Exercise 2 - Camera Calibration

Computer Vision

Alberto Montes (malberto@student.ethz.ch) October 18, 2016

Data Normalization

The first step after capturing the points from the taken images is to transform the coordinates to homogeneous vectors.

```
if size(xy,2) ~= 3
    xy_ = padarray(xy, [1, 0], 1, 'post');
    XYZ_ = padarray(XYZ, [1, 0], 1, 'post');
end
```

Listing 1: Transform to Homogeneous Vectors

The next one has been implement the data normalization, which consist in substracting the centroid and scale all points to have mean distance to the centroid equivalent to $\sqrt{2}$ and $\sqrt{3}$ for 2D and 3D points respectively. Finding the centroid and the scale factor, it is only necessary to construct the transformation matrix as it is done in Listing 2.

```
function [xyn, XYZn, T, U] = normalization(xy, XYZ)
nb_points = size(xy, 2);

% Data normalization
% First compute centroid
xy_centroid = mean(xy(1:2,:), 2);
XYZ_centroid = mean(XYZ(1:3,:), 2);

% Then, compute scale
d = sum((xy(1:2,:)-xy_centroid*ones(1,nb_points)).*(xy(1:2,:)-
    xy_centroid*ones(1,nb_points)), 1);
sigma_2d = mean(sqrt(d)) / sqrt(2);
D = sum((XYZ(1:3,:)-XYZ_centroid*ones(1,nb_points)).*(XYZ(1:3,:)-
    XYZ_centroid*ones(1,nb_points)), 1);
sigma_3d = mean(sqrt(D)) / sqrt(3);

% Create T and U transformation matrices
T = inv([sigma_2d, 0, xy_centroid(1);
         0, sigma_2d, xy_centroid(2);
         0, 0, 1]);
U = inv([sigma_3d, 0, 0, XYZ_centroid(1);
         0, sigma_3d, 0, XYZ_centroid(2);
         0, 0, sigma_3d, XYZ_centroid(3);
         0, 0, 0, 1]);

% And normalize the points according to the transformations
xyn = T * xy;
XYZn = U * XYZ;

end
```

Listing 2: normalization.m

Direct Linear Transform

The DLT consist in different steps, starting with the previous presented one, data normalization. In Listing 3 there is the differents steps to go through to use the DLT method.

```
[xy_n, XYZ_n, T, U] = normalization(xy_, XYZ_);

% Compute DLT
[P_n] = dlt(xy_n, XYZ_n);

% Denormalize camera matrix
P = T \ P_n * U;

[K, R, t] = decompose(P);
```

Listing 3: runDLT.m

The next step consists into find the matrix transformation between the 3D and 2D points. To find this matrix is mandatory to use the normalized points to find the normalized matrix to then denormalize the camera matrix found. In Listing 4 is the code use to compute the camera matrix.

```
function [P] = dlt(xy, XYZ)
%computes DLT, xy and XYZ should be normalized before calling this
    function
A = zeros(2*size(xy, 2), 12);

for i = 1:size(xy,2)
    A(i*2-1:i*2,:) = [XYZ(:,i)', zeros(1,4), -XYZ(:,i)'*xy(1,i);
                      zeros(1,4), -XYZ(:,i)', XYZ(:,i)'*xy(2,i)];
end

% SVD
[~, ~, V] = svd(A);
P = reshape(V(:,end), 4, 3)';

end
```

Listing 4: dlt.m

Once you have computed this matrix, it can be decomposed into the intrinsic matrix, the camera orientation rotation matrix and the camera center. To do this is used the code in Listing 5.

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

% Factorize camera matrix in to K, R
[inv_R, inv_K] = qr(inv(P(1:3,1:3)));
K = inv(inv_K);
R = inv(inv_R);
% SVD of P to find the camara center C and then t
[~, ~, V] = svd(P);
C = V(:,end);
t = -R*C(1:3);

end
```

Listing 5: decompose.m

Finally, the reprojected points have been computed from the computed transformation matrix as well as the error of this reprojection. When computing the reprojected points it has to take into account that the points need to normalize such that the last component is 1 and fits the homogeneus coordinates.

