COMP/ENGN6528 Computer Vision - 2023 S1
Computer Lab 3 (CLab-3)

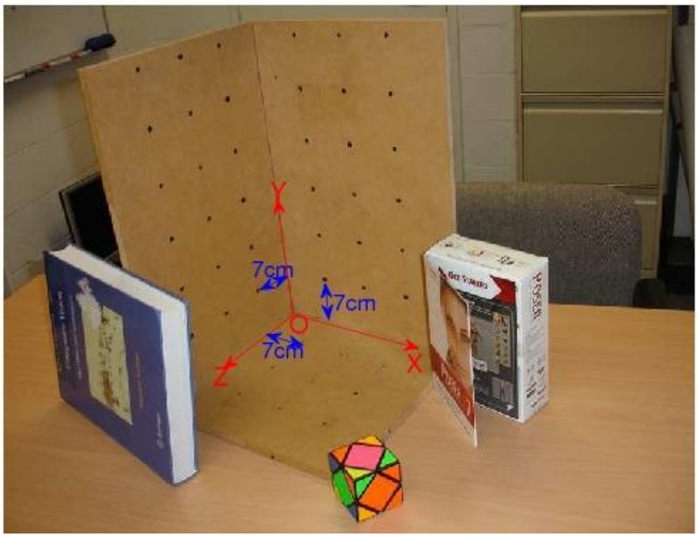
Yifan Zhu
u7560434

15/05/2023

Task-1: 3D-2D Camera Calibration (17 marks)

Camera calibration involves finding the geometric relationship between 3D world coordinates and their 2D projected positions in the image.

Four images, `stereo2012a.jpg`, `stereo2012b.jpg`, `stereo2012c.jpg`, and `stereo2012d.jpg`, are given for this CLab-3. These images are different views of a calibration target and some objects. For example, the diagram below is **stereo2012a.jpg** with some text superimposed onto it:



(Do not directly use the above image for your camera calibration work as it has been scaled for illustration. Use the original (unlabelled) image files provided.)

On the calibration target there are 3 mutually orthogonal faces. The points marked on each face form a regular grid. They are all 7cm apart.

Write a Matlab function with the following specification

Function to perform camera calibration		
Function C = calibrate(im, XYZ, uv)		
Input:	im:	is the image of the calibration target.
	XYZ:	is a Nx3 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.
Outputs:	C:	is the 3 x 4 camera calibration matrix.

The variable N should be an integer greater than or equal to 6.

This function should also plot 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.

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

Lines from the origin to the vanishing points (namely, world coordinate system) in the X, Y and Z directions are overlaid on the image.

Generally, we ask you to implement a function:

MATLAB user:

```
function C = calibrate(im, XYZ, uv)
```

Python user:

```
def calibrate(im, XYZ, uv)
    return C
```

For Task-1, you should include the following in your Lab-Report PDF file:

1. **List `calibrate` function in your PDF file. [3 marks]**

Following Figure 1.1 shows calibrate functions, as well as plot points and lines functions.

```
# calculate line intersection
def line_intersection(line1, line2):
    x1, y1, x2, y2 = line1[0][0], line1[0][1], line1[1][0],
line1[1][1]
    x3, y3, x4, y4 = line2[0][0], line2[0][1], line2[1][0],
line2[1][1]
    # Calculate the slope of line1
    k1 = (y2 - y1) / (x2 - x1)
    # Calculate the slope of line2
    k2 = (y4 - y3) / (x4 - x3)
    # Calculate the intercepts
    b1 = y1 - k1 * x1
    b2 = y3 - k2 * x3
    # Calculate the x-coordinate of the intersection point
    x = (b2 - b1) / (k1 - k2)
    # Calculate the y-coordinate of the intersection point
    y = k1 * x + b1
    return x, y

# draw lines from original to vanishing points
def drawLines(im):
    try:
        print(im.filename)
        origin = np.load('origin.npy')
        x_line1 = np.load('x_line1.npy')
        x_line2 = np.load('x_line2.npy')
        x_v = line_intersection(x_line1, x_line2)
        x = np.array((origin[0], x_v[0]))
        y = np.array((origin[1], x_v[1]))
        plt.plot(x, y, color='g')
```

```
        y_line1 = np.load('y_line1.npy')
        y_line2 = np.load('y_line2.npy')
        y_v = line_intersection(y_line1, y_line2)
        x = np.array((origin[0], y_v[0]))
        y = np.array((origin[1], y_v[1]))
        plt.plot(x, y, color='g')

        z_line1 = np.load('z_line1.npy')
        z_line2 = np.load('z_line2.npy')
        z_v = line_intersection(z_line1, z_line2)
        x = np.array((origin[0], z_v[0]))
        y = np.array((origin[1], z_v[1]))
        plt.plot(x, y, color='g')
        plt.xlim((0, im.size[0]))
        plt.ylim((im.size[1], 0))
    except:
        origin = np.load('origin_resize.npy')
        x_line1 = np.load('x_line1_resize.npy')
        x_line2 = np.load('x_line2_resize.npy')
        x_v = line_intersection(x_line1, x_line2)
        x = np.array((origin[0], x_v[0]))
        y = np.array((origin[1], x_v[1]))
        plt.plot(x, y, color='g')

        y_line1 = np.load('y_line1_resize.npy')
        y_line2 = np.load('y_line2_resize.npy')
        y_v = line_intersection(y_line1, y_line2)
        x = np.array((origin[0], y_v[0]))
        y = np.array((origin[1], y_v[1]))
        plt.plot(x, y, color='g')

        z_line1 = np.load('z_line1_resize.npy')
        z_line2 = np.load('z_line2_resize.npy')
        z_v = line_intersection(z_line1, z_line2)
        x = np.array((origin[0], z_v[0]))
        y = np.array((origin[1], z_v[1]))
        plt.plot(x, y, color='g')
        plt.xlim((0, im.size[0]))
        plt.ylim((im.size[1], 0))

# Draw points in the picture
def visualize_projection(im, uv):
    plt.close()
    uv_o = uv
    uv_p = np.load('uv_projection.npy')
    for i in range(uv_o.shape[0]):
        plt.plot(uv_o[i, 0], uv_o[i, 1], marker="x", color="r",
markersize=10)
        plt.plot(uv_p[i, 0], uv_p[i, 1], marker="o", mfc="none",
color="b", markersize=10)
    plt.imshow(im)
    plt.title("Visualise the projection and original points")
    drawLines(im)
    plt.show()

def calibrate(im, XYZ, uv):
    # If uv size is less than 6, then raise error
    if np.shape(uv)[0] <= 5:
        raise Exception("uv length should be greater then 5")

    "Calculate calibration matrix P"
```

```
    # size of XYZ
    len_XYZ = np.shape(XYZ)[0]

    # initiate matrix A, size 2n*12
    A = np.zeros((2*len_XYZ, 12))

    # Assemble 2n*12 matrix A
    for i in range(len_XYZ):
        A[2*i] = [XYZ[i][0], XYZ[i][1], XYZ[i][2], 1, 0, 0, 0, 0, -
1*uv[i][0]*XYZ[i][0], -1*uv[i][0]*XYZ[i][1], -1*uv[i][0]*XYZ[i][2], -
1*uv[i][0]]
        A[2*i+1] = [0, 0, 0, 0, XYZ[i][0], XYZ[i][1], XYZ[i][2], 1, -
1*uv[i][1]*XYZ[i][0], -1*uv[i][1]*XYZ[i][1], -1*uv[i][1]*XYZ[i][2], -
1*uv[i][1]]

    # Compute the SVD of A
    U, S, V = np.linalg.svd(A)
    # The solution is the last column of V
    C = V.T[:, -1]
    # Reshape C to 3*4
    C = C.reshape(3, 4)
    C = C / C[2, 3]

    "the projection of the XYZ coordinates back onto image"
    uv_projection_H = np.zeros((np.shape(uv)[0], np.shape(uv)[1]+1))
    uv_projection = np.zeros((np.shape(uv)[0], np.shape(uv)[1]))
    # transfer XYZ into homogeneous vector
    XYZ_H = np.zeros((np.shape(XYZ)[0], np.shape(XYZ)[1] +1))

    for i in range(len_XYZ):
        XYZ_H[i] = np.append(XYZ[i], 1)
        uv_projection_H[i] = np.transpose(C @ np.transpose(XYZ_H[i]))
        # Dehomogeneous for uv
        uv_projection[i] = uv_projection_H[i][:-1] /
uv_projection_H[i][-1]

    "Calculate reprojection error"
    projection_error = np.mean(np.square(uv - uv_projection))
    np.save('uv_projection',uv_projection)
    print("The mean squared error is: "+str(projection_error))

    # Draw points in the picture
    visualize_projection(im, uv)
    return C
```

Figure 1.1 Calibrate function and its following drawing functions

2. **List the image you have chosen for your experiment, and display the image in your PDF file. [0.5 mark]**

I have chosen ‘stereo2012a.jpg’, and result image is shown in Figure 1.2.

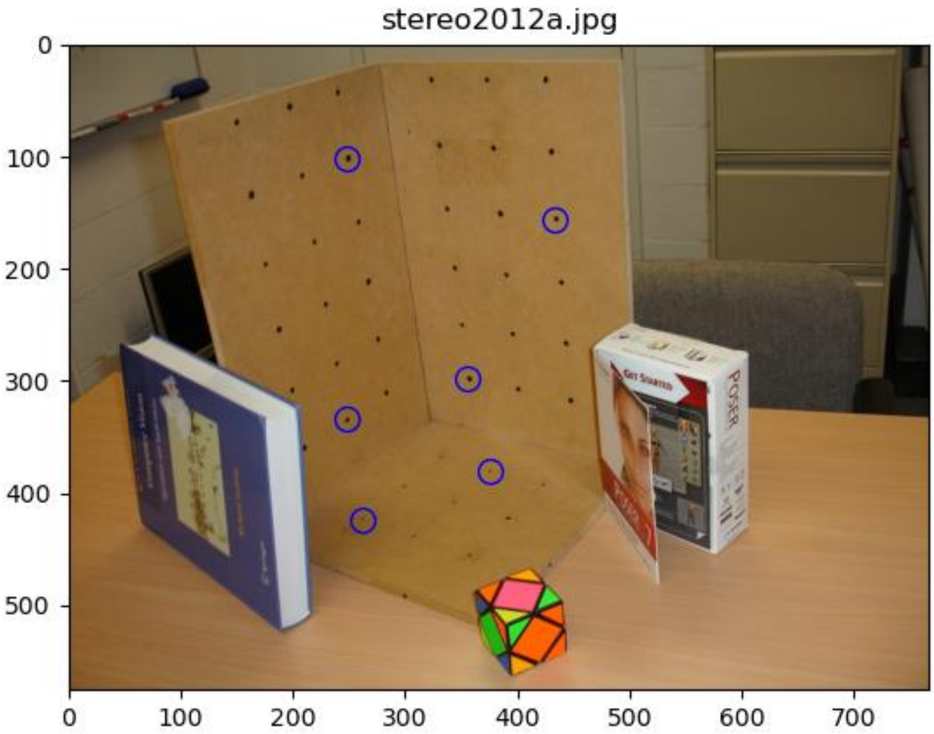


Figure 1.2 Original image ‘stereo2012a.jpg’. Blue circles are selected points

3. **List the 3x4 camera calibration matrix P that you have calculated for the selected image. Please visualise the projection of the XYZ coordinates back onto image using the calibration matrix P and report the reprojection error (The mean squared error between the positions of the uv coordinates and the projected XYZ coordinates using the estimated projection matrix)[2 marks]**

