Camera Calibration
Computer Vision Lab 1

Álvaro Belmonte Baeza
Student ID 19-940-386

October 2, 2020

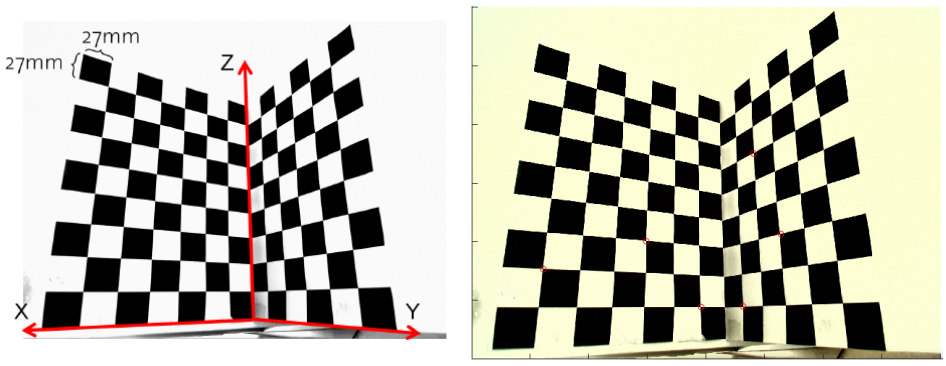
Abstract

In this first assignment of the Computer Vision course, the topic of Camera Calibration is addressed. First, the 2D-3D point correspondences obtained using the provided framework will be shown. Secondly, the task of data normalization and the proposed implementation it's presented. Then, the implementation of Direct Linear Transformation (DLT) algorithm is presented and the obtained results are discussed. Finally, the Gold Standard algorithm is implemented, particularly focusing on the optimization step, and the results are discussed and compared with the obtained using DLT alone.

0 Obtain Point Correspondences

In order to calibrate a camera, first of all a set of N 2D-3D point correspondences that allows us to compute the projection matrix value in the following sections is needed.

To do so, a convenient MATLAB function is provided by the instructors. This function pops a window with two perpendicular calibration patterns, and allows to click on a determined region and assign 3D coordinates to the clicked pixel of the image. The dimensions and coordinate frames of the calibration pattern used are as shown in Figure 1a, and it is also known that for the DLT algorithm at least 6 correspondences are needed. Taking this into account, the selected correspondences are displayed in Figure 1b.



(a) Pattern dimensions and Coord. Frame (b) Initial correspondence points

Figure 1: Selection of point correspondences in calibration pattern

For these selected points in the shown coordinate frame, the following 3D coordinates are assigned:

$$\mathbf{X} = \begin{bmatrix} 0.027 & 0 & 0.081 & 0 & 0.162 & 0 \\ 0 & 0.027 & 0 & 0.081 & 0 & 0.054 \\ 0.027 & 0.027 & 0.081 & 0.081 & 0.054 & 0.0162 \end{bmatrix}$$

Where each column corresponds to the 3D coordinates $\boldsymbol{X}_i = [X_i, Y_i, Z_i]^T$, measured in meters. For convenience, the 2D points clicked in the image, $\boldsymbol{x}$, and the 3D points indicated are stored in a .mat file *correspondences.mat*, attached to this report.

With our correspondences already selected, we can now proceed to the following steps in order to obtain our calibration matrix.

1 Data Normalization

In this section, the normalization process of the correspondences data and its implementation are discussed. The goal of this step is improve the numerical stability by transforming the input points $\boldsymbol{x}_i$, $\boldsymbol{X}_i$ to normalized points $\hat{\boldsymbol{x}}_i$, $\hat{\boldsymbol{X}}_i$ which have zero mean and unit distance to origin. To do so, matrices $\boldsymbol{T}$ and $\boldsymbol{U}$ need to be computed, such that $\hat{\boldsymbol{x}}_i = \boldsymbol{T}\boldsymbol{x}_i$ and $\hat{\boldsymbol{X}}_i = \boldsymbol{U}\boldsymbol{X}_i$.

