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

!!! This is just a template for CLab-3. Please also make sure you follow all the requirements in Clab-3 assignment file if it is not mentioned in this template !!!

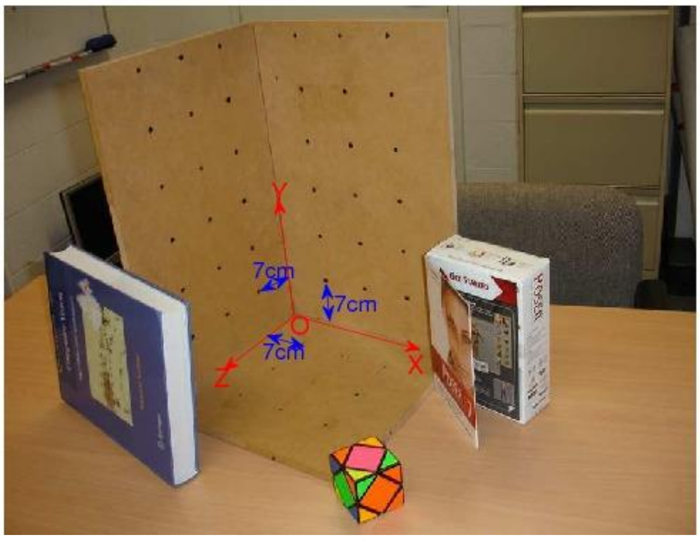
!!! Please also note that the sample outputs given in this document are not guaranteed to be correct. They just gives you an idea of how your outputs should look like !!!

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

(Acknowledgement: Lab material courtesy of Professor. Du Huynh of UWA).

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:

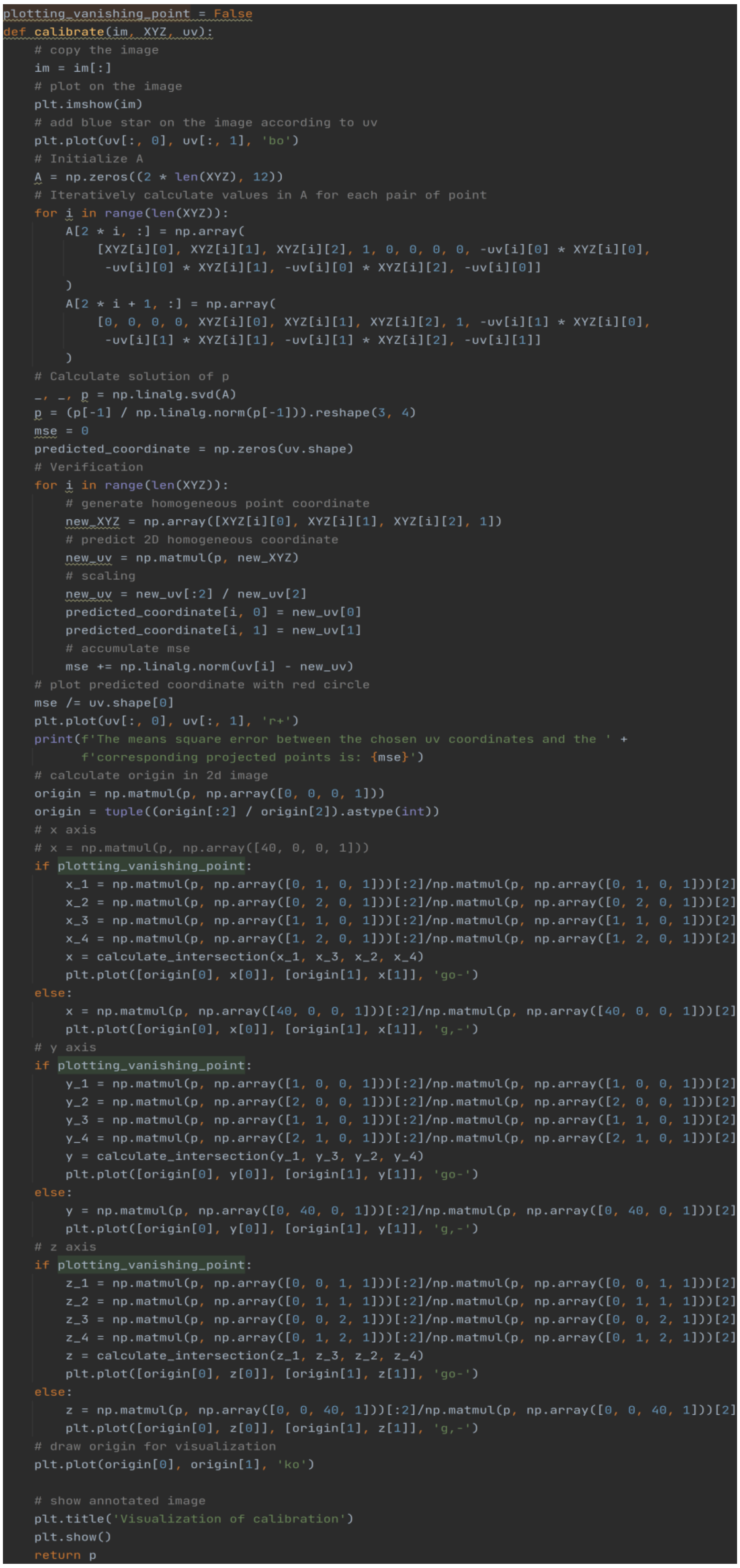
```
function C = calibrate(im, XYZ, uv)
```

