

cs512 Assignment 3

Jiranun Jiratrakanvong
Department of Computer Science
Illinois Institute of Technology

Nov 5, 2015

Abstract

The objective of this assignment is to learn about camera calibration by conducting two programs using python as a programming language. The programs of this assignment will extract feature points from calibration target by manual, and do *non-coplanar* calibration with RANSAC algorithm.

Problem Statement

The program from this assignment need to be separated into two parts. The first part is the program which can extract feature points of the calibration target. The output of this program is a text file containing object points and their corresponding image points. The second part is the program which can do the *non-coplanar* calibration with RANSAC algorithm. The input is the text file providing by the first part of the program, or any other correspondence file, and the output will be intrinsic and extrinsic parameter of the camera as determined by the *non-coplanar* calibration process.

Purpose Solution

The program was separated into 2 parts, producing correspondence file, and non-coplanar calibration. The first part is **Asg3Part1.py**, and the second part is **Asg3Part2.py**.

For the first part, **Asg3Part1.py**, the program need two input files which are a *text file* containing lists of object points from the world coordinates, and an *image file* of that object. The program allows users to interactively mark the points on the image. The number of marks depends on the number of points of the object in the text file. The number of points will be the number of lines in the text file, so the program will count the line without checking whether details in each line are object points or not.

The text file must contain only one point in each line, and the order must be X,Y,Z (separated by space). The program will create a new text file, and write a line X Y Z x y every time a user marks a point (x,y is the point that a user marks)

```
# mouse callback function
def mouse_callback(event,x,y,flags,param):
    global original_img, amt_marked_point, obj_pnts_file, correspondence_file
    if event == cv2.EVENT_LBUTTONDOWN:
        if (amt_marked_point < amt_of_obj_points):
            cv2.circle(original_img, (x, y), 2, (100,0,100), 2)
            cv2.putText(original_img, str(amt_marked_point+1), (x,y-4), cv2.FONT_HERSHEY_PLAIN, 1.0, (100,0,100), 1)
            amt_marked_point += 1
            corresponding_points = obj_pnts_file.readline().rstrip('\n')
            corresponding_points += ' '+str(x)+' '+str(y)+'\n'
            correspondence_file.write(corresponding_points)
```

Mouse callback function algorithm

The second part of the program is **Asg3Part2.py**. This part will perform non-coplanar calibration. The input file will be a file that contains correspondence points between object points (X,Y,Z) and image point (x,y). Each line of the file must be 'X Y Z x y' (separated by space). For example,

```
40.00 100.00 0.00 226 99
20.00 100.00 0.00 216 91
0.00 100.00 0.00 208 83
0.00 0.00 100.00 109 219
0.00 20.00 100.00 110 197
0.00 40.00 100.00 109 169
0.00 60.00 100.00 110 145
0.00 80.00 100.00 110 118
100.00 0.00 0.00 252 251
100.00 0.00 20.00 229 254
100.00 0.00 40.00 203 255
100.00 0.00 60.00 178 259
100.00 0.00 80.00 154 262
```

Moreover, all parameters of RANSAC Algorithm must be provided in RANSAC.config

After the program run, the calibration process will be completed. The process time will also depend on parameters in RANSAC.config. The algorithm of the calibration process is below.

1. Matrix A will be developed by object points and image points provided in the correspondence file. (If RANSAC enabled, only MINIMUM_POINTS will be used)

$$A = \begin{pmatrix} P_1^T & 0^T & -u_1 P_1^T \\ 0^T & P_1^T & -v_1 P_1^T \\ \dots & \dots & \dots \\ P_n^T & 0^T & -u_n P_n^T \\ 0^T & P_n^T & -v_n P_n^T \end{pmatrix} \text{ when P are object points and (u,v) are image points}$$

```

def get_matrix_A(obj_points, img_points):
    matA = []

    for i in range(len(obj_points)):
        minusX = -1.*float(img_points[i][0])
        minusY = -1.*float(img_points[i][1])
        Xi, Yi, Zi = float(obj_points[i][0]), float(obj_points[i][1]), float(obj_points[i][2])
        matA.append( [ Xi, Yi, Zi, 1., 0., 0., 0., 0., minusX*Xi, minusX*Yi, minusX*Zi, minusX])
        matA.append( [ 0., 0., 0., 0., Xi, Yi, Zi, 1., minusY*Xi, minusY*Yi, minusY*Zi, minusY])

    return matA

