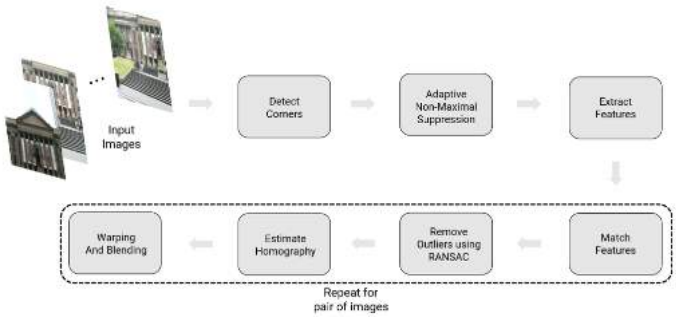

Fig. 1. Overview of Panorama Stitching using the Conventional Approach

As depicted in Figure 1, the key stages of the process are as follows:

1) Detect corners in each of the images.
2) Apply Adaptive Non-Maximal Suppression (ANMS) to remove redundant corners.
3) Create a feature descriptor vector for each detected corner.
4) Match features between overlapping regions using the feature descriptor vectors.
5) Use Random Sample Consensus (RANSAC) to eliminate outliers.
6) Compute the homography between pairs of images.
7) Blend the images to generate a seamlessly stitched panorama.

A. Dataset of Images

Figures 2 to 5 represent the dataset of images used to training and testing of the algorithm. The process begins by

Fig. 2. Set 1

Fig. 3. Set 2

Fig. 4. Custom Set 1

Fig. 5. Test Set 1

performing steps 1 to 7 on a pair of images. Once these steps are completed, the resulting stitched image is used as input for the subsequent iterations, with the process repeated for the remaining images in the dataset.

B. Corner Detection

Corners are defined as points where the gradient of intensity changes sharply in two directions. Corner detection is performed to identify overlapping regions or points in images. While edge detection could also be used for this purpose, edges only capture intensity changes in one direction, such as object boundaries or structural outlines. Unlike corners, edges lack a well-defined location in two dimensions, making them ambiguous for matching. For example, matching along an edge may result in multiple potential matches along its length (e.g., lines or curves), leading to errors in image alignment.

To detect corners, we used the `cv2.goodFeaturesToTrack` function along with the Corner Harris method. The `cv2.goodFeaturesToTrack` function identifies the corners, while the Corner Harris method calculates the corner score for each detected corner. The corner score is a numerical value that represents the strength of a corner-like feature. The Corner Harris corner score is computed using the equation:

$$R = \det(M) - k \cdot (\text{trace}(M))^2$$

Here:

- M and trace(M) are calculated based on the input image's intensity change.
- k: A user-defined parameter controlling the sensitivity of the detector to corner-like regions. In our implementation, we used $k = 0.04$. Higher values emphasize stronger corners but may also include edges.

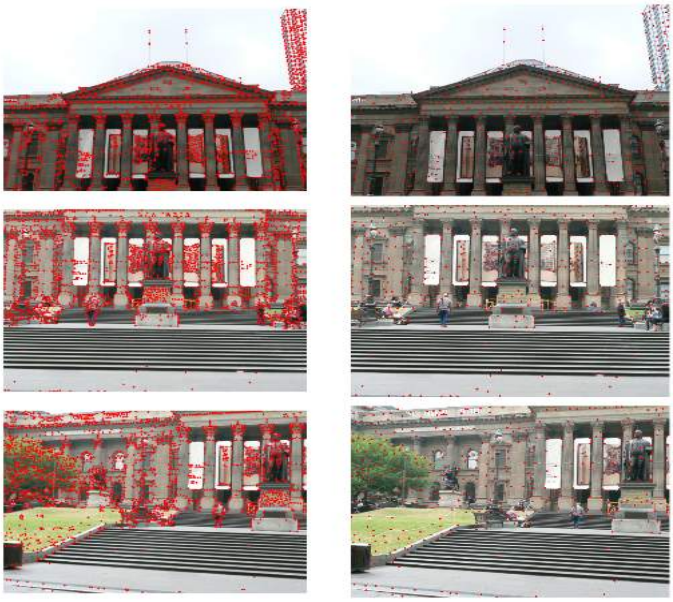

Fig. 6. Set 1: Corner Detection and Corners after ANMS

The output for Corner Detection and Corners after ANMS is shown above in Figure 6.

C. ANMS (Adaptive Non-Maximal Suppression)

Corner detection algorithms, such as Corner Harris, often detect multiple nearby pixels with strong corner responses around the same physical feature (e.g., the edge of a window or a corner of a building). These redundant corners can clutter the feature set and degrade the performance of matching algorithms. To address this issue, we employ **Adaptive Non-Maximal Suppression (ANMS)**, which reduces redundancy by retaining only the most representative corners based on their response strength and spatial distribution.

Algorithm 1 Pseudo-code for ANMS

Require: corners, scores, N
Ensure: best_corners
  n = length(corners)
  r = array of size n filled with ∞
  for i = 0 to n-1 **do**
    for j = 0 to n-1 **do**
      if $S[j] > S[i]$ **then**
        $d \leftarrow \|c_i - c_j\|^2$
        if d $< r[i]$ **then**
          r[i] = d
        end if
      end if
    end for
  end for
  indices = sort r in descending order and get corresponding indices
  best_indices = first N indices
  best_corners = corners[best_indices]
  return best_corners =0

D. Feature Descriptor

After the previous step, we have a set of N_{best} points and their corresponding coordinates. These points include some overlapping points in the two sets of images, but each overlapping point will have different coordinates in each image. To identify these overlapping points, we create a feature vector for each point.

To achieve this, we extract a 41×41 patch from the image, with the point at the center as the keypoint. A Gaussian blur is applied to this patch to smooth intensity variations. Next, the blurred patch is resized to 8×8 using sub-sampling. For resizing, we utilize `LINEAR` interpolation, which computes the average intensity of several pixels to determine the new pixel value.

