Computer Vision Assignment 1: Calibration

Charlotte Moraldo

1. Data Normalization

1.1 Moving centroids to the origin

In the function `normalization`, we begin by computing the centroids of the image points `xy` and object points `XYZ` the following way:

$$C_{xy} = \begin{bmatrix} \frac{xy[1,1] + xy[1,2] + ... + xy[1,n]}{n} \\ \frac{xy[2,1] + xy[2,2] + ... + xy[2,n]}{n} \end{bmatrix} \qquad C_{XYZ} = \begin{bmatrix} \frac{XYZ[1,1] + XYZ[1,2] + ... + XYZ[1,n]}{n} \\ \frac{XYZ[2,1] + XYZ[2,2] + ... + XYZ[2,n]}{n} \\ \frac{XYZ[3,1] + XYZ[3,2] + ... + XYZ[3,n]}{n} \end{bmatrix}$$

Where n is the number of points clicked. From these computed centroids, we can deduce the transformations matrix T_1 and U_1 such that when doing $T_1 \cdot xy$ and $U_1 \cdot XYZ$, we move the centroids to the origin:

$$T_1 = \begin{bmatrix} 1 & 0 & -C_{xy}[1] \\ 0 & 1 & -C_{xy}[2] \\ 0 & 0 & 1 \end{bmatrix} \qquad U_1 = \begin{bmatrix} 1 & 0 & 0 & -C_{XYZ}[1] \\ 0 & 1 & 0 & -C_{XYZ}[2] \\ 0 & 0 & 1 & -C_{XYZ}[3] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

1.2 Normalizing the Euclidian distance from origin

The second part of the normalization is transforming the input points such that the average Euclidian distance from the origin of $\sqrt{2}$ for the image points and $\sqrt{3}$ for the object points. To do so, we start by calculating the current average Euclidian distance for `xy` and `XYZ`, which we store in the variables `xy_dist` and `XYZ_dist`. The second transformation matrices T_2 and U_2 are thus simply:

$$T_2 = \frac{\sqrt{2}}{dist_{xy}} \qquad U_2 = \frac{\sqrt{3}}{dist_{XYZ}}$$

1.3 Final transformations matrices

The final transformations matrices such that the normalized points are $\widehat{\mathbf{x_i}} = T \cdot \mathbf{x_i}$ and $\widehat{\mathbf{X_i}} = U \cdot \mathbf{X_i}$ are therefore:

$$T = T_1 \cdot T_2 = \frac{\sqrt{2}}{dist_{xy}} \begin{bmatrix} 1 & 0 & -C_{xy}[1] \\ 0 & 1 & -C_{xy}[2] \\ 0 & 0 & 1 \end{bmatrix} \qquad U = U_1 \cdot U_2 = \frac{\sqrt{3}}{dist_{XYZ}} \begin{bmatrix} 1 & 0 & 0 & -C_{XYZ}[1] \\ 0 & 1 & 0 & -C_{XYZ}[2] \\ 0 & 0 & 1 & -C_{XYZ}[3] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. Direct Linear Transform

2.1 Compute normalized P with DLT Algorithm

In the function `dlt`, we must compute the normalized $\widehat{\mathbf{P}}$ using the DLT algorithm. We begin by computing the matrix $A \begin{bmatrix} w_i\widehat{\mathbf{X}}_\mathbf{i}^T & 0^T & -x_i\widehat{\mathbf{X}}_\mathbf{i}^T \\ 0^T & -w_i\widehat{\mathbf{X}}_\mathbf{i}^T & y_i\widehat{\mathbf{X}}_\mathbf{i}^T \end{bmatrix}$. This matrix is of dimension 2n x 12 and is such that $A\widehat{\mathbf{P}} = 0$. Therefore, we can deduce $\widehat{\mathbf{P}}$ as it is equal to the right-null vector of A. The right-null vector can be determined with the SVD decomposition of A: $A = U \cdot S \cdot V^T$, where U is an unitary matrix, S is a rectangular diagonal matrix, and V is also an unitary matrix. The far-left column of V (or the far bottom line of V^T) is the right-null vector of A, in our case this corresponds to a vector $\widehat{\mathbf{P}}$. The last thing left to do is to rearrange this vector into a matrix.

