AutoCalib

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

***Abstract*—This report focuses on the estimation of a camera's intrinsic and extrinsic parameters through the process of camera calibration. Specifically, we implement an automatic calibration method based on the approach proposed by Zhengyou Zhang [1]**

I. Introduction

The camera projects real-world points onto its image plane using perspective projection, which is governed by its intrinsic parameters. The camera's position and orientation in the world coordinate frame are defined by its extrinsic parameters. Camera calibration is essential for accurately mapping 3D points to 2D coordinates, a crucial step in reconstructing 3D scenes using images only.

II. Calibration Images

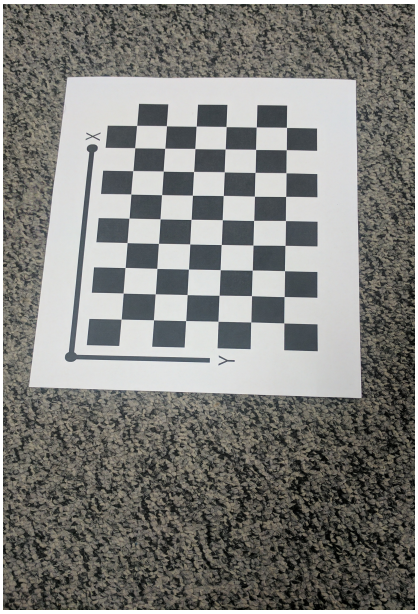

Fig. 1. Calibration target

Firstly, to calibrate the camera, we need **3D real-world coordinates** and their matching **2D image coordinates**. As seen in Figure 1, we use a **checkerboard** for this process.

We captured **13 images** of the checkerboard from different angles while keeping the camera focus fixed. From these images, we detected the **checkerboard corners**, which serve as our **2D image coordinates**.

Since the size of the checkerboard is known—each square measures **21.5 mm**—we can determine the **3D world coordinates** from its dimensions.

III. Initial Parameter Estimation

The relation between the real world coordinates and it's corresponding 2D image plane is given using:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Since the **checkerboard lies flat** on a surface, it exists only in the **XY plane**, meaning the **Z-coordinate is zero** ($Z = 0$).

- X, Y, Z are the **3D real-world coordinates**.
- u, v are the **2D image coordinates**.
- s is a **scaling factor**.
- r_i represents the **rotation and translation vectors**.

To estimate these parameters, we use the **detected corners** from the checkerboard images. The checkerboard has a **6×9 grid pattern**, giving us a total of **54 corners**. Figure 2 shows the detected corners using `cv2.findChessboardCorners`. Additionally, we need the **real-world 3D coordinates**, as shown in Figure 3.

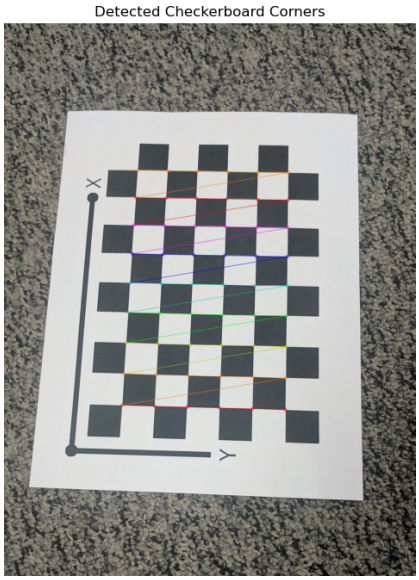

Fig. 2. Example of detected Checkerboard Corners

```
[[129.    21.5    0. ]
 [107.5   21.5    0. ]
 [ 86.    21.5    0. ]
 [ 64.5   21.5    0. ]
 [ 43.    21.5    0. ]
 [ 21.5   21.5    0. ]
 [129.    43.     0. ]
 [107.5   43.     0. ]
 [ 86.    43.     0. ]
 [ 64.5   43.     0. ]
 [ 43.    43.     0. ]
 [ 21.5   43.     0. ]
 [129.    64.5    0. ]
 [107.5   64.5    0. ]
 [ 86.    64.5    0. ]
 [ 64.5   64.5    0. ]
 [ 43.    64.5    0. ]
 [ 21.5   64.5    0. ]
 [129.    86.     0. ]
 [107.5   86.     0. ]
```

Fig. 3. Example of real-world coordinates

A. Intrinsic Paramters

For Intrisic Parameters we need to estimate camera calibration matrix, K.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

- f_x - Focal length along the x-axis
- f_y - Focal length along the y-axis
- c_x - Principal point x-coordinate (image center)
- c_y - Principal point y-coordinate (image center)

First, we calculate the **homography matrix** H for each image separately using the **Direct Linear Transform (DLT)** method. The homography matrix H establishes a transformation between **3D world points** and their corresponding **2D image points**.

