Computer Vision

Assignment 1: Camera calibration

Student: Ivan Alberico

The calibration was performed on the image provided in the file. The chosen coordinate system for the 3D correspondences is shown in *Figure 1*, and it is the same as the one presented in the assignment description. The results of the calibration provided in this report are obtained by considering only the 6 edge corners of the checkerboard. The points were selected with the function *getpoints.m*, which provides the coordinates of the clicked points. The corresponding 3D coordinates were manually typed considering that each square of our checkerboard has a side length of 27mm, as expressed in *Figure 1*.

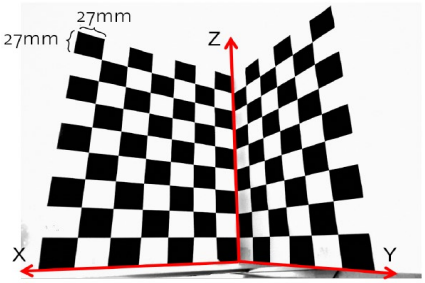


Figure 1 — Reference coordinate system of the checkerboard for the 3D points' coordinates.

2.1. Data normalization

The first thing that was done just after getting the coordinates of the calibration points was normalizing the data by computing the transformation matrices T and U. First of all, all the 2D-3D correspondences are transformed in their respective homogeneous configuration, and this was performed with the Matlab function *homogenization.m*. After that, I computed the centroids of the data by summing up the coordinates and dividing the sum by the total number of points (average).

$$x_i = \begin{bmatrix} x_{1i} \ , \ x_{2i} \ , \ 1 \end{bmatrix}^T \qquad X_i = \begin{bmatrix} X_{1i} \ , \ X_{2i} \ , \ X_{3i} \ , \ 1 \end{bmatrix}^T$$

$$\mu_{2D} = \begin{pmatrix} \sum_{i=1}^{n} x_{1i} \Big/ n \\ \sum_{i=1}^{n} x_{2i} \Big/ n \\ 1 \end{pmatrix} = \begin{pmatrix} \mu_{2D,1} \\ \mu_{2D,2} \\ 1 \end{pmatrix} \qquad \mu_{3D} = \begin{pmatrix} \sum_{i=1}^{n} X_{1i} \Big/ n \\ \sum_{i=1}^{n} X_{2i} \Big/ n \\ \sum_{i=1}^{n} X_{3i} \Big/ n \\ 1 \end{pmatrix} == \begin{pmatrix} \mu_{3D,1} \\ \mu_{3D,2} \\ \mu_{3D,3} \\ 1 \end{pmatrix}$$

By subtracting the centroids from our points, the dataset is centered around the origin. This is achieved by means of the following homogeneous transformation matrices:

$$T_0 = \begin{pmatrix} 1 & 0 & -\mu_{2D,1} \\ 0 & 1 & -\mu_{2D,2} \\ 0 & 0 & 1 \end{pmatrix} \qquad U_0 = \begin{pmatrix} 1 & 0 & 0 & -\mu_{3D,1} \\ 0 & 1 & 0 & -\mu_{3D,2} \\ 0 & 0 & 1 & -\mu_{3D,3} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Next step was scaling the vectors such that they had unit norm (unit mean distance from the origin). I computed the scale factors in the following way:

$$\lambda_{2D} = \frac{1}{\sum_{i=1}^{n} \left\| T_0 \cdot x_i \right\|} \qquad T = \begin{pmatrix} \lambda_{2D} & 0 & -\mu_{2D,1}\ \lambda_{2D} \\ 0 & \lambda_{2D} & -\mu_{2D,2}\ \lambda_{2D} \\ 0 & 0 & 1 \end{pmatrix} \qquad \hat{x}_i = Tx_i$$

$$\lambda_{3D} = \frac{1}{\sum_{i=1}^{n} \left\| U_0 \cdot X_i \right\|} \qquad U = \begin{pmatrix} \lambda_{3D} & 0 & 0 & -\mu_{3D,1}\ \lambda_{3D} \\ 0 & \lambda_{3D} & 0 & -\mu_{3D,2}\ \lambda_{3D} \\ 0 & 0 & \lambda_{3D} & -\mu_{3D,3}\ \lambda_{3D} \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \hat{X}_i = UX_i$$

