Camera Calibration Using Zhang's Method

Daniele Maijnelli, Cecilia Comar

February 5, 2025

1 Problem Statement

The goal of this project is to perform the following tasks using the images provided at the URL https://tinyurl.com/CVPR2024project1:

- calibrating using Zhang procedure, i.e., estimate the calibration matrix K that encodes the camera's intrinsic parameters and, for each image, estimate the pair R and t that encodes extrinsic parameters;
- computing the total re-projection error for all the grid points of an image from the available set;
- superimposing a parallelepiped onto the calibration plane in all the images employed for the calibration;
- performing the previous tasks on a set of images of a printed checkerboard acquired with a smartphone;
- adding radial distortion compensation to the basic Zhang's calibration procedure;
- computing the total reprojection error with radial distortion compensation and making a comparison to the one without compensation performed before;

2 Description of the approach

2.1 Zhang's Calibration Procedure

The main steps of our approach include:

1. detect corners (feature points) in each image using the OpenCV function `findChessboardCorners()`;
2. estimate the homography matrix H for each image;
3. use homographies to estimate the intrinsic parameters of the camera (K);
4. estimate the extrinsic parameters for each image: the rotation matrix R and translation vector t.

2. Estimate Homographies The goal of this step is to compute the transformation between the real-world (planar) coordinates of the calibration pattern and their corresponding image coordinates, enabling further steps in camera calibration. This is the methodology that we have followed:

1. Generating real-world coordinates
    - The real-world (2D planar) coordinates of the calibration pattern are computed using a known grid size and square size.
    - For each detected corner in the calibration pattern:
        - the row (u-index) and column (v-index) are determined.
        - the real-world coordinates (x, y) are calculated by multiplying these indices by the square size
        - these computed coordinates represent the true 2D positions of the pattern's corners in millimeters.
2. Formulating the homography matrix estimation
    - a homography matrix (H) is a 3 x 3 matrix that links the real-world (planar) coordinates to their image coordinates in the following manner:

$$w \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

      where (x, y) are real world coordinates, (u, v) are image coordinates and w is a scaling factor.
3. Constructing the linear system
    - for each point correspondence (x, y) in the real-world coordinate system and its associated image point (u, v), the following two equations are constructed:

$$xh_{11} + yh_{12} + h_{13} - u(xh_{31} + yh_{32} + h_{33}) = 0$$
$$xh_{21} + yh_{22} + h_{23} - v(xh_{31} + yh_{32} + h_{33}) = 0$$

    - these equations are rewritten in matrix form and stacked for all point correspondences to form a linear system $Ah = 0$
4. Solving for H using Singular Value Decomposition (SVD)
    - the Singular Value Decomposition (SVD) is applied to the matrix A, yielding:

$$A = USV^T$$

    - the homography matrix H is extracted as the last column of V (corresponding to the smallest singular value), reshaped into a 3×3 matrix

The computed homography matrices are printed for each image, providing the transformation between real-world and image-plane coordinates.

3. Estimate intrinsic parameters of the camera The homography matrices were used to compute the camera calibration matrix K.
This is the methodology that we have followed:

1. Since the camera projection model follows

$$H = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$$

where

$$\begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$$

represents the camera's rotation and translation, by exploiting the fact that R is a rotation matrix and other mathematical properties we obtain the following (for $i, j \in \{1, 2\}$):

$$v_{ij} b = h_i^T B h_j$$

where $B = K^T K^{-1}$, $b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B_{33}]^T$ and

$$v_{ij} = \begin{bmatrix} H_{1i}H_{1j} \\ H_{1i}H_{2j} + H_{2i}H_{1j} \\ H_{2i}H_{2j} \\ H_{3i}H_{1j} + H_{1i}H_{3j} \\ H_{3i}H_{2j} + H_{2i}H_{3j} \\ H_{3i}H_{3j} \end{bmatrix}$$

In our case we obtain 2 equations for each image (so 40 in total):

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

These constraints are then stacked into a matrix V.

2. Solving for B

   - The matrix V is decomposed using SVD

$$V = USV^T$$

   - The last column of V^T gives the solution vector b

3. To compute the intrinsic matrix K, since $B = K^{-T}K^{-1}$, we perform the following:

   - Cholesky factorization is applied to B:

$$B = LL^T$$

     where L is a lower triangular matrix
   - K is then computed as

$$K = (L^T)^{-1}$$

   - The result is normalized such that $K_{33} = 1$ ensuring the scale consistency.

4. Estimate extrinsic parameters for each image Given the estimated intrinsic matrix K and homography matrices H_i, we can compute the rotation matrix R_i and translation vector t_i for each calibration image ($i = 0, ..., 19$). The idea is to extract the extrinsic parameters of each calibration image using K and the H of the image. The following methodology is applied for each H (for each image):

1. estimate the scaling factor λ as

$$\lambda = \frac{1}{||K^{-1}h_1||}$$

2. compute the first 2 columns of R:

$$r_1 = \lambda K^{-1}h_1$$

$$r_2 = \lambda K^{-1}h_2$$

3. compute the translation vector t

$$t = \lambda K^{-1}h_3$$

4. compute the third column of R, r_3: since a valid rotation matrix must have orthogonal column vectors, r_3 is computed as the cross product of the first two:

$$r_3 = r_1 \times r_2$$

5. The computed R may not be perfect due to numerical errors. To correct this, (SVD) is used to find the closest valid rotation matrix, so we use $R' = UV^T$ where $R = U\Sigma V^T$.

2.2 Total Re-projection Error

Reprojection error measures the difference between the actual measured image points and the reprojected points obtained using the estimated camera parameters. These are the steps that we have followed for computing the total reprojection error:

1. selection of an image from the given set;

2. projecting $3D$ points to $2D$ image coordinates: the projected image coordinates (u, v) are computed as

$$u = \frac{p_1 \cdot m_i}{p_3 \cdot m_i}$$

$$v = \frac{p_2 \cdot m_i}{p_3 \cdot m_i}$$

where p_1, p_2,p_3 are the rows of P and m_i represents the $3D$ world coordinates of a point;

3. for each image point ($u_{measured}$, $v_{measured}$) and its corresponding projected point (u_{proj}, v_{proj}) the error is computed as:

$$Error = (u_{proj} - u_{measured})^2 + (v_{proj} - v_{measured})^2$$

4. computing the total reprojection error as the sum of errors over all points:

$$E_{total} = \sum_{i=1}^{N} [(u_{proj,i} - u_{measured,i})^2 + (v_{proj,i} - v_{measured,i})^2]$$

5. to visually analyze the reprojection error, the measured and projected points are displayed on a copy of the original image.

2.3 Parallelepiped Superimposition

Using the estimated intrinsic and extrinsic parameters, a parallelepiped is superimposed onto the calibration plane for all images used in the calibration. The detailed steps are:

1. Defining the $3D$ parallelepiped in real world coordinates:
   - a bottom face defined as a $2D$ rectangle in the calibration plane ($z = 0$).
   - a top face obtained by assigning a constant depth which determinates the height of the parallelepiped above the calibration plane;
2. Computing projected points: using the perspective projection matrix, the $3D$ points of the bottom and top faces are projected onto the $2D$ image plane.
3. Drawing the parallelepiped: to visualize the projected parallelepiped, we connected the points in order to form a parallelepiped with different colors (bottom face blue, top face green and vertical edges red connecting bottom and top faces)

2.4 Using a different set of images

The activities we performed are:

1. Finding a checkboard using an online generator and specifying its dimensions.
2. Printing it on a sheet of paper.
3. Taking 20 pictures from different angles and positions of the printed checkboard using the same smartphone camera.
4. Importing the images in the code and changing the value of variables in the code corresponding to the checkboard and to the parallelepiped.

For performing the calibration using Zhang procedure, computing the total reprojection error on one of the calibration images and superimposing a parallelepiped to all the calibration images we used the same code of the previous sections. The images captured by the smartphone camera are here reported:

