HW1: AutoCalib

RBE/CS-549: Computer Vision

Manav Bichu
Master's in Robotics Engineering
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: mbichu@wpi.edu
Using 1 Late Day

***Abstract*—This work implements Zhengyou Zhang's flexible camera calibration method, which estimates intrinsic and extrinsic parameters of a camera using various images of planar checkerboard taken from different angles. The process involves computing homographies, extracting camera parameters, and refining them through nonlinear optimization to minimize the reprojection error. The implementation also accounts for radial distortion and evaluates calibration accuracy by comparing errors before and after optimization. The results demonstrate reliable parameter estimation, making this approach practical for real-world applications.**

I. Introduction

Camera calibration is a fundamental step in computer vision, allowing us to determine a camera's intrinsic and extrinsic parameters to correct for lens distortions and accurately project 3D world points onto a 2D image plane.
The key steps for this implementation includes:

- Corner Detection
- Homography Estimation
- Intrinsic and Extrinsic Parameter Calculation
- Optimization

A. Corner Detection:

Corner detection is a crucial step in camera calibration, as it helps identify the pixel locations of known real-world reference points. I used a checkerboard pattern as the calibration target, where the intersections of black and white squares serve as feature points. The method follows these key steps -

1. Loading Calibration Images: A set of images of a printed checkerboard pattern is loaded from the given dataset. Each image is converted to grayscale to simplify feature detection.

2. Detecting Checkerboard Corners: I made use of the **cv2.findChessboardCorners()** function to locate the checkerboard corners in the grayscale images. This function detects an internal grid of intersections, excluding the outer edges of the squares. If the corners are successfully detected, they are stored for further processing.

3. Subpixel Refinement: To enhance the accuracy of detected corners, I used the **cv2.cornerSubPix()** function. This function refines the detected corners by iteratively minimizing the intensity difference in the surrounding region. A termination criterion is set based on either a maximum number of iterations or a minimum movement threshold.

4. Saving Results: The detected and refined corners are drawn onto the original images using **cv2.drawChessboardCorners()**. These annotated images are stored in a results directory for verification.

Mathematical Relationship Between World and Image Coordinates -
Once the checkerboard corners are detected in the image plane, a relationship between the real-world coordinates $\tilde{M}$ and the corresponding image coordinates $\tilde{m}$ is established. This transformation follows the *pinhole camera model*, given by:

$$s\tilde{m} = A \begin{bmatrix} R & t \end{bmatrix} \tilde{M} \tag{1}$$

where:

- s is an arbitrary *scale factor* (ensuring homogeneity).
- $\tilde{m} = (u, v, 1)^T$ is the *homogeneous image coordinate*.
- A is the *intrinsic matrix*, containing focal lengths and principal point:

$$A = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

where:

- α, β are the *scale factors* in the u and v directions.
- γ is the *skew* factor (typically 0 for most cameras).
- (u_0, v_0) is the *principal point*.
- $R \in \mathbb{R}^{3\times3}$ is the *rotation matrix*, which describes the camera's orientation in space.
- $t \in \mathbb{R}^{3\times1}$ is the *translation vector*, describing the camera's position relative to the world.
- $\tilde{M} = (X, Y, Z, 1)^T$ is the *homogeneous 3D world coordinate*.

This equation represents the *projection* of 3D points from the *world frame* onto the *image plane* through the camera.
The world coordinates $M = (X, Y, 0)$ of the checkerboard

corners lie on a flat plane ($Z = 0$), simplifying the transformation equation:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3}$$

where:

- r_1, r_2 are the first two columns of the rotation matrix R.
- The third column is removed because the *checkerboard plane is at* $Z = 0$.

This equation forms the basis of *homography estimation*, allowing us to map the *real-world checkerboard coordinates* onto the *detected 2D corner positions* in the image.

B. Homography estimation:

To establish the relationship between the 3D world coordinates and their corresponding 2D image coordinates, the concept of *homography* was utilized. Homography is a linear transformation that maps points from one plane to another. Given that the checkerboard is a flat surface, the transformation between the world and image plane can be expressed as:

$$\tilde{m} = H\tilde{M} \tag{4}$$

where:

- $\tilde{m} = (u, v, 1)^T$ represents the homogeneous 2D image coordinates.
- $\tilde{M} = (X, Y, 1)^T$ represents the homogeneous 3D world coordinates (assuming $Z = 0$ for the checkerboard).
- H is the 3×3 homography matrix that maps world points to image points.