```

2. The K^* , R^* , and T^* will be calculated from matrix A by algorithm below

```

def get_KRT(matA):
    U, D, VT = np.linalg.svd(matA)

    #Check if the matrix is singular
    if min(D) < 0.000000001 :
        return False, 0,0,0
    else :
        M = VT[len(VT)-1,:]

        a1 = np.array( [M[0],M[1],M[2]])
        a2 = np.array( [M[4],M[5],M[6]])
        a3 = np.array( [M[8],M[9],M[10]])

        b = np.array( [M[3],M[7],M[11]])

        rho = 1./np.sqrt(sum(i**2 for i in a3))

        u_zero = np.dot((rho**2)*a1,a3)
        v_zero = np.dot((rho**2)*a2,a3)

        alpha_v = np.sqrt(np.dot(rho**2*a2,a2)-v_zero**2)

        s = (rho**4/alpha_v)*np.dot(np.cross(a1,a3),np.cross(a2,a3))

        alpha_u = np.sqrt(np.dot(rho**2*a1,a1)-s**2-u_zero**2)

        sign = np.sign(b[2])

        r3 = sign*np.abs(rho)*a3
        r1 = (rho**2/(alpha_u))*np.cross(a2, a3)

        r2 = np.cross(r3,r1)
        R_star = np.array( [r1,r2,r3])

        K_star = np.array([[alpha_u, s, u_zero], [0.0, alpha_v, v_zero], [0.0, 0.0, 1.0]])
        T_star = sign*np.abs(rho)*np.dot(inv(K_star),b)

    return True, K_star, R_star, T_star

```

3. The projection matrix (M) will be calculated by R*,K*, and T*

```
def get_new_M(K_star, R_star, T_star):  
    RT = np.zeros((3,4))  
    RT[0:3,0:3] = R_star  
    RT[0:3,3] = T_star  
  
    return np.dot(K_star,RT)
```

4. Calculate distances between all original image points and all image points calculated by M, and find inliers by using median of the distances as threshold.

```
new_M = get_new_M(K_star, R_star, T_star)  
  
img_points_2DH = [np.dot(new_M,obj_points_3DH[i]) for i in range(nPoints)]  
homogenized_img_points = [(img_points_2DH[i][0]/img_points_2DH[i][2],img_points_2DH[i][1]/img_points_2DH[i][2]) for i in range(nPoints)]  
  
distances = [np.sqrt((orig_img_points[i][0]-homogenized_img_points[i][0])**2+(orig_img_points[i][1]-homogenized_img_points[i][1])**2) for i in range(nPoints)]  
median_distance = np.median(distances)  
inliers = [i for i in range(nPoints) if distances[i] < median_distance]
```

5. Calculate W, and K for RANSAC algorithm

```
ransac_w = float(len([i for i in distances if i < median_distance]))/float(nPoints)  
ransac_k = np.log(1.-DESIRED_PROB)/np.log(1.-ransac_w**MINIMUM_POINTS)
```

6. Find M again using inliers
7. Repeat all process again until the number of iterations exceed K or MAX_ITER
8. Get best M by choosing M that provide least error for inliers
9. Show all intrinsic and extrinsic parameters with their error

Note: if RANSAC is disabled, the program will just do the process 1-3 with all points provided in the file