Python user:

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

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

The screenshot of my function shown below:

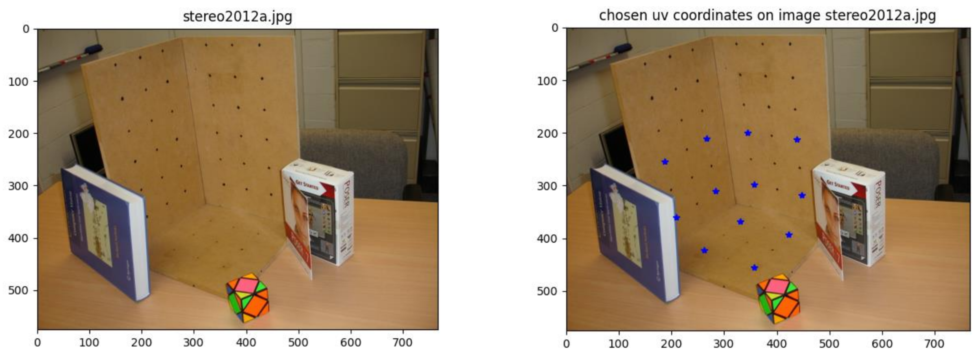


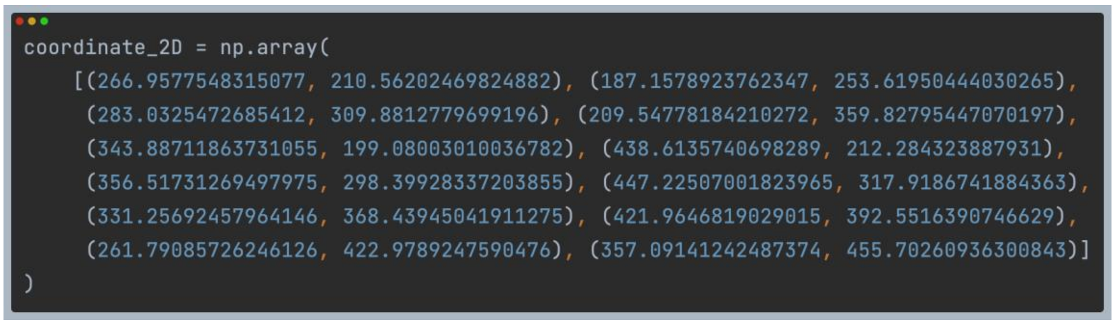
```python
plotting_vanishing_point = False
def calibrate(im, XYZ, uv):
    # copy the image
    im = im[:]
    # plot on the image
    plt.imshow(im)
    # add blue star on the image according to uv
    plt.plot(uv[:, 0], uv[:, 1], 'bo')
    # Initialize A
    A = np.zeros((2 * len(XYZ), 12))
    # Iteratively calculate values in A for each pair of point
    for i in range(len(XYZ)):
        A[2 * i, :] = np.array(
            [XYZ[i][0], XYZ[i][1], XYZ[i][2], 1, 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]]
        )
        A[2 * i + 1, :] = np.array(
            [0, 0, 0, 0, XYZ[i][0], XYZ[i][1], XYZ[i][2], 1, -uv[i][1] * XYZ[i][0],
             -uv[i][1] * XYZ[i][1], -uv[i][1] * XYZ[i][2], -uv[i][1]]
        )
    # Calculate solution of p
    _, _, p = np.linalg.svd(A)
    p = (p[-1] / np.linalg.norm(p[-1])).reshape(3, 4)
    mse = 0
    predicted_coordinate = np.zeros(uv.shape)
    # Verification
    for i in range(len(XYZ)):
        # generate homogeneous point coordinate
        new_XYZ = np.array([XYZ[i][0], XYZ[i][1], XYZ[i][2], 1])
        # predict 2D homogeneous coordinate
        new_uv = np.matmul(p, new_XYZ)
        # scaling
        new_uv = new_uv[:2] / new_uv[2]
        predicted_coordinate[i, 0] = new_uv[0]
        predicted_coordinate[i, 1] = new_uv[1]
        # accumulate mse
        mse += np.linalg.norm(uv[i] - new_uv)
    # plot predicted coordinate with red circle
    mse /= uv.shape[0]
    plt.plot(uv[:, 0], uv[:, 1], 'r+')
    print(f'The means square error between the chosen uv coordinates and the ' +
          f'corresponding projected points is: {mse}')
    # calculate origin in 2d image
    origin = np.matmul(p, np.array([0, 0, 0, 1]))
    origin = tuple((origin[:2] / origin[2]).astype(int))
    # x axis
    # x = np.matmul(p, np.array([40, 0, 0, 1]))
    if plotting_vanishing_point:
        x_1 = np.matmul(p, np.array([0, 1, 0, 1]))[:2]/np.matmul(p, np.array([0, 1, 0, 1]))[2]
        x_2 = np.matmul(p, np.array([0, 2, 0, 1]))[:2]/np.matmul(p, np.array([0, 2, 0, 1]))[2]