Calibration matrix P is shown in Figure 1.3.

```
P =  [[ 4.23901178e+00 -1.94334431e+00 -6.25973799e+00  3.21385434e+02]
 [-3.90616926e-02 -7.32819630e+00  1.03266138e+00  3.34258365e+02]
 [-4.21930743e-03 -3.20872927e-03 -6.25391645e-03  1.00000000e+00]]
```

Figure 1.3 Calibration matrix P

Visualisation of the projection of XYZ coordinate after using calibration matrix P as well as lines from the original point to vanishing point in each direction is shown in Figure 1.4.

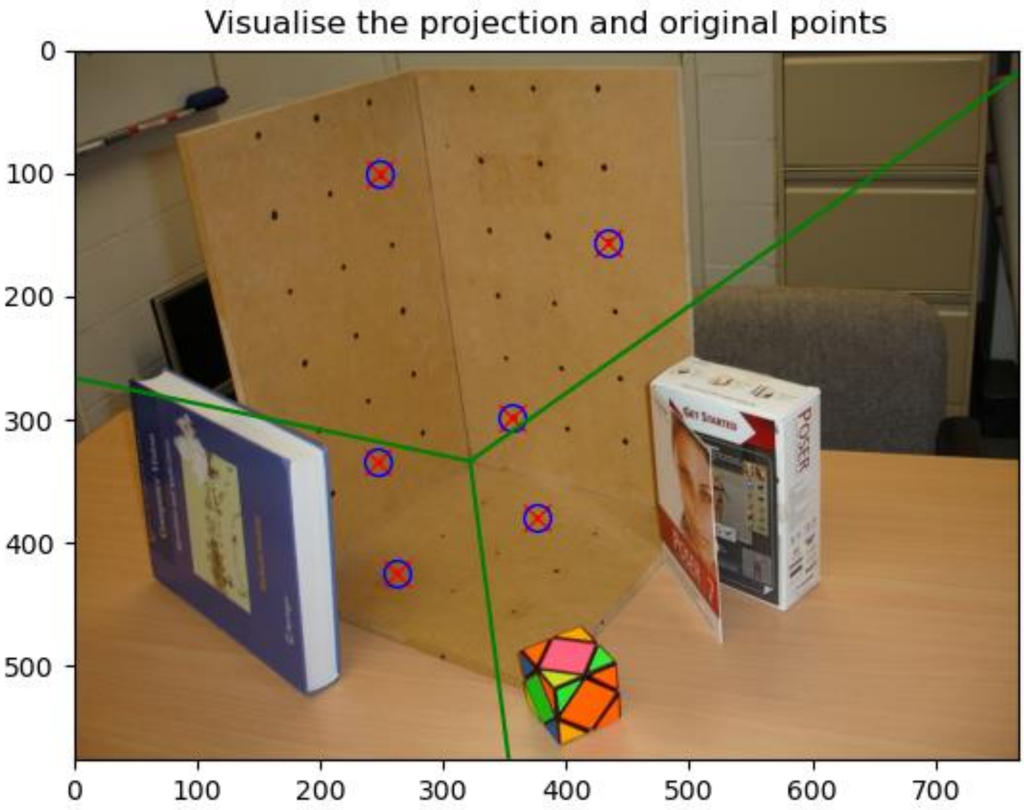


Figure 1.4 Visualisation of the projection and original points. In this image, the red Xs are the originally chosen points, and the blue circles are the projection of the calibration target. Green lines are connections between the origin of the world coordinate and the vanishing point in XYZ directions

The MSE between original points on the image and projection from world coordinate is shown in Figure 1.5.

```
The mean squared error is: 0.004858249002383643
```

Figure 1.5 The MSE between original points on the image and projection from the world coordinate

4. **Decompose the P matrix into K, R, t, such that P = K[R|t], by using the following provided code (`vgg_KR_from_P.m` or `vgg_KR_from_P.py`). List the results, namely the K, R, t matrices, in your PDF file. [1.5 marks]**

After decomposing matrix P using vgg_KR_from_P.py, we can get matrix K, R, t as shown is Figure 1.6.

```
K =  [[859.69705023   9.72577749 409.13350252]
 [  0.         865.6000953  256.22438647]
 [  0.           0.           1.        ]]
R =  [[ 0.84472378 -0.07909325 -0.52932599]
 [ 0.14684026 -0.91681707  0.37132788]
 [-0.51466463 -0.39139586 -0.76284311]]
t =  [71.78532867 56.82896649 82.31111895]
```

Figure 1.6 Matrix K, R, t

5. **Please answer the following questions:**

- what is the focal length (in the unit of pixel) of the camera? [1 mark]

According to the definition of matrix K shown in Figure 1.6, the focal length in the x direction is 859.70, while in the y direction is 865.60.