The 8×8 patch is then flattened into a 64×1 feature vector. Finally, the feature vector is standardized to have a mean of zero and a variance of one.

E. Feature Matching

In this section, we utilize the feature vectors extracted in the previous step. The overlapping regions in both images contain

similar points, and each of these points has an almost identical feature vector. Using feature matching, we identify and match points between the two images.

Fig. 7. Set 1: Feature Matching

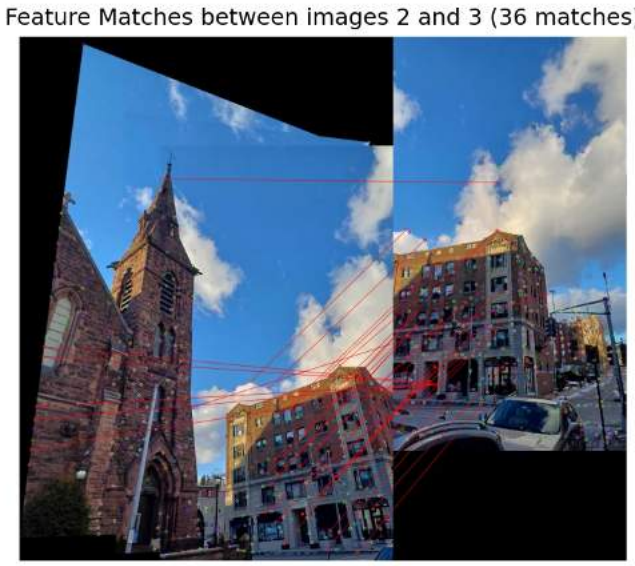

Fig. 8. Custom Set 1: Feature Matching

The corresponding matching features and points in the two image are shown in Figure 7 and Figure 8. Each line matches the corresponding output of the matched features in the overlapping image's region.

To achieve this, we compute the Sum of Squared Distances (SSD) between the feature vectors of both images. For each point, we take the two points with the least SSD values and compute the ratio of their distances. If this ratio is below a predefined threshold (in this case, 0.8), we consider the feature match valid. This process is repeated for all feature vectors and their corresponding coordinates.

F. RANSAC (Random Sample Consensus)

In feature matching, we identify similar points in the two images. However, some corners or points might not truly correspond to the same physical feature in the images but could still have similar feature vectors. These points, known as outliers, can cause misalignment during image stitching. Such outliers are highlighted in Figure 9 and Figure 10. We

Algorithm 2 Feature Matching Pseudo-code

Require:
F_1, F_2 (feature vectors of two images)
C_1, C_2 (coordinates of features)
τ (ratio threshold)
Ensure: matches M, matched coordinates MC_1, MC_2
$M, MC_1, MC_2 \leftarrow \emptyset$
for each feature $f_1^i \in F_1$ with index i **do**
$D \leftarrow \sum(f_1^i - F_2)^2$ {Compute distances to all features in F_2}
$idx_{best} \leftarrow (D)$ {Index of closest match}
$idx_{second} \leftarrow (D \setminus \{D[idx_{best}]\})$ {Index of second closest}
$ratio \leftarrow \frac{D[idx_{best}]}{D[idx_{second}]}$
if $ratio < \tau$ **then**
Append (i, idx_{best}) to M
Append $C_1[i]$ to MC_1
Append $C_2[idx_{best}]$ to MC_2
end if
end for
return M, MC_1, MC_2 =0

Fig. 9. Set 1: Feature Match with Outliers (Before RANSAC)

Fig. 10. CustomSet 1: Feature Match with Outliers (Before RANSAC)

use **RANSAC (Random Sample Consensus)** to remove these outliers and estimate the homography.

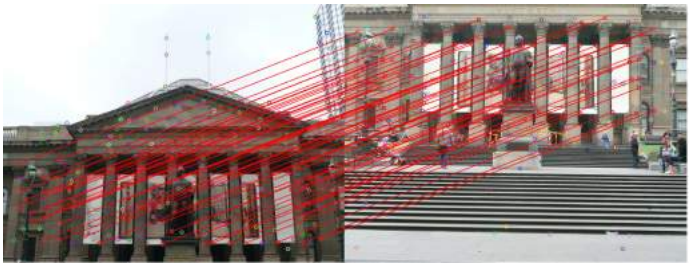

Fig. 11. Set 1: Feature Match after RANSAC

Fig. 12. CustomSet 1: Feature Match after RANSAC

During the RANSAC process, we simultaneously estimate the homography matrix. The homography matrix maps points from one plane in an image to another plane, allowing for alignment of the images.

G. Blending and Stitching of images

In panorama stitching, multiple images are aligned and blended to create a seamless composite image that spans over a larger field of view, the process involves the following steps:

- Image Alignment using homography transformations
- Canvas Creation to accommodate all images in the set given
- Image Warping to map images onto the same or similar coordinate system
- Blending: to eliminate visible seams and ensure a smooth transition between overlapping regions

First, the **homography matrix** is computed to geometrically transform one image onto the coordinate space of another. The corner points of the images are used to calculate the bounding box, ensuring that the new canvas accommodates both images without clipping. A **translation matrix** is applied to shift all coordinates to positive space, and the first image is warped onto the new canvas using a combined transformation. The second image is then overlaid onto the canvas, blending overlapping regions to eliminate visible seams. This process is iteratively repeated for all input images, resulting in a smooth and visually coherent panorama that combines all input images into a single cohesive output.

The results of the algorithm are depicted in Figures 13 to 16.