```
% Compute reprojection error
xy_pp = P * XYZ_;
xy_p = xy_pp ./ (ones(3,1)*xy_pp(3,:));
er = xy_(1:2,:) - xy_p(1:2,:);
error = mean(sum(er.^2));
```

Listing 6: Computing reprojected points (runDLT.m)

The resulting mean error of the DLT method to the test image has been: 37.606. The reprojected points computed with the Direct Linear Transform can be found on Figure 1.

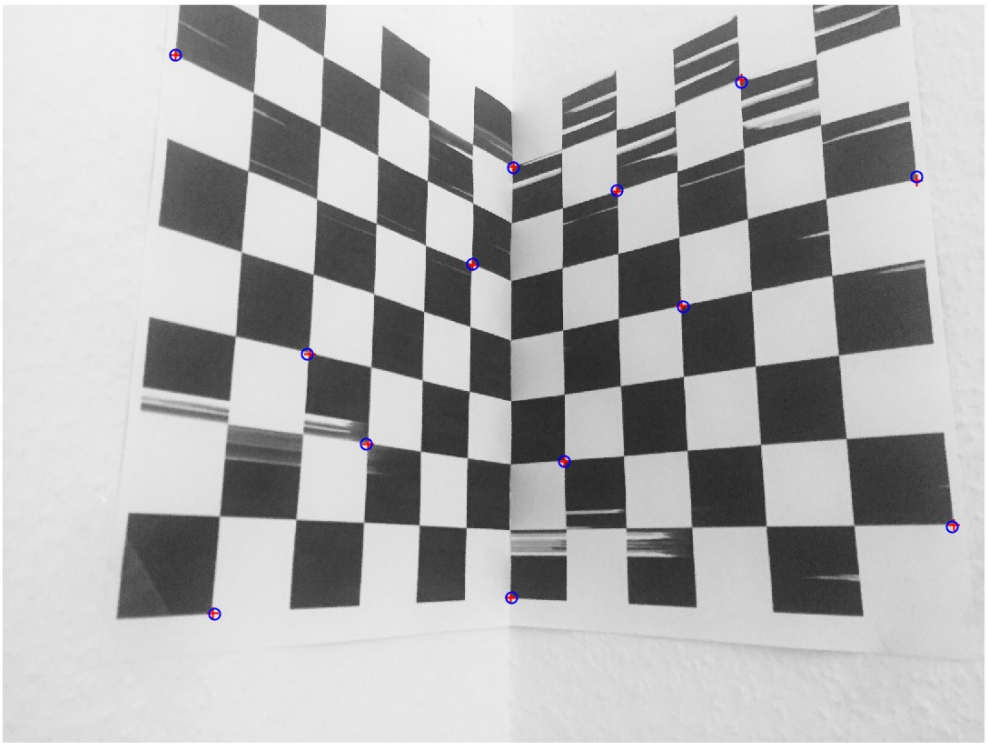


Figure 1: Reprojected points using the DLT method

Gold Standard algorithm

The Gold Standard algorithm works basically the same as the DLT method, but in addition, try to minimize the error caused by the camera matrix, trying to find its optimal values.

```
[xy_n, XYZ_n, T, U] = normalization(xy_, XYZ_);

%compute DLT
[P_n] = dlt(xy_n, XYZ_n);

%minimize geometric error
pn = [P_n(1,:) P_n(2,:) P_n(3,:)];
for i=1:20
    [pn] = fminsearch(@fminGoldStandard, pn, [], xy_n, XYZ_n, i/5);
end

% Denormalize camera matrix
P_n = [pn(1:4);pn(5:8);pn(9:12)];
P = T \ P_n * U;
```

```
15
16 % Factorize camera matrix in to K, R and t
17 [K, R, t] = decompose(P);
```

Listing 7: runGoldStandard.m

It basically starts with the same steps as the DLT, even computing the camera matrix with this method, and once it has an initial value for the matrix, tries to find the optimal values for the matrices that minimizes the sum squared error. To find the optimal values, it has been defined the function in Listing 8 which computes the sum of squared error of the reprojected points given a camera matrix, and iteratively is found the optimal values.

```
1  function f = fminGoldStandard(p, xy, XYZ, w)
2
3  %reassemble P
4  P = [p(1:4);p(5:8);p(9:12)];
5
6  %compute squared geometric error
7  xy_p = P*XYZ;
8  xy_p = xy_p ./ (ones(3,1)*xy_p(3,:));
9  dis = (xy - xy_p).^2;
10
11 %compute cost function value
12 f = sum(sum(dis));
13 end
```

Listing 8: fminGoldStandard.m

Using the search of the optimal values for the camera matrix which minimizes the error of the reprojected points, I have arrived into a decrease of the error to 36.344. The reprojected points are plot in Figure 2.

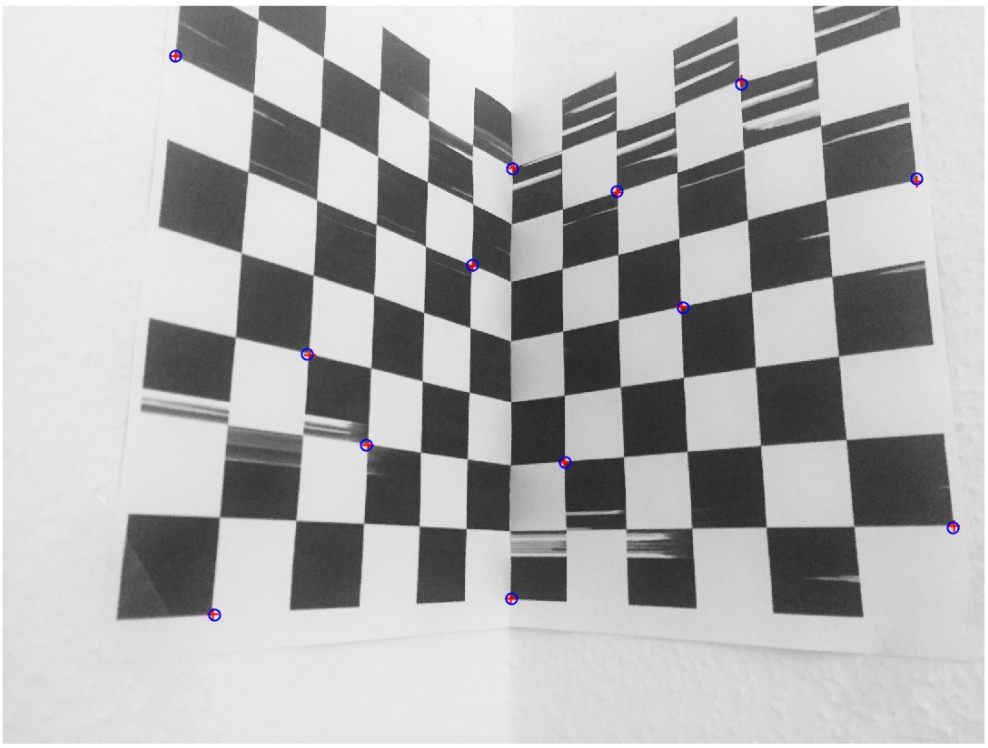
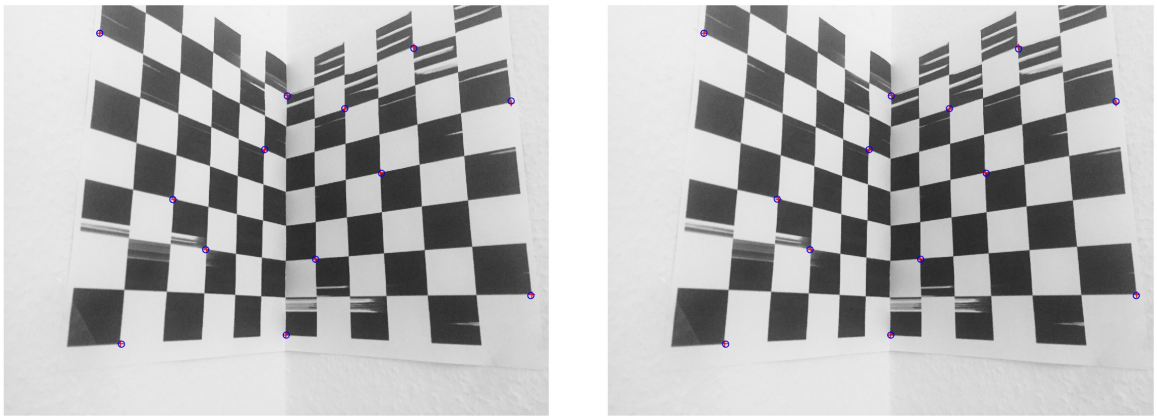


Figure 2: Reprojected points using the Gold Standard Algorithm