- What is the pitch angle of the camera with respect to the X-Z plane in the world coordinate system? (Assuming the X-Z plane is the ground plane, then the pitch angle is the angle between the camera's optical axis and the ground-plane.) Please provide the calculation process [2 marks]

Extract the elements R. Then calculate the pitch angle theta using the atan2() function then transfer it into degrees, as code shown in Figure 1.7 and result in Figure 1.8.

```
# Calculate pitch angle
theta = math.degrees(math.atan2(R[2, 0], np.sqrt(R[0, 0]**2 + R[1,
0]**2)))
print('pitch angle of the camera: ' + str(theta))
```

Figure 1.7 Code to calculate pitch angle

```
pitch angle of the camera: -30.975040845616828
```

Figure 1.8 Pitch angle of the camera

- What is the camera centre coordinate in the XYZ coordinate system (world coordinate system)? Please provide the calculation process [1 mark]

To calculate the camera centre, we need to multiply R and t, as code shown in Figure 1.9 and the result shown in Figure 1.10.

```
# Calculate camera centre
camera_centre = -R.T @ t
print('Camera centre: ', camera_centre)
```

Figure 1.9 Code to calculate camera centre

```
Camera centre:  [-26.62093283  89.99573238  79.6861295 ]
```

Figure 1.10 Camera centre coordinate

6. **Please resize your selected image using builtin function from matlab or python to (H/3, W/3) where H, and W denote the original size of your selected image. Using the interface function, (ginput in Matlab, and `matplotlib.pyplot.ginput in Python`) to find the uv coordinates in the resized image.** **[1 mark]**

The code for resizing is shown in Figure 1.11. And result image is shown in Figure 1.12.

```
# resize image to (H/3, W/3)
I_resize = I.resize((int(I.size[0]/3), int(I.size[1]/3)))
# get point actions. Already get suitable points and save as *.npy
# plt.imshow(I_resize)
# uv_p = plt.ginput(6, timeout=30000) # Graphical user interface to
get 6 points
# uv_p = np.array(uv_p)
# np.save('uv_p', uv_p)
plt.imshow(I_resize)
plt.title('Resized stereo2012a.jpg')
uv_p = np.load('uv_p.npy')
for i in range(uv_p.shape[0]):
    plt.plot(uv_p[i, 0], uv_p[i, 1], marker="o", mfc="none",
color="b", markersize=10)
plt.show()
```

Figure 1.11 Code for resizing image into (H/3, W/3)

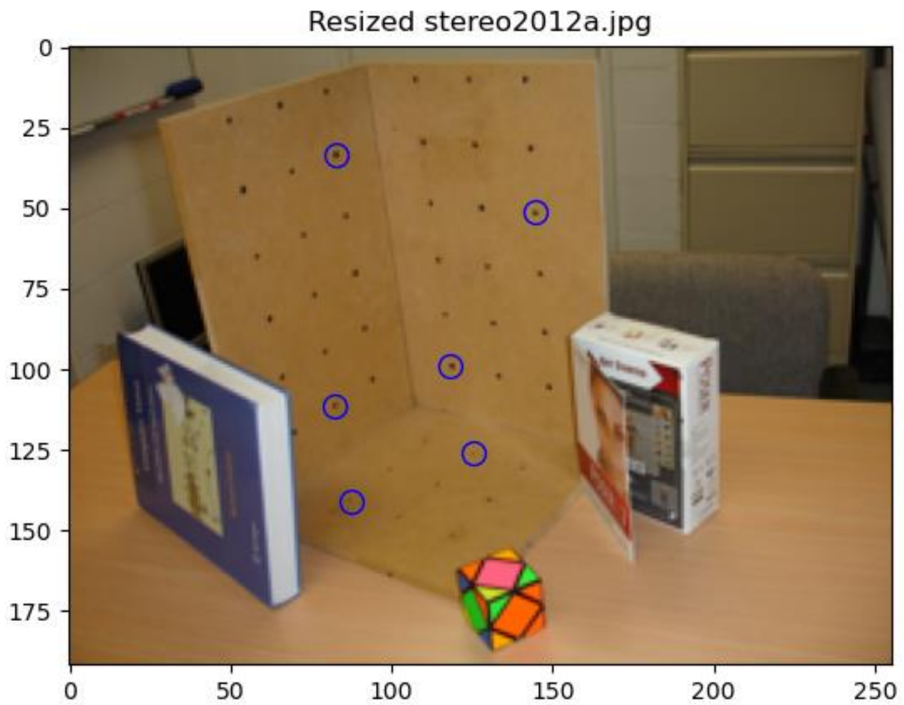


Figure 1.12 Result after resizing, with blue circles referring to uv coordinates

a. **Please display your resized image in the report, list your calculated 3x4 camera calibration matrix P' and the decomposed K', R', t' in your PDF file. [2 marks]**

After resizing, we draw uv coordinates and also the projection of XYZ coordinates, as well as lines linking the origin to the vanishing point in the image. Image is shown in Figure 1.13.