        x_3 = np.matmul(p, np.array([1, 1, 0, 1]))[:2]/np.matmul(p, np.array([1, 1, 0, 1]))[2]
        x_4 = np.matmul(p, np.array([1, 2, 0, 1]))[:2]/np.matmul(p, np.array([1, 2, 0, 1]))[2]
        x = calculate_intersection(x_1, x_3, x_2, x_4)
        plt.plot([origin[0], x[0]], [origin[1], x[1]], 'go-')
    else:
        x = np.matmul(p, np.array([40, 0, 0, 1]))[:2]/np.matmul(p, np.array([40, 0, 0, 1]))[2]
        plt.plot([origin[0], x[0]], [origin[1], x[1]], 'g,-')
    # y axis
    if plotting_vanishing_point:
        y_1 = np.matmul(p, np.array([1, 0, 0, 1]))[:2]/np.matmul(p, np.array([1, 0, 0, 1]))[2]
        y_2 = np.matmul(p, np.array([2, 0, 0, 1]))[:2]/np.matmul(p, np.array([2, 0, 0, 1]))[2]
        y_3 = np.matmul(p, np.array([1, 1, 0, 1]))[:2]/np.matmul(p, np.array([1, 1, 0, 1]))[2]
        y_4 = np.matmul(p, np.array([2, 1, 0, 1]))[:2]/np.matmul(p, np.array([2, 1, 0, 1]))[2]
        y = calculate_intersection(y_1, y_3, y_2, y_4)
        plt.plot([origin[0], y[0]], [origin[1], y[1]], 'go-')
    else:
        y = np.matmul(p, np.array([0, 40, 0, 1]))[:2]/np.matmul(p, np.array([0, 40, 0, 1]))[2]
        plt.plot([origin[0], y[0]], [origin[1], y[1]], 'g,-')
    # z axis
    if plotting_vanishing_point:
        z_1 = np.matmul(p, np.array([0, 0, 1, 1]))[:2]/np.matmul(p, np.array([0, 0, 1, 1]))[2]
        z_2 = np.matmul(p, np.array([0, 1, 1, 1]))[:2]/np.matmul(p, np.array([0, 1, 1, 1]))[2]
        z_3 = np.matmul(p, np.array([0, 0, 2, 1]))[:2]/np.matmul(p, np.array([0, 0, 2, 1]))[2]
        z_4 = np.matmul(p, np.array([0, 1, 2, 1]))[:2]/np.matmul(p, np.array([0, 1, 2, 1]))[2]
        z = calculate_intersection(z_1, z_3, z_2, z_4)
        plt.plot([origin[0], z[0]], [origin[1], z[1]], 'go-')
    else:
        z = np.matmul(p, np.array([0, 0, 40, 1]))[:2]/np.matmul(p, np.array([0, 0, 40, 1]))[2]
        plt.plot([origin[0], z[0]], [origin[1], z[1]], 'g,-')
    # draw origin for visualization
    plt.plot(origin[0], origin[1], 'ko')

    # show annotated image
    plt.title('Visualization of calibration')
    plt.show()
    return p
```

| Mistake | Marks Deducted |
|---|---|
| Not return calibration matrix | -1 |
| Not plot uv coordinates | -0.5 |
| Not plot projected 3D coordinates | -0.5 |
| Not plot the require lines | -0.5 |
| Not calculate the mean square error | -0.5 |
| Incorrect implementation | -3 |