Algorithm 3 RANSAC for Homography Estimation

Require:
P_1, P_2 (matched points from two images)
N (number of iterations)
τ (threshold for inlier distance)
Ensure: homography H, inliers I
$best\_count \leftarrow 0$
$H_{best}, I_{best} \leftarrow \emptyset$
for $i = 1$ to N **do**
  Select 4 random matches from P_1, P_2
  Compute homography H_i from selected matches
  $count \leftarrow$ number of points with error $< \tau$
  if $count > best\_count$ **then**
    $best\_count \leftarrow count$
    $H_{best} \leftarrow H_i$
    $I_{best} \leftarrow$ points with error $< \tau$
  end if
end for
if $|I_{best}| > 4$ **then**
  Recompute H_{best} using all points in I_{best}
end if
return H_{best}, I_{best} =0

Fig. 13. Panorama Generated for Set 1 images

Algorithm 4 Image Warping and Blending Pseudo-code

Require:
$img1, img2$ (input images)
h (homography matrix)
Ensure: warped and blended result image
$h1, w1 \leftarrow dimensions(img2)$
$h2, w2 \leftarrow dimensions(img1)$
// Define corner points for both images
$pts1 \leftarrow [[0,0],[0,h1],[w1,h1],[w1,0]]$ {Corners of img2}
$pts2 \leftarrow [[0,0],[0,h2],[w2,h2],[w2,0]]$ {Corners of img1}
// Transform corners of img1 using homography
$pts2_{warped} \leftarrow perspectiveTransform(pts2, h)$
// Combine all corner points
$pts \leftarrow concatenate(pts1, pts2_{warped})$
// Find bounding box
$xmin, ymin \leftarrow min(pts)$
$xmax, ymax \leftarrow max(pts)$
// Calculate canvas dimensions
$width \leftarrow xmax - xmin$
$height \leftarrow ymax - ymin$
if $width \leq 0$ OR $height \leq 0$ **then**
return $img1$ {Return original if invalid dimensions}
end if
// Create translation matrix
$t \leftarrow [-xmin, -ymin]$
$H_t \leftarrow \begin{bmatrix} 1 & 0 & t[0] \\ 0 & 1 & t[1] \\ 0 & 0 & 1 \end{bmatrix}$
// Warp img1 onto new canvas
$result \leftarrow warpPerspective(img1, H_t \cdot h, (width, height))$
// Place img2 in result with offset
$y_{start} \leftarrow t[1]$
$y_{end} \leftarrow t[1] + h1$
$x_{start} \leftarrow t[0]$
$x_{end} \leftarrow t[0] + w1$
if $y_{end} > height$ OR $x_{end} > width$ **then**
print("Warning: img2 exceeds canvas bounds")
end if
$result[y_{start} : y_{end}, x_{start} : x_{end}] \leftarrow img2$
return $result$ =0

H. Cylindrical Warping and Blending

We then tried implementing the cylindrical blending algorithm, which employs a sophisticated two-stage process of cylindrical warping and image blending. The algorithm first projects the planar images onto a cylindrical surface using intrinsic camera parameters and focal length calculations, followed by a precise alignment process utilizing homography transformations. This approach was specifically chosen to mitigate perspective distortion and maintain consistent scaling across the panoramic view. Through cylindrical warping, each image undergoes coordinate transformation and trigonometric calculations, while the subsequent blending phase ensures seamless integration of overlapping regions. In our implementation, we successfully processed a sequence of five consecutive images, achieving proper alignment and blending up to this point. However, the algorithm encountered limitations beyond the fifth image, where the cumulative transformation errors and increasing complexity of the panoramic canvas prevented further expansion. This implementation outcome suggests that while the algorithm effectively handles moderate-sized panoramas, additional optimization may be required for processing extended sequences of images. The output for the Cylindrical warping and blending are presented in the Figures:

