Computer Vision
Exercise 2

Xingze Tian (xt)

October 12, 2017

1 Select points

In this exercise, I selected 6 points on the two planes as `xy` and corresponding `XYZ` and stored the results in `points.mat`.

2 Data Normalization

The data normalization process is implemented in `normalization.m` with the following steps:

a) **Construct homogeneous coordinates of the input xy and XYZ**
This is simply done by adding another row of ones to the `xy` and `XYZ` matrices, resulting 2 homogeneous coordinates `xy_homo` and `XYZ_homo`, which would be used later.

b) **Compute the centroids of the points**
The center of mass would be the mean of the x coordinates and the y coordinates of the input points. This can be simply done using the matlab function `mean()`, and the centroid (C_x, C_y) of `xy` and (C_x, C_y, C_z) of `XYZ` are computed. To translate the centroids to the origin, the transformation matrix A(3x3) and B(4x4) can do the trick:

$$A = \begin{bmatrix} 1 & 0 & -C_x \\ 0 & 1 & -C_y \\ 0 & 0 & 1 \end{bmatrix} \quad B = \begin{bmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

c) **Compute the scaling factor**
The scaling factor is computed first by calculating the average Euclidean distance between the centroid and the points which can be simply done using `mean(sqrt(sum(xy-centroid).^2))`. Then use$\sqrt{2}$ to$\sqrt{3}$ to divide the mean distances to get the scaling factors s_{2D} and s_{3D}. The scaling matrices are:

$$S_1 = \begin{bmatrix} s_{2D} & 0 & 0 \\ 0 & s_{2D} & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad S_2 = \begin{bmatrix} s_{3D} & 0 & 0 & 0 \\ 0 & s_{3D} & 0 & 0 \\ 0 & 0 & s_{3D} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

d) **Construct transformation matrices**
As two transformations would need to be performed (translation and scaling), the resulting transformation matrices can thus be simply calculated by $\mathbf{T} = \mathbf{S_1 A}$ and $\mathbf{U} = \mathbf{S_2 B}$:

$$T = \begin{bmatrix} s_{2D} & 0 & -C_x/s_{2D} \\ 0 & s_{2D} & -C_y/s_{2D} \\ 0 & 0 & 1 \end{bmatrix} \quad U = \begin{bmatrix} s_{3D} & 0 & 0 & -C_x/s_{3D} \\ 0 & s_{3D} & 0 & -C_y/s_{3D} \\ 0 & 0 & s_{3D} & -C_z/s_{3D} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Then by applying the transformation matrices $\mathbf{T}$ and $\mathbf{U}$ to `xy` and `XYZ` the normalized points are thus:

$$xyn = \mathbf{T}xy, \quad XYZn = \mathbf{U}XYZ$$

3 Direct Linear Transform

3.1 Algorithm Implementation

The DLT algorithm is implemented in `runDLT.m` with the following steps:

a) **Compute the normalized camera matrix $\hat{\mathbf{P}}$**
The DLT algorithm is implemented in `dlt.m` which returns a normalized camera matrix $\hat{\mathbf{P}}$. This is done by finding a matrix `A` such that $\mathtt{A}\hat{\mathbf{P}} = 0$ where `A` is in the form:

$$A = \begin{bmatrix} X_{1x} & X_{1y} & X_{1z} & 1 & 0 & 0 & 0 & 0 & -x_1X_{1x} & -x_1X_{1y} & -x_1X_{1z} & -x_1 \\ 0 & 0 & 0 & 0 & X_{1x} & X_{1y} & X_{1z} & -1 & -y_1X_{1x} & -y_1X_{1y} & -y_1X_{1z} & y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{ix} & X_{iy} & X_{iz} & 1 & 0 & 0 & 0 & 0 & -x_iX_{ix} & -x_iX_{iy} & -x_iX_{iz} & -x_i \\ 0 & 0 & 0 & 0 & X_{ix} & X_{iy} & X_{iz} & -1 & -y_iX_{ix} & -y_iX_{iy} & -y_iX_{iz} & y_i \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ X_{6x} & X_{6y} & X_{6z} & 1 & 0 & 0 & 0 & 0 & -x_6X_{6x} & -x_6X_{6y} & -x_6X_{6z} & -x_6 \\ 0 & 0 & 0 & 0 & X_{6x} & X_{6y} & X_{6z} & -1 & -y_6X_{6x} & -y_6X_{6y} & -y_6X_{6z} & y_6 \end{bmatrix}$$