2. Direct Linear Transform (DLT)

In order to perform the Direct Linear Transform algorithm we have to compute matrix A for each calibration point, such that:

$$\left[x_i\right]_x \hat{P} X_i = 0 \ \rightarrow\ A_i \hat{P} = 0 \ \rightarrow\ \begin{bmatrix} \hat{w}_i \hat{X}_i^T & 0_{1\times4} & -\hat{x}_i \hat{X}_i^T \\ 0_{1\times4} & -\hat{w}_i \hat{X}_i^T & \hat{y}_i \hat{X}_i^T \end{bmatrix} \begin{pmatrix} \hat{P}^1 \\ \hat{P}^2 \\ \hat{P}^3 \end{pmatrix} = 0$$

Once A is obtained, the normalized projection matrix $\hat{P}$ can be computed as the right null-space of A. Singular-Value-Decomposition is therefore performed on matrix A:

$$A = USV^T$$

The values of matrix $\hat{P}$ correspond to the last column of V, which is the eigenvector with the smallest eigenvalues, and the final result is obtained by reshaping the vector in a 3x4 matrix. The projection matrix P is then obtained by de-normalizing $\hat{P}$, by means of T and U, in the following way:

$$P = T^{-1} \hat{P} U$$

In order to obtain the intrinsic camera matrix K, the rotation matrix R and the camera center C, I developed the *decompose.m* function in which I performed the QR decomposition on matrix M^{-1}:

$$P = K\left[R \mid t\right] = K\left[R \mid -RC\right] = \left[KR \mid -KRC\right]$$

$$M = KR \qquad M^{-1} = R^{-1}K^{-1}$$

In this way I obtain the rotation matrix R and the intrinsic matrix K, which is an upper triangular matrix. Then, K is normalized with respect to $K(3,3)$, hence we divide all the elements of the matrix by this value, and this is done along the degree of freedom of the homogeneous coordinates.

What needs to be done now, is to make sure that K has non-negative elements on the diagonal and R has a determinant equal to 1. In case matrix K presents a negative value on the diagonal, I need to change the sign of the elements on the corresponding column of K and adapt the corresponding row in R accordingly. At this point, if the determinant of R is negative I can simply flip the sign of the matrix, by simply setting $R = -R$. This check was performed in *decompose.m* in the following way:

```
for i = 1 : 3
    if K(i,i) < 0
        K(:,i) = -K(:,i);
        R(i,:) = -R(i,:);
    end
end

if det(R) == -1
    R = -R;
end
```

The camera matrix, instead, is obtained considering that $PC = 0$, so it comes from the Singular-Value-Decomposition of P, being C the right null vector of P. Once vector C is obtained, we normalize it with respect to its last component, and we calculate $t = -RC$. The average reprojection error is computed as the average norm of the difference between the coordinates of the selected points and their projected values obtained by means of the projection matrix P. The obtained values are the following:

$$P = \begin{pmatrix} 2579.2 & 120.2 & 418.8 & 518.7 \\ 878.2 & 1421.5 & 1797.9 & 675.8 \\ 0.8 & 1.3 & 0.6 & 0.6 \end{pmatrix} \qquad K = \begin{pmatrix} 1338.1 & 9.6 & 796.8 \\ 0 & 1341 & 580.6 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} -0.8475 & 0.5307 & -0.0056 \\ -0.1726 & -0.2856 & -0.9427 \\ -0.5019 & -0.7980 & 0.3337 \end{pmatrix} \qquad t = \begin{pmatrix} 0.018 \\ 0.1461 \\ 0.3556 \end{pmatrix} \qquad error = 0.3142$$

To visualize the reprojected points I used the *visualization_reprojected_points.m* function, which was provided in the package. Instead, to visualize all the checkerboard points on the grid, I developed another function called *visualization_all_points.m*, which calculates the projection of all points in the grid (both xz and yz planes), knowing that all squares have a side length of 27 mm.

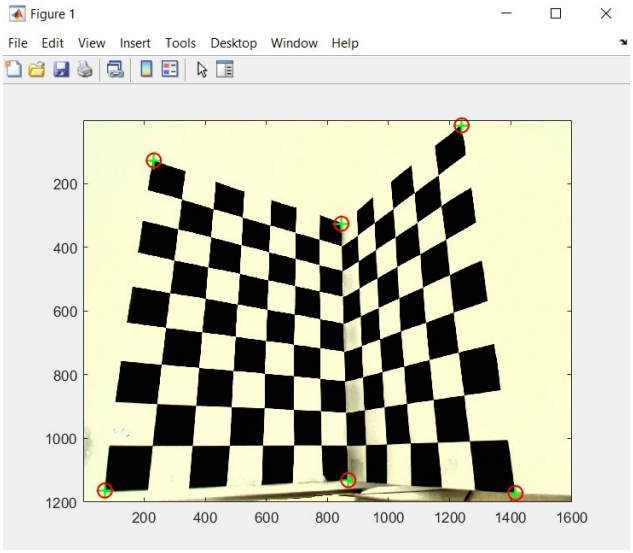


Figure 2 — Projection of the 6 edge corners of the checkerboard by means of the DLT algorithm ('+' represent the selected points, 'o' represent the projected points)