To compute the **intrinsic matrix** K, we use a **closed-form solution** based on the **B matrix**.

$$B = A^{-T}A^{-1} \equiv \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{12} & B_{22} & B_{23} \\ B_{13} & B_{23} & B_{33} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\alpha^2} - \frac{\gamma}{\alpha^2\beta} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} \\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2} + \frac{1}{\beta^2} - \frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} & -\frac{v_0}{\beta^2} \\ \frac{v_0\gamma - u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma - u_0\beta)}{\alpha^2\beta^2} - \frac{v_0}{\beta^2} & \frac{(v_0\gamma - u_0\beta)^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1 \end{bmatrix}$$

Here, B is a symmetric matrix that has only 6 unknowns, therefore:

$$b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$$

We need to solve for b. Using the homography for each of the image we will create a v vector and stack each of this vector to create a V matrix.

$$v_{ij} = \begin{bmatrix} h_{i1}h_{j1} \\ h_{i1}h_{j2} + h_{i2}h_{j1} \\ h_{i2}h_{j2} \\ h_{i3}h_{j1} + h_{i1}h_{j3} \\ h_{i3}h_{j2} + h_{i2}h_{j3} \\ h_{i3}h_{j3} \end{bmatrix}$$

Here V has a dimension of 2nx6. Now to get a unique solution we need at least n = 2, but for this we consider that image sensor alignment is correct meaning the skewness is negligible, i.e., [0, 1, 0, 0, 0, 0]b = 0. If we want a general unique solution we need n¿=3.

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0$$

stacking the vij it can be rewritten as:

$$Vb = 0$$

Here we know the V matrix using Singular Value Decomposition (SVD), we get b. Using b we can construct K.

B. Extrinsic Paramters

Since, now we know scaling factor s, K and individual homography H for each image.

$$s = \frac{1}{\mathbf{A}^{-1}\mathbf{h}_1}$$

The equations for computing the r1, r2, r3 and t are:

$$\mathbf{r}_1 = s\mathbf{A}^{-1}\mathbf{h}_1$$

$$\mathbf{r}_2 = s\mathbf{A}^{-1}\mathbf{h}_2$$

$$\mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$

$$\mathbf{t} = s\mathbf{A}^{-1}\mathbf{h}_3$$

For start we considering that there is zero distortion. Hence, k1 = k2 = 0.

IV. Radial Distortion

So far, we have ignored **radial distortion**. However, in reality, most cameras have some amount of distortion. In this case, we will consider only the first two terms of radial distortion.

$$r^2 = x^2 + y^2$$

$$\text{distortion} = 1 + k_1 r^2 + k_2 r^4$$

V. Projection Error

Here, we calculate the error between the original coordinates and the re-projected coordinates obtained using the method described above.

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\left\|k_{m_{ij}} - \hat{m}(K, k_1, k_2, R_i, t_i, M_j)\right\|^2$$

where:

- $k_{m_{ij}}$ - Observed 2D image coordinates of the j-th point in the i-th image.
- $\hat{m}(\cdot)$ - Projected 2D coordinates using the camera model.
- K - Intrinsic camera matrix.
- k_1, k_2 - Radial distortion coefficients.
- R_i, t_i - Rotation and translation (extrinsic parameters) for the i-th image.
- M_j - 3D world coordinates of the j-th point.

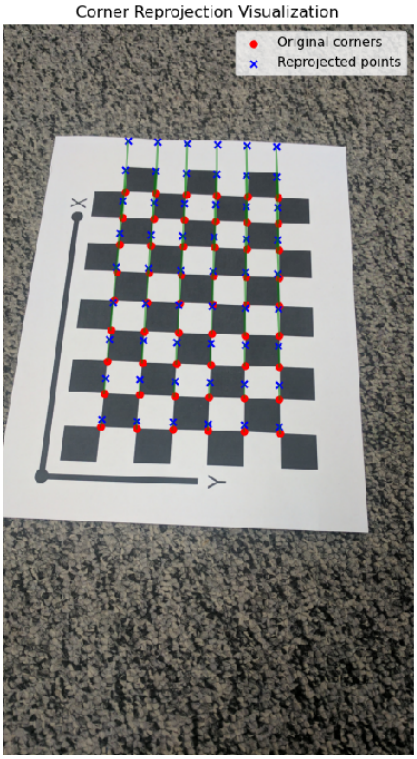

Fig. 4. Checkerboard Image 1 Without Optimization

VI. Error Optimization

As seen in Figure 4, the projected coordinates do not perfectly align with the original coordinates. This means there is an error that needs to be reduced. To fix this, we optimize the parameters using a non-linear iterative method. Specifically, we use the Levenberg-Marquardt algorithm. We implement this optimization with `scipy.optimize.least_squares`, which minimizes the difference between the projected and actual points.

