SHOBHIT RAJ.
2022482.

CV ASSIGNMENT - 2

Q1. THEORY

As rotation about Y axis by angle θ is given by:

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$\therefore R_y(-\pi/6) = \begin{bmatrix} \cos(-\pi/6) & 0 & \sin(\pi/6) \\ 0 & 1 & 0 \\ -\sin(-\pi/6) & 0 & \cos(-\pi/6) \end{bmatrix} = \begin{bmatrix} \sqrt{3}/2 & 0 & -1/2 \\ 0 & 1 & 0 \\ 1/2 & 0 & \sqrt{3}/2 \end{bmatrix}$$

As rotation about X axis by angle θ is given by:

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$\therefore R_x(\pi/4) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\pi/4) & -\sin(\pi/4) \\ 0 & \sin(\pi/4) & \cos(\pi/4) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

As reflection across XZ plane negates y coordinate $\rightarrow R_{XZ} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

PART 1 - Combined linear transformation matrix:

$$C = R_{XZ}\, R_x(\pi/4)\, R_y(-\pi/6)$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/\sqrt{2} & -1/\sqrt{2} \\ 0 & 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \begin{bmatrix} \sqrt{3}/2 & 0 & -1/2 \\ 0 & 1 & 0 \\ 1/2 & 0 & \sqrt{3}/2 \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3}/2 & 0 & -1/2 \\ -1/2\sqrt{2} & 1/\sqrt{2} & -\sqrt{3}/2\sqrt{2} \\ 1/2\sqrt{2} & 1/\sqrt{2} & \sqrt{3}/2\sqrt{2} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{-1}{2} \\ \frac{1}{2\sqrt{2}} & \frac{-1}{\sqrt{2}} & \frac{\sqrt{3}}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{\sqrt{3}}{2\sqrt{2}} \end{bmatrix}$$

PART 2 - $v_{new} = Cv + t$, $C \rightarrow$ combined linear transformation
$t \rightarrow$ translation vector $\rightarrow \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$

$$v = \begin{bmatrix} 3 \\ -1 \\ 4 \end{bmatrix} \rightarrow v_{new} = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{-1}{2} \\ \frac{1}{2\sqrt{2}} & \frac{-1}{\sqrt{2}} & \frac{\sqrt{3}}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{\sqrt{3}}{2\sqrt{2}} \end{bmatrix} \begin{bmatrix} 3 \\ -1 \\ 4 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{3\sqrt{3}-4}{2} \\ \frac{5+4\sqrt{3}}{2\sqrt{2}} \\ \frac{1+4\sqrt{3}}{2\sqrt{2}} \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix} = \begin{bmatrix} \frac{3\sqrt{3}-2}{2} \\ \frac{5+4\sqrt{3}}{2\sqrt{2}} \\ \frac{1+4\sqrt{3}-4\sqrt{2}}{2\sqrt{2}} \end{bmatrix}.$$

When $v = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow v_{new} = \underbrace{Cv}_{\downarrow 0} + t = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}$

$$\text{Origin}_{new} = \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix}.$$

PART 3 - Combined Rotation = $R_x(\pi/4)\, R_y(-\pi/6)$

$$C_R = \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{-1}{2} \\ \frac{-1}{2\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{-\sqrt{3}}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{\sqrt{3}}{2\sqrt{2}} \end{bmatrix}$$

For rotation matrix $R \rightarrow tr(R) = 1 + 2\cos\theta$ (where θ = rotation angle)

$$\therefore \theta = \arccos\left(\frac{tr(C_R) - 1}{2}\right) = \arccos\left(\frac{\sqrt{3} + 1 + \sqrt{6}}{4\sqrt{2}}\right)$$

$$= \arccos\left(\frac{\sqrt{6} + 2 + \sqrt{3} - 2\sqrt{2}}{4\sqrt{2}}\right)$$

$$\approx 53.65^\circ \rightsquigarrow \text{rotation angle}.$$

P. T. O.

As axis of rotation is eigenvector (unit) associated with eigenvalue 1.

$$u = \frac{1}{2\sin\theta}\begin{bmatrix} C_{R_{32}} - C_{R_{23}} \\ C_{R_{13}} - C_{R_{31}} \\ C_{R_{21}} - C_{R_{12}} \end{bmatrix} \quad \text{where } \theta = 53.65°$$

$$\therefore u = \begin{bmatrix} \left(\frac{1}{\sqrt{2}} + \frac{\sqrt{3}}{2\sqrt{2}}\right)/2\sin\theta \\ \left(\frac{-1}{2} - \frac{1}{2\sqrt{2}}\right)/2\sin\theta \\ \left(\frac{-1}{2\sqrt{2}} - 0\right)/2\sin\theta \end{bmatrix} \quad \& \sin\theta = \sin(53.65°) \approx 0.805$$

$$\Rightarrow u \approx \begin{bmatrix} 0.8195 \\ -0.5301 \\ -0.2196 \end{bmatrix} \rightsquigarrow \text{axis of rotation}$$

PART 4 -

Rodrigui's rotation formula:-

$$R = I + (\sin\theta)[u]_x + (1 - \cos\theta)[u_x]^2$$

where θ = angle of rotation

$[u]_x$ = skew symmetric matrix $\rightarrow \begin{bmatrix} 0 & -u_z & +u_y \\ +u_z & 0 & -u_x \\ -u_y & +u_x & 0 \end{bmatrix}$

$$\therefore R = I + (0.805)\begin{bmatrix} 0 & 0.2196 & -0.5301 \\ -0.2196 & 0 & -0.8195 \\ 0.5301 & 0.8195 & 0 \end{bmatrix} +$$

$$(0.407)\begin{bmatrix} -0.3292 & -0.4344 & -0.1799 \\ -0.4344 & -0.7198 & 0.1164 \\ -0.1799 & 0.1164 & -0.9525 \end{bmatrix}$$

$$= \begin{bmatrix} 0.8660 & 0 & -0.4999 \\ -0.3535 & 0.7070 & -0.6123 \\ 0.3535 & 0.7070 & 0.6123 \end{bmatrix} \approx \begin{bmatrix} \frac{\sqrt{3}}{2} & 0 & \frac{-1}{2} \\ \frac{-1}{2\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{-\sqrt{3}}{2\sqrt{2}} \\ \frac{1}{2\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{\sqrt{3}}{2\sqrt{2}} \end{bmatrix}$$

which is C_R obtained in part 3.

Hence, proved.

Q2. THEORY

As $x = K[R|t]X$, ($t=0$ as pure rotation R)

Camera 1 → $x_1 = K_1[I|0]X$ // world coordinate frame

Camera 2 → $x_2 = K_2[R|0]X$ // pure rotation

$\therefore x_1 = K_1 X \Rightarrow X = K_1^{-1} x_1$

$x_2 = K_2 R X$

putting here

$\Rightarrow x_2 = K_2 R (K_1^{-1} x_1)$

$\therefore x_2 = (K_2 R K_1^{-1}) x_1 \rightsquigarrow x_2 = H' x_1 \Rightarrow x_1 = (H')^{-1} x_2$

Given → $x_1 = H x_2 \Rightarrow H = (H')^{-1}$

$= (K_2 R K_1^{-1})^{-1}$

$= (K_1 R^{-1} K_2^{-1})$.

As K_1 & K_2 are camera intrinsic parameters matrices, they are invertible & R is pure rotation matrix (orthonormal i.e. $R^T = R^{-1}$), thus invertible.

$\therefore H = K_1 R^{-1} K_2^{-1}$ or $K_1 R^T K_2^{-1}$ is invertible. (as K_1, K_2 & R are invertible)

Question 3: Camera Calibration

This task involves performing **camera calibration** using a checkerboard pattern. The calibration was conducted separately for **25 provided images** and **25 self-captured images**. The goal was to estimate intrinsic parameters, extrinsic parameters, radial distortion coefficients, and reprojection errors while visualizing the accuracy of the calibration.

To perform calibration, I placed a chessboard pattern on a flat surface and captured 25 images at different angles. The images of my dataset are shown below :

Figure 1: Self-captured images of the chessboard pattern used for camera calibration

The `OpenCV library` was used to detect checkerboard corners automatically, refining corners and camera calibration. A JSON file was generated, containing the estimated intrinsic parameters, extrinsic parameters, and radial distortion coefficients. For the given dataset, the output was saved to `2022482_parameters.json` and for my self-captured dataset, the output was saved to `2022482_my_camera_parameters.json`.

Part 1. Intrinsic Camera Parameters

Estimated intrinsic camera parameters on the given dataset:

- **Focal lengths** (f_x, f_y): $(956.6372, 957.5514)$
- **Skew parameter**: 0.0
- **Principal point** (c_x, c_y): $(369.0549, 651.4142)$

Estimated intrinsic camera parameters on my self-captured dataset:

- **Focal lengths** (f_x, f_y): $(905.4646, 906.3595)$
- **Skew parameter**: 0.0
- **Principal point** (c_x, c_y): $(646.5824, 351.3841)$

Part 2. Extrinsic Camera Parameters

For the first two images, I extracted the rotation matrix and translation vector, which define how the checkerboard is positioned relative to the camera. These parameters describe the camera's movement and orientation with respect to the chessboard.

Extrinsic parameters for the first two images in the given dataset:

Image 1:

$$\text{Rotation matrix: } R_1 = \begin{bmatrix} 0.996998 & 0.013872 & -0.076168 \\ -0.025115 & 0.988563 & -0.148704 \\ 0.073234 & 0.150171 & 0.985944 \end{bmatrix}$$

$$\text{Translation vector: } t_1 = \begin{bmatrix} -3.9383 \\ -3.6172 \\ 14.6591 \end{bmatrix}$$

Image 2:

$$\text{Rotation matrix: } R_2 = \begin{bmatrix} 0.890522 & -0.241950 & -0.385268 \\ 0.122456 & 0.943077 & -0.309208 \\ 0.438151 & 0.228178 & 0.869459 \end{bmatrix}$$

$$\text{Translation vector: } t_2 = \begin{bmatrix} -1.4977 \\ -0.6579 \\ 13.9182 \end{bmatrix}$$

Extrinsic parameters for the first two images in my self-captured dataset:

Image 1:

$$\text{Rotation matrix: } R_1 = \begin{bmatrix} 0.998545 & 0.017923 & -0.050855 \\ -0.009972 & 0.988256 & 0.152484 \\ 0.052991 & -0.151755 & 0.986997 \end{bmatrix}$$

$$\text{Translation vector: } t_1 = \begin{bmatrix} -3.8255 \\ -3.0499 \\ 15.2810 \end{bmatrix}$$

Image 2:

$$\text{Rotation matrix: } R_2 = \begin{bmatrix} 0.994484 & 0.027041 & -0.101341 \\ -0.012743 & 0.990188 & 0.139161 \\ 0.104109 & -0.137102 & 0.985071 \end{bmatrix}$$

$$\text{Translation vector: } t_2 = \begin{bmatrix} -3.9050 \\ -4.0306 \\ 19.3159 \end{bmatrix}$$

Part 3. Radial Distortion and Image Undistortion

Radial distortion coefficients for the given dataset:

- k_1: 0.197637
- k_2: -0.706870
- k_3: 0.048796

Radial distortion coefficients for my self-captured dataset:

- k_1: 0.070855
- k_2: -0.187537
- k_3: -0.089375

Using these distortion coefficients, I applied undistortion to five images from each dataset. The results, displaying the checkerboard before and after undistortion, are shown below:

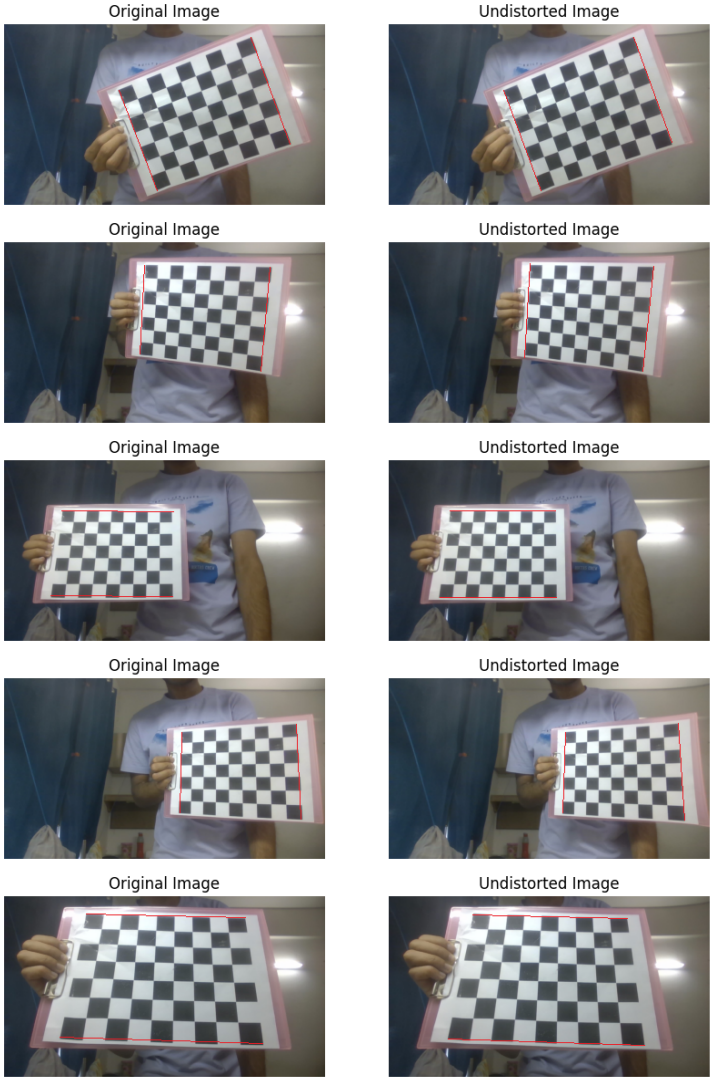

Figure 2: Undistorted images from my self-captured dataset. The correction straightens the edges of the checkerboard

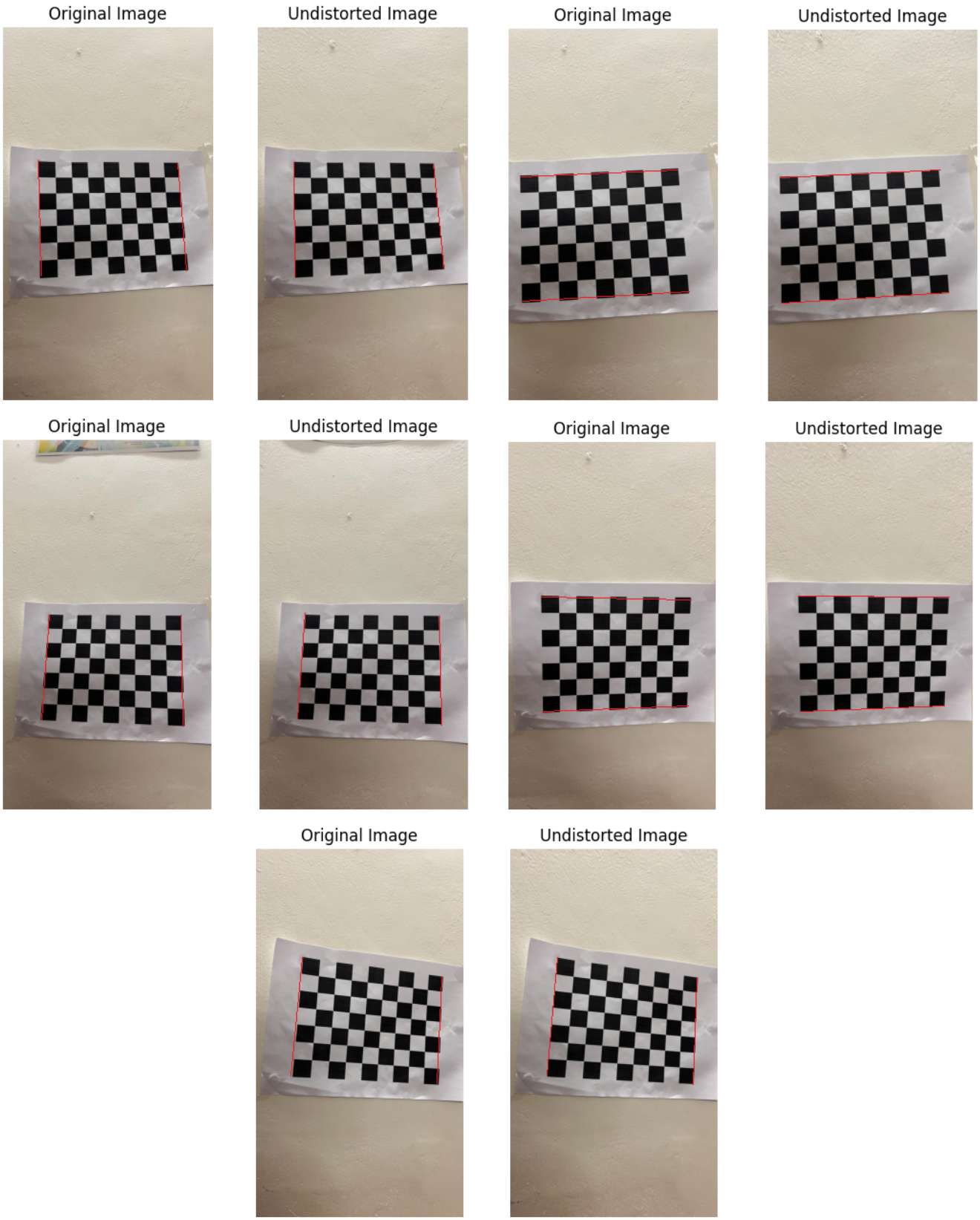

Figure 3: Undistorted images from the given dataset. The corrected images show reduced curvature in straight edges

To illustrate the effect of radial distortion, I have **annotated the edges** of the checkerboard with straight red lines. Before undistortion, the checkerboard edges near the corners of the images appear **slightly curved due to radial distortion**. After undistortion, these **lines appear straighter**, demonstrating the correction process for lens warping.

Part 4. Reprojection Error Computation and Analysis

For each of the 25 images, the reprojection error was computed by projecting the known 3D checkerboard points onto the image using the estimated intrinsic and extrinsic parameters. The error was then measured as the Euclidean distance between the actual detected corners and their corresponding reprojected points.

Reprojection error statistics for the given dataset:

- **Mean reprojection error:** 0.0697
- **Standard deviation:** 0.0232

Reprojection error statistics for my self-captured dataset:

- **Mean reprojection error:** 0.0802
- **Standard deviation:** 0.0279

To visualize the distribution of errors across all images, bar charts were plotted for both datasets, as shown below:

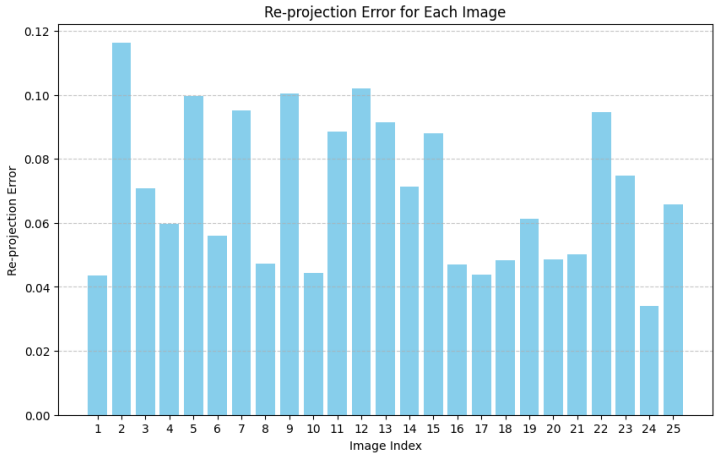

Figure 4: Reprojection error for each image in the given dataset

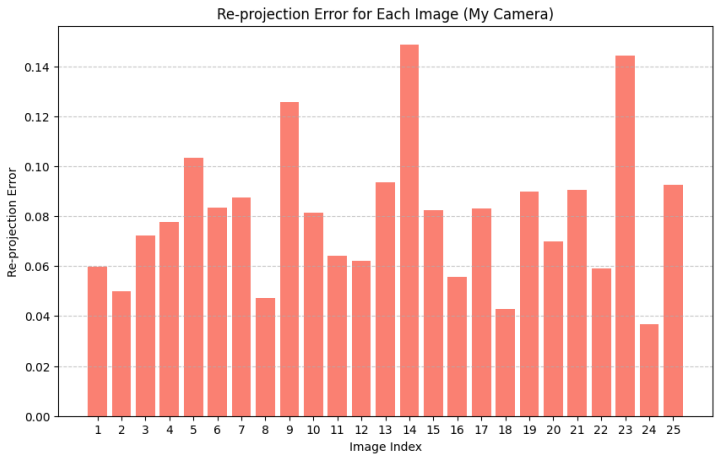

Figure 5: Reprojection error for each image in my self-captured dataset

The results indicate that the **mean reprojection error is low** for both datasets, indicating accurate calibration.

Part 5. Corner Detection vs Reprojection Visualization

To evaluate the accuracy of the calibration, the detected chessboard corners were compared against their reprojected counterparts. For each image, both sets of points were overlaid, allowing a visual inspection of the alignment. The closer the reprojected points are to the detected corners, the lower the reprojection error, indicating better calibration accuracy.

Visualization: Each image consists of three subplots:

- **Combined View:** Green circles represent detected corners, while red crosses represent reprojected corners.
- **Detected Corners Only:** Shows only the detected corners (green).
- **Reprojected Corners Only:** Shows only the reprojected points (red).

Detected vs. Reprojected Corners for Given Dataset:

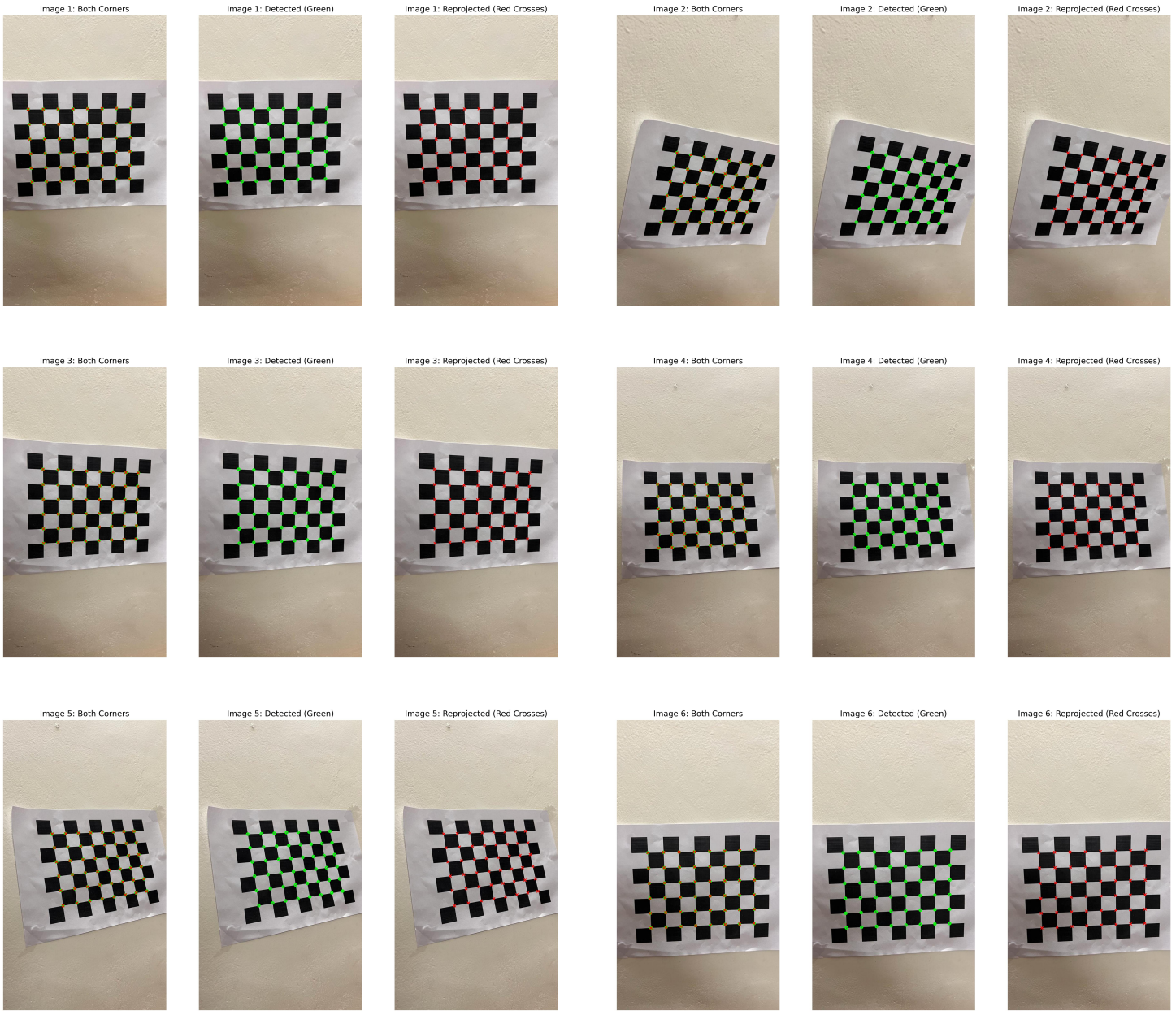

Figure 6: Detected and reprojected corners for images in the given dataset (Images 1-6).