Comparison between the DLT method and the Gold Standard algorithm can be found in Figure 3. The results are very similar, no distinguishable at the image but the error shows that the Gold Standard algorithm works better than the Direct Linear Transform.



(a) Direct Linear Transform (b) Gold Standard Algorithm

Figure 3: Comparing results

Bouget's Calibration Toolbox

It has been followed the tutorial given at the statement's reference to calibrate a camera from some taken images. The following Figures and Listings contains captures of each step as well as the calibration results. At the end there is the extrinsic representation of the camera and the images taken.

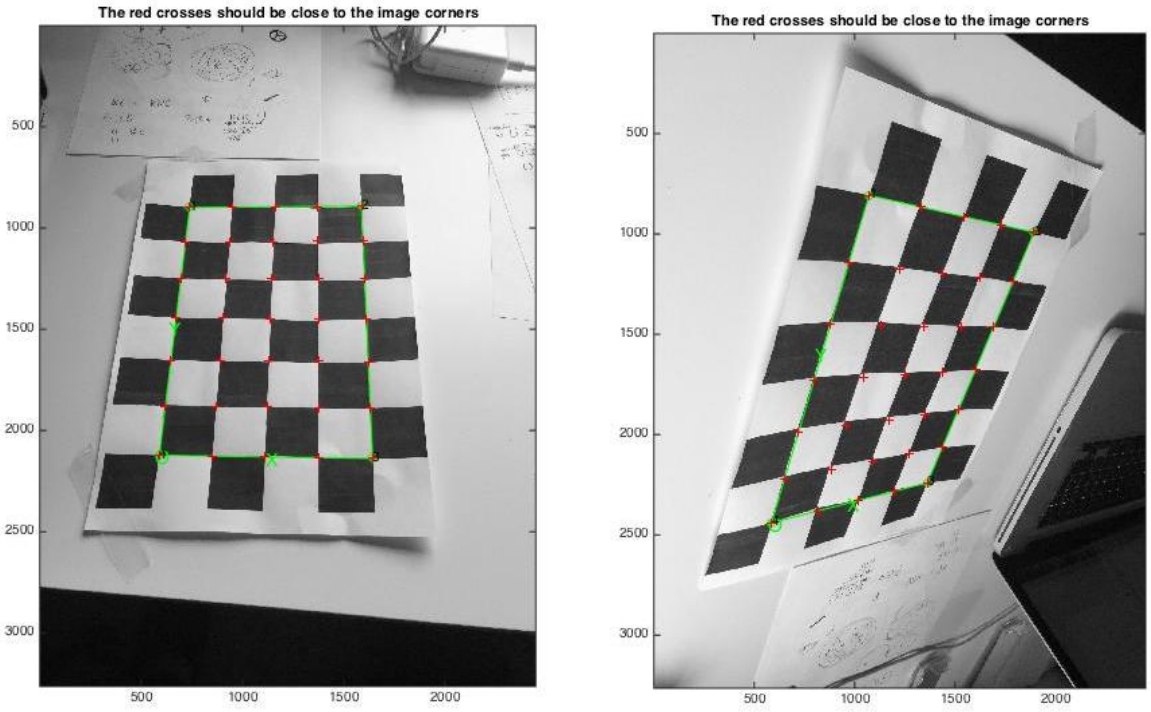


Figure 4: Corner Detection

