# Camera Calibration using Zhang's method

CVPR
Fall 2022
Final Project
Erika Lena

## Contents

**1 Introduction** **2**

**2 Problem Statement** **2**

**3 Camera Calibration** **3**
3.1 Implementation of Zhang's method . . . . . 3
3.2 3D object superimposition . . . . . 5

**4 Calibrate phone camera** **8**

**5 Radial Distortion Compensation** **9**
5.1 Adding distortion to calibration method . . . . . 9
5.2 Image undistortion . . . . . 10

**6 Results** **12**

# 1 Introduction

This project is meant to expose an implementation in Matlab of Zhang's Method for Camera Calibration.
It will be split into three parts.
In the first part, Zhang's method is followed in order to find the parameters of an unknown camera. The results achieved are then checked measuring global reprojection error. Obtained camera parameters will also be exploited to project a 3D object on the given planar patterns.

In the second part of this project, the same procedure will be repeated, this time using images taken using my smartphone camera.

Finally, third part completes Zhang's calibration procedure by adding compensation for radial distortion and see the improvement in global reprojection error.
The whole implementation for this project has been done using Matlab. For each of these sections a Matlab Live Script is provided, which summarizes the steps done and shows how to use the code functions, which can be found in */utils* folder.
Following the order of sections in this paper, the corresponding Matlab interactive documents are: *ZhangMethod.pdf*, *CalibratingPhoneCamera.pdf* and *CompensatingRadialDistortion.pdf*.

# 2 Problem Statement

Camera calibration is a very important procedure in Computer Vision; it aims to extract information from 2D images and use it to estimate the parameters of the camera model.
The main purpose is to estimate **perspective projection matrix** which provides a direct map between points in the image frame and those in the 3D camera frame.

In the following report, we will follow Zhang's procedure in order to perform camera calibration. By doing this, it will be possible to estimate the position of an object in the image reference frame by knowing its location in the world reference frame.
It will be explained how, by finding camera intrinsic and extrinsic parameters, is possible to estimate the position of an object.
The whole procedure is subject to approximation errors and most importantly to inaccuracies given by the radial distortion in real thick lenses.
This distortion can be partially compensated improving the reprojection error, as we will see.

# 3 Camera Calibration

## 3.1 Implementation of Zhang's method

As a first task, it was required to perform Calibration using Zhang's procedure [1].
The procedure was implemented as described in the paper and explained in lectures and laboratory during the course.
Twenty images of the same planar pattern at different orientations were provided to be used for calibration.
Images are loaded and Matlab *detectCheckerboardPoints* function is used to find the pixel coordinates of the checkerboard internal points. Corresponding 3D coordinates of checkerboard points are assigned, in order to have for each point a pair of representations $m' = [u\ v\ 1]^T$ and $m = [x\ y\ z\ 1]^T$, one for the image reference frame and the the other for world reference frame, which will be used for estimating homographies.
Zhang's method is based, indeed, on estimating homography, the map of a 3D plane $\Pi$ to its perspective image. That means that a model point $m$ and its image $m'$ are related by a homography $H$:

$$w \cdot m' = H \cdot m \tag{1}$$

Given at least 4 pairs of original and projected points, a matrix H, corresponding to a homography, can be computed for each of our images via Direct Linear Transform (DLT).
Once the homographies are estimated for each image, these matrices are exploited by Zhang's procedure to obtain camera intrinsics and extrinsics parameters.
Zhang procedure begins by estimating homographies for points collected on our planar patterns, so those for which third coordinate is zero.

Matrices $H$ are explicitly defined to be:

$$H = [h_1\ h_2\ h_3] = \lambda K[r_1\ r_2\ t]$$

and

$$\begin{cases} r_1 = \frac{1}{\lambda} K^{-1} h_1 \\ r_2 = \frac{1}{\lambda} K^{-1} h_2 \end{cases} \tag{2}$$

Where $K$ is the matrix of our intrinsic parameters, while $r_1$ and $r_2$ are the first two columns of rotational matrix $R$, whose values are the extrinsic parameters along with translation vector $t$.