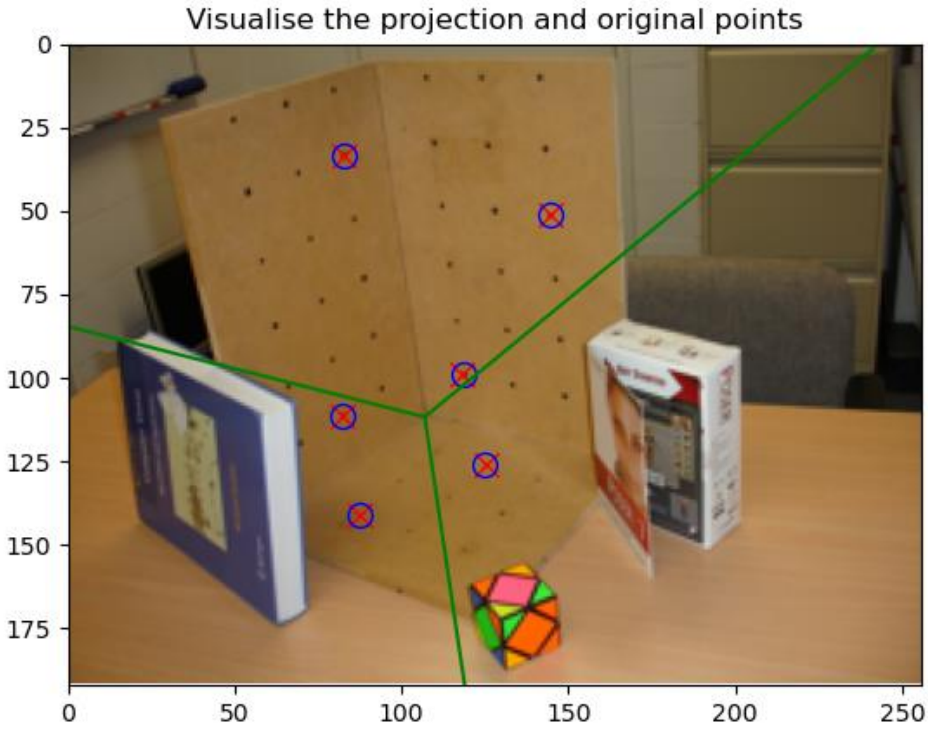


Figure 1.13 After resizing the image. In this image, the red Xs are the originally chosen points, and the blue circles are the projection of the calibration target. Green lines are connections between the origin of the world coordinate and the vanishing point in XYZ directions

In Figure 1.14, here shows the calibration matrix P', the intrinsic matrix K', the rotation matrix R' and the translation vector t'.

```
The mean squared error is: 0.004858249002383643
P' =  [[ 1.41904905e+00 -6.39961860e-01 -2.05704285e+00  1.06816367e+02]
 [-3.92268281e-02 -2.44463554e+00  3.68672436e-01  1.11158935e+02]
 [-4.27744726e-03 -3.15049352e-03 -6.11957234e-03  1.00000000e+00]]
K' =  [[290.59044904   1.08745886 129.95793932]
 [  0.         292.89737886  85.47841905]
 [  0.           0.           1.        ]]
R' =  [[ 0.83814179 -0.09446488 -0.53721014]
 [ 0.13751505 -0.91648006  0.37570455]
 [-0.52783327 -0.38876816 -0.7551499 ]]
t' =  [71.9167494  56.95721487 83.8190207 ]
```

Figure 1.14 Matrix P', K', R' and t'

b. **Please analyse the differences between 1) K and K’, 2) R and R’, 3) t and t’. Please provide the reasoning when changes happened or there are no changes. .[2 marks]**

Following Figure 1.15 shows two images which are K, R and t matrices and corresponding K’, R’ and t’ matrices before or after changes happen.

```
K =  [[859.69705023   9.72577749 409.13350252]
 [  0.         865.6000953  256.22438647]
 [  0.           0.           1.        ]]
R =  [[ 0.84472378 -0.07909325 -0.52932599]
 [ 0.14684026 -0.91681707  0.37132788]
 [-0.51466463 -0.39139586 -0.76284311]]
t =  [71.78532867 56.82896649 82.31111895]
```

```
K' =  [[290.59044904   1.08745886 129.95793932]
 [  0.         292.89737886  85.47841905]
 [  0.           0.           1.        ]]
R' =  [[ 0.83814179 -0.09446488 -0.53721014]
 [ 0.13751505 -0.91648006  0.37570455]
 [-0.52783327 -0.38876816 -0.7551499 ]]
t' =  [71.9167494  56.95721487 83.8190207 ]
```

Figure 1.15 Matrices K, R, t and K’, R’ and t’

Firstly, we compare matrix K. As we can see in the Figure 1.15, the focal length for both x and y directions are changed. After resizing the original image to a one-third size, it seems that the focal length is also the one-third of original one. Besides, the value of principal points are one-third of the original ones. Furthermore, the skew parameter is one-ninth of the original one. Changes can be shown as follow.

$$K = \begin{bmatrix} f_x & s & p_x \\ & f_y & p_y \\ & & 1 \end{bmatrix}, K' = \begin{bmatrix} \frac{f_x}{3} & \frac{s}{9} & \frac{p_x}{3} \\ & \frac{f_y}{3} & \frac{p_y}{3} \\ & & 1 \end{bmatrix}$$

The focal length, main point, and skew are a few of the intrinsic camera properties represented by the camera intrinsic matrix K. Due to changes in image size and resolution, inherent parameters may change when an image is resized. Resampling the pixels when resizing an image might impact how crisp the image is and distort the original pixel grid. As a result, while resizing the image, the focal length and principal point estimations within the camera matrix K may change.

For matrix R and t, it seems unchanged after changing.

The rotation matrix describing the camera's orientation in the world coordinate system is represented by the camera extrinsic matrix R. Because it represents the camera's intrinsic orientation and is unaffected by image size, the rotation matrix R should not change when the image is resized.

The camera extrinsic matrix t represents the translation vector that describes the camera's position in the world coordinate system. As mentioned earlier, resizing an image does not alter the camera's position or translation. Thus, t' should remain the same.

c. **Let us check the focal length (f and f') (in pixel unit) and the principal points extracted from K and K', respectively. Please discuss their relationship between (f and f') and its connection to the image size of the original image and the one after resizing.[2 marks]**

As we discuss in the last section, we define the following, where s_{resized} refers to the size of the resized image and s_{original} refers to the size of the original image.

$$r = \frac{s_{resized}}{s_{original}}$$

In this case, r equals to 1/3. So the relationship between f, f', principle point and size of original image are shown as below.