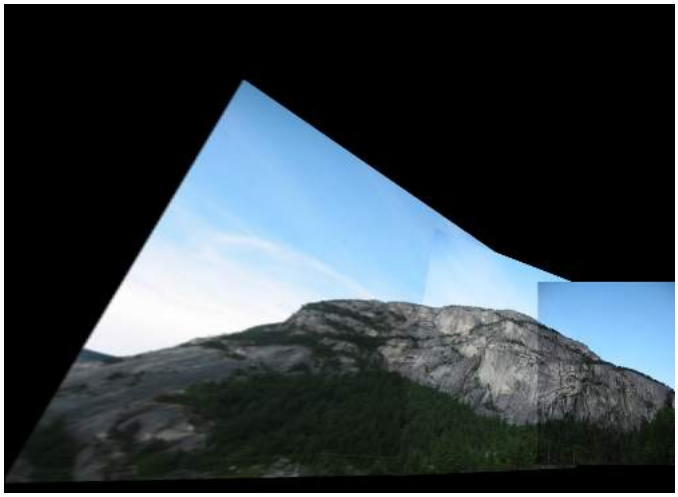

Fig. 14. Panorama Generated for Set 2 images

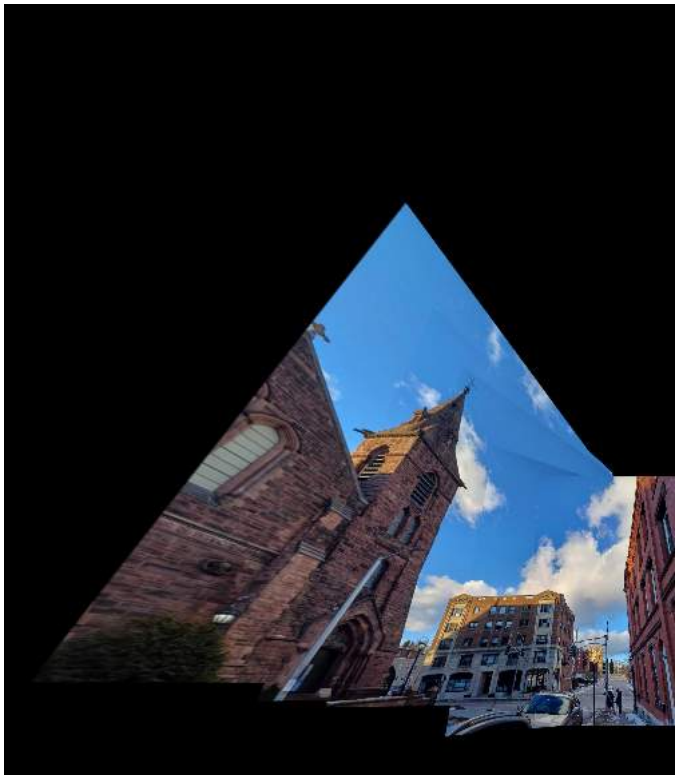

Fig. 15. Panorama Generated for Custom Set 1 images

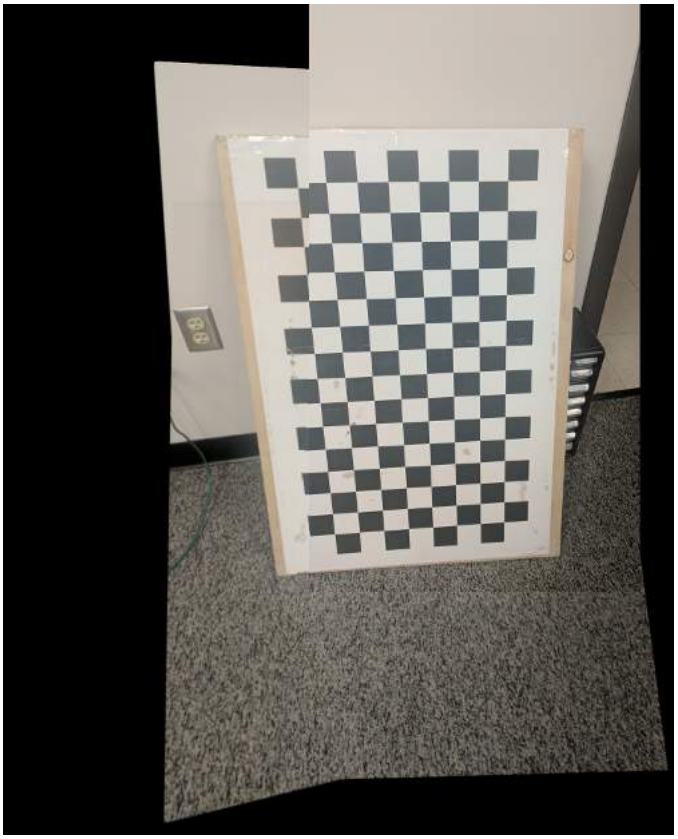

Fig. 16. Panorama Generated for test Set 1 images

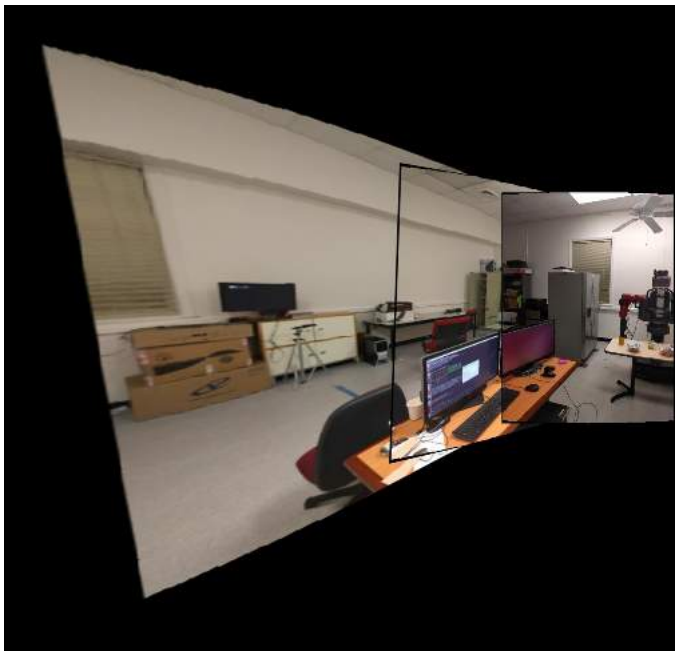

Fig. 17. Panorama Generated by Cylindrical Blending and warping

Fig. 18. Test Set 4 :No common region between the 3rd image and the 4th image

Fig. 19. Test Set 4: No feature matched with the 4th image in the test set

I. Analysis:

While testing our algorithm on **Test Set 4**, depicted in Figure 18, we observed that the first three images blended seamlessly due to adequate overlap and the presence of common homography points. However, when attempting to match features between the blended panorama of images 1, 2, and 3 with the fourth image, the process failed due to the absence of common matching feature points. This was because the fourth image was entirely different from the preceding ones, lacking any overlapping regions. This failure highlights the critical importance of ensuring sufficient overlap between consecutive images for successful panorama stitching.

Secondly, we observed that while working with image sets, such as the one depicted in Figure 20 containing more than five images, the algorithm occasionally fails due to inaccuracies in the computed homography. As the number of images increases, the cumulative effect of minor errors in feature detection and matching can compound, resulting in incorrect transformations. This issue becomes particularly problematic when the later images in the sequence have insufficient overlap with the blended panorama created from previous images.

Without enough common feature points between the current image and the accumulated panorama, the estimated homography matrix becomes unreliable, resulting in poor alignment

Fig. 20. Test Set 2 with 5+ images

or failure to align the images altogether. In some cases, the algorithm attempts to warp the image based on this inaccurate homography, which can cause severe distortions, misalignment, or large bounding boxes that exceed computational or memory constraints, leading to errors or crashes.

This highlights the importance of maintaining sufficient overlap between images and ensuring robust feature matching to produce accurate homographies, especially when dealing with larger image datasets. The Failed outputs for such case are depicted in the Figure 21.