Knowing that $R$ is an orthogonal matrix, we have that $r_1$ and $r_2$ are orthonormal, so we have the following system:

$$\begin{cases} r_1 r_2^T = 0 \\ r_1^T r_1 = r_2^T r_2 \end{cases} \tag{3}$$

and substituting $r1$ and $r2$ using equations in 2, we obtain:

$$\begin{cases} h_1^T (KK^T)^{-1} h_2 = 0 \\ h_1^T (KK^T)^{-1} h_1 = h_2^T (KK^T)^{-1} h_2 \end{cases} \tag{4}$$

So, from these we want to extract values of $K$, the matrix of intrinsic parameters, and our extrinsic $r_1, r_2, r_3, t$.
Let's define $B = K^{-T}K^{-1}$, so that the following holds:

$$h_i^T B h_j = v_{ij}^T b \tag{5}$$

where $b = [B_{11}, B_{12}, B_{22}, B_{13}, B_{23}, B33]$ and
$v_{ij} = [h_{i1}h_{j1},\ h_{i1}h_{j2} + h_{i2}h_{j2},\ h_{i2}h_{j2},\ h_{i3}h_{j1} + h_{i1}h_{j3},\ h_{i3}h_{j2} + h_{i2}h_{j3},\ h_{i3}h_{j3}]$.

Details about this are exposed in Zhang's paper [1].

By substituting equation 5 in system 4, we obtain:

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v22)^T \end{bmatrix} b = 0 \tag{6}$$

Vector $b$ is made of 6 unknowns, so in order to solve the system we need at least 3 images each of which provides two equations. Due to noise, $V$ (3x3 matrix made of $v_{ij}$ values) is usually full rank and the solution is found by solving the system through linear least square:

$$\arg\min_{||b||=1} ||Vb||^2$$

Once we have values of vector $b$, we can stack them in matrix $B$ and estimate $K$ through Cholesky factorization.

$$B = (KK^T)^{-1}$$

And for camera extrinsic, we exploit equations in system 2:

$$\begin{cases} r_1 = \dfrac{1}{\lambda} K^{-1} h_1 \\ r_2 = \dfrac{1}{\lambda} K^{-1} h_2 \\ t = \dfrac{1}{\lambda} K^{-1} h_3 \end{cases} \tag{7}$$

where $\lambda$ is the scale factor. And finally $r_3 = r_1 \times r_2$.
The whole procedure is implemented in function *estimateCamParam.m* inside *utils* folder. For each image, a data structure containing all of these parameters is saved for future use.

## 3.2 3D object superimposition

Once we have obtained camera intrinsic and extrinsic parameters, they can be exploited to obtain image coordinates of any 3D object we are interested in.
The same cylinder was superimposed to each of our planar patterns. It has center $(x_0, y_0) = (150, 150)$, height $h = 150$ and radius $r = 60$. By taking the 3D representation of 40 points on top and bottom basis and computing the corresponding image points, it was possible to represent the cylinder on each image with its different orientations. The 2D pixel coordinates are obtained by using equation 1, with **perspective projection matrix** $P$ in place of $H$.

$$w \cdot m' = P \cdot m \tag{8}$$

where $P$ is given by the combination of intrinsic and extrinsic parameters $P = K[R \mid t]$.

The only thing that we need to pay attention to is the value obtained for the $z$ coordinate. Indeed, it can happen that our resulting coordinate system is reversed with respect to $z$ axis, that is to say the direction of $z$ is wrong. This is due to the estimated homography, $H$, and how it projects our 3D plane $\Pi$ to its perspective image. In particular it depends on the direction of projection, which can be clockwise or counterclockwise. In particular, we can check the rotation angles $\alpha$, $\beta$ and $\gamma$ of obtained matrix $H$ and see that the determinant one is $\beta$, which sometimes is positive and other times is negative (measured in radians).

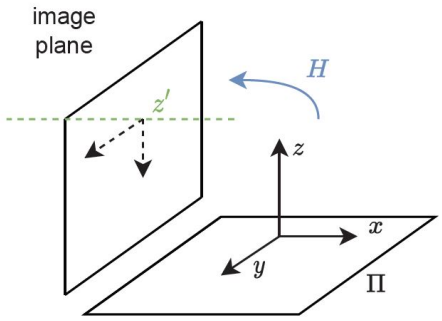


Figure 1: A visualization of the meaning of $H$ matrix, that is projecting 3D plane $\Pi$ into corresponding image plane.

This depends probably on how $SVD$ is computed, but the result is that is some cases the plane $\Pi$ is projected such that the $z$ axis has the right direction, while in the others $x$ and $y$ reciprocal position is flipped and when we apply cross product we get $z$ pointing in the wrong direction.
Then, by looking at $\beta$, we decide if the $z$ axis must be flipped or not in order to get the right values of $z$ when projecting points.