2.2 Compute and visualize reprojected points

In the function `runDLT`, we start by using the functions `normalization` and `dlt` in order to normalize the input data and to compute $\widehat{\mathbf{P}}$. We continue by denormalizing the camera matrix using the transformations matrices computed during normalization: $\mathbf{P} = \mathbf{T}^{-1}\widehat{\mathbf{P}}\mathbf{U}$. We can then factorize the camera matrix into K, R, and C, using the function `decompose(P)`. t can be computed the following way: $t = -R \cdot \tilde{C}$.
Finally, we can visualize the reprojected points of all checkerboard corners as well as compute the reprojection error. This error is found by calculating the average of the squared distance between initial and reprojected points.
Below is an example of the visualisation of the reprojected points of all the checkerboard corners (red + marquers for the xz plane, green + marquers for the yz one), as well as the selected points (cyan o marquers) and their reprojection (blue o marquers).

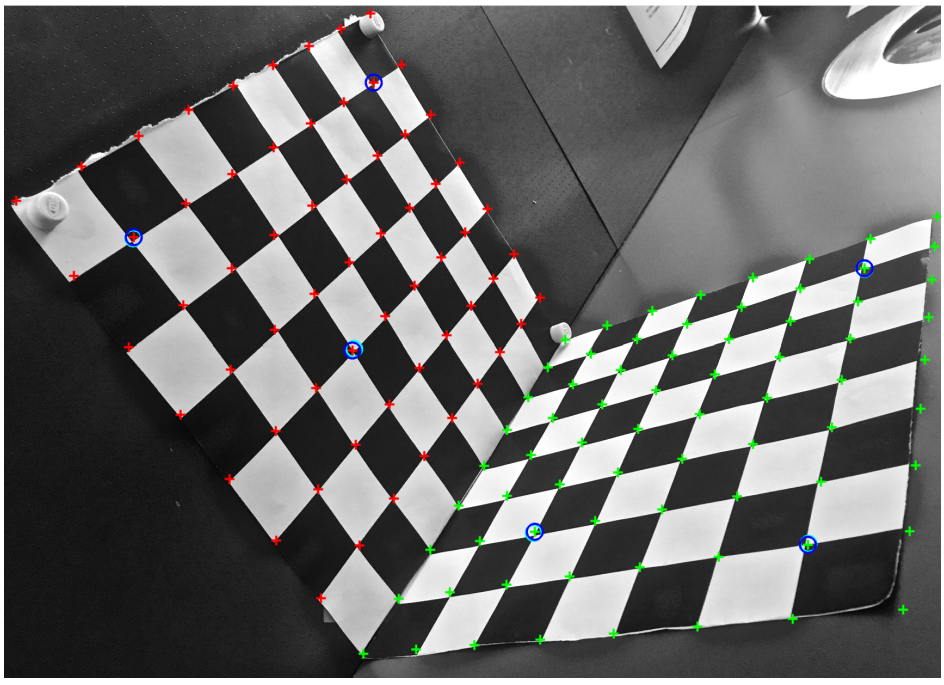

Figure 1: DLT

The obtained camera matrix is:

$$K = \begin{bmatrix} 3344 & 46.6 & 1724.4 \\ 0 & 3393.7 & 1256.7 \\ 0 & 0 & 1 \end{bmatrix}$$

and the reprojection error is 4.1122.

After a few different tests, we can observe that the quality of the calibration depends a lot on the points that we choose to click. Here is another example:

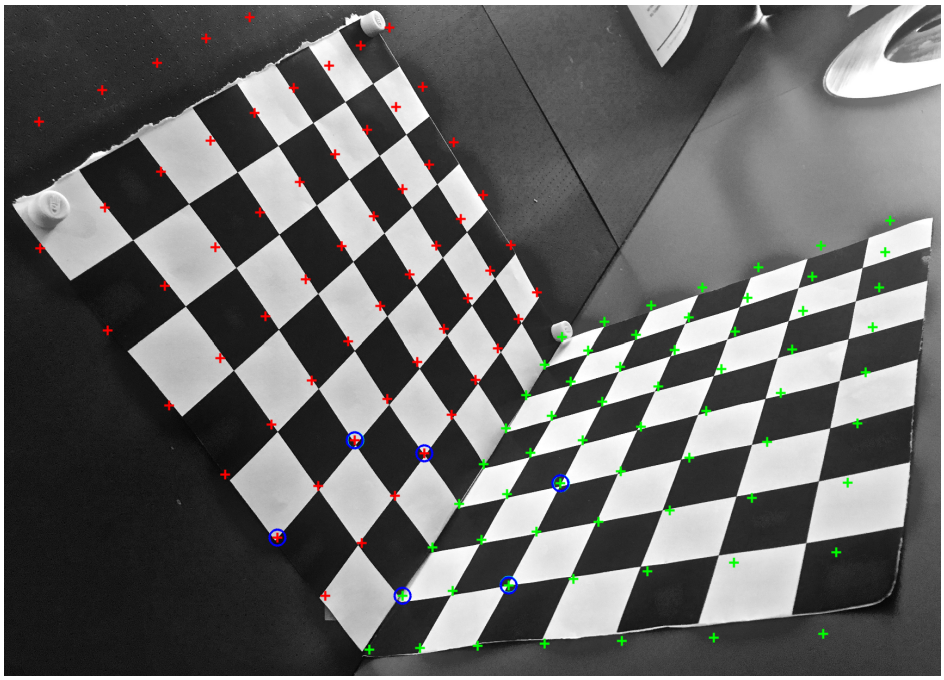

Figure 2: DLT with bad calibration

While figure 1 shows a pretty good calibration, the projected points on figure 2 are very far away from the original ones. Indeed, in the first example, the 6 points were chosen such that most of the pattern was covered: 4 corners, plus middle of each plane. On the opposite, in this second example, all the points selected are in the same central area. Because the distortion is less visible in a small area like the one covered in figure 2, the quality of the calibration is decreased a lot.
However, the surprising thing is that the error computed isn't higher, but much lower! We indeed find that the reprojection error is equal to 2.5738. This is due to the fact that we only compute the error over the clicked points, and not over the entire checkerboard corners. Therefore, since the points selected in figure 2 are very close to each other, their reprojection is closer to their original position. However, this is absolutely not representative of the calibration quality and for a better accuracy, it would be more relevant to compute the error over all the reprojected checkerboard corners. Therefore, in this case, the visual reprojected points are much more representative of the quality of the calibration.

3. Gold Standard Algorithm

Similarly as what was done in with the DLT, in the function `runGoldStandard` we start by using `normalization` and `dlt` in order to normalize the input data and to compute $\widehat{\mathbf{P}}$. We continue by denormalizing the camera matrix using the transformations matrices computed during normalization: $\mathbf{P} = \mathbf{T}^{-1}\widehat{\mathbf{P}}\mathbf{U}$, and we can then visualize the clicked points and their reprojection.
The second part of the algorithm is the optimization of $\widehat{\mathbf{P}}$. This is done in by minimizing the squared geometric error, using the function `fminGoldStandard`. After doing so, similarly as in DLT, we denormalize the camera matrix and decompose it into K, R, C, and we can then compute t and the reprojection error.

Below is the visualization of the reprojection before and after optimization, for the same points chosen with the DLT:

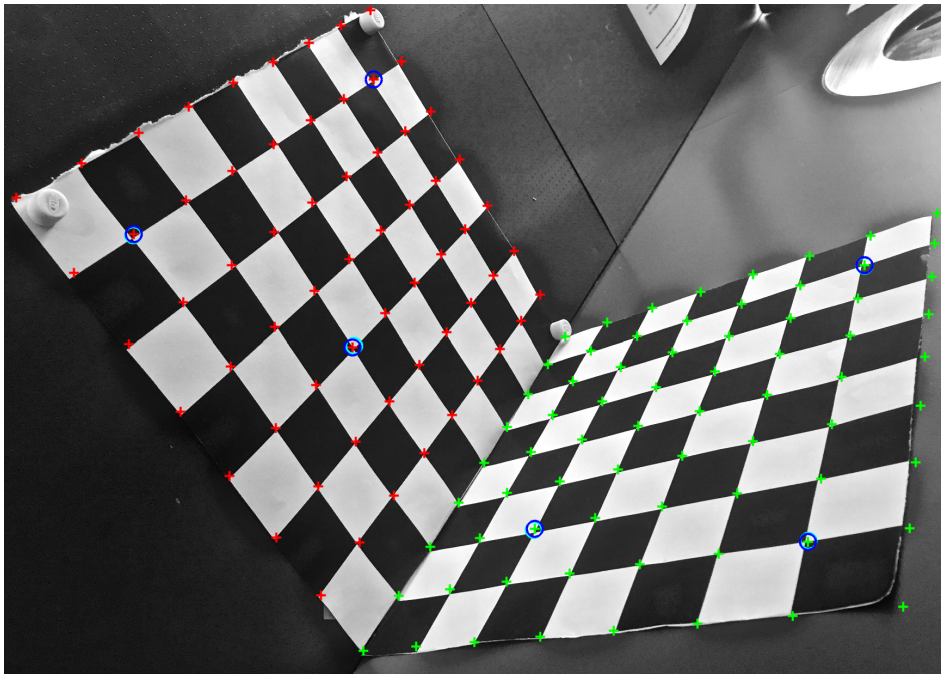

Figure 3: Gold Standard before optimization

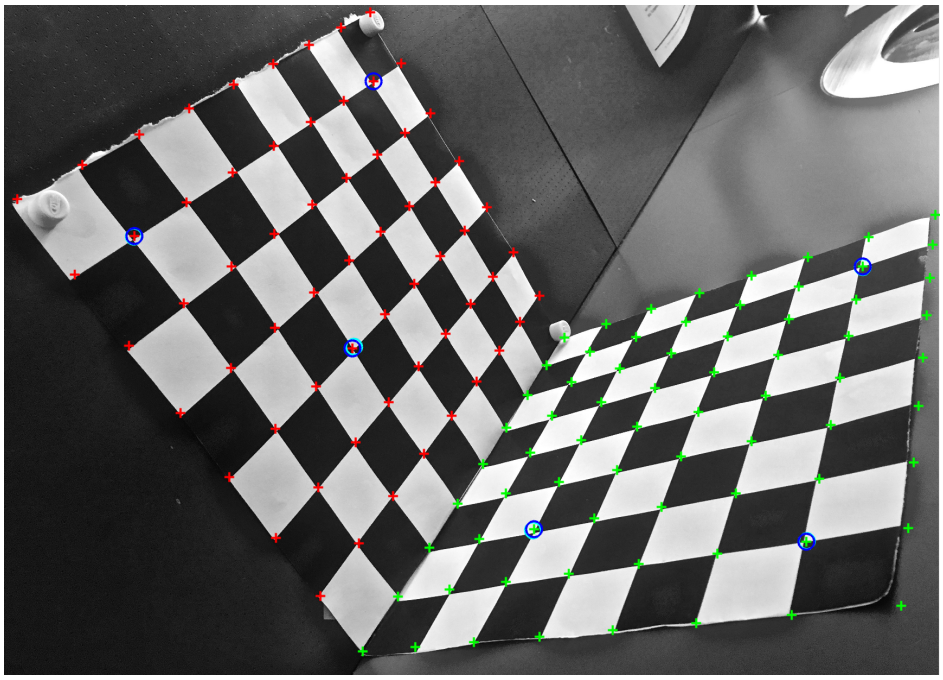

Figure 4: Gold Standard after optimization

We can see that the optimization doesn't make a real visible difference. This is due to the fact that after being normalized, the camera matrix doesn't change much:

$$P_{norm.} = \begin{bmatrix} 0.2779 & -0.4199 & -0.2085 & 0.0260 \\ 0.4499 & 0.1465 & 0.2854 & 0.0557 \\ 0.0352 & 0.0758 & -0.0914 & -0.6152 \end{bmatrix} \quad P_{norm.,opt.} = \begin{bmatrix} 0.2771 & -0.4212 & -0.2085 & 0.0252 \\ 0.4499 & 0.1469 & 0.2855 & 0.0555 \\ 0.0354 & 0.0754 & -0.0915 & -0.6154 \end{bmatrix}$$

However, we can see that in this case, the error calculated is equal to: **error = 3.8539**, which is smaller than with the DLT algorithm (for the same clicked points).
Furthermore, let's note that the camera matrix remains very close to what was obtained with the DLT, confirming our results:

$$K = \begin{bmatrix} 3352.5 & 38.2 & 1730.2 \\ 0 & 3399.2 & 1256.4 \\ 0 & 0 & 1 \end{bmatrix}$$

After several other tests, we realise that by using a higher number of points, an even better calibration and lower error can be achieved. For example, in the following pictures, 10 "good" points (well distributed over the two planes of the pattern) were chosen and the error was approximately equal to: **error = 3.4332**.