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

Image stereo2012a.jpg is used for the algorithm. The selected uv position is plotted below:



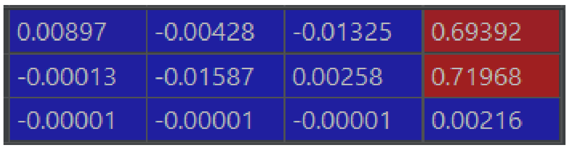


```
coordinate_2D = np.array(
    [(266.9577548315077, 210.56202469824882), (187.1578923762347, 253.61950444030265),
     (283.0325472685412, 309.8812779699196), (209.54778184210272, 359.82795447070197),
     (343.88711863731055, 199.08003010036782), (438.6135740698289, 212.284323887931),
     (356.51731269497975, 298.39928337203855), (447.22507001823965, 317.9186741884363),
     (331.25692457964146, 368.43945041911275), (421.9646819029015, 392.551639074662
9),
     (261.79085726246126, 422.9789247590476), (357.09141242487374, 455.70260936300843)]
)
```

| Mistake | Marks Deducted |
|---|---|
| Not show the chosen image | -0.5 |

**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]

Camera calibration matrix:

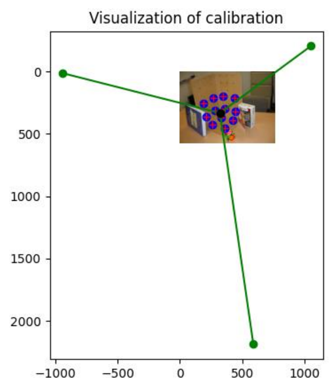


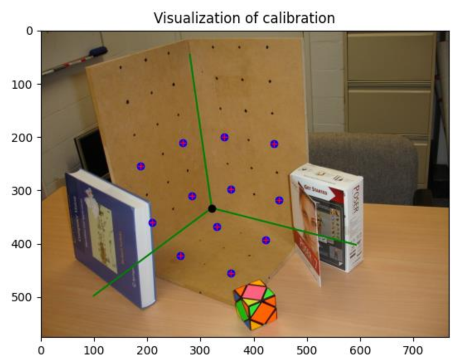
| | | | |
|---|---|---|---|
| 0.00897 | -0.00428 | -0.01325 | 0.69392 |
| -0.00013 | -0.01587 | 0.00258 | 0.71968 |
| -0.00001 | -0.00001 | -0.00001 | 0.00216 |

-

The means square error between the chosen uv coordinates and the corresponding projected points is: 0.43931130050496275

The plot for the projected XYZ coordinates and axis lines from origin is shown below:





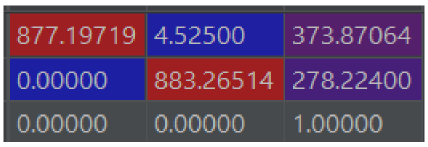
Where chosen dots are labelled with blue circles, origin is labelled with black circle, predicted dots are labelled red +s, the axis are labelled with green lines, and the vanishing points on each axis are plotted on the left graph with green dots. For visualization, the plot without vanishing points are shown on the right, with other labels the same.

We can see that the prediction points are very close to the labelled ones.

| Mistake | Marks Deducted |
|---|---|
| Incorrect calibration matrix | -1 |
| Incorrect annotated image | -0.5 |
| Incorrect reprojection error | -0.5 |

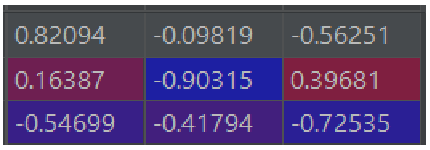
**4.** Decompose the P matrix into K, R, t, such that P = K[R|t], by using the following provided code (`vgg_KR_from_P.m` or `vgg_KR_from_P.py`). List the results, namely the K, R, t matrices, in your PDF file. [1.5 marks]

Matrix K:



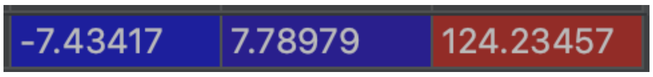
| | | |
|---|---|---|
| 877.19719 | 4.52500 | 373.87064 |
| 0.00000 | 883.26514 | 278.22400 |
| 0.00000 | 0.00000 | 1.00000 |

Matrix R:



| | | |
|---|---|---|
| 0.82094 | -0.09819 | -0.56251 |
| 0.16387 | -0.90315 | 0.39681 |
| -0.54699 | -0.41794 | -0.72535 |

Matrix t:



| | | |
|---|---|---|
| -7.43417 | 7.78979 | 124.23457 |

| Mistake | Marks Deducted |
|---|---|
| Incorrect matrix | -0.5 for each matrix |

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

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

focal length = 1244.8340219723277 (pixels)

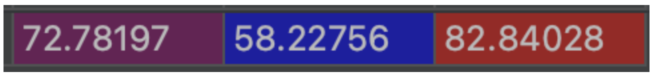
| Mistake | Marks Deducted |
|---|---|
| Incorrect focal length | -1 |

- 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]

Pitch angle = 0.5787684671717165

| Mistake | Marks Deducted |
|---|---|
| Incorrect angle | -2 |

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



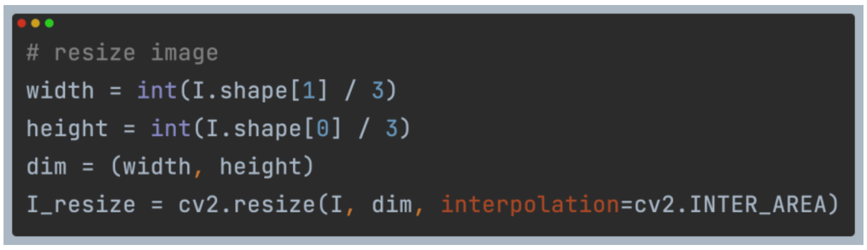
| 72.78197 | 58.22756 | 82.84028 |
|---|---|---|

| Mistake | Marks Deducted |
|---|---|
| Incorrect coordinate | -1 |

**6.** Please resize your selected image using builtin function from matlab or python to (H/2, W/2) 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]

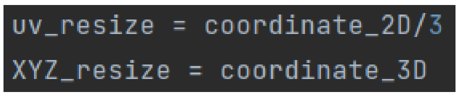
Display your code for resize the image and picking the uv coordinates of the resized image here.

Code for resizing image:



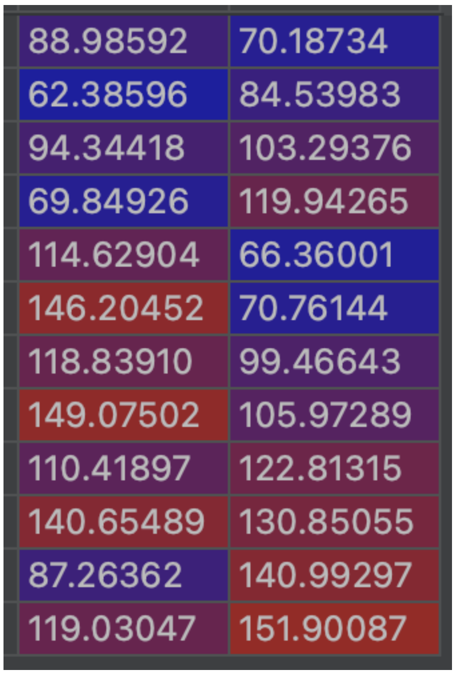
```
# resize image
width = int(I.shape[1] / 3)
height = int(I.shape[0] / 3)
dim = (width, height)
I_resize = cv2.resize(I, dim, interpolation=cv2.INTER_AREA)
```

The new coordinate pairs are generated by dividing the coordinate pairs in question 1 by 3.



```
uv_resize = coordinate_2D/3
XYZ_resize = coordinate_3D
```

The new coordinates of uv are shown below:



|  |  |
|---|---|
| 88.98592 | 70.18734 |
| 62.38596 | 84.53983 |
| 94.34418 | 103.29376 |
| 69.84926 | 119.94265 |
| 114.62904 | 66.36001 |
| 146.20452 | 70.76144 |
| 118.83910 | 99.46643 |
| 149.07502 | 105.97289 |
| 110.41897 | 122.81315 |
| 140.65489 | 130.85055 |
| 87.26362 | 140.99297 |
| 119.03047 | 151.90087 |

The new coordinates of XYZ are the same as before.

