Clab-3 Report

ENGN4528

Thao Pham

u7205329

23/05/2021

1 Task 1: 3D-2D Camera Calibration

1.1 Calibrate function

In this task, we will use the image `stereo2012a.jpg` to perform camera calibration. We first prepare the input.

- **XYZ**: Given the 3D coordinate system, we first selected 12 points (4 points on each plane) to be used in the DLT algorithm. The result XYZ is a 12×3 matrix.
- **uv**: For each of the chosen 12 points, we identified their 2D coordinates on the image using `ginput()`. The chosen points are plotted in Figure 1. The result uv is a 12×2 matrix.

Our goal of this task is to use the 3D world coordinates and the corresponding points landed on the image plane to estimate the calibration matrix. The calibrate function was implemented as follows:

```
def calibrate(im, XYZ, uv):
    N = len(uv)
    A = np.zeros((2*N, 12))

    # Compute Ai for each correspondence and assemble matrix A
    n = 0
    for i in range(N):
        Xi, Yi, Zi = XYZ[i]
        ui, vi = uv[i]
        Ai = np.array([[0, 0, 0, 0, -Xi, -Yi, -Zi, -1, vi*Xi, vi*Yi, vi*Zi, vi],
                       [Xi, Yi, Zi, 1, 0, 0, 0, 0, -ui*Xi, -ui*Yi, -ui*Zi, -ui]])
        A[n:n+2,:] = Ai
        n += 2

    # SVD and normalisation
    U,D,V = np.linalg.svd(A)
    p = V[-1]
    p = p/np.linalg.norm(p)

    C = np.reshape(p,(3,4))

    # Get projected points using the calibration matrix C
    XYZ_homogenous = np.column_stack((XYZ, np.ones(N)))
    projected_points = np.zeros(uv.shape)
    squared_error = []

    for i in range(N):
        x_hat = C@XYZ_homogenous[i] # projected point in homogenous coords
        x_hat = x_hat/x_hat[-1] # convert projected point to Cartesian coords
        projected_points[i] = x_hat[:2]
        err = x_hat[:2] - uv[i]
        squared_error.append(err*err) # add squared error

    # Overlay the the world coordinate system
    wcs = np.array([[0,0,0], [0,0,1], [0,1,0], [1,0,0]])*7
    wcs_homogenous = np.column_stack((wcs,np.ones(len(wcs))))

    image_cs = []
    for i in range(len(wcs_homogenous)):
        x_hat = C@wcs_homogenous[i]
        x_hat = x_hat/x_hat[-1]
        image_cs.append(x_hat[:2])

    # Report the error
    print('Mean squared error between uv and projected points:
        {}'.format(np.mean(squared_error)))
```

```
# Plot the image with uv coords as well as projected points
fig,axs = plt.subplots(figsize = (9,6.923))
axs.imshow(im)
axs.scatter(projected_points[:,0], projected_points[:,1], marker = 'o', color = 'yellow', s
    = 30, label = 'Projected')
axs.scatter(uv[:,0], uv[:,1], marker = 'x', color = 'red', s = 30, label = 'uv')

# Overlay the world coordinate system
coords = ('z', 'y', 'x')
for i in range(1,len(image_cs)):
    axs.arrow(image_cs[0][0], image_cs[0][1],
            image_cs[i][0] - image_cs[0][0],
             image_cs[i][1] - image_cs[0][1], head_width = 10, color = 'k')
    axs.text(image_cs[i][0]+10, image_cs[i][1]+10, coords[i-1], c = 'white', fontsize = 15)
axs.legend()
axs.axis('off')
plt.show()

return C
```

Using the prepared parameters as above, we yielded the 3×4 camera calibration matrix P as follows:

```
[[ 8.93387541e-03 -4.28501327e-03 -1.32964565e-02  6.94166414e-01]
 [-1.04508048e-04 -1.58422754e-02  2.54442448e-03  7.19441374e-01]
 [-9.58216349e-06 -7.22470683e-06 -1.29438062e-05  2.15972720e-03]]
```

Multiply the XYZ coordinates by this matrix, we projected the 3D points onto the 2D image plane. The projections can be seen in Figure 1. The mean squared error reported by the calibrate function was 0.1345 (4 dp). The result indicates that the distances between the uv coordinates and the projected points are very small, which is desirable. Indeed, the error is inevitable given the systematic error when using `ginput()`.

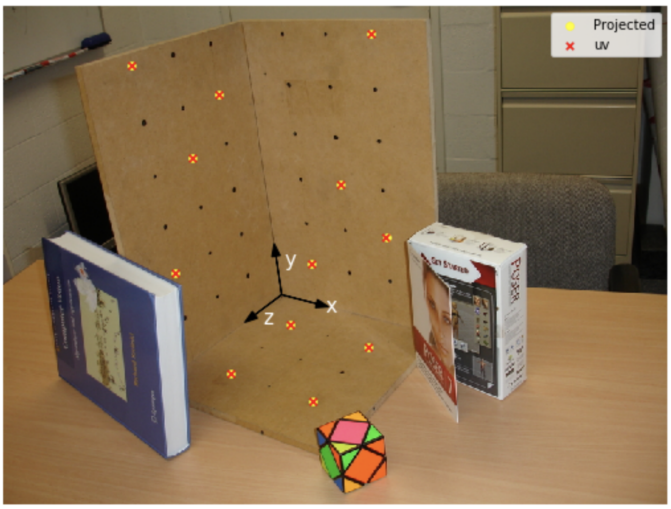

Figure 1: `stereo2012a.jpg` with the uv coordinates (in red crosses) and projected points (in yellow dots) of the chosen 12 points.

1.2 Decompose matrix P

Matrix M can be further decomposed into intrinsic ($\mathbf{K}$) and extrinsic ($\mathbf{R}, \mathbf{t}$) parameters as $\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}]$, where

$$\mathbf{K} = \begin{bmatrix} 860.455 & 7.408 & 377.007 \\ 0 & 869.608 & 264.872 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} 0.8247 & -0.0950 & -0.5575 \\ 0.1585 & -0.9074 & 0.3891 \\ -0.5429 & -0.4093 & -0.7333 \end{bmatrix}, \quad \mathbf{t} = \begin{bmatrix} 71.4896 \\ 58.0358 \\ 81.5373 \end{bmatrix}$$

From $\mathbf{K}$, we have the focal lengths $f_x = 860.455$ and $f_y = 869.608$, both measured in pixels. We now attempt to find the pitch angle with respect to the XZ-plane. According to LaValle (2012), with respect to a pre-defined world coordinate system, the rotation matrix $\mathbf{R}$ is made up of the following three rotations:

- a yaw is counter-clockwise rotation of α about the y-axis, $R_y(\alpha) = \begin{bmatrix} cos(\alpha) & 0 & sin(\alpha) \\ 0 & 1 & 0 \\ -sin(\alpha) & 0 & cos(\alpha) \end{bmatrix}$
- a pitch is a counter-clockwise rotation of β about the x-axis, $R_x(\beta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\beta) & -sin(\beta) \\ 0 & sin(\beta) & cos(\beta) \end{bmatrix}$
- a roll is a counter-clockwise rotation of γ about the z-axis, $R_z(\gamma) = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Following this, the rotation matrix $\mathbf{R}$ as we know is formed by multiplying the above matrices in the order: yaw, pitch, and roll.

$$\begin{aligned} \mathbf{R}(\alpha, \beta, \gamma) &= R_z(\gamma) R_x(\beta) R_y(\alpha) \\ &= \begin{bmatrix} cos(\gamma)cos(\alpha) - sin(\gamma)sin(\beta)sin(\alpha) & -sin(\gamma)cos(\beta) & cos(\gamma)sin(\alpha) + sin(\gamma)sin(\beta)cos(\alpha) \\ sin(\gamma)cos(\alpha) + cos(\gamma)sin(\beta)sin(\alpha) & cos(\gamma)cos(\beta) & sin(\gamma)sin(\alpha) - cos(\gamma)sin(\beta)cos(\alpha) \\ -cos(\beta)sin(\alpha) & sin(\beta) & cos(\beta)cos(\alpha) \end{bmatrix} \end{aligned}$$

From this, we have $sin(\beta) = -0.4093$ which in turn gives $\beta = -24.16^o = 335.840^o$.

Acknowledgement: The answer for this section was inspired by http://planning.cs.uiuc.edu/node102.html. This was used in understanding the definition of pitch angle and how it relates to rotation matrix as well as pre-defined world coordinates. The answer was constructed independently by myself with details adjusted according to the question asked.

1.3 Resize image

For this task, we resized `stereo2012a.jpg` to (H/2, W/2) and identified new uv coordinates similar to above. Following this, we applied the calibrate function to find a new matrix $\mathbf{P'}$ as can be seen below

```
[[ 9.04242532e-03 -4.23977582e-03 -1.31279191e-02  6.93543378e-01]
 [-3.78358528e-05 -1.58932468e-02  2.80046489e-03  7.20032288e-01]
 [-1.86441981e-05 -1.40328552e-05 -2.35566448e-05  4.30864468e-03]]
```

Using $\mathbf{P'}$, the new projected points are displayed in Figure 2. The MSE reported is 0.0270 (4 dp).

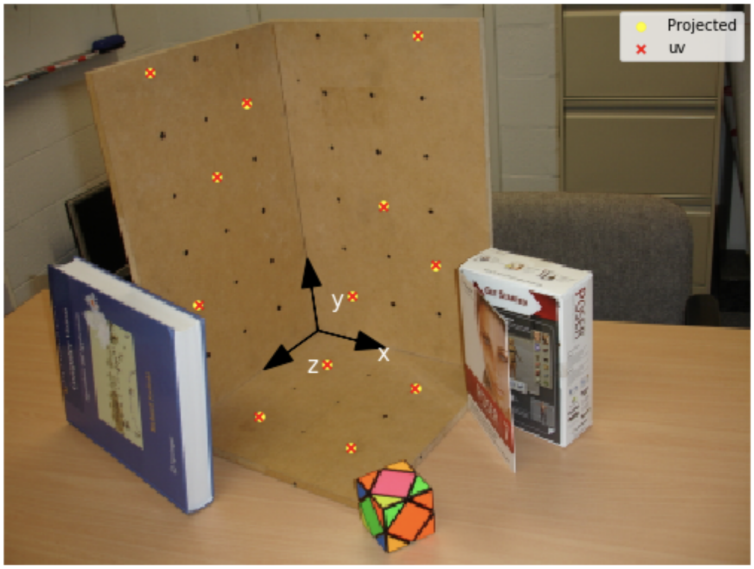

Figure 2: Resized `stereo2012a.jpg` with uv coordinates and projected points marked.

Once again, we decomposed $\mathbf{P}'$ into $\mathbf{P}' = \mathbf{K}'[\mathbf{R}'|\mathbf{t}']$ where

$$\mathbf{K}' = \begin{bmatrix} 462.948 & 3.043 & 182.053 \\ 0 & 465.073 & 143.494 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{R}' = \begin{bmatrix} 0.8091 & -0.1039 & -0.5785 \\ 0.1710 & -0.9001 & 0.4008 \\ -0.5623 & -0.4232 & -0.7104 \end{bmatrix}, \quad \mathbf{t}' = \begin{bmatrix} 76.7607 \\ 60.3144 \\ 86.2228 \end{bmatrix}$$

Comparing the decomposition of $\mathbf{P}'$ to that of $\mathbf{P}$, we see some drastic changes in the intrinsic parameters. Mainly, we see the elements of $\mathbf{K}'$ (focal length, offset of camera centre, and skew) are more than half of the elements of $\mathbf{K}$. These changes are expected since scaling by half will give the following transformation:

$$u_i' = \frac{u_i}{2}, \quad v_i' = \frac{v_i}{2}$$

where u_i and v_i are the coordinates of any pixel in the original image and u_i' and v_i' are that of resized image. Indeed, ideally we would have seen the elements in $\mathbf{K}'$ to be exactly one half of the elements in $\mathbf{K}$. We can write this in terms of matrix multiplication:

$$\begin{aligned} \begin{bmatrix} u_i' \\ v_i' \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} &= \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R}|\mathbf{t}] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.5f_x & 0.5s & 0.5c_x \\ 0 & 0.5f_y & 0.5c_y \\ 0 & 0 & 1 \end{bmatrix} [\mathbf{R}|\mathbf{t}] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \end{aligned}$$

With extrinsic parameters, it appears that there are very minor changes. This is expected since there was no alteration in how the image was taken, thus the process of projecting points from the world coordinates to camera remained the same. Thus, ideally we would not have seen any changes in $\mathbf{R}$ and $\mathbf{t}$. Indeed, the small changes are not due to the scaling of the image. It is rather due to the inconsistencies when using `ginput()` to get the 2D coordinates.

Acknowledgement: The answer for this section was partially inspired by https://dsp.stackexchange.com/questions/6055/how-does-resizing-an-image-affect-the-intrinsic-camera-matrix (answered by Rubshtein, A. (Nov, 2012)). This was used in helping understanding the question. The final answer was constructed independently by myself according to the question asked.

2 Task 2: Two-View DLT based homography estimation

2.1 Homography function

Our goal for this task is to use pairs of matched points from two images to estimate the homography matrix that best explains the projective transformation that has occurred between the source and destination images.

Similar to Task 1, we selected 6 soure coordinates in `left.jpg` and identified their 2D coordinates using `ginput()`. We then found the 6 corresponding coplanar destination points in `right.jpg` and displayed them in Figure 3. We have the matrices `uv-left` and `uv-right` (each of the size 6×2) where each row represents the image coordinates of the each chosen point in the left and right image, respectively. `uv-left` was then spitted to two different vectors `uBase` and `vBase`. Similar procedure was done on `uv-rightt` to obtain `u2Trans` and `v2Trans`.

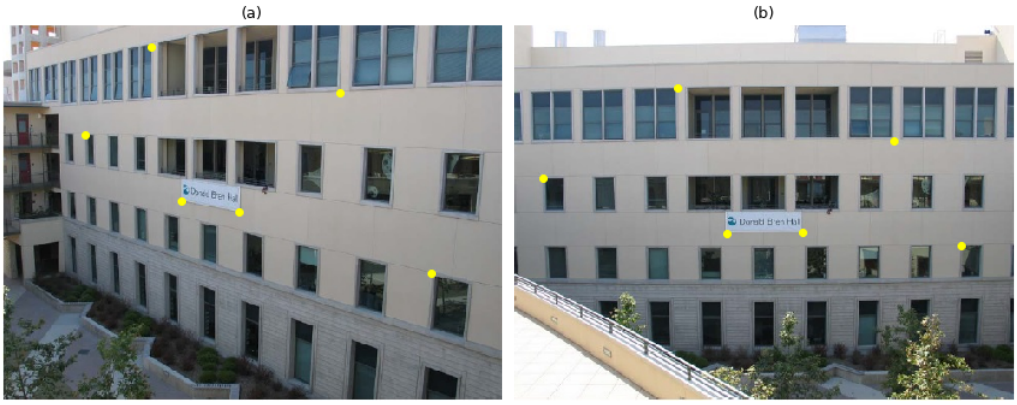

Figure 3: `left.jpg` and `right.jpg` with the chosen points marked with yellow dots

The homography function was then implemented using the DLT algorithm as follows:

```
def homography(u2Trans, v2Trans, uBase, vBase):
    N = len(uBase)
    A = np.zeros((2*N, 9))

    # Calculate Ai for each correspondence and assemble A
    n = 0
    for i in range(N):
        xi, yi = uBase[i], vBase[i]
        xi_, yi_ = u2Trans[i], v2Trans[i]
        Ai = np.array([[0, 0, 0, -xi, -yi, -1, yi_*xi, yi_*yi, yi_],
                       [xi, yi, 1, 0, 0, 0, -xi_*xi, -xi_*yi, -xi_]])
        A[n:n+2,:] = Ai
        n += 2

    # SVD and normalisation
    U,D,V = np.linalg.svd(A)
    h = V[-1]
    h = h/np.linalg.norm(h)

    H = np.reshape(h,(3,3))
    return H
```

2.2 Homography matrix

By applying the homography function with the inputs as described above, we obtained an estimation of the 3×3 matrix H as follows:

```
[[ 1.42056426e-02 -4.20179144e-05 -9.99142030e-01]
 [ 2.23954006e-03  6.36855850e-03 -3.80737848e-02]
 [ 1.73865674e-05 -1.03505527e-06  4.26793023e-03]]
```

2.3 Warp

Transforming the left image using the estimated homography, we yielded a rectified image as shown in Figure 4. This was done using the Python in-built function `cv2.warpPerspective()`. For better visualisation, the warped image was overlaid on the `Right.jpg` image with the destination coordinates marked in yellow dots and the warped points in red crosses. It can be seen that aside from the occluded areas, the estimated H did a good job in restoring the destination image. To achieve better results, we can try selecting more points or experiment with different combinations of matched points. Furthermore, it is important to choose points that are visible and have similar appearance across both images.

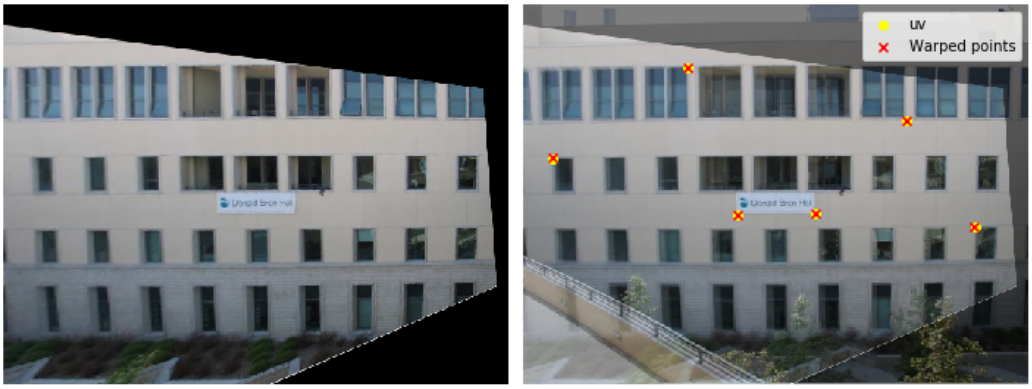

Figure 4: `Left.jpg` rectified using the estimated H

3 Reference

LaValle, S. (2006). *Yaw, pitch, roll rotations.* Planning Algorithms. Cambridge University Press. Retrieved from: http://planning.cs.uiuc.edu/node102.html

Rubshtein, A. (2012). *How does resizing an image affect the intrinsic camera matrix?* Stack Exchange. Retrieved from https://dsp.stackexchange.com/questions/6055/how-does-resizing-an-image-affect-the-intrinsic-camera-matrix