Then I applied singular value decomposition (SVD) of `A` using the matlab function `svd()` and the returned right null-vector is the desired $\hat{P}$, which is then reshaped into a 3x4 matrix.

b) **Denormalize $\hat{\mathbf{P}}$**
Use $\mathbf{P} = \mathbf{T}^{-1}\hat{\mathbf{P}}\mathbf{U}$ to get the matrix without normalization.

c) **Factorize camera matrix in to K, R and t**
The factorization process takes place in `decompose.m` with

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|t] = [\mathbf{KR}| - \mathbf{KRC}]$$

QR decomposition is applied on the inverse of the left 3x3 part of $\mathbf{P}$ using the matlab function `qr()`. The result is then $\mathbf{R^{-1}}$ and $\mathbf{K^{-1}}$, then $\mathbf{R}$ and $\mathbf{K}$ can be retrieved by inverting the result.

The value t is calculated using $t = -\mathbf{KRC}$ and $\mathbf{C}$ is computed by solving $\mathtt{P}\mathbf{C} = \mathtt{0}$, which is done by performing singular value decomposition on $\mathbf{P}$ and the result is the right null-vector. The calculated $\mathbf{C}$ is then converted into inhomogeneous form before it is used to compute t.

d) **Compute reprojection error**
The reprojection error is computed by summing the Euclidean difference between the projection points and the reprojection points(calculated as `xy_reprojected =` $\mathbf{P}$`XYZ`).

3.2 Evaluation

3.2.1 Reprojection Results

The reprojection result of the selected 6 points are shown in Figure 1 as the blue asterisks, and the points marked with red circle are the reprojected points. The error computed using the 6 points is 1.2439. The resulting projection matrix is:

$$P = \begin{bmatrix} -82.4438 & 3.2002 & 14.2453 & 606.8606 \\ -27.9833 & -44.7293 & -56.6388 & 787.0141 \\ -0.0277 & -0.0432 & 0.0188 & 0.6979 \end{bmatrix} \quad K = \begin{bmatrix} 71.1403 & 0.4613 & 44.1474 \\ 0 & 71.3589 & 29.9940 \\ 0 & 0 & 0.0546 \end{bmatrix}$$

$$R = \begin{bmatrix} -0.8434 & 0.5373 & -0.0076 \\ -0.1792 & -0.2947 & -0.9386 \\ -0.5066 & -0.7902 & 0.3448 \end{bmatrix} \quad t = \begin{bmatrix} 606.8606 \\ 787.0141 \\ 0.6979 \end{bmatrix}$$

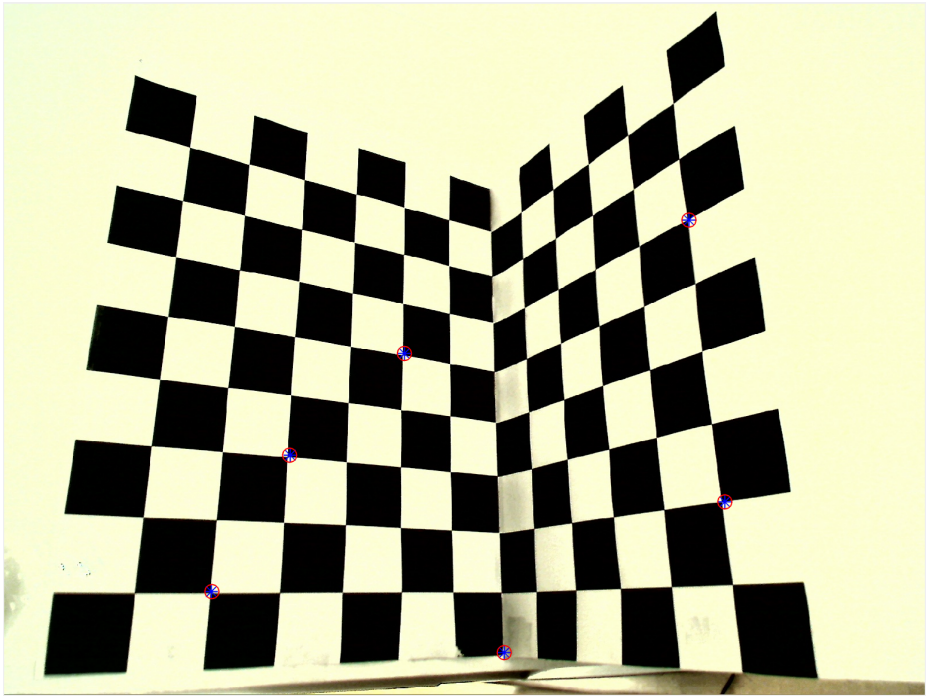

Figure 1: Reprojection result,the blue asterisks are the pre-picked points, and the red circle is the reprojection result

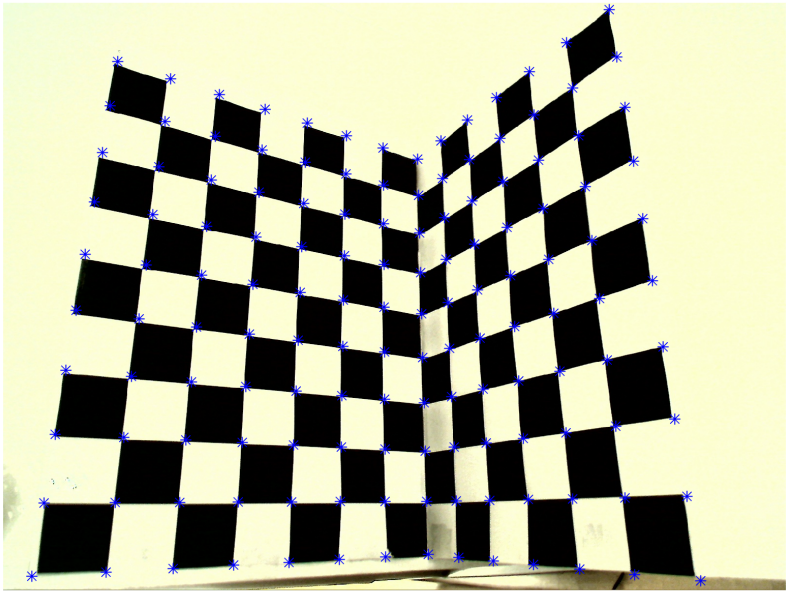

Figure 2: Reprojection result, the blue asterisk are the reprojected corners

3.2.2 Comparison with using unnormalized data

If unnormalized points are used, generally the points are noisier than normalized ones, and the reprojection results would be more erroneous and unstable. Using the 6 points chosen in this exercise, the unnormalized data produced the error as 1.2424 which is slightly smaller the normalized ones. However, with a different points selection, the unnormalized error may be slightly bigger. I experimented with other points settings, the normalized results give slightly more stable results than the unnormalized values.

$$P = \begin{bmatrix} -0.0824 & 0.0032 & 0.0142 & 0.6067 \\ -0.0280 & -0.0447 & -0.0566 & 0.7867 \\ -0.0000 & -0.0000 & 0.0000 & 0.0007 \end{bmatrix} \quad K = \begin{bmatrix} 0.0711 & 0.0005 & 0.0441 \\ 0 & 0.0713 & 0.0300 \\ 0 & 0 & 0.0001 \end{bmatrix}$$

$$R = \begin{bmatrix} -0.8433 & 0.5374 & -0.0076 \\ -0.1792 & -0.2946 & -0.9387 \\ -0.5067 & -0.7902 & 0.3447 \end{bmatrix} \quad t = \begin{bmatrix} 0.6067 \\ 0.7867 \\ 0.0007 \end{bmatrix}$$

4 Gold Standard algorithm

4.1 Algorithm Implementation

The Gold Standard Algorithm is implemented in `runGoldStandard.m` with the following steps:

a) **Data normalization and use DLT to compute $\hat{\mathbf{P}}$**
Same as section 2 and 3.1a).

b) **Minimize geometric error**
The geometric error is computed in `fminGoldStandard.m` using the same method as calculating the error in section 3.1d). First the input `p` is reshaped to 3x4 matrix $\mathbf{P}$ and then the reprojected points are calculated the same as in section 3.1d). Matlab function `fminsearch()` is then used to find the minimize the geometric error.

c) **Denormalize and factorize the camera matrix**
Same as section 3.1b) and 3.1c).