Implementation Details

Problem and Solution

- The object points and image points which teacher gave are in separated file. In order to make a correspondence file, a new program, make_calibration_data.py, conducted. This program will just combine object points and image points to a new file.

```
obj_pnts_file = open("object1.txt",'r')  
img_pnts_file = open("image1.txt",'r')  
  
correspondence_file = open("cali_data1.txt",'w')  
  
for oline in obj_pnts_file:  
    iline = (img_pnts_file.readline()).rstrip('\n')  
    ob = oline.rstrip('\n')  
    new = ob+" "+iline+'\n'  
  
    correspondence_file.write(new)
```

- The correspondence file has more than one space between corresponding points, so the program became error because it got None as the points. The solution is that cutting all points that contain None using filter(None, points)
- There are many formula of the calibration process, so if the result was incorrect, it's hard to debug where the mistake is. The solution is checking each parameters and result step by step, and hard code some value in order to check the correctness of some process. For example, $AxB = C$, $DxE = F$, and $CxF = G$. If we want to check if process of $CxF = G$ is correct, we don't need to do AxB and DxE . We can just assign the value of C and F, and check the result.

Instructions for using the program

- **Asg3Part1.py**

- Running via command line :
python Asg3Part1.py [object_points.txt] [image.jpg]
- The program will show the image and number of points that user have to mark.
- During marking the point, the program will show that points marked with its number.
- Press 'o' to print the object point file name
- Press 'h' for help
- Press <ESC> to quit the program

- **Asg3Part2.py**

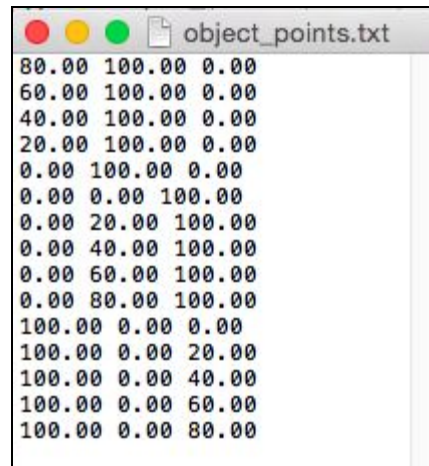
- Config RANSAC parameters in RANSAC.config

```
[RANSAC]
ENABLE_RANSAC=True
DESIRED_PROB=0.99
MAX_ITER=1000
MINIMUM_POINTS=6
```

- Running via command line :
python Asg2_part2.py [correspondence_file.txt]