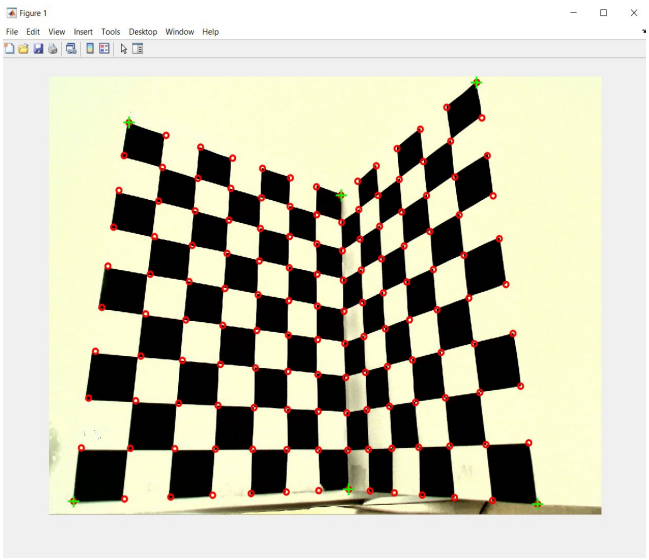


Figure 3 — Projection of all the corners of the checkerboard by means of the DLT algorithm and *visualization_all_points.m* function.

Using unnormalized points could lead the projection points to be way further from their actual position (however, this depends a lot also on the calibration points we choose). In my case, performing the DLT algorithm with unnormalized points led to a very high reprojection error, therefore this means that if we want the algorithm to perform well, points must be normalized.

To check whether the algorithm was actually working or not, I performed other two simulations, the first by picking 10 random corner points on the two planes, and the second by selecting all the points of the checkerboard. In the first case I obtained an error of 2.1733, while in the second an error of 2.707. Hence, I came to the conclusion that the more calibration points I selected for the calibration, the higher would have been the reprojection error.

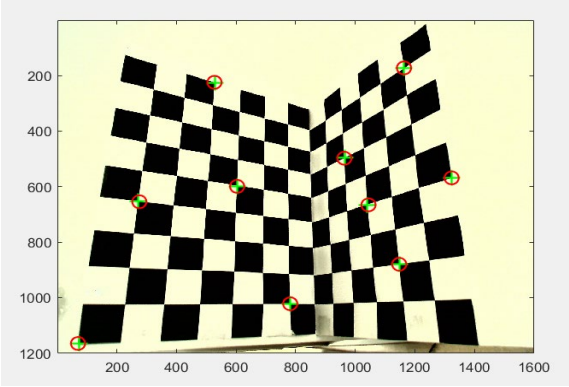


Figure 4 — Projection of 10 random corners of the checkerboard by means of the DLT algorithm

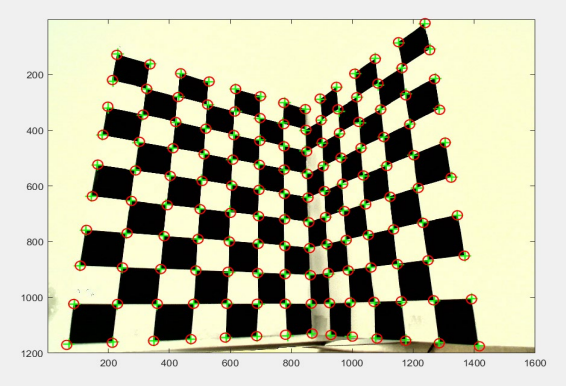


Figure 5 — Projection of all the corners of the checkerboard by means of the DLT algorithm

3. Gold Standard Algorithm

To run the Gold Standard Algorithm, I first normalize the points and then I run the DLT algorithm in order to obtain the normalized projection matrix P. After that, the algorithm minimizes the geometric error $d(\hat{x}_i\ ,\ \hat{P}\hat{X}_i)^2$ with respect to the matrix P, thanks to the function *fminsearch*:

$$\min_{\hat{P}} f = \min_{\hat{P}} \sum_{i=1}^{n} d\left(\hat{x}_i, \hat{P}\hat{X}_i\right)^2 = \min_{\hat{P}} \sum_{i=1}^{n} \left(\hat{x}_i - \hat{P}\hat{X}_i\right)^2$$