4.2 Evaluation

The reprojected image using the Gold Search Algorithm are as shown in Figure 3 and Figure 4.

The reprojection error is now 0.6036, which is much smaller than the DLT algorithm(1.2439). The computed camera matrix is as below:

$$P = \begin{bmatrix} -82.4894 & 3.1808 & 14.2631 & 607.0422 \\ -28.0256 & -44.7553 & -56.6474 & 787.2619 \\ -0.0277 & -0.0432 & 0.0188 & 0.6981 \end{bmatrix} \quad K = \begin{bmatrix} 71.1502 & 0.4623 & 44.2210 \\ 0 & 71.3784 & 30.0420 \\ 0 & 0 & 0.0547 \end{bmatrix}$$

$$R = \begin{bmatrix} -0.8431 & 0.5377 & -0.0077 \\ -0.1793 & -0.2945 & -0.9387 \\ -0.5069 & -0.7901 & 0.3447 \end{bmatrix} \quad t = \begin{bmatrix} 607.0422 \\ 787.2619 \\ 0.6981 \end{bmatrix}$$

I also tried with other point selections, and the Gold Search Algorithm generally gives less error than DLT algorithm.

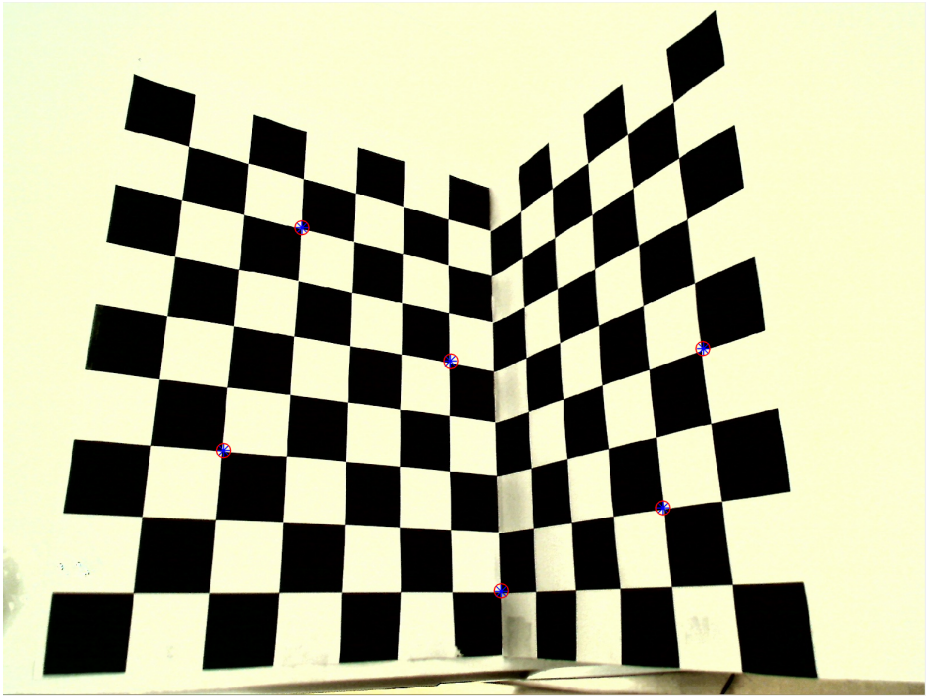

Figure 3: Reprojection result,the blue asterisks are the pre-picked points, and the red circle is the reprojection result