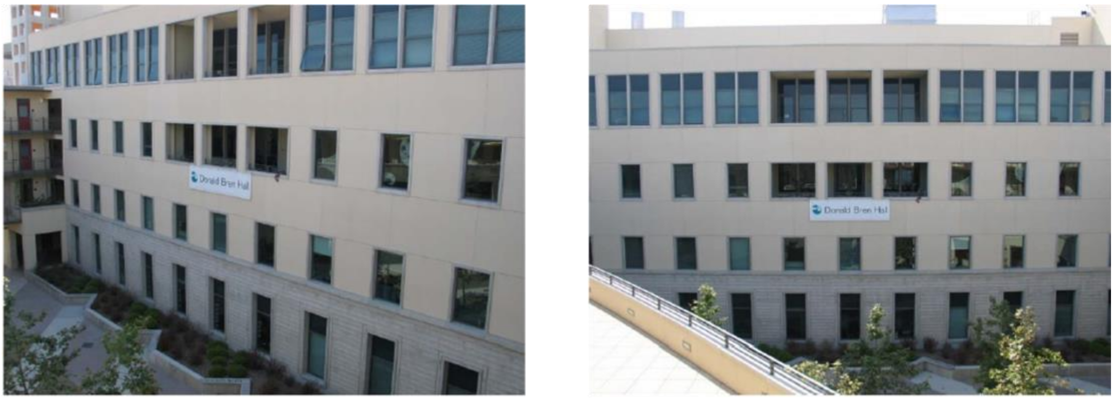
$$\begin{cases} f' = f \times r \\ p'_x = p_x \times r \\ p'_y = p_y \times r \end{cases}$$

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

A transformation from the projective space P^3 to itself is called homography. A homography is represented by a 3x3 matrix with 8 degree of freedom (scale, as usual, does not matter)

$$\begin{bmatrix} x^C w \\ y^C w \\ w \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x^R \\ y^R \\ 1 \end{bmatrix}$$

The goal of this task is to the DLT algorithm to estimate a 3x3 homography matrix.



(a) Left (b) Right

Pick any **6** corresponding coplanar points in the images `left.jpg` and `right.jpg` and get their image coordinates.

In doing this step you may find it useful to check the `Matlab` function `ginput`.

Calculate the 3x3 homography matrix between the two images, from the above 6 pairs of corresponding points, using DLT algorithm. You are required to implement your function in the following syntax.

Function to calculate homography matrix		
H = homography (u2Trans, v2Trans, uBase, vBase)		
Usage:	Computes the homography H applying the Direct Linear Transformation	
Inputs:	u2Trans, v2Trans:	are vectors with coordinates u and v of the transformed image point (p')
	uBase, vBase:	are vectors with coordinates u and v of the original base image point p
Output:	H:	is a 3x3 Homography matrix

In doing this lab task, you should include the following in your lab report:

1. **List your source code for homography estimation function and display the two images and the location of six pairs of selected points (namely, plotted those points on images). Explain the steps about what you have done for the homography and what is shown in the images.** [5 marks]

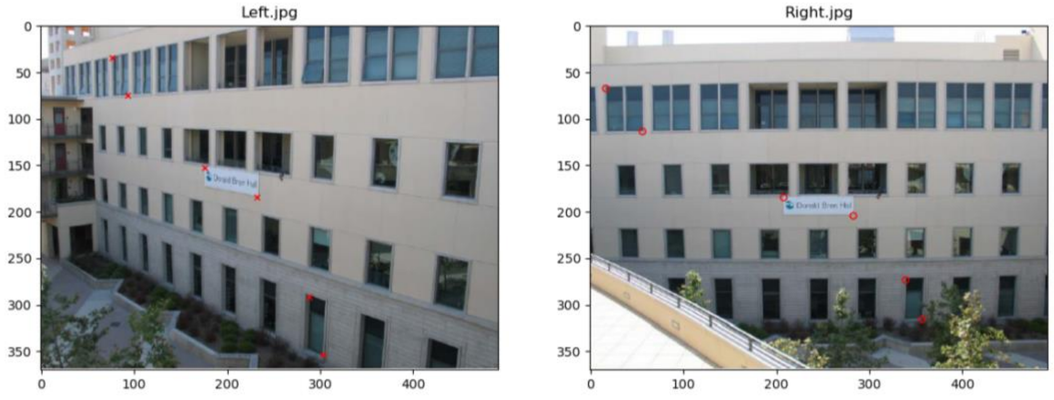
Below Figure 2.1 is homography estimation function homography().

```
def homography(u2Trans, v2Trans, uBase, vBase):
    if uBase.shape != vBase.shape or u2Trans.shape != v2Trans.shape
or uBase.shape != u2Trans.shape:
        raise Exception('Points are inconsistent')
    if uBase.shape[0] < 4:
        raise Exception('Should give greater than or equal to 4
points')

    "Calculate the matrix H"
    points = uBase.shape[0]
    # initiate matrix A, size 2n*12
    A = np.zeros((2 * points, 9))

    # Assemble 2n*12 matrix A
    for i in range(points):
        A[2 * i] = [uBase[i], vBase[i], 1, 0, 0, 0, -
1*u2Trans[i]*uBase[i], -1*u2Trans[i]*vBase[i], -1*u2Trans[i]]
        A[2 * i + 1] = [0, 0, 0, uBase[i], vBase[i], 1, -
1*v2Trans[i]*uBase[i], -1*v2Trans[i]*vBase[i], -1*v2Trans[i]]
```

```
    # Compute the SVD of A
    U, S, V = np.linalg.svd(A)
    # The solution is the last column of V
    H = V.T[:, -1]
    # reshape it into 3x3 matrix
    H = H.reshape(3, 3)
    return H
```

Figure 2.1 Homography function

And following Figure 2.2 shows the points we choose in two corresponding images.