First of all, the centroid of the input points is computed such that $C = \frac{\sum_{i=1}^{k} \boldsymbol{x}_i}{k}$. Then, input points are shifted so that the centroid equals to the origin by substracting the current centroid values to the point coordinates. With these shifted points, the scale factor for normalization is computed by computing the mean euclidean distance from the shifted points to the origin, such that $\sigma = \frac{\sum_{i=1}^{k} \sqrt{\boldsymbol{x}^2_{shifted,i} - \mathbf{0^2}}}{k}$.

After this series of computations, a series of interesting values have been obtained: $\boldsymbol{C}_{xy} = [C_x, C_y]^T$ and $\boldsymbol{C}_{XYZ} = [C_X, C_Y, C_Z]^T$ are the centroids of the 2D and 3D input points respectively. In addition, the scale factors σ_{2D} and σ_{3D} have also been obtained.

Thus, the normalization matrices $\boldsymbol{T}$ and $\boldsymbol{U}$ can be built as follows:

$$\boldsymbol{T^{-1}} = \begin{bmatrix} \sigma_{2D} & 0 & C_x \\ 0 & \sigma_{2D} & C_y \\ 0 & 0 & 1 \end{bmatrix}; \boldsymbol{U^{-1}} = \begin{bmatrix} \sigma_{3D} & 0 & 0 & C_X \\ 0 & \sigma_{3D} & 0 & C_Y \\ 0 & 0 & \sigma_{3D} & C_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Now, data normalization can be completed by just premultiplying these matrices to the input points, such that $\hat{\boldsymbol{x}}_i = \boldsymbol{T}\boldsymbol{x}_i$ and $\hat{\boldsymbol{X}}_i = \boldsymbol{U}\boldsymbol{X}_i$.

The implemented code can be found in the *normalization.m* file attached, as it is only an implementation of the above mentioned equations, and it has not been considered to be included in this report.

Now that the normalized values have been computed, the next step is to implement the DLT algorithm.

2 Direct Linear Transform

In this section, the implementation of the DLT algorithm is presented. With this algorithm, we can obtain our camera model minimizing the algebraic error.

First, normalization of the point correspondences is carried out by calling the function implemented in last section.

Then, the normalized camera matrix $\hat{\boldsymbol{P}}$ is computed by inserting each 2D-3D point correspondences in equation (1):

$$\begin{bmatrix} \hat{\boldsymbol{X}}_i^T & \mathbf{0}_{1X4} & -\hat{x}_i\hat{\boldsymbol{X}}_i^T \\ \mathbf{0}_{1X4} & -\hat{\boldsymbol{X}}_i^T & \hat{y}_i\hat{\boldsymbol{X}}_i^T \end{bmatrix} \begin{pmatrix} \hat{\boldsymbol{P}}^1 \\ \hat{\boldsymbol{P}}^2 \\ \hat{\boldsymbol{P}}^3 \end{pmatrix} = \mathbf{0} \tag{1}$$

Thus, two rows will be added to the left matrix in equation (1) for each correspondence. Hence, it results in a $2N \times 12$ matrix A for N correspondences, whose right nullvector equals to the solution for $\hat{\boldsymbol{P}}$. This nullvector is computed with the Singular Value Decomposition (SVD) of A.

The implementation in MATLAB of this part of the algorithm is the following:

```
function [P_normalized] = dlt(xyn, XYZn)
%computes DLT, xy and XYZ should be normalized before calling this function

% 1. For each correspondence xi <-> Xi, computes matrix Ai
for i = 1:2:(2*size(XYZn,2))
    A(i,:) = [XYZn(:,round(i/2))', zeros(1,4), ...
        -xyn(1,round(i/2)).*(XYZn(:,round(i/2))')];
    A(i+1,:) =[ zeros(1,4), -XYZn(:,round(i/2))', ...
        xyn(2,round(i/2)).*(XYZn(:,round(i/2))')];
end

% 2. Compute the Singular Value Decomposition of Usvd*S*V' = A
[Usvd, S, V] = svd(A);

% 3. Compute P_normalized (=last column of V if D = matrix with positive
% diagonal entries arranged in descending order)
P_column = V(:,size(V,2));
% Rearrange values so that we end with a 3x4 matrix
P_normalized = [P_column(1:4)'; P_column(5:8)';P_column(9:12)'];
end
```

