Note: -

1. All the matrix and results(not screenshots) calculated have been listed in this report at the end. Screenshots of the results are provided at the required task number.
2. Everywhere, instead of using '**ginput**' for selecting 2D co-ordinates from image, I have used some functions from cv2 library. This was done to avoid unnecessary conversions from BGR to RGB colour format. Attached is a screenshot of the with comments. Also, the below code plots the selected points on the image.

```
# Callback function for the mouse
positions=[]
count=0
def draw_circle(event,x,y,flags,param):
    # If the event is a Left Button Click, the coordinates should be saved in the lists.
    global positions,count
    if event == cv2.EVENT_LBUTTONUP:
        cv2.circle(stereo,(x,y),2,(255,0,0),-1)
        positions.append([x,y])
        count+=1


# Reading the image and storing it in variable stereo
stereo = cv2.imread('Python user/stereo2012a.png')

# window defining
cv2.namedWindow('image')

cv2.setMouseCallback('image',draw_circle)

while(True):
    cv2.imshow('image',stereo)
    k = cv2.waitKey(20) & 0xFF
    # Exits when pressed Escape or when all 6 points are selected
    if k == 27 or count==6:
        break

cv2.destroyAllWindows()
```

Task 1

1)

```
def Normalization(nd, x):
    '''
    Coordinate normalisation (centroid to origin and sqrt mean distance) (2 or 3).
    Input
    -----
    nd: number of dimensions, 3 in our case
    x: the data that needs to be normalised (various directions in various columns and points in various rows)
    Output
    ------
    Tr: the matrix of transformations (translation plus scaling)
    x: the data that has been altered
    '''

    x = np.asarray(x)
    m, s = np.mean(x, 0), np.std(x)
    if nd == 2:
        Tr = np.array([[s, 0, m[0]], [0, s, m[1]], [0, 0, 1]])
    else:
        Tr = np.array([[s, 0, 0, m[0]], [0, s, 0, m[1]], [0, 0, s, m[2]], [0, 0, 0, 1]])

    Tr = np.linalg.inv(Tr)
    x = np.dot( Tr, np.concatenate( (x.T, np.ones((1,x.shape[0]))) ) )
    x = x[0:nd, :].T

    return Tr, x
```

```
def calibrate(nd, xyz, uv):
    '''
    %% TASK 1: CALIBRATE
    %
    % Function to perform camera calibration
    %
    % Usage:   calibrate(image, XYZ, uv)
    %          return C
    %   Where:   image - is the image of the calibration target.
    %            XYZ - is a N x 3 array of  XYZ coordinates
    %                  of the calibration target points.
    %            uv  - is a N x 2 array of the image coordinates
    %                  of the calibration target points.
    %            K   - is the 3 x 4 camera calibration matrix.
    %  The variable N should be an integer greater than or equal to 6.
    %
    %  This function plots the uv coordinates onto the image of the calibration
    %  target.
    %
    %  It also projects the XYZ coordinates back into image coordinates using
    %  the calibration matrix and plots these points too as
    %  a visual check on the accuracy of the calibration process.
    %
    %  Lines from the origin to the vanishing points in the X, Y and Z
    %  directions are overlaid on the image.
    %
    %  The mean squared error between the positions of the uv coordinates
    %  and the projected XYZ coordinates is also reported.
    %
    %  The function should also report the error in satisfying the
    %  camera calibration matrix constraints.
    %
    % Saswat Panda, 30/05/2021
    '''
```

```
    if (nd != 3):
        raise ValueError('%dD DLT unsupported.' %(nd))

    # Converting all variables to numpy array
    xyz = np.asarray(xyz)
    uv = np.asarray(uv)

    n = xyz.shape[0]

    # Validating the parameters:
    if uv.shape[0] != n:
        raise ValueError('Object (%d points) and image (%d points) have different number of points.' %(n, uv.shape[0]))

    if (xyz.shape[1] != 3):
        raise ValueError('The number of coordinates is incorrect (%d) for %dD DLT (It ought to be %d).' %(xyz.shape[1],nd,nd))

    if (n < 6):
        raise ValueError('%dD DLT requires a minimum of %d calibration points. There were only %d points chosen.' %(nd, 2*nd, n))

    # To improve the DLT quality, normalise the data (Since, The system of coordinates has an impact on DLT.).
    # When there is a significant perspective distortion, this is important.
    # Normalization: mean position at origin and mean distance equals to 1 at each direction.
    Txyz, xyzn = Normalization(nd, xyz)
    Tuv, uvn = Normalization(2, uv)

    A = []

    for i in range(n):
        x, y, z = xyzn[i, 0], xyzn[i, 1], xyzn[i, 2]
        u, v = uvn[i, 0], uvn[i, 1]
        A.append( [x, y, z, 1, 0, 0, 0, 0, -u * x, -u * y, -u * z, -u] )
        A.append( [0, 0, 0, 0, x, y, z, 1, -v * x, -v * y, -v * z, -v] )

    # Converting A to numpy array
    A = np.asarray(A)

    # Find the 11 parameters:
    U, S, V = np.linalg.svd(A)

    # In the last line of Vh the parameters should be normalized
    L = V[-1, :] / V[-1, -1]
    # Camera projection matrix
    H = L.reshape(3, nd + 1)

    # Denormalizing
    # pinv is the Moore-Penrose pseudo-inverse of a matrix, which is the generalised inverse of
    # a matrix, calculated using its SVD.
    H = np.dot( np.dot( np.linalg.pinv(Tuv), H ), Txyz )

    H = H / H[-1, -1]
    L = L.reshape((3,4))


    # The DLT's mean error (In units of camera coordinates, the mean residual of the DLT transformation) is:
    uv2 = np.dot( H, np.concatenate( (xyz.T, np.ones((1, xyz.shape[0]))) ) )
    uv2 = uv2 / uv2[2, :]
    # The mean distance is calculated as below
    err = np.sqrt( np.mean(np.sum( (uv2[0:2, :].T - uv)**2, 1)) )

    return L, err
```

Using the selected co-ordinates and corresponding uv to find Calibration Matrix and Error.

