ELL793 - Computer Vision
Assignment 1

Varun Gupta 2017EE30551
Sarthak Garg 2017EE30546

October 30, 2020

1 Taking input points for calibration

First we choose 9 (>=6) point and input the space coordinate (X,Y,Z) of these points. Then, to get the image coordinates of these points (x,y) we used OpenCV library. The image of the chessboard is displayed and the user has to click on the image to select the points. Fig 1 shows the input image clicked by our smartphone camera. The green points marked in the image are the 9 selected points that will act as the fiduciary points for our camera calibration. The 2D and corresponding 3D coordinates of these points are shown in Table 1. The code for getting the image coordinates of the selected points using OpenCV is done in the very beginning of the attached Python notebook.

2 Data Normalisation

The normalisation process hs been done in the function ***normalise()*** in the attached Python notebook.

2.1 Moving centroids to the origin

From the input points, we perform the task of computing the centroid of each dimension of the xy and XYZ points.

$$C_{xy} = \begin{bmatrix} c_x & c_y \end{bmatrix}^T \qquad\qquad C_{XYZ} = \begin{bmatrix} c_X & c_Y & c_Z \end{bmatrix}^T$$

where the individual c_i's are the centroids of the corresponding dimension of all the N points.

X	Y	Z	x	y
3.8	0	0	724	847
7.6	0	7.6	847	923
11.4	0	15.2	997	553
7.6	0	22.8	850	242
0	7.6	3.8	366	1091
0	11.4	7.6	216	921
0	11.4	15.2	215	556
0	15.2	22.8	31	118
0	3.8	22.8	500	302

Table 1: Coordinates of input points

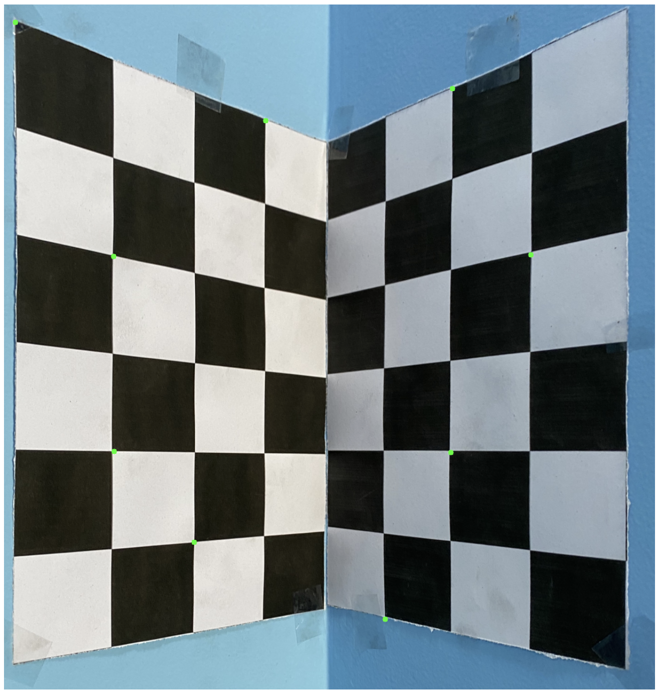

Figure 1: Input image with the selected points marked in green

After computing these centroids, we can deduce the transformations matrix T1 and U1 such that when doing $T_1 \cdot xy$ and $U_1 \cdot XYZ$, we move the centroids to the origin:

$$T_1 = \begin{bmatrix} 1 & 0 & -c_x \\ 0 & 1 & -c_y \\ 0 & 0 & 1 \end{bmatrix} \qquad U_1 = \begin{bmatrix} 1 & 0 & 0 & -c_X \\ 0 & 1 & 0 & -c_Y \\ 0 & 0 & 1 & -c_Z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.2 Normalising Euclidian distance from origin

The second part of the normalization is transforming the input points such that the average Euclidian distance from the origin is $\sqrt{2}$ 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 dist_xy and dist_XYZ. The second transformation matrices T2 and U2 are thus simply:

$$T_2 = \sqrt{2}/dist\_xy \qquad U_2 = \sqrt{3}/dist\_XYZ$$

2.3 Final transformation matrices T and U

The final matrices such that the normalized points are $\hat{x}_i = T \cdot x$ and $\hat{X}_i = U \cdot X_i$ are therefore:

$$T = T_1 \cdot T_2 \qquad U = U_1 \cdot U_2$$

For our selected 9 points, we obtained the following transformation matrices:

$$T = \begin{bmatrix} 0.0029 & 0 & -1.553 \\ 0 & 0.0029 & -1.942 \\ 0 & 0 & 1 \end{bmatrix} \qquad U = \begin{bmatrix} 0.163 & 0 & 0 & -0.551 \\ 0 & 0.163 & 0 & -0.896 \\ 0 & 0 & 0.163 & -2.137 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3 Obtaining Projection Matrix through Direct Linear Transformation (DLT) method

Next, we use the DLT algorithm to estimate the normalised projection matrix m_{norm}. As taught in class and explained in the book, we can form an equation as follows:

$$\mathscr{P} m_{norm} = 0$$

$$\mathscr{P} = \begin{bmatrix} P_1^T & 0^T & -x_1 P_1^T \\ 0^T & P_1^T & -y_1 P_1^T \\ ... & ... & ... \\ P_n^T & 0^T & -x_n P_n^T \\ 0^T & P_n^T & -y_n P_n^T \end{bmatrix} \qquad m_{norm} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix}$$

where P_i are the columns of the homogeneous world coordinates matrix XYZ. Therefore, we have 2 rows in $\mathscr{P}$ for each fiduciary point taken as input by us. Dimension of $\mathscr{P}$ is 2n rows and 12 columns, where n is the number of input points.

According to theory taught in class, the way to find matrix *m* is to find the eigenvector corresponding to the lowest possible eigenvalue of the matrix $\mathscr{P}^T \mathscr{P}$. As suggested in the book, we perform **Singular Value Decomposition (SVD)** on matrix $\mathscr{P}$ to obtain the matrix *m*. To perform SVD, we use the inbuilt function in numpy - *np.linalg.svd()*. This function returns the three vectors U,S and V. The last column of V gives us 12 values which form the 12 elements of the matrix *m*.

In the attached code file, the normalised projection matrix is computed in the function ***DLT()***. After applying the above mentioned SVD and computation to our set of input points, we obtain this **normalised projection matrix:**

$$m_{norm} = \begin{bmatrix} 3.42506351e-01 & -4.06885167e-01 & 1.26126885e-04 & 2.01644139e-02 \\ -7.53079615e-02 & -7.16589609e-02 & -5.27968265e-01 & 2.33704139e-02 \\ -8.71540616e-02 & -8.32371537e-02 & -1.46803834e-03 & 6.41948496e-01 \end{bmatrix}$$

Finally, to obtain the actual projection matrix, we need to **denormalise the normalised projection matrix** calculated above. To do this, we use the following transformation:

$$m = T^{-1} \cdot m_{norm} \cdot U$$

where T and U are the normalisation transformation matrices computed above.
Applying this transformation in the function **i**n the Python, we get:

$$m = \begin{bmatrix} 1.14838224e+01 & -2.97271291e+01 & -1.19433627e-01 & 4.71308849e+02 \\ -1.35707764e+01 & -1.29462101e+01 & -2.94302341e+01 & 9.33799832e+02 \\ -1.42332119e-02 & -1.35935379e-02 & -2.39746724e-04 & 7.67776560e-01 \end{bmatrix}$$

Thus, we have successfully obtained the projection matrix for our phone's camera using our set of 9 input fiduciary points.

4 Obtaining Intrinsic Parameters from Projection Matrix

There are 5 intrinsic parameter $X_0, Y_0, \theta, \alpha, \beta$. To obtain these parameter use the projection matrix m. We use the following formulae to calculate the parameter:

$$m = (A \;\; b) \qquad A = \begin{pmatrix} a_1^T \\ a_2^T \\ a_3^T \end{pmatrix} \qquad \rho = 1/||a_3||$$

$$x_0 = \rho^2(a_1 \cdot a_3) \qquad y_0 = \rho^2(a_2 \cdot a_3)$$

$$cos\theta = -\frac{(a1 \times a3) \cdot (a2 \times a3)}{||a1 \times a3||||a2 \times a3||} \qquad \alpha = \rho^2 ||a1 \times a3|| sin\theta \qquad \beta = \rho^2 ||a2 \times a3|| sin\theta$$

Using the above formulae we got the following values:

$$\rho = 50.8049 \qquad \alpha = 1384.9391 \qquad \beta = 1374.1285 \qquad \theta = 89.9217^\circ$$

Camera's Optic Center coordinates:

$$x_0 = 621.2122 \qquad y_0 = 971.0145$$

Skew Angle $= 90^\circ - \theta = 0.0783^\circ$

From the aspect ratio of the image from the EXIF tag attached to the image, we get the value of pixel width (k) = $3.068 * 10^{-6}$ m. Since $\alpha = kf$, we get the value of **focal length**(f) = 4.25 mm.

Next we obtain the calibration matrix K from these parameters as:

$$K = \begin{pmatrix} \alpha & -\alpha cot\theta & x_0 \\ 0 & \frac{\beta}{sin\theta} & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$K = \begin{pmatrix} 1.38493912e+03 & 5.63407958e+02 & 6.21212230e+02 \\ 0.00000000e+00 & 1.48348292e+03 & 9.71014507e+02 \\ 0.00000000e+00 & 0.00000000e+00 & 1.00000000e+00 \end{pmatrix}$$

There are two extrinsic parameter matrices to be calculated R(contains rotation parameters) and t (contains the translation parameters)

$$r_3 = \rho a_3 \qquad r_1 = \frac{\rho^2 sin\theta}{\beta}(a2 \times a3) \qquad r_2 = r_3 \times r_1$$

$$R = \begin{pmatrix} -6.90672689e-01 & 7.23167411e-01 & 3.63141056e-04 \\ 8.55760972e-03 & 8.67519928e-03 & -9.99925751e-01 \\ -7.23116868e-01 & -6.90618300e-01 & -1.21803077e-02 \end{pmatrix}$$

$$t = \begin{pmatrix} -2.83011492 \\ 6.44795219 \\ 39.00680933 \end{pmatrix}$$

From the rotation matrix, we obtain the Euler rotation angles (3 out of the 6 extrinsic parameters; other 3 are the translation parameters):

$$\theta_x = 90.69^\circ \qquad \theta_y = 0.0201^\circ \qquad \theta_z = -133.682^\circ$$

5 Flowchart

Fig 2 shows the flowchart of our entire camera calibration process that we have performed in the assignment. It summarizes all the steps explained the above sections. The corresponding code functions for the various operations have been mentioned in the above sections and have also been explained in the README file.

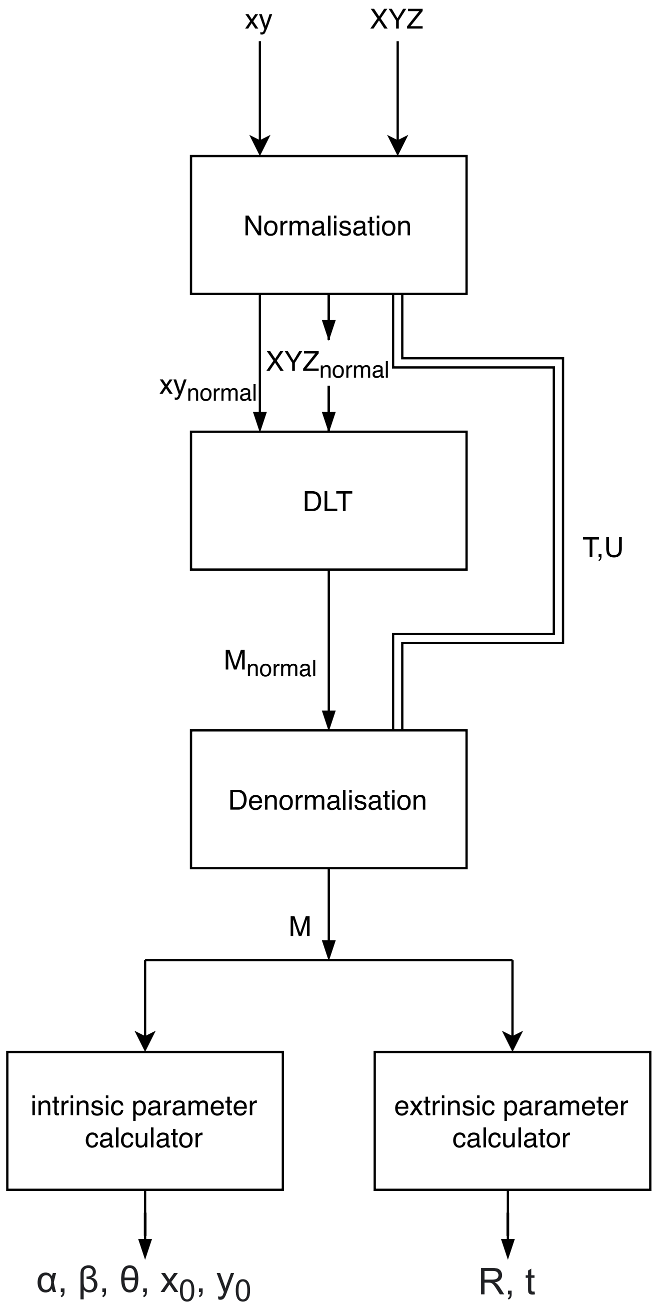

Figure 2: Flowchart of camera calibration process

6 Results and Verification

As explained above, we have successfully obtained the denormalised projection matrix and the intrinsic parameters of the smartphone camera. Next, we verify that the projection matrix is correctly estimated.

6.1 Reprojecting points using computed projection matrix

The reprojected point image coordinates are obtained by multiplying the computed projection matrix with the space coordinates matrix XYZ. Fig. 3 shows the reprojected points on the original image, along with the original selected points. We can see clearly that the reprojected points almost exactly overlap the originally selected points (in this image scale).

