Clab-3 Report

ENGN6528

Shreya Chawla
u7195872

23/05/2021

Task-1: 3D-2D Camera Calibration (15 marks)

1. The code for `calibrate` function is as follows (Fig 1). It first normalizes the parameters and then performs DLT, followed by denormalization by making p34 = 1. It first computes A_i and then combines them to form A. Next the solution for P is found. The normalization of datapoints is done using normalize_2d and normalize_3d functions (Fig 1 (b)). The camera calibration matrix is computed in calibrate function (Fig 1 (a)).

```
43 def calibrate(im, XYZ, uv):
44     # Normalize the data in homogenized form
45     norm_uv, T_norm = normalize_2d(im, uv)
46     norm_XYZ, S_norm = normalize_3d(im, XYZ)
47 
48     # Find A
49     A = []
50     # Assemble n 12x2 Ai matrices into 2nx12 matrix A
51     for i in range(XYZ.shape[0]):
52         X, Y, Z = norm_XYZ[i, 0], norm_XYZ[i, 1], norm_XYZ[i, 2]
53         u, v = norm_uv[i, 0], norm_uv[i, 1]
54         # Compute Ai
55         A.append(np.array([[X, Y, Z, 1, 0, 0, 0, 0, -u * X, -u * Y, -u * Z, -u],
56                            [0, 0, 0, 0, X, Y, Z, 1, -v * X, -v * Y, -v * Z, -v]]))
57     A = np.array(A)
58     A = A.reshape((A.shape[0] * A.shape[1], -1))
59     # Compute SVD of A
60     U, S, V = np.linalg.svd(A)
61     # The solution for P is the last column of V <- smallest singular value of SVD of A
62     p = V[-1]
63     # Camera projection matrix
64     P = p.reshape(3, 4)
65     # Normalize P such that P34 = 1
66     P = P / P[-1, -1]
67 
68     # Denormalization
69     # pinv: Moore-Penrose pseudo-inverse of a matrix, generalized inverse of a matrix using its SVD
70     # P = T_norm ^ -1 @ P_normalized @ S_norm
71     P = np.linalg.pinv(T_norm) @ P @ S_norm
72     P = P / P[-1, -1]  # Re-normalize so that P34 = 1
73 
74     # Mean squared error between the positions of the uv coordinates and the projected XYZ coordinates
75     uv2 = np.dot(P, np.concatenate((XYZ.T, np.ones((1, XYZ.shape[0])))))  # Projected XYZ
76     uv2 = uv2 / uv2[2, :]
77     err = np.sqrt(np.mean((uv2[:2].T - uv) ** 2))
78     print("Error = ", err)
79 
80     return P
```

(a)

```
'''
normalize_2d function: Normalize 2D image points using T_norm matrix (formula from lecture slides)
Arguments:
im = image                    - needed to find height and width for T_norm matrix
x  = uv points marked on image - to be normalized using T_norm
Returns:
x      = Normalized uv coordinates of x
T_norm = Matrix used to normalize and denormalize the 2D points in x
'''
def normalize_2d(im, x):
    H, W, _ = im.shape
    T_norm = np.linalg.pinv(np.array([[W + H, 0, W / 2], [0, W + H, H / 2], [0, 0, 1]]))  # formula from lecture slides
    x = np.dot(T_norm, np.concatenate((x.T, np.ones((1, x.shape[0])))))
    x = x.T  # Normalized x
    # Need T_norm matrix later for denormalization
    return x, T_norm

'''
normalize_3d function: Normalize 3D world points using S_norm matrix
Arguments:
X  = XYZ real world points - to be normalized using S_norm
Return:
X      = Normalized XYZ coordinates of X
S_norm = Matrix used to normalize and denormalize the 3D points in X
'''
def normalize_3d(X):
    V_diag_Vinv = 0
    mu = np.mean(X, axis=0)
    for i in range(len(X)):
        V_diag_Vinv += ((np.reshape((X[i] - mu), (-1, 1))) @ (np.reshape((X[i] - mu), (1, -1))))
    diag, V = np.linalg.eig(V_diag_Vinv)
    V_inv = np.linalg.pinv(V)
    diag_inv = np.array([[1 / diag[0], 0, 0], [0, 1 / diag[1], 0], [0, 0, 1 / diag[2]]])
    first = V @ diag_inv @ V_inv
    second = (-V @ diag_inv @ V_inv @ mu).reshape(-1, 1)
    third = np.array([0, 0, 0, 1])
    S = np.append(first, second, axis=1)
    S_norm = np.vstack((S, third))  # formula from lecture slides
    X = S_norm @ np.concatenate((X.T, np.ones((1, X.shape[0]))))
    X = X.T  # Normalized X
    # Need T_norm and S_norm matrices for denormalization later
    return X, S_norm
```