$$\sum_{i=1}^{N}\sum_{j=1}^{M}\|m_{ij} - \hat{m}(K, k_1, k_2, R_i, t_i, M_j)\|$$

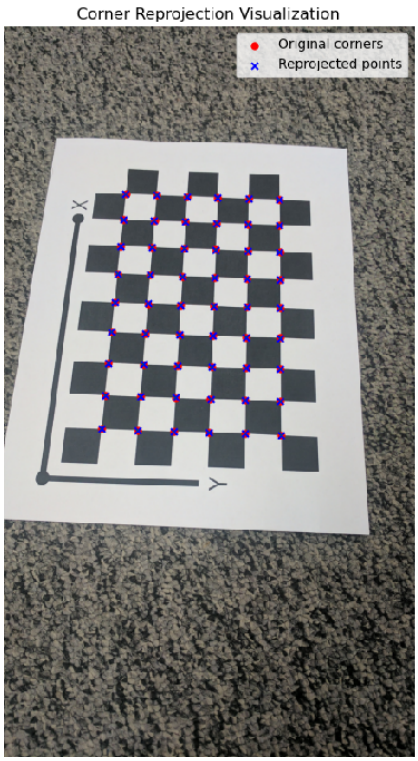

Fig. 5. Checkerboard Image 1 After Optimization

VII. Results

$$K = \begin{bmatrix} 2241.35 & 0.00 & 758.27 \\ 0 & 2226.99 & 0.6185 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{k} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$K_{\text{Opt}} = \begin{bmatrix} 2045.84 & -1.641 & 759.42 \\ 0 & 2037.55 & 1345.26 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{k}_{\text{opt}} = \begin{bmatrix} 0.171 \\ -0.736 \end{bmatrix}$$

TABLE I
REPROJECTION ERROR BEFORE AND AFTER OPTIMIZATION

Image	Reprojection Error	
	Before Optimization	**After Optimization**
0	18.0019	0.7846
1	115.9079	0.7153
2	71.1192	0.5646
3	81.2501	0.5569
4	113.0678	0.8144
5	86.9725	0.5221
6	107.9300	0.8224
7	64.1707	0.6264
8	2.9791	0.5102
9	104.7782	0.7362
10	133.4759	1.0199
11	18.9528	0.6075
12	130.4827	0.8996
Total Error	**80.6991**	**0.7062**

REFERENCES

[1] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000.

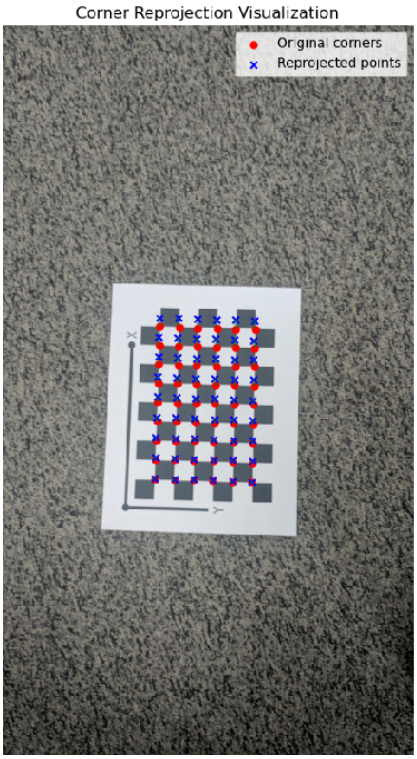

Fig. 6. Checkerboard Image 2 Before Optimization

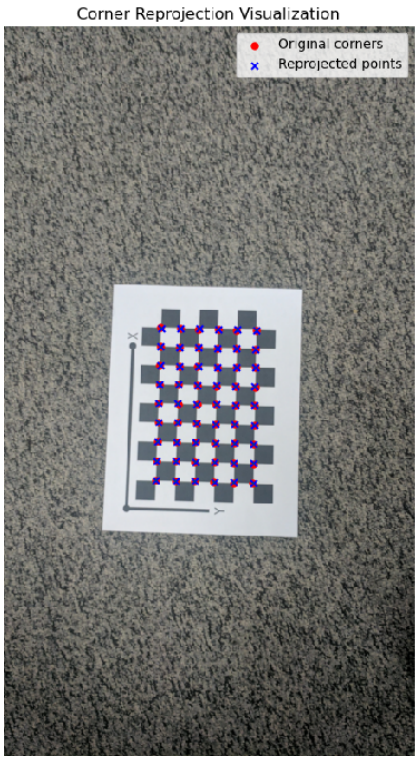

Fig. 7. Checkerboard Image 2 After Optimization

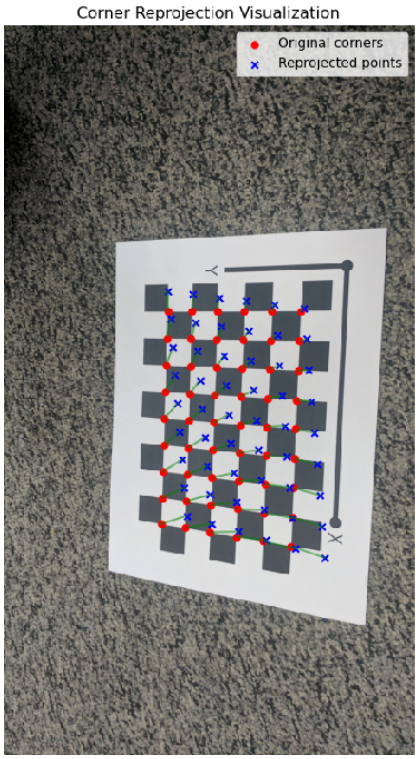

Fig. 8. Checkerboard Image 3 Before Optimization

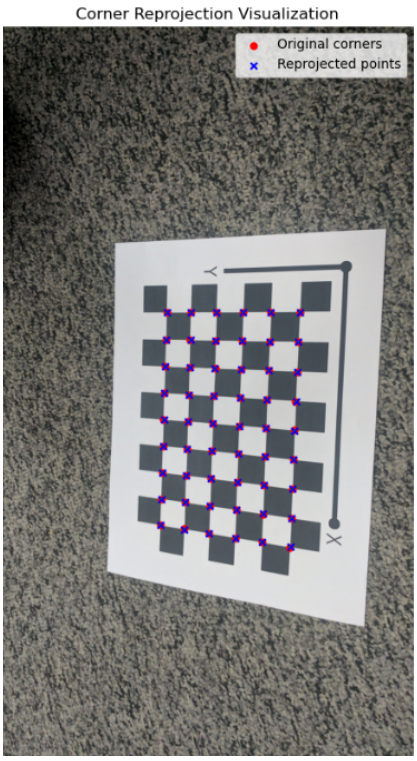

Fig. 9. Checkerboard Image 3 After Optimization

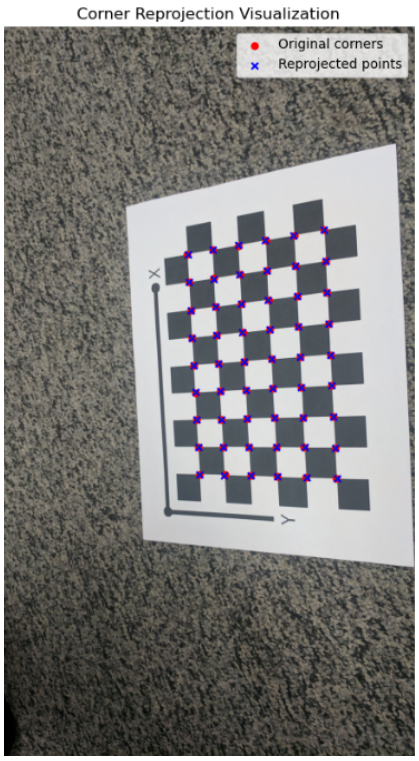

Fig. 11. Checkerboard Image 4 After Optimization