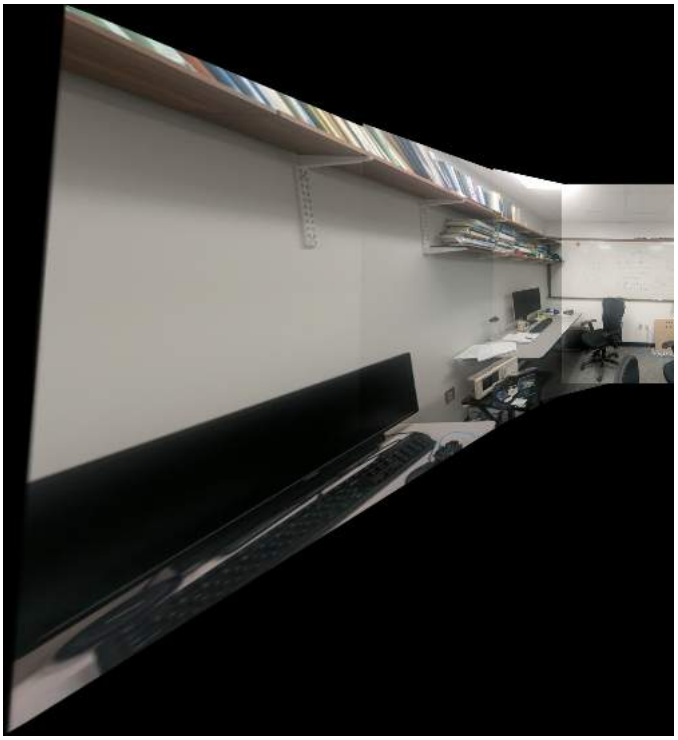

Fig. 21. Test Set 2, Failed Case 1

As we can see in the Figure 22 the algorithm blends the first 4 images of test set 2. As observed in images with more than five inputs, the increasing scale of the panorama leads to larger canvas sizes, making the homography computation unstable. This is evident in the skewed and stretched matches in this highlights the limitations of feature-based methods in high-scale scenarios.

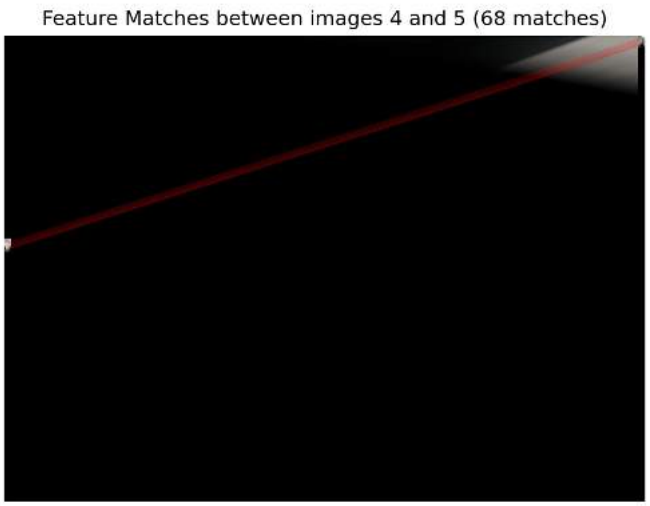

Fig. 22. Test Set 2, Failed Case 2

I. PHASE2

TThe implementation of a deep learning-based approach to generate a seamless panoramic image has been explored using two distinct methodologies:

1) Supervised Approach
2) Unsupervised Approach

In both cases, deep learning is employed to estimate the homography between pairs of images, enabling accurate alignment and stitching.

To train a Convolutional Neural Network (CNN) for estimating the homography between a pair of images, we require ground truth homography data. However, obtaining such data is challenging. To address this issue, we generate a synthetic dataset for training. Specifically, we utilize the MOSOCO Dataset to create image pairs with known homographies, ensuring a controlled and effective learning process.

Fig. 1. IA, PA, PB

Fig. 2. IA, PA, PB

Now, to generate this dataset from an image, we first extract a **random patch** of **size 128 × 128**. Let's denote this patch as P_A. After extraction, we apply a random perturbation to its corner points within the range of $[-\rho, \rho]$. For our dataset, we set the maximum perturbation as $\boldsymbol{\rho = 32}$.

In the next step, we apply a random perturbation to these corner points. Additionally, we introduce a translation component to enhance robustness. A sample image is shown in Fig. 1 and Fig. 2.

P_A: Random taken patch from the original Image I_A.
P_B: P_A + ρ

A. Supervised Approach

While generating the dataset for this approach, we also compute $H4_{Pt}$, which represents the difference between the corner points of P_A and P_B. This serves as the ground truth transformation for training the homography estimation model.

$$H4_Pt = C_B - C_A$$

C_A: Corner coordinates of P_A
C_B: Corner coordinates of P_B

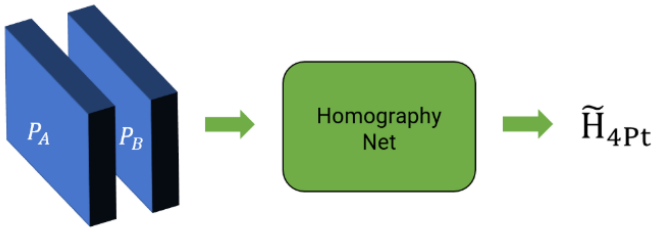

Fig. 3. Overview of Supervised Learning

In this case, P_A and P_B serve as the inputs to the Homography Network, while $H4_{Pt}$ is the predicted output for the supervised model. Hence, it is used as the ground truth, as shown in Figure 3. The predicted homography transformation, denoted as $\tilde{H}_{4pt}$, is an 8×1 vector representing the perturbations of the four corner points from the original patch.