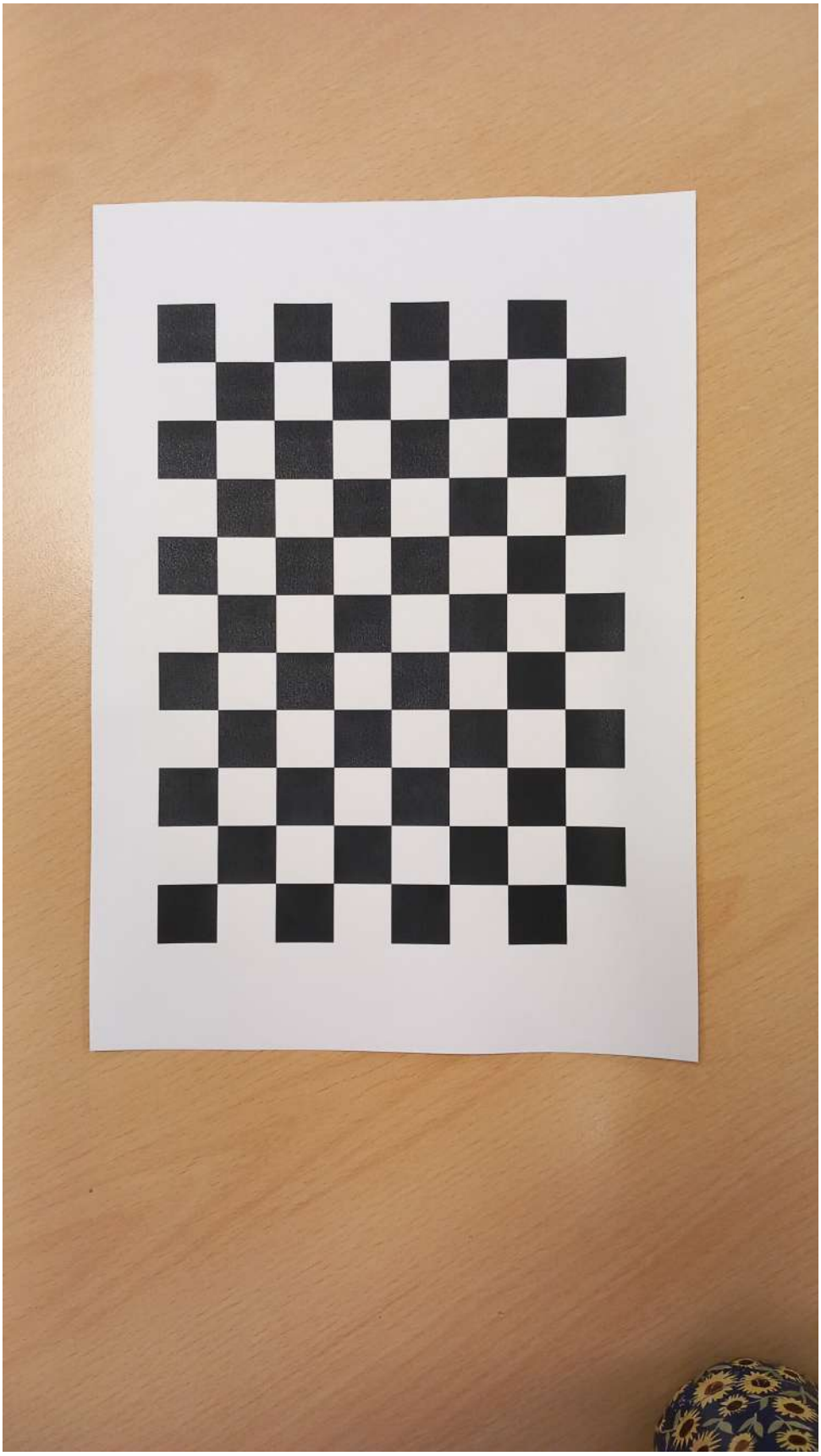


Figure 1: image 0

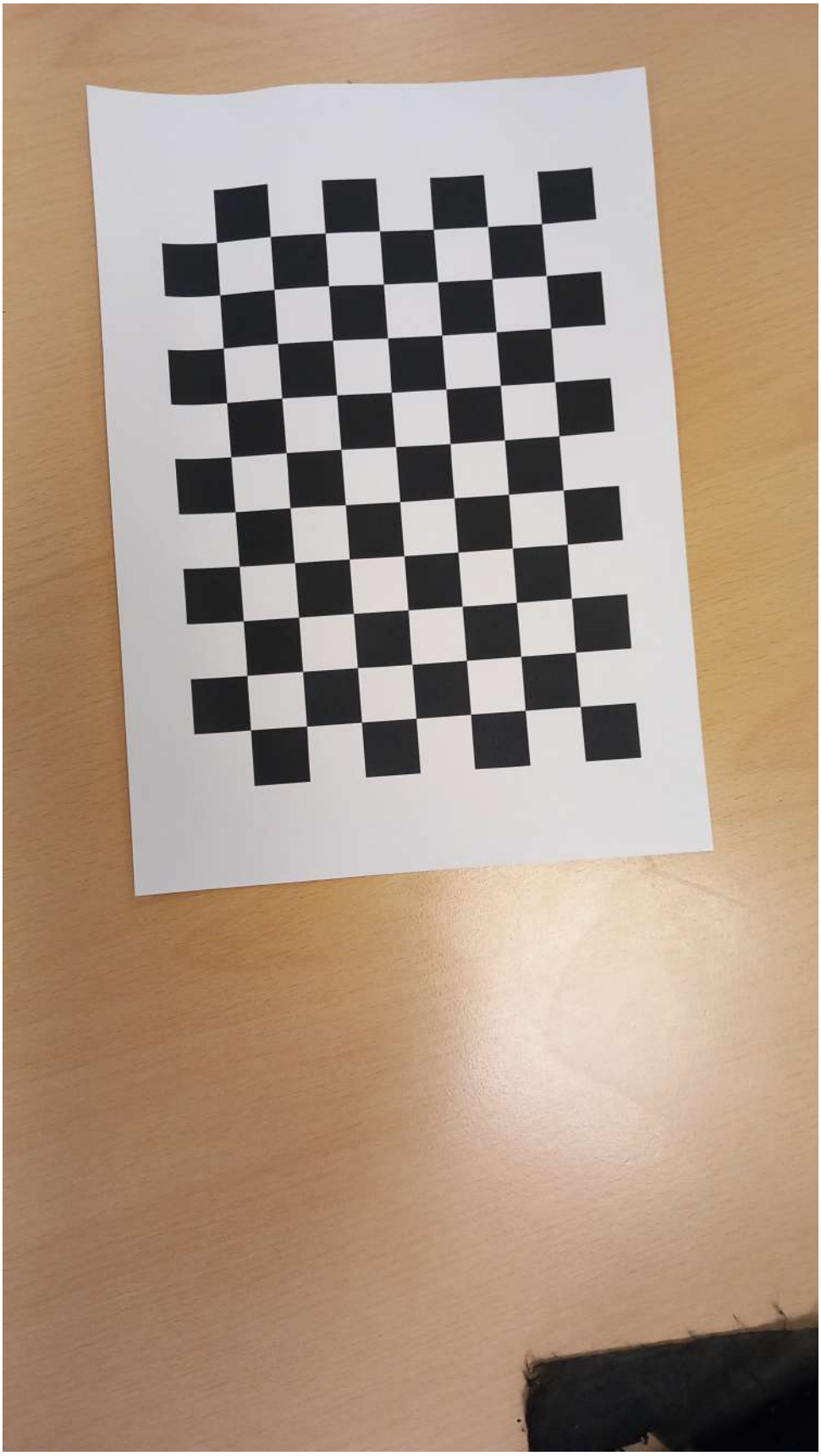


Figure 2: image 1

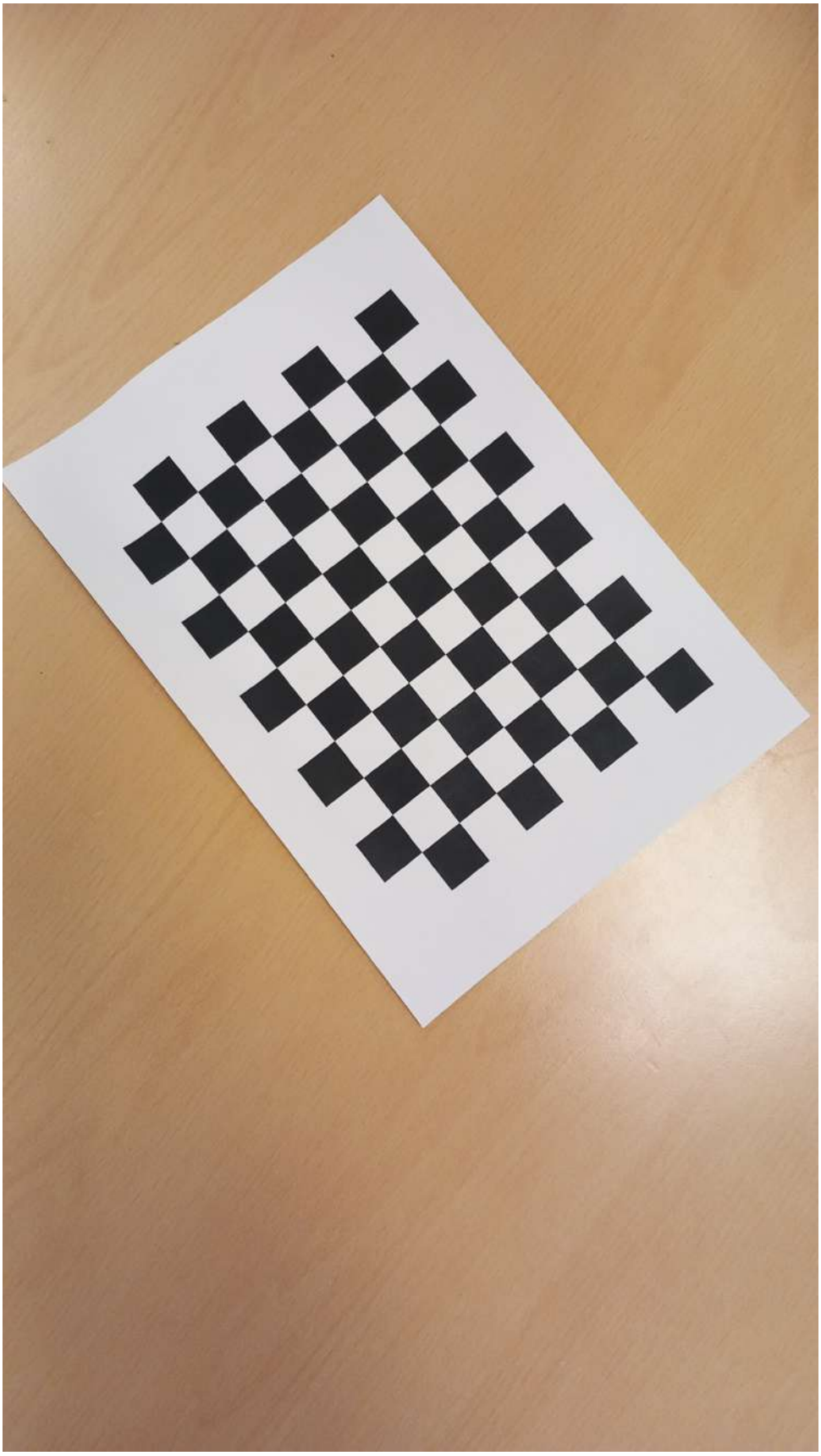


Figure 3: image 2

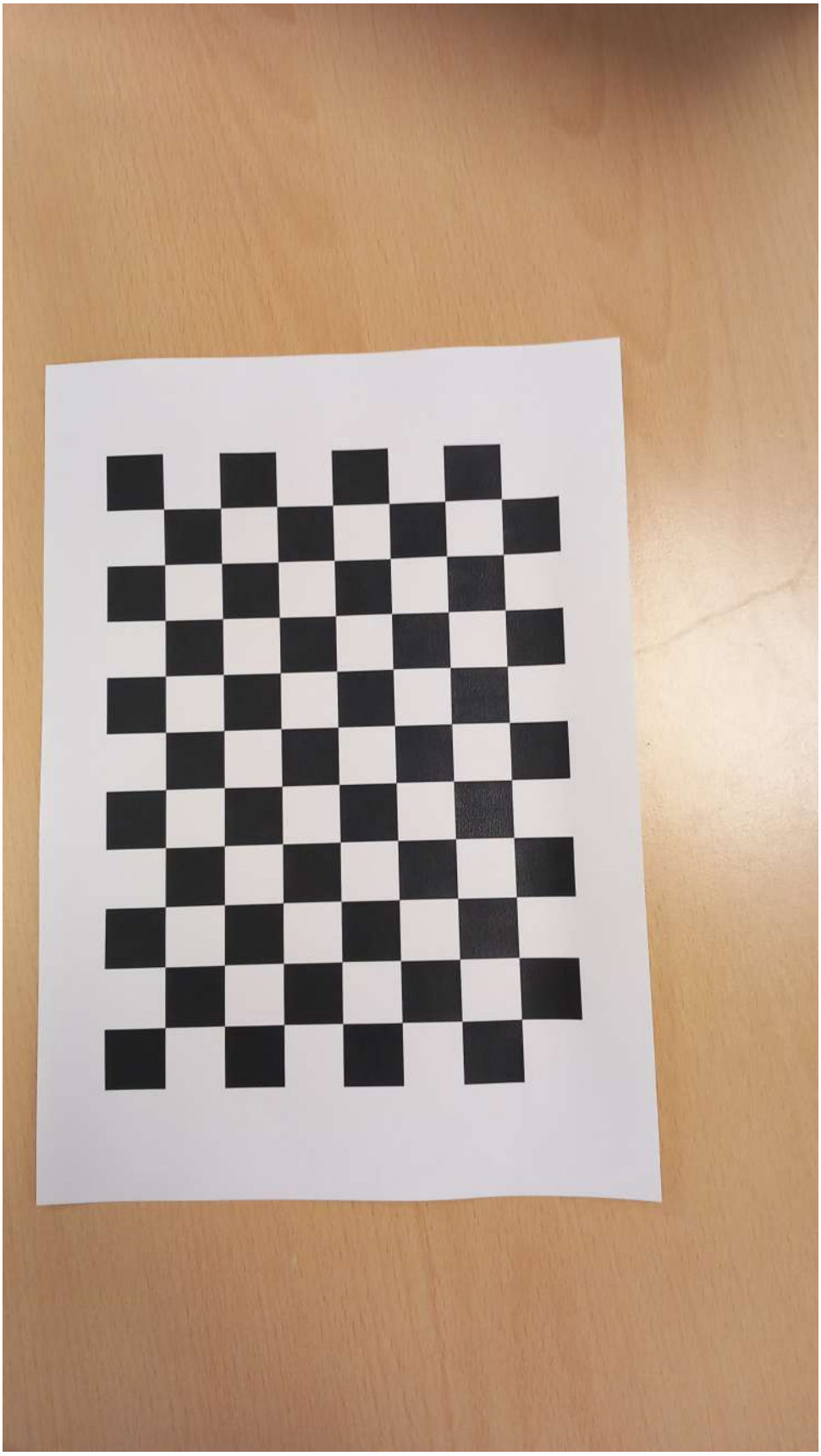


Figure 4: image 3

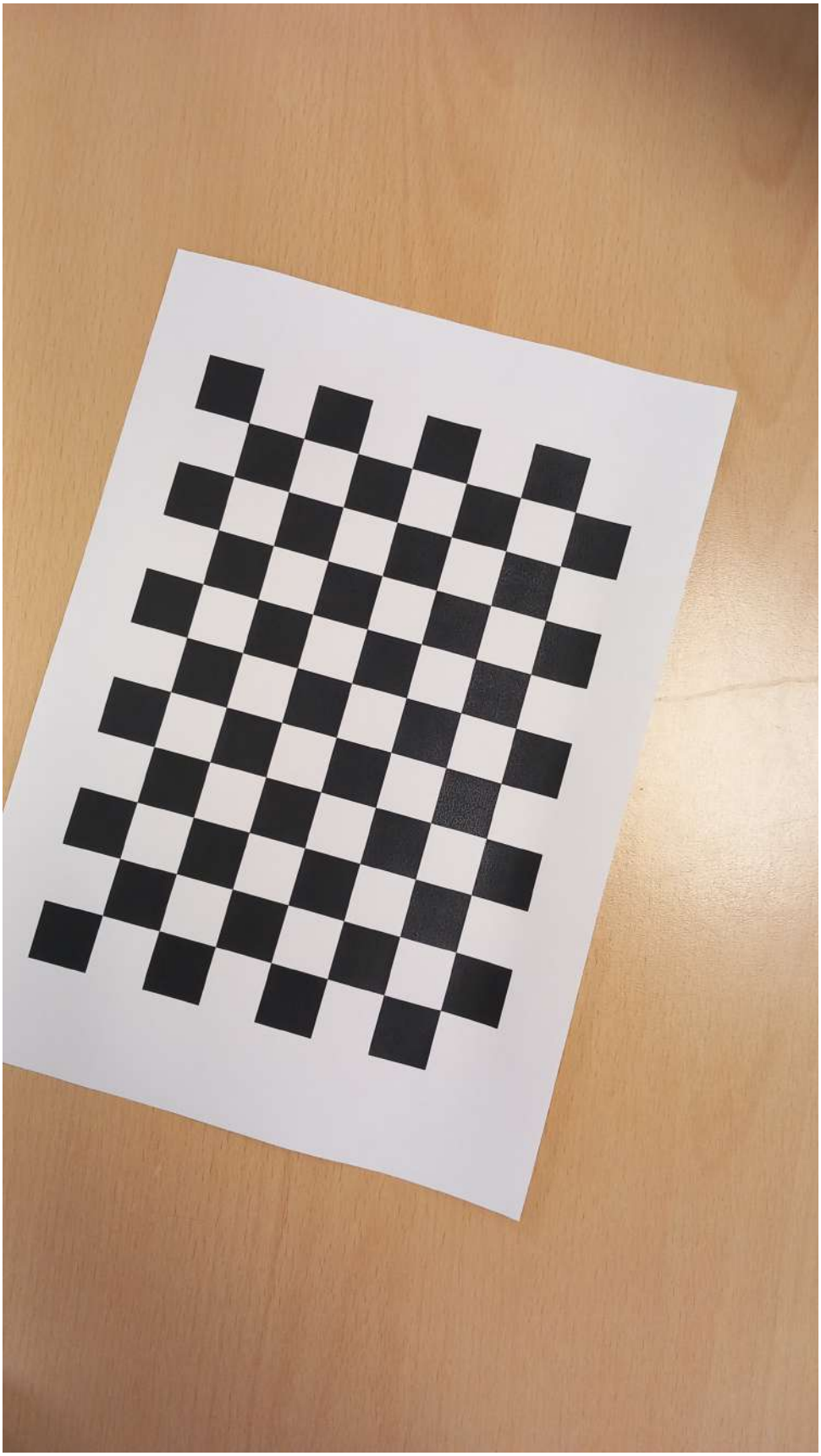


Figure 5: image 4

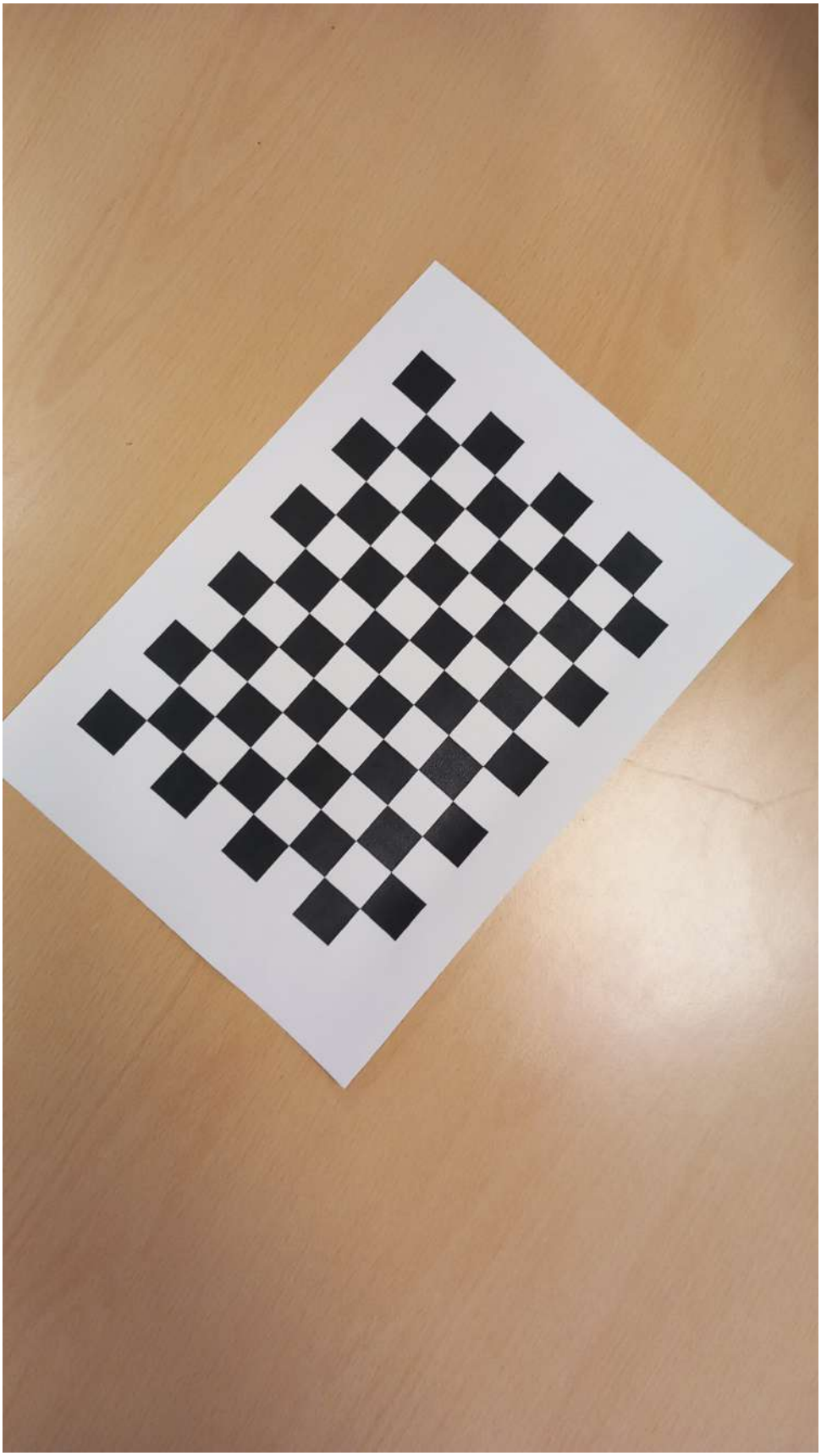


Figure 6: image 5

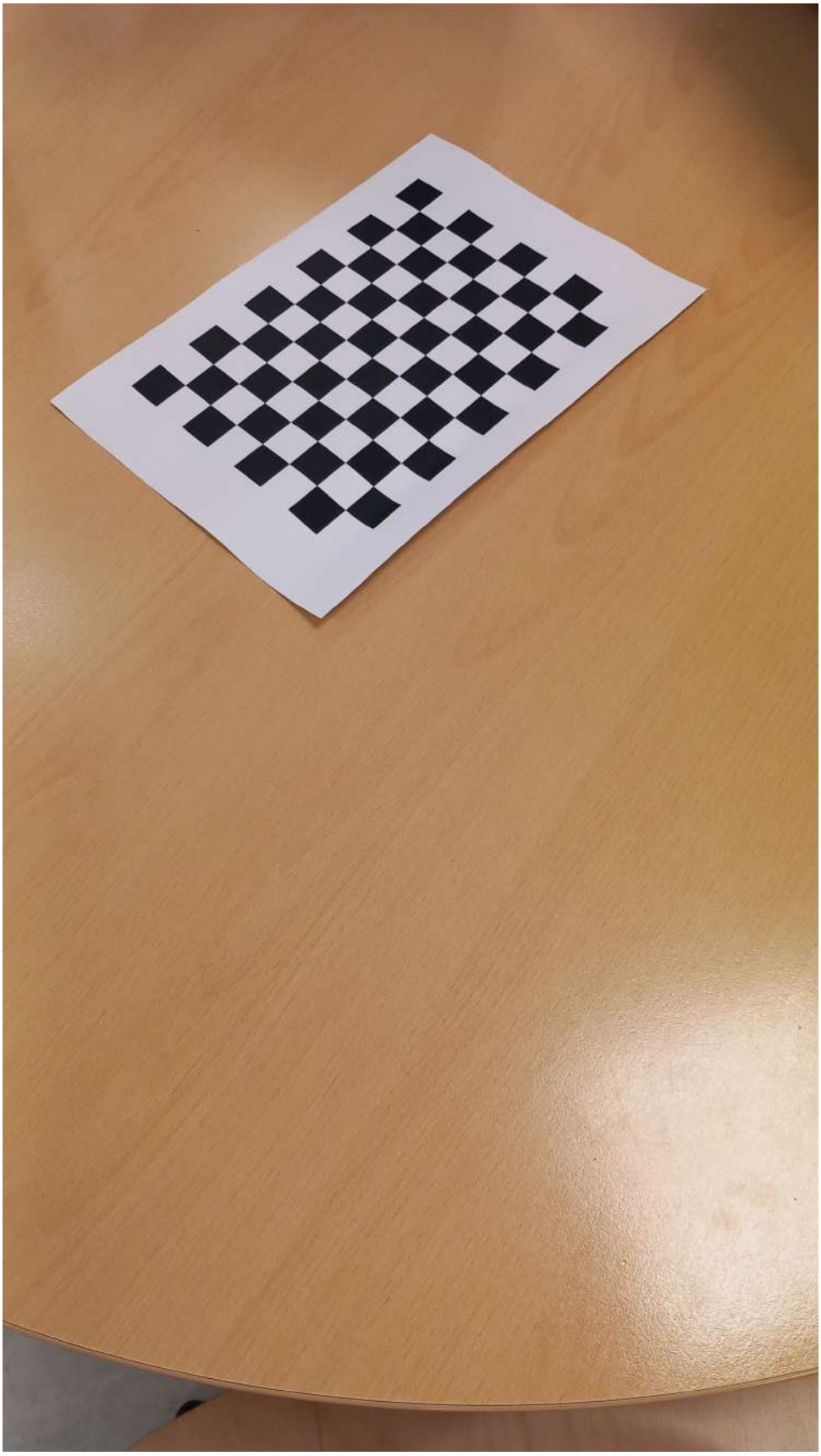


Figure 7: image 6

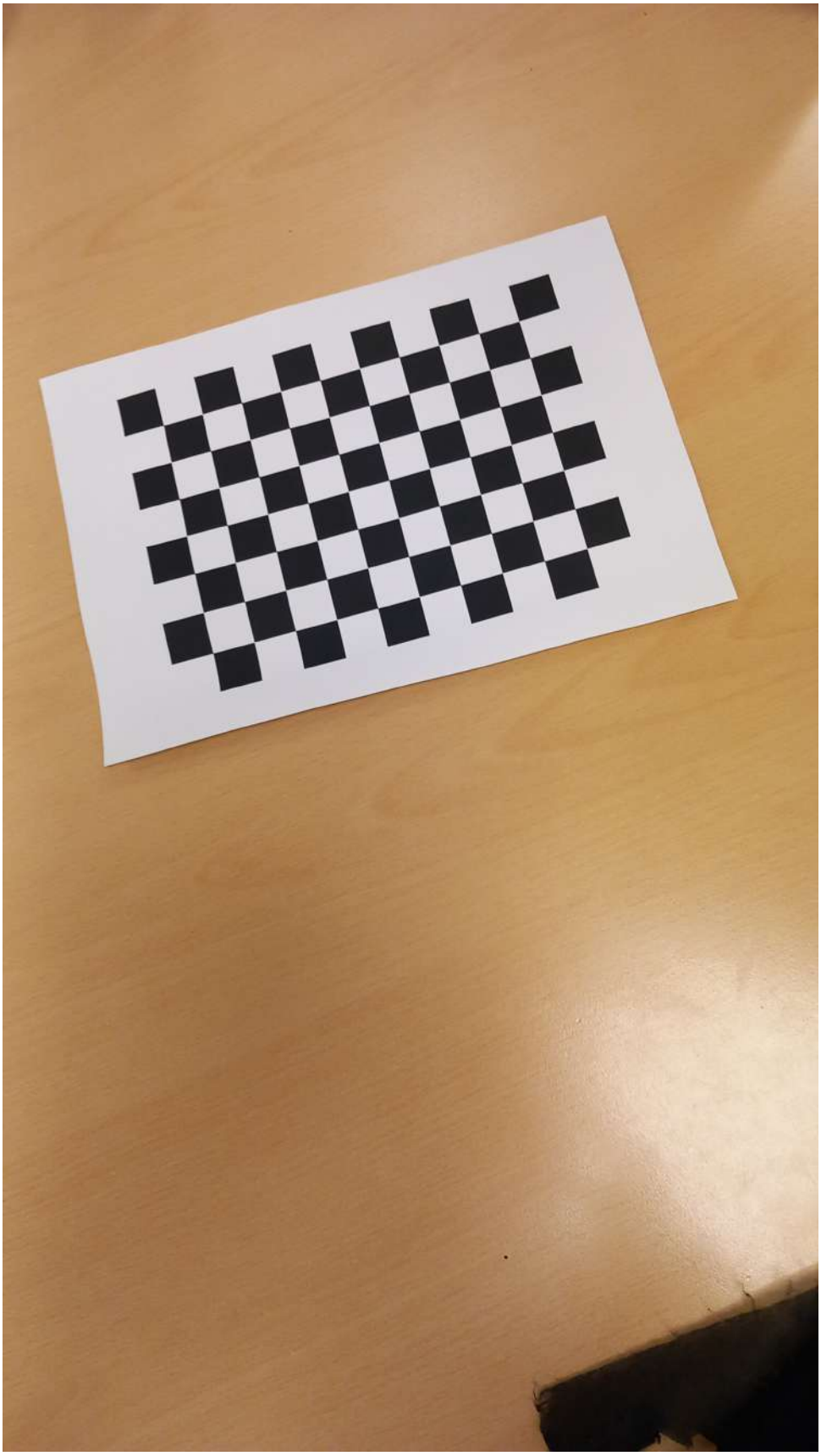


Figure 8: image 7

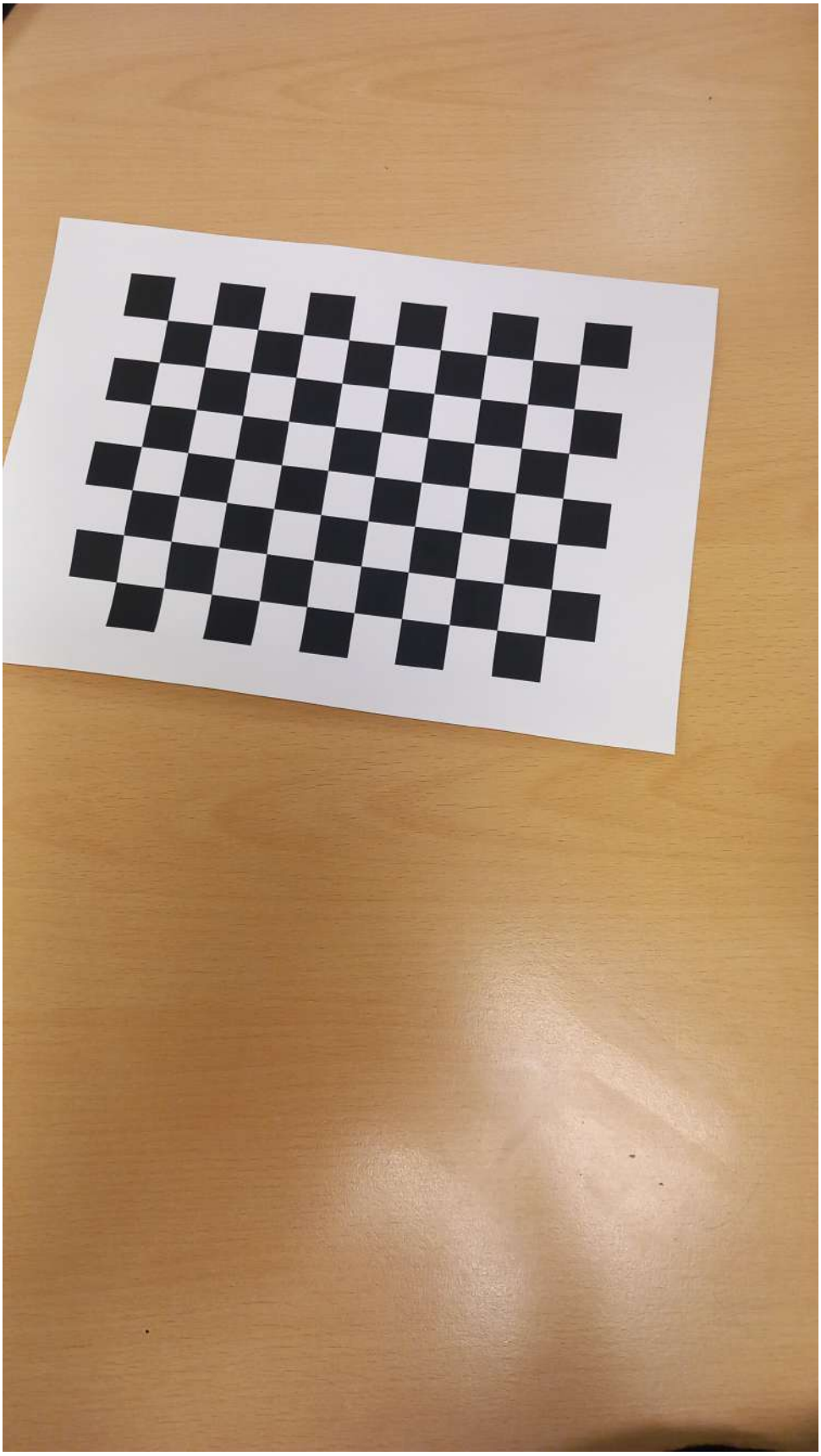


Figure 9: image 8

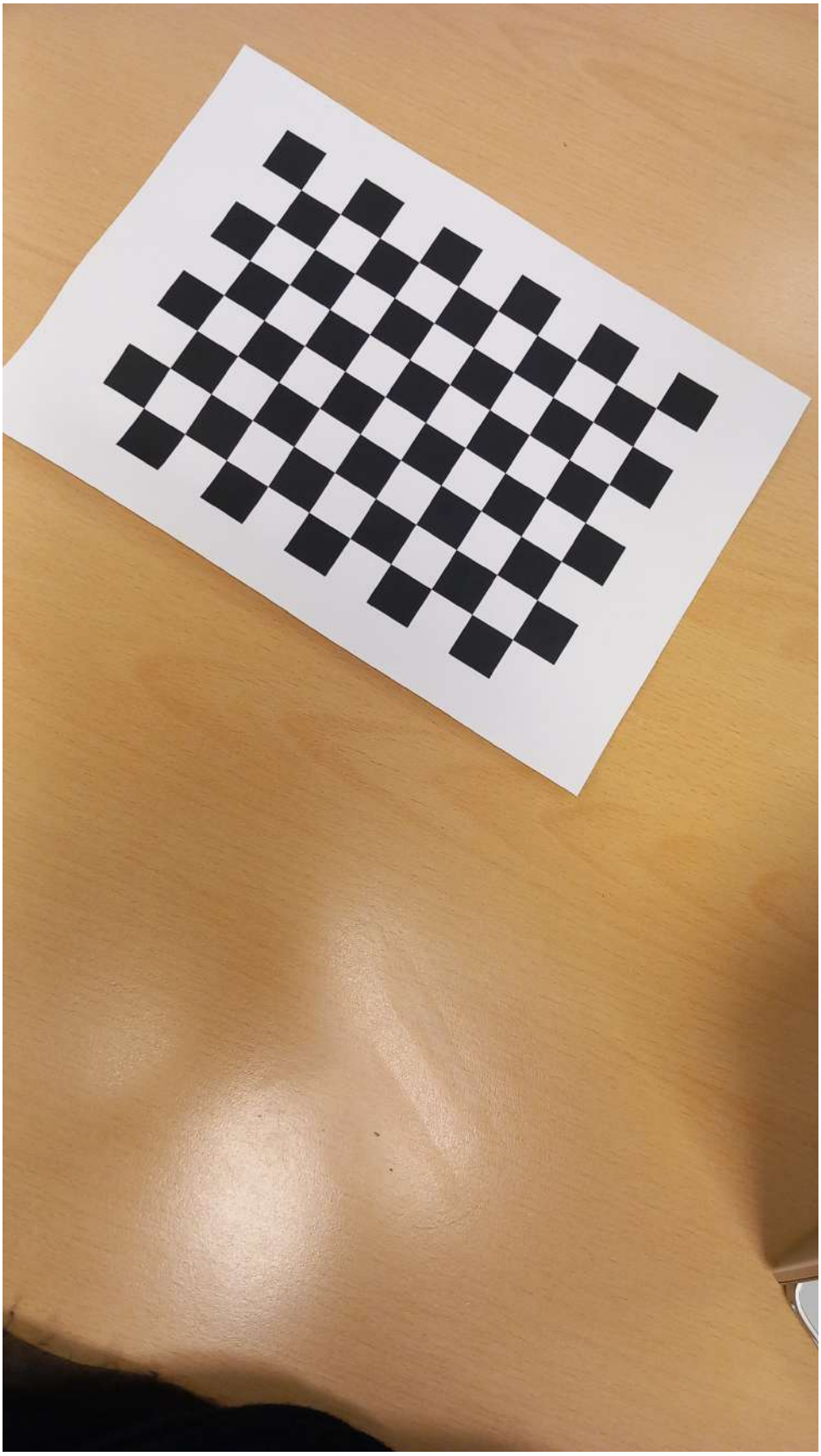


Figure 10: image 9

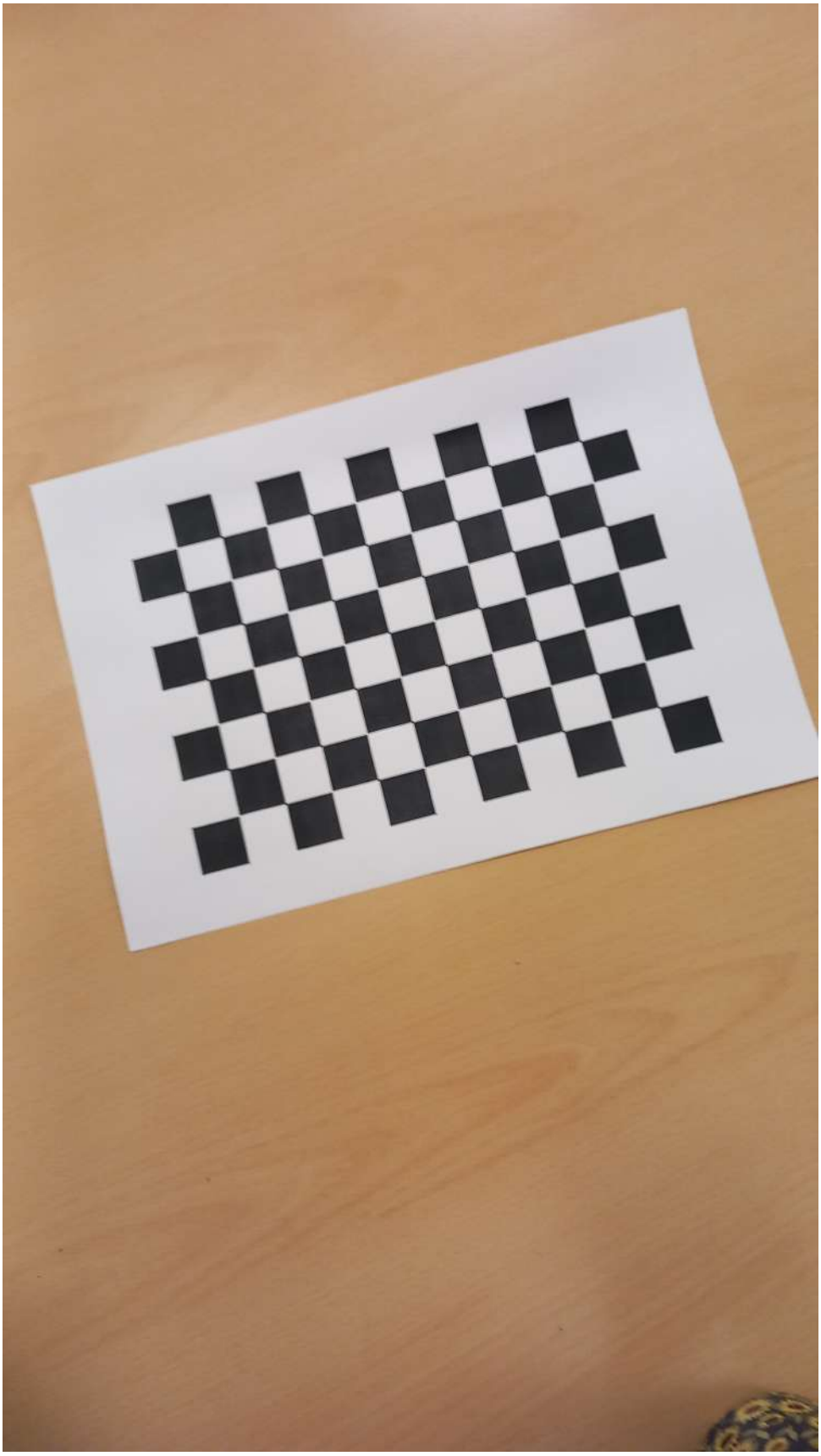


Figure 11: image 10

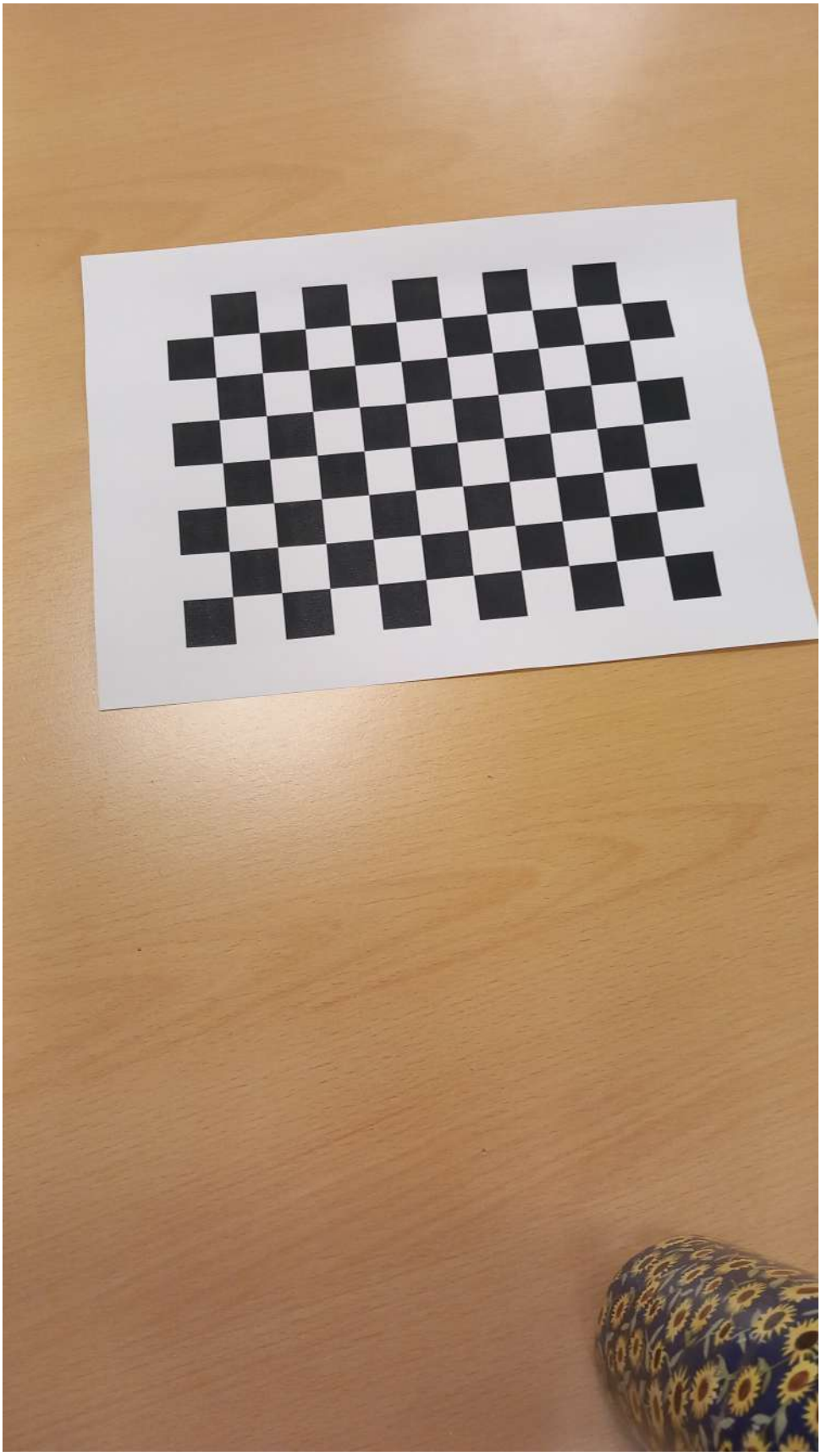


Figure 12: image 11

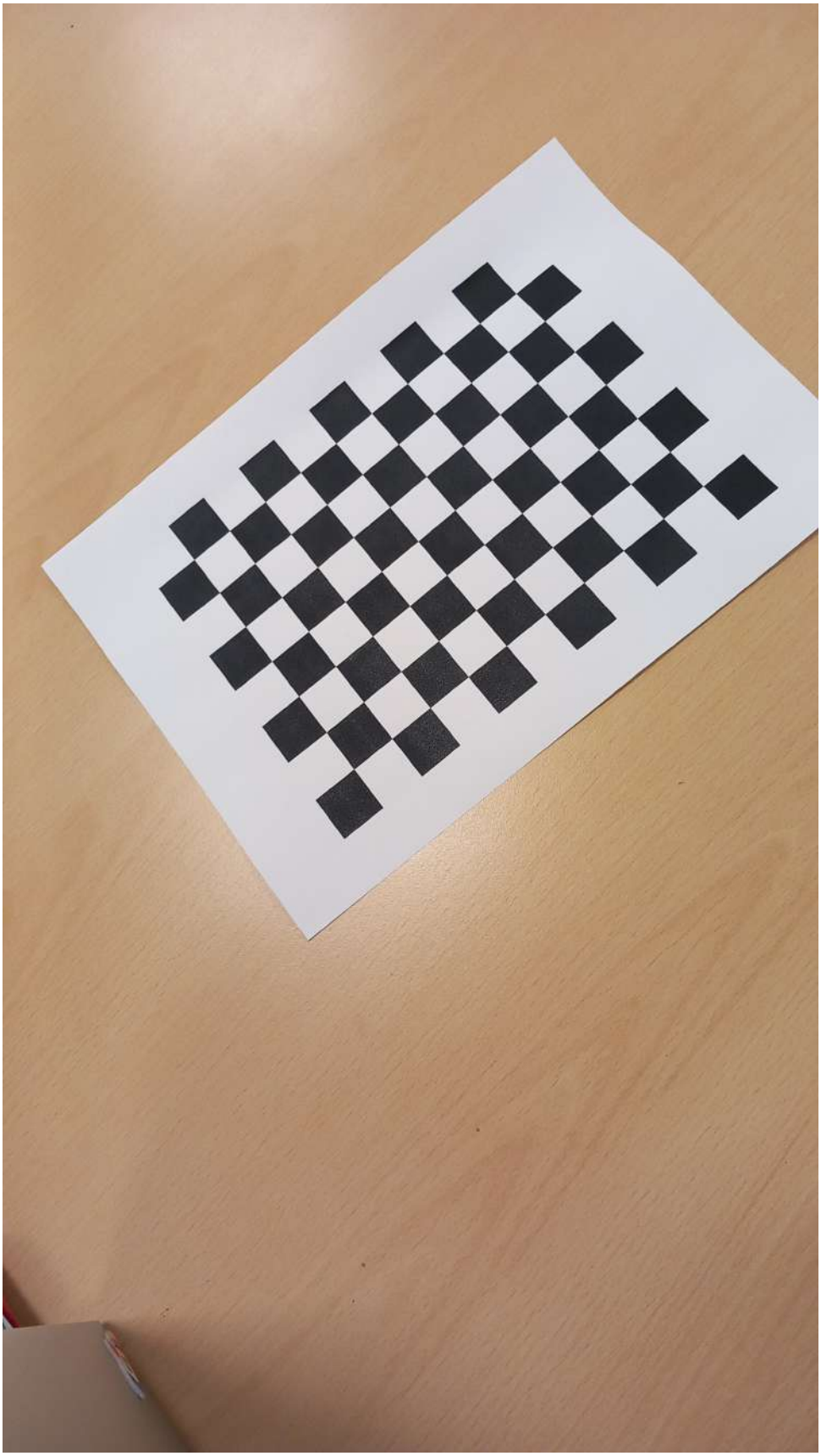


Figure 13: image 12

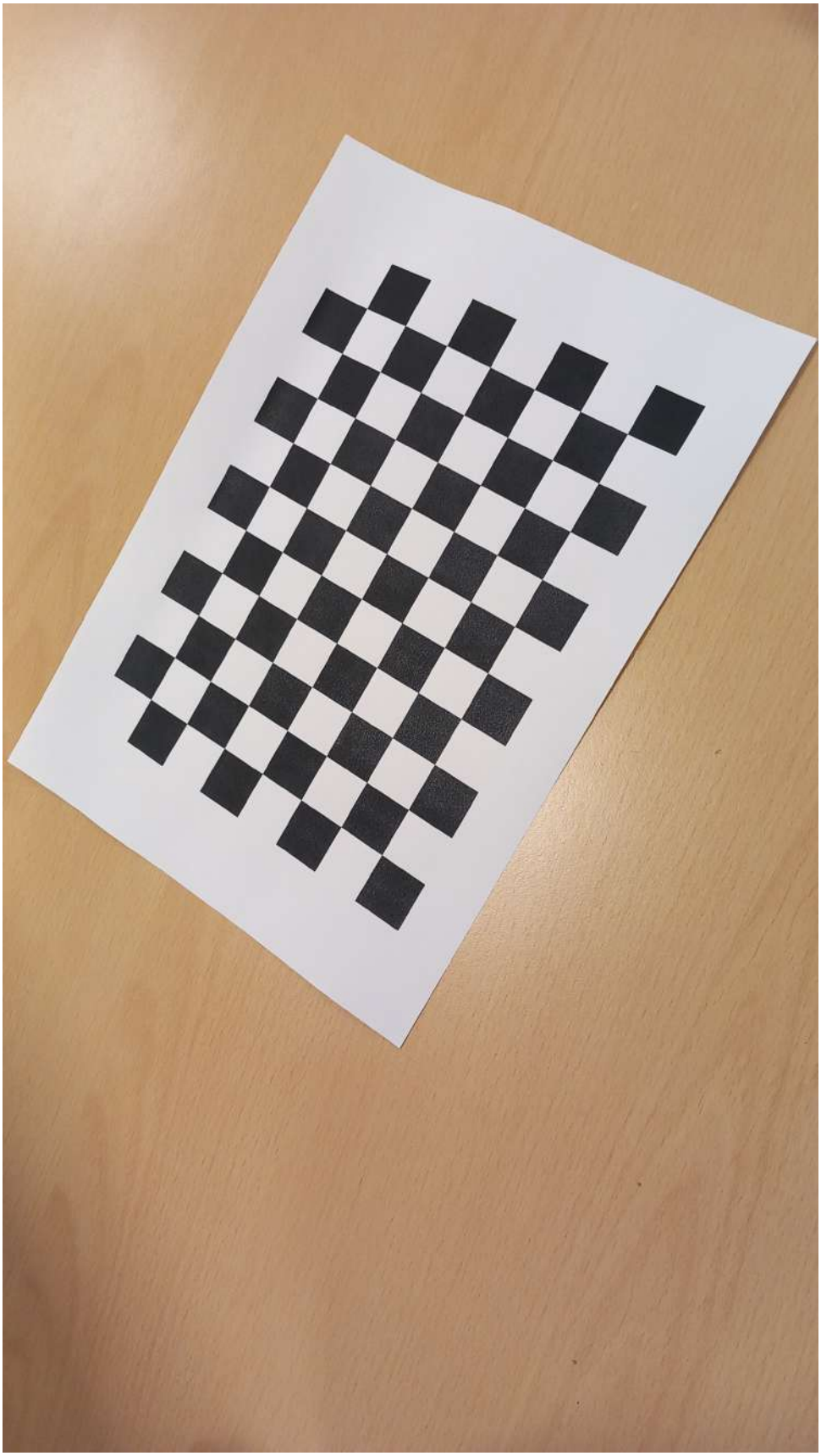


Figure 14: image 13

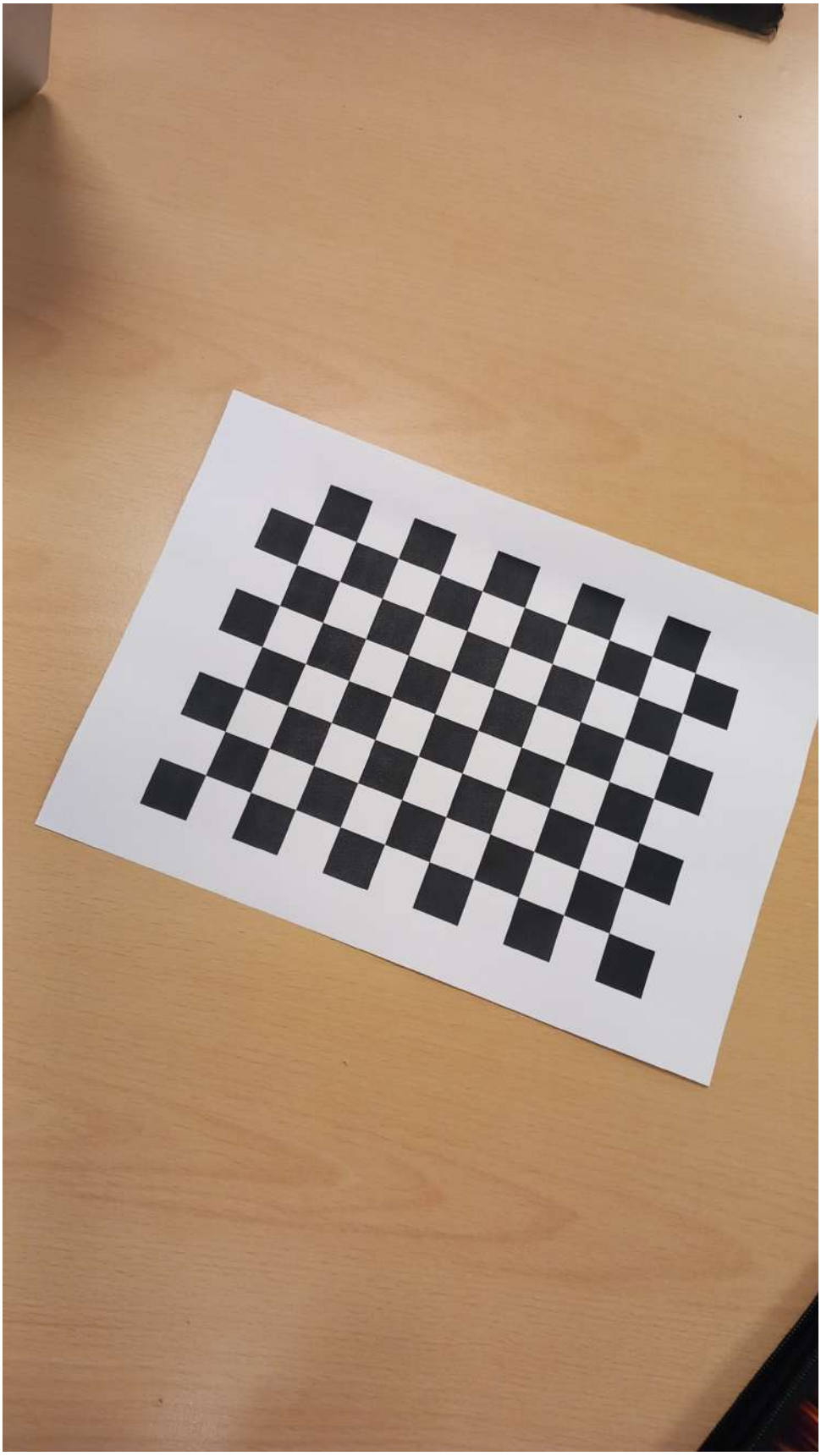


Figure 15: image 14

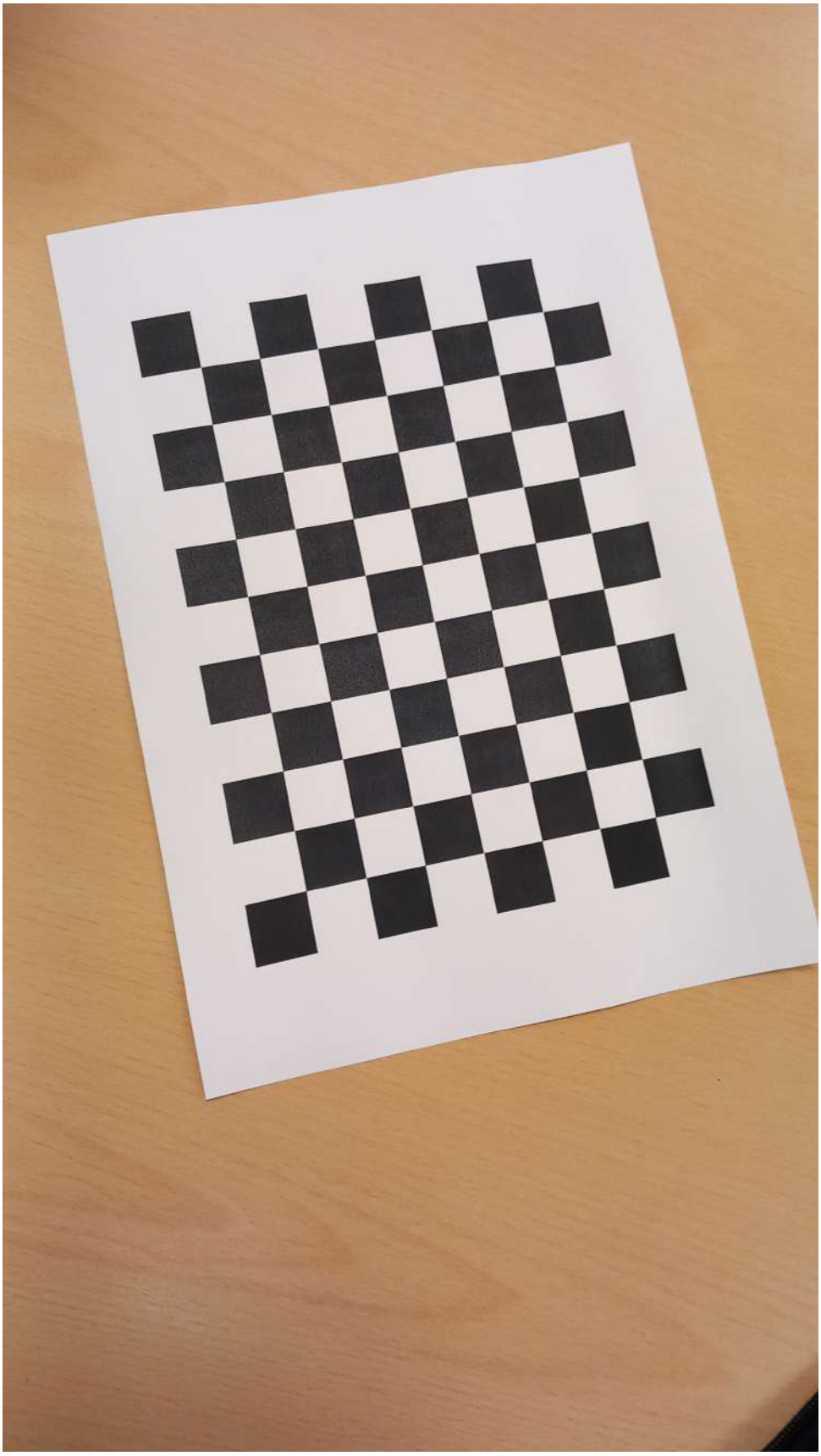