Results and discussion

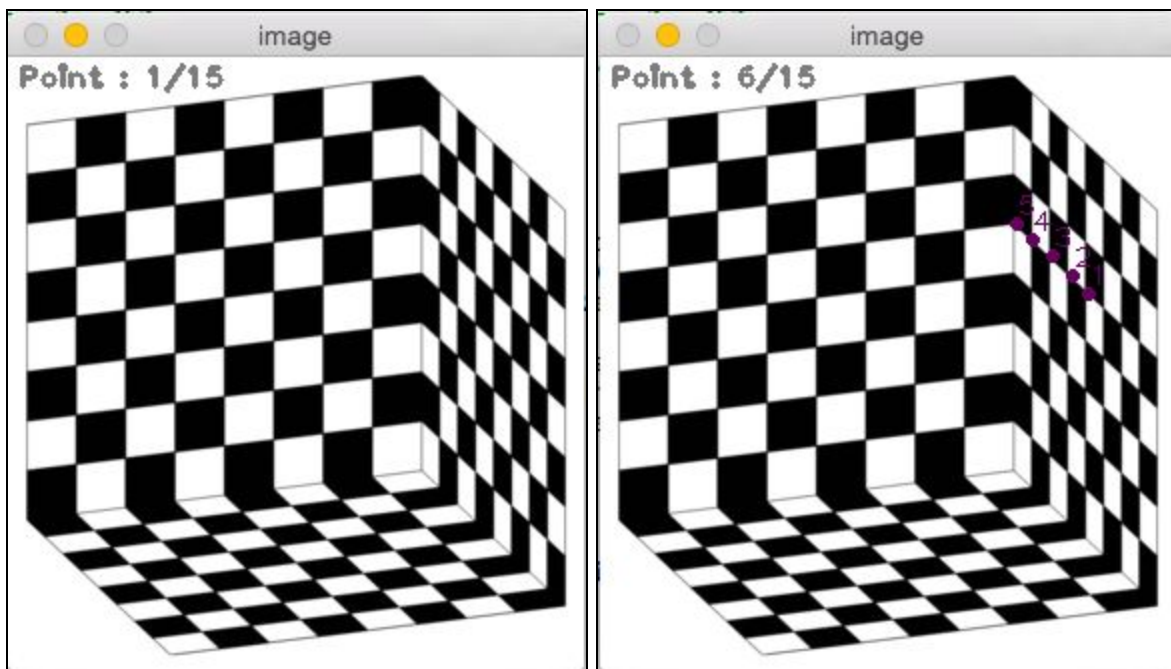
- Asg3Part1.py

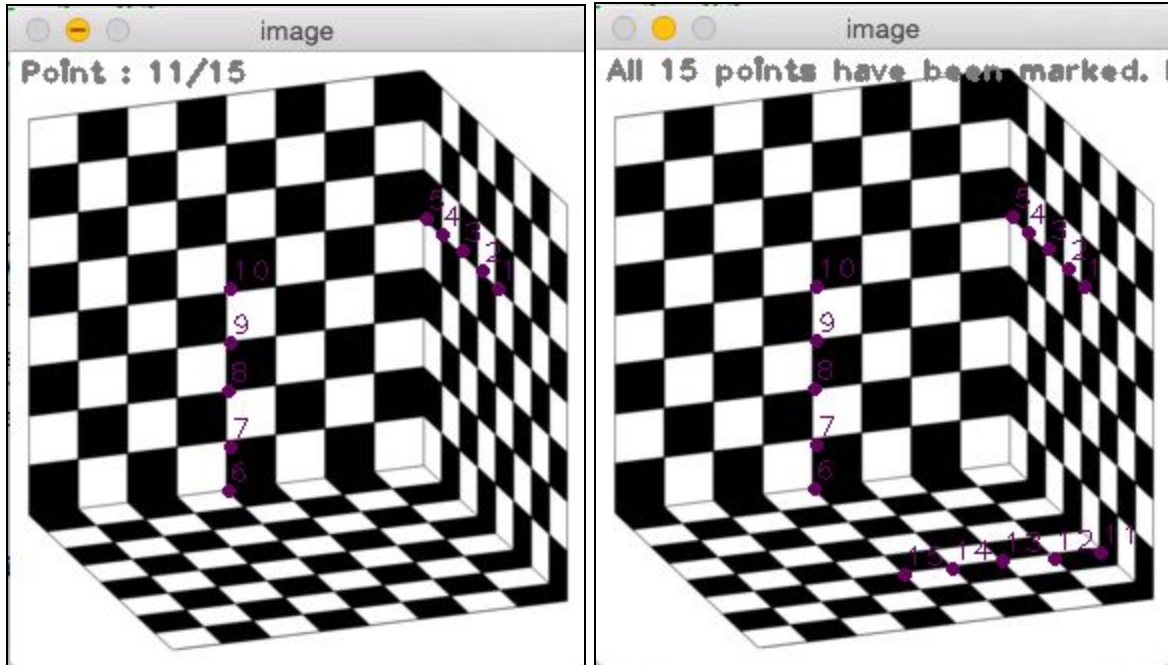


```
80.00 100.00 0.00
60.00 100.00 0.00
40.00 100.00 0.00
20.00 100.00 0.00
0.00 100.00 0.00
0.00 0.00 100.00
0.00 20.00 100.00
0.00 40.00 100.00
0.00 60.00 100.00
0.00 80.00 100.00
100.00 0.00 0.00
100.00 0.00 20.00
100.00 0.00 40.00
100.00 0.00 60.00
100.00 0.00 80.00
```