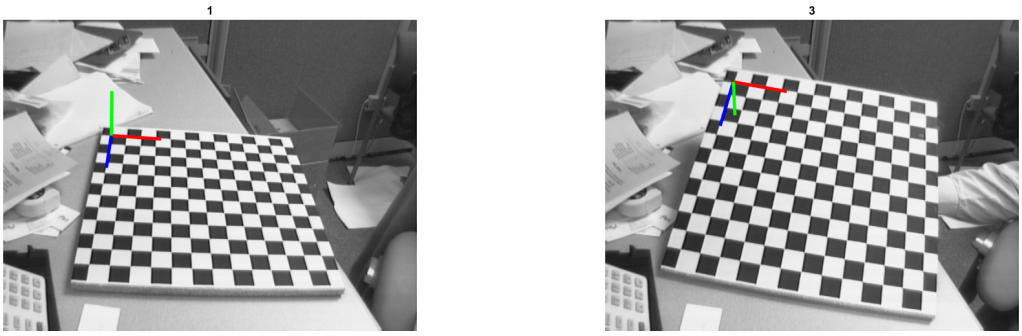


Figure 2: Visualization of coordinate systems for images 1 and 3. It can be seen how z axis points in the wrong direction on image 3.

| **Image** | $\beta$ **values (radians)** |
|---|---|
| 1 | -0.753 |
| 2 | -0.015 |
| 3 | 1.119 |
| 4 | 1.147 |
| 5 | -1.38 |
| 6 | -0.605 |
| 7 | 0.955 |
| 8 | 0.605 |
| 9 | 1.434 |
| 10 | 1.506 |
| 11 | 1.398 |
| 12 | 1.251 |
| 13 | 1.113 |
| 14 | 1.158 |
| 15 | 0.356 |
| 16 | 1.296 |
| 17 | -1.131 |
| 18 | -1.131 |
| 19 | 0.839 |
| 20 | 0.624 |

Once this correction is being applied, we can proceed with the visualization of the projection of a 3D object.

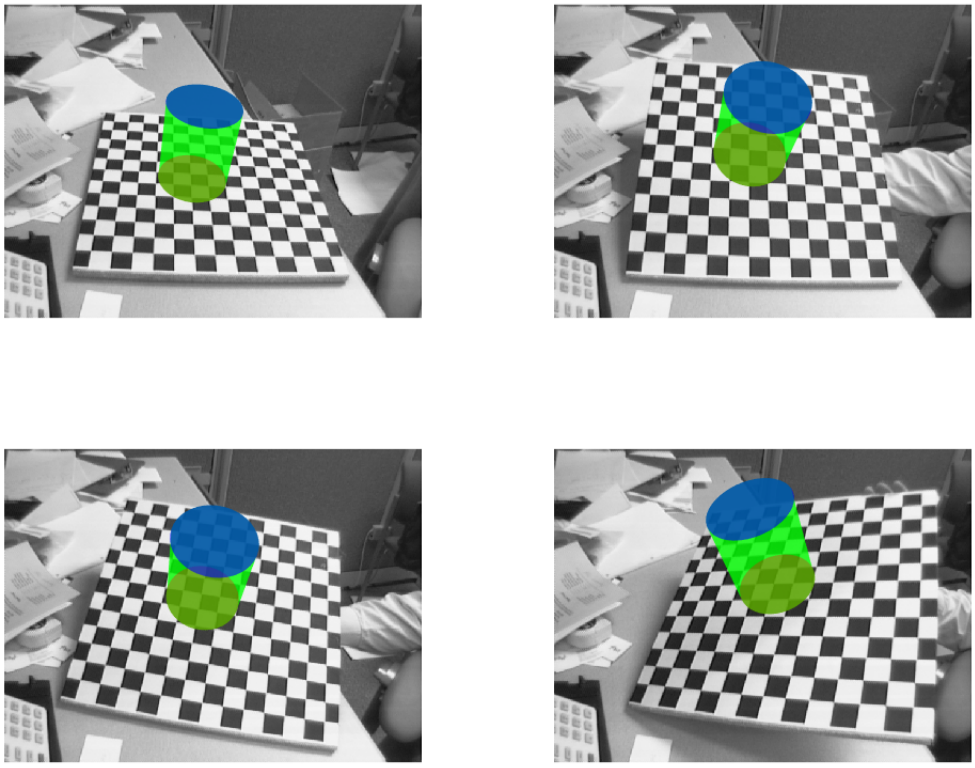

Figure 3: Superimposition of a cylinder to 4 of the different planar pattern used for calibration.

In *cylinder_projection* folder, a representation of cylinder superimposition to calibration plane is saved for each image.

# 4 Calibrate phone camera

The same steps explained in Section 3 are repeated, but this time the images were taken using my 12MP smartphone camera.
Before applying Zhang's procedure, the images were converted to gray scale.
Then, the function *estimateCamParam.m*, which implements Zhang procedure, is used to obtain an estimation of camera parameters.
Through the use of these parameters the superimposition of 3D object is performed like in previous section. Cylinder parameters are slightly different, also because the checkerboard used was smaller (square size = $17mm$).

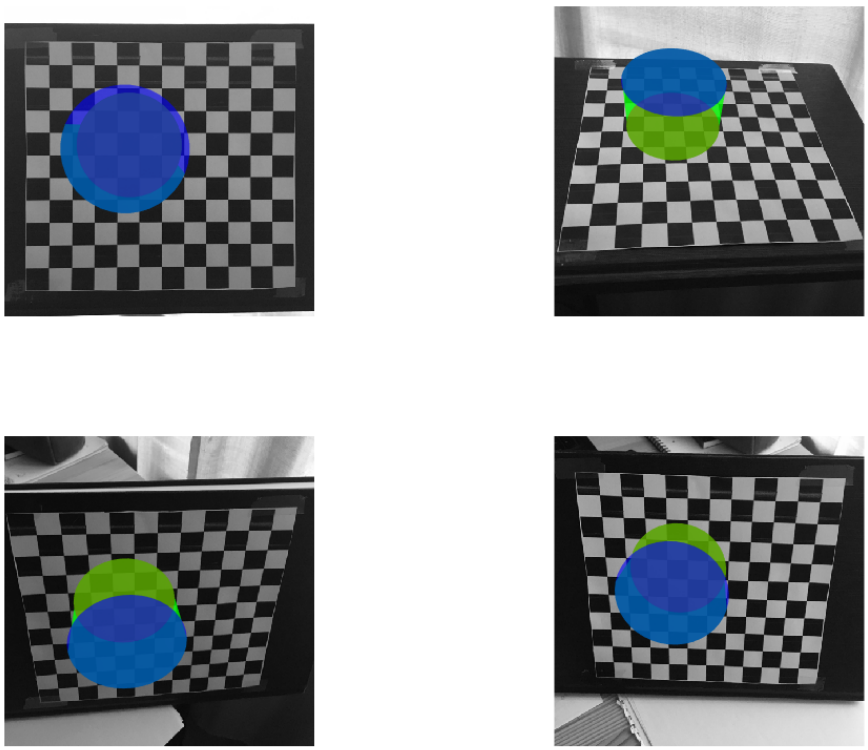

Figure 4: Superimposition of a cylinder to 4 of the different orientations of the planar pattern used for calibration. Here the images were obtained by using my personal smartphone camera.

# 5 Radial Distortion Compensation

## 5.1 Adding distortion to calibration method

This last part is about how to add radial distortion compensation to Zhang procedure discussed in previous sections.
The images used are the same used for Section 3. The function already implemented for Zhang procedure is called in order to obtain an estimation of **perspective projection matrix** $P$ for each image.
As we saw, by applying the computed matrix $P$ to obtain the projection of our points on the image plane, we obtain a significant reprojection error. This is also due to lens distortion of the camera. We will now take into account the radial lens distortion and try to add it to our model to see if we can improve our results.
We are going to consider only radial distortion coefficients because they are the most relevant ones. [1]
Let's say that $(u, v)$ are our ideal, distortion-free, pixel image coordinates, while $(\hat{u}, \hat{v})$ are the corresponding observable image coordinates.
The simplest model, which takes into account radial distortion, adds two coefficients $k_1$ and $k_2$:

$$\begin{cases} \hat{u} = (u - u_0)(1 + k_1 r_d^2 + k_2 r_d^4) + u_0 \\ \hat{v} = (v - v_0)(1 + k_1 r_d^2 + k_2 r_d^4) + v_0 \end{cases} \tag{9}$$

where

$$r_d^2 = \left(\frac{u - u_0}{\alpha_u}\right)^2 + \left(\frac{v - v_0}{\alpha_v}\right)^2 \tag{10}$$

Once we have computed an estimation of $P$, as we previously did, we can exploit system 9 by rewriting it as:

$$\begin{cases} \hat{u} - u = (u - u_0)k_1 r_d^2 + (u - u_0)k_2 r_d^4 \\ \hat{v} - v = (v - v_0)k_1 r_d^2 + (v - v_0)k_2 r_d^4 \end{cases} \tag{11}$$

where the unknowns are $k_1$ and $k_2$, while an estimation of $u_0$, $v_0$, $\alpha_u$ and $\alpha_v$ is given by $P$ and the estimation of $u$, $v$ is given by the projection of the point $m$ we are interested in through $P$; $\hat{u}$ and $\hat{v}$ are the corresponding distorted coordinates.

We obtain a linear system made of two equations for each point we consider. Two equations are sufficient to estimate $k_1$ and $k_2$, but it is advisable to use more correspondences and solve the corresponding overdetermined system. In the implementation provided, four points of each image were used for a total of $2 \cdot 4 \cdot nImages = 160$ equations.
Once we have an estimation of $k_1$ and $k_2$, we can use them to compensate radial distortion by inverting the transformation. To do this, we solve the following system:

[1] It has been verified by using already implemented Matlab functions that tangential distortion is zero.

$$\begin{cases} \hat{x} = x(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \\ \hat{y} = y(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \end{cases} \tag{12}$$

where our unknowns are the *undistorted normalized coordinates* $x, y$:

$$x = \frac{u - u_0}{\alpha_u}, \quad y = \frac{v - v_0}{\alpha_v} \tag{13}$$

While $\hat{x}$, $\hat{y}$ are the actual distorted ones:

$$\hat{x} = \frac{\hat{u} - u_0}{\alpha_u}, \quad \hat{y} = \frac{\hat{v} - v_0}{\alpha_v} \tag{14}$$

This system of non linear equations in $x, y$ is solved by using Newton's method; the implementation can found in *utils* folder.
Once new values for $x, y$ have been found, the coordinates $u, v$ are computed using equations 13 and they are used to estimate matrix $P$. Using this new $P$ we can repeat the whole procedure from the begin until convergence.
The code which implements what just explained is contained in *radialDistortionCompensation.m*.

## 5.2 Image undistortion

We can use the resulting $k_1$ and $k_2$ to obtain an undistorted representation of our image.
To achieve this result, for each $(u, v)$ pair of final *undistorted* image, the corresponding $(\hat{u}, \hat{v})$ coordinates have been found by using equations 11. So that intensity of undistorted image in $(u, v)$ is set equal to intensity of original image in $(\hat{u}, \hat{v})$.
Since undistortion algorithm can produce non-integer values for $\hat{u}$ and $\hat{v}$, interpolation is needed. The implementation provided makes use of bilinear interpolation, well described in Matlab Documentation resources about image undistortion [2].
Let's say $(\hat{u}, \hat{v})$ is the coordinate of the pixel obtained through undistortion. The intensity of the output pixel is provided by a linear combination of the values of the four neighboring pixels of $(\hat{u}, \hat{v})$, which are weighted with respect to the the distance from the point $(\hat{u}, \hat{v})$ itself.

$$I(u, v) = p_1(1 - \Delta_u)(1 - \Delta_v) + p_2\Delta_u(1 - \Delta_v) + p_3 * (1 - \Delta_u) * \Delta_v + p_4\Delta_u\Delta_v \tag{15}$$

where $p_1, p_2, p_3, p_4$ are the four neighboring pixels.

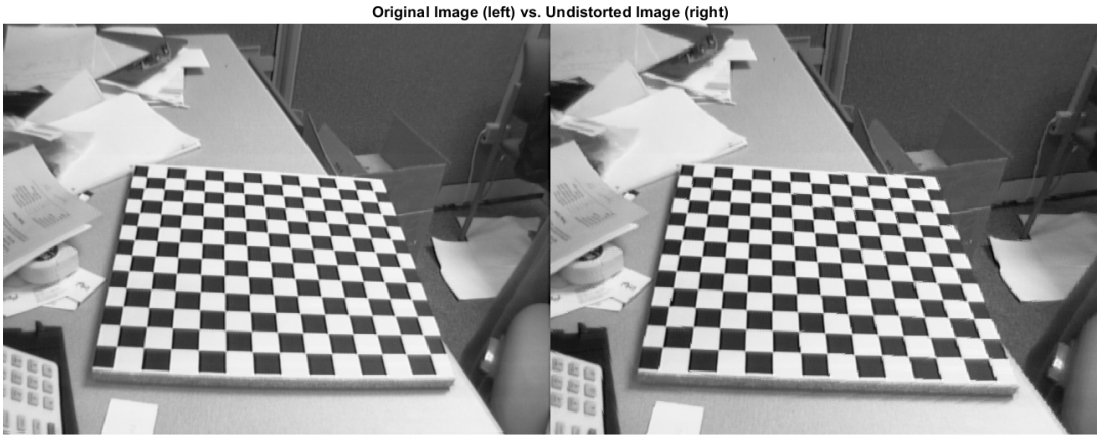


Figure 5: Comparison between one of the original images of the planar pattern and the corresponding undistorted image obtained through the use of resulting radial distortion coefficients $k_1$ and $k_2$.

In the figure above the result of this undistortion is shown for the first image. In *radial_compensation* folder a representation of this comparison is given for each one of the twenty original images.
The newly obtained matrix $P$, along with radial distortion coefficients, is being used to recompute projections of our points and estimate reprojection error. The results obtained are reported in next section.

# 6 Results

In this section we will look at results obtained for each of the previous ones and try to draw conclusions from them.
In the first section, Zhang method for camera calibration has been followed in order to estimate camera parameters and get an approximation of perspective reprojection matrix $P$, This estimation has then been used to get reprojected points on our checkerboard as:

$$m' = P \cdot m \tag{16}$$

where $m$ is the vector representing each point in space in homogeneous coordinates, $m = [x\ y\ z\ 1]^T$.
Once we have done this we can compute the total reprojection error as the sum of square differences between projected points and the distorted ones captured on the image by Matlab *detectCheckerboardPoints* function.

Figure 6: Measured (in red) and reprojected (in blue) points comparison on first image of planar pattern.

Reprojection error has been measured as geometric residual given by:

$$\epsilon(P) = \sum_{i=1}^{n} \left( \frac{p_1^T m_i}{p_3^T m_i} - u_i \right)^2 + \left( \frac{p_2^T m_i}{p_3^T m_i} - v_i \right)^2 \tag{17}$$

The measured error was quite high, for the image shown it was $\approx 1.32 \cdot 10^4 \text{pixels}^2$.

After recomputing perspetive projection matrix adding radial distortion compensation, the points were projected and the error was measured again, obtaining a total reprojection error of 7.21pixels$^2$ this time.

Figure 7: Same image as before, but this time reprojected points are obtained adding compensation for radial distortion.

In the following table, reprojection errors before and after radial distortion compensation are reported for each image used.

| | error before compensation | error after compensation |
|---|---|---|
| 1 | 1.3242e+04 | 7.2105 |
| 2 | 3.7657e+03 | 13.6627 |
| 3 | 1.7748e+03 | 6.9512 |
| 4 | 1.2950e+03 | 9.1192 |
| 5 | 1.5729e+03 | 8.3466 |
| 6 | 2.5793e+03 | 8.2850 |
| 7 | 735.4847 | 8.9170 |
| 8 | 8.9451e+03 | 18.1028 |
| 9 | 7.2403e+03 | 20.6533 |
| 10 | 657.3862 | 23.3672 |
| 11 | 3.9306e+03 | 38.3506 |
| 12 | 8.5224e+03 | 20.2210 |
| 13 | 2.2743e+03 | 41.5716 |
| 14 | 6.6372e+03 | 17.3535 |
| 15 | 8.7708e+03 | 51.1317 |
| 16 | 4.0709e+03 | 72.3710 |
| 17 | 3.1699e+03 | 14.7640 |
| 18 | 2.2371e+03 | 5.0123 |
| 19 | 1.7515e+03 | 4.0301 |
| 20 | 1.9420e+04 | 45.7813 |

Figure 8: Total reprojection error for each of the original images measured before and after radial distortion compensation.

Reprojection error has also been computed for the images taken with my smartphone camera. The errors in this case were much smaller with respect to the image dimensions and radial distortion was negligible.

# References

[1] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[2] Inc. The MathWorks. Image undistortion. https://it.mathworks.com/help/visionhdl/ug/image-undistort.html.