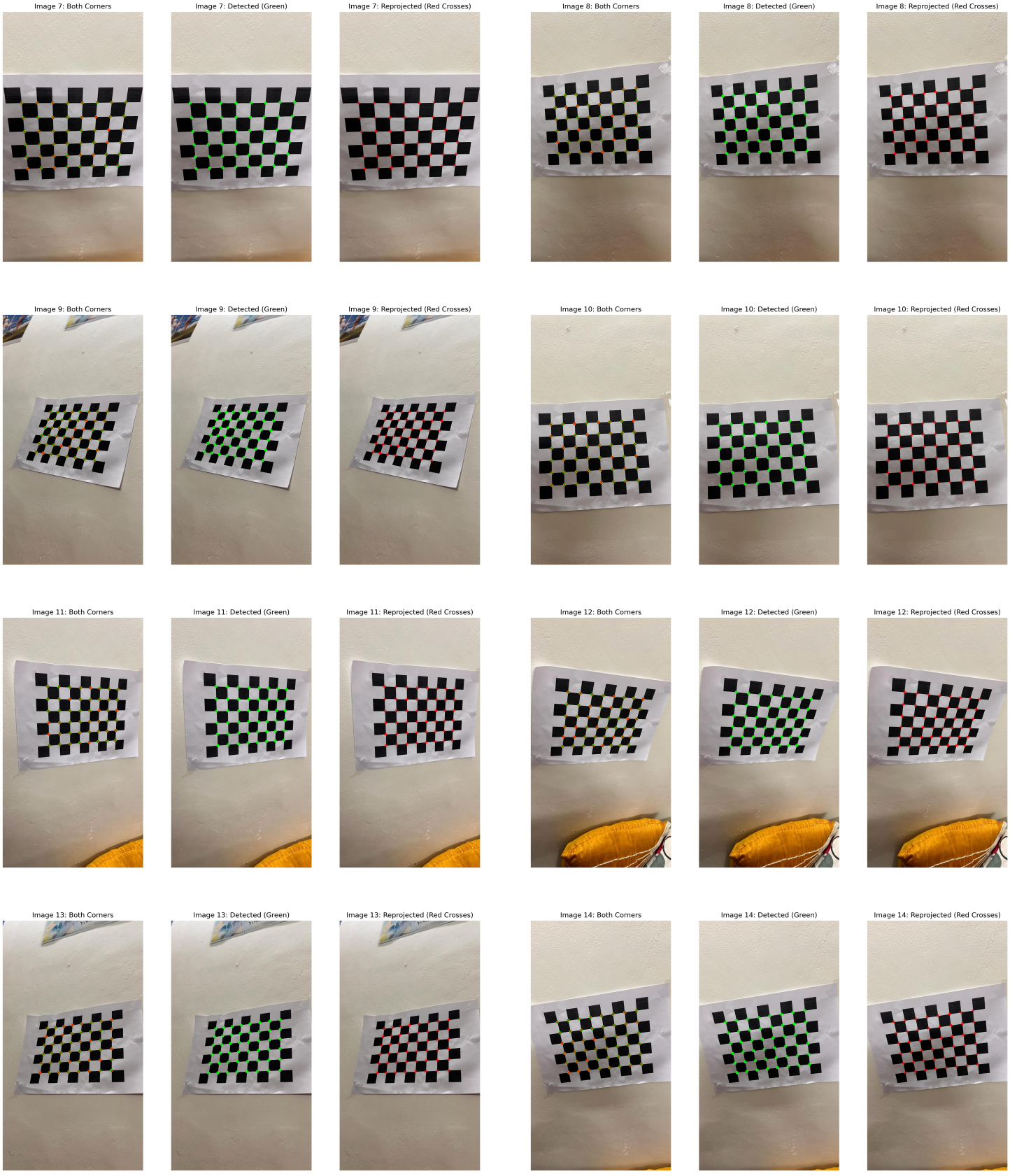

Figure 7: Detected and reprojected corners for images in the given dataset (Images 7-14).

Figure 8: Detected and reprojected corners for images in the given dataset (Images 15-22).

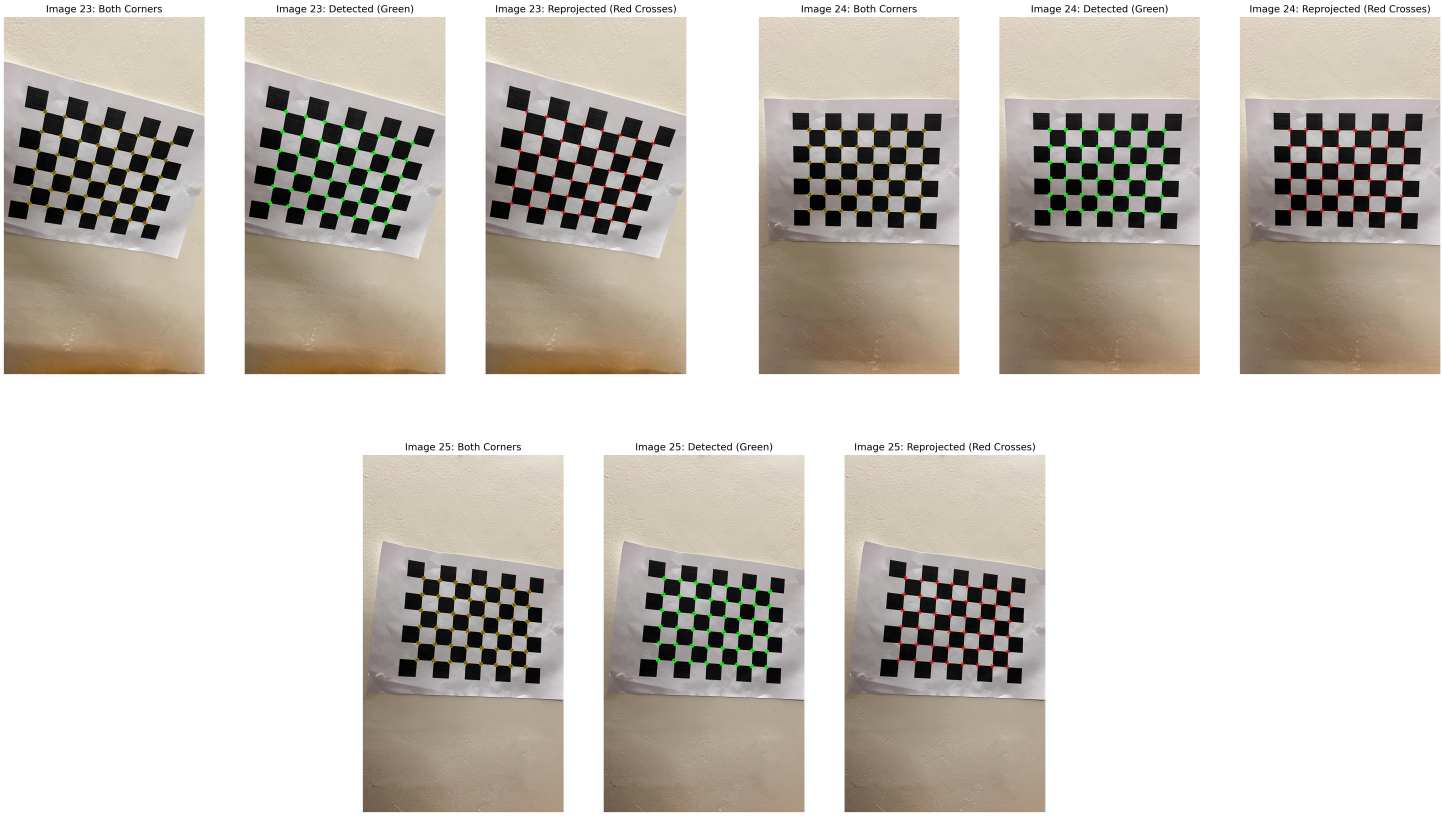

Figure 9: Detected and reprojected corners for images in the given dataset (Images 23-25).

Detected vs. Reprojected Corners for My Self-Captured Dataset:

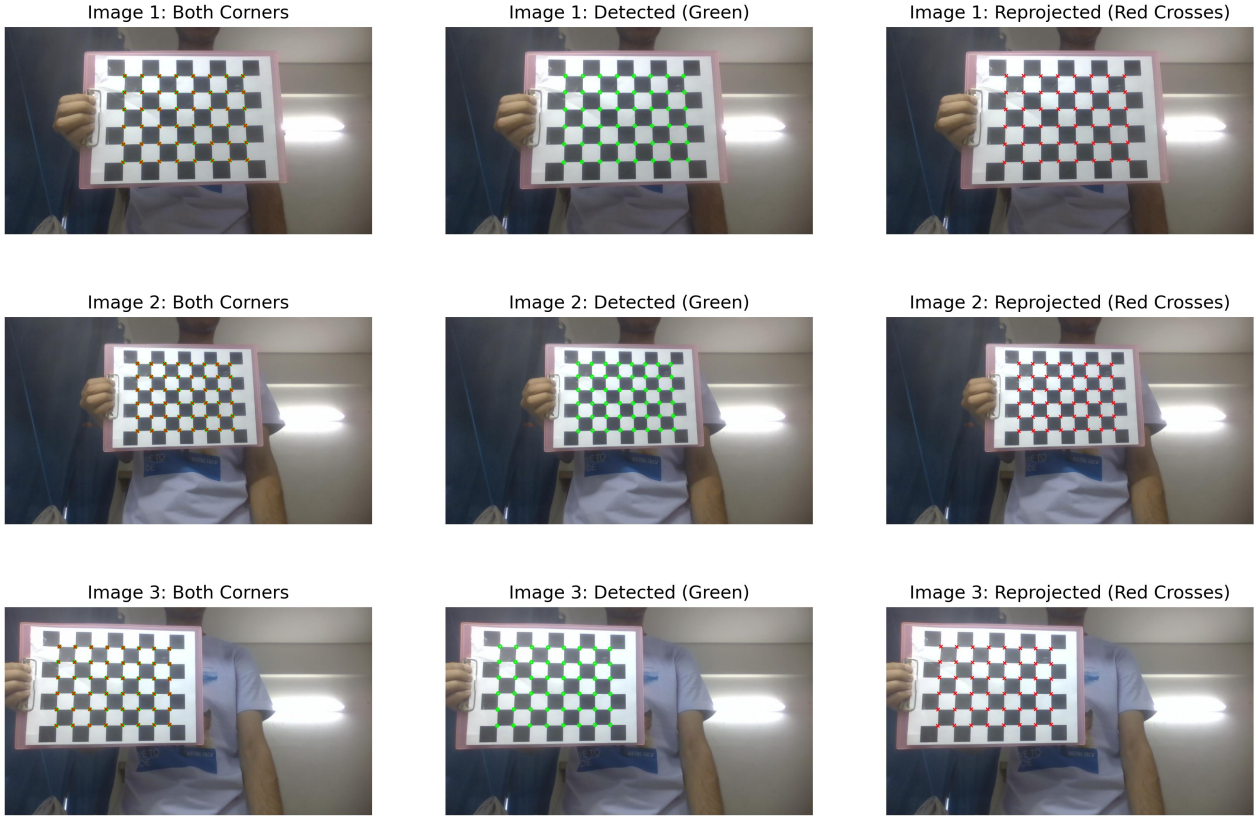

Figure 10: Detected and reprojected corners for images in my self-captured dataset (Images 1-3).