Example of object points file

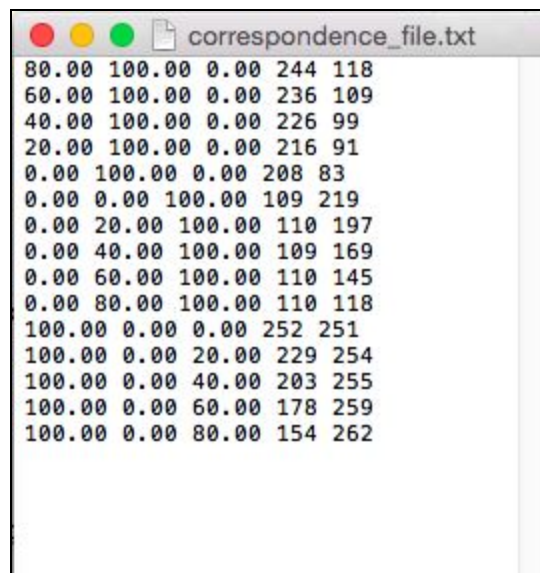
In this case, there are 15 points in the object points file. After the program run, the image will be shown, and display the number of points user has to mark. When a user marks a point the number of marked points increased and marked points will be shown with its number.





After all points marked, program will not allow user to mark any point anymore. User have to press <ESC> to close the program.

After the program closed, a new file, correspondence_file.txt, will be created. The file will contain object points and corresponding image points as below.



- **Asg3Part2.py**

The output depends on parameter **ENABLE_RANSAC** in RANSAC.config

- **ENABLE_RANSAC = False**

The program will print out all intrinsic and extrinsic parameters of the camera calibration along with its error.

```
##### Result #####
( u0 , v0 )      = ( 320.000162754 , 239.99993468 )
s                = -2.988809784e-05
( alphaU , alphaV ) = ( 652.174088815 , 652.174096338 )
T*              = [ -2.44061075e-04  8.79390487e-05  1.04880907e+03]
R*              =
[[ -7.68221208e-01  6.40184503e-01  1.37165543e-07]
 [  4.27274277e-01  5.12729161e-01 -7.44678133e-01]
 [ -4.76731460e-01 -5.72077464e-01 -6.67423771e-01]]
M ( K*[R*[T*])   =
[[ -6.53568124e+02  2.34446848e+02 -2.13575604e+02  3.35618915e+05]
 [  1.64241696e+02  1.97090124e+02 -6.45841450e+02  2.51714167e+05]
 [ -4.76731460e-01 -5.72077464e-01 -6.67423771e-01  1.04880907e+03]]
Error            = 3.88427783332e-05
```

Note : the program will not support when all points are in the same plane. If all points are in the same plane the program will print “ERROR! All points are in the same plane”

- **ENABLE_RANSAC = True**

The program will print out all intrinsic and extrinsic parameters of the camera calibration along with its error, and also number of experiments. The processing time will be increased if the number of experiments is large.

```
##### Result #####
Number of experiments = 293 times
( u0 , v0 )      = ( 320.000318785 , 239.998923198 )
s                = -6.21394986674e-05
( alphaU , alphaV ) = ( 652.172757083 , 652.172768034 )
T*              = [ -4.41136724e-04  1.67127721e-03  1.04880731e+03]
R*              =
[[ -7.68221188e-01  6.40184536e-01  2.27600129e-07]
 [  4.27273572e-01  5.12728380e-01 -7.44679083e-01]
 [ -4.76732134e-01 -5.72078133e-01 -6.67422716e-01]]
M ( K*[R*[T*])   =
[[ -6.53567392e+02  2.34445697e+02 -2.13575287e+02  3.35618387e+05]
 [  1.64240989e+02  1.97089351e+02 -6.45840152e+02  2.51713716e+05]
 [ -4.76732134e-01 -5.72078133e-01 -6.67422716e-01  1.04880731e+03]]
Error            = 2.63032924953e-05
```