```
%-- Focal length:
fc = [ 2878.381810891617079 ; 2870.281574007985910 ];
```

```
3
4  %-- Principal point:
5  cc = [ 1259.339783849922924 ; 1472.437156191800568 ];
6
7  %-- Skew coefficient:
8  alpha_c = 0.000000000000000;
9
10 %-- Distortion coefficients:
11 kc = [ 0.157771338872593 ; -0.274850780438951 ; -0.027134856721037 ;
       0.007190782977225 ; 0.000000000000000 ];
12
13 %-- Focal length uncertainty:
14 fc_error = [ 46.583884021056349 ; 50.852465978430288 ];
15
16 %-- Principal point uncertainty:
17 cc_error = [ 62.158217016359664 ; 71.234874691483881 ];
18
19 %-- Skew coefficient uncertainty:
20 alpha_c_error = 0.000000000000000;
21
22 %-- Distortion coefficients uncertainty:
23 kc_error = [ 0.089103452718634 ; 0.535128637433302 ; 0.011016076688067 ;
        0.009489405361836 ; 0.000000000000000 ];
24
25 %-- Image size:
26 nx = 2448;
27 ny = 3264;
28
29
30 %-- Various other variables (may be ignored if you do not use the Matlab
       Calibration Toolbox):
31 %-- Those variables are used to control which intrinsic parameters
       should be optimized
32
33 n_ima = 9;                      % Number of calibration images
34 est_fc = [ 1 ; 1 ];                % Estimation indicator of the two
       focal variables
35 est_aspect_ratio = 1;             % Estimation indicator of the aspect
        ratio fc(2)/fc(1)
36 center_optim = 1;                 % Estimation indicator of the
       principal point
37 est_alpha = 0;                   % Estimation indicator of the skew
       coefficient
38 est_dist = [ 1 ; 1 ; 1 ; 1 ; 0 ];   % Estimation indicator of the
       distortion coefficients
```

Listing 9: Calibration Results

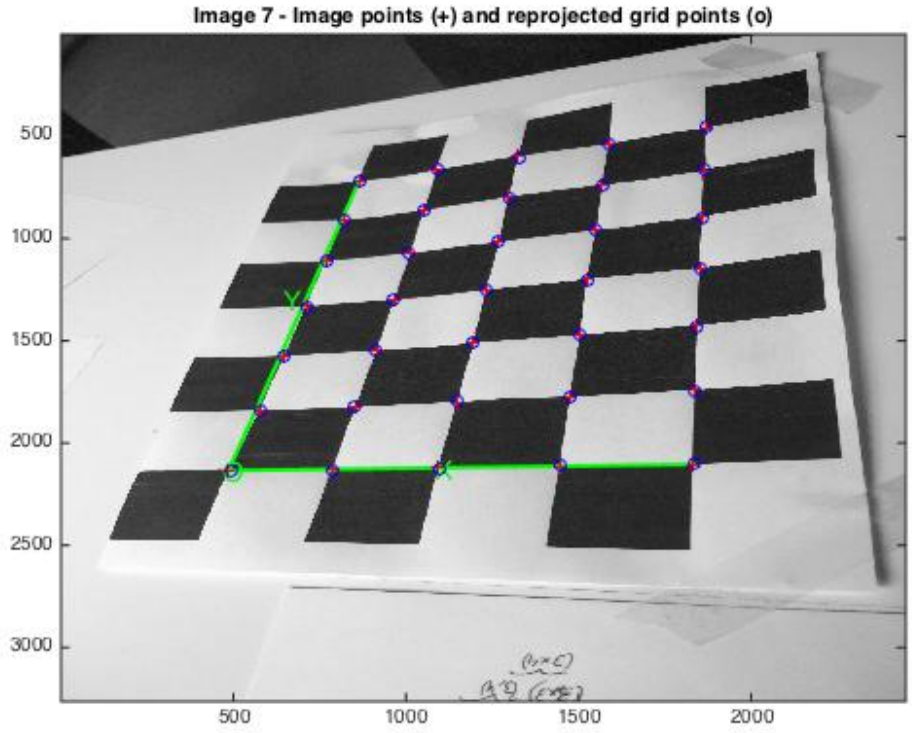


Figure 5: Reprojected points after calibration

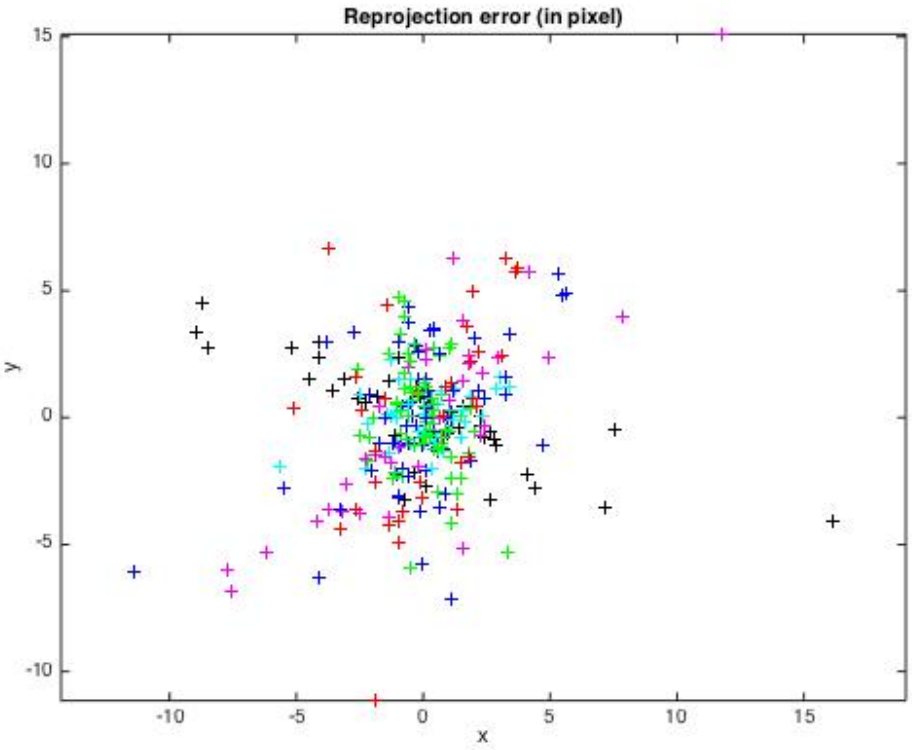


Figure 6: Reprojected error

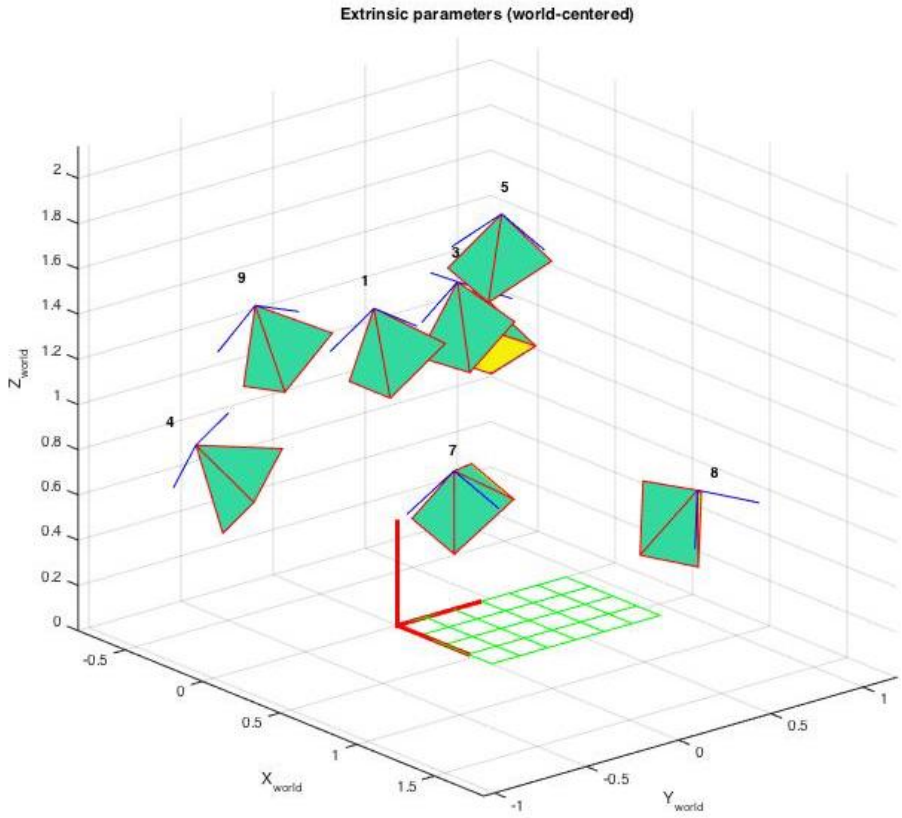


Figure 7: Extrinsic parameters in world centered view