It can be seen that first the $2N \times 12$ matrix is computed using equation (1), and then the SVD of A is obtained. Finally, by reorganizing the elements in the last column of matrix V obtained from the SVD, the normalized camera matrix, $\hat{\boldsymbol{P}}$, is computed, and using the matrices T and U from the normalization step, the actual camera matrix is obtained such that $\boldsymbol{P} = \boldsymbol{T}^{-1}\hat{\boldsymbol{P}}\boldsymbol{U}$.

At this point, the following step is factorizing the camera matrix to obtain separately the intrinsic camera matrix K, a rotation matrix R and the camera center $\boldsymbol{C}$. This factorization is computed as shown in equation (2).

$$\boldsymbol{P} = [\boldsymbol{M}| - \boldsymbol{M}\boldsymbol{C}] = K[R| - R\boldsymbol{C}] \tag{2}$$

Where $M = KR$. On the one hand, M can be decomposed using an RQ-decomposition, that will give a upper-diagonal matrix like K and a orthogonal matrix like R. On the other hand, the camera center is the point that satisfies $P\boldsymbol{C} = \mathbf{0}$, which can be computed again using the SVD, this time applied to P.

This step is implemented using the QR-decomposition included in MATLAB. However, in order to obtain the correct values for K and R, the decomposition is applied to $\boldsymbol{M}^{-1}$ and then the obtained values, $R^{-1} and K^{-1}$, are inverted again to obtain the actual matrices.

In addition, the computed matrices are adjusted so that the diagonal of K is non-negative and $|R| = 1$. Finally, the translation vector $\boldsymbol{t}$ is computed so that $\boldsymbol{t} = -R\boldsymbol{C}$. All the decomposition process is implemented in the *decompose.m* function as follows:

```
function [K, R, t] = decompose(P)
%Decompose P into K, R and t using QR decomposition

% Compute R, K with QR decomposition such M=K*R
M = P(1:3,1:3);
[q,r] = qr(inv(M));
R = inv(q);
K = inv(r);

% Compute camera center C=(cx,cy,cz) such P*C=0
% C is the right nullvector of P, which can be obtained using SVD
[Usvd, S, V] = svd(P);
C = V(:,size(V,2));

% normalize K such K(3,3)=1
K = K./K(3,3);

% Adjust matrices R and Q so that the diagonal elements of K (= intrinsic matrix) ...
    are non-negative values and R (= rotation matrix = orthogonal) has det(R)=1
% Obtain diagonal matrix to adjust sign of K
Tk = diag(sign(diag(K)));
%Adjust sign
K = K*Tk;
R = inv(Tk)*R;

if det(R) == -1
    R = -R;
end
% Compute translation t=-R*C
t = -R*C(1:3);

end
```

To conclude this section, the resulting camera instrinsic and extrinsic matrices as well as a visualization of the reprojected points are presented. After executing all previous steps, the results are the following:

$$\boldsymbol{P} = \begin{bmatrix} 2574 & -522.1 & -310 & -517.7 \\ 894.1 & 1095.6 & 1901.8 & -673 \\ 0.9 & 0.9 & -0.5 - 0.6 & \end{bmatrix}$$

$$\boldsymbol{K} = \begin{bmatrix} 1655 & 50 & 1067 \\ 0 & 1689 & 513 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} -0.731 & 0.681 & -0.049 \\ -0.196 & -0.277 & -0.9407 \\ -0.654 & -0.678 & 0.336 \end{bmatrix}$$

$$\boldsymbol{t} = \begin{bmatrix} -0.052 \\ 0.147 \\ 0.4 \end{bmatrix}$$

$$\boldsymbol{reprojection\_error} = 1.55px$$

Analyzing the obtained results, specially some interesting intrinsic parameters can be discussed. It can be seen from the results of K that the principal point is located at coordinates $[x_0, y_0] = [1067, 513]$ which is slightly different that the actual center of the image. In addition, the ratio $\frac{f_x}{f_y} = 0.98$ shows that the scale is almost 1, and therefore a good approximation has been obtained. Finally, the average reprojection error, computed as the average difference between the projected value and the manual inputs has a value of 1.55 pixels, which is a good indicator of the calibration.