Strength and weaknesses

For both parts, the programs assume that input file do not contain garbage data. The program will count the line without checking whether details in each line are object points or not. For the first part, the program will not be error, but the output will be incorrect. However, the program of the second part will produce error if input data contains garbage.

For the camera calibration process, the program do NOT SUPPORT when the corresponding points are on the SAME PLANE.

If RANSAC enabled, the process time might be long because the program was implemented only in Python.

Evaluation

Testing by calibration data from <http://www.cs.iit.edu/~agam/cs512/data/calibration/index.html> , which have both data with noise and without noise.

Assignment 3 Question : RANSAC is not handling well one of the provided cases. Explain the reason for RANSAC not being able to handle this case properly.

The second case of <http://www.cs.iit.edu/~agam/cs512/data/calibration/index.html> is the case that RANSAC is not handling well. In each experiment of RANSAC, small number of points will be randomly picked in order increase the possibility that all points will not be noisy data, but in this case, the points are noisy although small number of points picked. As a result, RANSAC can handle well in the case only if noisy data is similar to IMPULSIVE noise (only some points are noisy data.) If all points are noisy, RANSAC will not handle well.

The some comparison of noisy data provided below. The yellow highlight are points that do not have similar value as data without noise. It can be seen that although noisy data 1 have small amount of difference from data without noise, but all points have noise. That's why RANSAC can not handle well in this case.