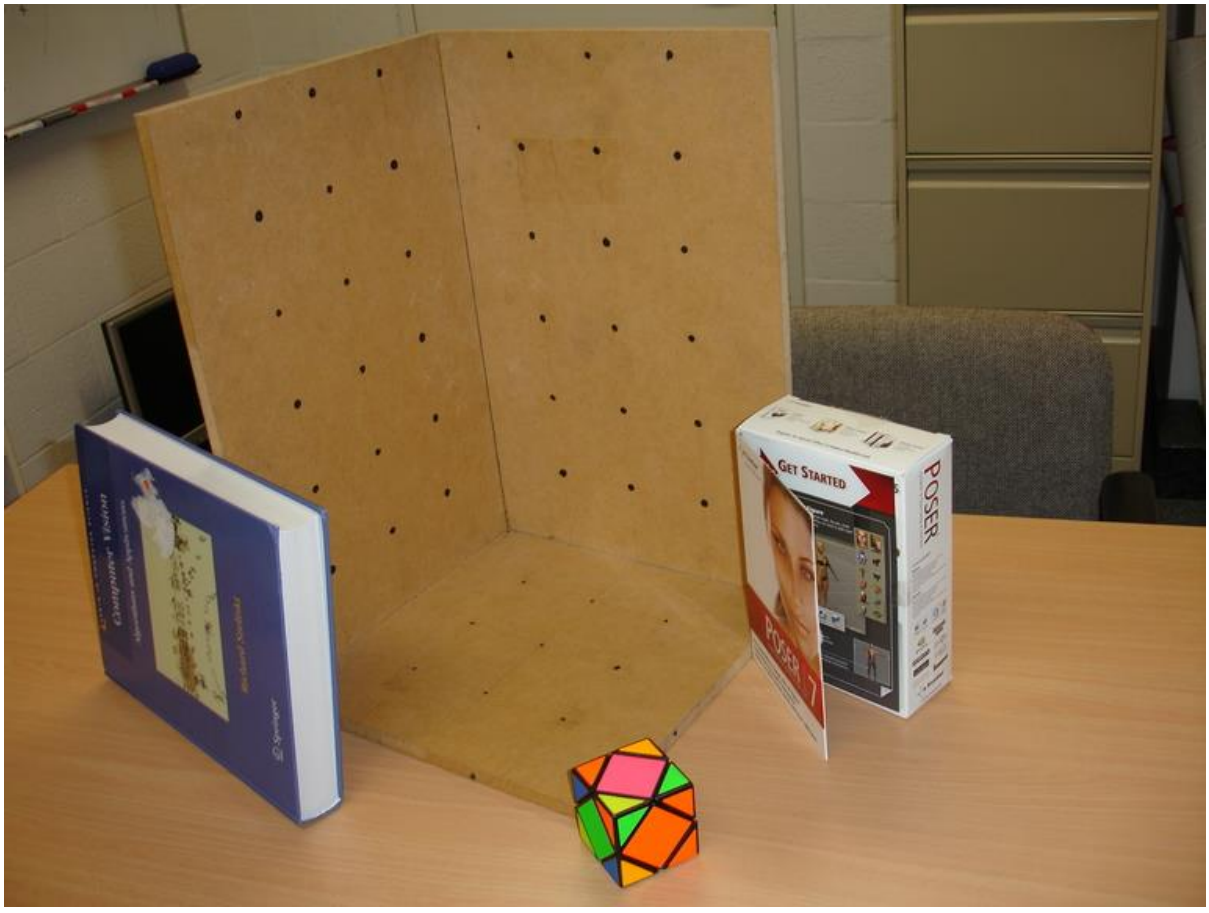
```
# Selected 3D co-ordinates, set of 6
xyz = [[7,7,0], [14,7,0], [14,0,7], [7,0,7], [0,7,7], [0,14,7]]
# uv found using cv2 for the corresponding xyz co-ordinates
uv = [[358, 299], [402, 308], [376, 380], [333, 368], [284, 312], [275, 264]]

nd = 3
C, err = calibrate(nd, xyz, uv)
print('Calibration Matrix(C)')
print(C)
print('\n DLT Mean Error')
print(err)
```