Computing the Homography Matrix - To compute the homography matrix H, I utilized multiple correspondences between detected image corners and known world coordinates. The relationship is derived from:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{5}$$

Expanding this equation results in two linear constraints for each point correspondence:

$$Xh_1 + Yh_2 + h_3 - u(Xh_7 + Yh_8 + h_9) = 0 \tag{6}$$

$$Xh_4 + Yh_5 + h_6 - v(Xh_7 + Yh_8 + h_9) = 0 \tag{7}$$

where h_i are the elements of the homography matrix H, represented as:

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \tag{8}$$

Rearranging these equations into matrix form gives:

$$Ah = 0 \tag{9}$$

where A is a $2N \times 9$ matrix formed by stacking the constraints from N correspondences, and h is the vectorized form of H:

$$h = (h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9)^T \tag{10}$$

Estimating Homography via SVD - To solve for H, we use **Singular Value Decomposition (SVD)** on matrix A. The optimal h is the right singular vector corresponding to the smallest singular value:

$$Ah = 0 \quad \Rightarrow \quad h = \arg\min ||Ah|| \tag{11}$$

This ensures that H is estimated in a least-squares sense, minimizing the projection error.

Normalization for Robust Estimation - Before solving for H, it is good to normalize the input points by:

- Scaling and translating the image points so that the mean is at the origin and the average distance from the origin is $\sqrt{2}$.
- Applying the same transformation to the world coordinates.

After computing H, the inverse transformation is applied to recover the true homography.

Implementation in Code - In my implementation, the homography estimation is performed using **cv2.findHomography()**, which applies **Direct Linear Transform (DLT)** and RANSAC to refine the estimation. The computed homographies are then used in intrinsic and extrinsic parameter estimation.

C. Intrinsic and Extrinsic Parameter Calculation

After estimating the homography matrix H, the intrinsic and extrinsic parameters of the camera must be computed. The intrinsic parameters define the camera's internal characteristics, while the extrinsic parameters represent its position and orientation in the world.

Intrinsic Parameter Estimation - The intrinsic parameters are estimated using the B matrix, which is computed as:

$$B = A^{-T}A^{-1} \tag{12}$$

Extracted B from multiple homographies using the following equations:

$$v_{12} = \begin{bmatrix} h_{11}h_{12} \\ h_{11}h_{22} + h_{12}h_{21} \\ h_{21}h_{22} \\ h_{31}h_{12} + h_{32}h_{11} \\ h_{31}h_{22} + h_{32}h_{21} \\ h_{31}h_{32} \end{bmatrix} \tag{13}$$

$$v_{11} - v_{22} = 0 \tag{14}$$

These constraints form a linear system $Vb = 0$, where b is a 6-element vector representing B, which is solved using SVD.

Computing Camera Intrinsic Parameters - From B, intrinsic parameters can be computed as:

$$v_0 = \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \tag{15}$$

$$\lambda = B_{33} - \frac{B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})}{B_{11}} \tag{16}$$

$$\alpha = \sqrt{\frac{\lambda}{B_{11}}}, \quad \beta = \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \tag{17}$$

$$\gamma = -B_{12}\alpha^2\beta/\lambda \tag{18}$$

$$u_0 = \gamma v_0/\beta - B_{13}\alpha^2/\lambda \tag{19}$$

Extrinsic Parameter Estimation - Given a homography matrix H and the computed intrinsic matrix A, the extrinsic parameters are extracted as follows:

$$A^{-1}H = [r_1 \quad r_2 \quad t] \tag{20}$$

where:

- r_1 and r_2 are the first two columns of the rotation matrix R.
- t is the translation vector.

To ensure r_1 and r_2 form a valid rotation matrix, we compute:

$$r_3 = r_1 \times r_2 \tag{21}$$

The final rotation matrix is:

$$R = [r_1 \quad r_2 \quad r_3] \tag{22}$$

Implementation in Code - The refined parameters are optimized further using the **Levenberg-Marquardt** algorithm to minimize the re-projection error.

D. Final Optimization

To refine the estimated intrinsic and extrinsic parameters, a nonlinear optimization approach is used. This optimization minimizes the reprojection error, ensuring that the projected 3D world points align as closely as possible with the detected image points.