(a) (b)

Figure 2.2 Chosen points in two images (a) Chosen points in left.jpg (b) Chosen points in right.jpg

In order to calculate homography matrix, we follow the algorithm below.

(1) Find n >= 6 2D-to-3D point correspondences {xi <-> Xi}

(2) For each correspondence point {xi <-> Xi}, compute Ai, where A is shown as below

$$A = \begin{pmatrix} X & Y & Z & 1 & 0 & 0 & 0 & 0 & -xX & -xY & -xZ & -x \\ 0 & 0 & 0 & 0 & X & Y & Z & 1 & -yX & -yY & -yZ & -y \end{pmatrix}$$

(3) Assemble n 2*12 matrices Ai into a single 2n*12 matrix A

(4) Compute the SVD of A. The solution for p is the last column of V

2. **List the 3x3 camera homography matrix H that you have calculated. [2 mark]**

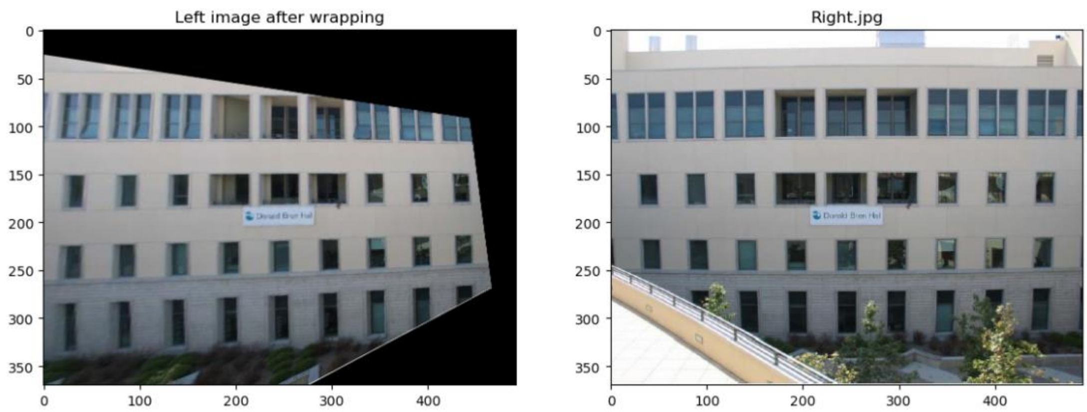
Homography matrix is shown in Figure 2.3.

```
H= [[-1.47021880e-02  7.66349670e-04  9.98864829e-01]
 [-2.73001265e-03 -5.75446618e-03  4.46729727e-02]
 [-2.04117359e-05  3.52550743e-06 -4.00874244e-03]]
```

Figure 2.3 Homography matrix

3. **Warp the left image according to the calculated homography. Study the factors that affect the rectified results, e.g., the distance between the corresponding points, e.g the selected points and the warped ones. [3 mark] (Note: you can use builtin image warping functions in matlab and python.)**

According to the points chosen in Figure 2.2 and applying warping to the left image, the result is shown in Figure 2.4, as well as the original right image for reference.



(a) (b)

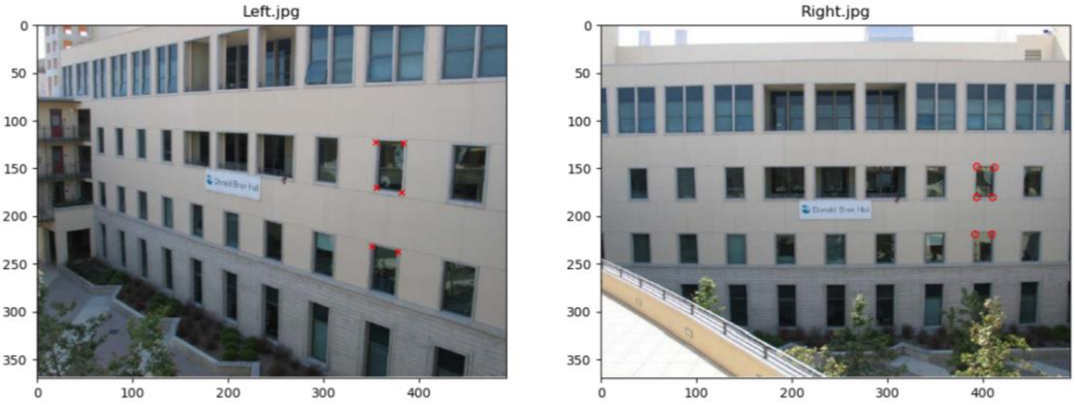
Figure 2.4 Warped image and original right image (a) Warped image (b) original right image.