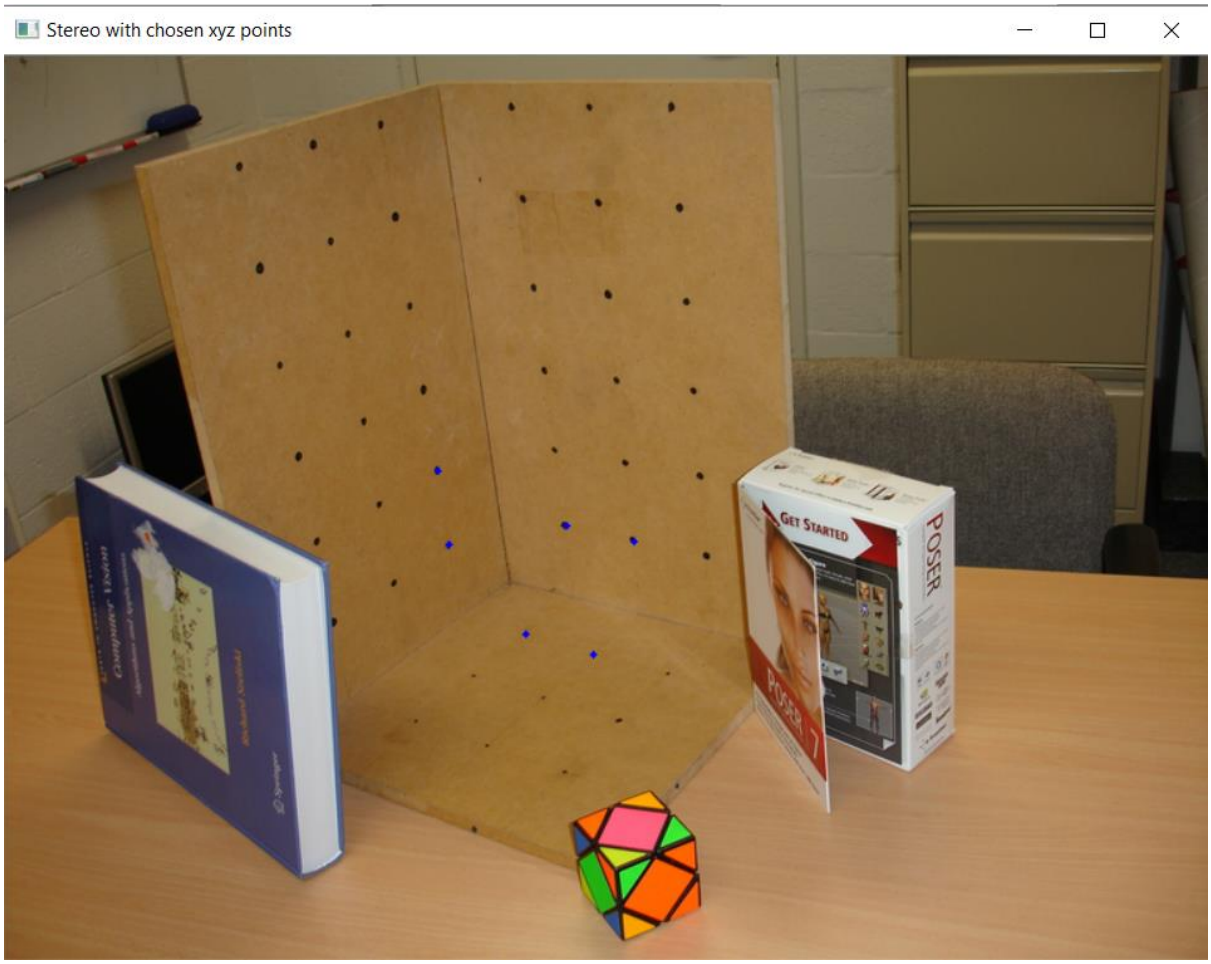
Overall code explanation(*Formulas referred from lecture slides & CLab3 Tutorial*): -

The code started with converting all the **XYZ** and **uv** to numpy array from the list type so that I can get support of the predefined functions in the numpy library. I then improved the quality of xyz and uv by normalizing them. Then I computed A_1 and A_2. Finally the svd of A was computed and the solution for Calibration Matrix is the last column of V generated from the calculation of SVD. Then denormalised the data using Moore-Penrose theorem and finally calculates the mean squared error.

2)

Chosen image: - Stereo2012a.png





3)

```
Calibration Matrix(C)
[[ 0.66089574 -0.11542858 -0.51452179 -0.00853066]
 [ 0.15304782 -0.7352157   0.35986903 -0.00180281]
 [-0.02592132 -0.02144844 -0.02461616  1.        ]]

 DLT Mean Error
0.1028974043451584
```

4)

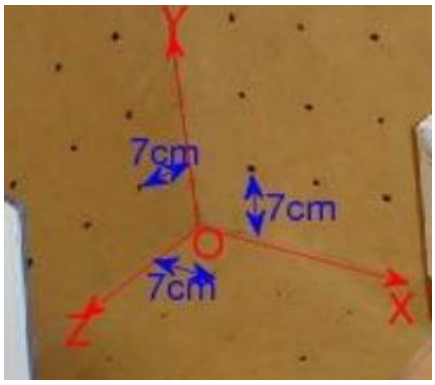
```
K obtained from C
[[20.24836923  0.12209739 -1.14504132]
 [ 0.         19.90373409  1.6936681 ]
 [ 0.          0.          1.        ]]
R obtained from C
[[ 0.74634901 -0.16076031 -0.64584772]
 [ 0.23736035 -0.84228809  0.48395334]
 [-0.62179033 -0.51449673 -0.59048277]]
t obtained from C
[14.38620037 10.83888278 16.0306739 ]
```

While calculating the projections back onto image of the XYZ co-ordinates using the Calibration matrix, it was found that the points were slightly displaced away from the actual points due to the mean error of 0.102. The back projection was calculated using the intrinsic parameters, i.e, K, R and t and also by using the formula x = K[R t] X, referred from lecture slides.

5)

$$K_{\{3\times3\}} = \begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

As theta is -pi/2, alpha is f_x and beta is f_y. So, the focal lengths are f_x = 20.25 and f_y = 19.90 in the units of camera co-ordinates.