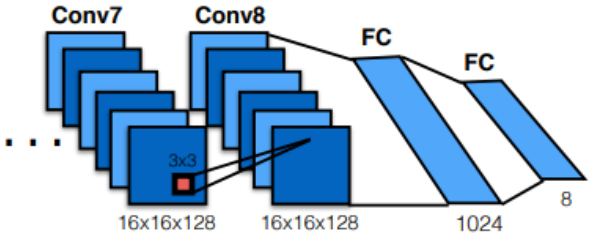

Fig. 4. Architecture for Supervised Homography Net

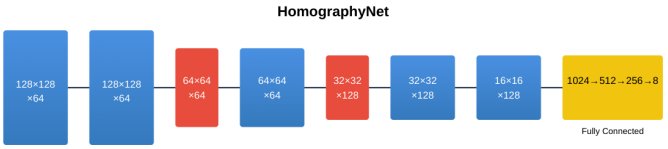

Fig. 5. Pipeline for Supervised Homography Net

1) Architecture:

2) Training Parameters: For this model, we utilized the AdamW optimizer with a learning rate of 0.0001, a mini-batch size of 45, and trained for a total of 75 epochs.

Supervised Model		
	Train	**Val**
EPE	0.2196	0.6351
Run time (ms)	22.84	29.75
Run time (ms) (With CUDA)	1.89	3.02

TABLE I
SUPERVISED MODEL PERFORMANCE

3) Loss Function and Loss Curves: The loss function computes the mean Euclidean norm (L2 norm) of the difference between the predicted and ground truth values. Mathematically, it can be written as $L2_{\text{loss}} = \left\| H4_{\text{pt}} - \tilde{H}4_{\text{pt}} \right\|^2$.

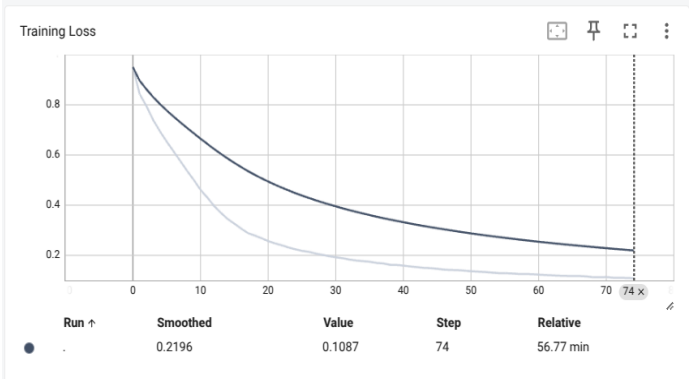

Fig. 6. Supervised: Training Loss v/s Epoch

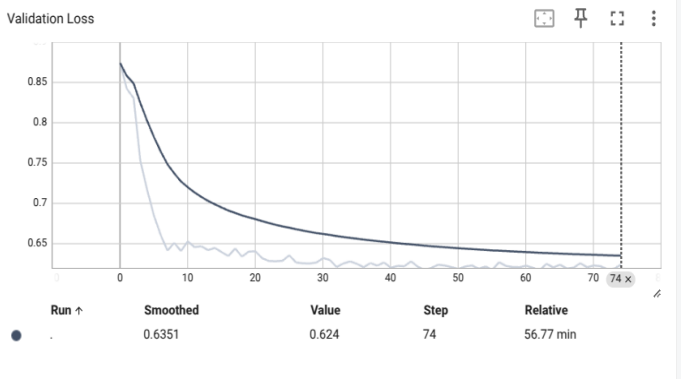

Fig. 7. Supervised: Validation Loss v/s Epoch

4) Output Images:

a) Predicted Homography Patches: This section presents the predicted homography patches.

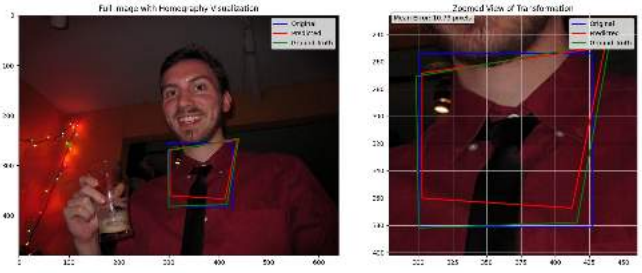

Fig. 8. Output for Supervised

B. Un-Supervised Approach

For the unsupervised model, we also include C_A as an additional input. In the supervised approach, the model predicts $H4_{Pt}$, whereas in the unsupervised approach, the model directly predicts P_B.