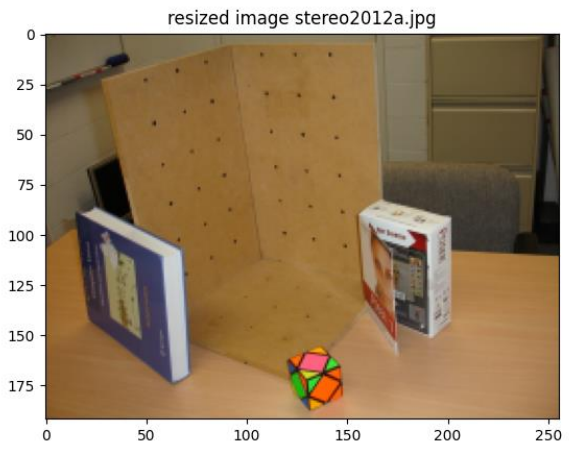
| Mistake | Marks Deducted |
|---|---|
| Resized image has an incorrect size | -0.5 |

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]

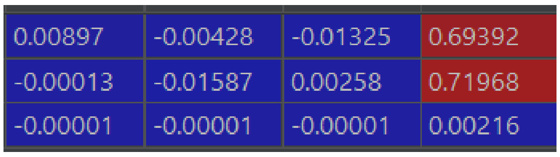
Show 5 things here:

- The calibration matrix P’
- The intrinsic matrix K’
- The rotation matrix R’
- The translation vector t’

The resized image:

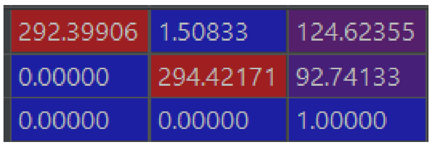


The calibration matrix P':



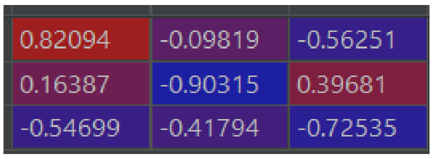
| | | | |
|---|---|---|---|
| 0.00897 | -0.00428 | -0.01325 | 0.69392 |
| -0.00013 | -0.01587 | 0.00258 | 0.71968 |
| -0.00001 | -0.00001 | -0.00001 | 0.00216 |

The intrinsic matrix K':



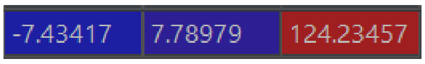
| | | |
|---|---|---|
| 292.39906 | 1.50833 | 124.62355 |
| 0.00000 | 294.42171 | 92.74133 |
| 0.00000 | 0.00000 | 1.00000 |

The rotation matrix R'



| | | |
|---|---|---|
| 0.82094 | -0.09819 | -0.56251 |
| 0.16387 | -0.90315 | 0.39681 |
| -0.54699 | -0.41794 | -0.72535 |

The translation vector t':



| | | |
|---|---|---|
| -7.43417 | 7.78979 | 124.23457 |

| Mistake | Marks Deducted |
|---|---|
| Not show the resize image | -0.4 |
| Incorrect matrix | -0.4 for each matrix |

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]

Analyse the differences:

We can see that K has about three times larger values K’s values. R and R’ are almost the same. t does not change.

Provide reasoning when changes happened or there are no changes.

Rotation matrix R will not change because the image is resized and the angle from camera will not change the relative orientation or rotation of the camera in 3D space. The intrinsic matrix K (except the last roll) will change in factor of the resizing factor to reflect the change caused by resizing: the change of pixel coordinates. The translation vector t will not change because after resizing, the camera’s position in 3D coordinate should be the same.

| Mistake | Marks Deducted |
|---|---|
| Not analyse the difference Or incorrect discussion. | -1 |
| Not provide reasoning Or incorrect discussion. | -1 |
| Discussion is partially correct | -0.5 |

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]

The focal length f’ = 414.94467366828013

The focal length and principal points x_0 and y_0 will change in factor of the resizing factor as it will re-scale the coordinates into pixels. If the pixels resized, it will change correspondingly.