A counter-clockwise rotation of beta about the x-axis is known as the pitch [1] for the co-ordinate system used in this assignment. Beta = f_y = 19.90 . The matrix is as follows

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

$$R_x(\alpha) =$$

1	0	0
0	cos 20.25	-sin 20.25
0	-sin 20.25	cos 20.25

$$R_x(\alpha) =$$

1	0	0
0	0.17	-0.985
0	-0.985	0.17

Similarly we can calculate $R_y(\beta)$ for other co-ordinate system

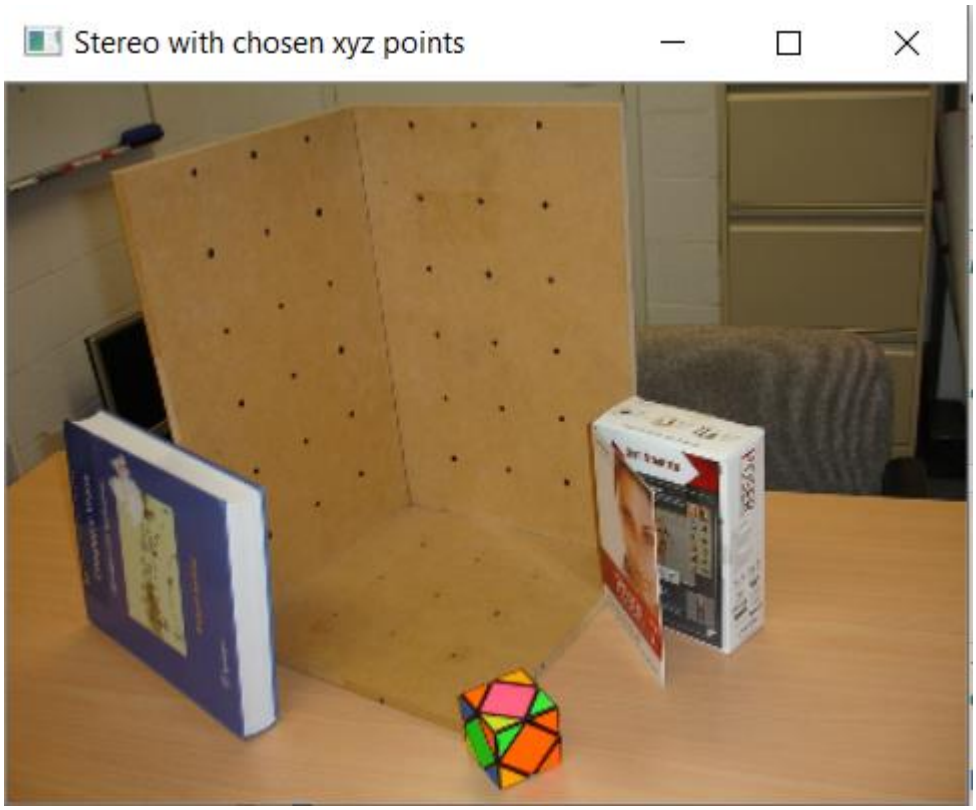
$$R_y(\beta) =$$

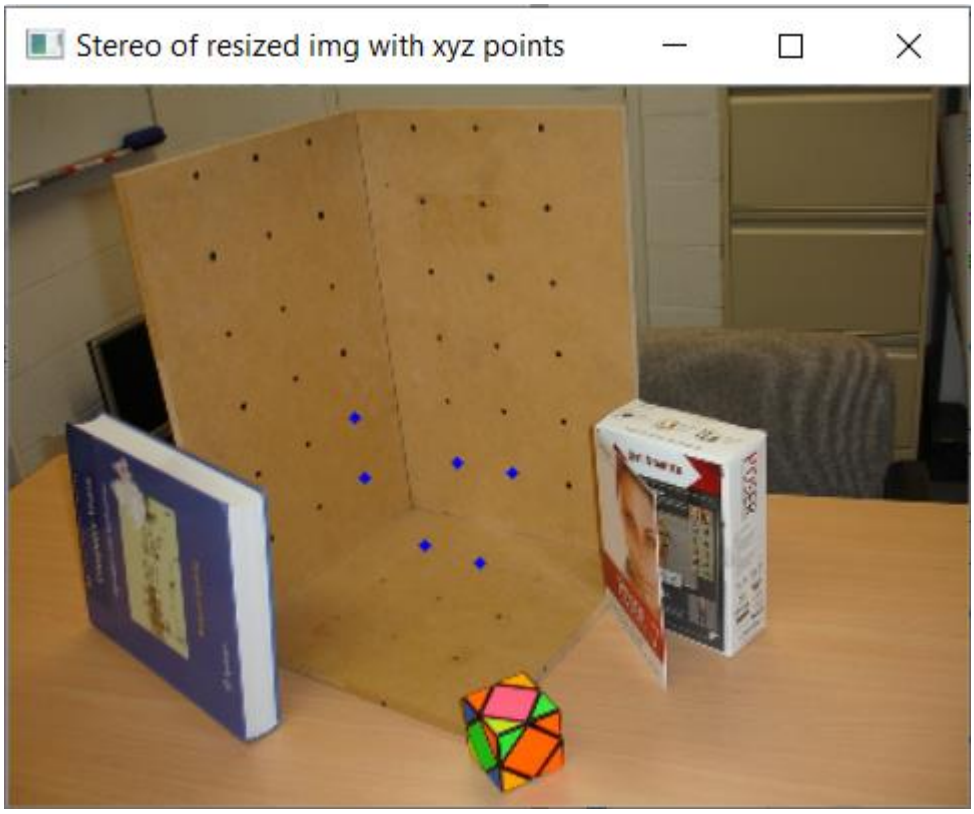
0.497	0	0.867
0	1	0
-0.867	0	0.497

6)

a)

Resized H/2 and W/2





The points obtained are

```
uv_rez = [[179, 149.5], [201, 154], [188, 190], [166.5, 184], [142, 156], [137.5, 132]]
```

```
 Calibration Matrix(C') of resized image
 [[ 0.66089574 -0.11542858 -0.51452179 -0.00853066]
  [ 0.15304782 -0.7352157   0.35986903 -0.00180281]
  [-0.02592132 -0.02144844 -0.02461616  1.        ]]

  DLT Mean Error of resized image
 0.051448702172584924

 K' obtained from C
 [[20.24836923  0.12209739 -1.14504132]
  [ 0.         19.90373409  1.6936681 ]
  [ 0.          0.          1.        ]]
 R' obtained from C
 [[ 0.74634901 -0.16076031 -0.64584772]
  [ 0.23736035 -0.84228809  0.48395334]
  [-0.62179033 -0.51449673 -0.59048277]]
 t' obtained from C
 [14.38620037 10.83888278 16.0306739 ]
```

b) K, R, & t and K', R' & t' are exactly equal. There is no change in the intrinsic parameters because they all depend upon the ratio of the input co-ordinates, and not directly upon the input co-ordinates. Since the input co-ordinates are halved, therefore the ratio of the input co-ordinates does

not change because the half factor cancels out. So, it can be commented that the resized image has same pitch and focal length.