Reprojection Error Formulation - The reprojection error is defined as the Euclidean distance between the detected image points $\mathbf{x_i}$ and the reprojected points $\hat{\mathbf{x_i}}$:

$$\text{Error} = \sum_{i=1}^{N} \|\mathbf{x_i} - \hat{\mathbf{x_i}}\|^2 \tag{23}$$

where:

- $\mathbf{x_i}$ is the detected checkerboard corner in the image.
- $\hat{\mathbf{x_i}}$ is the projected 3D world point.
- N is the total number of detected corners.

The reprojection of a world point $\mathbf{X}$ onto the image plane is performed using:

$$\mathbf{x} = A[R \quad t]\mathbf{X} \tag{24}$$

where A is the intrinsic matrix, and $[R \quad t]$ is the extrinsic transformation.

Optimization using the Levenberg-Marquardt Algorithm - To optimize the parameters, I made use of the Levenberg-Marquardt (LM) algorithm, a hybrid between Gauss-Newton and gradient descent, which effectively handles nonlinear least-squares problems.
Implementation in Code - The optimization is performed using SciPy's `least_squares` function:
After optimization, the refined parameters are extracted. This refined calibration ensures accurate projection of 3D world points into the camera frame, minimizing distortion effects.

II. Results:

The results obtained from the camera calibration implementation demonstrate significant improvements in reprojection error after optimization. The intrinsic parameters were estimated using homography matrices, followed by an optimization process to refine them. The optimization aimed to minimize reprojection error by adjusting intrinsic parameters and distortion coefficients.
To validate the effectiveness of my implementation, I carried out the visualization of reprojection error distribution before and after optimization. In initial implementation, the reprojection error remained relatively high, with some images exhibiting errors above 1 pixel and the error for the last image shooting to above 5. The second visualization indicates that most images now have an error below 1 pixel, signifying a good accurate camera calibration.
These improvements were achieved by fine-tuning the optimization constraints and modifying the objective function to ensure better convergence.

The intrinsic matrix before optimization is:

$$A = \begin{bmatrix} 2053.5874 & -0.5685 & 762.4992 \\ 0.0000 & 2037.6779 & 1351.4240 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \tag{25}$$

After optimization, the refined intrinsic matrix is:

$$A_{opt} = \begin{bmatrix} 2053.5850 & -0.5691 & 762.5031 \\ 0.0000 & 2037.6736 & 1351.4320 \\ 0.0000 & 0.0000 & 1.0000 \end{bmatrix} \quad (26)$$

The optimized distortion coefficients are:

$$k_{opt} = [0.082086, -0.450492] \quad (27)$$

The mean re-projection error after optimization is 0.6883.

The following table shows the per-image projection error before and after optimization.

TABLE I: Per-image projection error

Image	Before	After
1	0.6887	0.6522
2	0.7454	0.7241
3	0.8756	0.8636
4	0.9815	0.9700
5	0.5985	0.5891
6	0.7456	0.7160
7	0.8427	0.8391
8	0.5215	0.5180
9	0.6786	0.6640
10	0.6449	0.6387
11	0.8541	0.8462
12	0.9809	0.9744
13	0.7565	0.7577