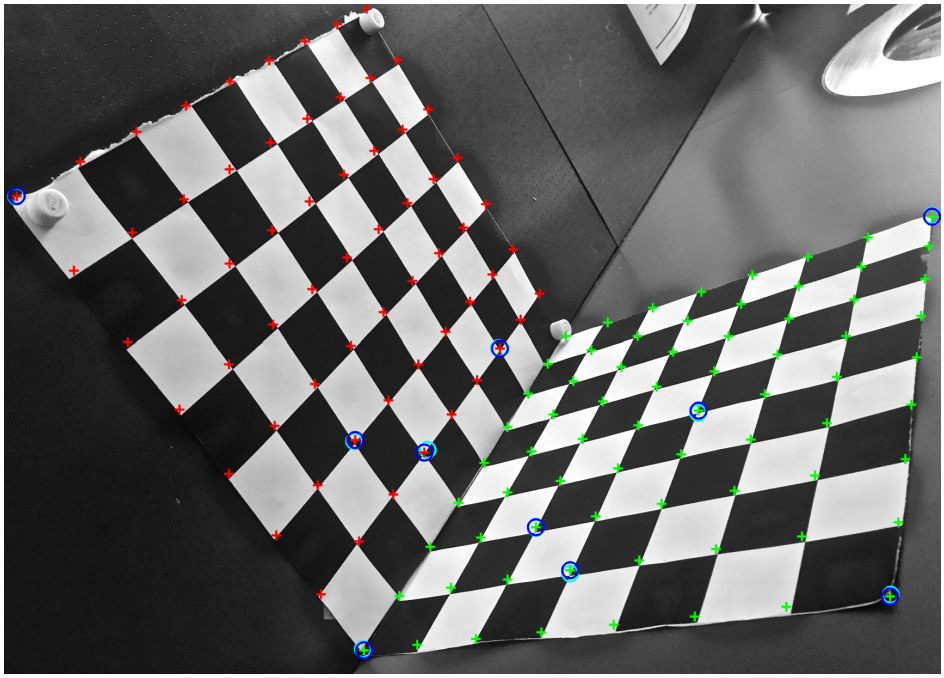

Figure 5: Gold Standard Algorithm with 10 points

4. Bouget's Calibration Toolbox

Here are the 10 images I used with the calibration toolbox:

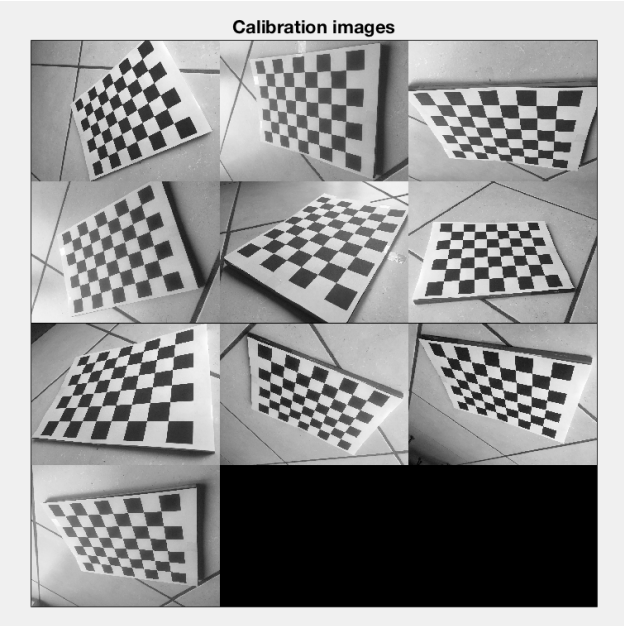

Figure 6

I decided to track the image number 7 to illustrate the different steps of the calibration. After the corner extraction, we can calibrate the camera, resulting in the reprojected image:

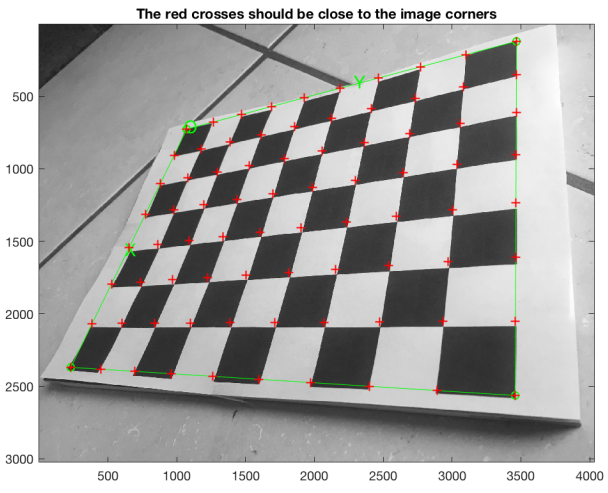

Figure 7: Corner extraction

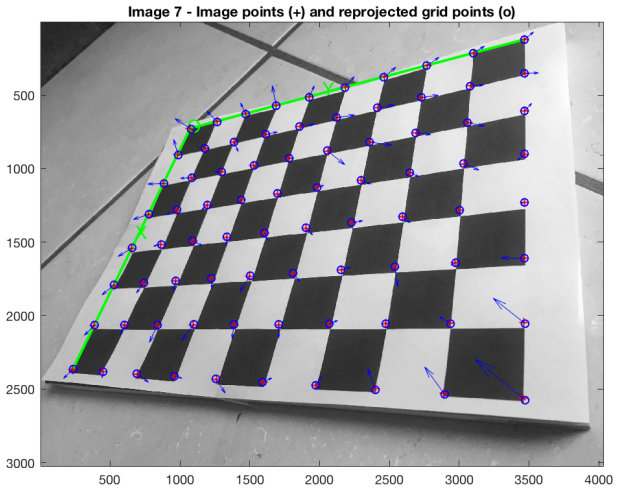

Figure 8: Image reprojection

The calibration is done in 2 steps: initialization, then nonlinear optimization. Here are the calibration parameters I obtained:

```
Calibration parameters after initialization:

Focal Length:          fc = [ 3461.78744   3461.78744 ]
Principal point:       cc = [ 2015.50000   1511.50000 ]
Skew:             alpha_c = [ 0.00000 ]   => angle of pixel = 90.00000 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000   0.00000   0.00000 ]

Main calibration optimization procedure - Number of images: 10
Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...12...13...14...15...16...17...18...19...20...21...22...23...24..
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 3460.68712   3421.60077 ] +/- [ 10.65660   14.07549 ]
Principal point:       cc = [ 2045.90845   1512.71455 ] +/- [ 23.66487   15.13716 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ 0.01698   -0.03435   0.00161   0.00218  0.00000 ] +/- [ 0.01630   0.04761   0.00170   0.00230  0.00000 ]
Pixel error:          err = [ 3.53352   2.56846 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figure 9: Calibration

From these parameters, we can reconstruct the camera matrix K, since:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \qquad fc \simeq \begin{bmatrix} 3461 \pm 11 \\ 3422 \pm 14 \end{bmatrix} \qquad cc \simeq \begin{bmatrix} 2046 \pm 24 \\ 1513 \pm 15 \end{bmatrix}$$

Therefore, we can approximate K the following way:

$$K_{toolbox} \simeq \begin{bmatrix} 3440 & 0 & 2046 \\ 0 & 3440 & 1513 \\ 0 & 0 & 1 \end{bmatrix}$$

And as a reminder, here are the camera matrices we obtained with the DLT and the GoldStandard algorithm:

$$K_{DLT} = \begin{bmatrix} 3344 & 46.6 & 1724.4 \\ 0 & 3393.7 & 1256.7 \\ 0 & 0 & 1 \end{bmatrix}$$

$$K_{GoldStandard} = \begin{bmatrix} 3352.5 & 38.2 & 1730.2 \\ 0 & 3399.2 & 1256.4 \\ 0 & 0 & 1 \end{bmatrix}$$

$K_{toolbox}$, K_{DLT}, and $K_{GoldStandard}$ are all very close! This confirms that our implementation of DLT and GoldStandard gives us correct results, even though the calibration could be further optimized to obtain a lower error.

From figure 9, we can observe that the pixel error obtained with the toolbox after optimization is between **error = 3.53352** and **error = 2.56846**, which is lower than the results obtained with the DLT and GoldStandard functions. This result was predictable, as the toolbox used has a much better optimization than the one we implemented with DLT and the GoldStandard.
The error of all the pictures can be visualized in figure 10, below. The toolbox also allows us to view the extrinsic parameters (relative positions of the grids with respect to the camera) in the form of a 3D plot, relatively to the camera (figure 11) or to the world (figure 12):

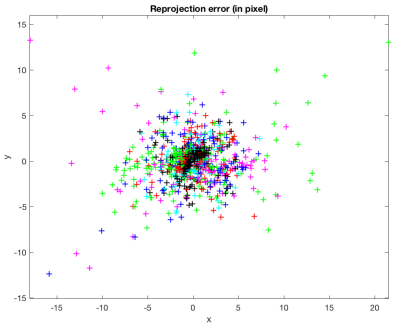

Figure 10

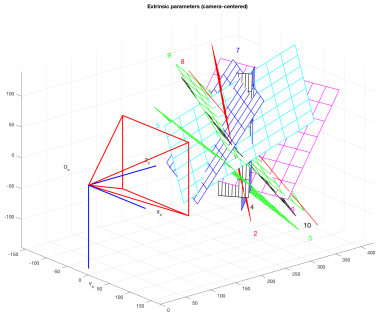

Figure 11

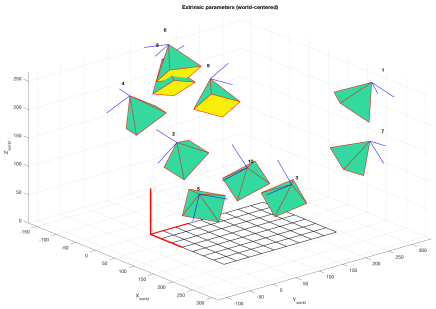

Figure 12

With this first optimization, the reprojection error over the 10 images remains very large, even though it is already smaller than the one obtained with our implementation. The solution proposed by the tutorial is to recompute the image corners on all images automatically: the re-projected grid is used to as initial guess locations of the corners.
After doing so, we can re-calibrate, and this time we obtain a much smaller reprojection error, all the while keeping the same calibration matrix:

```
Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 3460.95782   3422.89186 ] +/- [ 5.73792   7.58439 ]
Principal point:       cc = [ 2048.59720   1512.53773 ] +/- [ 12.73010   8.13481 ]
Skew:             alpha_c = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc = [ 0.01693   -0.03841   0.00137   0.00231  0.00000 ] +/- [ 0.00877   0.02560   0.00091   0.00123  0.00000 ]
Pixel error:          err = [ 1.69741   1.62445 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).
```

Figure 13: 2nd Calibration Parameters

The error significantly decreased with this correction, it is almost twice smaller! It went from $[3.53352, 2.56846]$ to $[1.69741, 1.62445]$, and can be viewed for all 10 pictures on this 2D plot:

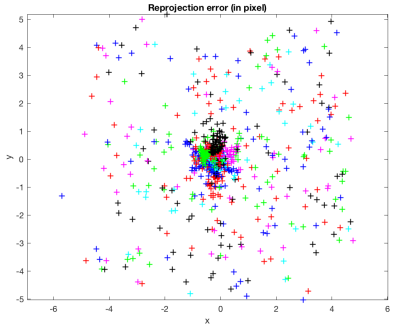

Figure 14

In the same way as before, we can also see the reprojected grid points, as well as the extrinsic parameters. However, the real difference can mostly be seen in the error plot shown above.

Conclusion

The results that we obtained in the implementation of the DLT and the Gold Standard algorithms can already offer a reliable calibration, under the condition that the points are well chosen and that there are a lot of them. The error remains however very large, and there is a lot of room for improvement. Bouget's Calibration Toolbox offers a very good alternative, as its optimization and thus calibration are much better, allowing us to highly reduce the reprojection error in comparison to our implementation.
To finish, we can point out that the camera matrix K computed with the DLT and GoldStandard algorithms is very close to the one calculated by the toolbox, which confirms our results and our implementation.