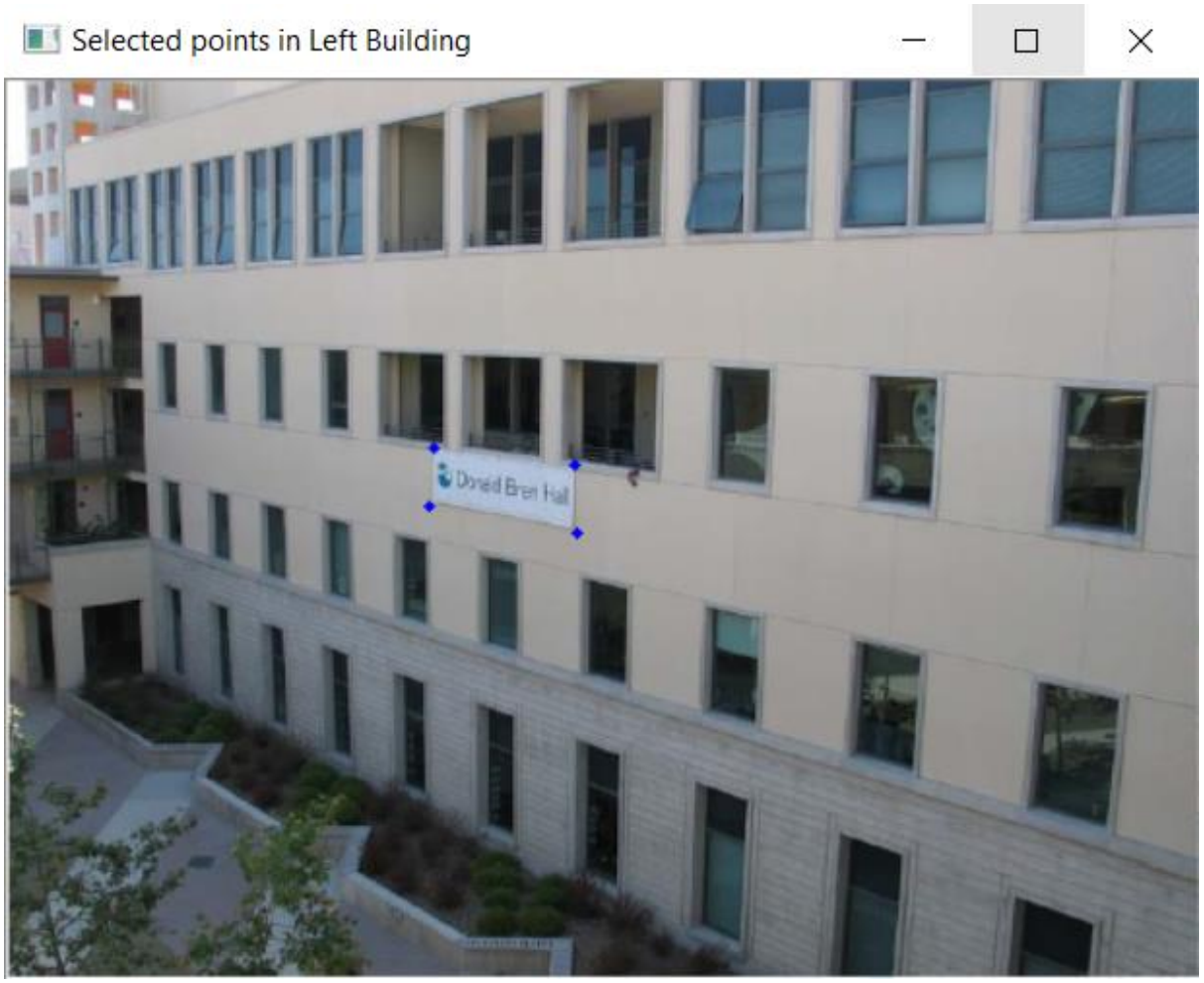
Task 2

1)

```
def homography(u2Trans, v2Trans, uBase, vBase):
    '''
    %% TASK 2:
    % Computes the homography H applying the Direct Linear Transformation
    % The transformation is such that
    % p = np.matmul(H, p.T), i.e.,
    % (uBase, vBase, 1).T = np.matmul(H, (u2Trans , v2Trans, 1).T)
    % Note: we assume (a, b, c) => np.concatenate((a, b, c), axis), be careful when
    % deal the value of axis
    %
    % INPUTS:
    % u2Trans, v2Trans - vectors with coordinates u and v of the transformed image point (p')
    % uBase, vBase - vectors with coordinates u and v of the original base image point p
    %
    % OUTPUT
    % H - a 3x3 Homography matrix
    %
    % Saswat Panda, 30/05/2021
    '''
    global points, points1
    points = np.column_stack((points,np.ones(points.shape[0]))).T
    points1 = np.column_stack((points1,np.ones(points1.shape[0]))).T

    p1 = points[:-1,:].T
    p2 = points1[:-1,:].T



    # Computing the 0th row of A
    A_up = np.column_stack((p1,np.ones(p1.shape[0]),np.zeros((p1.shape[0],3)),-p1[:,0]*p2[:,0],-p1[:,1]*p2[:,0],-p2[:,0]))
    # Computing the 1st row of A