After having found the matrix P that minimizes the geometric error, the next steps require to obtain the values of K, R, C, t and the reprojection error, in the same way as it was done in DLT algorithm.

$$P = \begin{pmatrix} -2580.6 & 121 & 419 & 519.1 \\ 878.8 & 1422.4 & 1798.9 & 676.3 \\ 0.8 & 1.3 & 0.6 & 0.6 \end{pmatrix} \qquad K = \begin{pmatrix} 1338 & 9.6 & 796.4 \\ 0 & 1341 & 580.3 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R = \begin{pmatrix} -0.8475 & 0.5307 & -0.0056 \\ -0.1727 & -0.2858 & -0.9426 \\ -0.5019 & -0.7979 & 0.3339 \end{pmatrix} \qquad t = \begin{pmatrix} 0.0181 \\ 0.1461 \\ 0.3556 \end{pmatrix} \qquad error = 0.2633$$

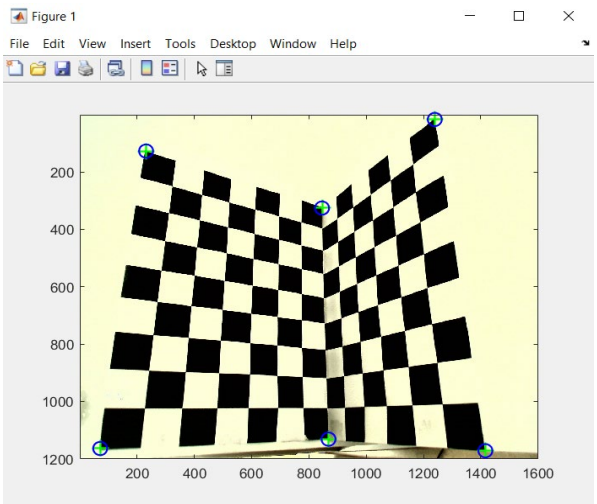


Figure 6 — Projection of the 6 edge corners of the checkerboard by means of the Golden Standard algorithm.

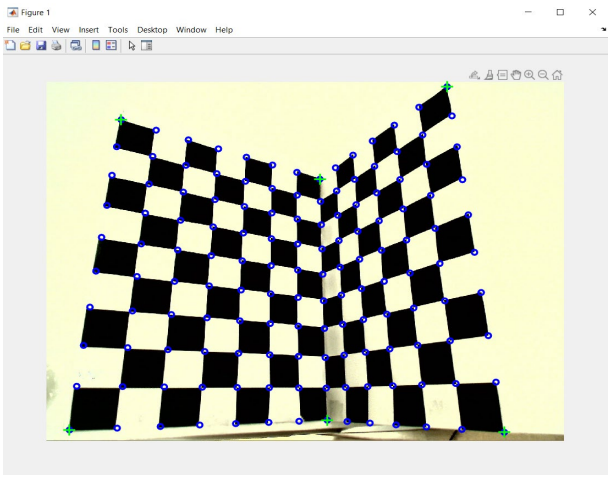
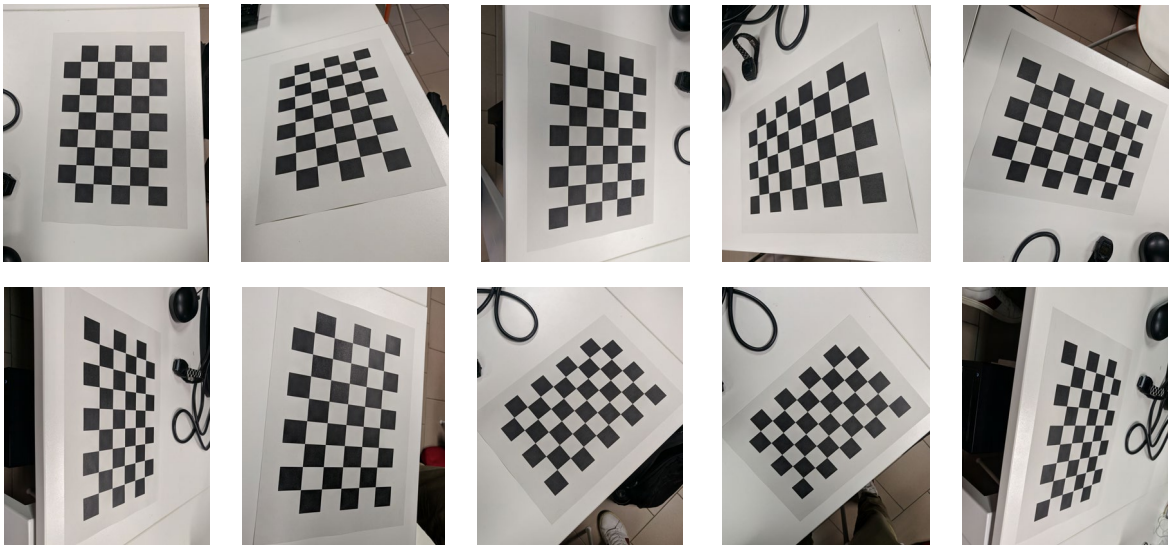


Figure 7 — Projection of all the corners of the checkerboard by means of the Golden Standard algorithm and *visualization_all_points.m* function.

From the images above is not possible to spot a tangible difference between the DLT and the Golden Standard algorithm. However, by looking at the reprojection error, it is possible to see that in this second case the error is lower, as we expected (0.2633 with GS algorithm VS 0.3142 with DLT algorithm). This confirms the fact that, in general, the Golden Standard algorithm actually performs better that the DLT algorithm.

4. MATLAB Calibration Toolbox

In order to achieve valuable results with the MATLAB Calibration Toolbox, you need to provide at least 10-20 images to the program. I printed a basic checkerboard and took 10 different pictures from different angulations, as shown below:



The toolbox is very intuitive and easy to use, and helps you to achieve good results with just few simple steps. Moreover, it also has some useful graphical illustrations that helps you better understand and visualize the calibration problem.

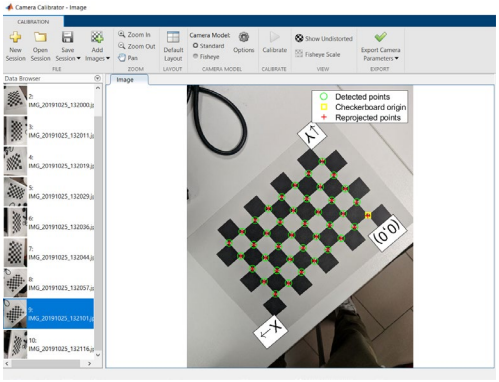


Figure 8 — Calibration Toolbox window in Matlab.

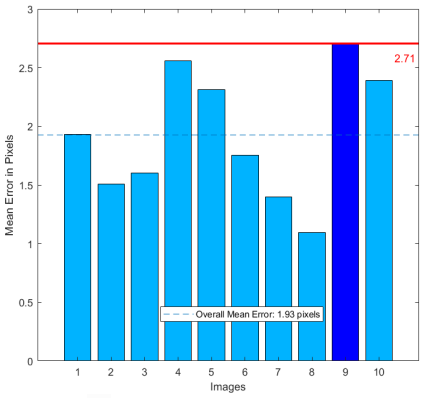


Figure 9 — Reprojection errors of the 10 images.

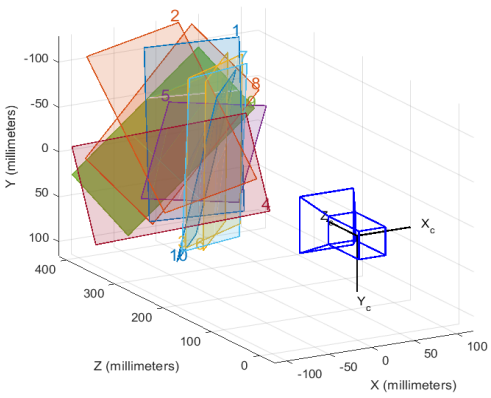


Figure 10 — "Camera centric" visualization of the camera calibration.

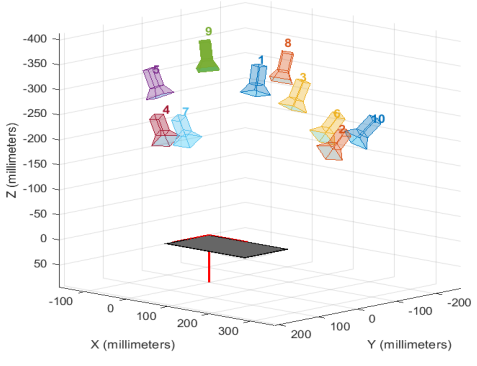


Figure 11 — "Pattern centric" visualization of the camera calibration.