Clab-3 Report

ENGN4528

UID: u6579712

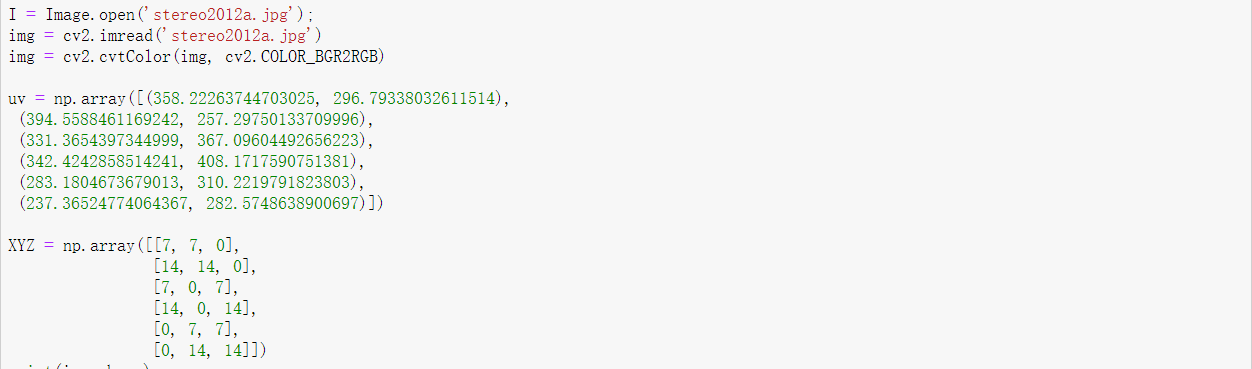
Jieli Zheng

5.22.2021

**Task-1: 3D-2D Camera Calibration**

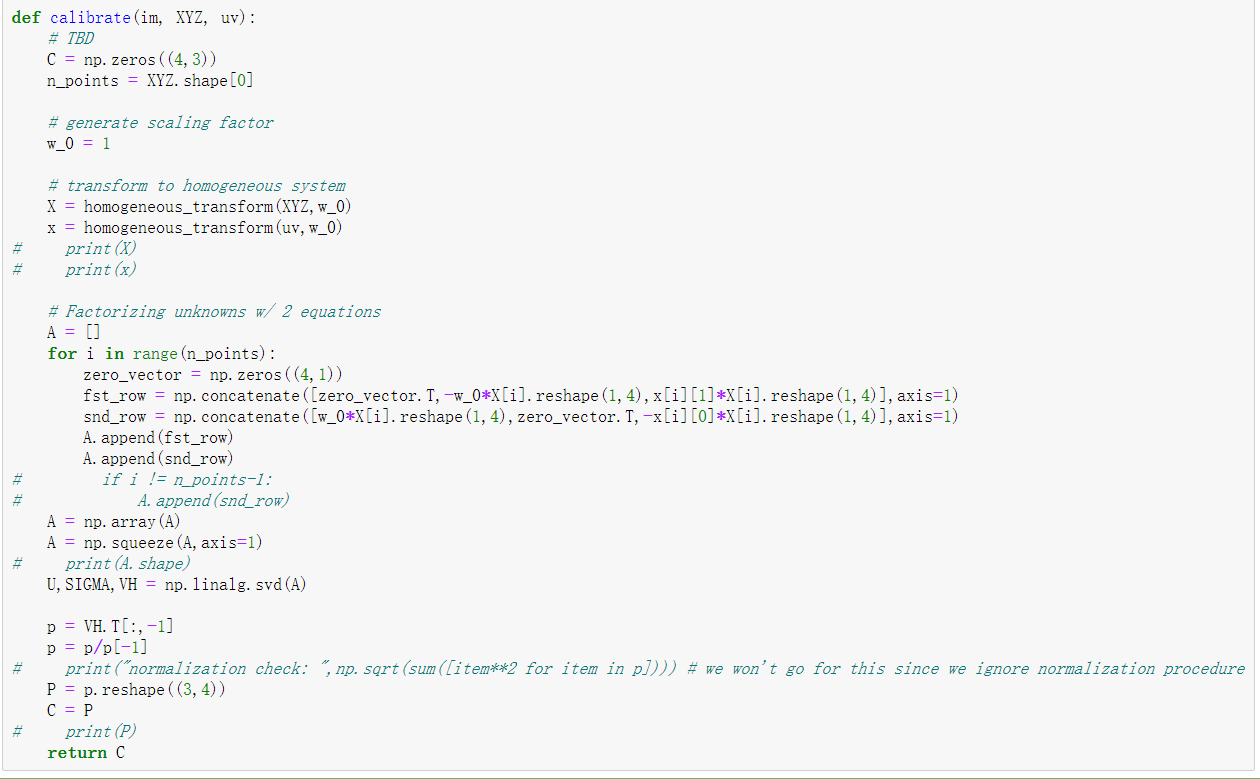
**1.1**

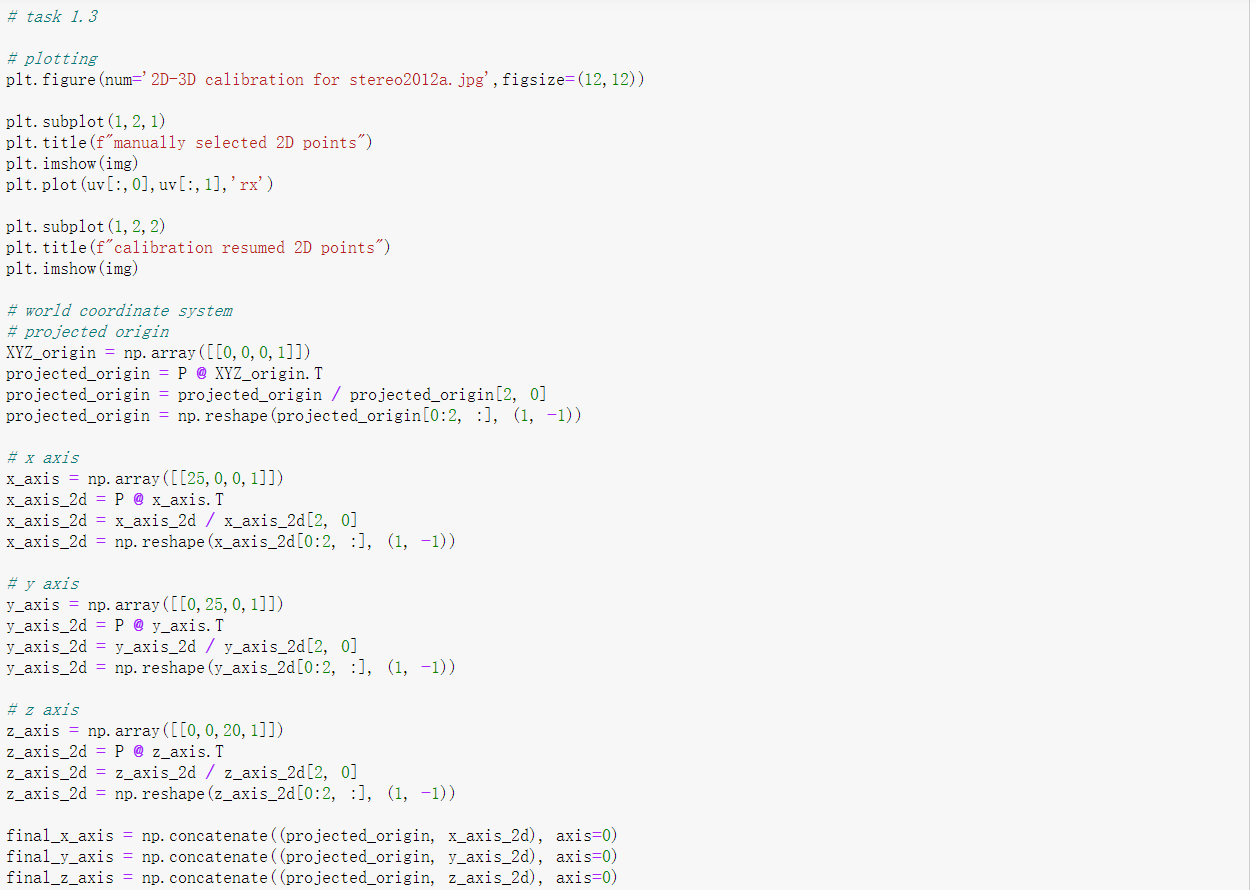
Firstly, in the preprocessing stage, the camera calibration function will take a couple of real-world coordinates, their correspoding image coordinates and image information as inputs. Since we always need to manually give real-world coordinates, I just use the ginput method once and record the coordinates for the quick re-run purpose. (See below)