Given a point with coordinate (x, y), the transformation by K is shown below:

$$\begin{pmatrix} fx + x_0 \\ fy + y_0 \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & x_0 \\ 0 & f & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

After resizing with factor k, the new output coordinate should be change in factor k. Fixing the input coordinates, it is obvious that f, x_0, y_0 should change in factor k.

| Mistake | Marks Deducted |
|---|---|
| Not discuss the relationship Or incorrect discussion | -1 |

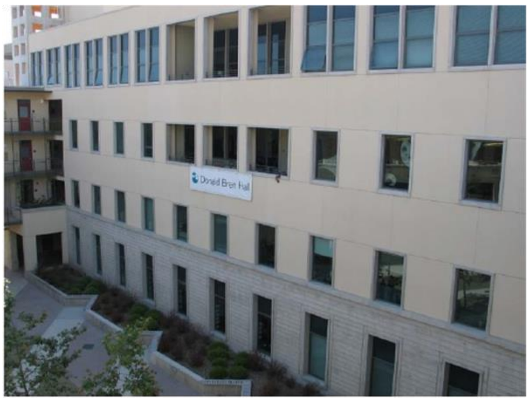
|  |  |
|---|---|
| Not discuss the connection to image size<br>Or incorrect discussion | -1 |
| Discussion is partially correct | -0.5 |

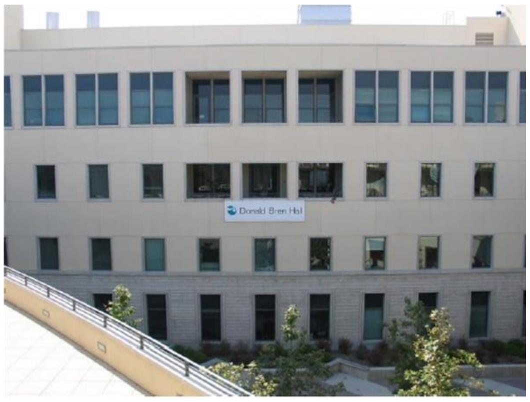
## Task-2: Two-View DLT based homography estimation. (10 marks)

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, | are vectors with coordinates u and v of the transformed |
| | v2Trans: | image point (p') |
| | uBase, | are vectors with coordinates u and v of the original base |
| | vBase: | image point p |
| Output: | H: | is a 3x3 Homography matrix |

---

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

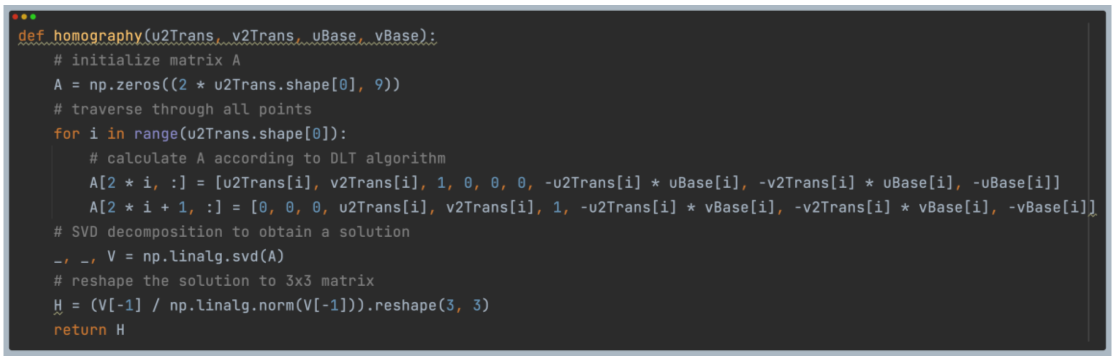
1. List your source code for homography estimation function and display the two images and the location of six pairs of selected points (namely, plotted those points on images). Explain what you have done for the homography and what is shown in the images. [5 marks]

I used DLT algorithm on Homography slide in week nine. Firstly, I initialized matrix A, then I filled in the values calculated by:

$$\begin{pmatrix} x & y & 1 & 0 & 0 & 0 & -x'x & -x'y & -x' \\ 0 & 0 & 0 & x & y & 1 & -y'x & -y'y & -y' \end{pmatrix}$$

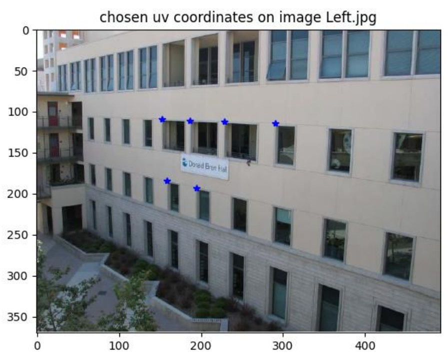
In for each selected points in order. Finally, a solution is obtained by SVD decomposition and returned after normalization.