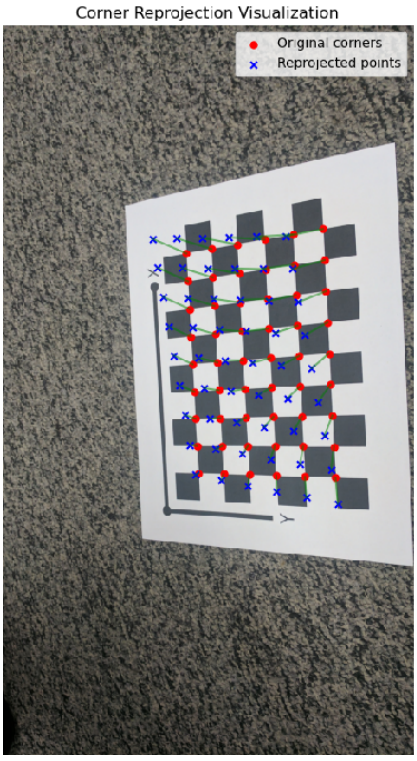

Fig. 10. Checkerboard Image 4 Before Optimization

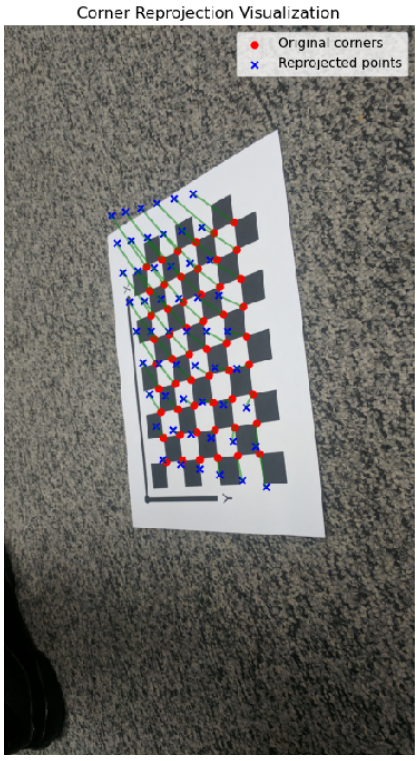

Fig. 12. Checkerboard Image 5 Before Optimization

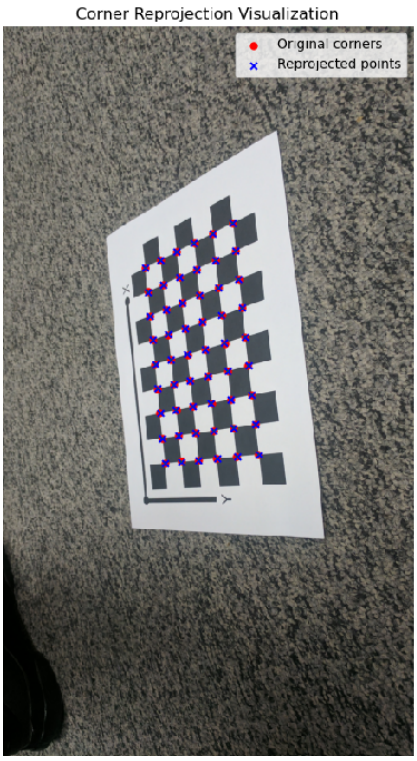

Fig. 13. Checkerboard Image 5 After Optimization

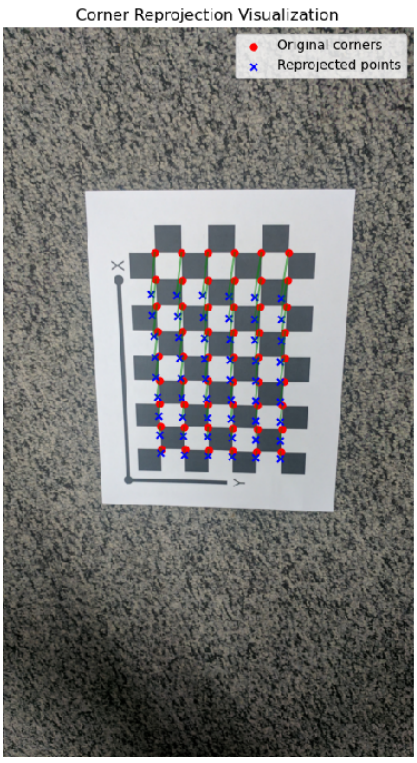

Fig. 14. Checkerboard Image 6 Before Optimization

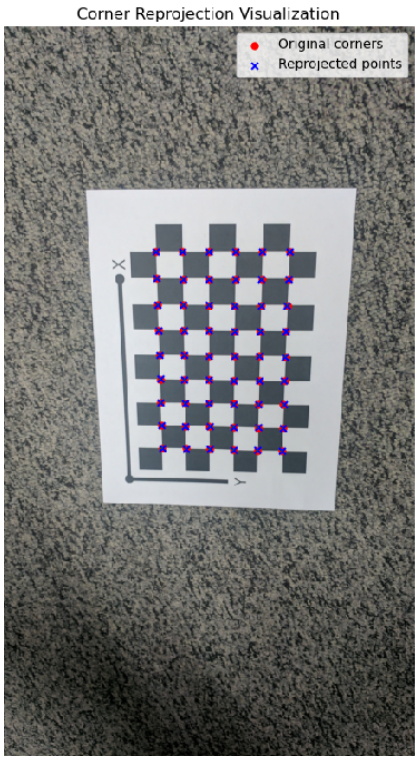