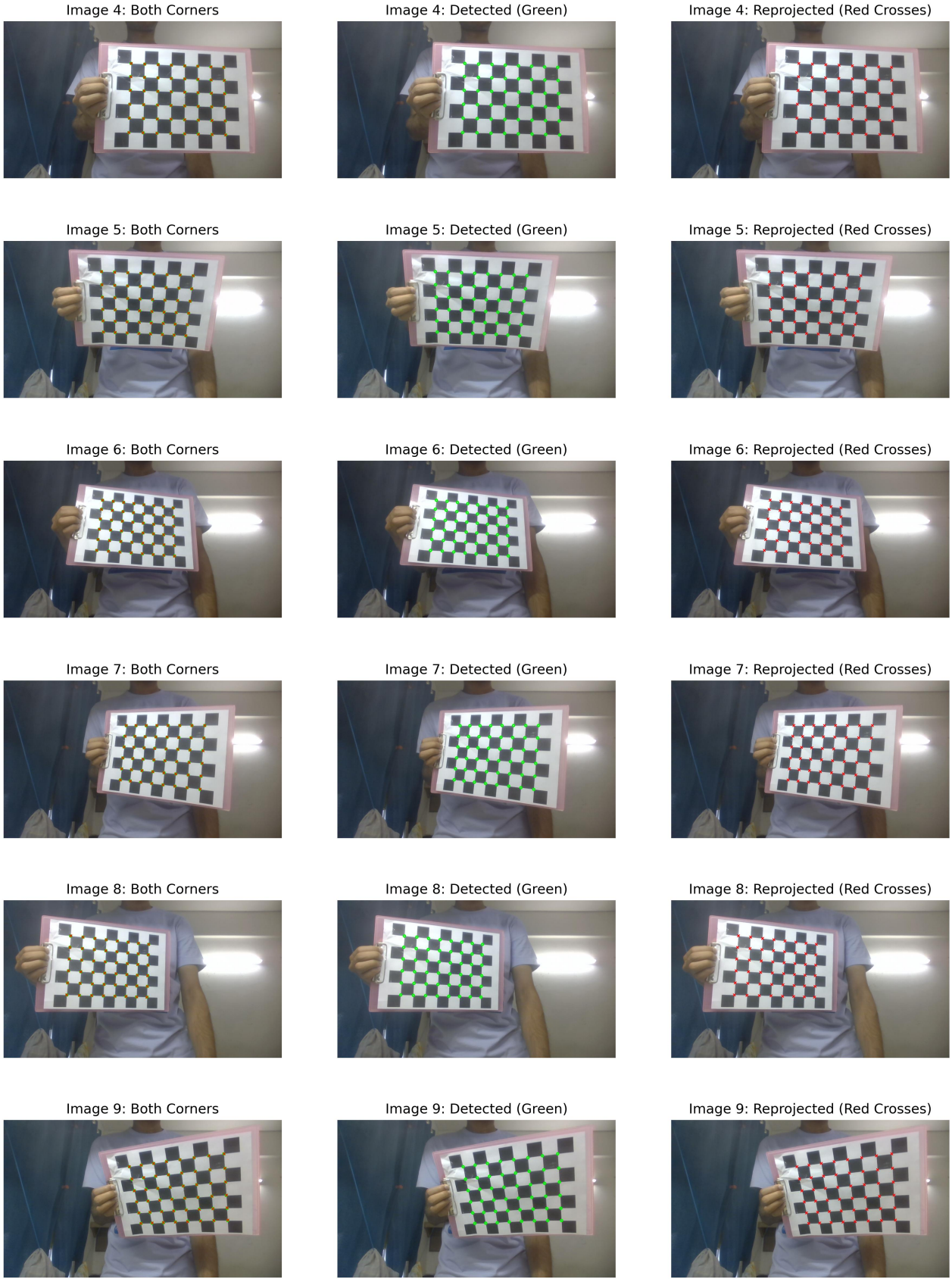

Figure 11: Detected and reprojected corners for images in my self-captured dataset (Images 4-9).

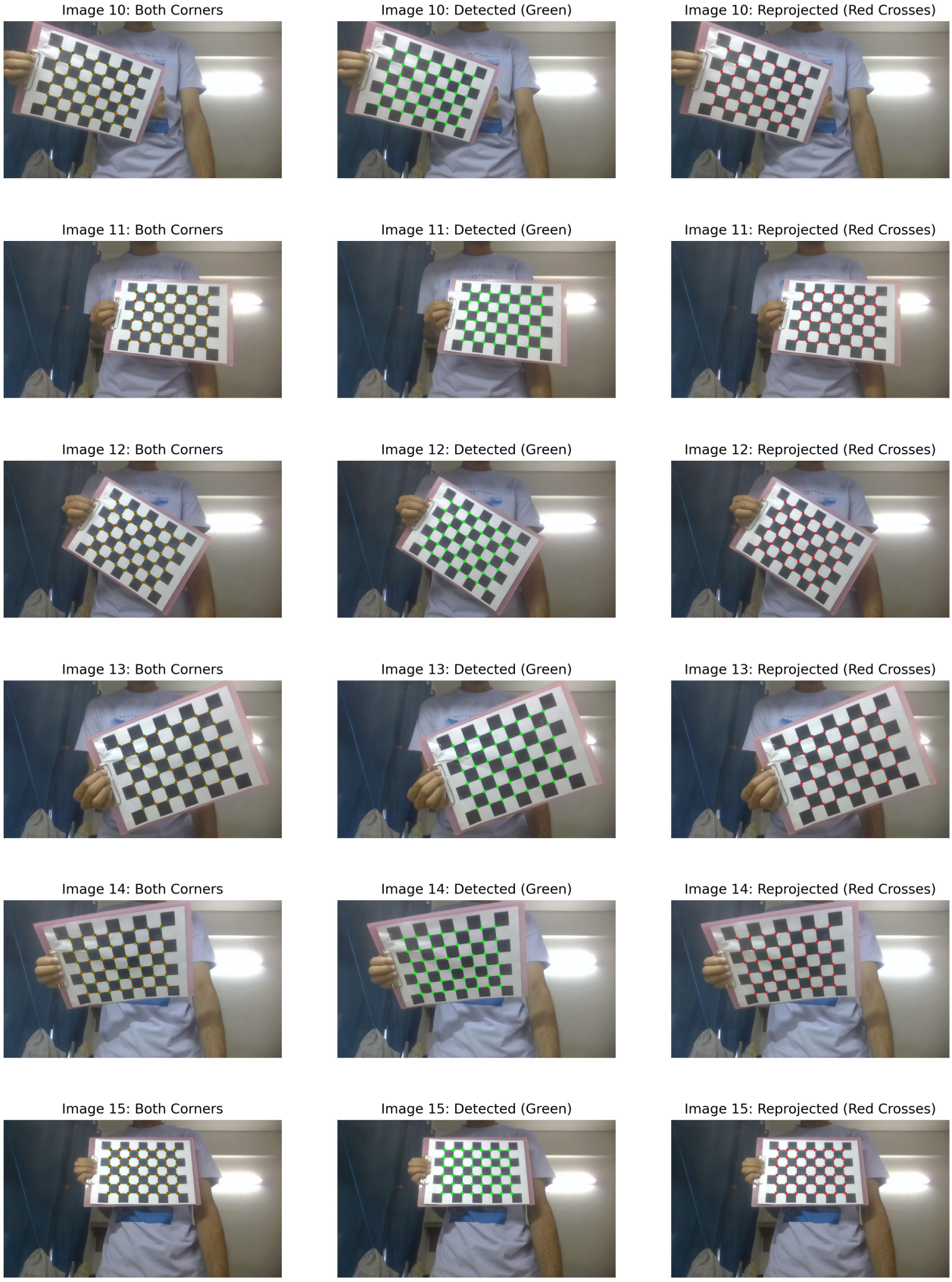

Figure 12: Detected and reprojected corners for images in my self-captured dataset (Images 10-15).

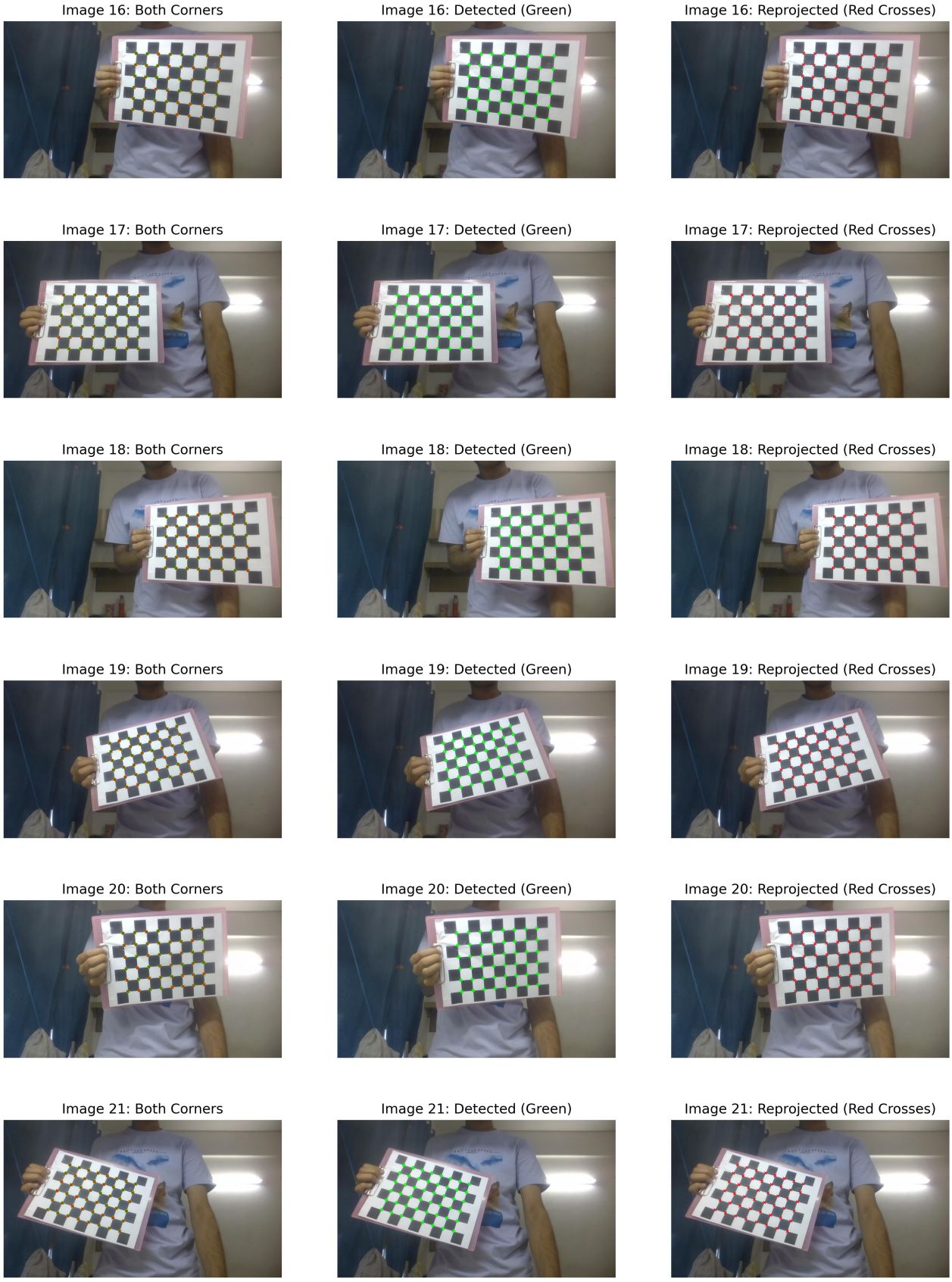

Figure 13: Detected and reprojected corners for images in my self-captured dataset (Images 16-21).