It can be checked visually that the reprojected points overlap with the clicked points almost perfectly in Figure 2.

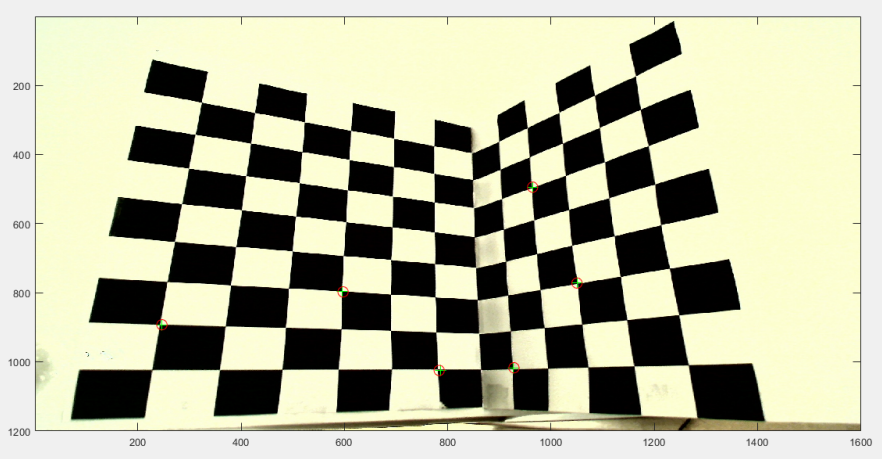


Figure 2: Reprojected points using DLT algorithm

With this, it can be seen that even though the points weren't accurately clicked at the corners of the checkerboard, the algorithm still provides a good response as it is shown in Figure 2.

An extra question arises in this section. What would happen if the unnormalized points were used in the computations? The reprojection in Figure 3 is obtained when omitting the normalization step on the points.It can be seen that the computed reprojection is slightly less accurate, due to the loss of stability in the data. However, the result is still valid, as it should be, but less stable when new data is provided.

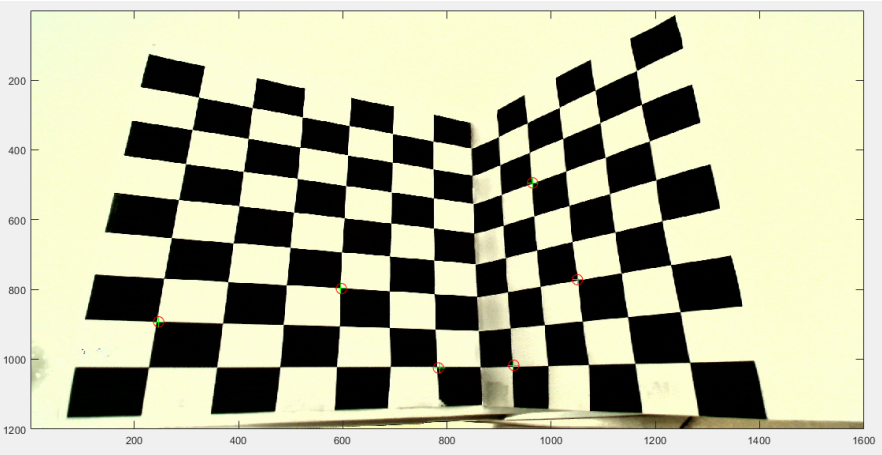


Figure 3: Reprojected points using DLT algorithm without normalization

3 Gold Standard Algorithm

In this third section of the report, the Gold Standard Algorithm is implemented, and its results analysed and compared to the ones obtained using the DLT algorithm only.