Fig. 15. Checkerboard Image 6 After Optimization

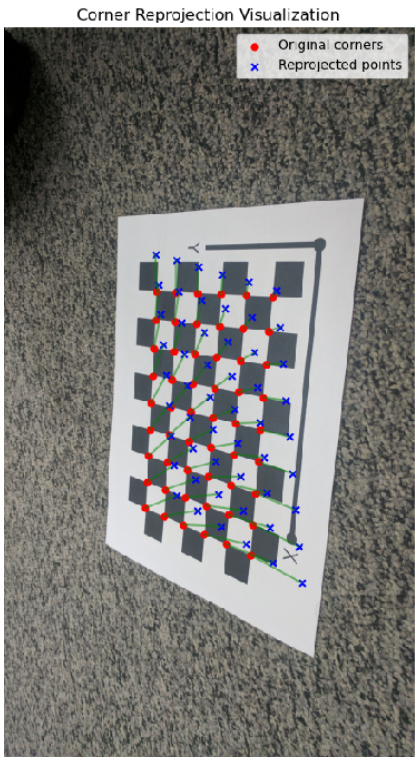

Fig. 16. Checkerboard Image 7 Before Optimization

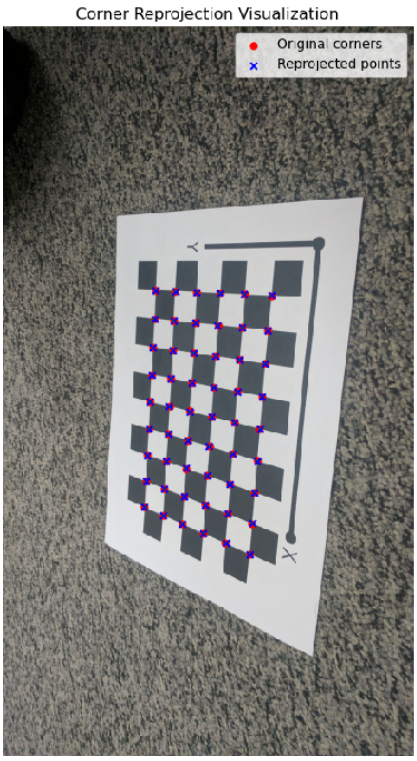

Fig. 17. Checkerboard Image 7 After Optimization

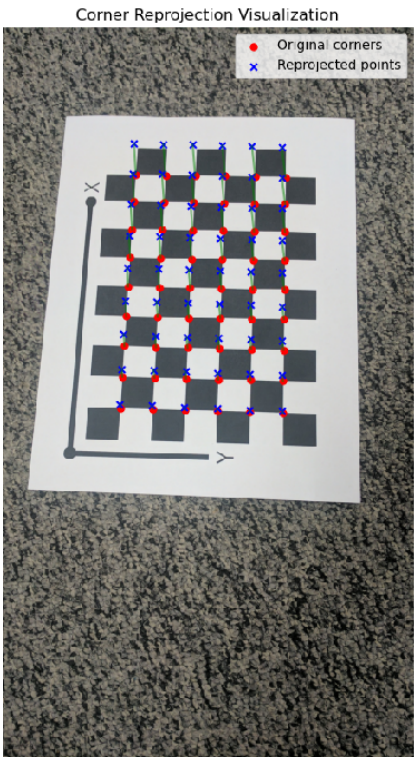

Fig. 18. Checkerboard Image 8 Before Optimization

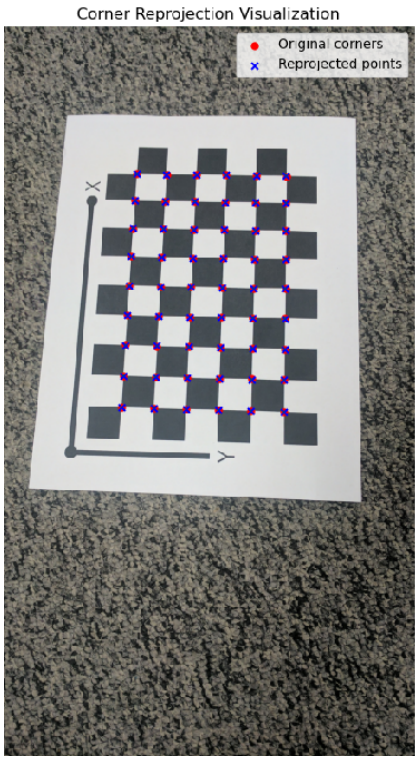

Fig. 19. Checkerboard Image 8 After Optimization

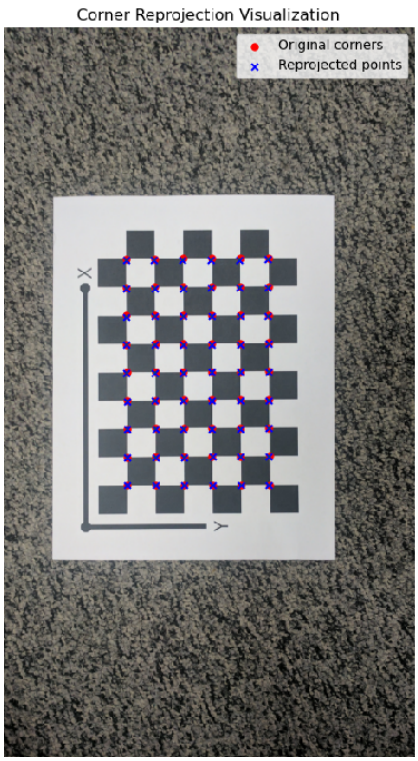