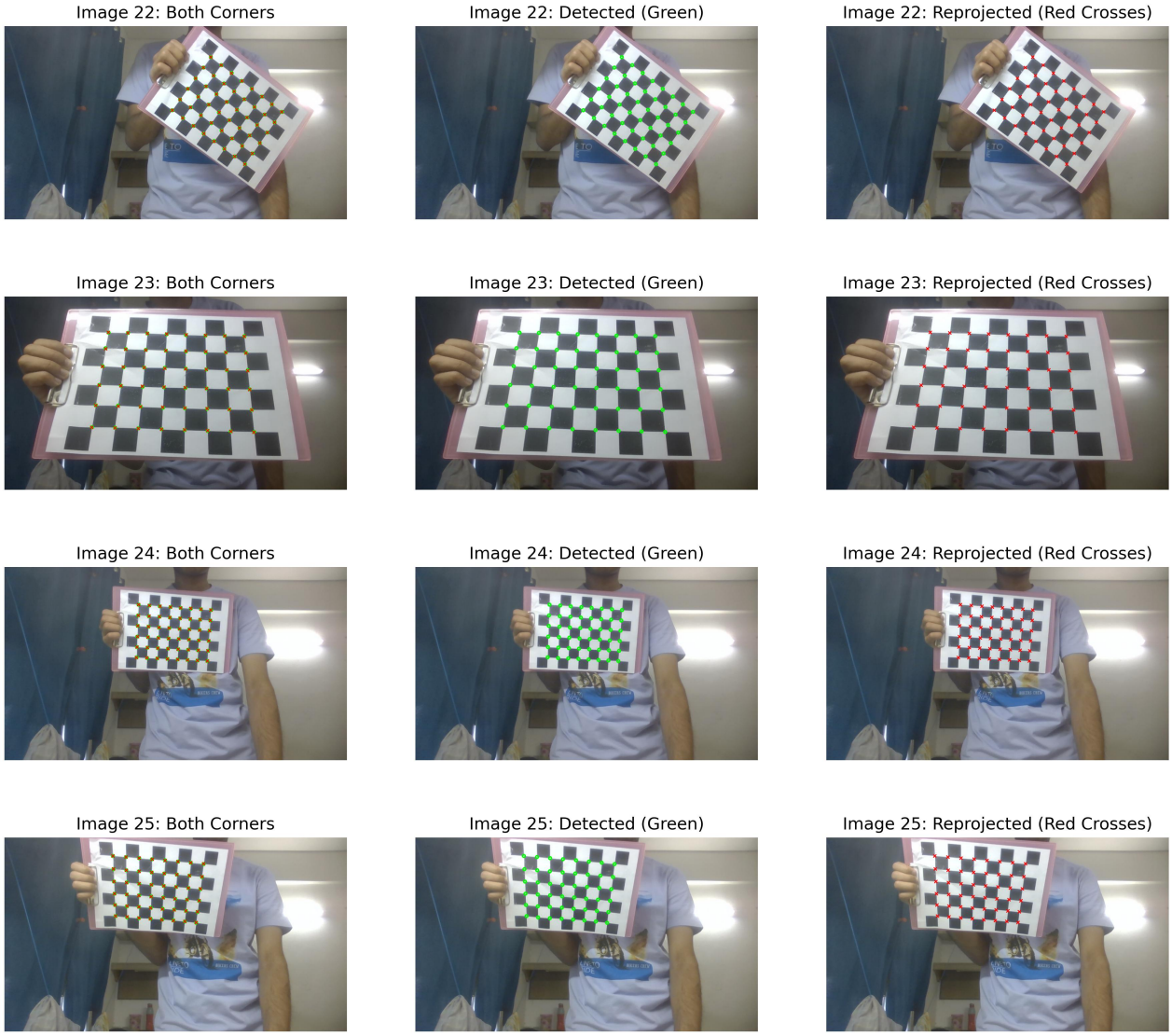

Figure 14: Detected and reprojected corners for images in my self-captured dataset (Images 22-25).

Computation of Reprojection Error:
Reprojection error quantifies how accurately the estimated camera parameters map real-world points to image coordinates. It is computed as the Euclidean distance between:

- The actual detected 2D corner positions $\mathbf{p_i}$ obtained using `cv2.findChessboardCorners`.
- The reprojected 2D positions $\hat{\mathbf{p}}_\mathbf{i}$ obtained using `cv2.projectPoints`.

Mathematically, the reprojection error for each corner i is given by:

$$\text{error}_i = \|\mathbf{p_i} - \hat{\mathbf{p}}_\mathbf{i}\|_2$$

where $\|\cdot\|_2$ represents the Euclidean norm.
The mean reprojection error across all N corners is given by:

$$\frac{1}{N}\sum_{i=1}^{N} \|\mathbf{p_i} - \hat{\mathbf{p}}_\mathbf{i}\|_2$$

Observations: The visualization confirms that the reprojected points closely match the detected corners, with only minor deviations, indicating accurate calibration. The low reprojection error mean validate the same.

Part 6. Checkerboard Plane Normals in Camera Coordinates

To determine the checkerboard plane normals in the camera coordinate system, I extracted the **third column of the rotation matrix** for each image. These normals represent the orientation of the checkerboard relative to the camera in 3D space. The variation in these values across images reflects different angles at which the images were captured.

Checkerboard Plane Normals for the Given Dataset:

- Image 1: $[\mathbf{-0.0762}, \mathbf{-0.1487}, \mathbf{0.9859}]$
- Image 2: $[\mathbf{-0.3853}, \mathbf{-0.3092}, \mathbf{0.8695}]$
- Image 3: $[\mathbf{-0.2714}, \mathbf{-0.0962}, \mathbf{0.9577}]$
- Image 4: $[\mathbf{-0.1658}, \mathbf{0.1601}, \mathbf{0.9731}]$
- Image 5: $[\mathbf{-0.3974}, \mathbf{0.1908}, \mathbf{0.8976}]$
- Image 6: $[\mathbf{-0.0833}, \mathbf{-0.1275}, \mathbf{0.9883}]$
- Image 7: $[\mathbf{-0.1031}, \mathbf{-0.4017}, \mathbf{0.9100}]$
- Image 8: $[\mathbf{0.1888}, \mathbf{-0.0664}, \mathbf{0.9798}]$
- Image 9: $[\mathbf{0.4546}, \mathbf{0.4001}, \mathbf{0.7958}]$
- Image 10: $[\mathbf{-0.0229}, \mathbf{0.1238}, \mathbf{0.9920}]$
- Image 11: $[\mathbf{-0.3983}, \mathbf{-0.1872}, \mathbf{0.8980}]$
- Image 12: $[\mathbf{-0.3464}, \mathbf{-0.5281}, \mathbf{0.7753}]$
- Image 13: $[\mathbf{0.3582}, \mathbf{0.3986}, \mathbf{0.8443}]$
- Image 14: $[\mathbf{0.0452}, \mathbf{-0.3252}, \mathbf{0.9446}]$
- Image 15: $[\mathbf{-0.1278}, \mathbf{-0.5073}, \mathbf{0.8523}]$
- Image 16: $[\mathbf{0.0720}, \mathbf{0.2172}, \mathbf{0.9735}]$
- Image 17: $[\mathbf{-0.0954}, \mathbf{0.1972}, \mathbf{0.9757}]$
- Image 18: $[\mathbf{-0.1461}, \mathbf{0.0739}, \mathbf{0.9865}]$
- Image 19: $[\mathbf{-0.1077}, \mathbf{-0.0864}, \mathbf{0.9904}]$
- Image 20: $[\mathbf{-0.1355}, \mathbf{-0.1060}, \mathbf{0.9851}]$
- Image 21: $[\mathbf{-0.0976}, \mathbf{-0.2745}, \mathbf{0.9566}]$
- Image 22: $[\mathbf{-0.3603}, \mathbf{0.0181}, \mathbf{0.9326}]$
- Image 23: $[\mathbf{-0.1646}, \mathbf{-0.3628}, \mathbf{0.9172}]$
- Image 24: $[\mathbf{0.0357}, \mathbf{-0.1105}, \mathbf{0.9932}]$
- Image 25: $[\mathbf{-0.2233}, \mathbf{0.0683}, \mathbf{0.9724}]$

Checkerboard Plane Normals for My Self-Captured Dataset:

- Image 1: $[\mathbf{-0.0509}, \mathbf{0.1525}, \mathbf{0.9870}]$
- Image 2: $[\mathbf{-0.1013}, \mathbf{0.1392}, \mathbf{0.9851}]$
- Image 3: $[\mathbf{-0.1038}, \mathbf{0.0844}, \mathbf{0.9910}]$
- Image 4: $[\mathbf{-0.0602}, \mathbf{0.1203}, \mathbf{0.9909}]$
- Image 5: $[\mathbf{0.0180}, \mathbf{-0.3358}, \mathbf{0.9418}]$
- Image 6: $[\mathbf{-0.1644}, \mathbf{0.3430}, \mathbf{0.9248}]$
- Image 7: $[\mathbf{0.3616}, \mathbf{0.1128}, \mathbf{0.9255}]$
- Image 8: $[\mathbf{-0.2843}, \mathbf{0.0737}, \mathbf{0.9559}]$
- Image 9: $[\mathbf{0.2653}, \mathbf{-0.3504}, \mathbf{0.8983}]$
- Image 10: $[\mathbf{-0.1792}, \mathbf{-0.2318}, \mathbf{0.9561}]$
- Image 11: $[\mathbf{0.1583}, \mathbf{0.3077}, \mathbf{0.9382}]$
- Image 12: $[\mathbf{0.0930}, \mathbf{0.0673}, \mathbf{0.9934}]$
- Image 13: $[\mathbf{0.0883}, \mathbf{-0.0674}, \mathbf{0.9938}]$
- Image 14: $[\mathbf{0.0390}, \mathbf{-0.4861}, \mathbf{0.8730}]$
- Image 15: $[\mathbf{-0.0303}, \mathbf{0.3226}, \mathbf{0.9461}]$
- Image 16: $[\mathbf{0.2783}, \mathbf{-0.0688}, \mathbf{0.9580}]$
- Image 17: $[\mathbf{-0.0255}, \mathbf{0.2752}, \mathbf{0.9610}]$
- Image 18: $[\mathbf{0.3270}, \mathbf{0.1935}, \mathbf{0.9250}]$
- Image 19: $[\mathbf{0.1393}, \mathbf{0.4293}, \mathbf{0.8924}]$
- Image 20: $[\mathbf{-0.1092}, \mathbf{-0.1436}, \mathbf{0.9836}]$
- Image 21: $[\mathbf{-0.0704}, \mathbf{0.3709}, \mathbf{0.9260}]$
- Image 22: $[\mathbf{-0.0821}, \mathbf{-0.0171}, \mathbf{0.9965}]$
- Image 23: $[\mathbf{-0.0103}, \mathbf{0.4032}, \mathbf{0.9150}]$
- Image 24: $[\mathbf{-0.1122}, \mathbf{-0.1039}, \mathbf{0.9882}]$
- Image 25: $[\mathbf{0.0179}, \mathbf{-0.4585}, \mathbf{0.8885}]$

Each normal vector $\mathbf{n_i^C}$ describes the orientation of the checkerboard plane in the camera's coordinate frame $\mathcal{O}_c$. Since the checkerboard is always facing the camera, the normal vectors $\mathbf{n_i^C}$ have a positive Z-component.