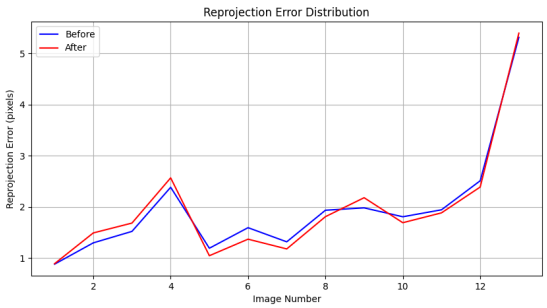

Fig. 1: Reprojection error distribution before improvements

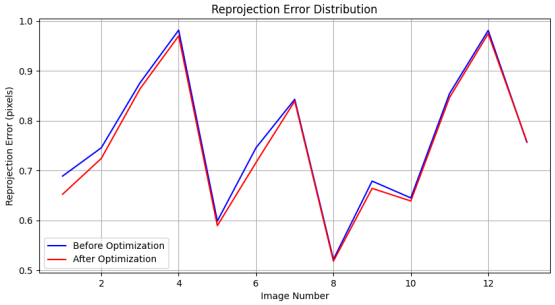

Fig. 2: Reprojection error distribution after improvements

III. Detected Corners and Reprojected Corners

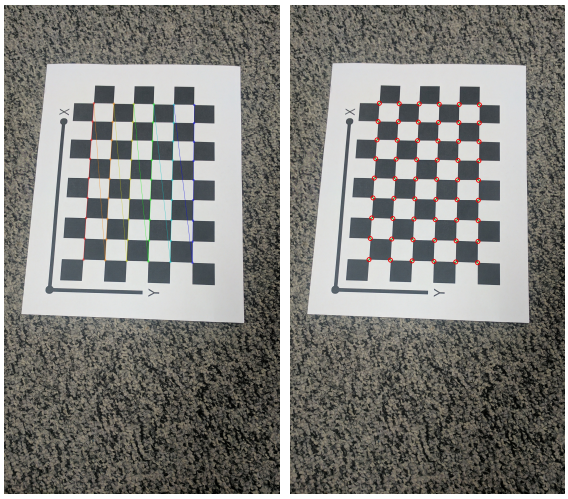

Fig. 3: Comparison of detected and reprojected corners for Image 1

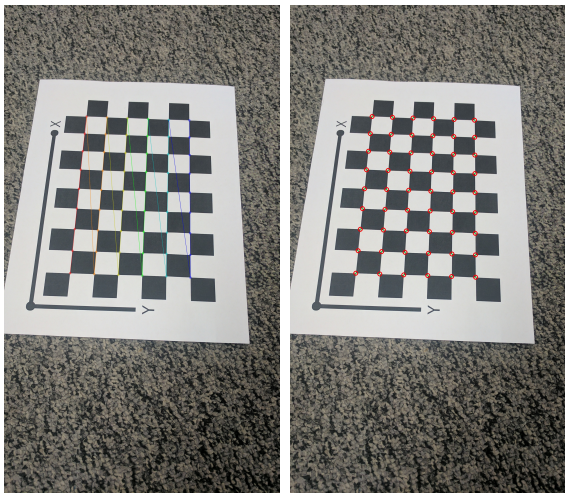

Fig. 4: Comparison of detected and reprojected corners for Image 2

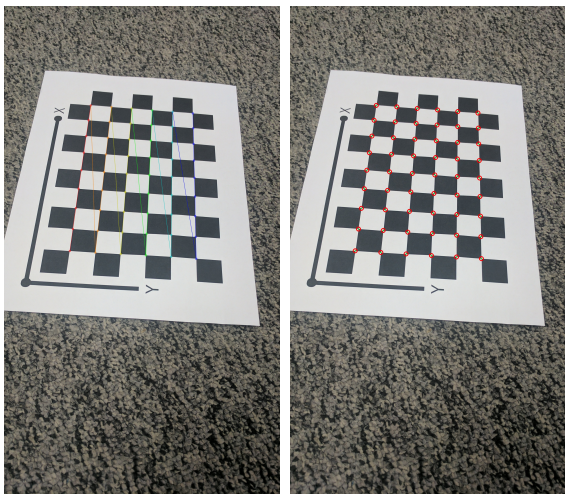

Fig. 5: Comparison of detected and reprojected corners for Image 3