Figure 16: image 15

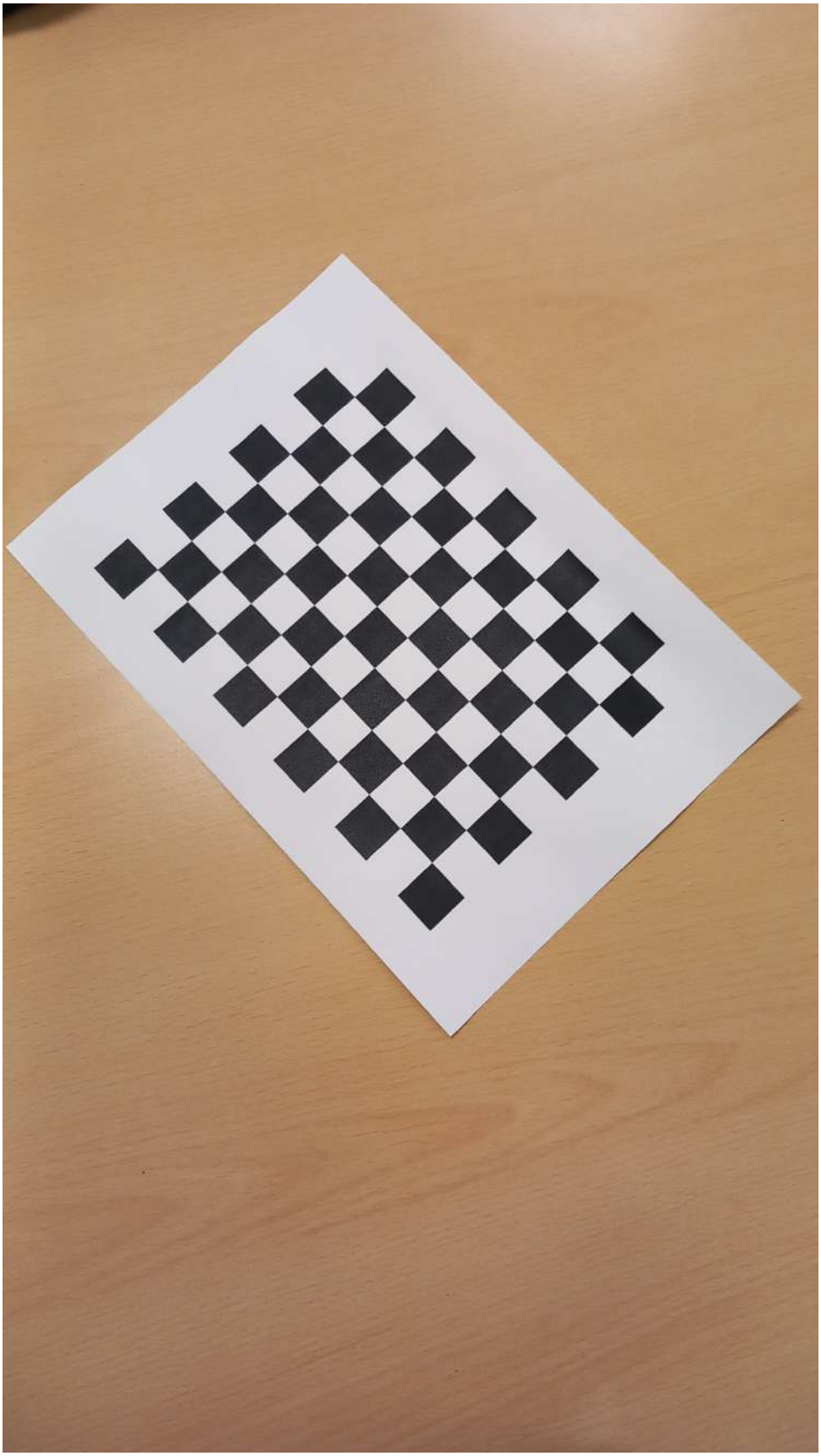


Figure 17: image 16

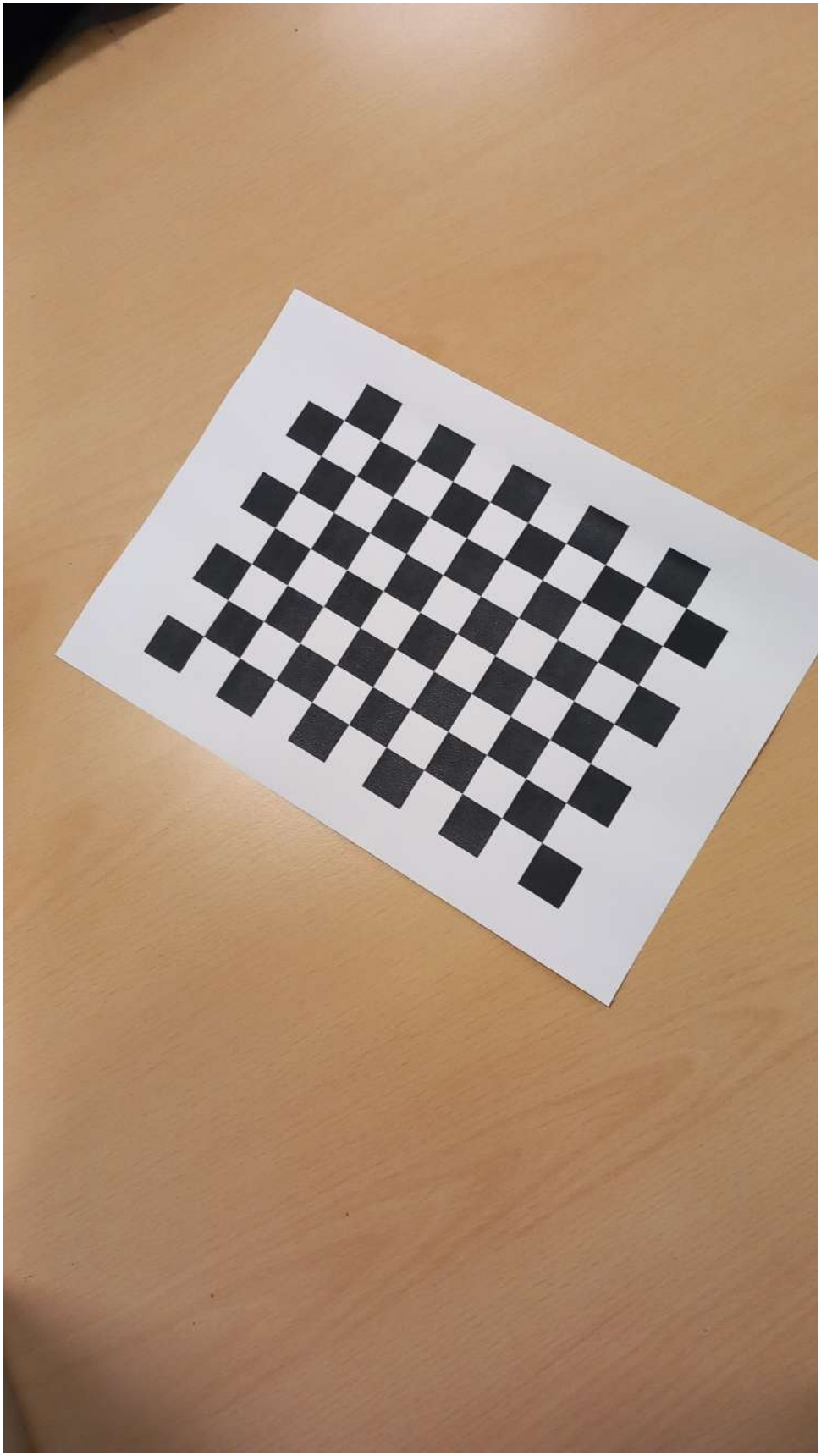


Figure 18: image 17

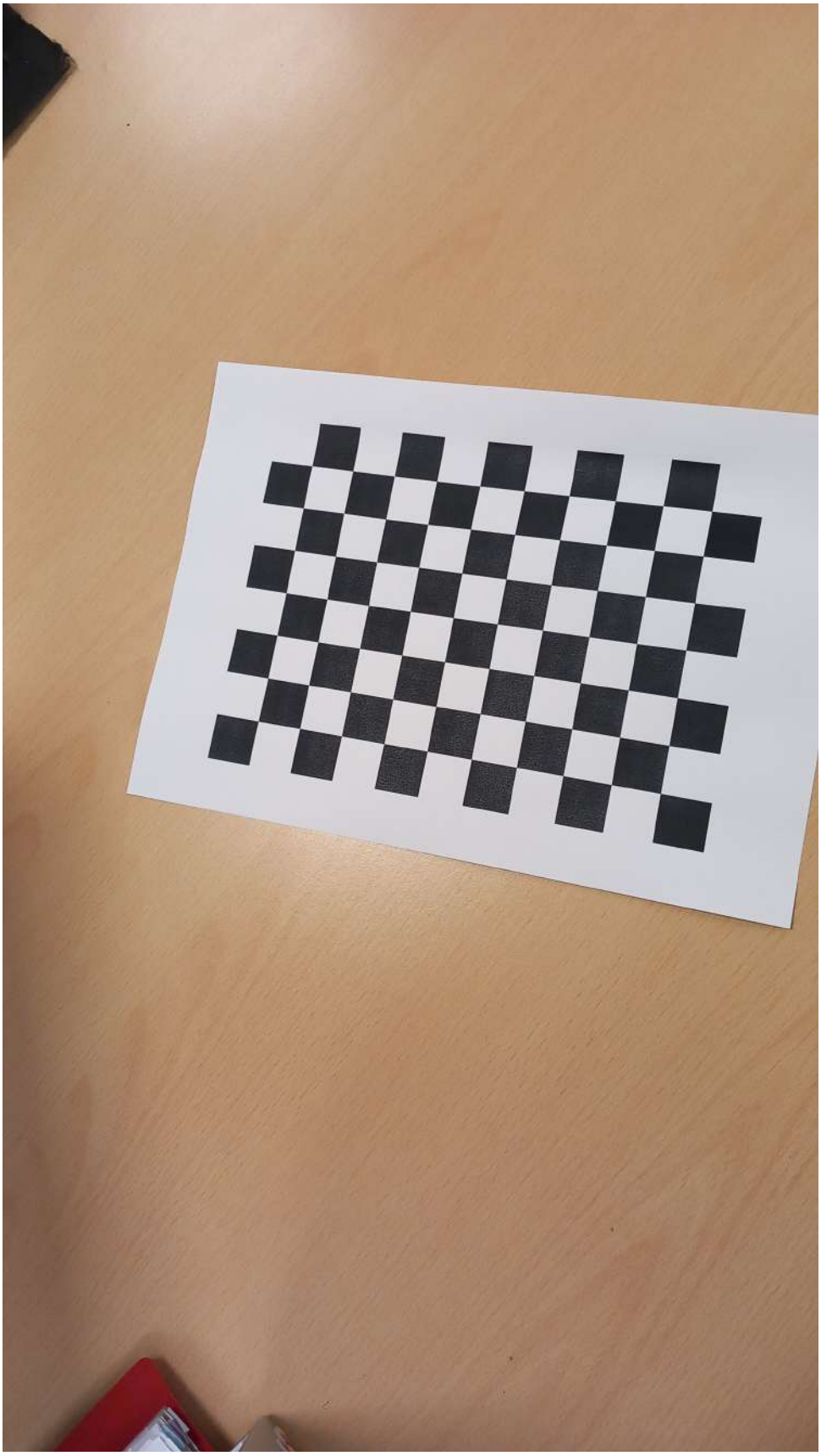


Figure 19: image 18

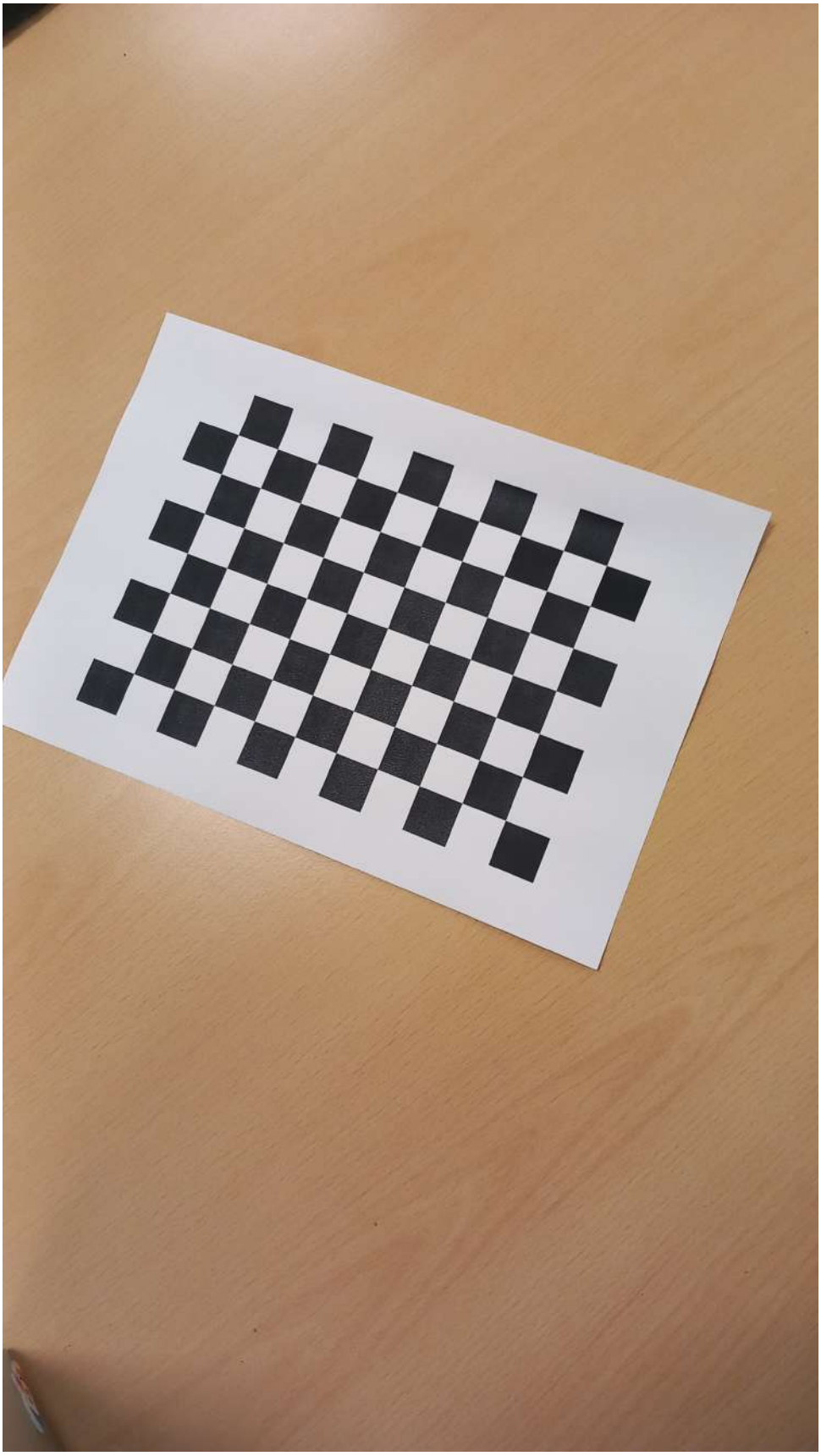


Figure 20: image 19

2.5 Adding radial distortion to the calibration procedure

1. We obtained a first estimate of k_1 and k_2 by solving the following linear system

$$\begin{cases} \hat{u} - u = (u - u_0) r_d^2 k_1 + (u - u_0) r_d^4 k_2 \\ \hat{v} - v = (v - v_0) r_d^2 k_1 + (v - v_0) r_d^4 k_2 \end{cases}$$

   where:

   - $\hat{u}$ and $\hat{v}$ are the distorted coordinates with `findChessboardCorners()`
   - u and v are the ideal coordinates obtained from the projection of the $3D$ coordinates of the corners through the perspective projection matrix P
   - r_d^2 is defined as

$$r_d^2 = \left(\frac{u - u_0}{\alpha_u} \right)^2 + \left(\frac{v - v_0}{\alpha_v} \right)^2$$

   The solution is found by stacking the equations for several correspondences and then using least squares applied to the system rewritten in the form

$$A \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = b$$

2. With the first estimate of k_1 and k_2 we performed the compensation of the radial distortion in a cycle that:

   - solves the following system of equations by using Newton's method (the equations are much more, since several points have been used)

$$\begin{cases} \hat{x} = x \left(1 + k_1 \left(x^2 + y^2\right) + k_2 \left(x^4 + 2x^2 y^2 + y^4\right)\right) \\ \hat{y} = y \left(1 + k_1 \left(x^2 + y^2\right) + k_2 \left(x^4 + 2x^2 y^2 + y^4\right)\right) \end{cases}$$

     where normalization and denormalization are applied when needed.

$$x = \frac{u - u_0}{\alpha_u} \quad \text{and} \quad y = \frac{v - v_0}{\alpha_v}.$$

   - With the new coordinates obtained, a new estimate of P and $k1, k1$ is obtained. The measured ones will be used only to compute the total reprojection error.

2.6 Total Reprojection error with radial distortion compensation

The steps are the same as in **2.2**.

3 Implementation Choices

The description of the approach already describes in detail our approach, but some choices have been made:

1. Environment choice: for implementing all the tasks we used Python and executed the code in a Google Colab.

2. We chose the twelfth image for calculating the reprojection error in **2.2** and **2.6**

3. The images in **2.4** were takes using a smartphone, the checkerboard has a square size of 20.

4. In images, we used circles of different sizes to better display the error, otherwise it would be necessary to zoom in order to visualize differences.

5. Superimposing a parallelepiped onto the calibration plane:

   - set different sizes of the parallelepiped in the case of the assignment images and in the case of the images captured by the smartphone. In particular, for the assignment images:
     - width = 99
     - height = 44
     - depth = 60

     for the images captured by the smartphone:
     - width = 180
     - height = 80
     - depth = 60

6. Radial distortion compensation:

   - stopped iterations when changes in k_1 and k_2 fell below a predefined threshold set to e^{-6} or when the number of iterations has reached 10.

4 Results

4.1 Calibration Results

The calibration process yielded the different results in the case of its application using the assignment images and in the case of using the images captured by the smartphone.

4.1.1 Assignment images

1. The estimated K is:

$$K = \begin{bmatrix} \alpha_x & \alpha_x \cot(\theta) & c_x \\ 0 & \frac{\alpha_y}{\sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{1}$$

   where:

   - $\alpha_x = 1.75891885e+03$, $\frac{\alpha_y}{\sin\theta} = 1.75634860e+03$
   - $c_x = 6.21685121e+02$, $c_y = 4.96948995e+02$
   - $\alpha_x \cot\theta = -2.63962169e+00$

2. The estimated R and t for the first image (Image 0) are:

$$R_0 = \begin{bmatrix} 0.33675521 & -0.91233785 & -0.23288531 \\ -0.94043102 & -0.33816973 & -0.03508166 \\ -0.04674843 & 0.2308265 & -0.97187124 \end{bmatrix}$$

$$t_0 = \begin{bmatrix} 92.11347632 & 44.54928253 & 557.3506719 \end{bmatrix}$$

The rotation matrices and translation vectors for the other images are not reported for simplicity.

4.1.2 Smartphone images

1. The estimated K is:

$$K_{smartphone} = \begin{bmatrix} \alpha_x & \alpha_x \cot(\theta) & c_x \\ 0 & \frac{\alpha_y}{\sin\theta} & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2}$$

where:

- $\alpha_x = 2.98949969e+03$, $\frac{\alpha_y}{\sin\theta} = 2.99056105e+03$
- $c_x = 1.15620468e+03$, $c_y = 2.04529646e+03$
- $\alpha_x \cot(\theta) = -1.44217530e+00$

2. The estimated R and t for the first image (Image 0) are:

$$R_0 = \begin{bmatrix} 0.00177957 & 0.99971436 & -0.02383355 \\ 0.99986795 & -0.00139386 & 0.01619047 \\ 0.01615262 & -0.02385921 & -0.99958483 \end{bmatrix}$$

$$t_0 = \begin{bmatrix} -67.9840199 & -126.63102626 & 366.28420402 \end{bmatrix}$$

The rotation matrices and translation vectors for the other images are not reported for simplicity.

4.2 Reprojection Error

The reprojection error was calculated for both an assignment image and an image captured by the smartphone:

- the total reprojection error for the calibration image number 11 is $e = 2.5009802477307796$. To visually analyze the reprojection error, the measured and projected points are displayed on the original image (green circles represent the measured points while red circles represent the projected points):

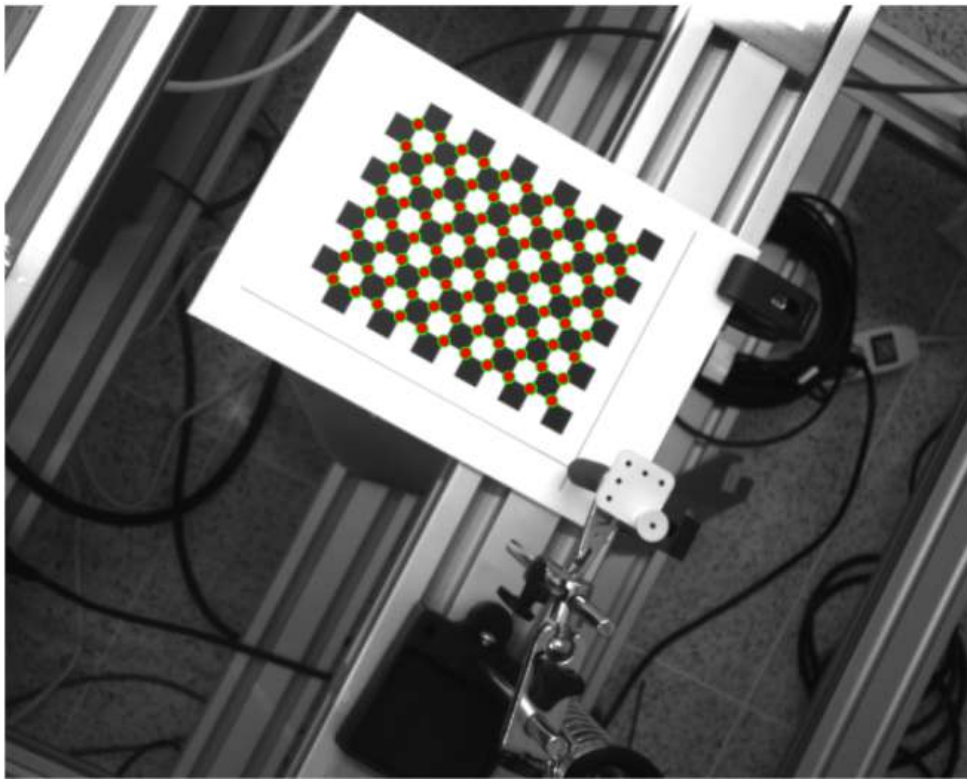


Figure 21: measured (green) and projected (red) points

- the total reprojection error for the smartphone image number 11 is $e = 501.22893059099073$
  To visually analyze the reprojection error, the measured and projected points are displayed on the original image (green circles represent the measured points while red circles represent the projected points):

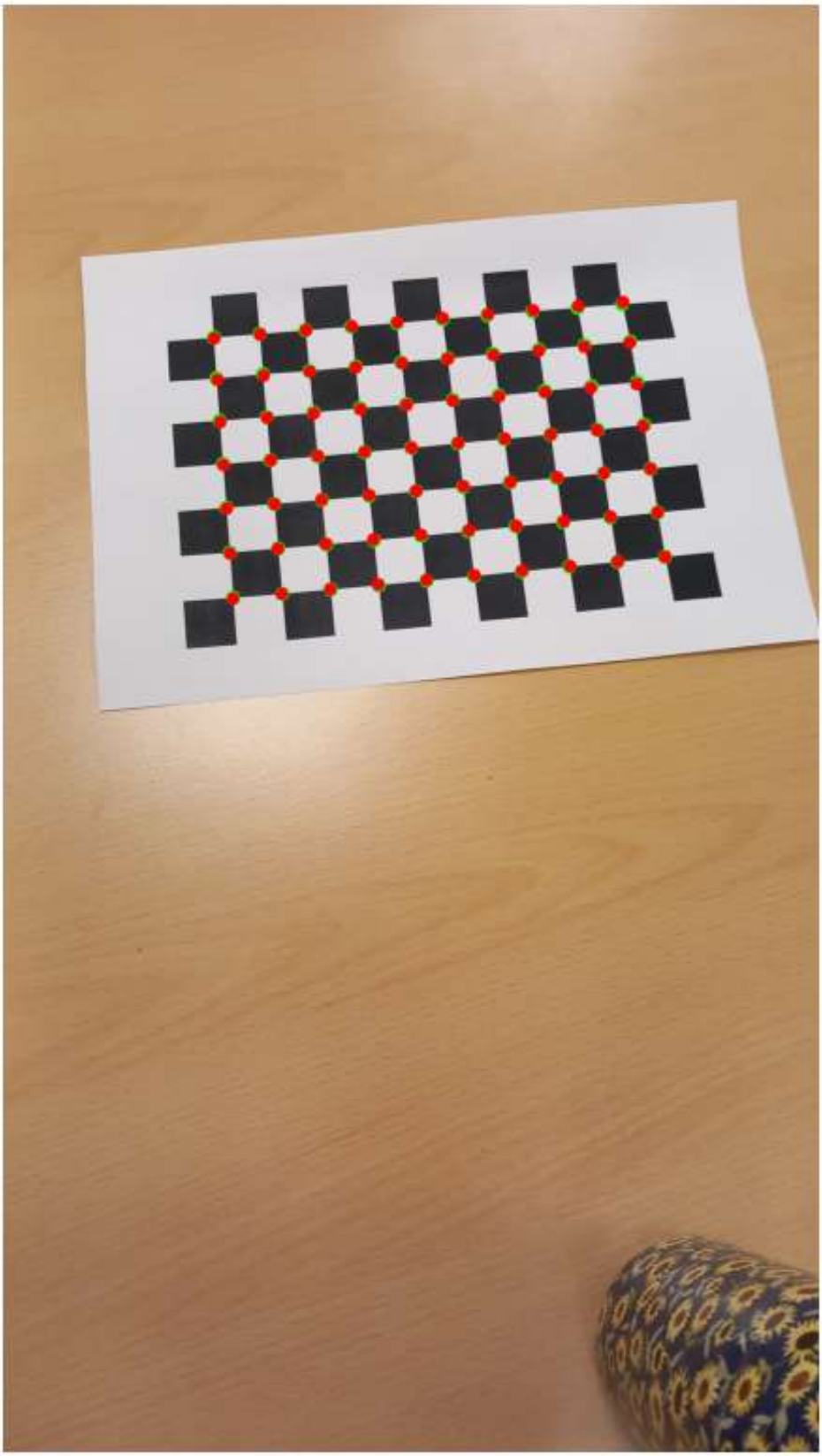


Figure 22: measured (green) and projected (red) points

4.3 3D Object Superimposition

4.3.1 Assignment Image

The following figure shows a 3D parallelepiped superimposed onto the calibration plane: the superimposition has been performed for every calibration images but for simplicity here is reported only one:

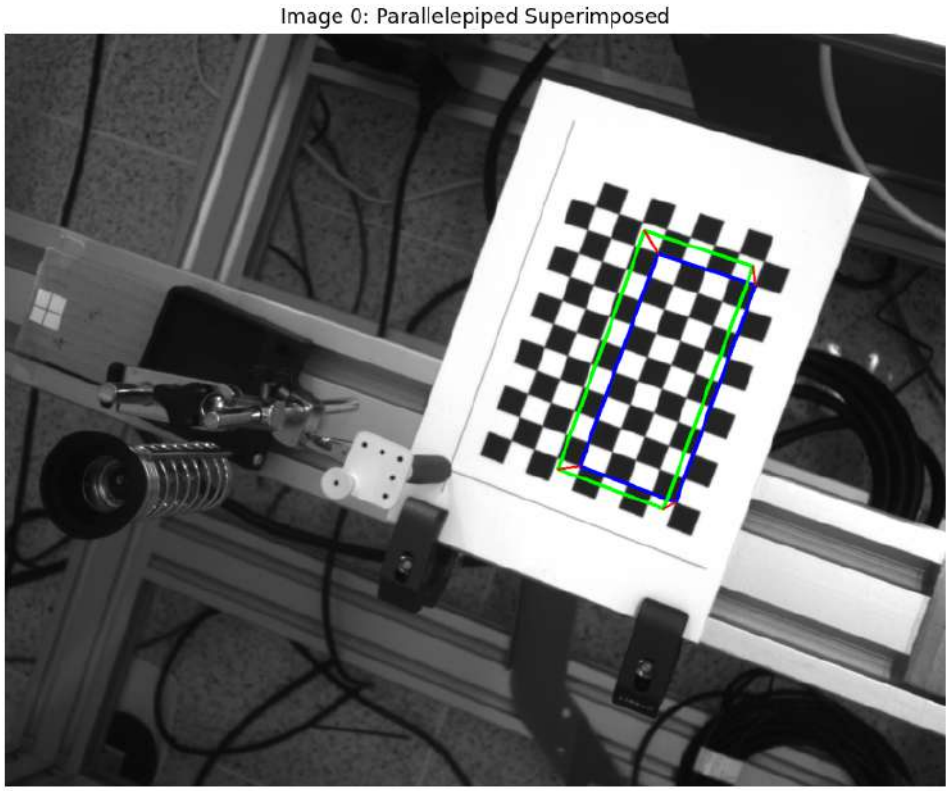


Image 1: Parallelepiped Superimposed

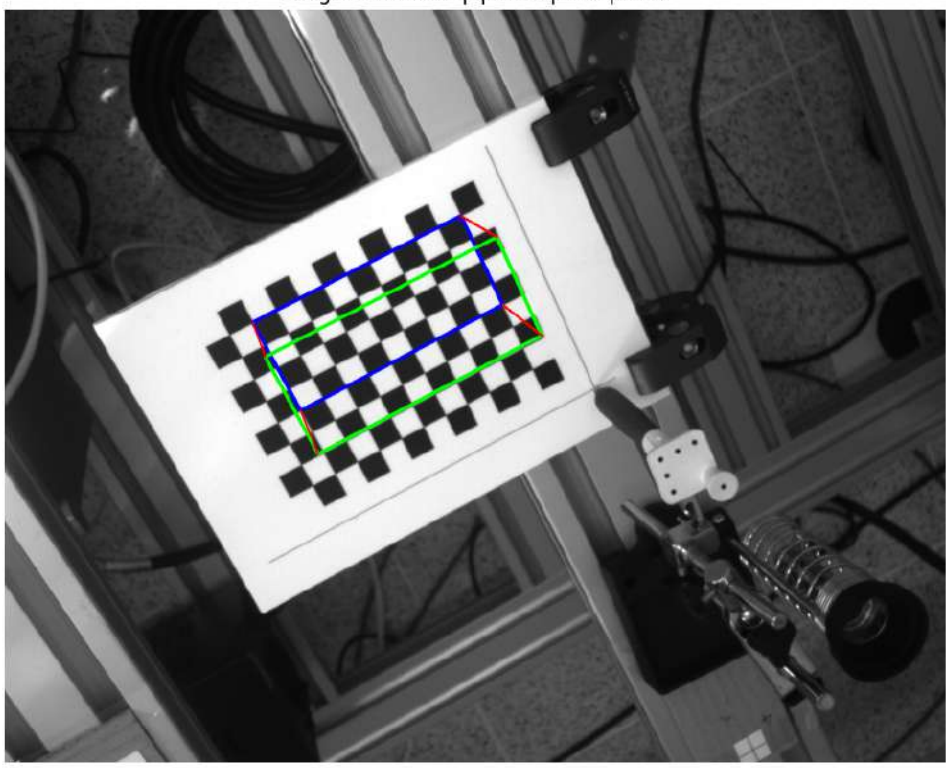


Image 2: Parallelepiped Superimposed

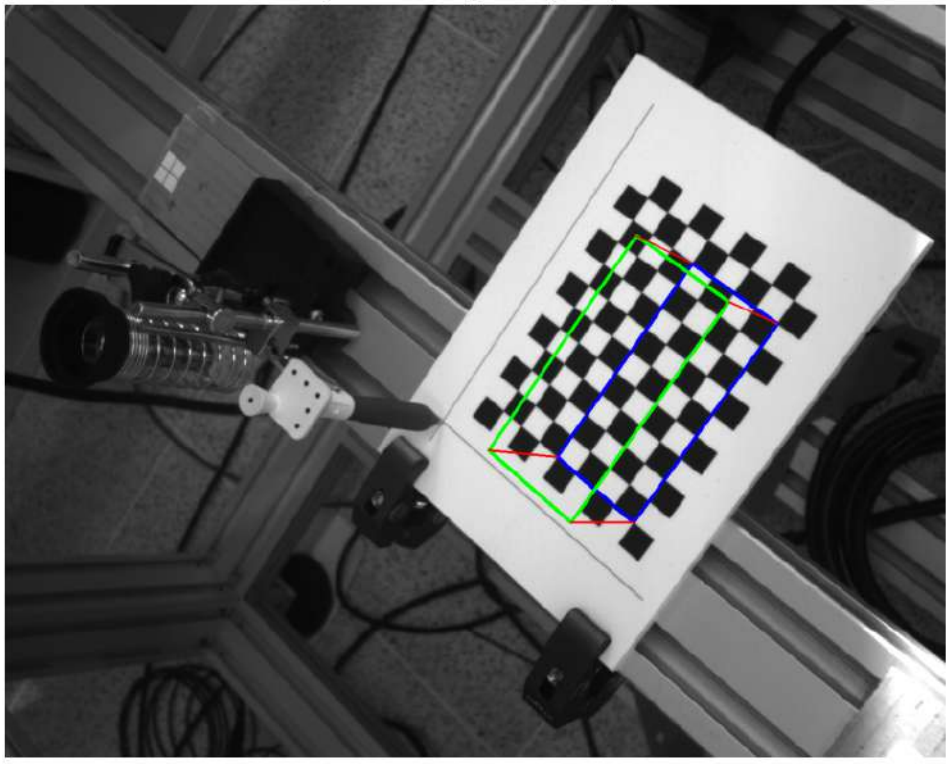


Image 3: Parallelepiped Superimposed

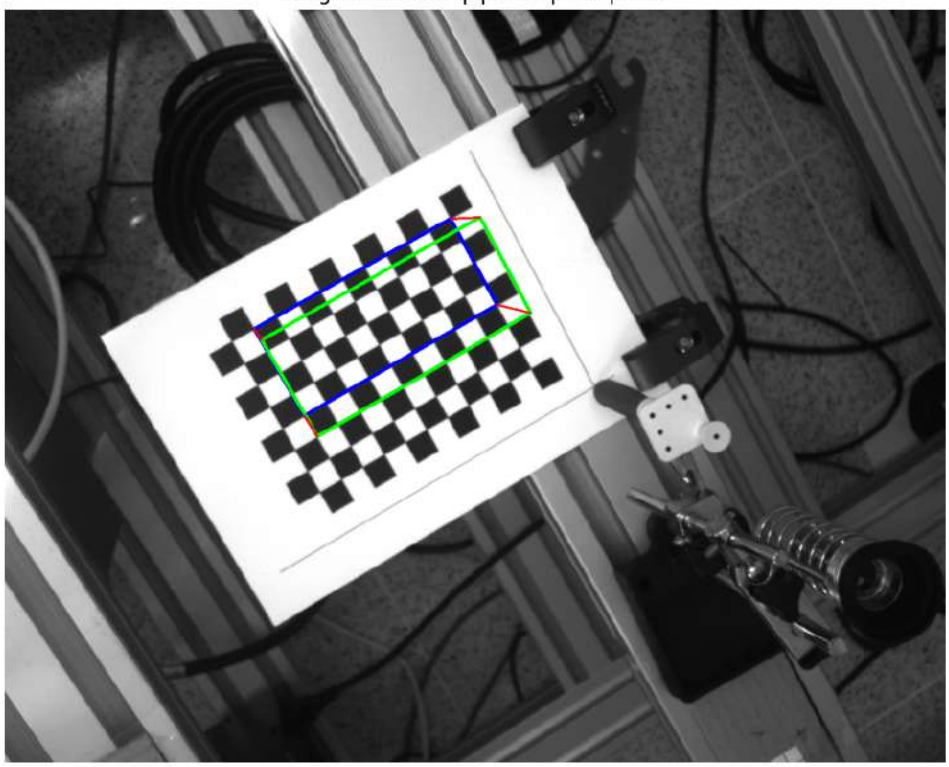


Image 4: Parallelepiped Superimposed

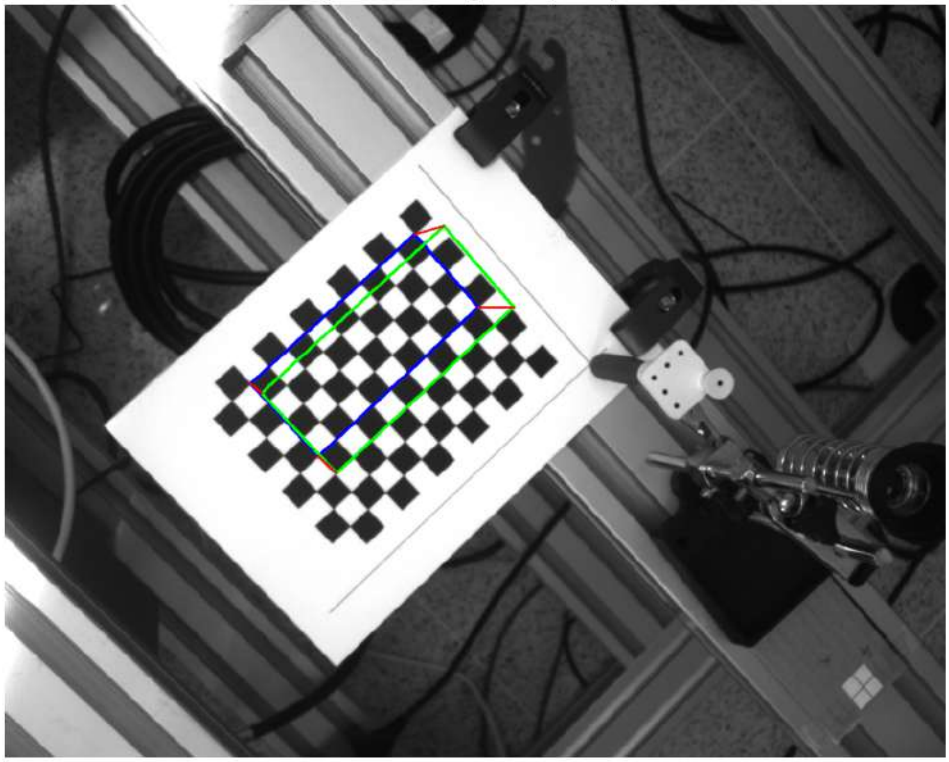


Image 5: Parallelepiped Superimposed

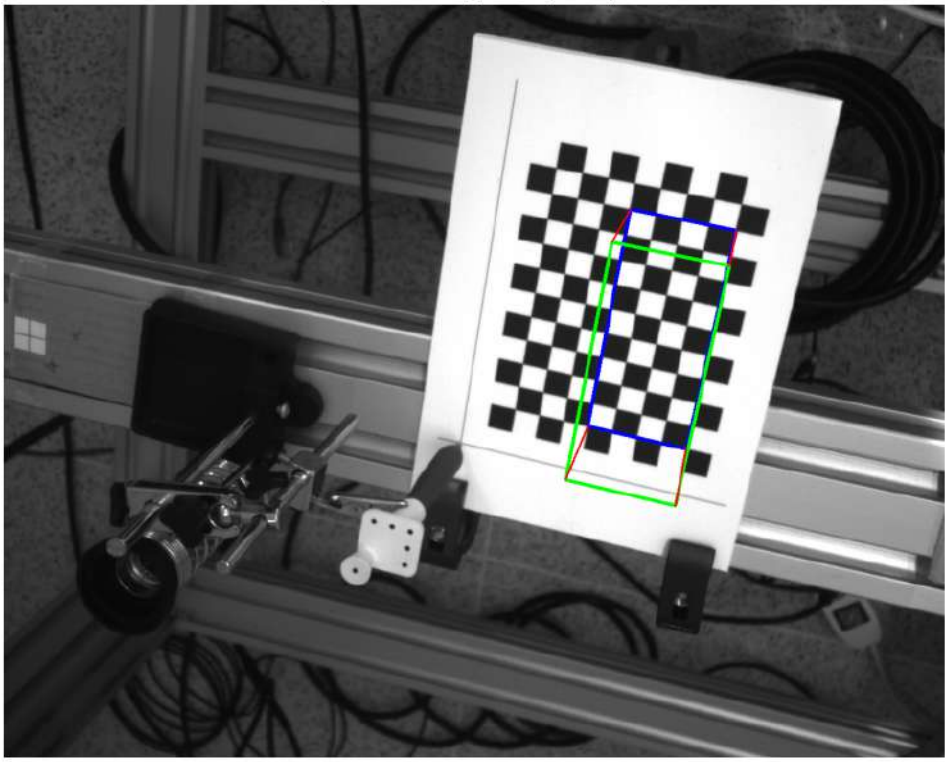


Image 6: Parallelepiped Superimposed

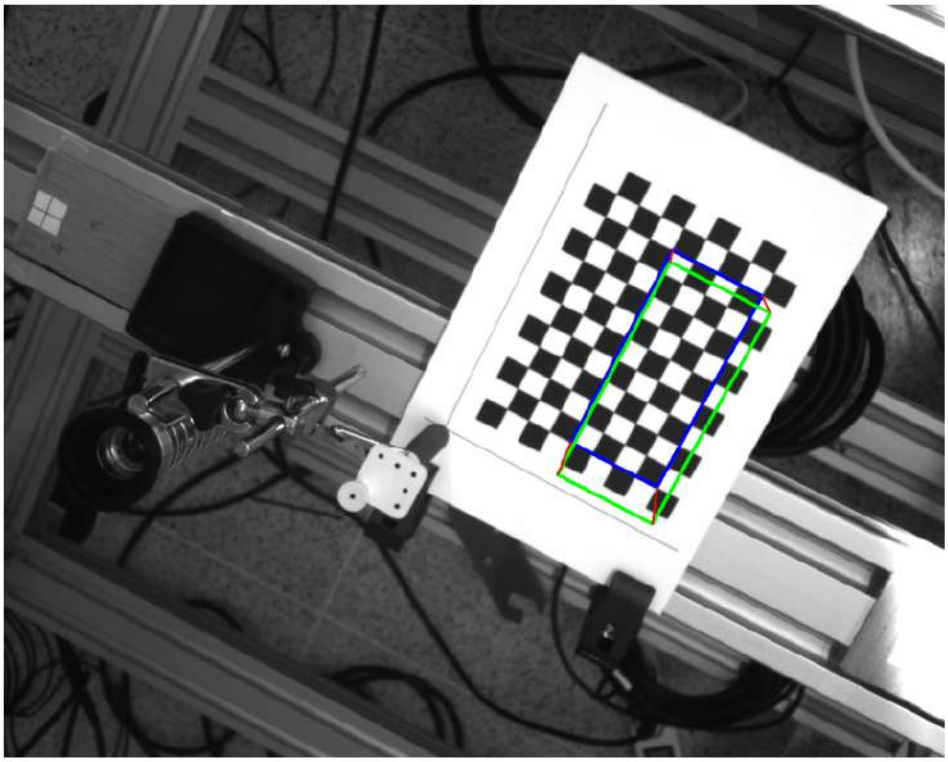


Image 7: Parallelepiped Superimposed

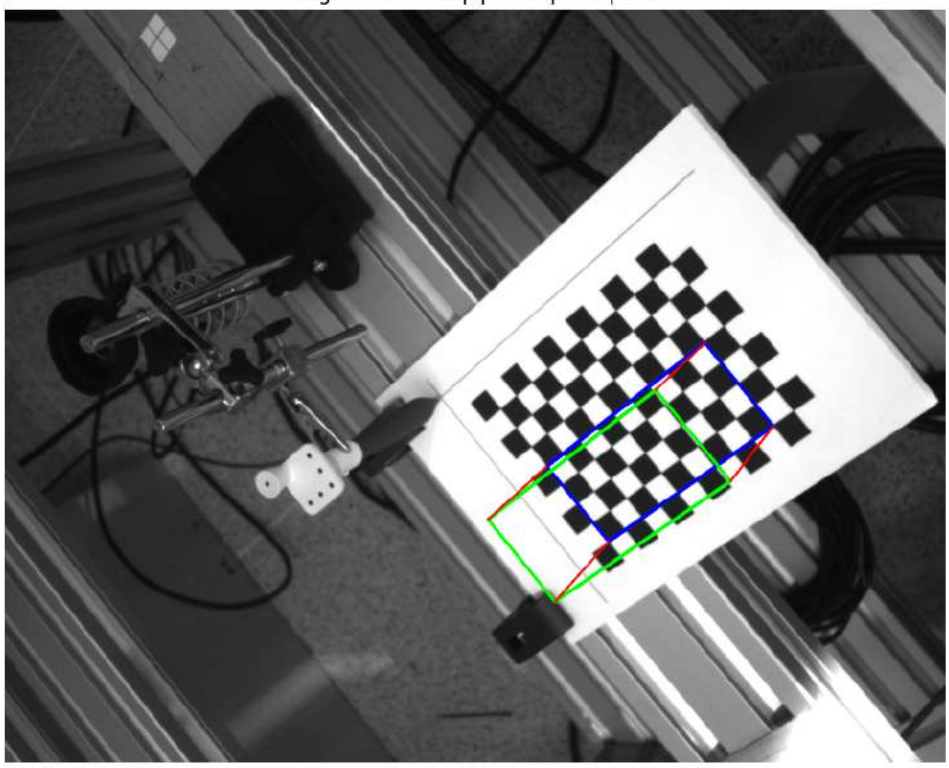


Image 8: Parallelepiped Superimposed

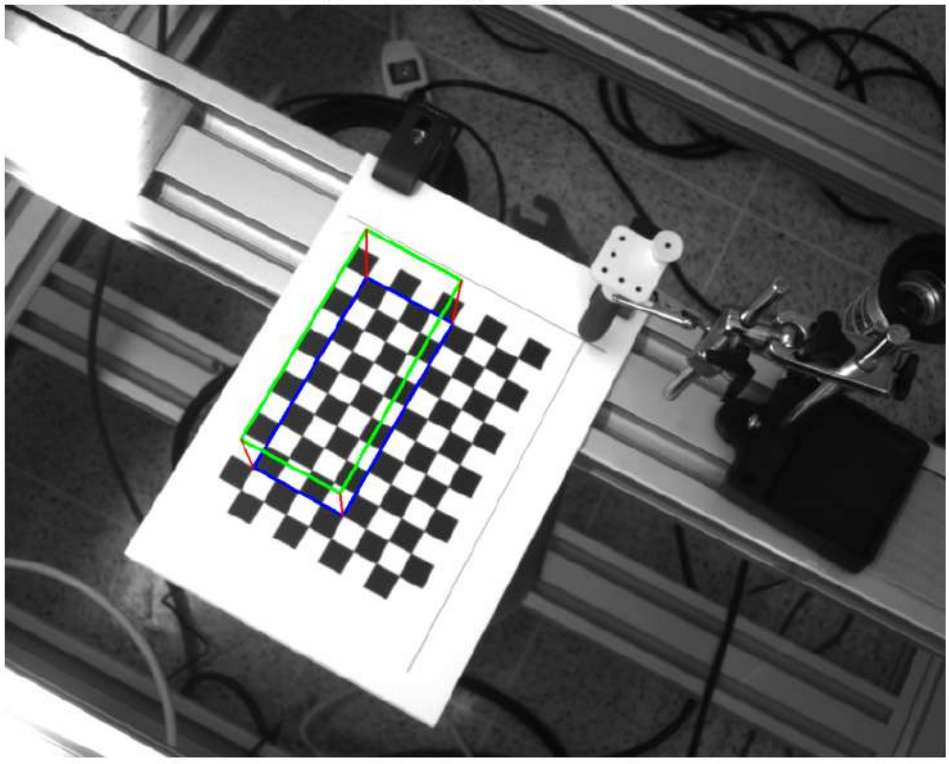


Image 9: Parallelepiped Superimposed

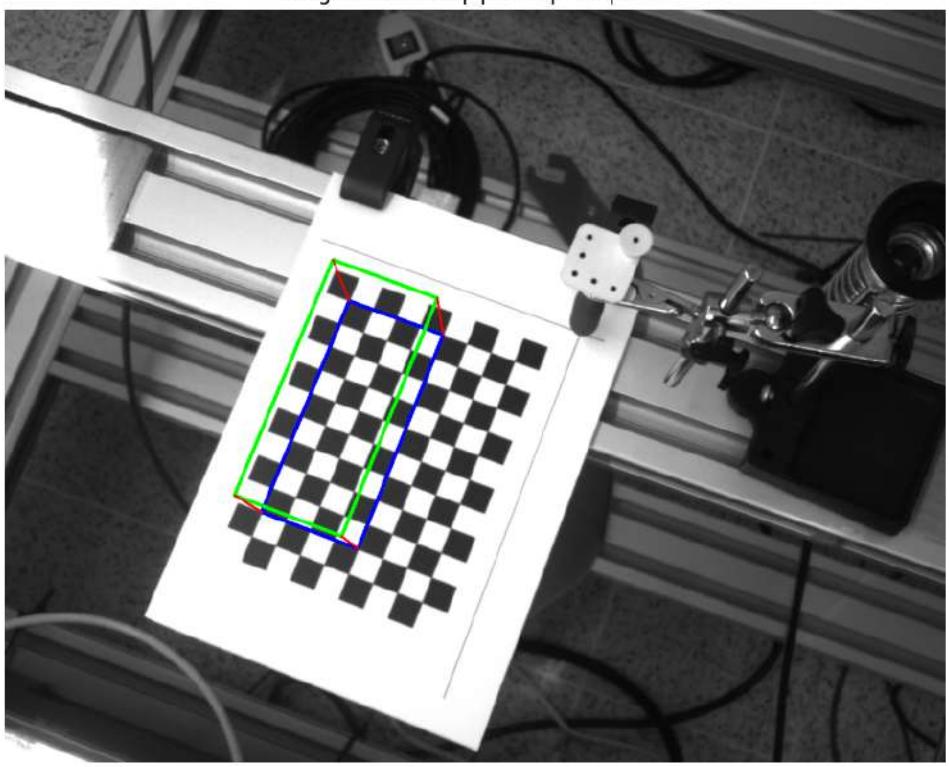


Image 10: Parallelepiped Superimposed

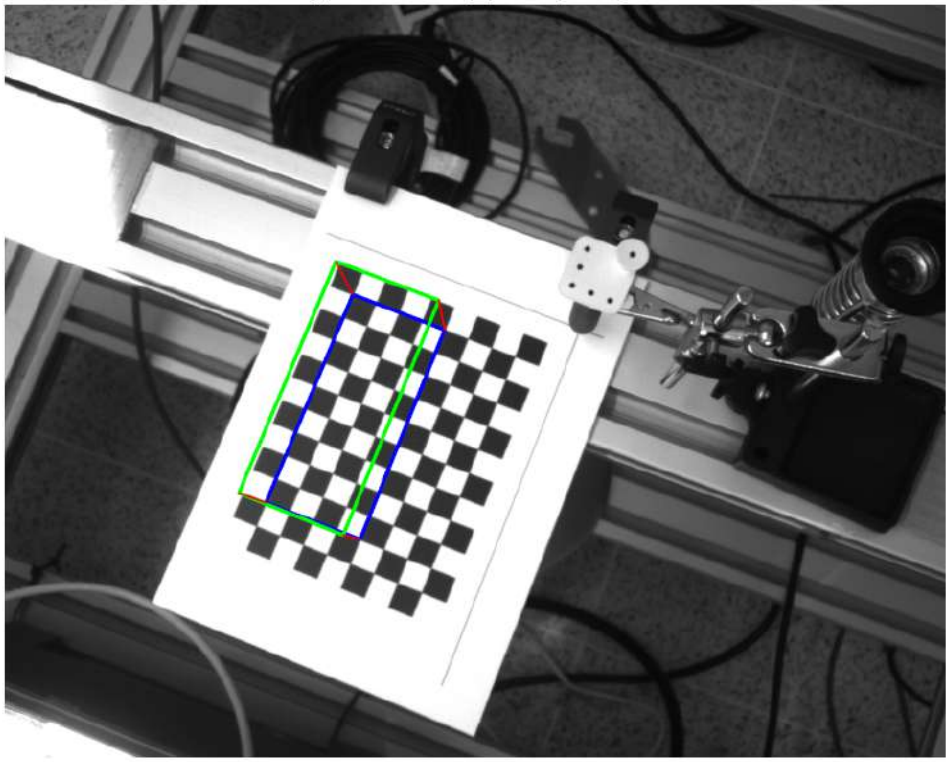


Image 11: Parallelepiped Superimposed

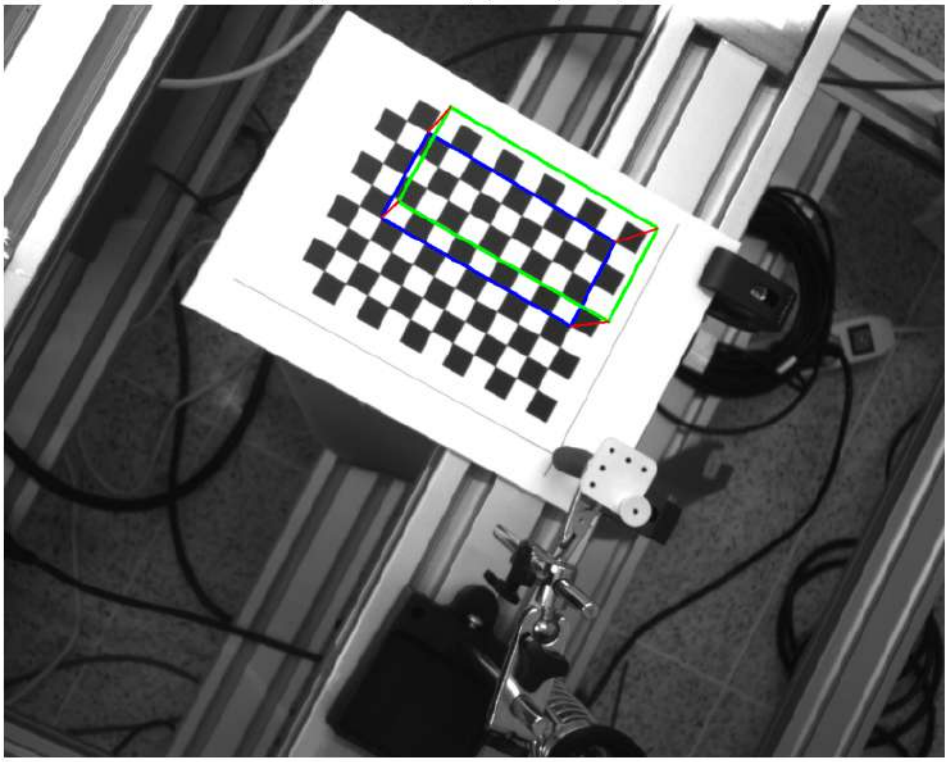


Image 12: Parallelepiped Superimposed

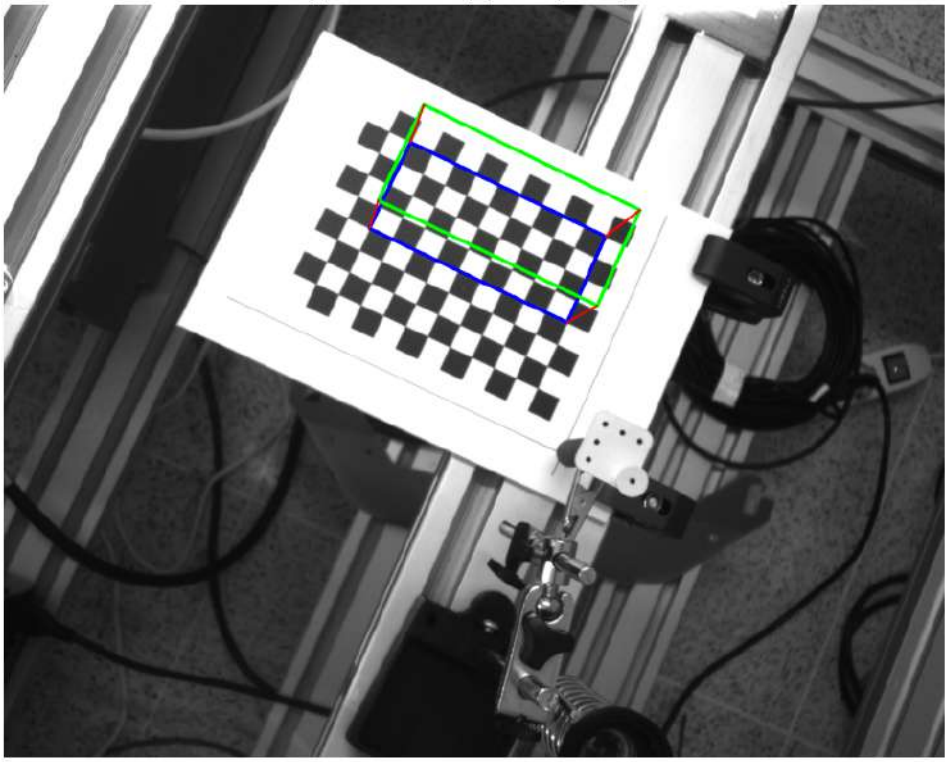


Image 13: Parallelepiped Superimposed

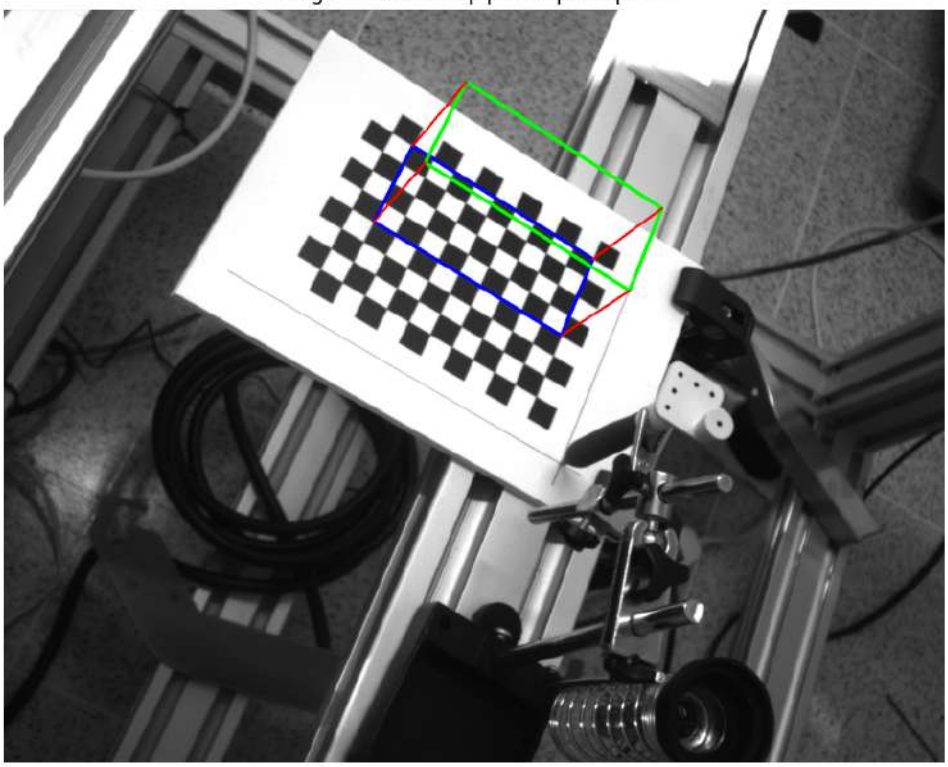


Image 14: Parallelepiped Superimposed

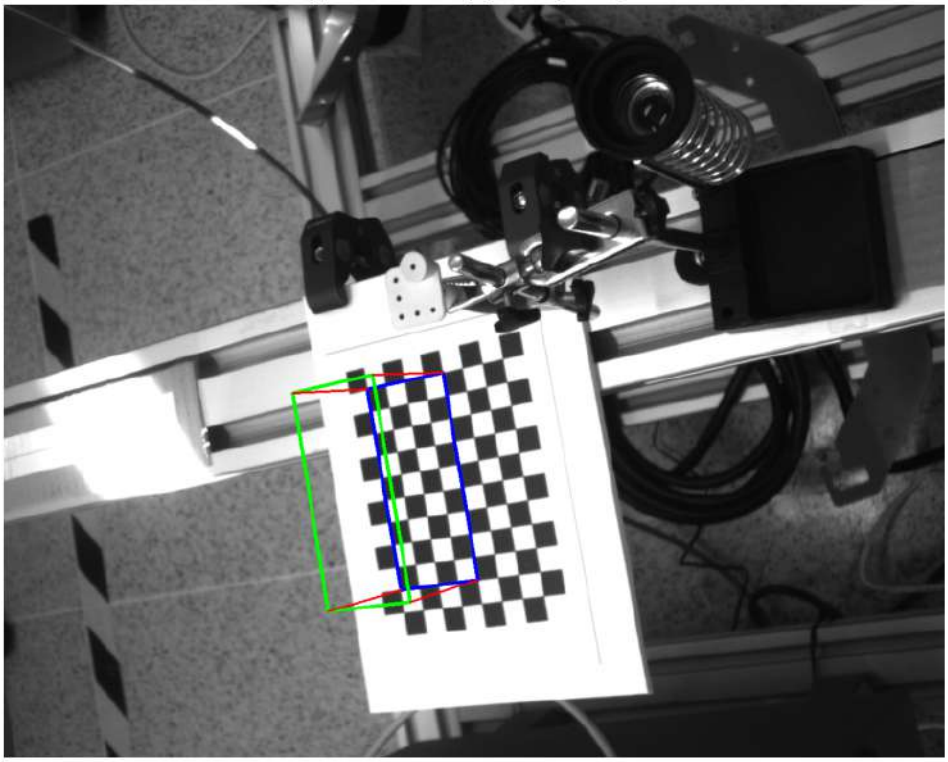


Image 15: Parallelepiped Superimposed

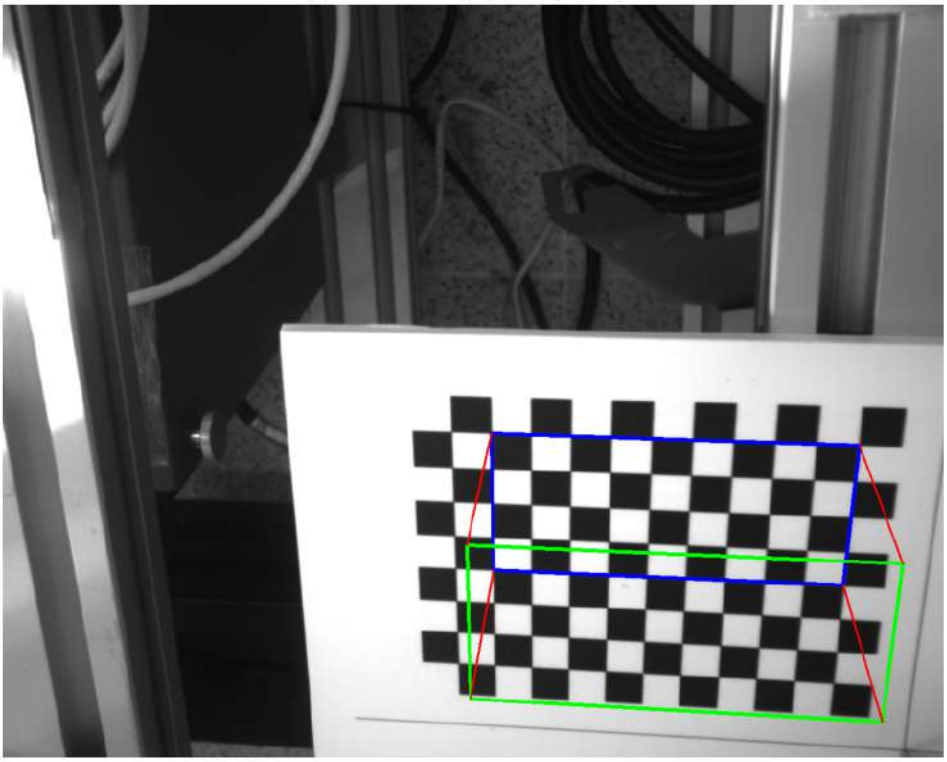


Image 16: Parallelepiped Superimposed

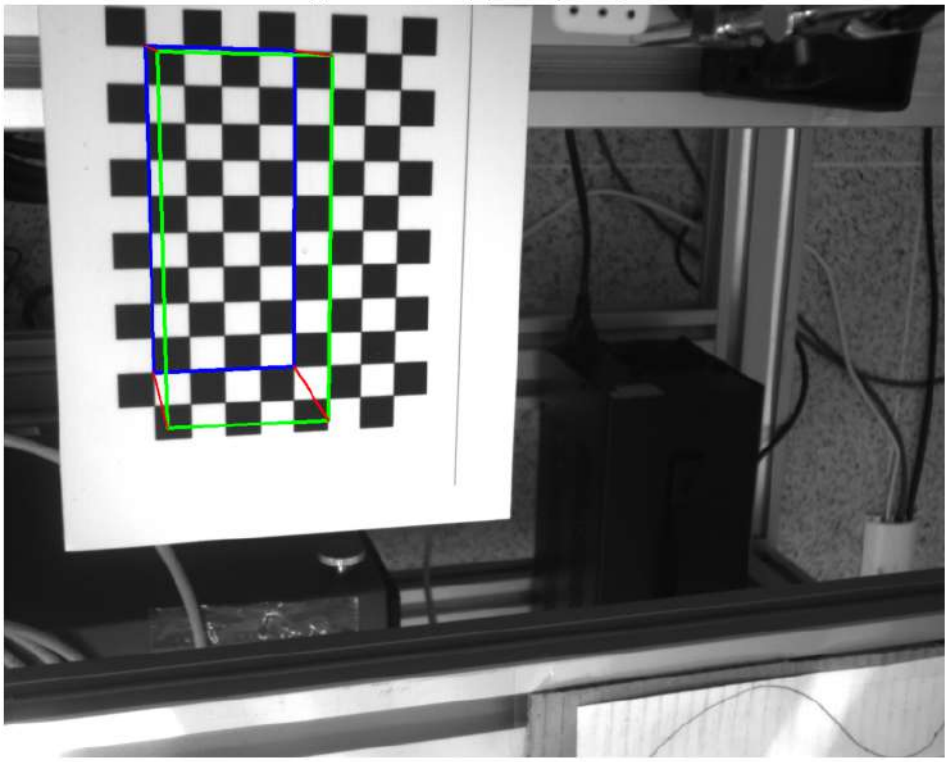


Image 17: Parallelepiped Superimposed

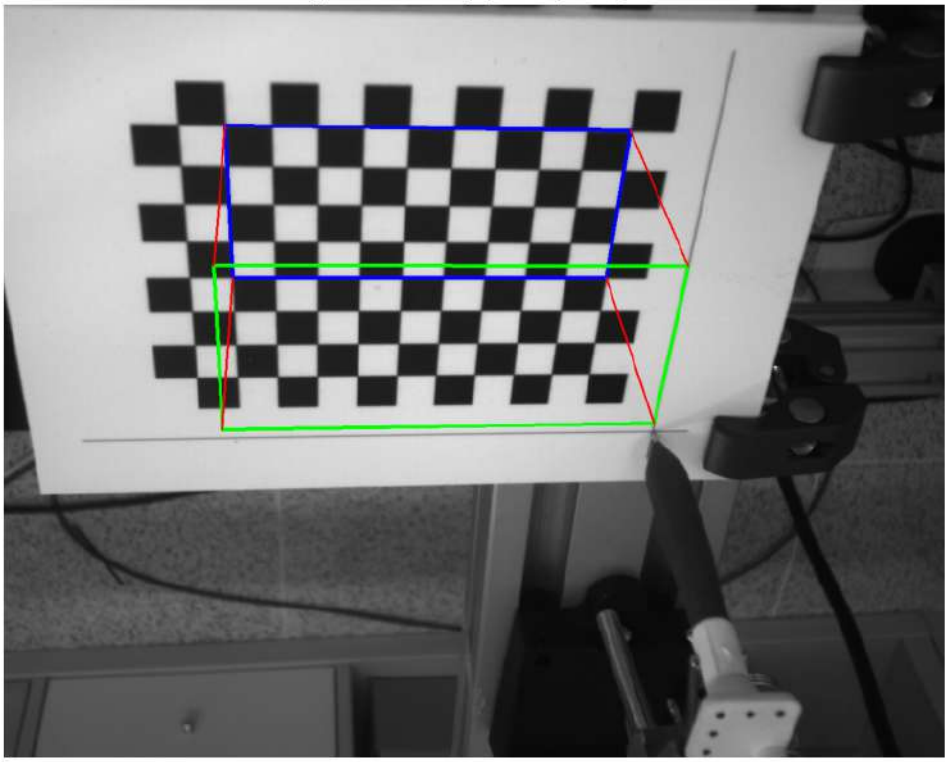


Image 18: Parallelepiped Superimposed

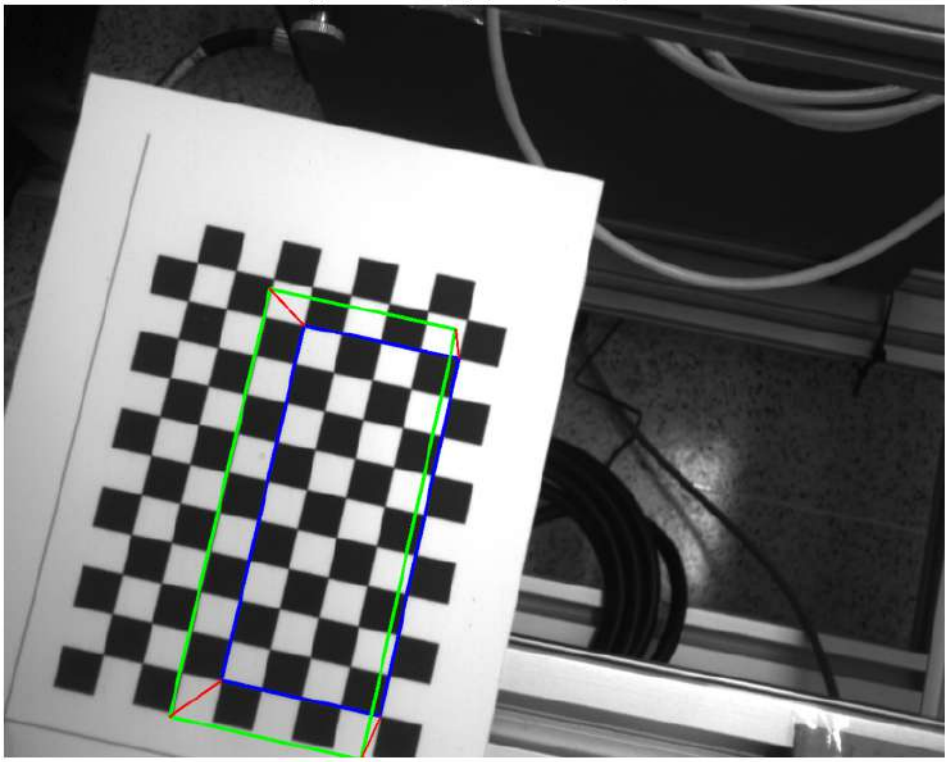
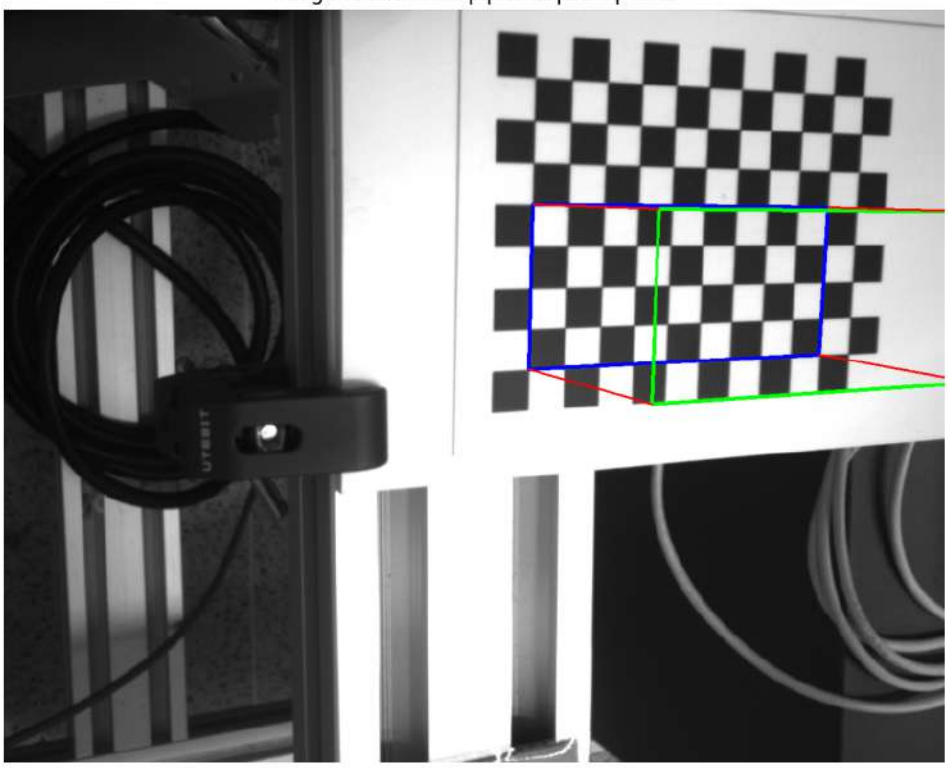


Image 19: Parallelepiped Superimposed



4.3.2 Smartphone Image

The following figure shows a 3D parallelepiped superimposed onto the calibration plane: the superimposition has been performed for every calibration images.

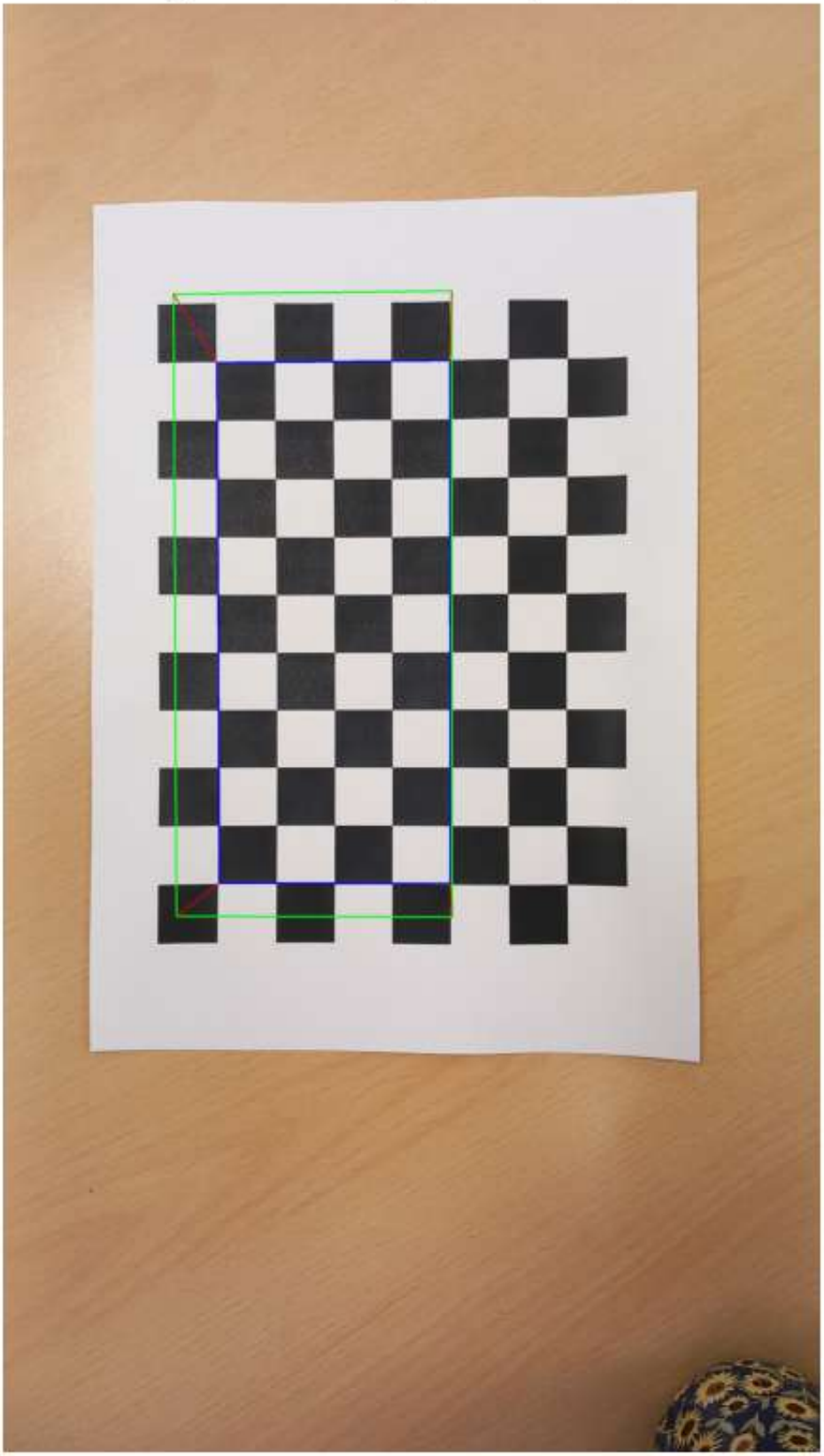


Image 0: Parallelepiped Superimposed

Image 1: Parallelepiped Superimposed

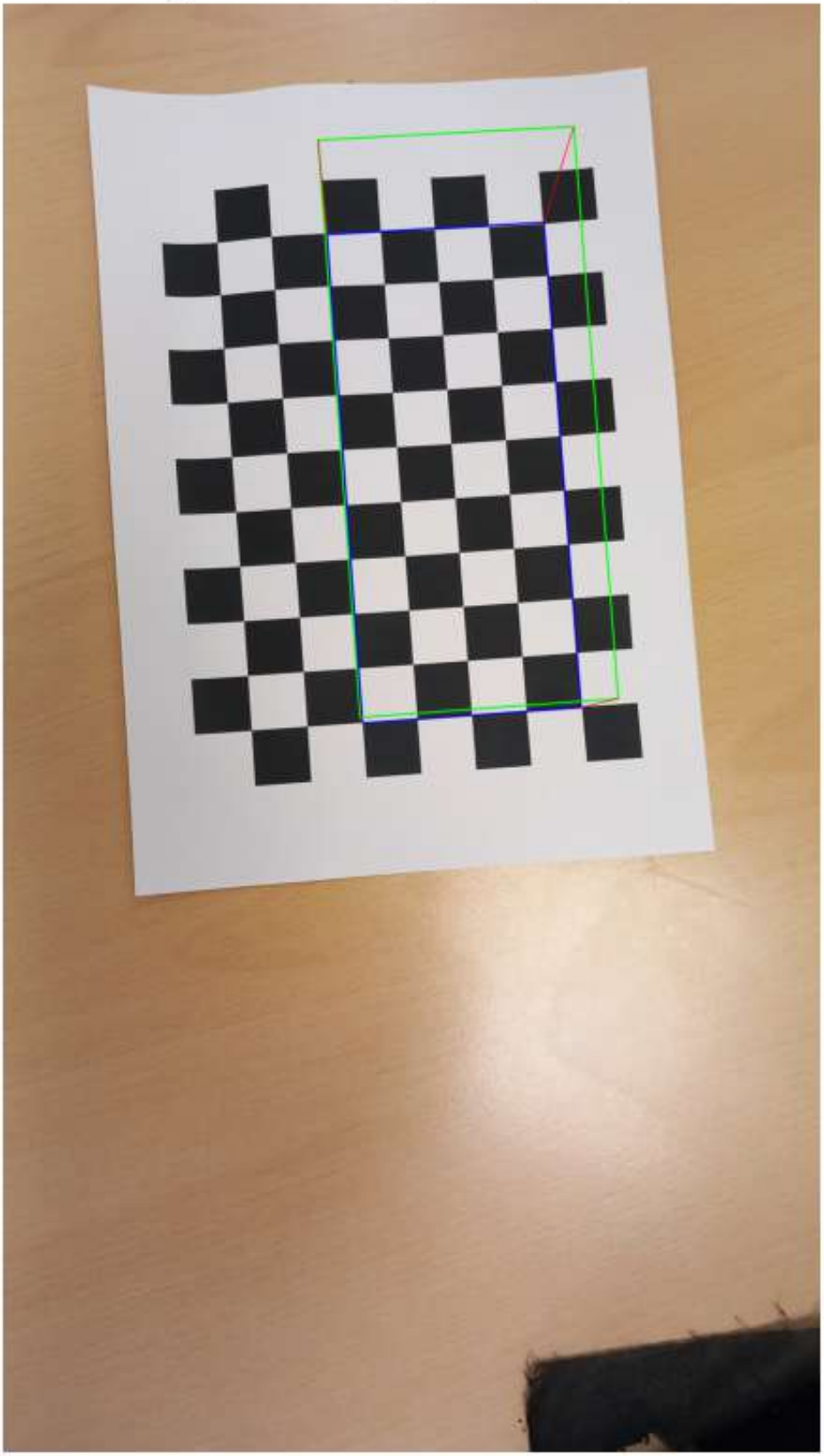


Image 2: Parallelepiped Superimposed

Image 3: Parallelepiped Superimposed

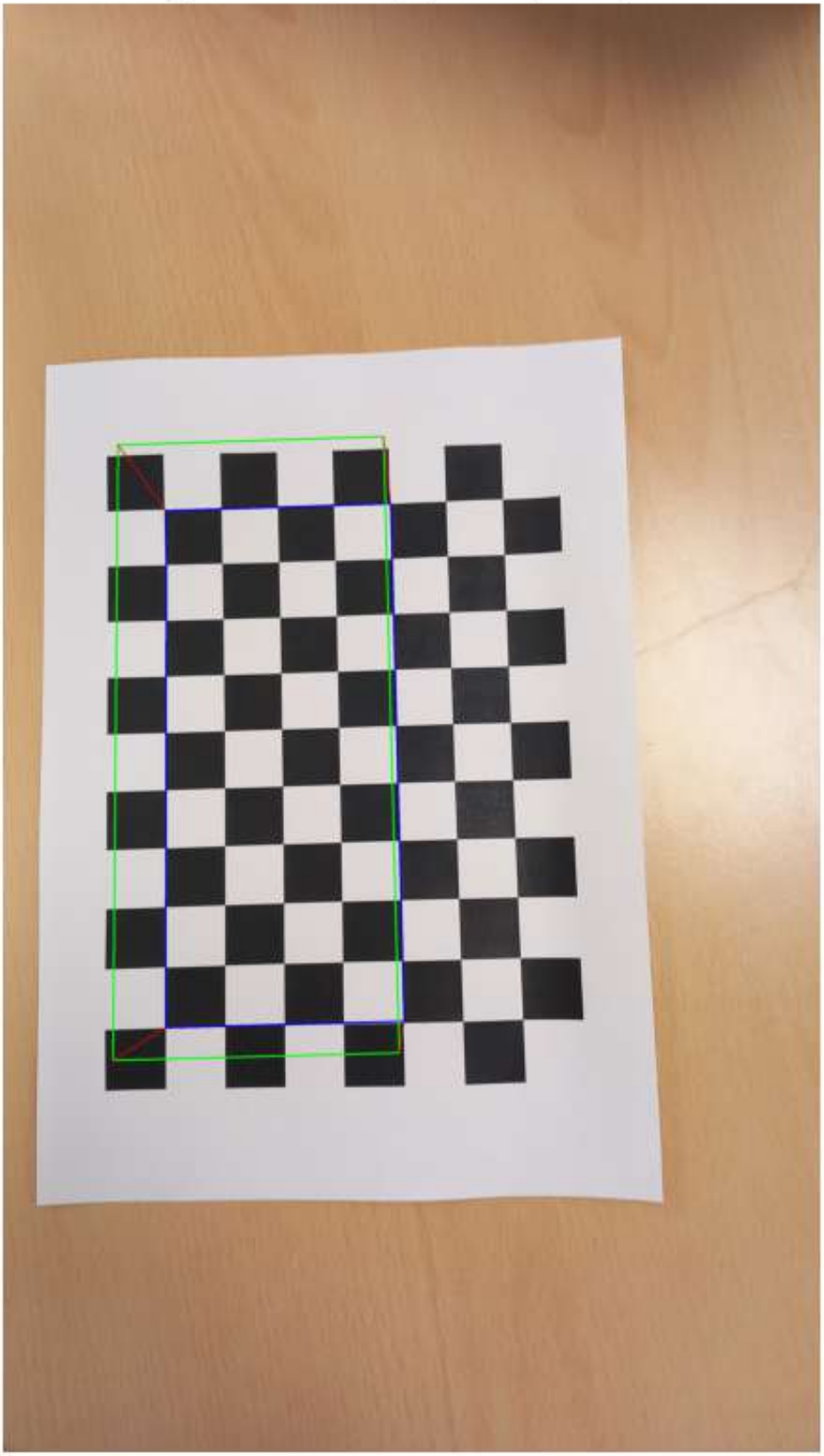


Image 4: Parallelepiped Superimposed

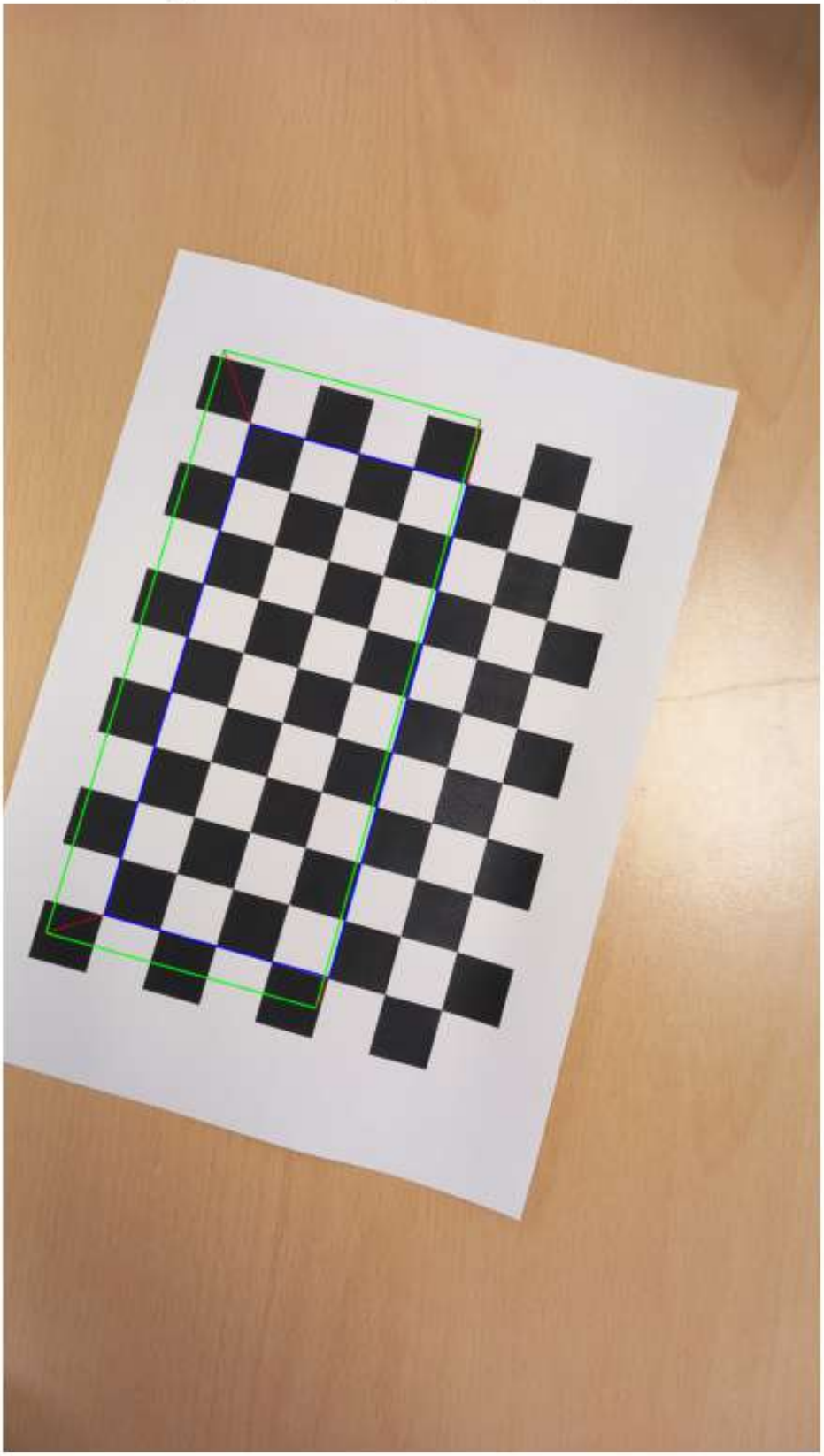


Image 5: Parallelepiped Superimposed

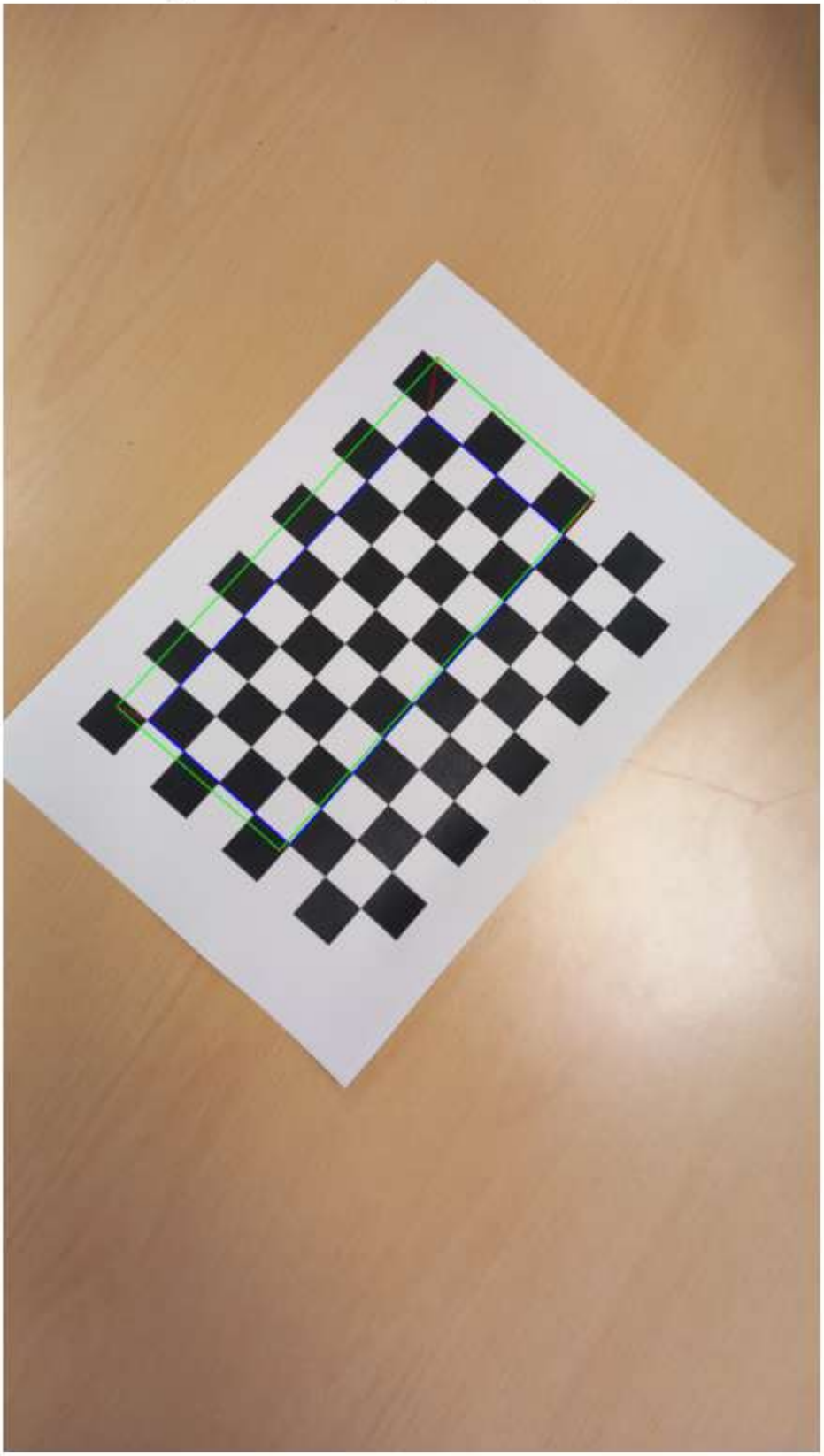


Image 6: Parallelepiped Superimposed

Image 7: Parallelepiped Superimposed

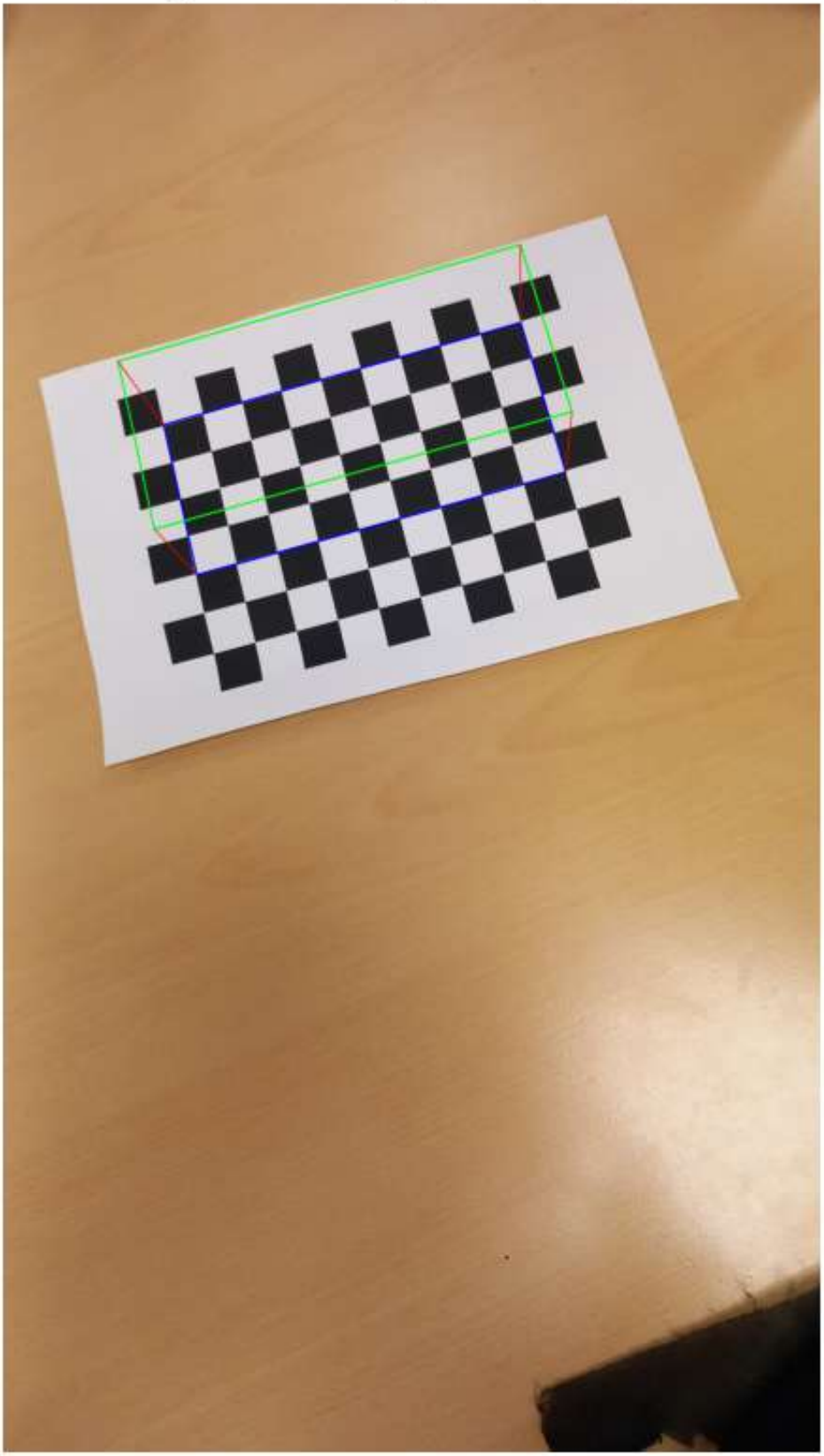


Image 8: Parallelepiped Superimposed

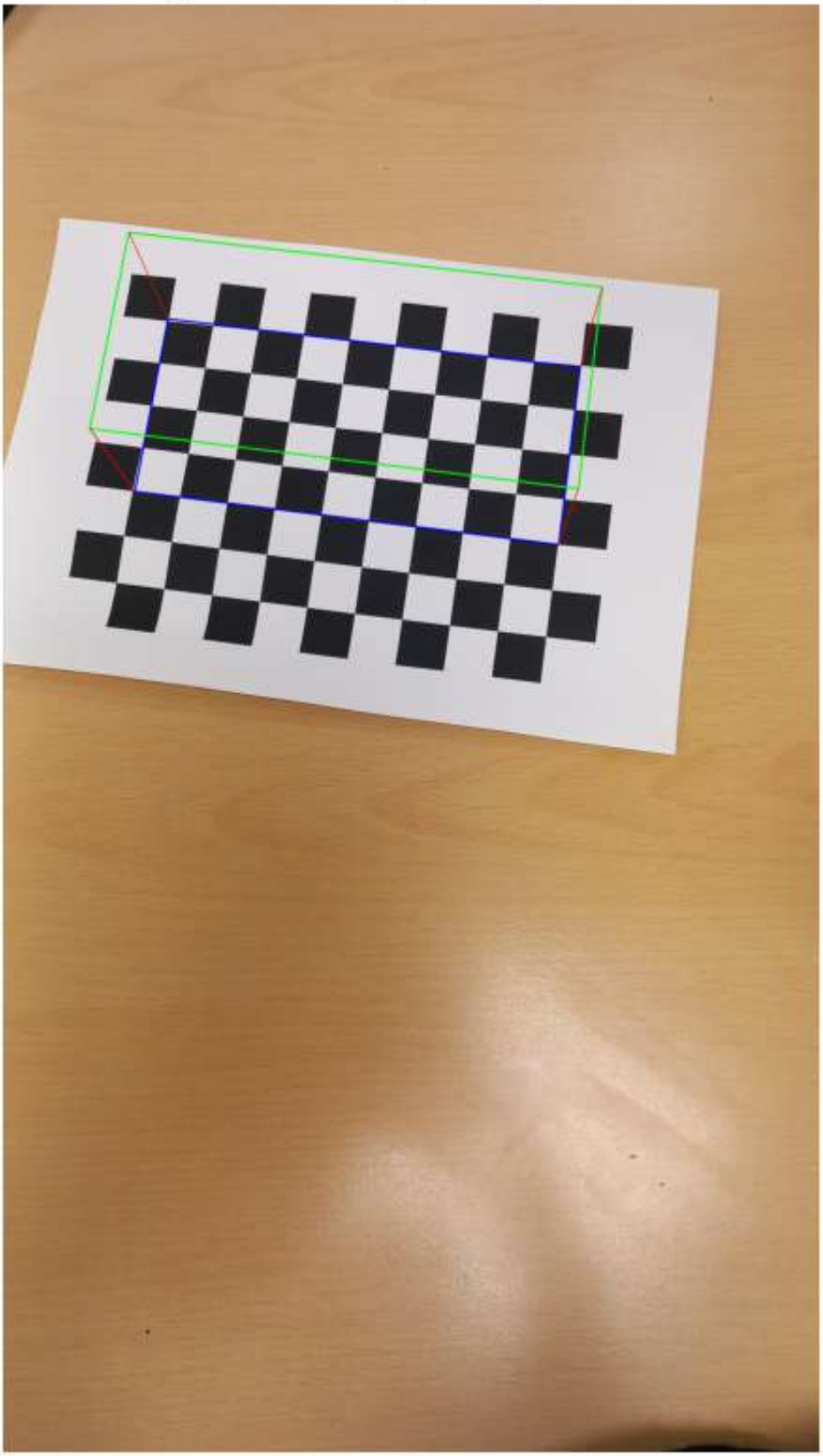


Image 9: Parallelepiped Superimposed

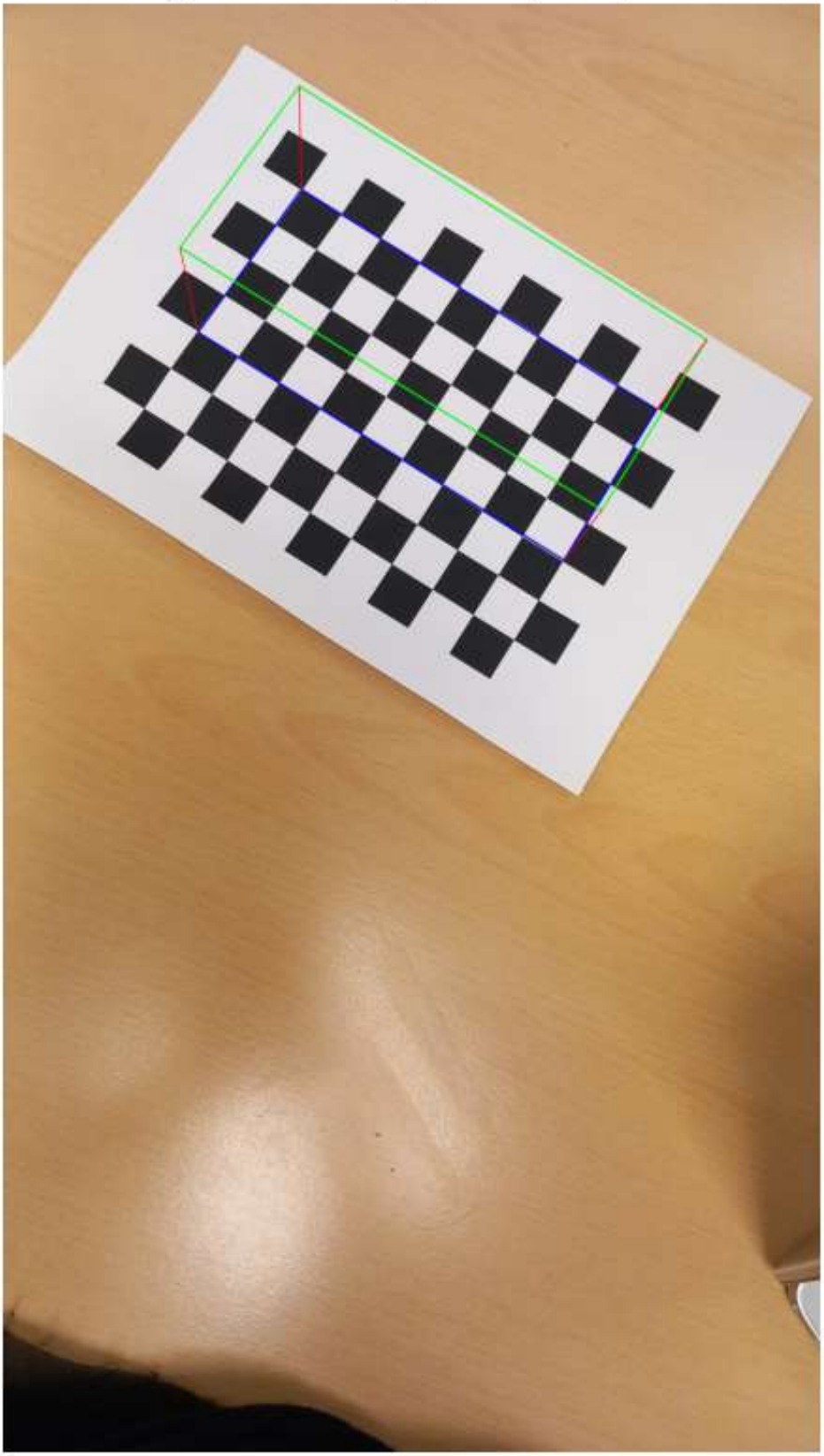


Image 10: Parallelepiped Superimposed

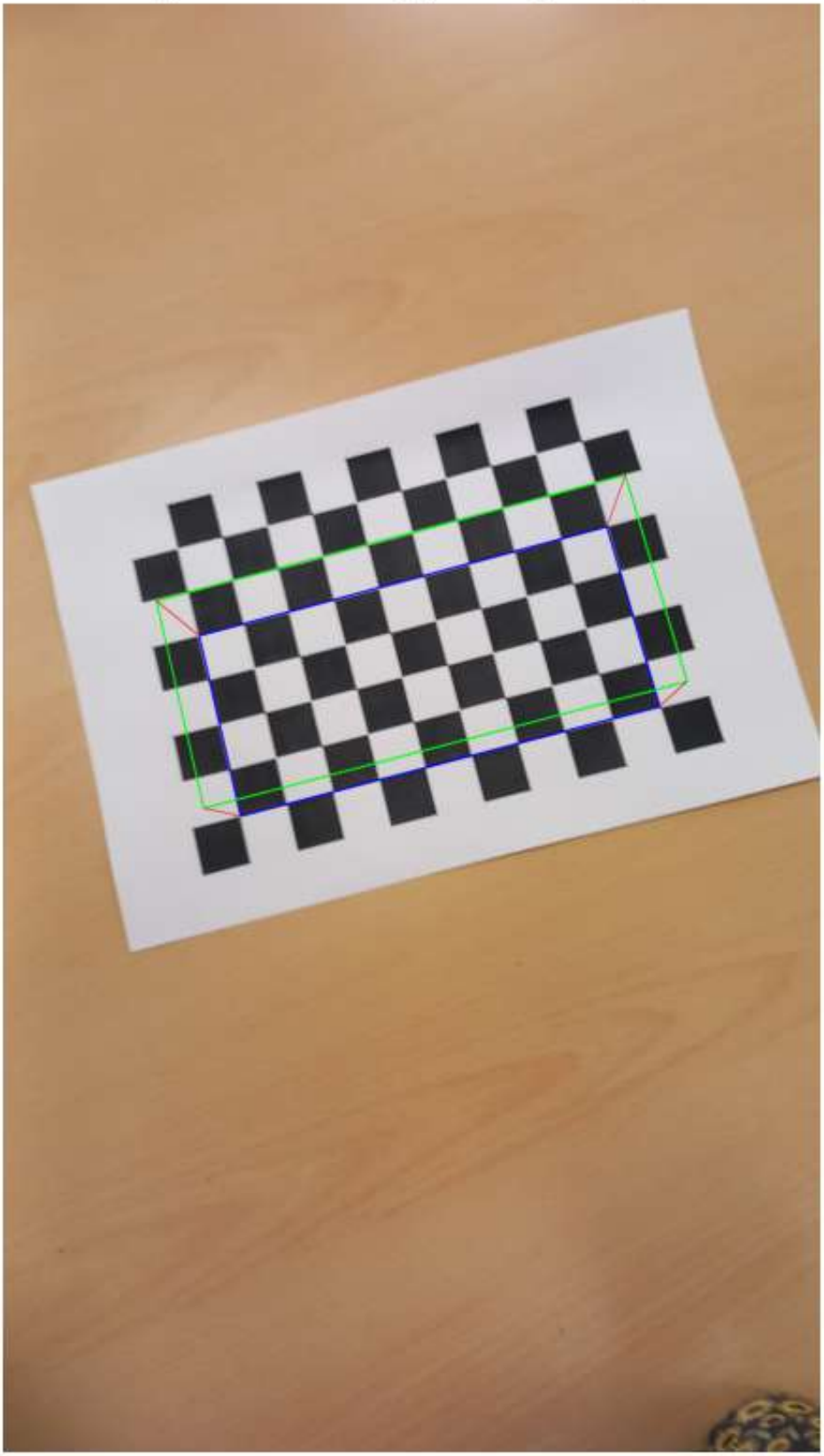


Image 11: Parallelepiped Superimposed

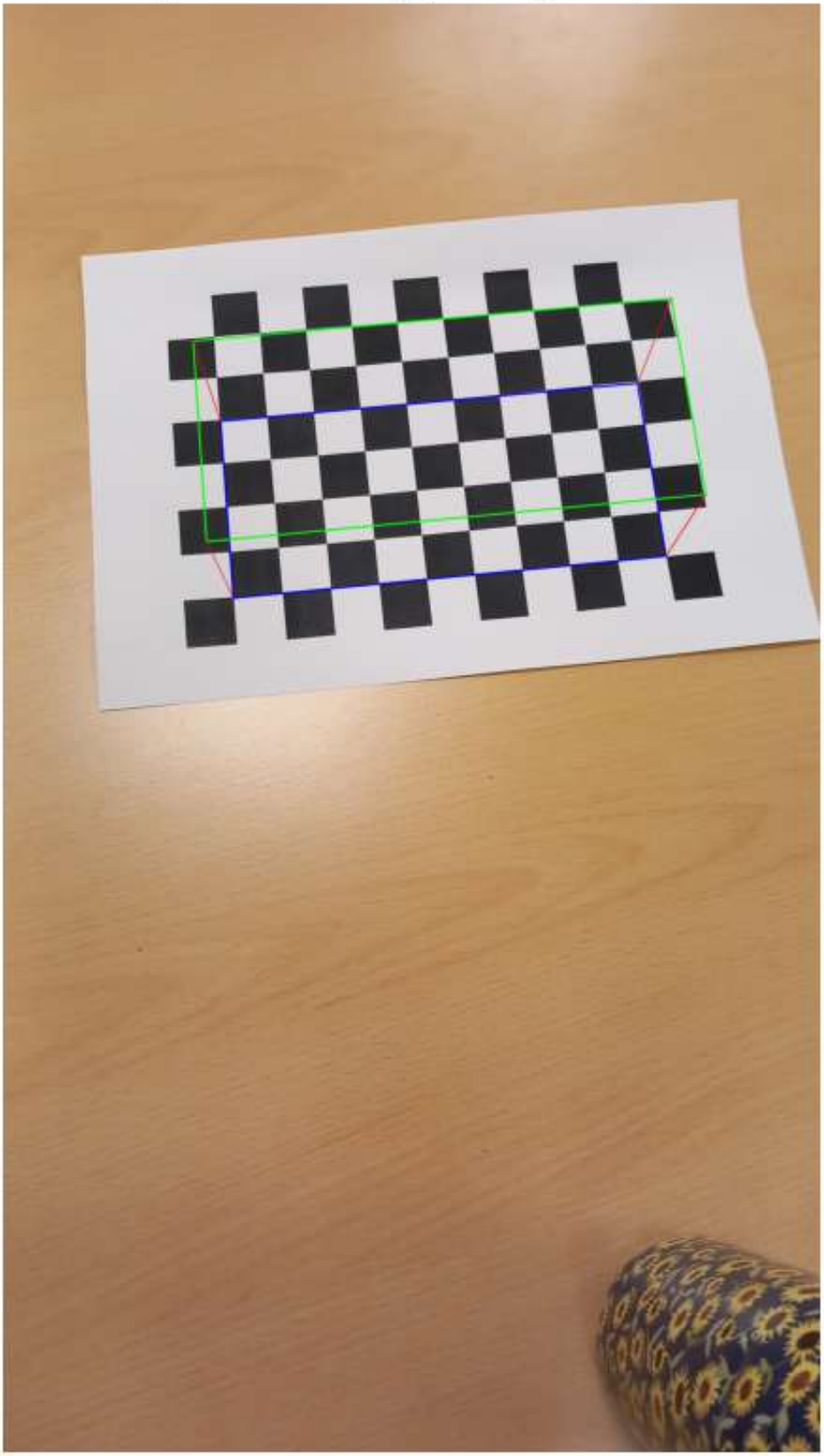


Image 12: Parallelepiped Superimposed

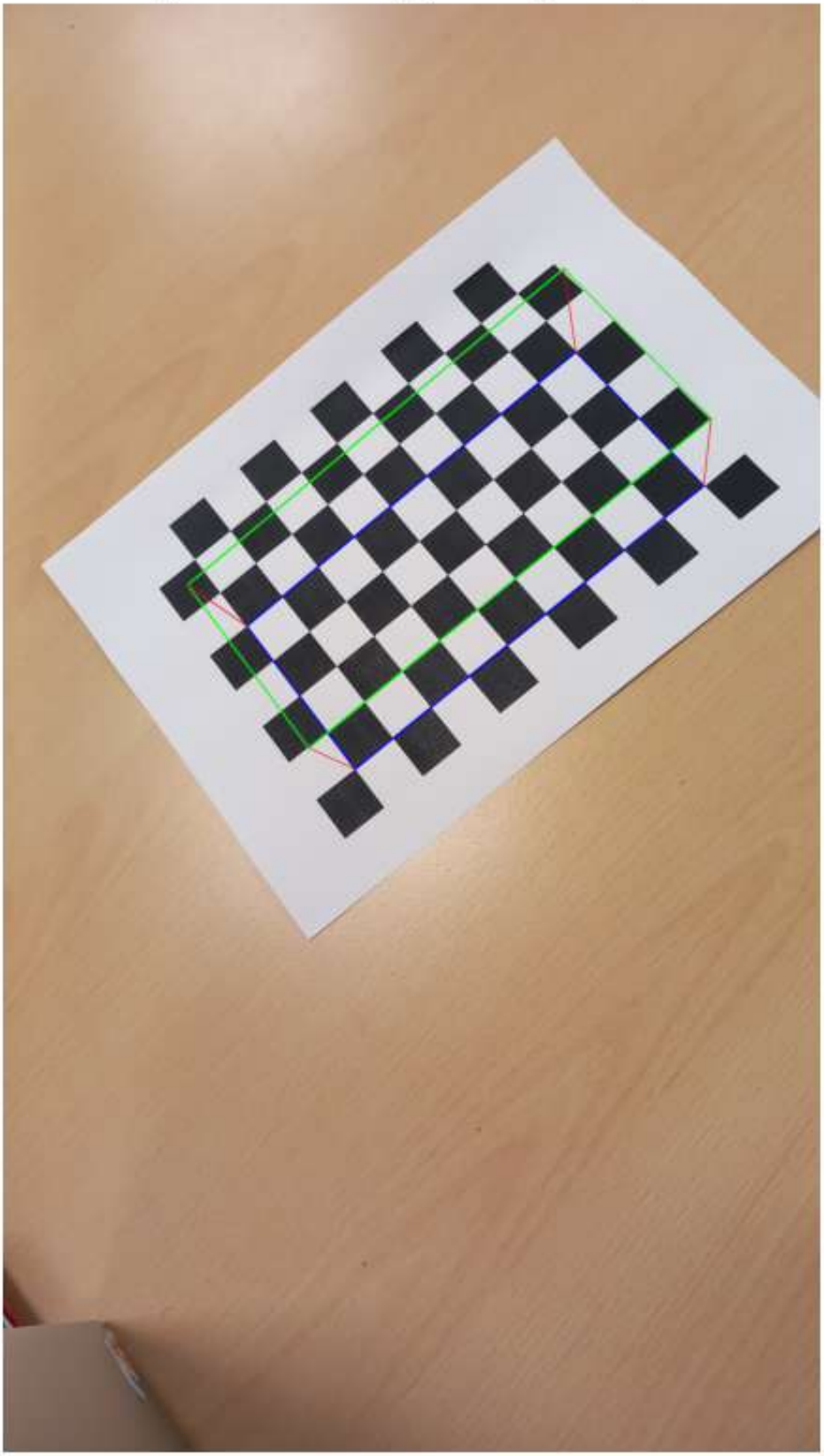


Image 13: Parallelepiped Superimposed

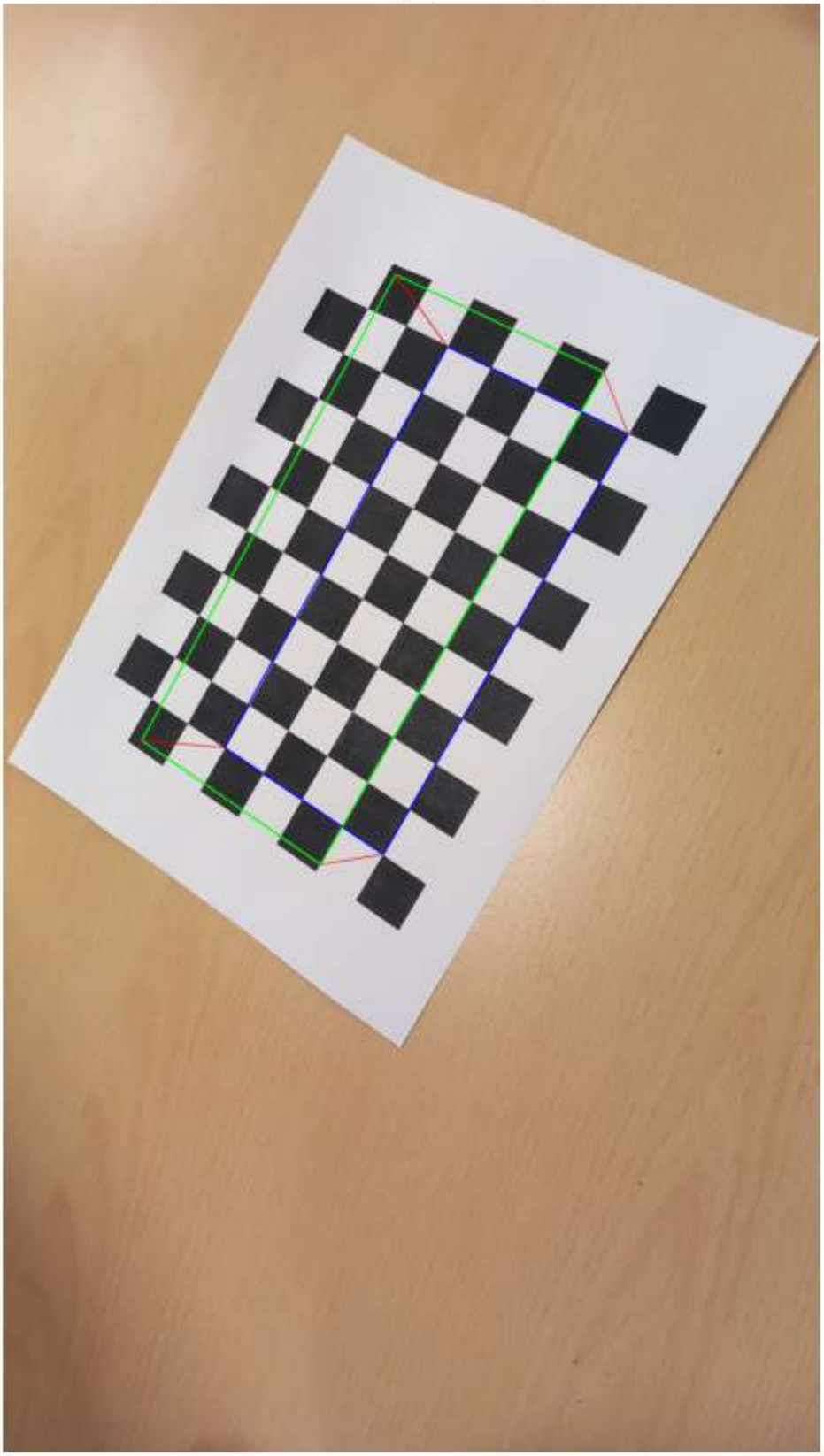


Image 14: Parallelepiped Superimposed

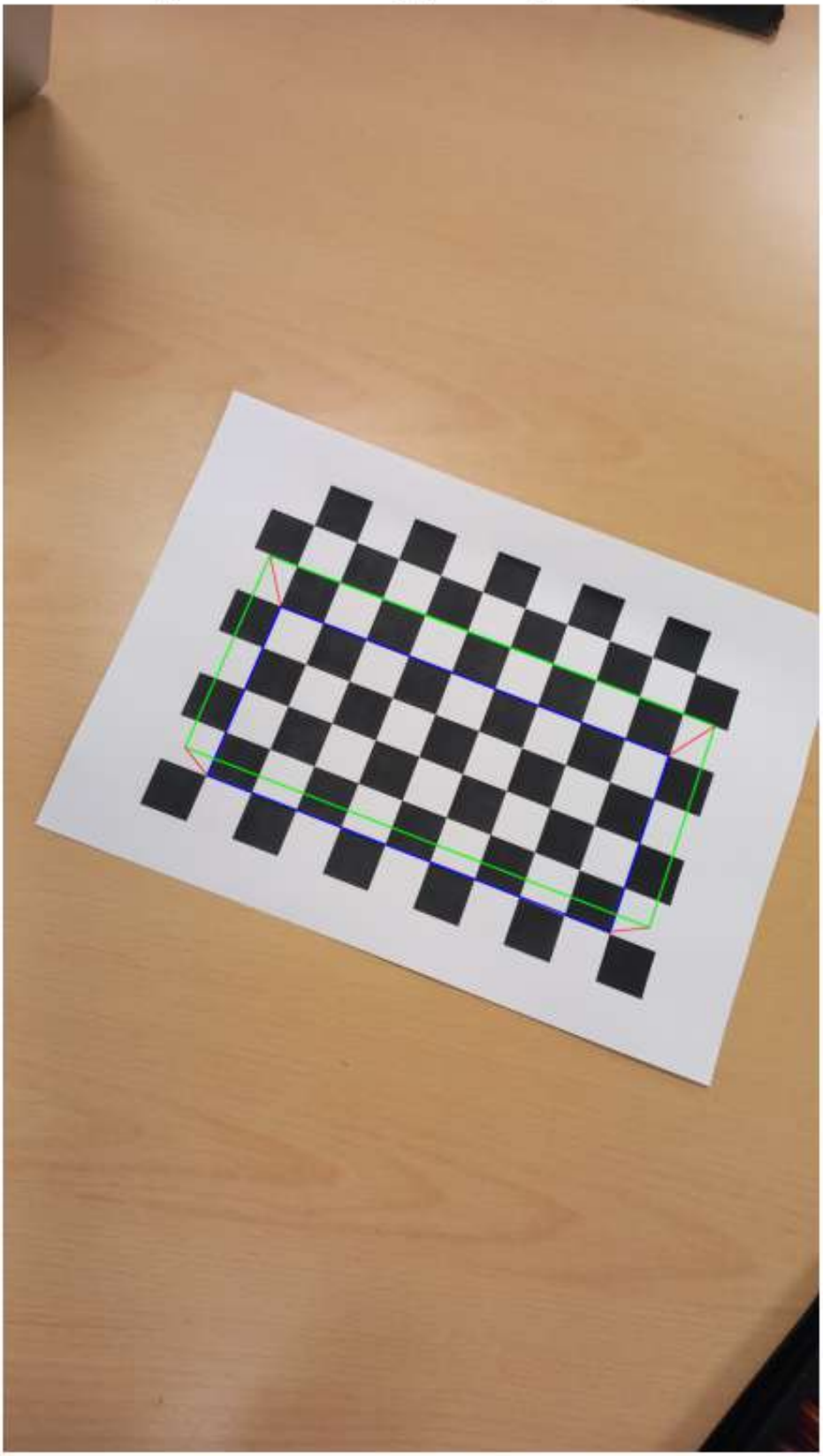


Image 15: Parallelepiped Superimposed

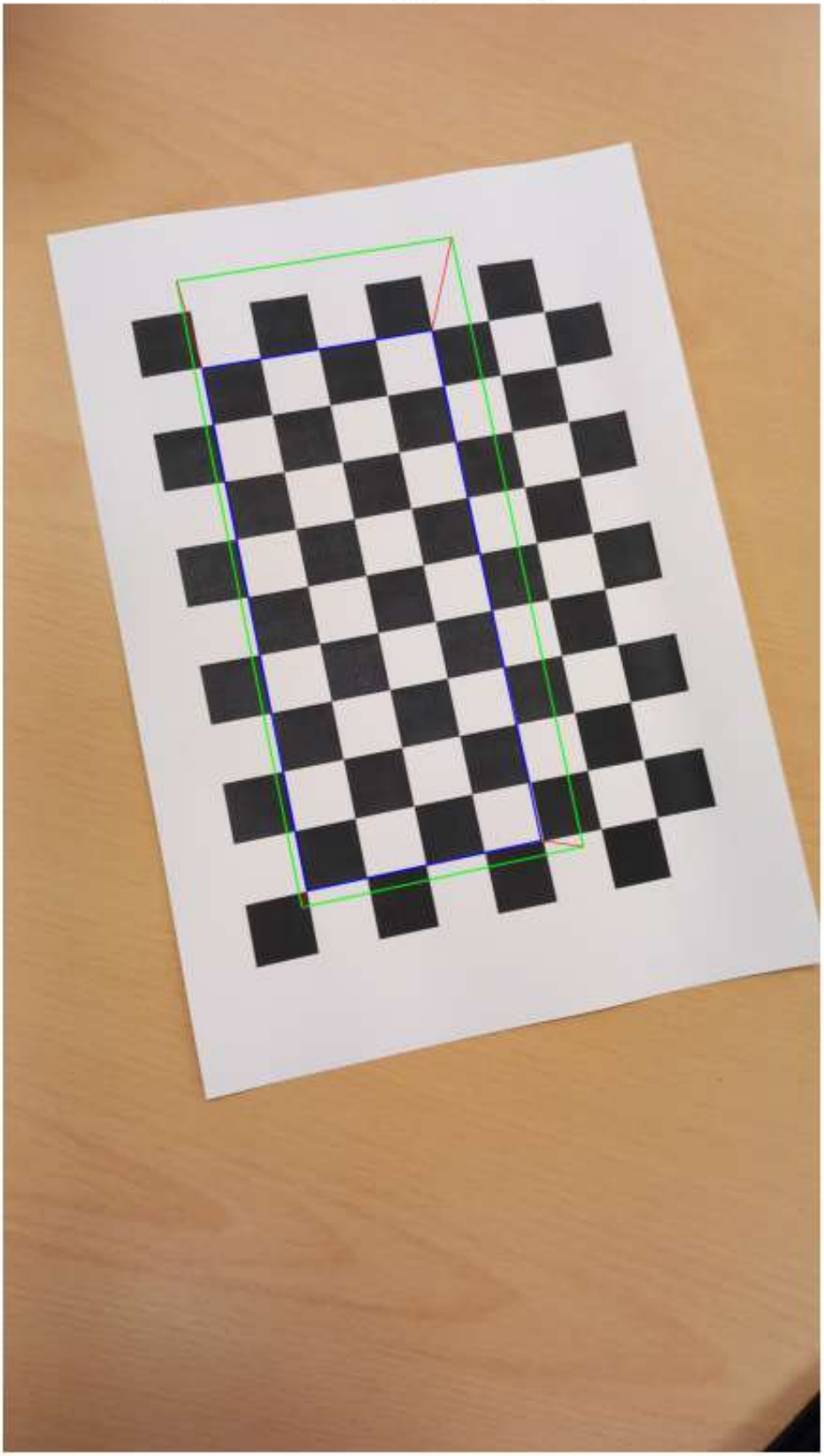


Image 16: Parallelepiped Superimposed

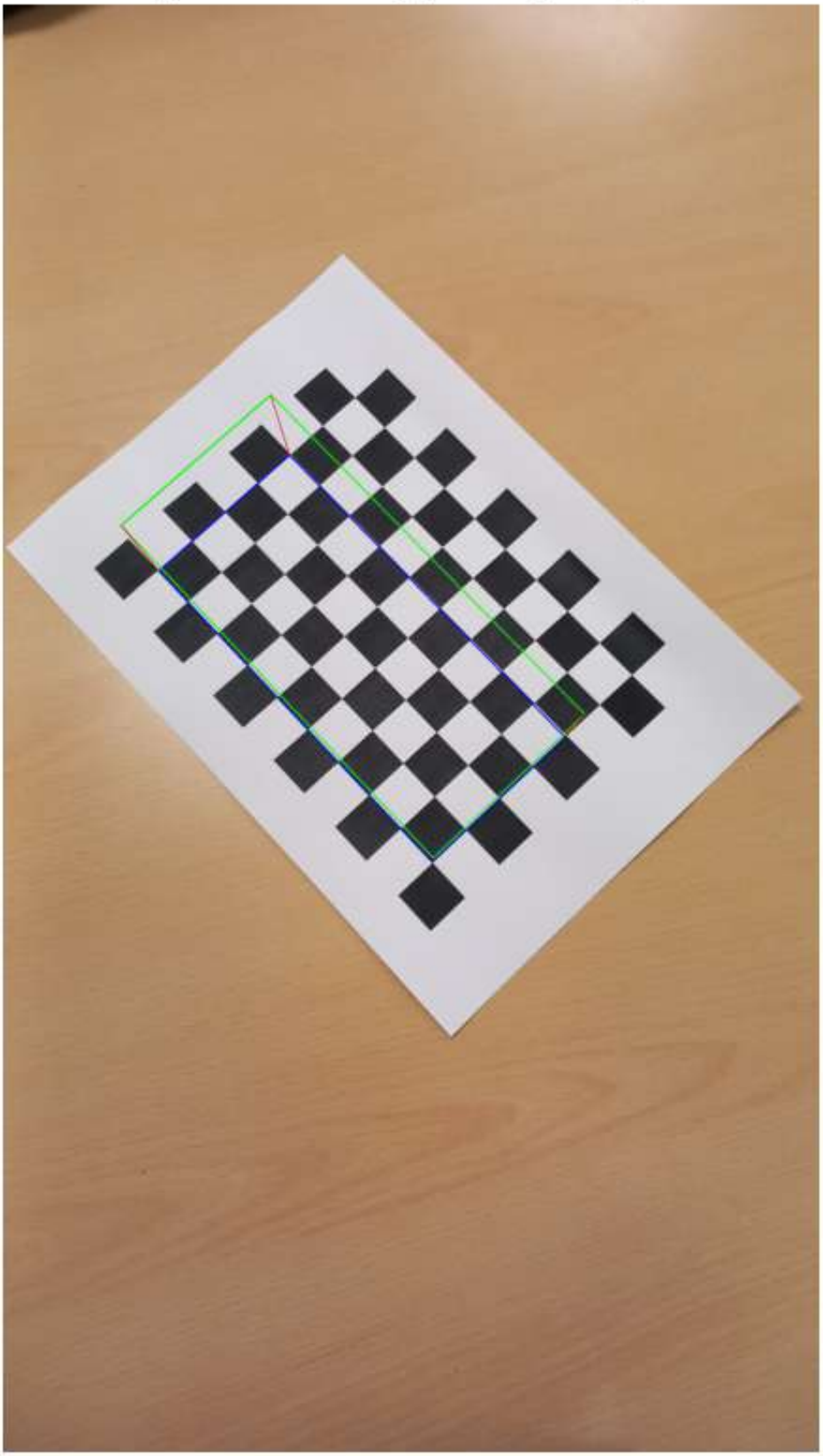


Image 17: Parallelepiped Superimposed

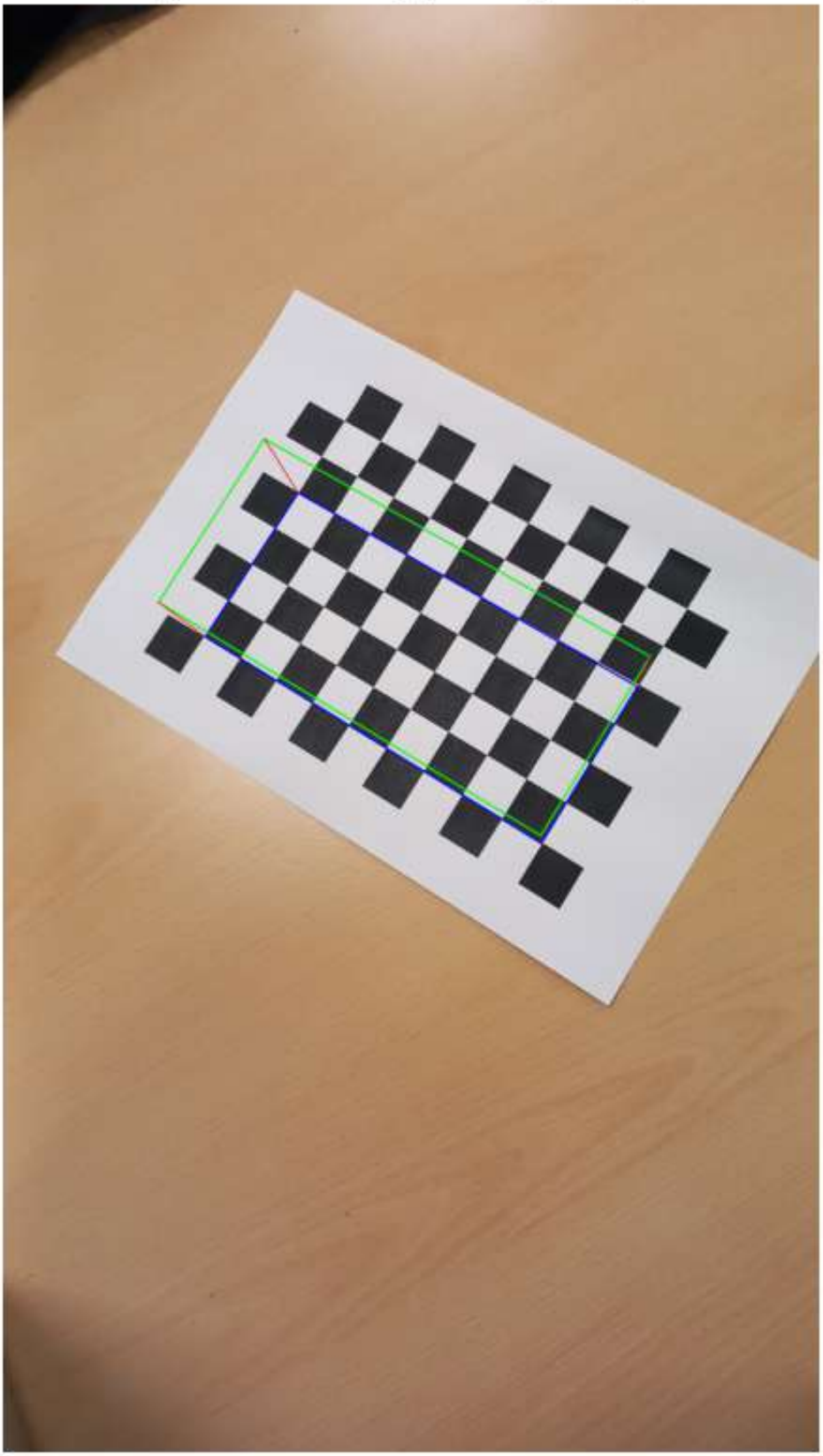


Image 18: Parallelepiped Superimposed

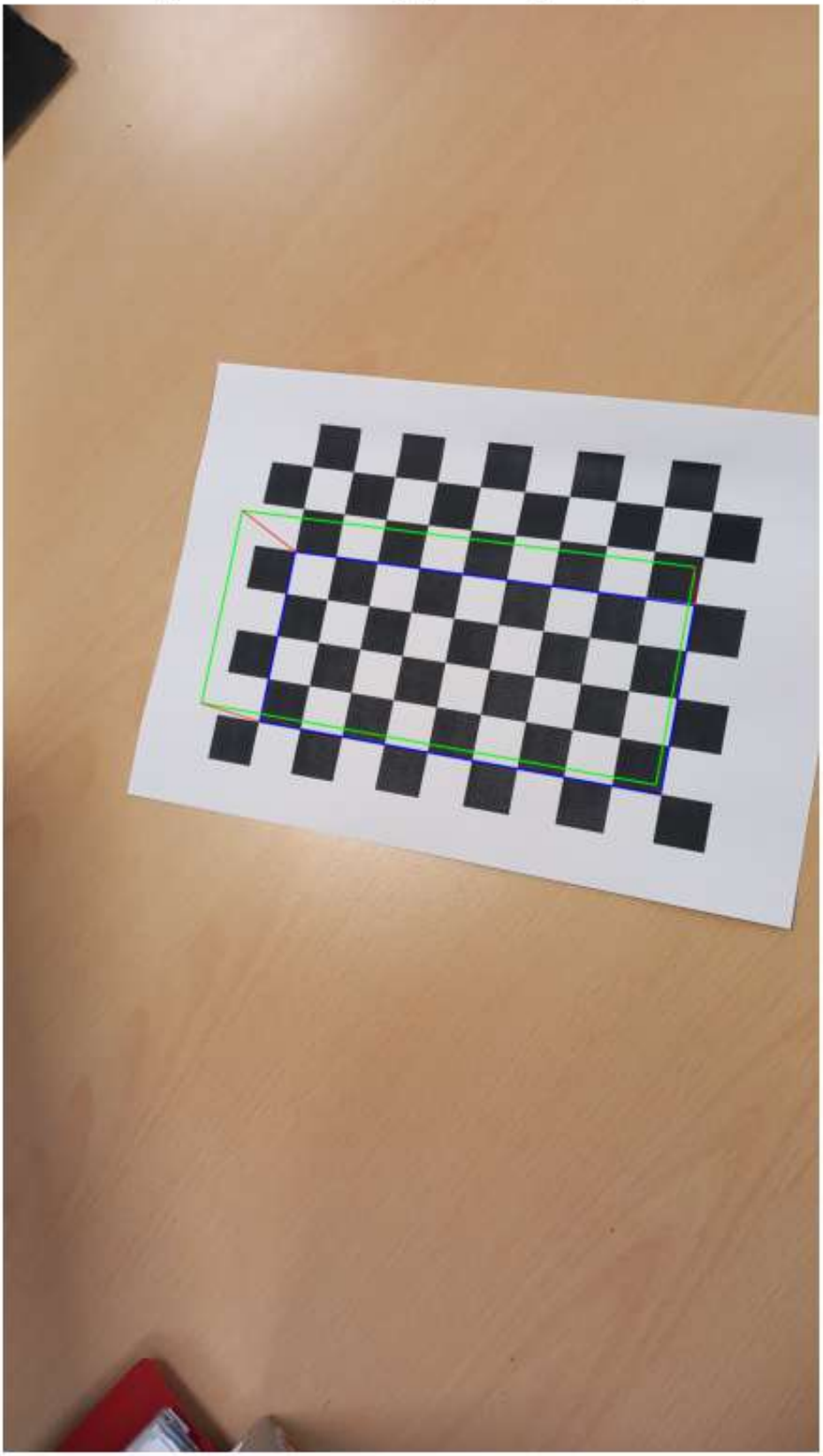
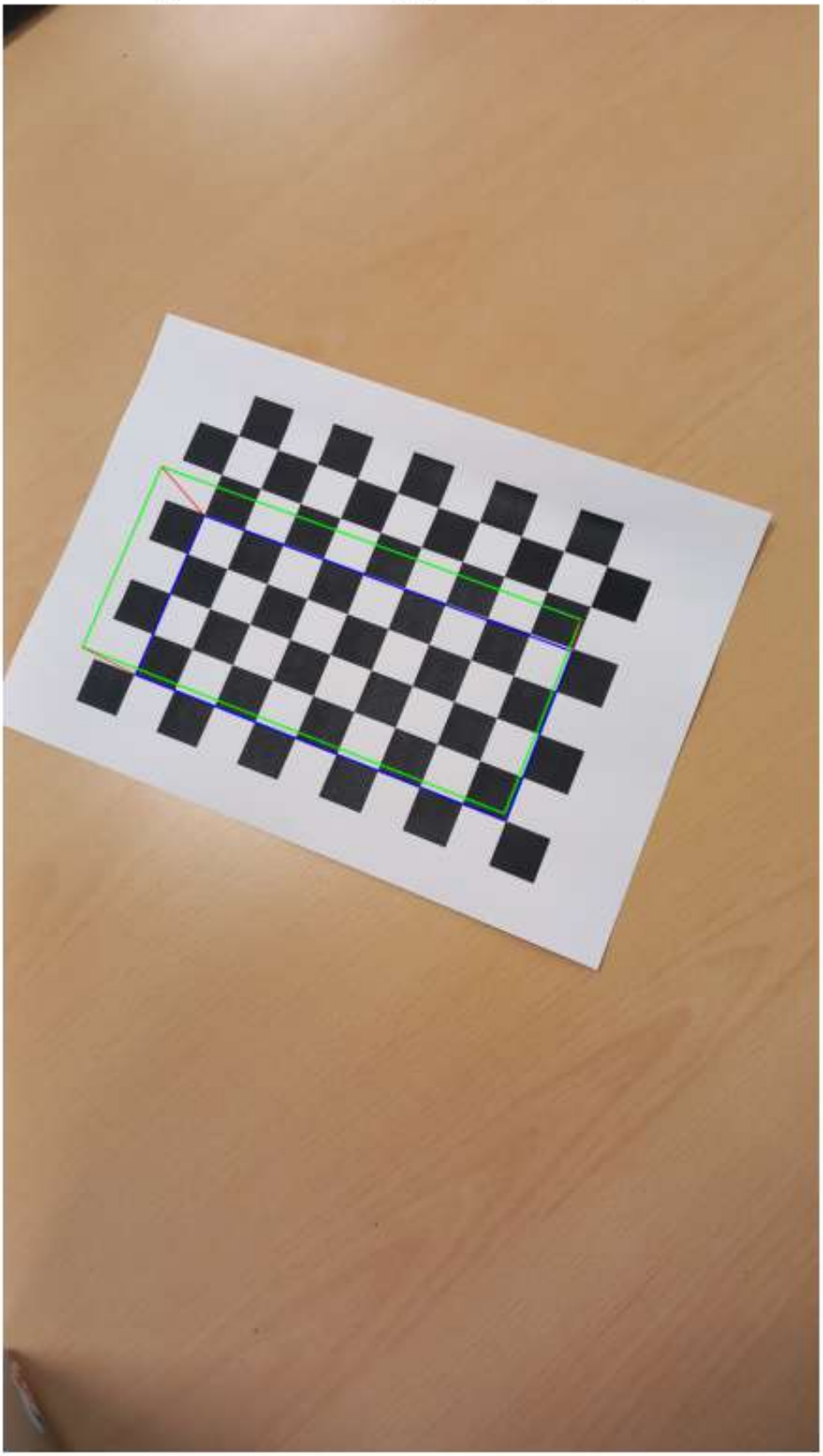


Image 19: Parallelepiped Superimposed



4.4 Total reprojection error with radial distortion

4.4.1 Assignment Images

The first estimate of k_1 and k_2 obtained was:

$$k_1 = -0.017530675101594343$$

$$k_2 = 0.19216854944230485$$

After the compensation the final estimated intrinsics were:

$$K = \begin{bmatrix} 1.76640158e+03 & -2.07066361e+01 & 6.24933464e+02 \\ 0.00000000e+00 & 1.76328598e+03 & 5.04657793e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

The final estimated k_1 and k_2 were:

$$k_1 = -0.10646623050425857$$

$$k_2 = 1.342417792390562$$

And the total reprojection error estimated was:

$$e = 277.90016502669135$$

To visually analyze the reprojection error, the measured and projected points are displayed on the original image (green circles represent the measured points while red circles represent the projected points):

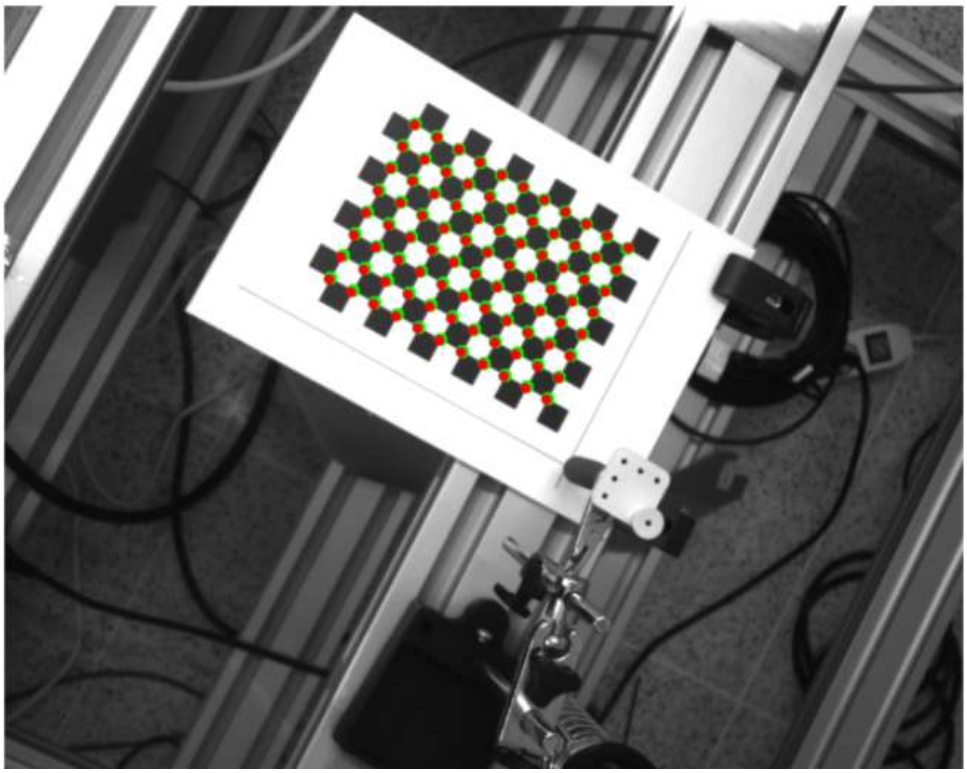


Figure 23: measured (green) and projected (red) points

4.4.2 Smartphone images

The first estimate of k_1 and k_2 obtained was:

$$k_1 = -0.0029302201940947454$$

$$k_2 = -0.0020444938152362835$$

After the compensation the final estimated intrinsics were:

$$K = \begin{bmatrix} 3.01873630e+03 & -2.16074497e+00 & 1.15241513e+03 \\ 0.00000000e+00 & 3.02003935e+03 & 2.05512604e+03 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{bmatrix}$$

The final estimated k_1 and k_2 were:

$$k_1 = -0.02442552928937491$$

$$k_2 = 0.022707530673052118$$

And the total reprojection error estimated was:

$$e = 769.5690430798793$$

To visually analyze the reprojection error, the measured and projected points are displayed on the original image (green circles represent the measured points while red circles represent the pro jected points):

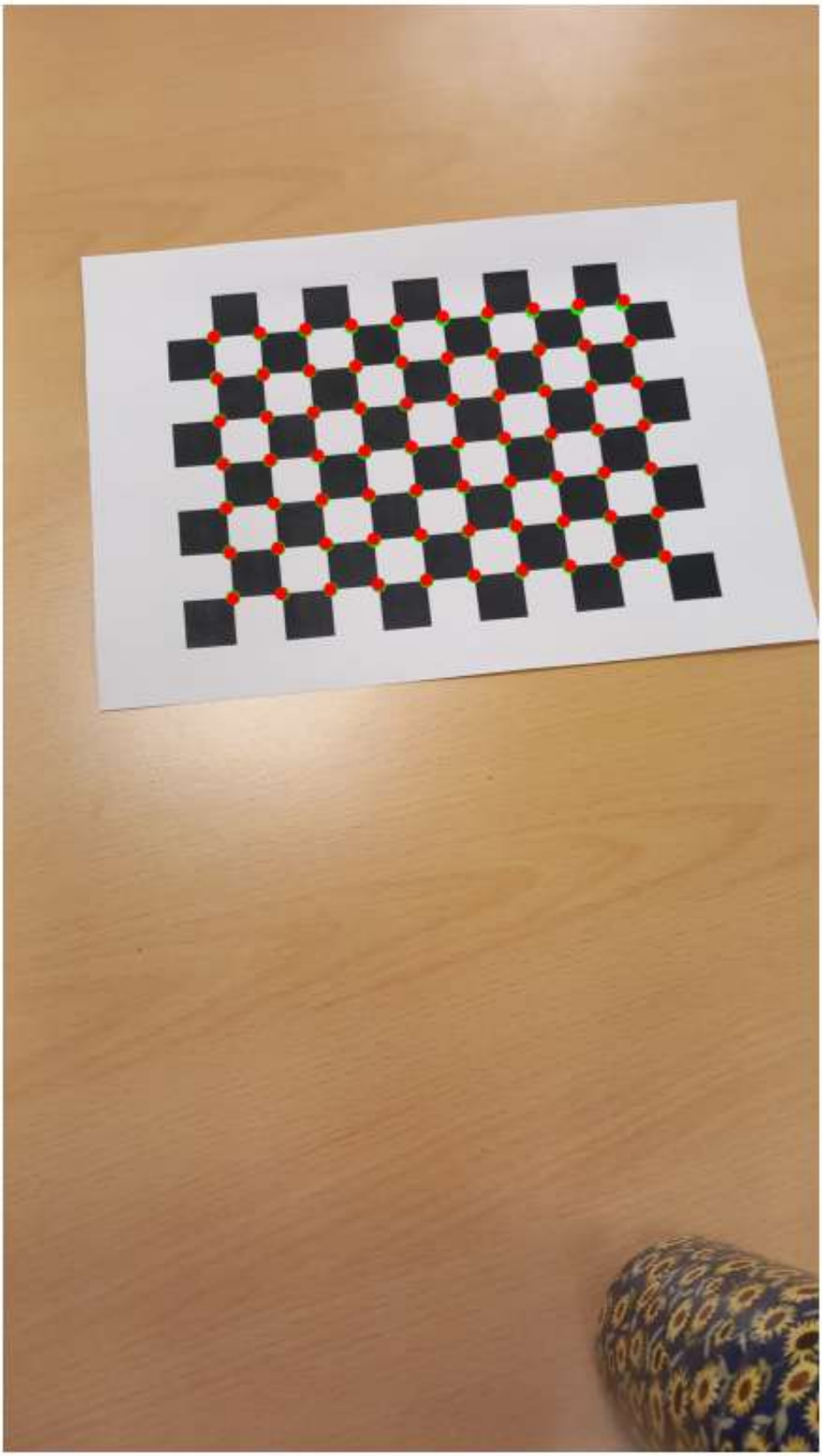


Figure 24: measured (green) and projected (red) points

5 References

We used ChatGPT version o4 mainly for writing Python code in a clearer and more elegant way (refactoring). In some cases also for writing code quicker (when plotting points in images or drawing the lines of the parallelepiped).