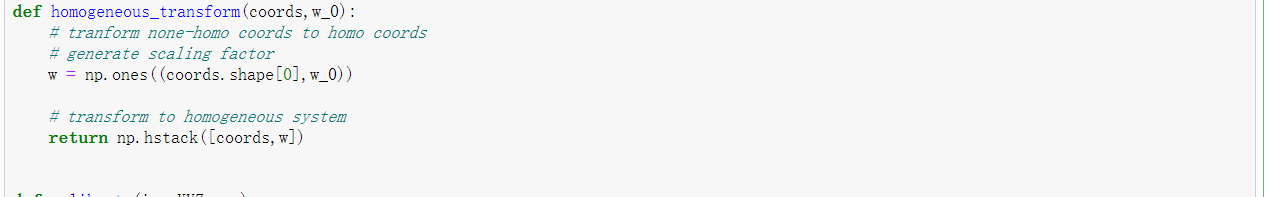
Then we have got all inputs for calibration.

The core calibration function is as below: (NOTICE: world coordinate system and MSE error are separated outside for code tidiness)





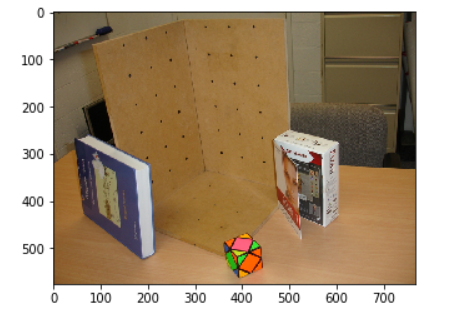
For the core function calibrate, we firstly need to transform the non-homogeneous coordinates into homogeneous coordinates. Here I use a helper method homogeneous\_transformto transform both **X** and **x** into the homogeneous format. The scaling factor w\_0 can be decided by outer calls.



Once **X** and **x** are transformed, we make use of them to compute coefficient matrix **A** as is described in the lecture. Since there are 11 d.o.f in matrix **P**, and for each point we can retrieve 2 independent equations of **p**, hence we only need 6 points (more precisely 5.5 but not possible here) to find the result of **p**. Then we use SVD decomposition to find **p** candidates. Since there are measurement errors in the real world, then we should choose the last column of **V** as **p**, which is the smallest singular value and the smallest errors for **Ap=0**. Finally, we reshape **p** (12x1)into **P** (3x4) and get the output of calibrate.

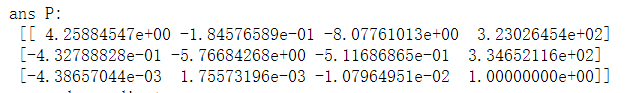
**1.2**

The image I choose is stereo2012a.jpg. As is shown below:

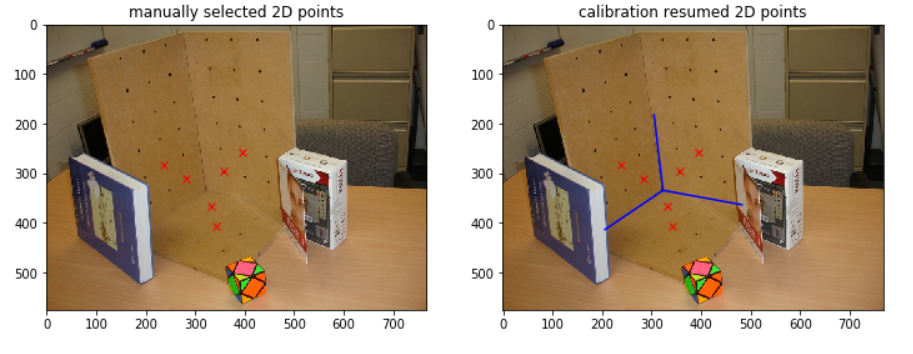


**1.3**

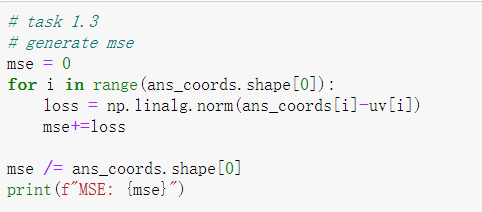
The 3x4 camera calibration matrix P is as below:



And the resumed 2D points results are as below:

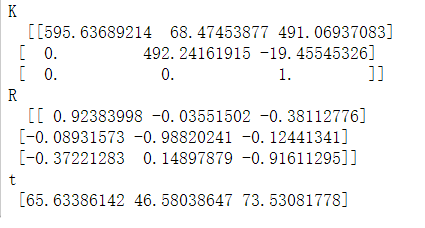


From the comparison pictures above, we can clearly find that the projection is solid and accurate by human eyes. Also, the MSE for this P is 0.10838, which is a good result for this algorithm, and code below:



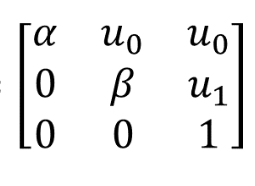
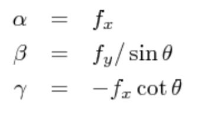
**1.4**

K, R, t matrix are shown below:



**1.5**

The overall focal length 𝐹 is the distance between the focal point and the image plane. If we take a look at the definition of the intrinsic matrix *K*, as shown below (from Lecture slide):

, where 

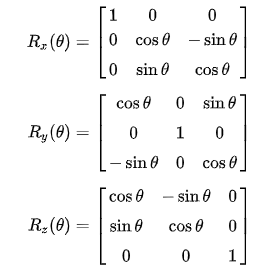
Here, and are focal length in the horizontal and vertical axis. Then, we can compute the overall focal length 𝐹 by

As a result, is K[0,0] and we have is . With matrix above we can compute:

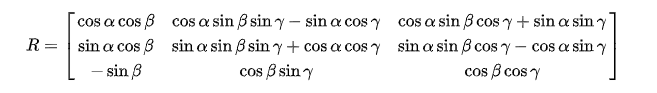
*595.637-489.02*

*770.665 (pixels)*

Then, the pitch angle of the camera with respect to the X-Z plane is the angle between the optical axis and ground plane. According to Wikipedia, the rotation matrices along three axes are as follows:



And if we assume , we have combined rotation matrix:

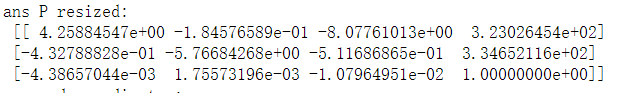


And we find that R[2,0] is *-0.3722* , . Therefore, we can compute the pitch angle *0.38139 (rad) = 21.852 (degree)*.

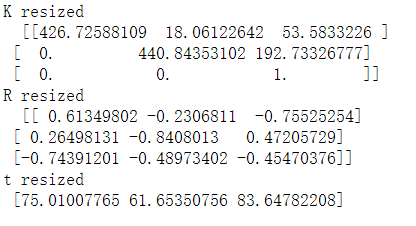
**1.6**

**a)**

The 3x4 camera calibration matrix P’ is as below:



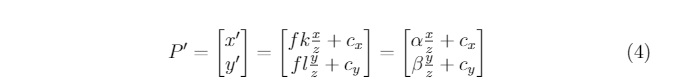
K’, R’, t’ matrix are shown below:



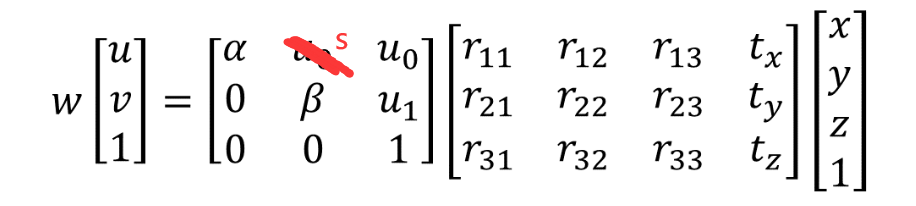
**b)**

**1) K and K’**

Firstly, from Stanford CS231A Course Notes (1):



and lecture slides:



where k and l are the aspect ratio in x and y axes.

