Clab-3 Report

ENGN4528

Benjamin Hofmann

u6352049

8/06/2020

ENGN4528 Computer Vision
Computer Lab-3

Task-1: 3D-2D Camera Calibration

1. List calibrate function

The calibration function identified the number of inputs through the dimensions of uv. Converted XYZ to homogenous coordinate system by adding a column of ones. For each input a for loop was used to calculate the corresponding A matrix values. This was achieved using the following 2 lines of code:

B[0,:] = [0,0,0,0,-XYZ[i,0],-XYZ[i,1],-XYZ[i,2],-XYZ[i,3],uv[i,1]*XYZ[i,0],uv[i,1]*XYZ[i,1],uv[i,1]*XYZ[i,2],uv[i,1]*XYZ[i,3]]

B[1,:] = [XYZ[i,0],XYZ[i,1],XYZ[i,2],XYZ[i,3],0,0,0,0,-uv[i,0]*XYZ[i,0],-uv[i,0]*XYZ[i,1],-uv[i,0]*XYZ[i,2],-uv[i,0]*XYZ[i,3]]

This B matrix was then added to the overall A matrix. Then SVD of A was undertaken with the last column of v being reshaped into the calibration matrix. THe calibration matrix was then normalised and returned. The entire function can be found in appendix A below or in the coding submission.

2. Image Used

The image, stereo2012d.jpg was selected for the experiment. This is shown in figure 1.

Figure 1: stereo2012d.jpg used in the camera calibration tests.

3. Camera Calibration Matrix C

The camera calibration matrix calculated using the calibration function described in part 1 is given in table 1.

Table 1: Camera Calibration matrix			
17.815	-8.958	-49.283	378.615
4.421	-49.892	3.452	319.808
-0.039	-0.024	-0.037	1

4. Result of decomposition of C matrix

The results of the decomposition of the calibration matrix given in part 3 is shown below in tables 2, 3 and 4. These tables refer to the K, R and t matrices respectively. Values in each of these tables has been rounded to 3 decimal points.

Table 2: K matrix

818.195	3.424	395.632
0	819.342	255.551
0	0	1

Table 3: R matrix

0.690	0.011	-0.724
0.298	-0.916	0.270
-0.660	-0.402	-0.635

Table 4: T matrix

11.127
8.104
10.232

The projection of the XYZ values with the calibration matrix can be seen in figure 2 along with the calibration's uv inputs and a projection of unit dimensions.

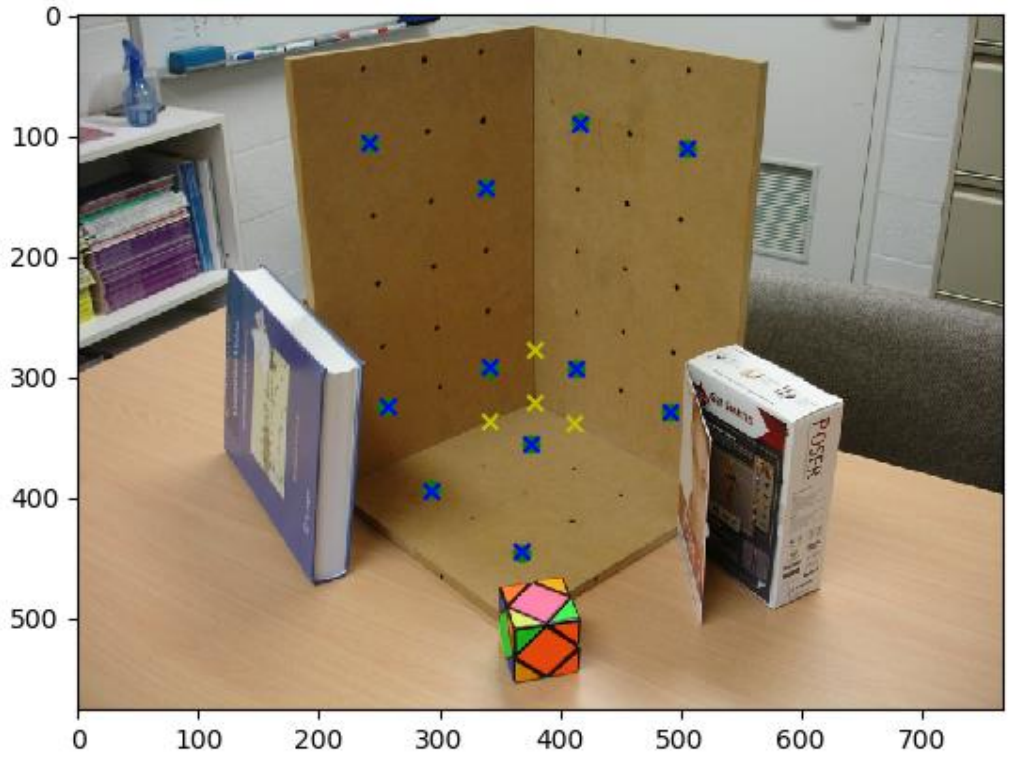

Figure 2: Result of projection of XYZ coordinates using calibration matrix (blue), projection of unit dimensions (yellow) and uv inputs (green).