The main difference between these two approaches is that the Gold Standard algorithm minimizes the geometric error $\sum_i d(\hat{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_i)^2$ between the normalized clicked points and the projected object points. This error is computed using the Euclidean distance between this points.

The first step is normalizing the data and running the DLT algorithm until the point where matrix $\hat{\boldsymbol{P}}$ should be denormalized. Then, the estimated camera matrix $\hat{\boldsymbol{P}}$ is refined by minimizing the geometric error. This cost function is minimized using *fminsearch* from MATLAB. This optimization process is implemented as follows in the attached code:

```
%minimize geometric error to refine P_normalized
% TODO fill the gaps in fminGoldstandard.m
pn = P_normalized;
for i=1:20
    [pn] = fminsearch(@fminGoldStandard, pn, [], xy_normalized, XYZ_normalized);
end

function f = fminGoldStandard(pn, xy_normalized, XYZ_normalized)
%reassemble P
P = pn;

% Apply projection matrix P
xyz_projected=P*XYZ_normalized;

% Compute Inhomogeneous projected points
NB_PTS=size(XYZ_normalized,2);
xy_projected=zeros(2,NB_PTS);
for i=1:NB_PTS
    xy_projected(1,i)=xyz_projected(1,i)./xyz_projected(3,i); % compute ...
        inhomogeneous coordinates x=x/z
    xy_projected(2,i)=xyz_projected(2,i)./xyz_projected(3,i); % compute ...
        inhomogeneous coordinates y=y/z
```

```
21 end
22
23 % TODO compute reprojection errors
24 distances = sqrt((xy_normalized(1,:)-xy_projected(1,:)).^2 + ...
       (xy_normalized(2,:)-xy_projected(2,:)).^2);
25
26 % TODO compute cost function value
27 f = sum(distances.^2);
28 end
```

An iterative call to MATLAB's *fminsearch* function is done in order to perform the optimization process. This function starts with a provided value, in this case $\hat{\boldsymbol{P}}$, and attempts to find a local minimizer for the function specified as cost function.

This cost function provided to *fminsearch* computes the projected points using the normalized calibration matrix, and then obtains the value of the cost function by computing the sum of the squared euclidean distances, as stated previously in this section.

After the optimization process, $\hat{\boldsymbol{P}}$ is finally denormalized and factorized the same way as it was performed in the DLT algorithm, finally obtaining the camera calibration matrix, intrinsic and extrinsic parameters. The resulting values are shown here:

$$\boldsymbol{P} = \begin{bmatrix} 2573 & -530 & -309 & -517.6 \\ 892.1 & 1089.3 & 1903 & -673 \\ 0.9 & 0.9 & -0.5 - 0.6 & \end{bmatrix}$$

$$\boldsymbol{K} = \begin{bmatrix} 1662 & 51 & 1071 \\ 0 & 1695 & 508 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{R} = \begin{bmatrix} -0.729 & 0.683 & -0.049 \\ -0.196 & -0.278 & -0.940 \\ -0.656 & -0.676 & 0.337 \end{bmatrix}$$

$$\boldsymbol{t} = \begin{bmatrix} -0.053 \\ 0.148 \\ 0.401 \end{bmatrix}$$

$$\boldsymbol{reprojection\_error} = 1.53px$$

If these results are compared with the obtained in section 2, it is obvious that both are almost the same, with a slight improvement achieved using the Gold Standard algorithm. In Figure 4, the reprojected points are shown overlapped almost perfectly to the clicked input points, as it happened with the DLT algorithm.