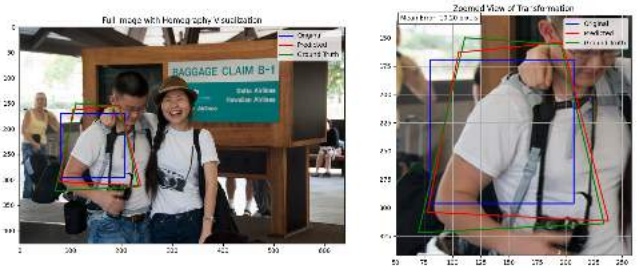

Fig. 9. Output for Supervised

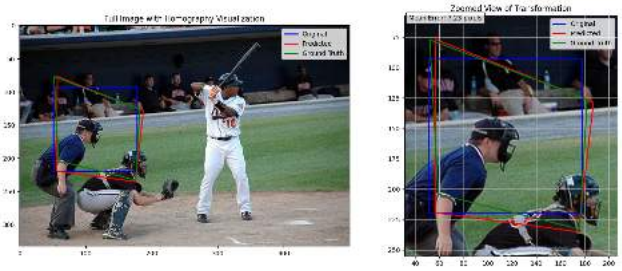

Fig. 10. Output for Supervised

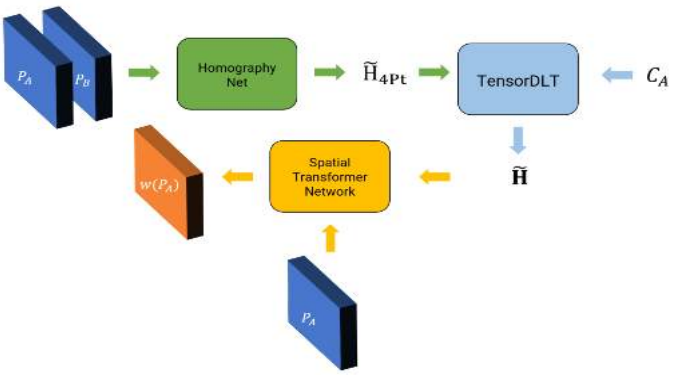

Fig. 11. Overview of Un-Supervised Learning

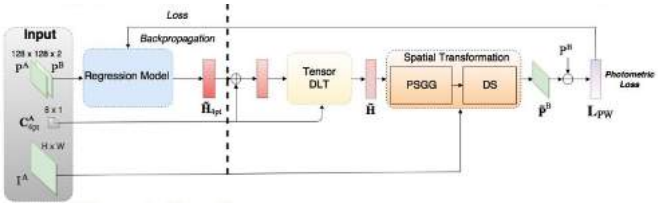

Fig. 12. Architecture of Un-Supervised Learning

In the initial stages, the output remains the same as in supervised learning until the computation of $H4_{\text{Pt}}$. After this step, the output is passed into the **Tensor DLT** layer for further transformation.

1) Tensor DLT (Direct Linear Transform): The **Tensor DLT** module takes the estimated displacement vector $H4_{\text{Pt}}$ and converts it into a 3×3 homography matrix. This step is crucial as it enables the propagation of gradients, which are essential for optimizing the model during backpropagation.

2) Spatial Transformation Network (STN): The **Spatial Transformation Network (STN)** receives the computed 3×3 homography matrix along with the original image I_A. Instead of employing **forward warping**, where the transformation follows:

$$P_A = HP_B$$

we adopt an **inverse warping** approach, where the transformation is computed as:

$$P_B = H_{\text{inv}} P_A$$

This inverse transformation allows for better alignment and interpolation, leading to the predicted warped image P_B.

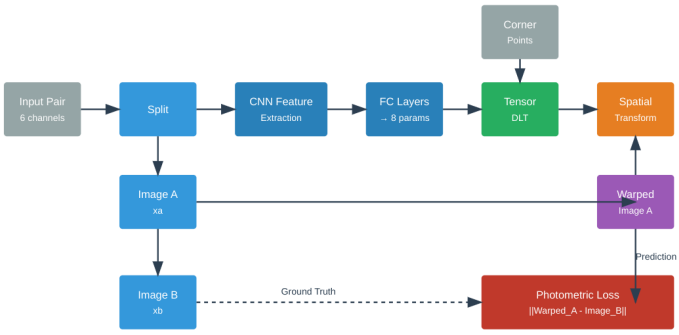

Fig. 13. Flowchart of Un-Supervised Learning

3) Training Parameters: For this model, we utilized the AdamW optimizer with a learning rate of 0.0001, a mini-batch size of 45, and trained for a total of 100 epochs.

Un-Supervised Model		
	Train	**Val**
Photometric Loss	0.4728	0.4709
Run time (ms)	34.26	44.625
Run time (ms) (With CUDA)	2.835	4.53

TABLE II
SUPERVISED MODEL PERFORMANCE

4) Loss Function and Loss Curves: The photometric loss in the unsupervised setting is computed based on the pixel-wise difference between the predicted warped image $\hat{I}_B$ and the ground truth warped image I_B. It can be expressed as:

$$L_{\text{photo}} = \frac{1}{N} \sum_{i=1}^{N} \left\| I_B^i - \hat{I}_B^i \right\|^2$$

where:

- I_B^i is the ground truth warped image.
- $\hat{I}_B^i$ is the predicted warped image.
- N is the total number of pixels.

Alternatively, if **SSIM loss** is used:

$$L_{\text{photo}} = 1 - \text{SSIM}(I_B, \hat{I}_B)$$

where **SSIM** represents the Structural Similarity Index, which measures the perceptual difference between the predicted and actual warped images.

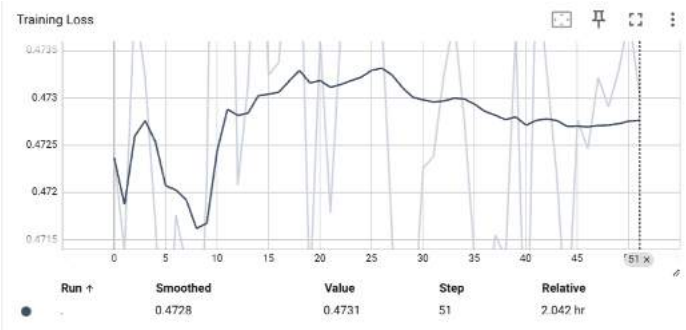

Fig. 14. Un-Supervised: Training Loss v/s Epochs

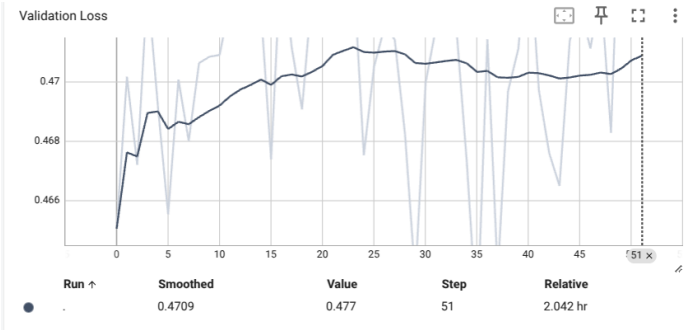

Fig. 15. Un-Supervised: Validation Loss v/s Epochs

As observed in the training and validation loss curves, the loss is not decreasing as expected. For the training loss, it exhibits oscillations, indicating instability during optimization. In the case of validation loss, it is continuously increasing, suggesting that the model is not generalizing well to unseen data.

This behavior implies that the gradient might not be computed correctly, leading to an improper optimization process. One possible reason for this issue is improper normalization, which results in scaling inconsistencies. Specifically, in this case, the normalization of C_A may not be handled correctly, leading to discrepancies in the learning process.

Some of the outputs from the unsupervised learning approach are shown below:

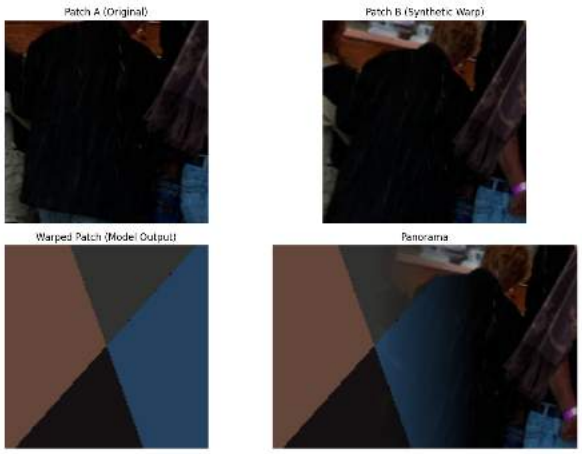

Fig. 16. Un-Supervised: Panorama

II. Panorama Generation Using Deep Learning

A. Supervised Approach

B. Approach 1: Patch-Based Local Homography Stitching

To address the limitations of predicting only local homographies, we explored an alternative method for constructing the final panorama. Instead of relying on a single patch-based transformation, we sampled multiple overlapping patches from the images, predicted their corresponding H4Pt displacements using our trained model, and then aggregated these predictions. We used the Direct Linear Transformation (DLT) algorithm to compute individual homographies from each patch prediction. Finally, we attempted to merge these local transformations into a refined global homography that could be applied consistently across the entire image. This method improved alignment compared to a single local estimate, but it introduced new challenges in blending and merging the transformations, as inconsistent patch predictions could lead to misalignments in different regions of the panorama.

1) Challenges and Limitations: While this approach enabled localized alignment, it introduced several challenges:

- **Patchwise Inconsistencies:** Since the homography predictions were made per patch, inconsistencies arose between neighboring patches, leading to visible seams and discontinuities in the final stitched panorama.
- **Lack of a Global Homography:** Unlike traditional feature-matching-based methods that estimate a single homography for the entire image, our patch-based approach lacked a unified transformation. This resulted in a panorama where different regions were misaligned relative to each other.
- **Warping Artifacts:** Due to independent patch transformations, some areas of the panorama appeared distorted, resembling a grid of warping effects rather than a continuous and smooth stitched image.

2) Output and Observations: The output of this approach resulted in a panorama where the patches were individually warped and blended together. Instead of forming a seamless transition between images, the final result exhibited noticeable square artifacts, as each patch had a slightly different transformation. This was due to the absence of a globally consistent transformation, causing abrupt transitions at patch boundaries and affecting the overall visual coherence of the panorama.

In our supervised learning approach, we trained our model using pairs of image patches: *patch A* and *patch B*, each of size 128×128. The model was trained to predict the ground truth displacement of the four corners, known as *H4Pt*, which defines a local transformation between these patches. This means that, instead of estimating a *global homography*—which would attempt to align the entire image—we are computing multiple *local homographies* for different image regions. While this approach allows for better handling of local distortions and non-planar scenes, it does not provide a unified transformation for the entire panorama.

The output of this approach is depicted in Figure 17, where we demonstrate the stitched panorama using the predicted local homographies.

Fig. 17. Approach 1:Panorama stitching result using the local homography-based approach

C. Approach 2: Resizing Images to Model Input Size

1) Resizing-Based Panorama Stitching: Given that our deep learning model was trained on 128×128 patches, another approach we considered was resizing the entire input images to this fixed size before passing them through the model. Instead of extracting local patches, we directly resized the two original images to 128×128, treating them as inputs similar to our training patches. The model was then used to estimate a homography between the two resized images, rather than between smaller extracted patches.

2) Challenges and Limitations: While this approach simplified the pipeline by removing the need for patch extraction and aggregation, it introduced several limitations:

- **Loss of Detail:** Resizing images to 128×128 significantly reduced their resolution, leading to a loss of fine details that could have been useful for accurate alignment.
- **Scaling Issues:** The model was trained to handle small displacements within patches, but when applied to resized images, the predicted homography often failed to capture the full-scale transformation needed to align the original high-resolution images.
- **Mismatched Transformations:** Since the resizing operation altered the aspect ratio and spatial relationships between keypoints in the images, the predicted transformation did not always align the images correctly when applied to their original scale.

3) Output and Observations: The result of this approach led to an output where the two resized images were successfully overlaid but did not exhibit true panorama stitching. Instead of a seamless transformation aligning the images correctly, the images appeared as overlapping layers without proper geometric alignment, primarily due to the reduced resolution and inaccurate homography estimation at the resized scale.

The output of this approach is depicted in Figure 18, where the overlapping effect of the resized images is visible.

Fig. 18. Approach 2: Resizing Images to Model Input Size

D. Other Panaroma generation outputs

Fig. 19. Supervised: Pano2

Fig. 20. Classical output for test images

Fig. 21. Supervised output for test images

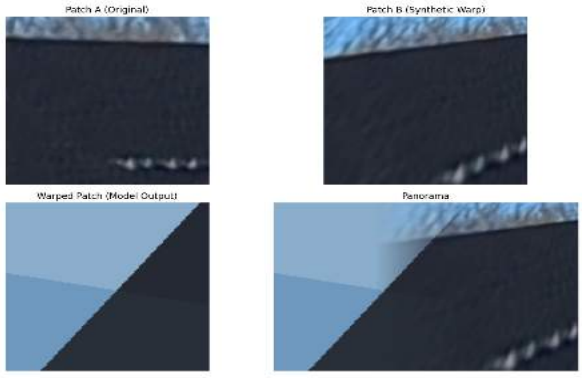

Fig. 22. Unsupervised output for test images