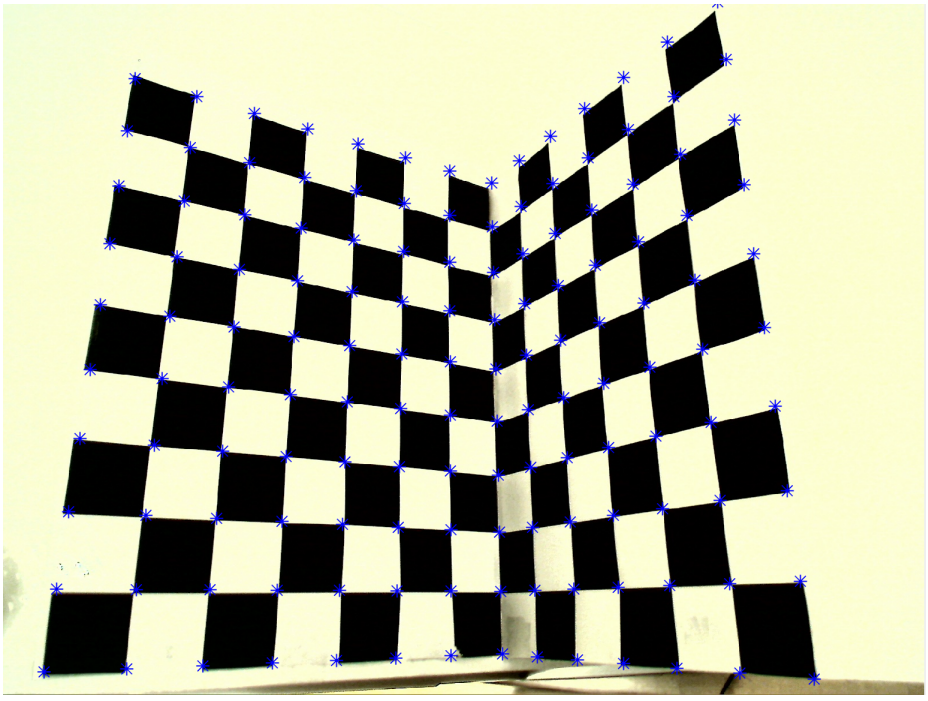

Figure 4: Reprojection result, the blue asterisk are the reprojected corners

5 Bouget's Calibration Toolbox

I used 6 images of the same object from different view points to calibrate the camera as shown in Figure 5: Following the tutorial, I got the following camera intrinsics after the optimization process:

Focal Length: f_c = [1680.01967 1680.92114] +/- [36.59704 40.88545]