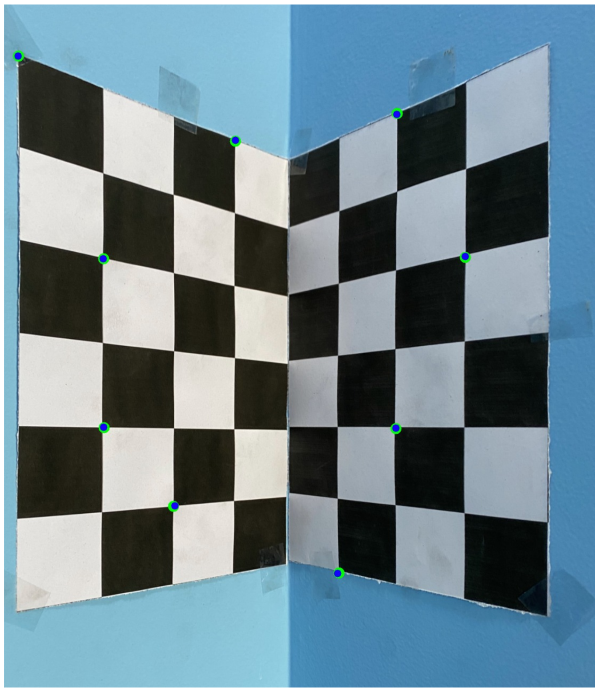

Figure 3: Reprojected points (blue) and the original selected points (green)

As a fun exercise, we used the computed projection matrix and tried to project all the corners of the grids by passing in the real world coordinates of all the corners. The result is shown in Fig 4.

6.2 Root Mean Squared Loss

To quantify our results, we calculate the RMSE loss between our projected points' coordinates and originally selected points' coordinates. We simply use ***(np.square(np.subtract(A, B)).mean())**(1/2)*** for this purpose. On doing this, we obtain:

RMSE Loss = .18499

As we can see, the RMSE loss is really low and we have achieved a good performance on the set of 9 points chosen by us. The performance might be even better if we choose more than 9 points.

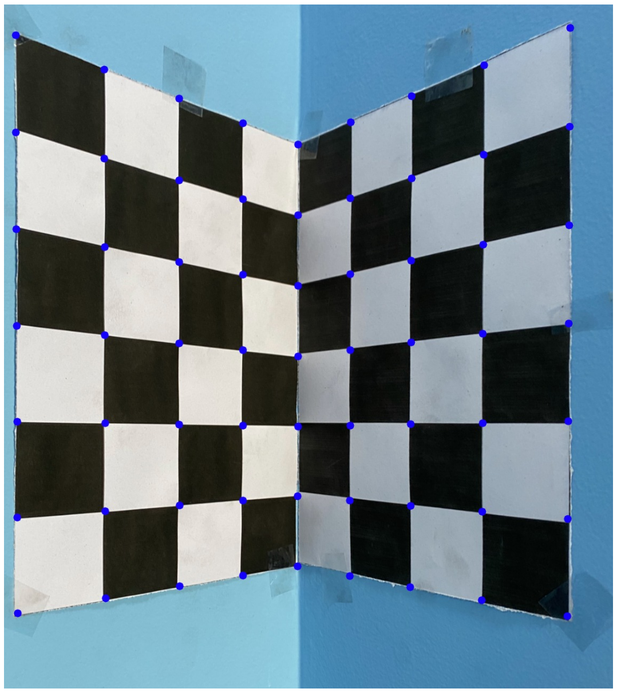

Figure 4: Reprojected points for all the corners of the grid

6.3 Why did we normalise points before computing DLT?

- Some images can have the origin at the top left and others can have it at the bottom left or somewhere else. Also, size of image in pixels varies. Normalisation centers the coordinate origin and also scales the effective size of the image.
- We do not know the distribution of our data. In multiple attempts of the algorithm, we may choose any N(greater than 5) points on the image. They may be near or may be far away which changes the mean and variance of the data in each case and can possibly cause the algorithm to perform differently on different input points. Normalising the input points makes the mean and variance of the data standardised every time and we can expect better consistency due to this.
- When the values in the matrix aren't normalised, they may be very large in magnitude(often the case with pixel values) and the magnitude of different points varies by a large scale. If we take the world coordinates in metres, then the values will be very small. This leads to large magnitude values in the projection matrix and other matrices. As a result, all the mathematical operations that we do in Numpy or otherwise become computationally more expensive since now we are dealing with larger variables and matrices with large determinants. For eg: Dividing a large number by a small number can lead to numerical overflow
- In the famous paper by Hartley (https://www.cse.unr.edu/ bebis/CS485/Handouts/hartley.pdf) which introduced this normalisation process, it is claimed that the singularity of the matrix $\mathscr{P}^T\mathscr{P}$ is changed by normalisation which improves the numerical stability of the algorithm.