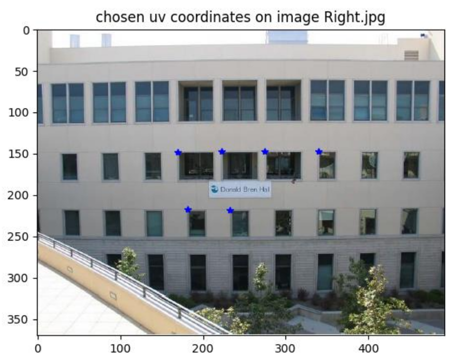
The code is shown below:



```python
def homography(u2Trans, v2Trans, uBase, vBase):
    # initialize matrix A
    A = np.zeros((2 * u2Trans.shape[0], 9))
    # traverse through all points
    for i in range(u2Trans.shape[0]):
        # calculate A according to DLT algorithm
        A[2 * i, :] = [u2Trans[i], v2Trans[i], 1, 0, 0, 0, -u2Trans[i] * uBase[i], -v2Trans[i] * uBase[i], -uBase[i]]
        A[2 * i + 1, :] = [0, 0, 0, u2Trans[i], v2Trans[i], 1, -u2Trans[i] * vBase[i], -v2Trans[i] * vBase[i], -vBase[i]]
    # SVD decomposition to obtain a solution
    _, _, V = np.linalg.svd(A)
    # reshape the solution to 3x3 matrix
    H = (V[-1] / np.linalg.norm(V[-1])).reshape(3, 3)
    return H
```

The selected corresponding dost pairs labelled with blue stars in two images are shown below:

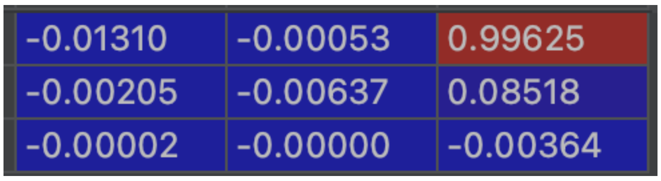




| Mistake | Marks Deducted |
|---|---|
| Incorrect homography estimation | -3 |
| Not display two images | -1 |
| No explanation Or incorrect explanation | -1 |
| Discussion is partially correct | -0.5 |

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

The homography matrix is shown below:

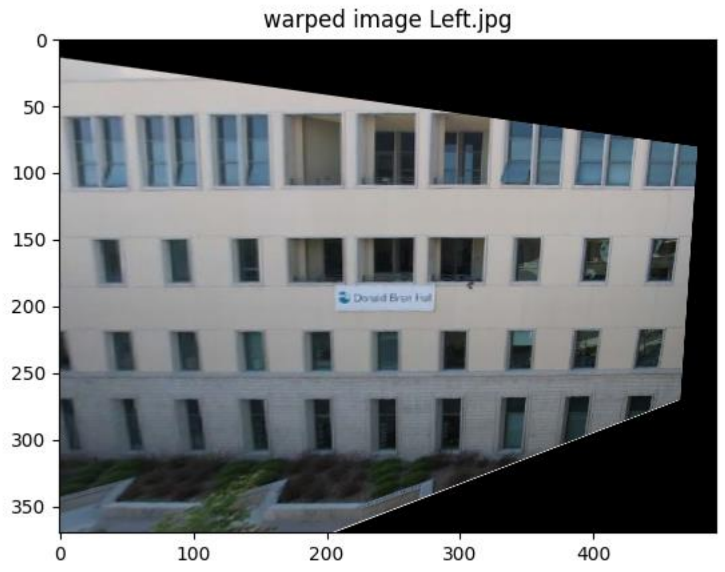


| | | |
|---|---|---|
| -0.01310 | -0.00053 | 0.99625 |
| -0.00205 | -0.00637 | 0.08518 |
| -0.00002 | -0.00000 | -0.00364 |

| Mistake | Marks Deducted |
|---|---|
| Incorrect homography | -2 |

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.)

Warped image is shown below:



The means square error between predicted selected trans points and corresponding base points is: 0.7285379514764299.

There are several factors affecting the rectified results:

1. The resolution of the image will greatly affect the result as the pixels in the image are discrete, but the DLT method is operated in continuous space and errors will occur while determining the location.
2. Incorrect pairs of points selected will result in wrong input data. Since in this question the points are hand-picked, they are not strictly corresponding, thus caused errors.

| Mistake | Marks Deducted |
|---|---|
| Not show the warped image Or Incorrect warped image | -1.5 |
| No discussion Or Incorrect discussion | -1.5 |
| Discussion is partially correct | -0.5 |

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