    A_below = np.column_stack((np.zeros((p1.shape[0],3)),p1,np.ones(p1.shape[0]),-p1[:,0]*p2[:,1],-p1[:,1]*p2[:,1],-p2[:,1]))

    # Stacking oth and 1st rows
    A = np.vstack((A_up,A_below))

    # Computing the SVD
    result = np.linalg.svd(A)[-1][-1]
    # The parameters should be normalised for the last line too.
    result = result/result[-1]

    # Reshaping the result
    result = result.reshape((p1.shape[1]+1,-1))

    H = result

    return H


# horizontally splitting the points and points1 for converting to desired input
H = homography(np.hsplit(points1, 2)[0],np.hsplit(points1, 2)[1],np.hsplit(points, 2)[0],np.hsplit(points, 2)[1])
print("The homography matrix(H) calculated using the DLT algorithm is:")
print(H)
```

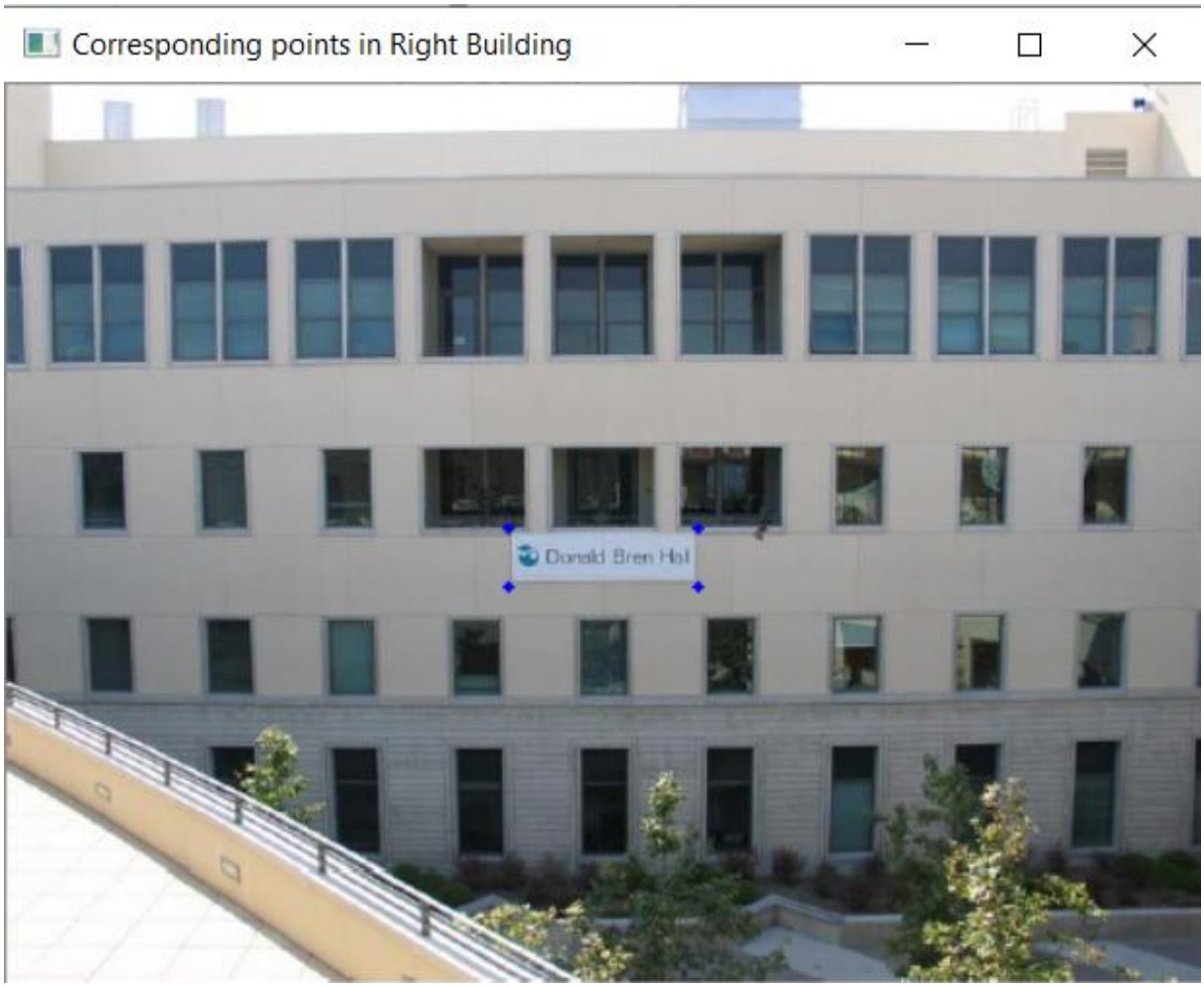
Overall Code Explanation (Formulas referred from **CMU** slides [2]): -

The transpose of the four points from *Left* and *Right* image are taken so that the points can be compatible for calculating A. Then the two rows of A were computed using the formulas, A_1 and A_2. Then we compute the SVD and store the eigen vector of the smallest eigen value. We reshape the Homography matrix to 3X3 and finally return it.




```
print(points)
```

```
[[175, 152], [177, 175], [235, 185], [234, 159]]
```