References

- **Camera Calibration using OpenCV tutorial:** [OpenCV Documentation]
- **Step-by-step Guide to Camera Calibration:** [LearnOpenCV]
- **cv2.Rodrigues() Documentation for converting rotation vectors to rotation matrices:** [OpenCVSharp Documentation]
- **Computing Checkerboard Plane Normals:** [Stack Overflow]

Question 4: Panorama Generation

The task was to generate **panoramas** from multiple images belonging to different scenes. The process involved several computer vision techniques including clustering, keypoint detection, feature matching, homography estimation, perspective warping, and finally, multi-stitching. Each task built upon the previous one to gradually transform a set of overlapping images into smooth panoramic images.

I began by **clustering the mixed dataset into three distinct sets** using **color histograms** and **visual bag of words**. For each image, a normalized 3D color histogram was computed & a visual bag of words representation was generated using SIFT descriptors and K-means quantization, and K-means clustering was applied to group similar images. This step ensured that images belonging to the same scene were processed together in later stages. The clusters obtained from both methods are shown below:

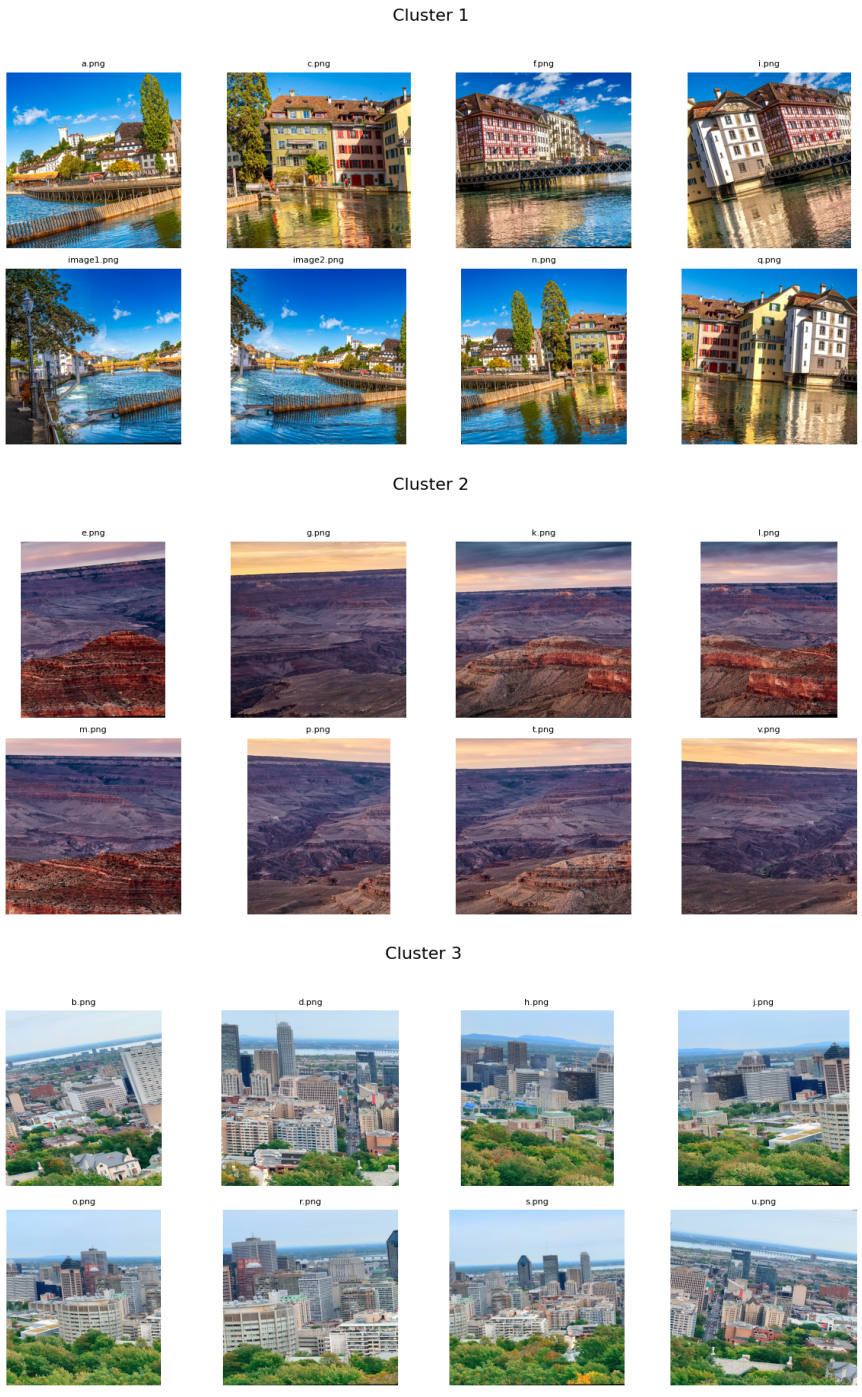

Figure 15: Clusters obtained using color histogram-based features

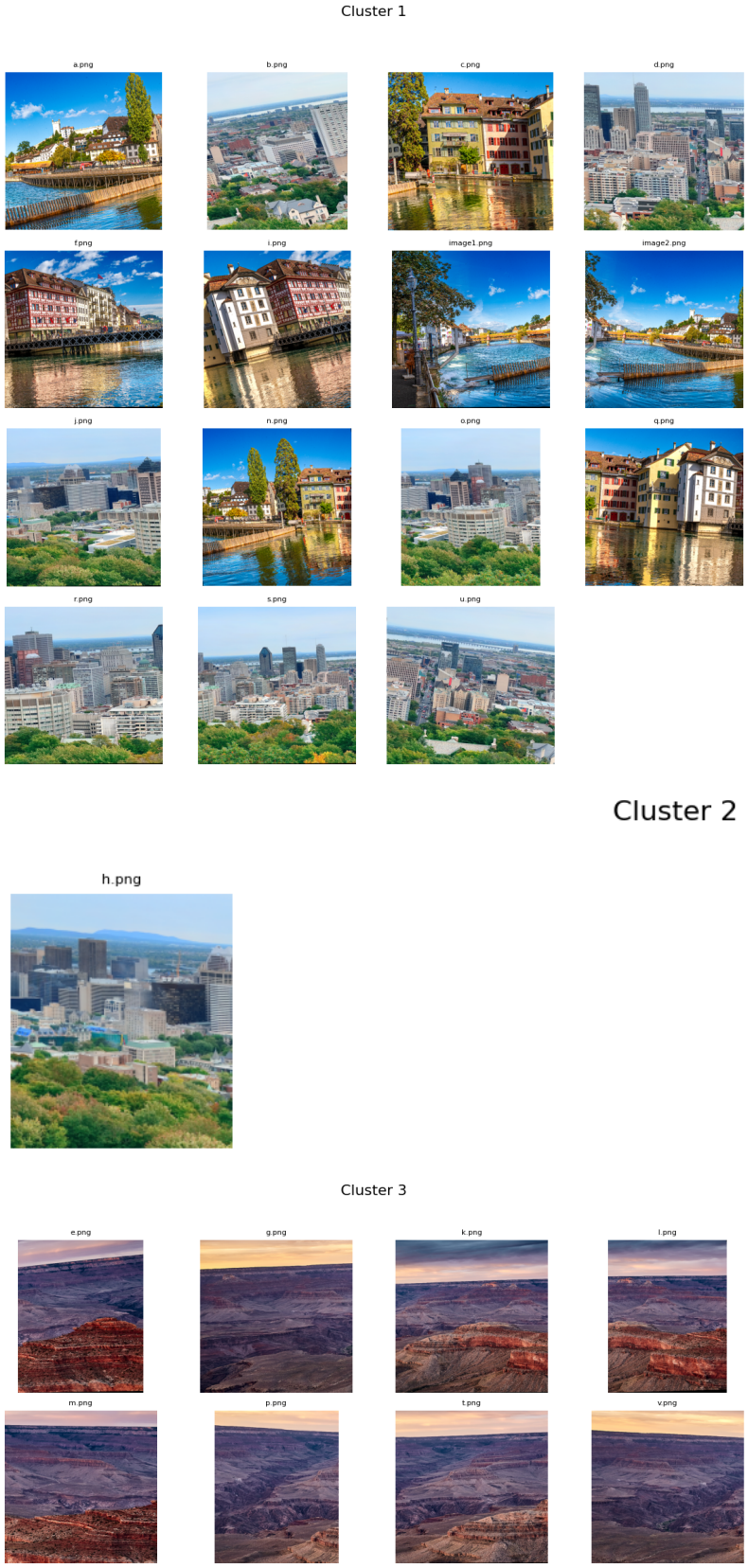

Figure 16: Clusters obtained using visual bag of words (VBoW) features

By visual inspection, the clusters generated using color histograms are more consistent and visually coherent compared to those obtained using VBoW. Therefore, **clustering based on color histograms** was selected for the multi-image stitching in `Part 6`.

Part 1. Keypoint Detection and Feature Extraction

Keypoints and descriptors were extracted from `image1.png` and `image2.png` using the **Scale-Invariant Feature Transform (SIFT)** algorithm.

Results:

- Number of keypoints in `image1.png`: **5890**
- Number of keypoints in `image2.png`: **4510**
- Descriptor shape for `image1.png`: **(5890, 128)**
- Descriptor shape for `image2.png`: **(4510, 128)**

The extracted keypoints were overlaid on the original images to visualize their spatial distribution and verify correctness. As shown in the figure below, the keypoints are densely and evenly distributed across salient regions such as corners, edges, and textured areas, confirming that the SIFT algorithm has successfully detected meaningful and distinctive features.

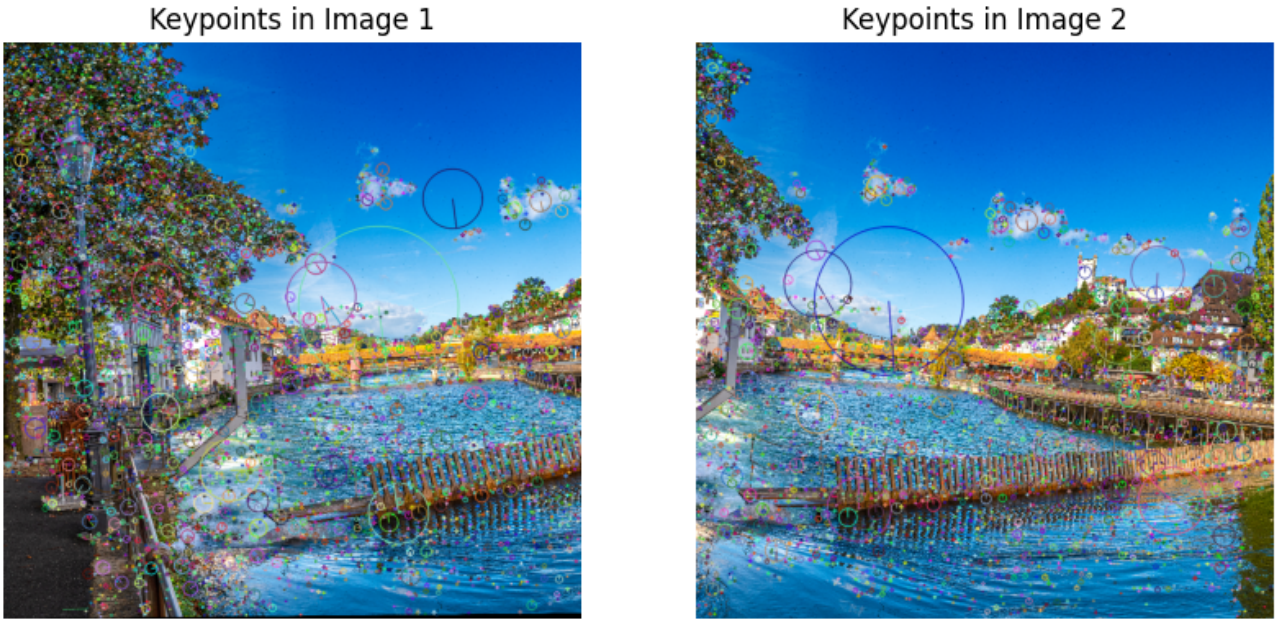

Figure 17: SIFT keypoints overlaid on `image1.png` and `image2.png`

Part 2. Feature Matching

Features extracted using SIFT were matched between the image pair using two different algorithms: the `BruteForce` matcher and the `FLANN-based` matcher. Matches were filtered using **Lowe's ratio test** to ensure that only high-quality correspondences were retained.

The resulting matches were visualized by drawing lines between corresponding keypoints in both images. As shown in the figures below, both algorithms successfully identified several correct correspondences across the images.

Figure 18: Feature matches obtained using the BruteForce matcher

Figure 19: Feature matches obtained using the FLANN-based matcher

Part 3. Homography Estimation

With the set of good feature matches, I estimated the homography matrix using RANSAC. This robust method computes the transformation between the overlapping images while rejecting outliers. The computed homography matrix was saved as `2022482_homography.csv` for further use in the image stitching pipeline.

The estimated homography matrix M is given by:

$$M = \begin{bmatrix} 0.8990 & -0.1314 & -164.4315 \\ 0.1432 & 0.9905 & -69.4275 \\ -1.52 \times 10^{-7} & 1.72 \times 10^{-7} & 1.0000 \end{bmatrix}$$

Part 4. Perspective Warping

Using the computed homography matrix, I performed perspective warping to align the views of the two overlapping images, required in panorama creation.

Two types of warping were performed:

- **Image1 to Image2:** `image1` was warped using the homography matrix M to align its perspective with `image2`. The warped result is shown side-by-side with the original `image2` in the image below.
- **Image2 to Image1:** `image2` was warped using the inverse of the homography matrix M^{-1} to align with `image1`. This output is shown side-by-side with the original `image1`.

Figure 20: Warped `image1` aligned with `image2` using the homography matrix

Figure 21: Warped `image2` aligned with `image1` using the inverse homography matrix

Part 5. Stitching

To generate the panorama, two versions were created:

- **Without Blending/Cropping:** `image1` (warped) and `image2` were stitched together onto a dynamically computed canvas, preserving all black regions and showing the raw overlay.
- **With Blending/Cropping:** A simple blending technique (masking) was applied to merge overlapping regions smoothly, followed by cropping to remove excess black borders, resulting in a polished panorama.

Figure 22: Panorama generation - left: stitched image without blending/cropping, right: final result after blending/cropping

Part 6. Multi-Stitching

For the final task, **multi-stitching** was performed on each of the three image clusters obtained during the clustering stage. A helper function was implemented to sequentially stitch two images at a time using the pipeline developed earlier—`feature matching`, `homography estimation`, and `perspective warping`. The goal was to create one panorama for each cluster, resulting in three final stitched scenes.

To determine the optimal stitching sequence within each cluster, a custom strategy was employed. This involved computing the best image pairs to stitch first based on two different metrics:

- **Inliers Count:** After computing the homography using RANSAC, the number of inliers (i.e., feature matches that conform to the estimated transformation) was used as an indicator of alignment quality. A higher number of inliers generally implies better geometric consistency between images.
- **Reprojection Error:** This measures how accurately the transformed points match the actual corresponding points in the second image. Lower reprojection error indicates a more precise transformation.

Based on these two metrics, two sets of panoramas were generated for each cluster. The resulting images are shown below.

Figure 23: Panoramas generated using inliers as the stitching metric

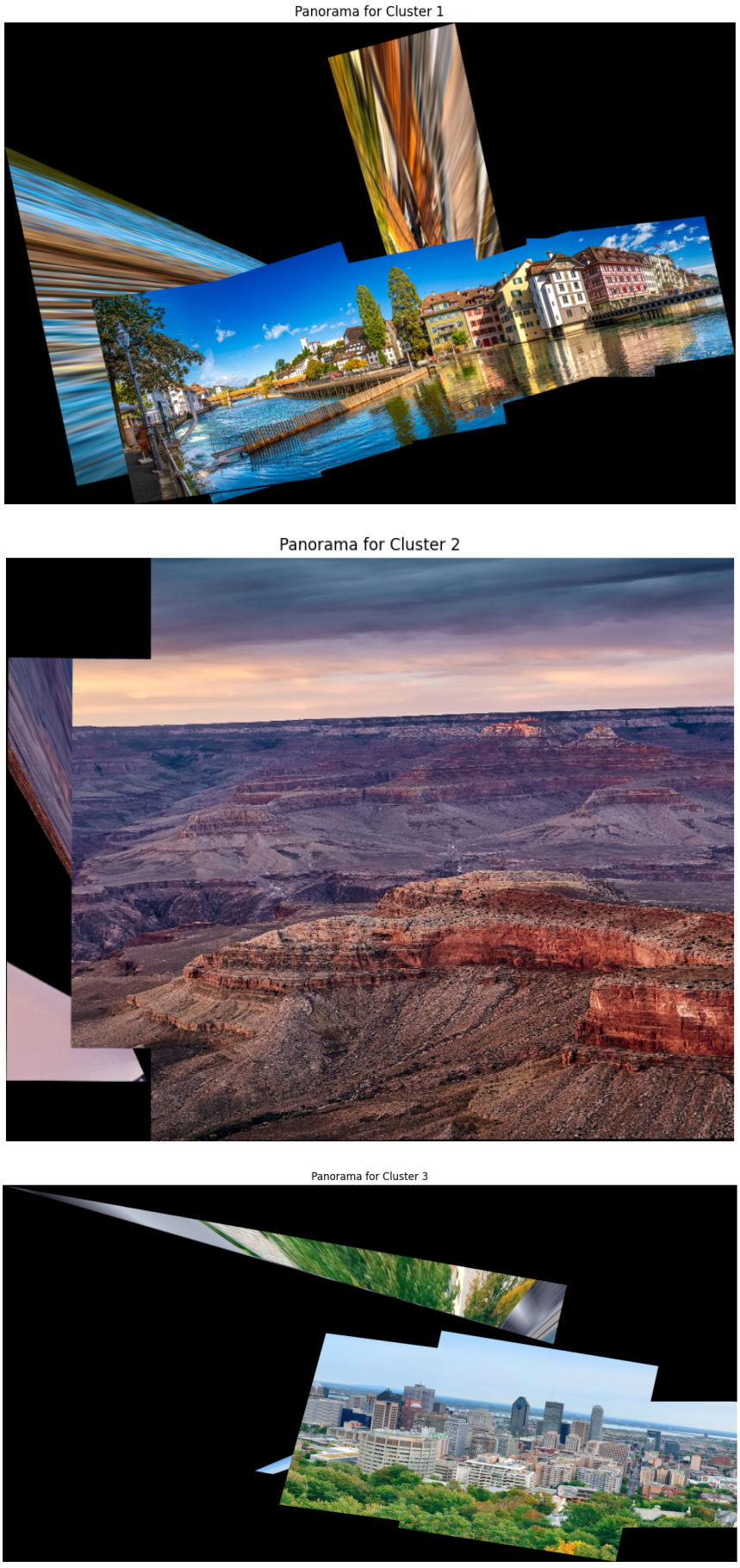

Figure 24: Panoramas generated using reprojection error as the stitching metric

Upon **visual inspection**, the panoramas generated using `inliers-based` selection showed better alignment and fewer artifacts compared to the reprojection error-based ones. Hence, the panoramas from the inliers method were further refined by cropping excess black borders to produce the final panoramas:

Figure 25: Final stitched panoramas for each cluster after cropping

References

- **Color Histograms for images using OpenCV:** [PyImageSearch - Image Histograms]
- **SIFT for Keypoints and Descriptors:** [OpenCV SIFT Tutorial]
- **Feature Matching with Brute-Force and FLANN:** [OpenCV Feature Matching Tutorial]
- **Homography Matrix Estimation using RANSAC:** [OpenCV Homography Tutorial]
- **Image Alignment using Homography (Perspective Warping):** [LearnOpenCV - Image Alignment]
- **Image Stitching for panorama creation with OpenCV:** [PyImageSearch - Image Stitching Tutorial]

Question 5: Point Cloud Registration (BONUS)

This task focuses on estimating the trajectory of a **TurtleBot** equipped with a **3D LiDAR** sensor by registering sequentially captured point cloud data. I applied **point-to-point ICP (Iterative Closest Point) and RANSAC-based techniques** using the Open3D library to align point clouds and reconstruct the robot's motion and global map.

Part 1. ICP on Two Consecutive Point Clouds

I selected two consecutive point clouds, `pointcloud_0000.pcd` and `pointcloud_0004.pcd`, to perform point-to-point ICP registration using `Open3D`. An initial transformation guess $\mathbf{T}_{\text{init}}$ was generated using a random orthonormal rotation matrix and a random translation vector.

The initial transformation matrix $\mathbf{T}_{\text{init}} \in \mathbb{R}^{4\times4}$ was:

$$\mathbf{T}_{\text{init}} = \begin{bmatrix} 0.3637 & 0.1365 & 0.9215 & 0.4809 \\ 0.2085 & 0.9522 & -0.2234 & 0.0505 \\ -0.9079 & 0.2733 & 0.3178 & -0.6830 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

To ensure that the rotation matrix was valid, the following checks were performed:

- **Orthogonality check:** Verified that $R^T R = I$ within a tolerance of 10^{-6}.
- **Determinant check:** Confirmed that $\det(R) \approx 1$, ensuring a proper rotation (no reflection).
- **Inverse check:** Confirmed that $R^{-1} \approx R^T$.

All three checks passed successfully, ensuring that the initial guess was a valid transformation matrix.

Initial alignment using $\mathbf{T}_{\text{init}}$ resulted in:

- **Fitness:** 0.00218
- **Inlier RMSE:** 0.01450

After running point-to-point ICP, the algorithm refined the alignment and produced the estimated transformation matrix $\mathbf{T}_{\text{ICP}}$:

$$\mathbf{T}_{\text{ICP}} = \begin{bmatrix} 0.3423 & 0.1430 & 0.9287 & 0.4640 \\ 0.2126 & 0.9509 & -0.2248 & 0.0577 \\ -0.9152 & 0.2743 & 0.2951 & -0.6826 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Post-ICP results:

- **Fitness:** 0.00244
- **Inlier RMSE:** 0.01345

ICP successfully reduced the inlier RMSE and slightly increased fitness. Notably, the initial transformation guess $\mathbf{T}_{\text{init}}$ and the ICP-estimated transformation $\mathbf{T}_{\text{ICP}}$ are *not identical*, demonstrating that the estimated transformation is indeed a learnt correction over the original guess. This confirms that the ICP process refined the pose to better align the two point clouds.