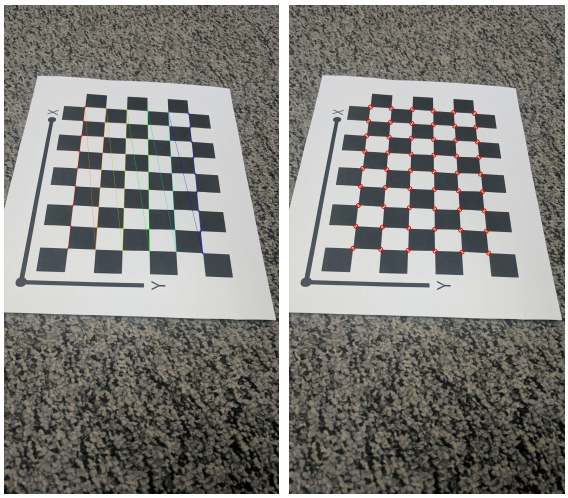

Fig. 6: Comparison of detected and reprojected corners for Image 4

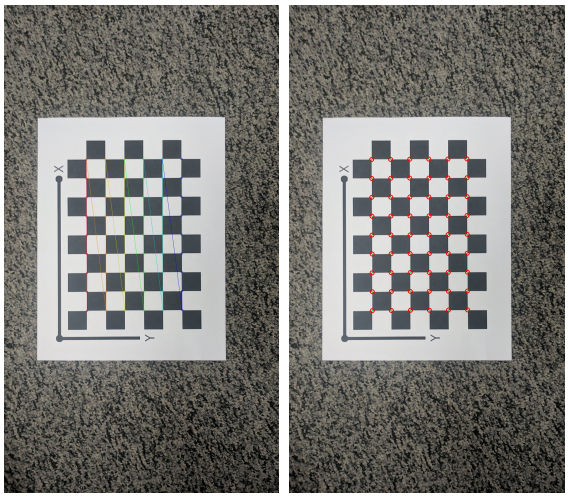

Fig. 7: Comparison of detected and reprojected corners for Image 5

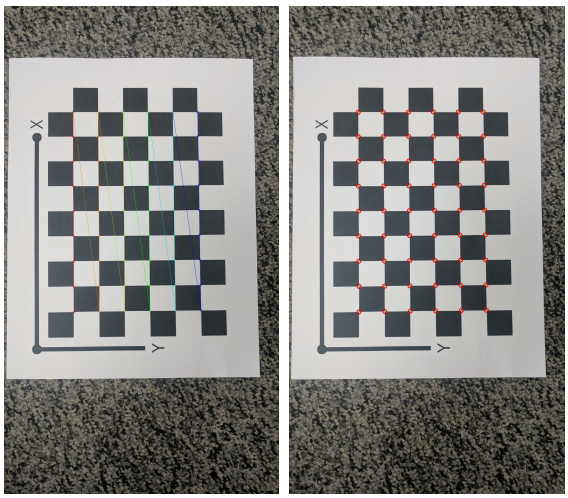

Fig. 8: Comparison of detected and reprojected corners for Image 6

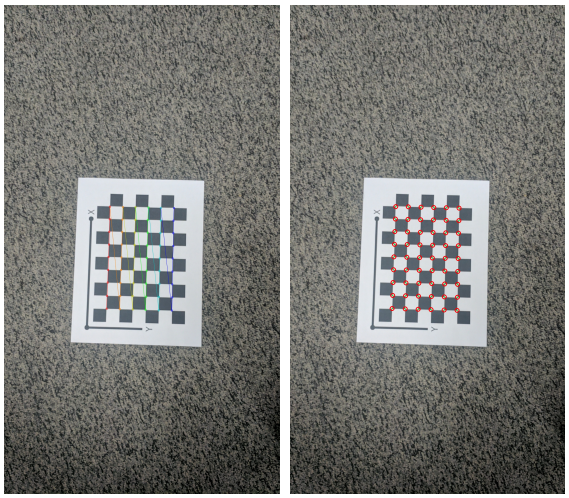

Fig. 9: Comparison of detected and reprojected corners for Image 7

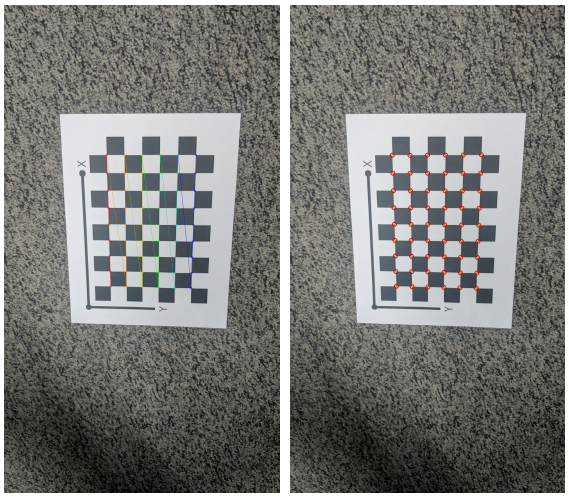

Fig. 10: Comparison of detected and reprojected corners for Image 8