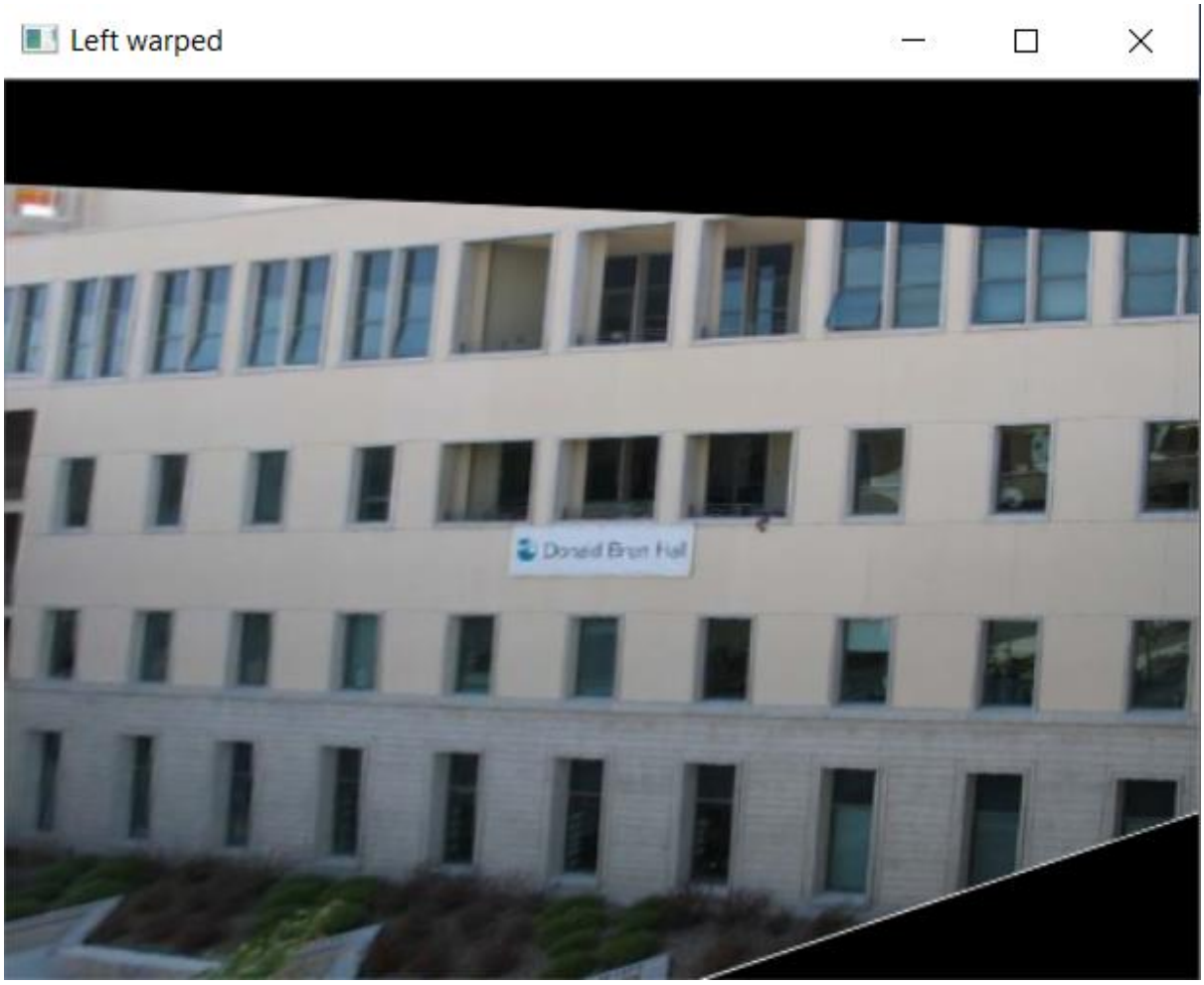

```
print(points1)
```

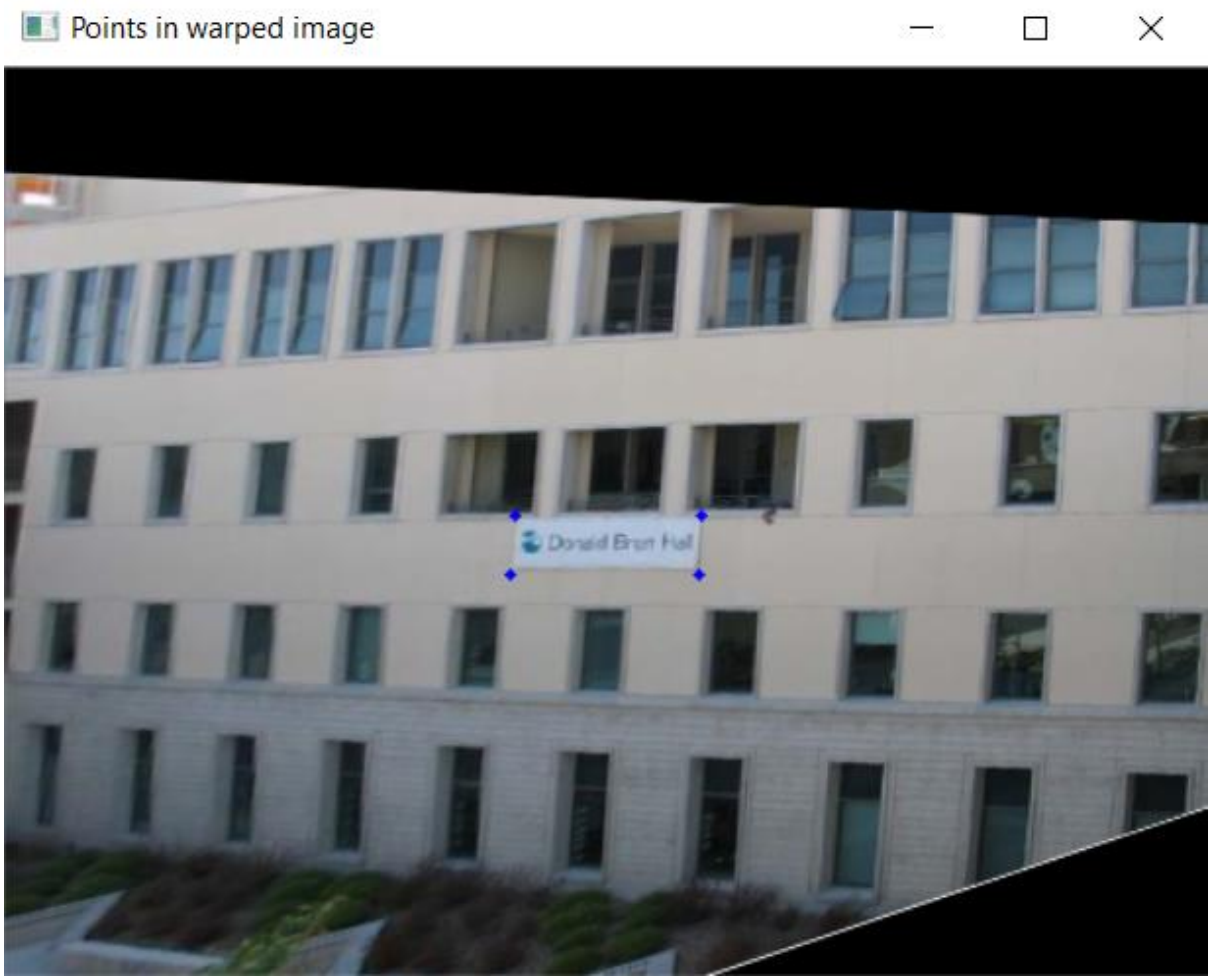
```
[[208, 184], [209, 205], [284, 204], [284, 182]]
```

2)

```
The homography matrix(H) calculated using the DLT algorithm is:
[[ 2.20267020e+00 -2.04532968e-01 -9.03400798e+01]
 [ 1.77345385e-01  1.07672588e+00  3.88744727e+01]
 [ 1.97181842e-03 -4.97722561e-04  1.00000000e+00]]
```

3)


Left warped
Donald Bren Hall



Now let's find the distance between the projected points (in the warped image) and corresponding targets (in the right image).

```
The distance between invidual corresponding points are  [0.0, 2.0, 1.0, 1.0]
The mean of distances  1.0
```

The mean of distances is simply the arithmetic mean of all the distances found between every corresponding point of the warped image and the right image.

By observing the distance between the corresponding points and the mean distance, the error was found pretty less that may be a result of the human error while choosing the points using ginput(I have used cv2 though).

Important Results and matrix for Task 1 and 2.

Task 1

Calibration matrix and intrinsic parameters

Selected 3D co-ordinates, set of 6

xyz = [[7,7,0], [14,7,0], [14,0,7], [7,0,7], [0,7,7], [0,14,7]]

uv found using cv2 for the corresponding xyz co-ordinates

uv = [[358, 299], [402, 308], [376, 380], [333, 368], [284, 312], [275, 264]]

Calibration Matrix(C)
array([[0.66089574, -0.11542858, -0.51452179, -0.00853066],
[0.15304782, -0.7352157 , 0.35986903, -0.00180281],
[-0.02592132, -0.02144844, -0.02461616, 1.]])

K obtained from C
array([[20.24836923, 0.12209739, -1.14504132],
[0. , 19.90373409, 1.6936681],
[0. , 0. , 1.]])
R obtained from C
array([[0.74634901, -0.16076031, -0.64584772],
[0.23736035, -0.84228809, 0.48395334],
[-0.62179033, -0.51449673, -0.59048277]])
t obtained from C
array([14.38620037, 10.83888278, 16.0306739])

Calibration matrix and intrinsic parameters for resized image

xyz = [[7,7,0], [14,7,0], [14,0,7], [7,0,7], [0,7,7], [0,14,7]]

uv_rez = [[179, 149.5], [201, 154], [188, 190], [166.5, 184], [142, 156], [137.5, 132]]

Calibration Matrix(C) of resized image
array([[0.66089574, -0.11542858, -0.51452179, -0.00853066],
[0.15304782, -0.7352157 , 0.35986903, -0.00180281],
[-0.02592132, -0.02144844, -0.02461616, 1.]])

K resized image

array([[20.24836923, 0.12209739, -1.14504132],
[0. , 19.90373409, 1.6936681],
[0. , 0. , 1.]])

R resized image

array([[0.74634901, -0.16076031, -0.64584772],
[0.23736035, -0.84228809, 0.48395334],
[-0.62179033, -0.51449673, -0.59048277]])

t resized image

array([14.38620037, 10.83888278, 16.0306739])

Task 2

points = np.array([[175, 152], [177, 175], [235, 185], [234, 159]])

points1 = np.array([[208, 184], [209, 205], [284, 204], [284, 182]])

The homography matrix(H) calculated using the DLT algorithm is:

```
array([[ 2.20267020e+00, -2.04532968e-01, -9.03400798e+01],
       [ 1.77345385e-01,  1.07672588e+00,  3.88744727e+01],
       [ 1.97181842e-03, -4.97722561e-04,  1.00000000e+00]])
```

points_w = [[208, 184], [207, 205], [284, 205], [285, 182]]

References

[1] https://piazza.com/class/kla8ncbl4rj3fd?cid=529

[2] http://www.cs.cmu.edu/~16385/s17/Slides/10.2_2D_Alignment__DLT.pdf