(b)

Fig 1: The code for (a) calibrate function to find the 3x4 camera calibration matrix P, find error and return P, and (b) normalize_2d and normalize_3d functions to normalize 2D image coordinates (x) and 3D world coordinates (X).

2. The "stereo2012d.jpg" image has been experimented on as displayed below in Fig 2. The selected points are displayed below in Fig 3.

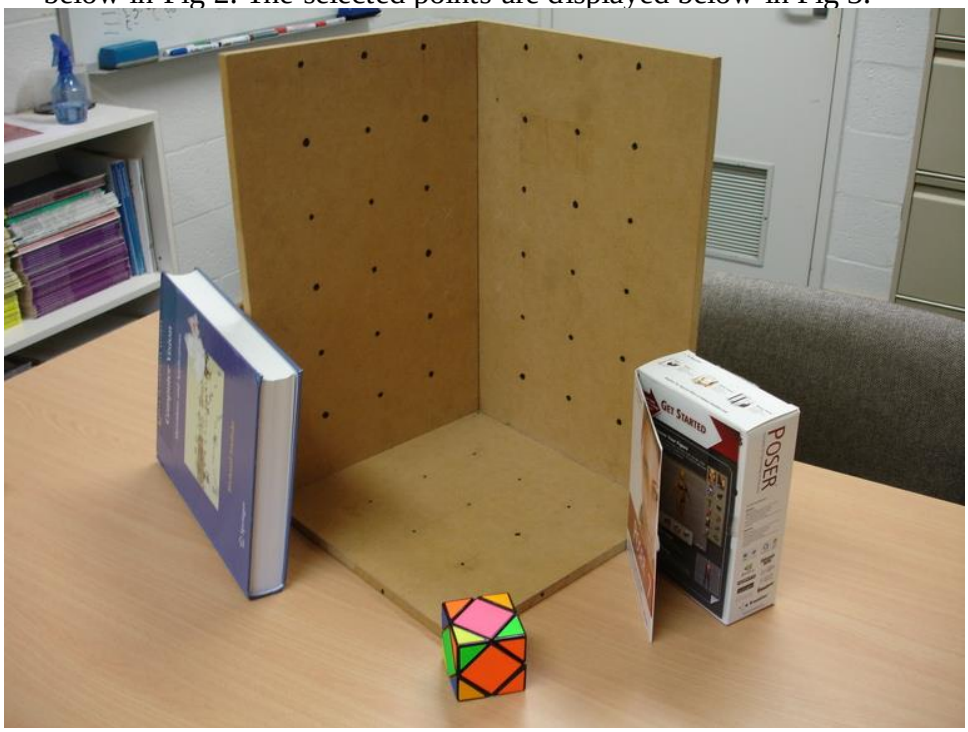

Fig 2: "stereo2012d.jpg" image

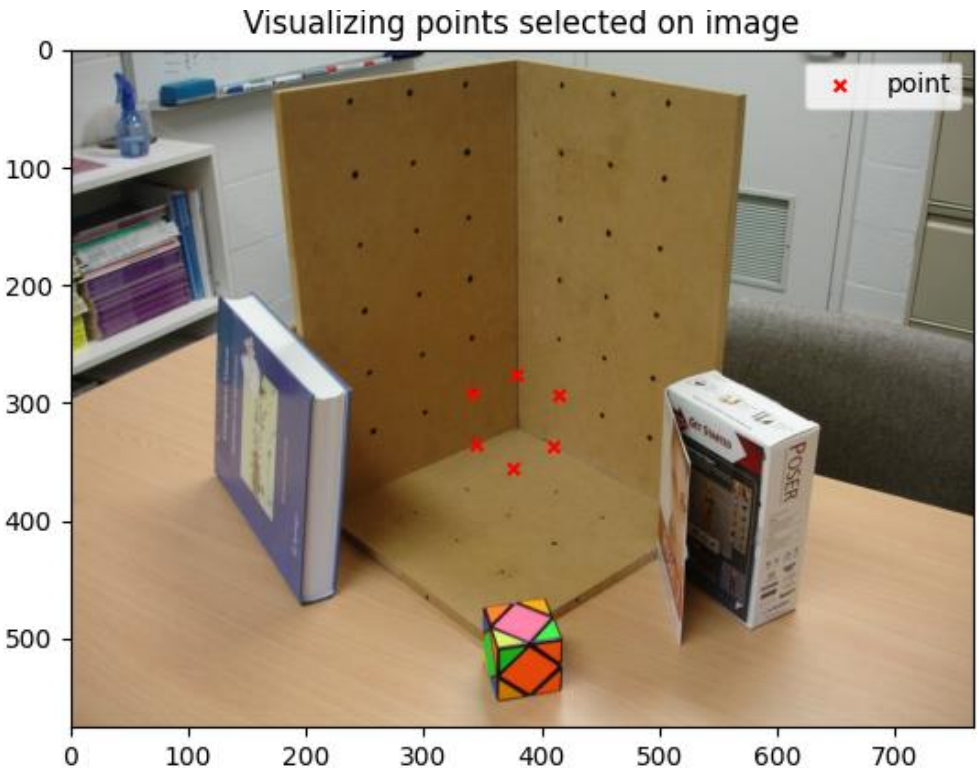

Fig 3: Points selected on image are marked with a red cross

The uv coordinates of the 6 points are:
[(409.3304745052724, 336.66585012245076),
(415.45088341890676, 294.04868980867457),
(379.8274868931626, 278.7025640404842),
(341.9255565276484, 291.0008576484182),
(343.6562469223144, 335.1164969699158),
(375.5099708809897, 355.2266955586122)]

3. The 3x4 camera calibration matrix P calculated for the selected image is given as:

P =
[[1.51411381e-01, -5.04003440e+00, -9.44281466e+00, 3.77752366e+02],
[-1.16158720e+00, -9.79471515e+00, -2.27444884e+00, 3.19798281e+02],
[-1.06350210e-02, -1.40816082e-02, -1.32875447e-02, 1.00000000e+00]]

The projection of the XYZ coordinates onto image is displayed below (Fig 4).

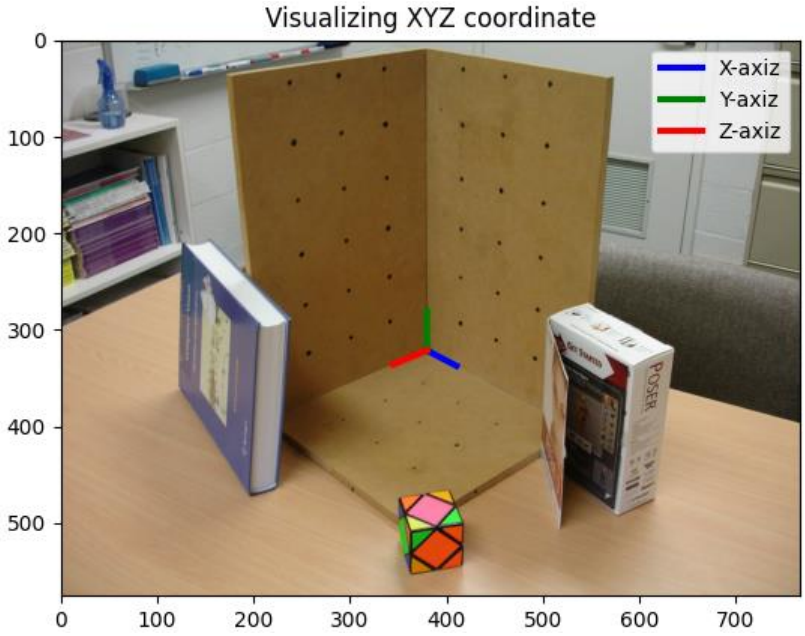

Fig 4: X,Y,Z axis on image

The mean squared error between the positions of the uv coordinates and the projected XYZ coordinates is 0.06645263289185949.

4. The P matrix is decomposed into K, R, t, such that P = K[R|t], by using the code provided (importing function from `vgg_KR_from_P.py`). The results, namely the K, R, t matrices are listed below.

4.1) K =
[[274.40528396, -10.64221367, 399.28613434],
[0.0, 270.44520123, 369.9131688],
[0.0, 0.0, 1.0]]

4.2) R =
[[0.74352981, 0.06633712, -0.66540424],
[0.46408172, -0.76761046, 0.44204336],
[-0.48144737, -0.63747436, -0.60152711]]

4.3) t =
[28.57150669, 22.67640693, 28.35897512]

5. The answer to q5 are as follows:
   i. The focal length of the camera is calculated using the K matrix as follows:

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Where, $\alpha_x = f \cdot m_x$ and $\alpha_y = f \cdot m_y$

The skew angle θ is assumed to be 90°(i.e. γ =0). And $m_x = 1$ and $m_y = 1$. Hence, $\alpha_x = f_x$ and $\alpha_y = f_y$, where f_x and f_y are the focal length along x and y axis. So, using the K matrix calculated in task 1.4 above, we have $\alpha_x = 274.40528396$ and $\alpha_y = 270.44520123$, hence $\mathbf{f_x = 274.40528396}$ and $\mathbf{f_y = 270.44520123}$.

   ii. To find the pitch angle, we use the following formula which uses R rotation matrix of task 1.4:
   pitch angle = $\tan^{-1}$ (r31 / sqrt($r32^2 + r33^2$))
   Hence, pitch angle = $\tan^{-1}$ (-0.31362102895) = -0.00547377°

6. The solution for 1.6 is as follows:
   a. The selected image is resized using cv2's build-in function to (H/2, W/2) as displayed below in Fig 5.

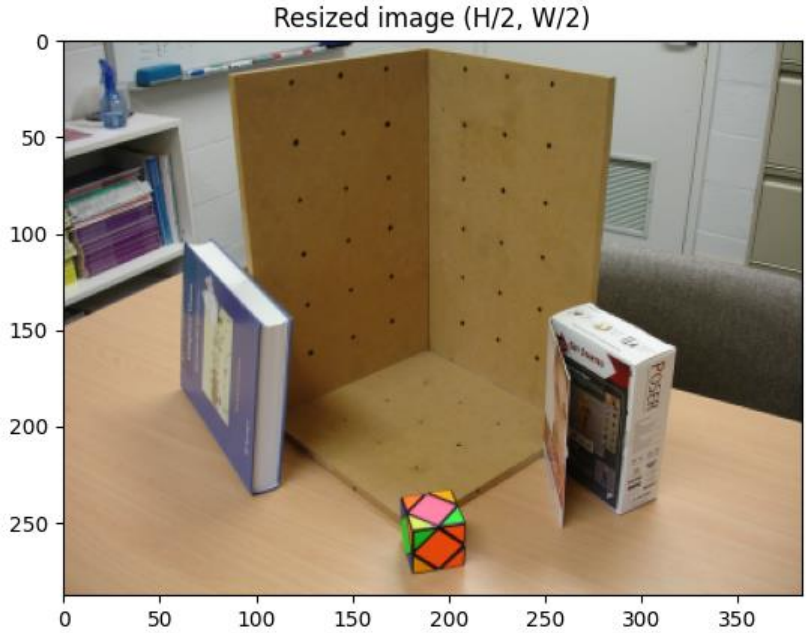

Fig 5: Resized image I_resize

The uv coordinates on the resized image I_resize are taken as half of the points chosen for image I. They can also be selected using the `matplotlib.pyplot.ginput` inbuilt function, by uncommenting code on line 137. The uv coordinates of these 6 points are:
[(204.66523725, 168.33292506)
(207.72544171, 147.0243449)
(189.91374345, 139.35128202)
(170.96277826, 145.50042882)
(171.82812346, 167.55824848)
(187.75498544, 177.61334778)]

The image showing these 6 points is given below (Fig 6)

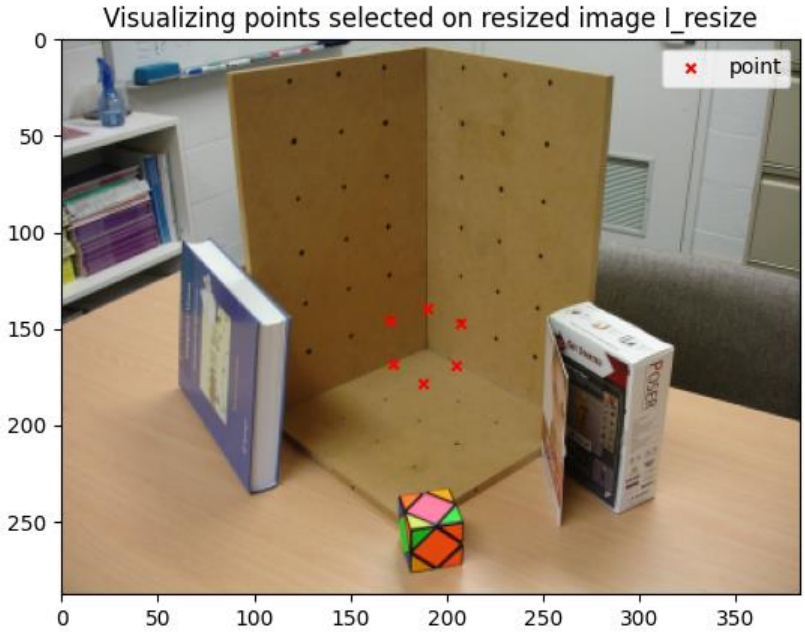

Fig 6: Points selected on I_resize image are marked with a red cross

The error in calculating P' for I_resize is 0.033226316445816965.
The calculated 3x4 camera calibration matrix P' and the decomposed K', R', and t' for this image are as given below:

6.a.1) P' =
[[7.57056904e-02, -2.52001720e+00, -4.72140733e+00, 1.88876183e+02],
[-5.80793600e-01, -4.89735758e+00, -1.13722442e+00, 1.59899141e+02],
[-1.06350210e-02, -1.40816082e-02, -1.32875447e-02, 1.00000000e+00]]

6.a.2) K' =
[[137.20264198, -5.32110683, 199.64306717],
[0.0, 135.22260062, 184.9565844],
[0.0, 0.0, 1.0]]

6.a.3) R' =
[[0.74352981, 0.06633712, -0.66540424],
[0.46408172, -0.76761046, 0.44204336],
[-0.48144737, -0.63747436, -0.60152711]]

6.a.4) t' =
[28.57150669, 22.67640693, 28.35897512]

b. The analysis of the differences is as follows:

1) K and K': The top 2x3 K' matrix is half of the top 2x3 K matrix. This is because when image is scaled to half, then the focal length in X and Y direction would be halved as well. Only with the change in camera intrinsics can give an image scaled by half.

2) R and R': The rotation matrices are the same. This is because by resizing the image, we have not rotated the image but only scaled it to half. Hence R = R'.

3) t and t': The translation matrices are the same. This is because even though the image is resized to half, it would still translate the same in the same directions. The camera extrinsics (R and t matrices) would remain the same as the objects are not moved at all. Hence, t = t'.

Task-2: Two-View DLT based homography estimation. (10 marks)

The original "left.jpg" and "right.jpg" are given below (Fig 7).

(a) Left (b) Right

Fig 7: The original "left.jpg" (a) and "right.jpg" (b) for comparison with results in tasks below.

1. The source code for homography estimation function is given below (Fig 8).

```
194 def homography(u2Trans, v2Trans, uBase, vBase):
195     # The separated coordinates are concatenated
196     trans = np.array(list(zip(u2Trans, v2Trans)))
197     base = np.array(list(zip(uBase, vBase)))
198 
199     # Find A
200     A = []
201     # Assemble n 2x9 Ai matrices into 2nx9 matrix A
202     for i in range(trans.shape[0]):
203         x, y = base[i, 0], base[i, 1]
204         x_p, y_p = trans[i, 0], trans[i, 1]  # x' and y'
205         # Compute Ai
206         A.append(np.array([[x, y, 1, 0, 0, 0, -x_p * x, -x_p * y, -x_p],
207                            [0, 0, 0, x, y, 1, -y_p * x, -y_p * y, -y_p]]))
208     A = np.array(A)
209     A = A.reshape((A.shape[0] * A.shape[1], -1))
210 
211     U, D, V = np.linalg.svd(A)
212     h = V[-1].reshape(3, 3)
213     H = h / h[-1, -1]
214 
215     return H
216 
217
```

Fig 8: Code (screenshot) for homography function

The location of 6 pairs of corresponding selected uv points for the two - left, right - images found using `matplotlib`'s `ginput` function are as follows:

- Points on left image (uvBase) are:
  [(473.7879965645927, 127.95684101055195),
  (154.33057664483135, 113.66474825376037),
  (454.2879114852839, 306.97468411664426),
  (159.69682110542857, 221.18946897828735),
  (234.04957184234223, 184.32341138831464),
  (176.71164495294744, 173.60525070124373)]
- Points on right image (uvTrans) are:
  [(464.27973151929456, 149.07097569585363),
  (172.7535084645985, 148.2839615396607),
  (459.84427236581786, 247.1597446590275),
  (183.40888765673617, 250.86626635795034),
  (282.83730068454497, 200.85654383407527),
  (208.37325409067654, 202.89237510383904)]

These selected points are displayed for the base or the "Left.jpg" image (top) and the trans or the "Right.jpg" image (bottom) in Fig 9. The points are marked with red crosses and are selected such that they are not close.

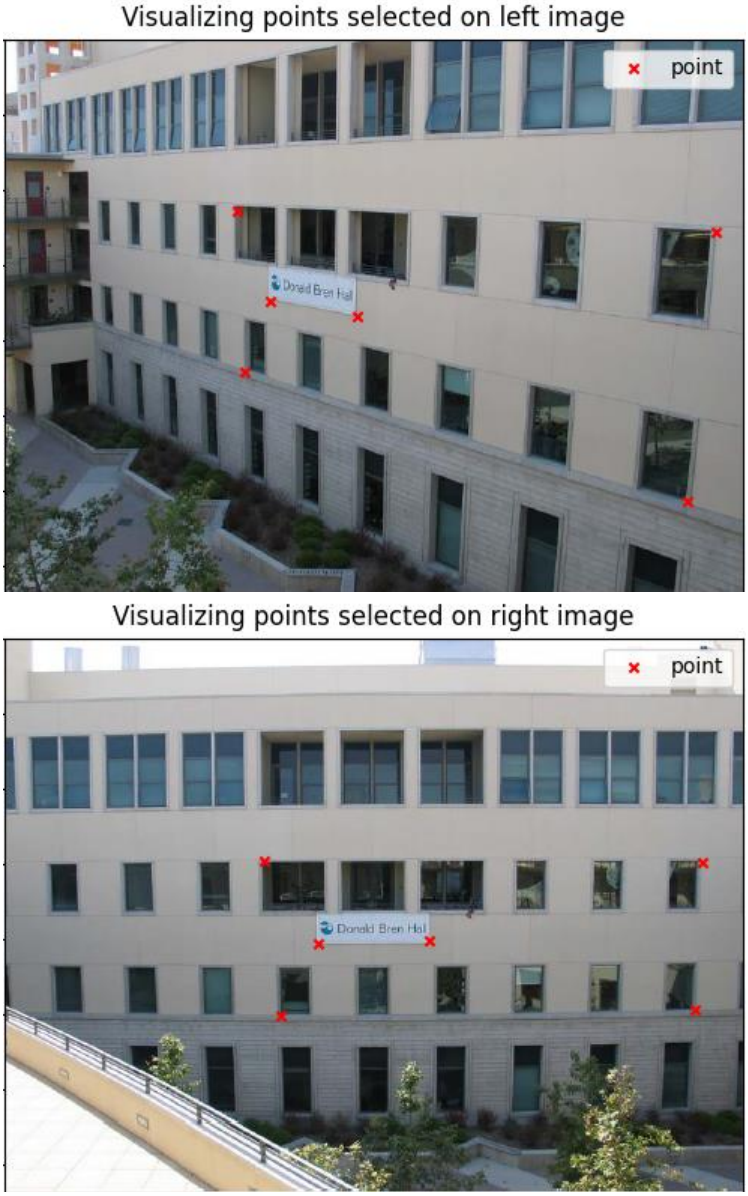

Fig 9: Visualizing points on "Left.jpg" (top) and "Right.jpg" (bottom) images

2. The 3x3 camera homography matrix H calculated by my homography function is as follows:

H = [[3.43598177e+00, -1.26095171e-02, -2.48079929e+02],
[5.62866975e-01, 1.51944423e+00, -1.85795661e+01],
[4.21492716e-03, -2.22267817e-04, 1.00000000e+00]]

3. The left image is warped according to the calculated homography matrix H calculated in 2.1 task. The warping is done using the build-in `cv2.warpPerspective` and the warped image is displayed below in Fig 10.

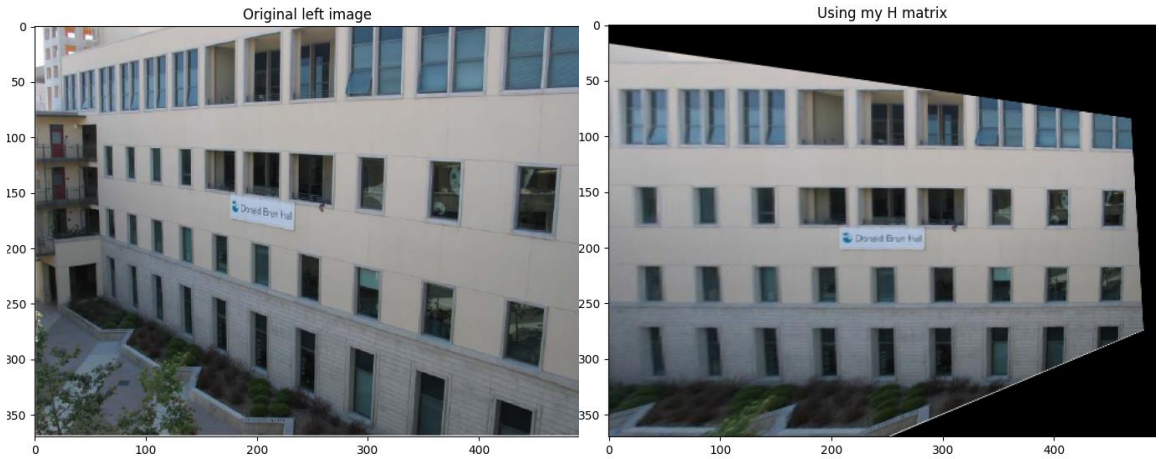

Fig 10: Original "Left.jpg" image (left) and the resulting warped image (right) are compared.

The board on the building wall in the resulting warped (Fig 10 - right) is almost similar to the board in the "right.jpg" image. Also, the windows in center are almost straight. The lines intersect perpendicularly compared to the "left.jpg" (Fig 10 – left) image.

Some of the factors that affect the result of warping are:

- The distance between the corresponding points. The farther the points in image, the better the result of warping. This is because if close points are chosen then there is a big chance that the homograph calculated will not generalize well for the whole image and only for the local area.
- Since the points chosen are not precise, it negatively impacts the result. Because of imprecision, the H matrix is not very accurate.
- Correctly choosing corresponding points in both images. If the points chosen in base and target image do not correspond to each other, then the result would be absurd.
- Whether the chosen corresponding points lie on different line or not. If the points lie on different vanishing lines then the mapping would better.
- The number of points chosen for calculating a homograph. The number of points must be greater than 4. The more the number of points, the better solution for homograph and hence the warped image would be better.