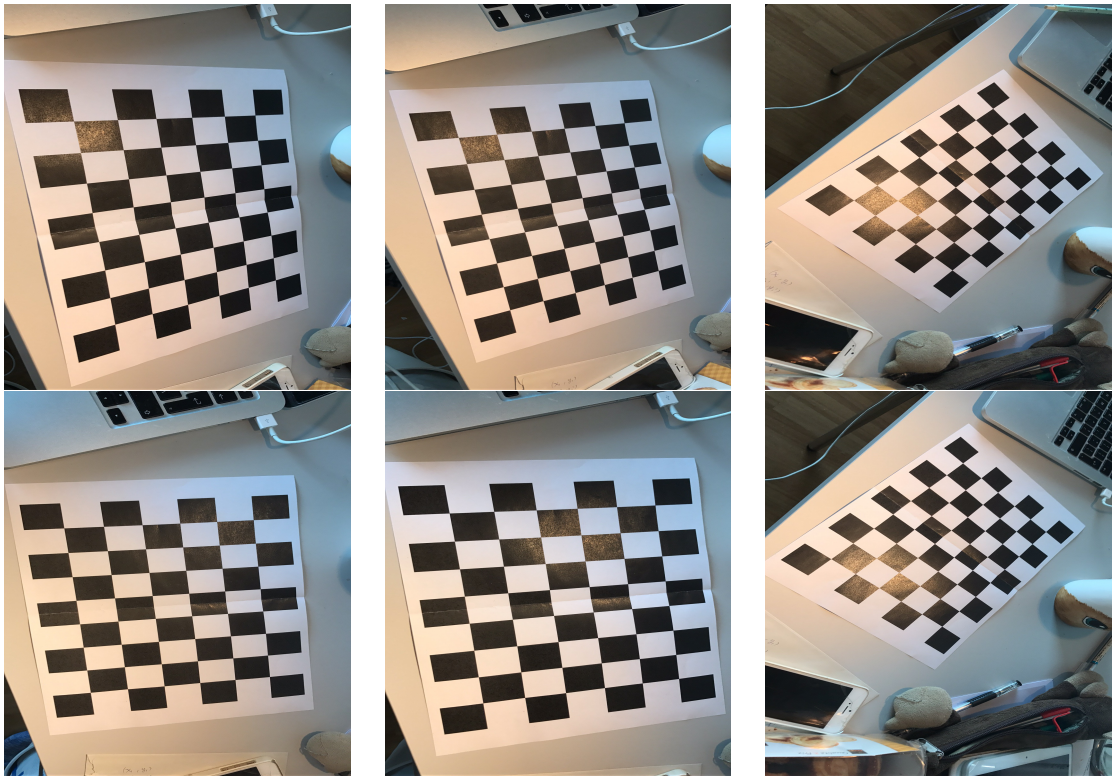

Figure 5: Reprojection result, the blue asterisk are the reprojected corners

Principal point: cc = [505.40793 936.67663] +/- [29.32053 12.49912]

Skew: alpha_c = [0.00000] +/- [0.00000] => angle of pixel axes = 90.00000 +/- 0.00000 degrees

Distortion: kc = [0.09234 -0.35994 -0.00431 -0.00953 0.00000] +/- [0.02579 0.10743 0.00305 0.00238 0.00000]

Pixel error: err = [1.27156 0.89956]

The error is visualized in Figure 6:

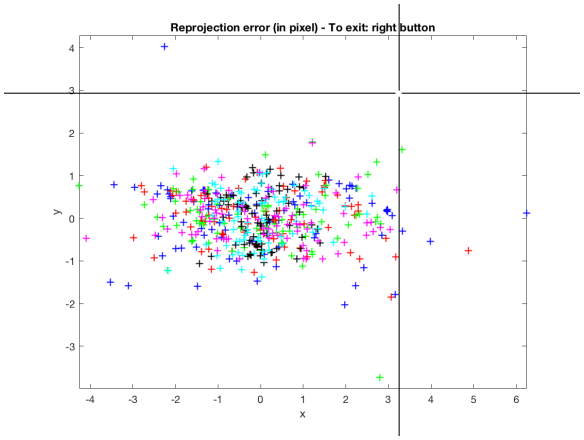

Figure 6: Camera extrinsic parameters in local and world coordinates

The extrinsic parameters can be seen in Figure 7:

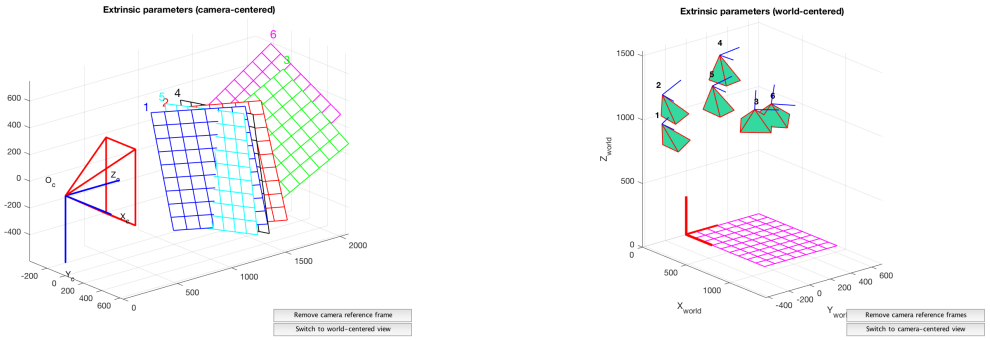

Figure 7: Camera extrinsic parameters in local and world coordinates

I used the same camera to take pictures of the same pattern, and extracted corners using my DLT method, the results are as shown in Figure 8:

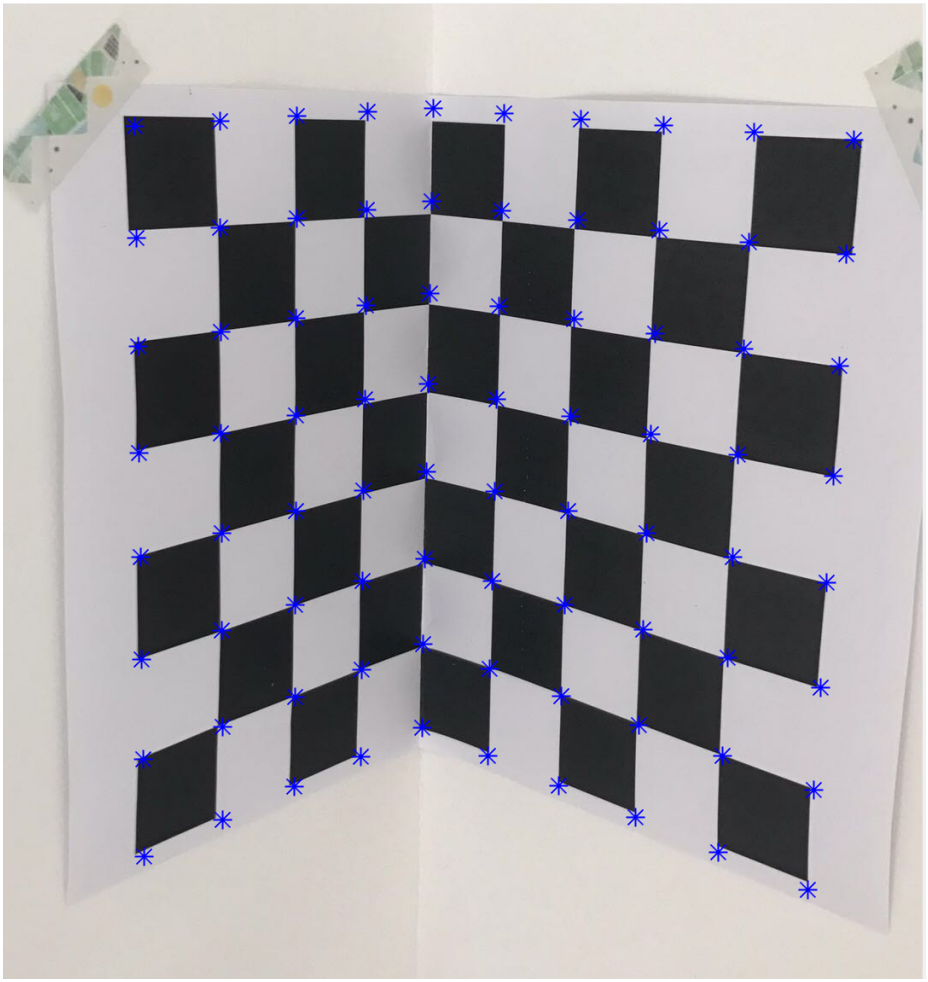

Figure 8: Using my DLT to reproject

The camera matrix is:

$$P = \begin{bmatrix} -60.2063 & 39.1031 & -1.8572 & 338.4108 \\ -0.3570 & 0.8533 & -71.9105 & 582.2608 \\ -0.0262 & -0.0243 & -0.0062 & 0.6856 \end{bmatrix} \quad K = \begin{bmatrix} 69.6109 & 0.3078 & -17.6500 \\ 0 & -70.9231 & -11.9117 \\ 0 & 0 & -0.0362 \end{bmatrix}$$

$$R = \begin{bmatrix} -0.6811 & 0.7321 & 0.0121 \\ -0.1164 & -0.1245 & 0.9854 \\ 0.7229 & 0.6697 & 0.1700 \end{bmatrix} \quad t = \begin{bmatrix} 338.4108 \\ 582.2608 \\ 0.6856 \end{bmatrix}$$

By normalizing the right bottom value of **K** to 1, we can get a normalized K as:

$$K = \begin{bmatrix} -1922.953 & -8.503 & 487.569 \\ 0 & 1959.20 & 329.052 \\ 0 & 0 & 1 \end{bmatrix}$$

In this case the there is a negative number in K's diagonal, which can be easily removed by converting the right-handed coordinate system to left-handed coordinate system. The selection of points also affects the K's value.

The focal length calculated using the toolbox is about 200 less than my calculation and the reprojection error 1.27 while mine is 3.43 using DLT and 1.62 using Gold Standard Algorithm. The toolbox computes a better calibration result than mine.