Fig. 20. Checkerboard Image 9 Before Optimization

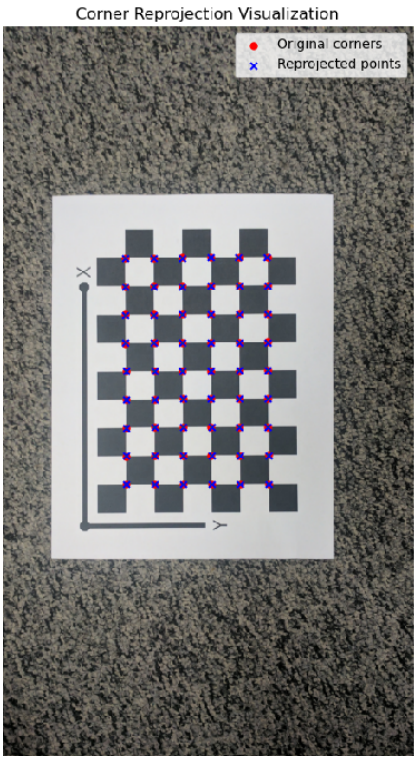

Fig. 21. Checkerboard Image 9 After Optimization

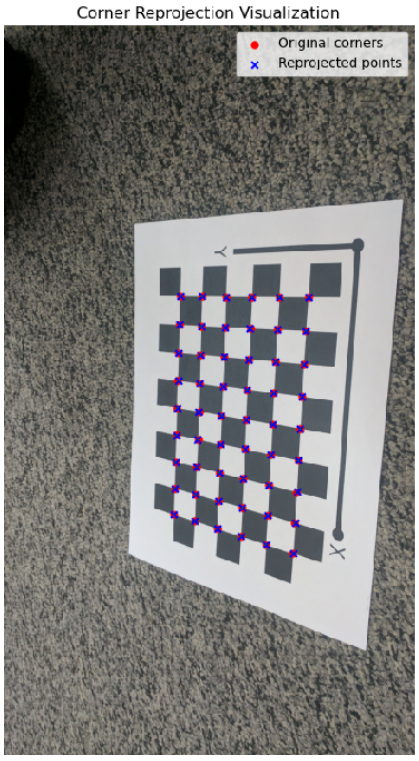

Fig. 23. Checkerboard Image 10 After Optimization

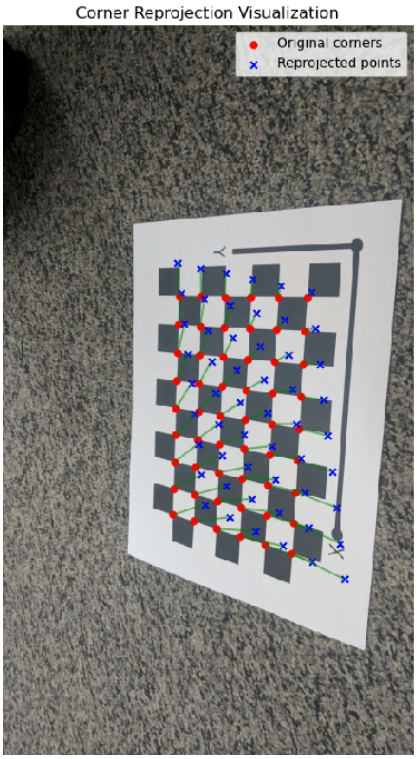

Fig. 22. Checkerboard Image 10 Before Optimization

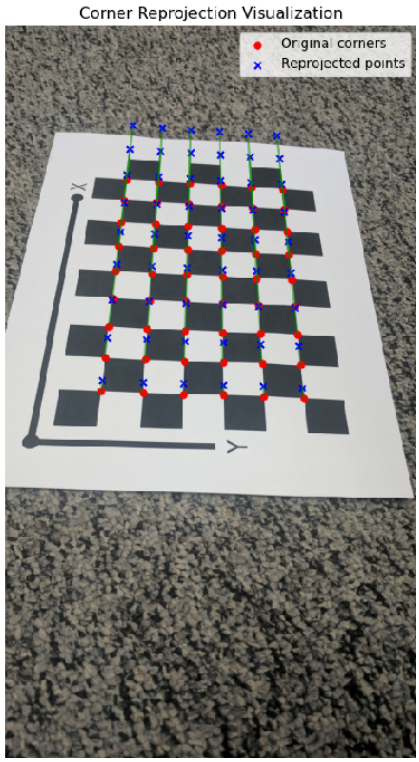

Fig. 24. Checkerboard Image 11 Before Optimization

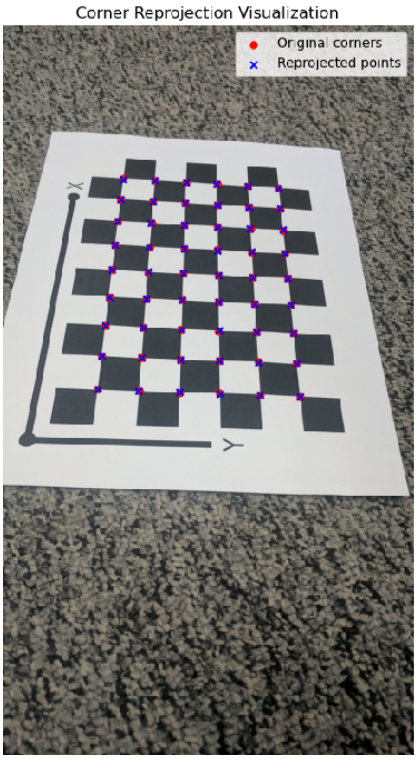

Fig. 25. Checkerboard Image 11 After Optimization

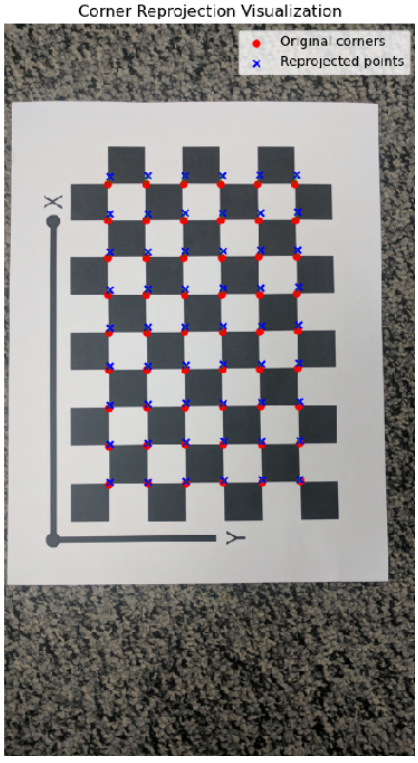

Fig. 26. Checkerboard Image 12 Before Optimization

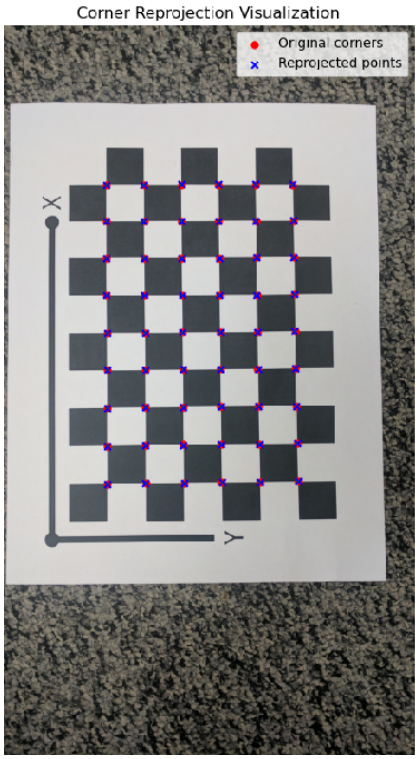

Fig. 27. Checkerboard Image 12 After Optimization

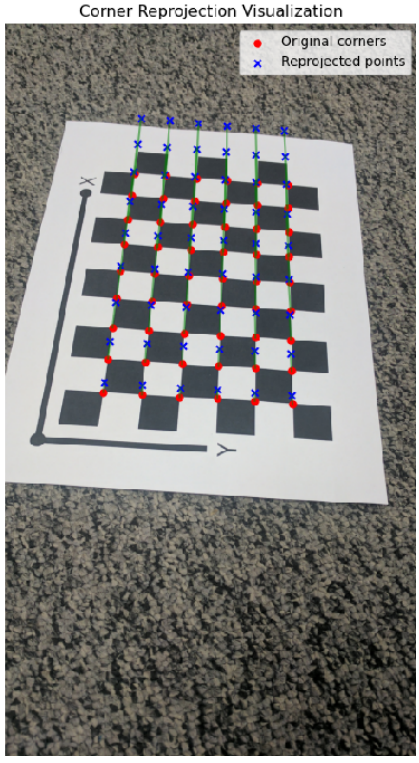

Fig. 28. Checkerboard Image 13 Before Optimization

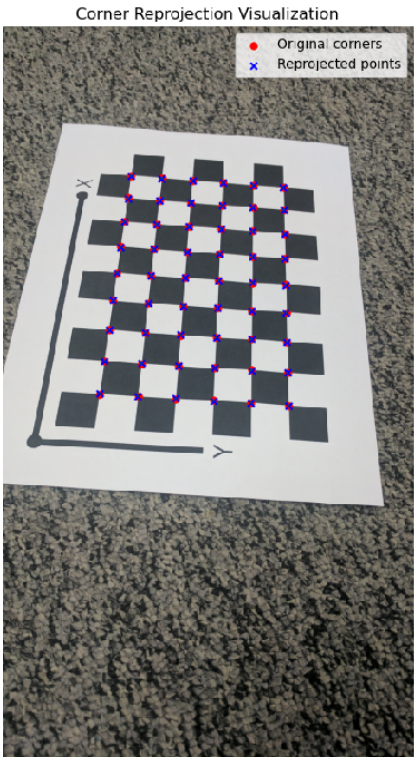

Fig. 29. Checkerboard Image 13 After Optimization