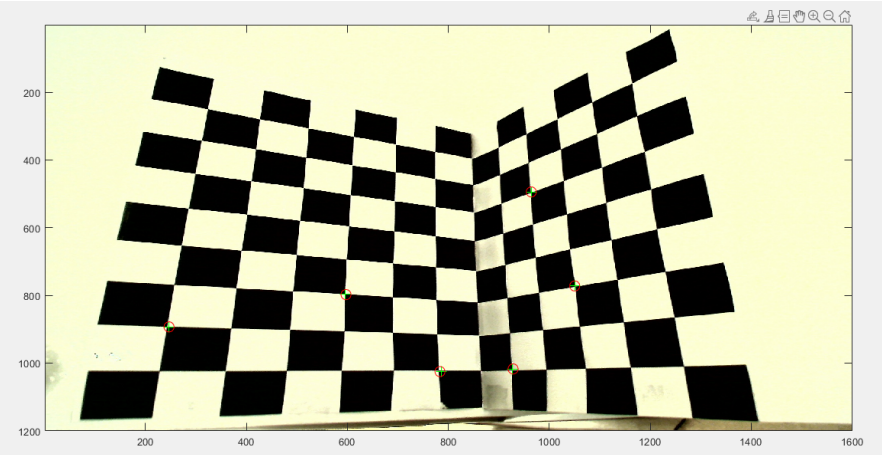
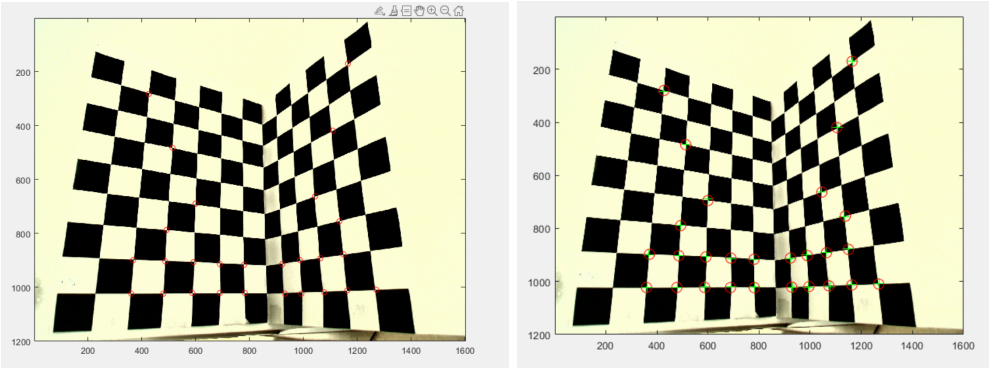


Figure 4: Reprojected points using Gold Standard algorithm

With these results, it doesn't become clear were would the advantage of the Gold Standard algorithm be. This is because for the computation of the algorithm only 6 correspondences have been used. This is the normal procedure for DLT, but with the Gold Standard algorithm more correspondences can be used in order to improve the accuracy of the calibration. Thus, considering the points shown in figure 5a, computing the Gold Standard algorithm the reprojections shown in figure 5b are obtained.



(a) Bigger set of initial points

(b) Reprojection with bigger set of points

Figure 5: Results after increasing number of correspondences

As it can be seen, the reprojection over the new set of correspondences also performs correctly. Furthermore, the obtained average reprojection errors are the following:

$$error_{DLT} = 2.78px$$

$$error_{GS} = 2.74px$$

In this case, the error has increased due to more points contributing to the sum of distances. However, the difference of errors between algorithms is slightly higher now, and would continue increasing as long as the number of correspondences increases as well.

In conclusion, there is no great difference between DLT and Gold Standard Algorithm, as the second one is just a extension of the first. Nevertheless, the optimization step taking place in Gold Standard allows to profit from extra information that could be obtained from extra correspondences using more advanced techniques, such as a Harris detector for example.

4 Conclusion

In this first lab assignment of the Computer Vision course, firstly a data normalization process has been applied to manually determined 2D-3D correspondences in a calibration object. Then, this normalization has been used in order to stabilize these data before using it to compute the camera calibration matrices. Two different approaches have been addressed in this regard: On the one hand, the DLT algorithm has been implemented, obtaining accurate results as shown in section 2. On the other hand, the Golden Standard Algorithm has been implemented as well, including the oprimization step that makes it differ from the DLT algorithm. Finally a discussion comparing both results has been presented, concluding that both algorithms provide sufficiently accurate results, but that, when the number of correspondences increases, the Gold Standard algorithm should perform better due to its optimization step that takes into account all correspondences available.