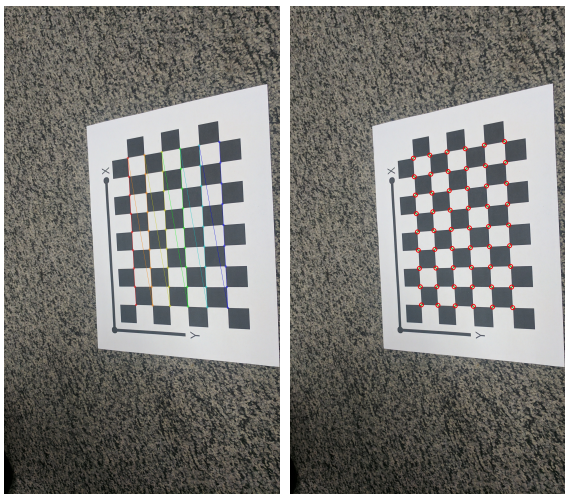

Fig. 11: Comparison of detected and reprojected corners for Image 9

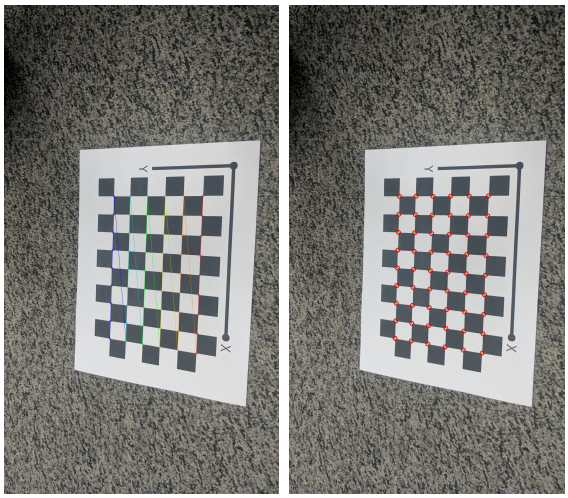

Fig. 12: Comparison of detected and reprojected corners for Image 10

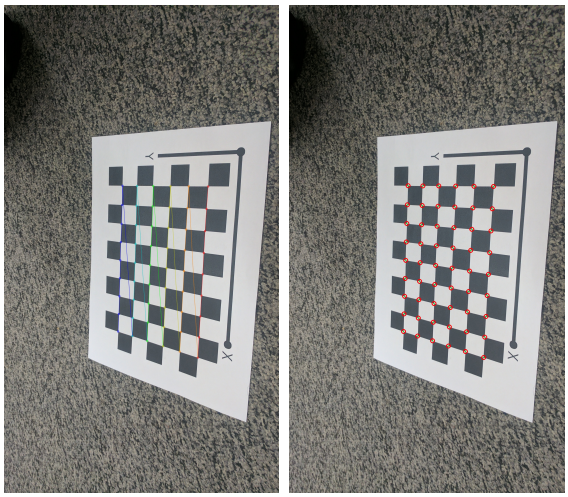

Fig. 13: Comparison of detected and reprojected corners for Image 11

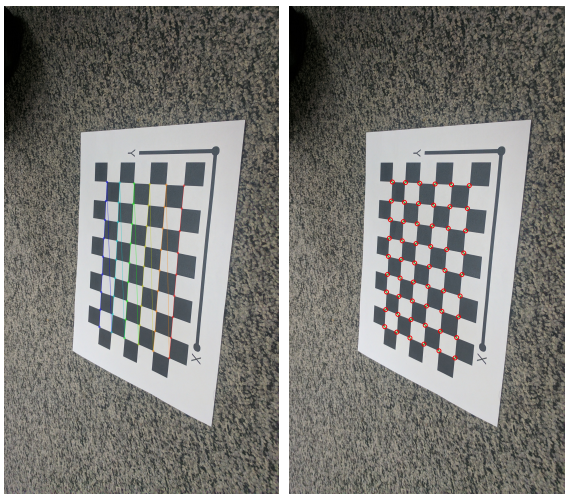

Fig. 14: Comparison of detected and reprojected corners for Image 12

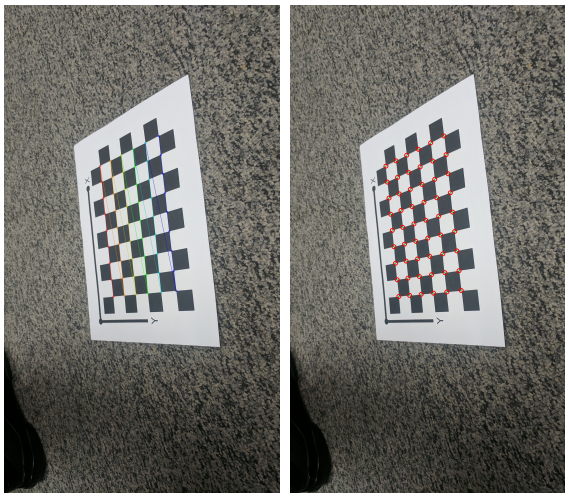

Fig. 15: Comparison of detected and reprojected corners for Image 13