The distance between wrapping points and original points is shown in Figure 2.5.

```
Distance between wrapping points and original points:
[ 82.32612597 117.69776549 231.20748808 295.00821747 408.49616025
 464.09324885]
```

Figure 2.5 Distance between wrapping points and original points for first group of points

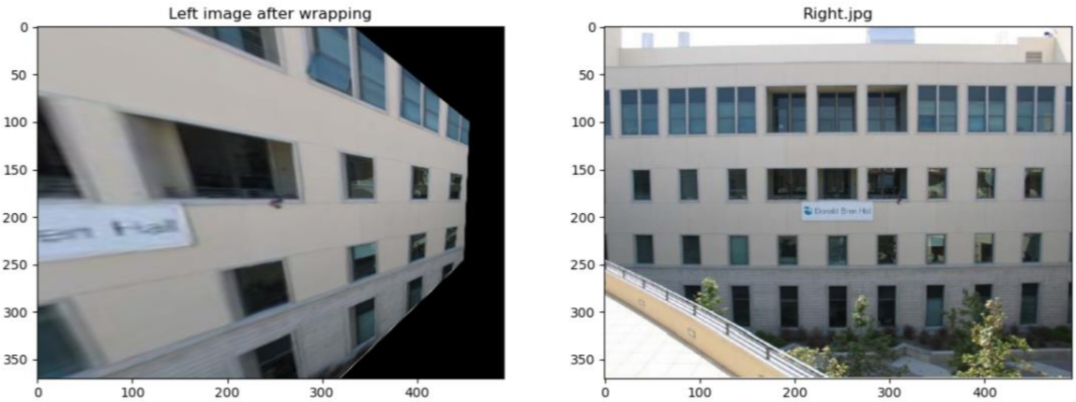
In order to consider the factors that affect the result, we consider choosing points as below in Figure 2.6.



(a) (b)

Figure 2.6 Chosen points in two images (2^{nd} group) (a) Chosen points in left.jpg (b) Chosen points in right.jpg

And we warp the left image we can get the following results in Figure 2.7 and the distance between wrapping points and original points in Figure 2.8.



(a) (b)

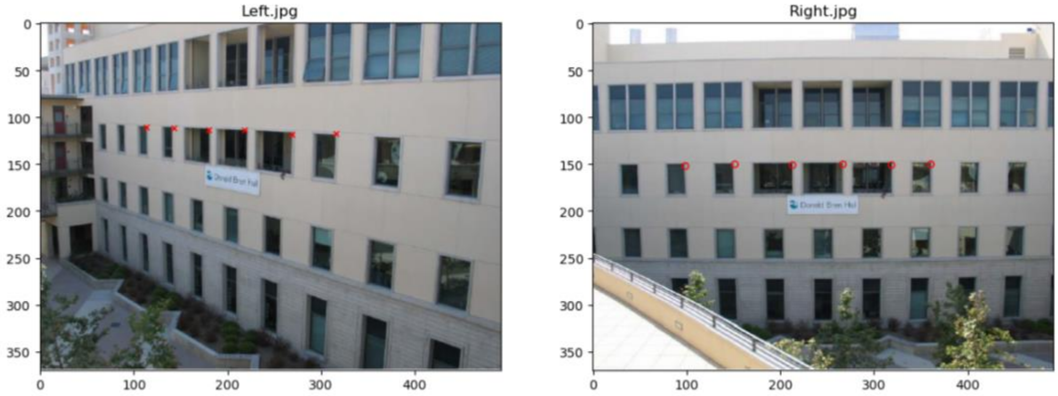
Figure 2.7 Warped image and original right image (2^{nd} group) (a) Warped image (b) original right image.

```
Distance between wrapping points and original points:
[373.75730579 401.78990653 392.19948925 417.90514694 418.14970409
 444.29019093]
```

Figure 2.8 Distance between wrapping points and original points for first group of points

Basically, we can get our first factor. The distance between the corresponding points can affect the warping. Large distances may introduce more distortion or inaccuracies in the rectified image.

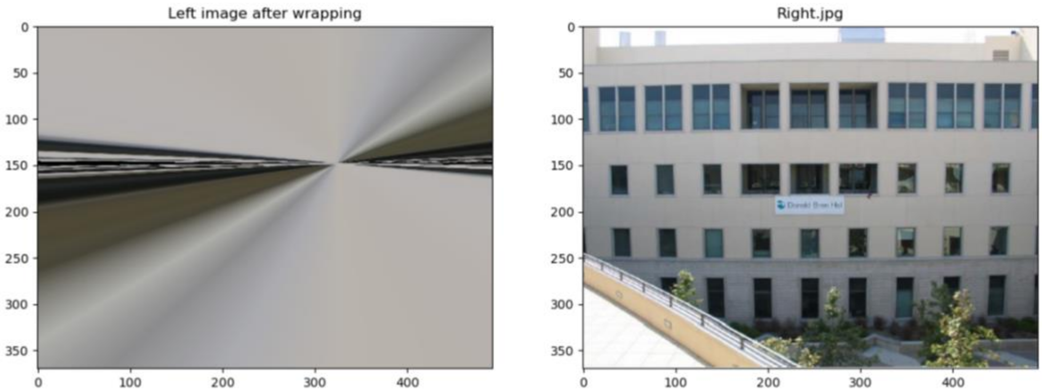
Then we consider choosing the following points as our 3rd group. All corresponding chosen points are shown in Figure 2.9.



(a) (b)

Figure 2.9 Chosen points in two images (3rd group) (a) Chosen points in left.jpg (b) Chosen points in right.jpg

And we warp the left image we can get the following results in Figure 2.10 and the distance between wrapping points and original points in Figure 2.11.



(a) (b)

Figure 2.10 Warped image and original right image (a) Warped image (b) original right image.

```
Distance between wrapping points and original points:
[157.26249386 179.45927197 211.36961288 245.08757891 292.00197418
 335.46473468]
```

Figure 2.8 Distance between wrapping points and original points for first group of points

Now we can get the second factor is that when choosing corresponding points in a horizon or vertical line, we can't get a good warping result. Ideally, selecting points evenly distributed on the diagonal can get a better result.

====================== End of CLab-3 =====================