Part 2. Hyperparameter Tuning

To improve the performance of the ICP registration, I experimented with various hyperparameter configurations. Specifically, I varied the distance threshold values and the initialization method used for the transformation matrix. The two initial guess strategies used were:

- **Random orthogonal matrix**
- **RANSAC-based initial guess**

The goal was to optimize ICP alignment performance in terms of **fitness**, **inlier RMSE**, and **transformation error**. The following table summarizes the initial and final performance metrics across all experiments:

Method	Threshold	Init Fitness	Init RMSE	Final Fitness	Final RMSE	Transf. Error
Random Orthogonal	0.01	0.000393	0.00741	0.000655	0.00732	0.01628
Random Orthogonal	0.02	0.002182	0.01450	0.002444	0.01345	0.03847
Random Orthogonal	0.05	0.013659	0.03592	0.016234	0.03427	0.07760
Random Orthogonal	0.10	0.045254	0.06766	0.055291	0.06555	0.15205
Random Orthogonal	0.50	0.327209	0.28394	0.784552	0.21944	1.86076
RANSAC	0.01	0.286232	0.00738	0.619201	0.00547	0.02030
RANSAC	0.02	0.718263	0.01241	0.887279	0.00914	0.02035
RANSAC	0.05	0.984333	0.01790	0.986166	0.01247	0.02057
RANSAC	0.10	0.997687	0.01940	0.997774	0.01442	0.02051
RANSAC	**0.50**	**0.999956**	**0.02075**	**0.999956**	**0.01614**	**0.02049**

Best configuration: RANSAC-based initial guess with a threshold of **0.5**. *This setup achieved the highest final fitness (0.9999), low inlier RMSE (0.0161), and minimal transformation error (0.0205), outperforming all other combinations*

Estimated transformation matrix from best configuration:

$$\mathbf{T}_{\text{best}} = \begin{bmatrix} 1.000 & -5.930 \times 10^{-4} & -1.300 \times 10^{-5} & -1.680 \times 10^{-4} \\ 5.930 \times 10^{-4} & 1.000 & -7.000 \times 10^{-6} & -1.018 \times 10^{-3} \\ 1.300 \times 10^{-5} & 7.000 \times 10^{-6} & 1.000 & 3.400 \times 10^{-5} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Part 3. Transformation and Visualization

Using the best-performing configuration — **RANSAC-based initialization** with a **threshold of 0.5**. I applied the estimated transformation matrix to the source point cloud `pointcloud_0000.pcd`. The transformed source was then visualized along with the target point cloud `pointcloud_0004.pcd`.

The resulting visualization showed a strong alignment and significant overlap between the transformed source and the target point clouds, confirming accurate registration.

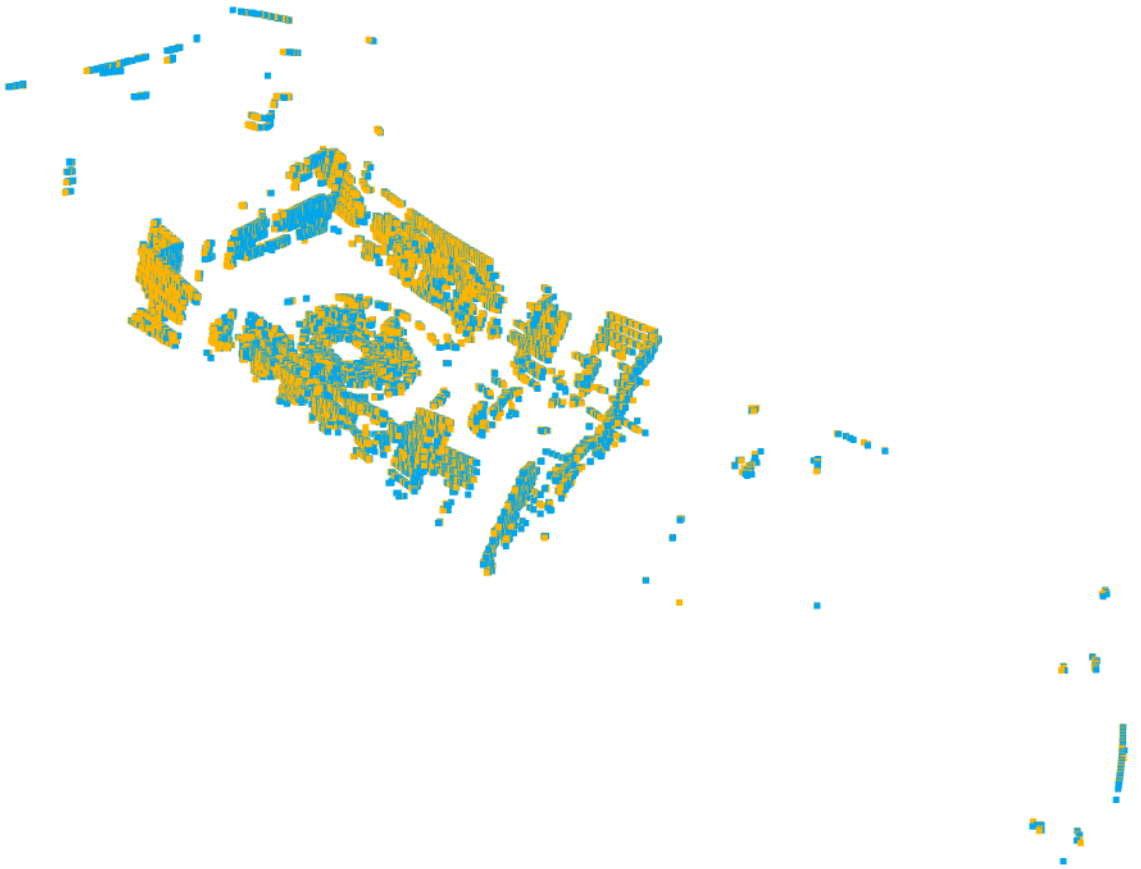

Figure 26: Visualization of aligned point clouds after applying the estimated transformation to the source

Reasoning: The successful alignment is due to:

- The high-quality initial guess provided by the RANSAC-based method, which brought the source close to the target before fine-tuning.
- The relatively high threshold (0.5), which allowed more correspondences to be considered during optimization, improving robustness in noisy or sparse areas.
- The transformation matrix itself had minimal deviation from identity, indicating the original frames were already well-aligned and only needed minor refinement.

This demonstrates that with proper initialization and hyperparameter tuning, the ICP algorithm is capable of achieving near-perfect point cloud registration.

Part 4. Global Registration and Trajectory Estimation

To construct a globally registered map, I extended the ICP-based registration process across all 100 sequential point clouds captured by the TurtleBot (from `pointcloud_0000.pcd` to `pointcloud_0396.pcd` in steps of 4). The registration pipeline is as follows:

- **Initial Pair (0000 → 0004)**: Used RANSAC-based initial guess followed by ICP refinement, as this consistently yielded the best alignment performance.
- **Subsequent Pairs**: For all remaining consecutive point clouds, the ICP algorithm was applied using the previously computed transformation as the initial estimate.
- **Global Transformation Accumulation**: Each transformation was composed with the previous ones to transform every point cloud into a common reference frame.
- **Point Cloud Merging**: The transformed point clouds were merged into a single global point cloud to obtain a complete 3D reconstruction.
- **Trajectory Estimation**: The translation components of each transformation matrix were extracted to compute the estimated 3D trajectory of the TurtleBot.

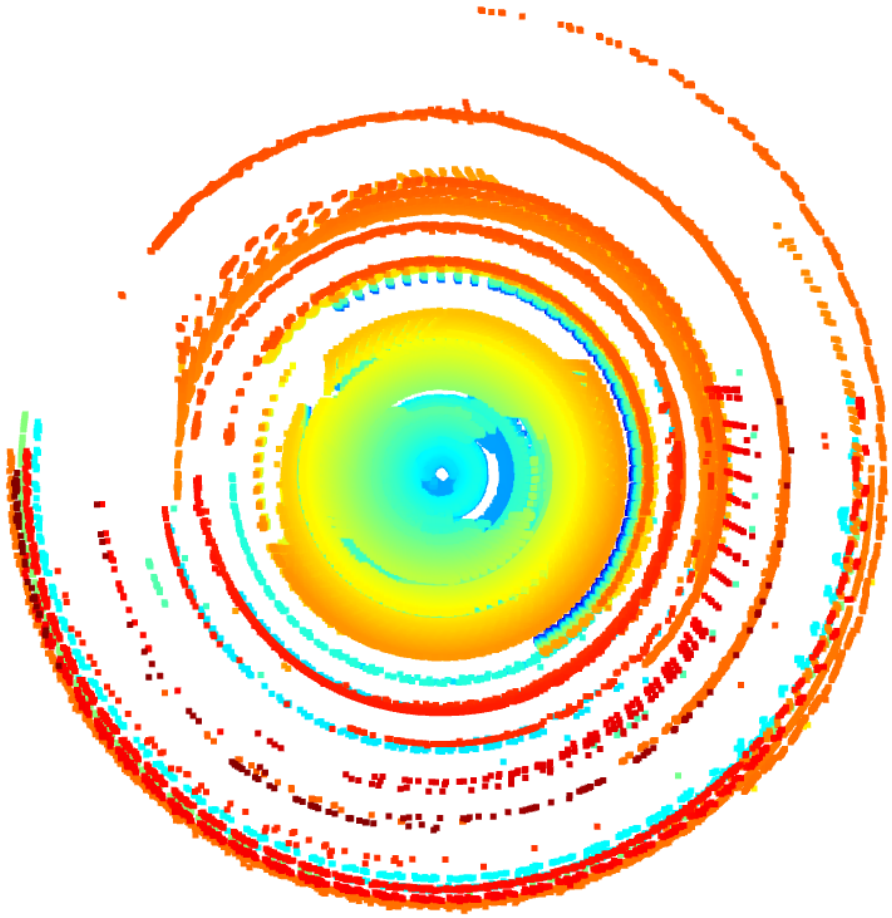

Figure 27: Global registered point cloud constructed from all 100 sequential point clouds

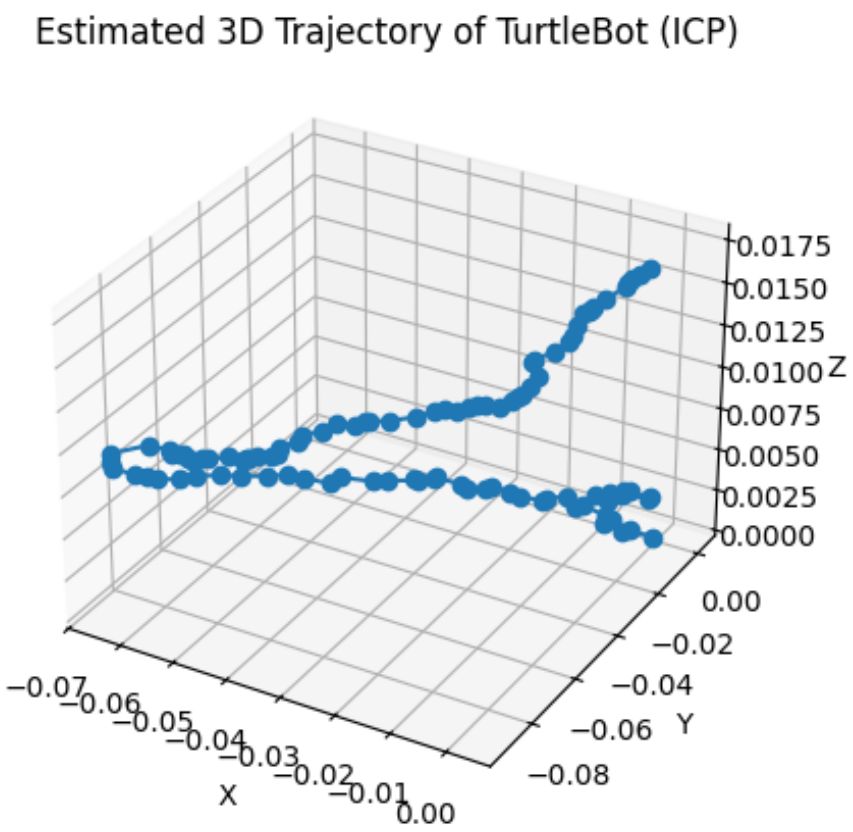

Figure 28: Estimated 3D trajectory of the TurtleBot extracted from transformation matrices

The final 3D trajectory shows a smooth, continuous path, which is consistent with realistic robot motion. The successful global registration and trajectory estimation validate the robustness of the ICP pipeline with proper initialization and hyperparameter tuning.

The trajectory data was saved in a CSV file named `2022482_icp.csv`, containing the x, y, z positions for each step of the TurtleBot's motion.

References

- **Point-to-Point ICP Registration Algorithm:** [Open3D ICP Registration Tutorial]
- **Orthogonal Rotation Matrix using `scipy.stats.ortho_group`:** [SciPy Ortho Group Documentation]
- **RANSAC-Based Transformation Matrix Estimation:** [Open3D Global Registration Tutorial]
- **Point Cloud Geometry and Global Registration:** [Open3D PointCloud Geometry Tutorial]