The yellow crosses mark the origin and 1-unit movement along each axis. Green circles represent the uv coordinates selected and the blue crosses are the projection of the XYZ coordinates onto the image. One can see that the difference between the green circles and blue crosses is very little and this reflects the accuracy of the result.

The residual was found by calculating the distance between the uv inputs and the estimated projection. The average residual was 0.2607.

5. Discussion

a. Focal Length

The focal lengths in the x and y direction can be found using the following 3 formulas:

[1,1] = fx
[2,2] = fy/sin(theta)
[1,2] = -fx/cot(theta)

Through this the skew was found to be approximately 1.57. The focal length in x direction is 818.195 and in the y direction it is 819.3348.

b. Pitch Angle

Ar

$$R_Z(\theta)=\begin{bmatrix}\cos(\theta) & -\sin(\theta) & 0\\ \sin(\theta) & \cos(\theta) & 0\\ 0 & 0 & 1\end{bmatrix} \quad R_X(\phi)=\begin{bmatrix}1 & 0 & 0\\ 0 & \cos(\phi) & -\sin(\phi)\\ 0 & \sin(\phi) & \cos(\phi)\end{bmatrix}$$

$$R_Y(\kappa)=\begin{bmatrix}\cos(\kappa) & 0 & -\sin(\kappa)\\ 0 & 1 & 0\\ \sin(\kappa) & 0 & \cos(\kappa)\end{bmatrix}$$

Figure 3: Rotation Matrix formation [2]

The matrix was combined into a single matrix. The value of rotation k was isolated and each corresponding angle calculated through comparison with the r matrix in table 3. The corresponding angles were found to be: θ = 0, γ = -23.7, κ = 129.42. The pitch angle corresponding to the x-z plane is γ = -23.7.

Task-2: Two-View DLT based homography estimation

1. **List code for homography estimation and image with identified points.**

The corresponding points selected are shown in figure 4 below. Points were chosen with consideration for visibility and range in both x and y dimensions.

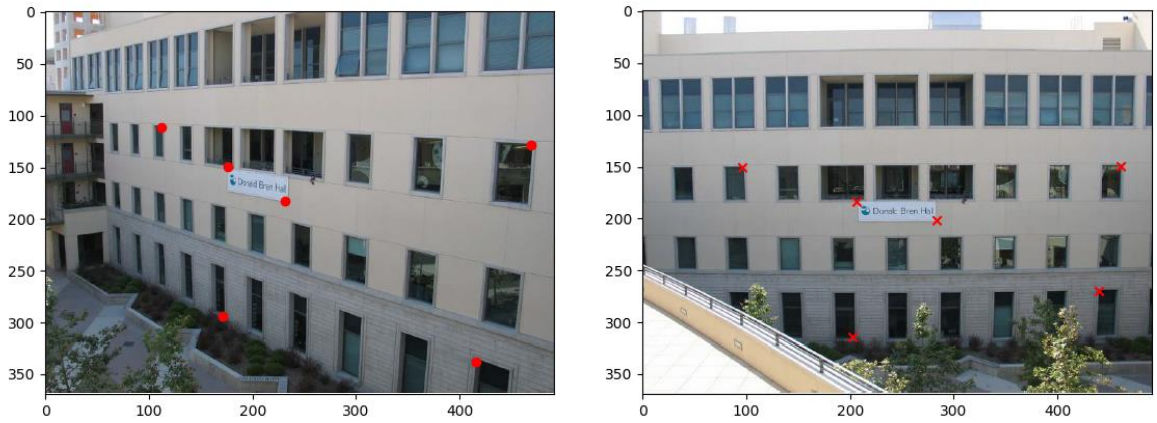

Figure 4: Selection of corresponding points for left view(left) and right view (right)

The homography function takes 4 inputs; u2Trans, v2Trans, uBase and vBase. It calculates the number of inputs from the length of the u2Trans array. Similar to the calibration function, it determines the A matrix values for each input using a for loop. Within the loop it assigns the appropriate uv coordinates within the B matrix for both the leftand right view imagesand then adds the B matrix to the overall A matrix. SVD of A is then undertaken and the appropriate v matrix determined. The last column of this matrix is then reshaped and the

matrix normalised before being returned. A copy of this code can be found in Appendix B or in the coding submission.

2. Camera Homography matrix

The homography matrix shown in table 5 is found using the results of the points selected in part 1.

Table 5: Homography Matrix		
0.294914	0.020207	70.59195
-0.11576	0.898189	-24.0594
-0.00125	0.00017	1

3. Warped image according to homography. Discuss rectification results.

Figure 5 shows the image warped using the homography matrix described in part 2. Different selections of corresponding points naturally produced slightly different homography matrices, thus affecting the rectification results.

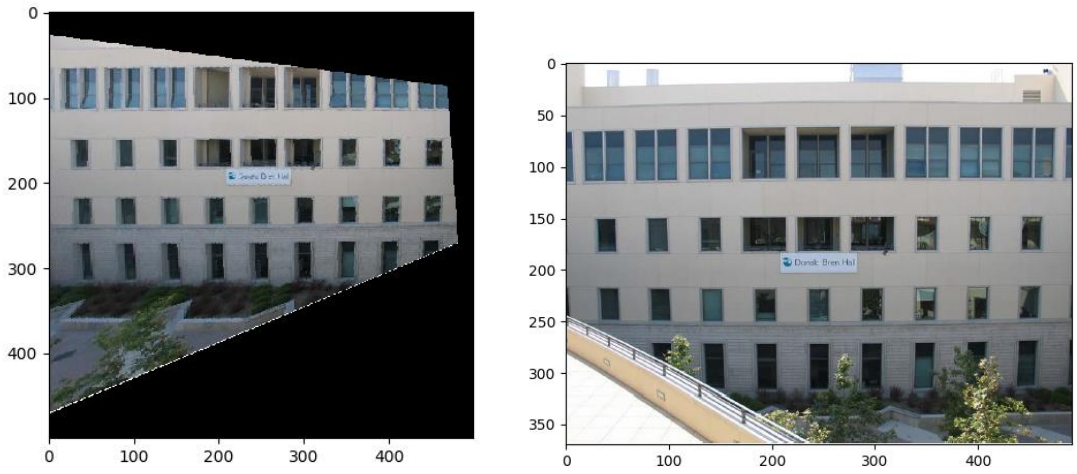

Figure 5: Comparison of Warped image (left) to front view (right)

General it was found that the greater the distance between corresponding points, the greater the area of focus. This meant that more of the image could be seen and recognised when points with greater distance between themselves were chosen. It was also found that small errors or misalignments in inputs had a much greater effect on warping when corresponding points were close together. If values are far apart then this error is relatively small compared to the distance between points, but it can be quite significant if points are compact. This is shown to some extent in figure 6 where the shape of the warping is significantly different from other warped images. This error extends to the over distortion of some areas. In

contrast, figure 7 doesn't appear to have significant over distortion and most of the image in in view.

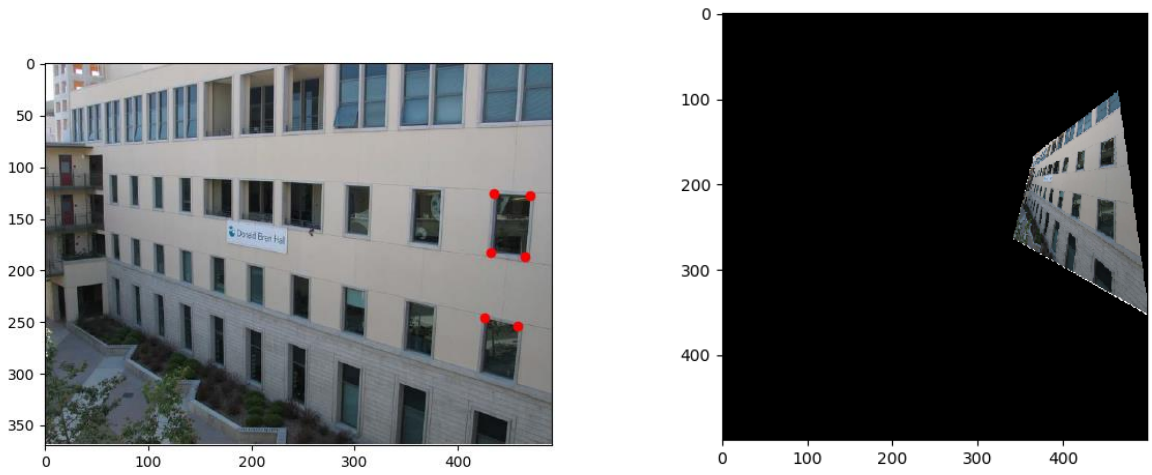

Figure 6: Warping using compact corresponding points (points shown left, warping on the right)

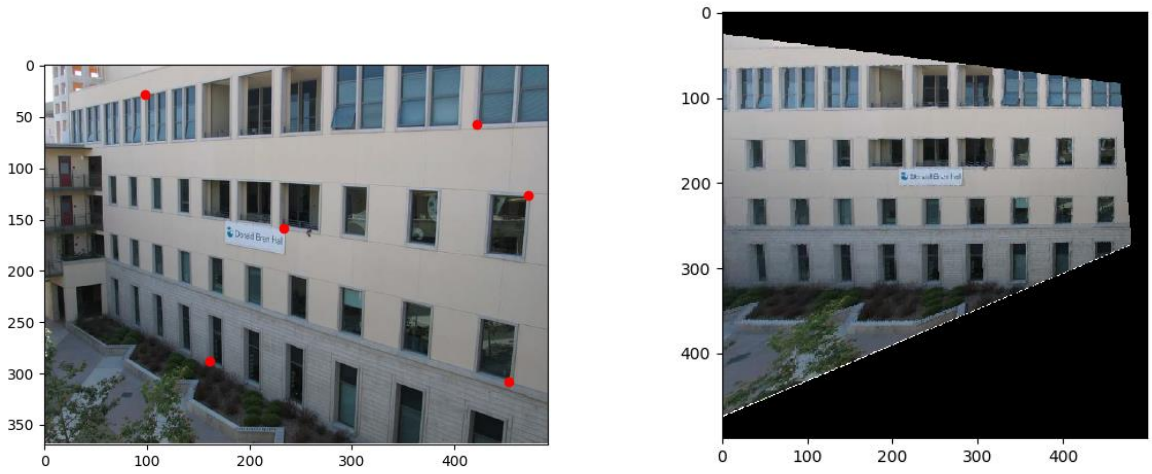

Figure 7: Warping using more distant corresponding points (points shown left, warping on the right)

Point selections on a single direction vector yield very distorted or no images. This is due to the homography process not being able to effectively determine dimensions.

It is better to select corresponding coplanar points in each image that are further apart from one another and extend across as much of the image as possible to produce the most useful and most consistent homography matrices.

It was also observed that increasing the number of corresponding points selected seemed to yield more consistent results and tended to reduce the input error or misalignments. This is due to the increased number of inputs reducing the effect of individual errors.

Resources:

[1] N. Barnes, "ENGN4528: Camera calibration, DLT, homography, DLT, Camera pose, PNP, 1", 2020.

[2] A. Bobick, "Calibration and Projective Geometry", 2020.

[3]A. Mian and M. Ravanbakhsh, "Lecture 8 - Camera Calibration", University of Western Australia, 2020.

[3] G. Roth, "COMP4102A: Homography", Carleton University, Ottawa Canada, 2020.

Other Coding resources

[4] "The Image Module", Effbot.org, 2020. [Online]. Available: http://effbot.org/imagingbook/image.htm. [Accessed: 07- Jun- 2020].

Appendix

Appendix A: Calibrate function code

```
def calibrate(im, XYZ, uv):
    x, _ = uv.shape
    XYZ = np.hstack((XYZ, np.ones((x, 1))))
    B = np.zeros((2, 12))
    A = np.zeros((1,12))
    for i in range(x):
        B[0,:] = [0,0,0,0,-XYZ[i,0],-XYZ[i,1],-XYZ[i,2],-XYZ[i,3],uv[i,1]*XYZ[i,0],uv[i,1]*XYZ[i,1],uv[i,1]*XYZ[i,2],uv[i,1]*XYZ[i,3]]
        B[1,:] = [XYZ[i,0],XYZ[i,1],XYZ[i,2],XYZ[i,3],0,0,0,0,-uv[i,0]*XYZ[i,0],-uv[i,0]*XYZ[i,1],-uv[i,0]*XYZ[i,2],-uv[i,0]*XYZ[i,3]]
        A =np.vstack((A,B))
    A = A[1::,:]
    _,_,c =np.linalg.svd(A) #take last col, reshape
    c = np.reshape(c.T[:, -1], (3, 4))
    c = c / c[2, 3]
    return c
```

Appendix B: Homography function code

```
def homography(u2Trans, v2Trans, uBase, vBase):
    x = len(u2Trans)
    B = np.zeros((2, 9))
    A = np.zeros((1, 9))
    for i in range(x):
        B[0, :] = [-uBase[i],-vBase[i],-1,0,0,0,uBase[i]*u2Trans[i],vBase[i]*u2Trans[i],u2Trans[i]]
        B[1, :] = [0,0,0,uBase[i],vBase[i],1,-uBase[i]*v2Trans[i],-vBase[i]*v2Trans[i],-v2Trans[i]]
        A = np.vstack((A, B))
    A = A[1::, :]
    _, _, v = np.linalg.svd(A)  # = take last col, reshaped
    H = np.reshape(v.T[:, -1], (3, 3))
    H = H / H[2, 2]
    return H
```