200.00	0.00	0.00	214.9064	298.4516	200.00	0.00	0.00	214.2627	297.7435	200.00	0.00	0.00	214.9064	298.4516
200.00	20.00	0.00	222.4942	306.2609	200.00	20.00	0.00	226.6419	306.8645	200.00	20.00	0.00	227.8209	309.2874
200.00	40.00	0.00	230.2685	314.2623	200.00	40.00	0.00	228.3564	314.3918	200.00	40.00	0.00	230.2685	314.2623
200.00	60.00	0.00	238.2363	322.4629	200.00	60.00	0.00	239.4772	323.9749	200.00	60.00	0.00	238.2363	322.4629
200.00	80.00	0.00	246.4051	330.8702	200.00	80.00	0.00	245.2069	329.7931	200.00	80.00	0.00	246.4051	330.8702
200.00	100.00	0.00	254.7824	339.4921	200.00	100.00	0.00	251.7734	339.0150	200.00	100.00	0.00	254.7824	339.4921
200.00	120.00	0.00	263.3763	348.3371	200.00	120.00	0.00	261.5213	347.0148	200.00	120.00	0.00	263.3763	348.3371
200.00	140.00	0.00	272.1954	357.4137	200.00	140.00	0.00	272.9432	357.1872	200.00	140.00	0.00	272.1954	357.4137
200.00	160.00	0.00	281.2487	366.7314	200.00	160.00	0.00	281.9947	368.0102	200.00	160.00	0.00	281.2487	366.7314
200.00	180.00	0.00	290.5455	376.2997	200.00	180.00	0.00	291.8626	375.3146	200.00	180.00	0.00	290.5455	376.2997
200.00	200.00	0.00	300.0959	386.1290	200.00	200.00	0.00	298.1362	388.3383	200.00	200.00	0.00	294.6623	387.4211
180.00	200.00	0.00	312.1278	377.9196	180.00	200.00	0.00	311.3673	379.1758	180.00	200.00	0.00	312.2952	401.1928
160.00	200.00	0.00	323.8924	369.8925	160.00	200.00	0.00	323.2379	365.9815	160.00	200.00	0.00	323.8924	369.8925
140.00	200.00	0.00	335.3983	362.0418	140.00	200.00	0.00	335.3904	361.6404	140.00	200.00	0.00	335.3983	362.0418
120.00	200.00	0.00	346.6542	354.3619	120.00	200.00	0.00	346.5074	354.8716	120.00	200.00	0.00	346.6542	354.3619
100.00	200.00	0.00	357.6679	346.8470	100.00	200.00	0.00	360.2696	347.9236	100.00	200.00	0.00	357.6679	346.8470
80.00	200.00	0.00	368.4474	339.4921	80.00	200.00	0.00	369.5172	338.1977	80.00	200.00	0.00	368.4474	339.4921
60.00	200.00	0.00	378.9999	332.2920	60.00	200.00	0.00	379.8929	333.6778	60.00	200.00	0.00	378.9999	332.2920
40.00	200.00	0.00	389.3326	325.2419	40.00	200.00	0.00	387.4355	325.3209	40.00	200.00	0.00	389.3326	325.2419
20.00	200.00	0.00	399.4522	318.3372	20.00	200.00	0.00	400.9237	318.4992	20.00	200.00	0.00	399.4522	318.3372
0.00	200.00	0.00	409.3653	311.5734	0.00	200.00	0.00	407.9839	310.0333	0.00	200.00	0.00	409.3653	311.5734
200.00	0.00	40.00	211.8791	279.1739	200.00	0.00	40.00	212.0901	278.1616	200.00	0.00	40.00	211.8791	279.1739
200.00	20.00	40.00	219.6502	286.9701	200.00	20.00	40.00	219.1443	287.6034	200.00	20.00	40.00	219.6502	286.9701
200.00	40.00	40.00	227.6182	294.9635	200.00	40.00	40.00	227.2039	295.1391	200.00	40.00	40.00	227.6182	294.9635
200.00	60.00	40.00	235.7904	303.1620	200.00	60.00	40.00	234.1477	302.5981	200.00	60.00	40.00	235.7904	303.1620
200.00	80.00	40.00	244.1749	311.5734	200.00	80.00	40.00	244.1152	313.2368	200.00	80.00	40.00	244.1749	311.5734
200.00	100.00	40.00	252.7801	320.2062	200.00	100.00	40.00	256.6909	322.4714	200.00	100.00	40.00	252.7801	320.2062
200.00	120.00	40.00	261.6147	329.0691	200.00	120.00	40.00	262.3081	329.9645	200.00	120.00	40.00	261.6147	329.0691
200.00	140.00	40.00	270.6881	338.1716	200.00	140.00	40.00	270.0116	340.1539	200.00	140.00	40.00	270.6881	338.1716
200.00	160.00	40.00	280.0101	347.5235	200.00	160.00	40.00	278.0702	347.8289	200.00	160.00	40.00	280.0101	347.5235

without noise

noisy data 1

noisy data 2

References

- Gady Agam - *CS512: Computer Vision, Introduction to Python, and Using Python with OpenCV*, Department of Computer Science, Illinois Institute of Technology
- Kyle Simek, August 22, 2012, *Dissecting the Camera Matrix, Part 2: The Extrinsic Matrix*
<http://ksimek.github.io/2012/08/22/extrinsic/>
- <https://www.cse.unr.edu/~bebis/CS791E/Notes/CameraParameters.pdf>
- A. Elgammal, Rutgers University, CS 534: *Computer Vision Camera Calibration*
<http://www.cs.rutgers.edu/~elgammal/classes/cs534/lectures/Calibration.pdf>
- Zhengyou Zhang, *CAMERA CALIBRATION*
<http://www.cs.rutgers.edu/~elgammal/classes/cs534/lectures/CameraCalibration-book-chapter.pdf>
- Konstantinos G. Derpanis, May 13, 2010, *Overview of the RANSAC Algorithm*
http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf