

IMAGE WARPING and MOSAICING

Part A: Image Warping and Mosaicing

Part A.1: Shoot the Pictures

Description: Take two or more pictures with a projective transformation (perspective transformation) between them. The most common approach is to fix the center of projection (COP) and rotate the camera to take the photos.

My approach: I took multiple photos of the same scene, keeping the camera position roughly fixed and only changing the shooting angle to obtain image pairs with projective transformation relationships. This ensures that there is the required perspective transformation relationship between the images.

Results: The following shows examples of the images I captured.



Image 1



Image 2



Image 3



Image 1



Image 2



Image 3

Part A.2: Recover Homographies

Description: Before aligning images, you need to recover the transformation parameters between each pair of images. In our case, the transformation is a homography: $p' = Hp$, where H is a 3×3 matrix with 8 degrees of freedom (the lower right corner is a scaling factor that can be set to 1).

My approach: Implemented the `computeH` function to calculate the homography matrix H through n corresponding point pairs. Using linear system solving method, the problem is transformed into the form $Ah=b$, where h is a vector of 8 unknown entries of H . When $n > 4$, the system is overdetermined, and the least squares method is used to solve it.

Main code:

▶ [Show Code](#)

Results: Example of homography matrix obtained by this function: [[
 1.05280819e+00 8.95665586e-02 -2.70773387e+01] [-8.27119946e-02
 1.05850720e+00 -2.57703311e+01] [-7.77872637e-05 3.13425327e-04
 1.00000000e+00]]

Part A.3: Warp the Images

Description: Knowing the parameters of the homography transformation, you can use this homography matrix to transform each image toward the reference image. Two interpolation methods were implemented: nearest neighbor interpolation and bilinear interpolation.

My approach: Implemented the `warplImage` function, supporting two interpolation methods. Using inverse transformation (to avoid holes in the

output image), calculating the bounding box of the output image, and then performing inverse transformation for each pixel to find the corresponding position in the source image.

Main code:

► Show Code

Results: The following shows the results of image transformation using two interpolation methods.



Nearest Neighbor
Interpolation



Bilinear Interpolation



Nearest Neighbor
Interpolation



Bilinear Interpolation



Nearest Neighbor
Interpolation



Bilinear Interpolation

Part A.4: Blend Images into a Mosaic

Description: Fuse the transformed images to create an image mosaic.

Alpha blending is used to achieve seamless stitching.

My approach: Implemented the `create_mosaic` function, which transforms and aligns multiple images according to the calculated homography matrices, and then uses alpha blending technique to fuse them into a seamless panoramic image. The alpha value is calculated based on the distance from the center of the image to achieve smooth transition.

Main code:

► Show Code

Results: The following shows the image mosaic results generated using two interpolation methods.



Nearest Neighbor Interpolation
Mosaic



Bilinear Interpolation Mosaic



Nearest Neighbor Interpolation
Mosaic



Bilinear Interpolation Mosaic



Nearest Neighbor Interpolation
Mosaic



Bilinear Interpolation Mosaic



Bilinear Interpolation Mosaic

Nearest Neighbor Interpolation

Mosaic



Nearest Neighbor Interpolation

Mosaic



Bilinear Interpolation Mosaic

Part A.5: Summary

Project completion status:

1. Implemented the homography matrix calculation function `computeH`
2. Implemented two interpolation methods: nearest neighbor interpolation and bilinear interpolation
3. Completed the image rectification function
4. Completed the image stitching function
5. Used Alpha blending technology to achieve seamless stitching

Problems encountered and solutions:

1. During image transformation, boundary processing is a key issue. By calculating the bounding box of the transformed image, all content is ensured to be included.
2. When stitching images, direct stitching produces obvious seams. By using Alpha blending technology, the mixing weight is determined according to the pixel distance from the center of the image, achieving smooth transition.
3. Coordinate system conversion needs to be handled correctly during inverse transformation to ensure accurate pixel position mapping.

Comparison of two interpolation methods:

The nearest neighbor interpolation method is simple and fast, but produces jagged edges and lower image quality.

The bilinear interpolation method has slightly higher computational complexity, but can produce smoother image effects and higher quality.

In practical applications, bilinear interpolation is a better choice.

FEATURE MATCHING for AUTOSTITCHING

Part B: Feature Matching for Autostitching

Part B.1: Harris Corner Detection

Description: Implemented Harris corner detection to identify interest points in images. Also implemented Adaptive Non-Maximal Suppression (ANMS) to select a well-distributed subset of corners.

My approach: Used the provided harris.py implementation for corner detection. For ANMS, implemented the algorithm to select corners with good spatial distribution by maximizing the minimum distance to stronger corners.

Results: The following shows Harris corners overlaid on images, with and without ANMS.



Harris Corners - Center Image



Harris Corners - Side Image



ANMS Corners - Center Image



ANMS Corners - Side Image

Harris Corners - Center Image
(Group 2)ANMS Corners - Center Image
(Group 2)

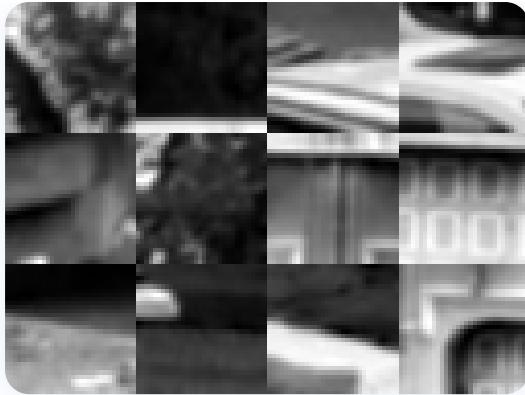
Part B.2: Feature Descriptor Extraction

Description: Extracted feature descriptors for each corner point.

Following the paper, we extract 8x8 descriptors from 40x40 windows with bias/gain normalization.

My approach: For each corner point, sampled a 40x40 window, applied Gaussian blur, resized to 8x8, and normalized to zero mean and unit variance.

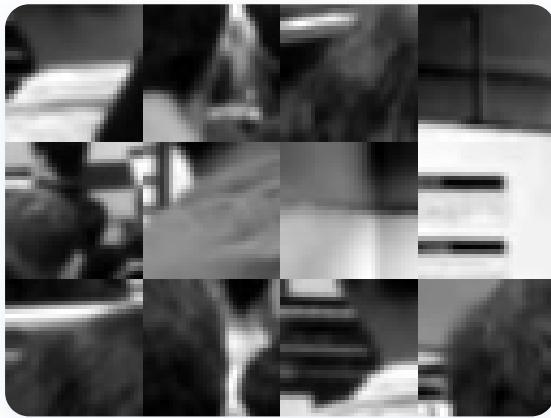
Results: Example extracted feature patches shown below.



Feature Patches – Center
Image



Feature Patches – Side
Image



Feature Patches – Center
Image (Group 2)



Feature Patches – Side Image
(Group 2)

Part B.3: Feature Matching

Description: Matched feature descriptors between image pairs using Lowe's ratio test to identify good matches.

My approach: For each descriptor in one image, found the two nearest neighbors in the other image and used the ratio of their distances as a threshold to determine good matches.

Results: Feature matches between image pairs shown below.



Feature Matches



Feature Matches (Group 2)

Part B.4: RANSAC for Robust Homography

Description: Used RANSAC to compute a robust homography estimate from the feature matches, then applied it to create image mosaics.

My approach: Implemented 4-point RANSAC to estimate homography. Randomly sampled 4 matching point pairs, computed the homography, and counted inliers. Re-estimated using all inliers for the final homography.

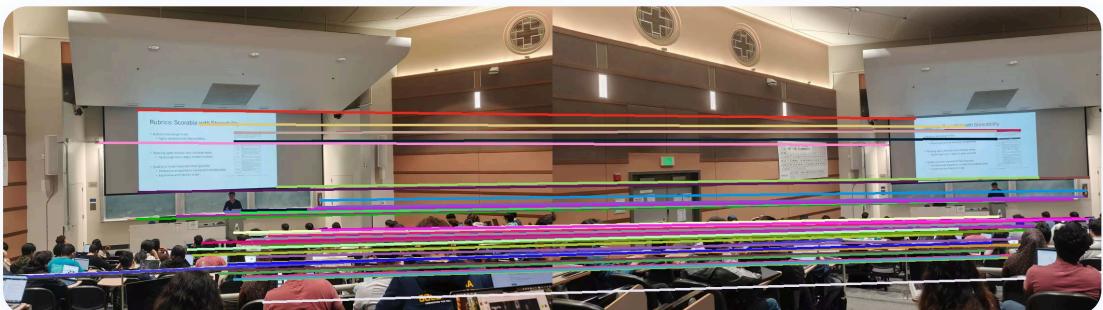
Results: Inlier matches and resulting mosaics shown below.



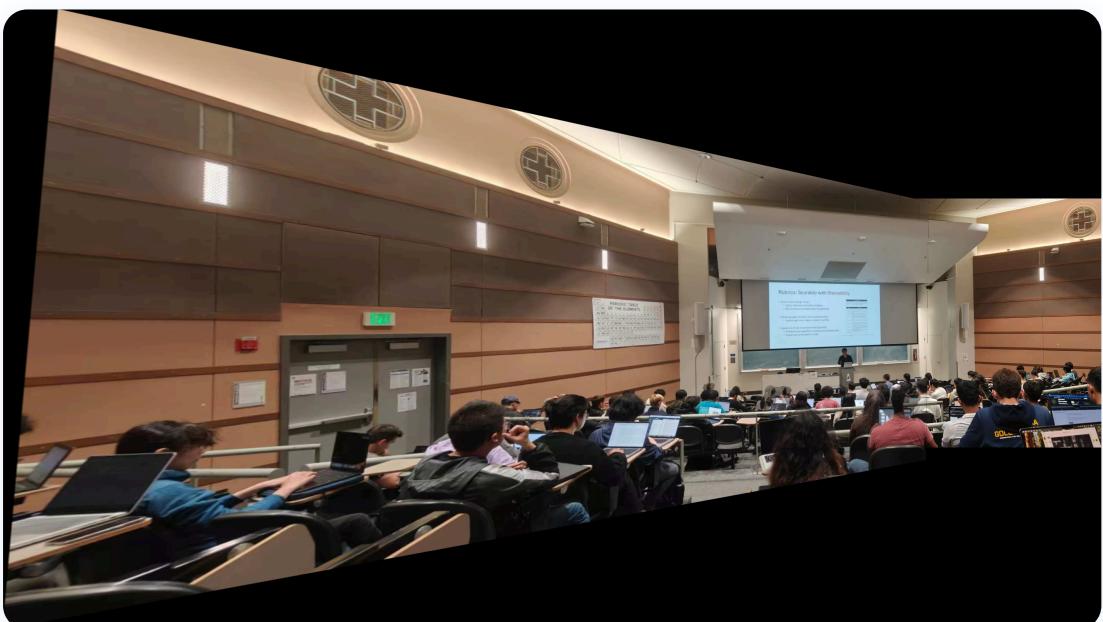
Inlier Matches



Panorama Result



Inlier Matches (Group 2)



Panorama Result (Group 2)



Panorama Result (Group 4)



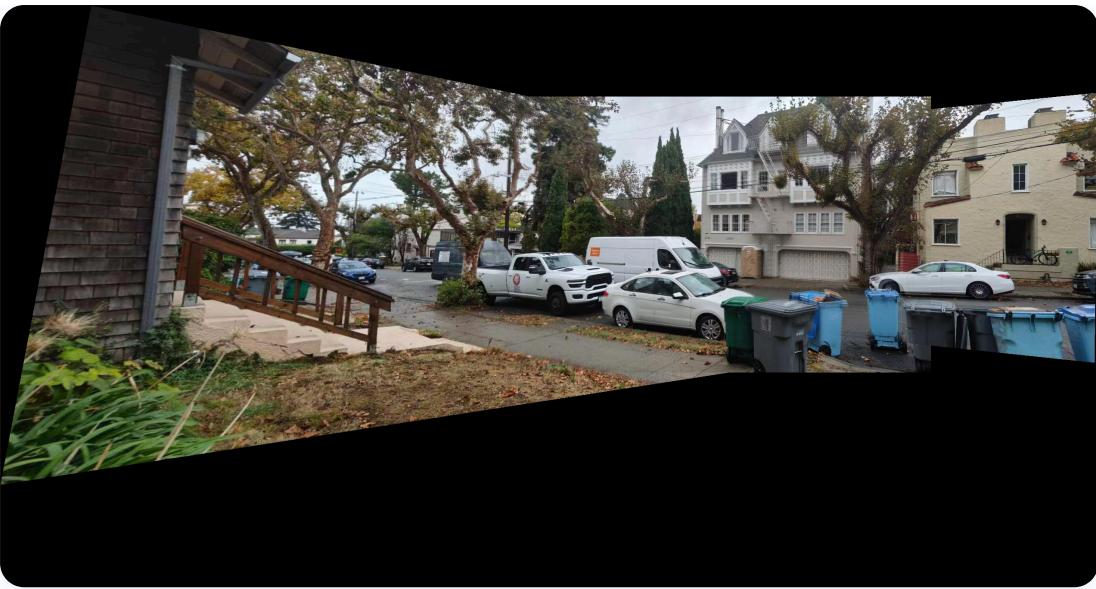
Panorama Result (Group 5)

Automatic vs Manual Stitching Comparison

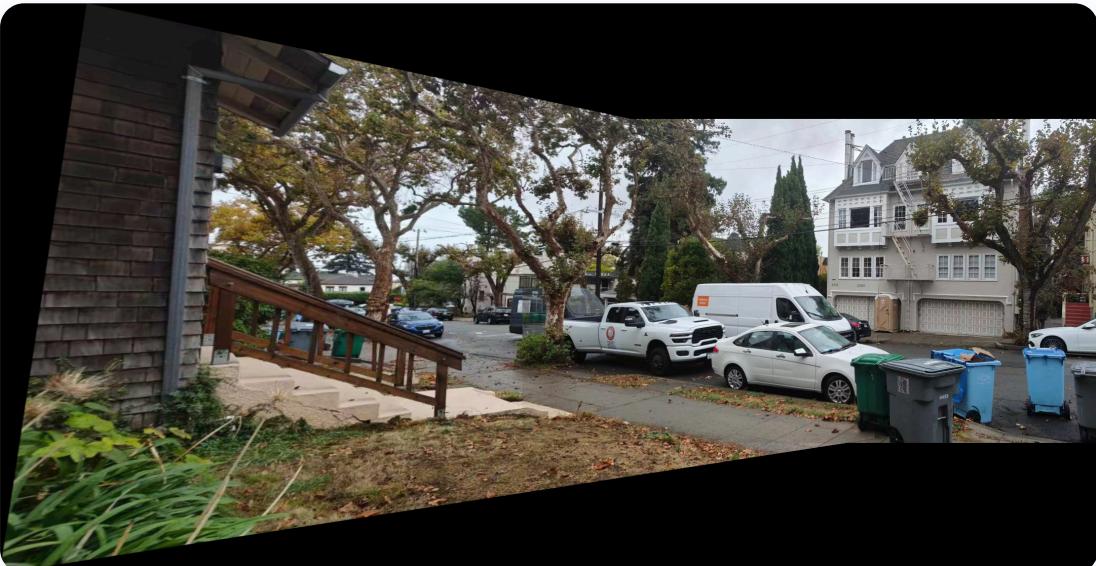
Description: Comparison between automatically stitched panoramas (using feature matching) and manually stitched panoramas (from Part A).

My approach: Used the same image sets for both automatic and manual stitching to provide a fair comparison.

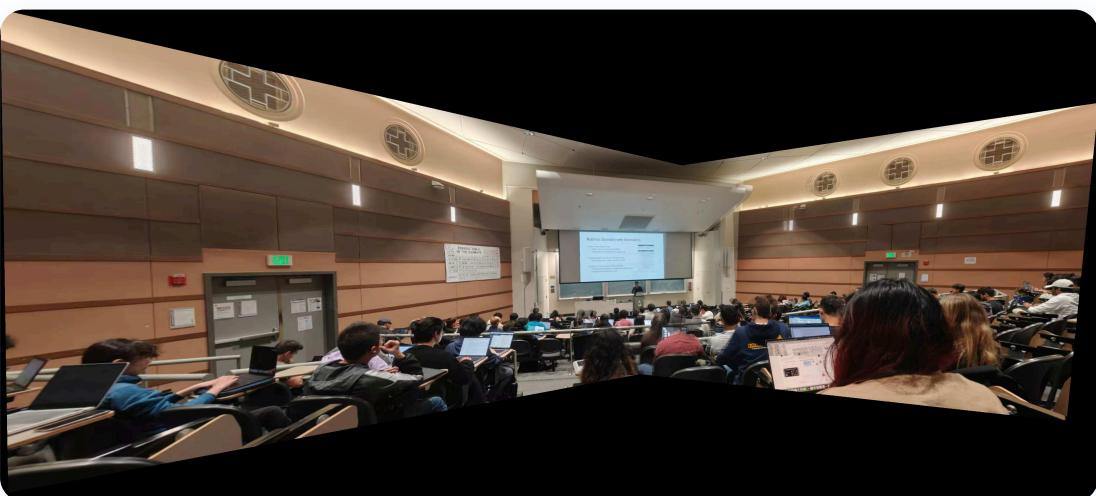
Results: Comparison shown below.



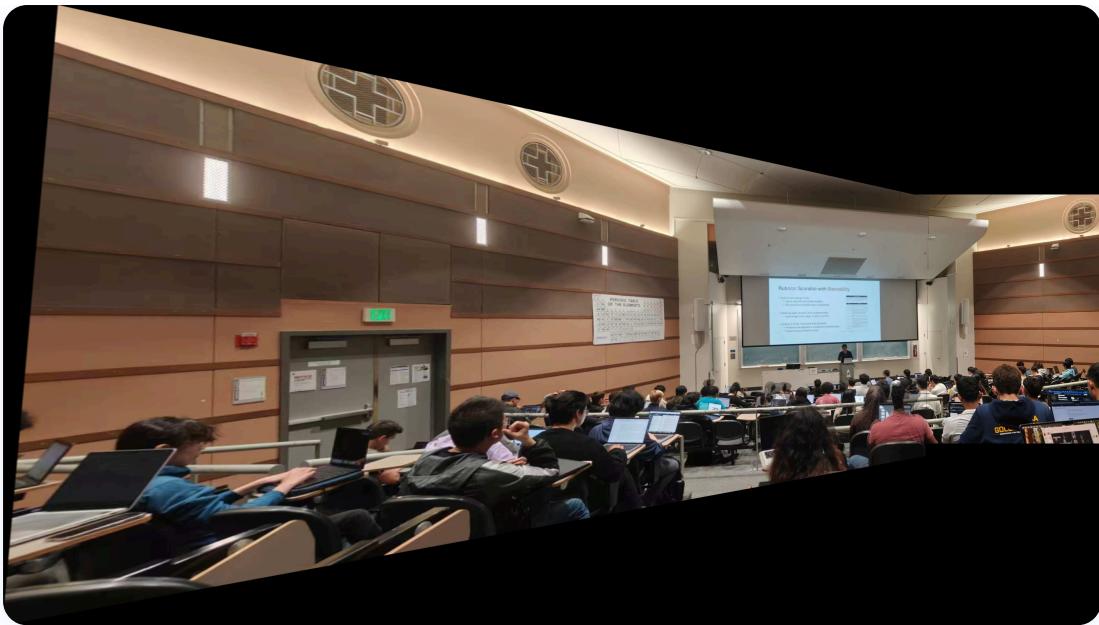
Manual Stitching Result (Group 1)



Automatic Stitching Result (Group 1)



Manual Stitching Result (Group 2)



Automatic Stitching Result (Group 2)

Part B.5: Summary

Project completion status:

1. Implemented Harris corner detection and ANMS for feature point selection
2. Extracted 8x8 feature descriptors from 40x40 windows with normalization
3. Implemented feature matching with Lowe's ratio test
4. Implemented 4-point RANSAC for robust homography estimation
5. Created automatic image mosaics from unaligned images

Problems encountered and solutions:

1. Feature matching sometimes produced false matches. Solved by using Lowe's ratio test to filter matches.
2. Homography estimation was sensitive to outliers. Solved by using RANSAC to find the best set of inliers.
3. Some images failed to stitch due to insufficient matches. Improved by adjusting ANMS parameters to get more features.

Comparison with manual stitching:

Manual stitching requires user interaction to select corresponding points, which is time-consuming and error-prone.

Automatic stitching using feature matching is much faster and more robust, especially for multiple images.

However, automatic methods may fail when images have repetitive patterns or low texture.

Future improvements:

1. Add multi-scale processing for corner detection
2. Implement rotation invariance for feature descriptors
3. Add panorama recognition from unordered image sets