Since here we resize the image to half size of the original image and the pixels in each axis are halved, it’s expected that k and l are also halved. Because resize is the same as put camera farther than before and skew is proportional to , then we can find that , are smaller than before.

**2) R and R’**

Since it’s the same angle and we can treat the resize as the camera is farther than before, the three angles should not change. Although the value in R is different from R’, the three angles (pitch, yaw, roll) should not change at all.

**3) t and t’**

Since it’s the same angle and we can treat the resize as the camera is farther than before, t’ is larger than before.

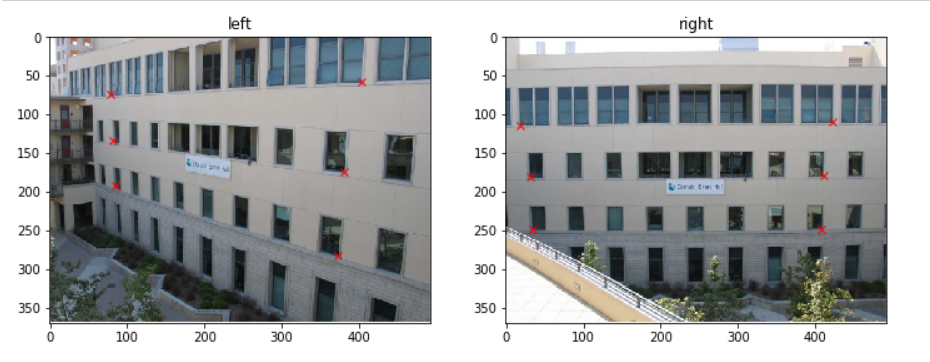
**Task-2: Two-View DLT based homography estimation**

**2.1**

The core homography function is as below:

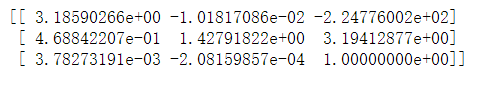


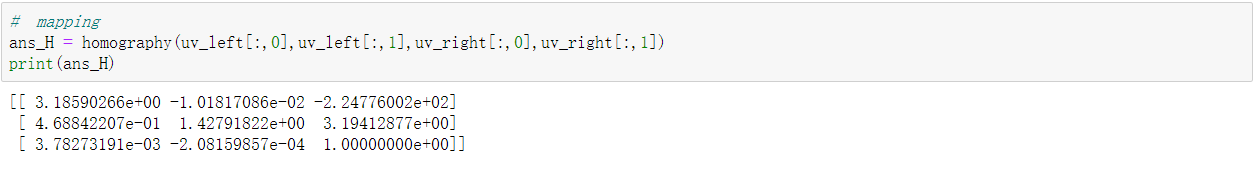
What it does is almost the same as in task1, which is standard DLT algorithm. The only difference is the dimension of the inputs are both 2D. The plotted points are as below:



**2.2**

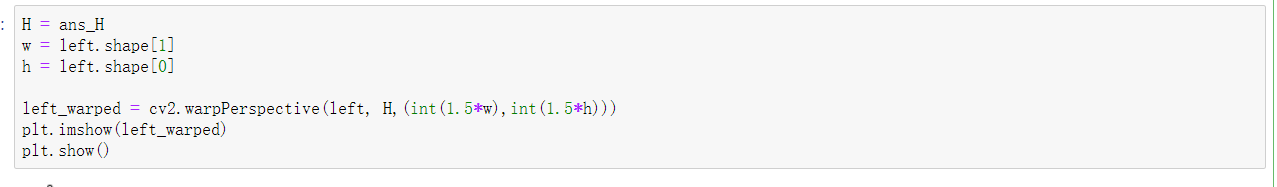
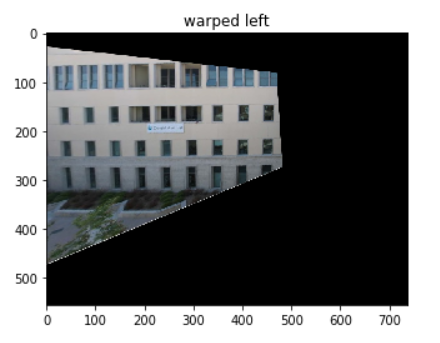
the 3x3 camera homography matrix H and code are as below:





**2.3**

The warped image and code are as below:

Which shows how left image will look like in the right.jpg position.