EE 417 COMPUTER VISION

ASSIGNMENT #1

CAMERA CALIBRATION

Goktug Korkulu
27026

INTRODUCTION

As the most brief description, I will calibrate a camera from the camera projection matrix in this homework assignment.

I have a calibration rig (3D calibration object) shown below.

I will extract the corners using Harris Corner Detection method and keep at least 30 distinct corner points from each planes on the figure. In other words, 60 distinct corner points in total. The reason for me to do this is to increase the accuracy of the parameter estimation. Then, I will apply some matrix calculations which will be shown in detail on upcoming parts. After I

estimate the projection matrix, I will obtain intrinsic and extrinsic parameters from it.

The purpose of this homework is to understand and investigate the precision of camera calibration using single 3D calibration object.

METHOD

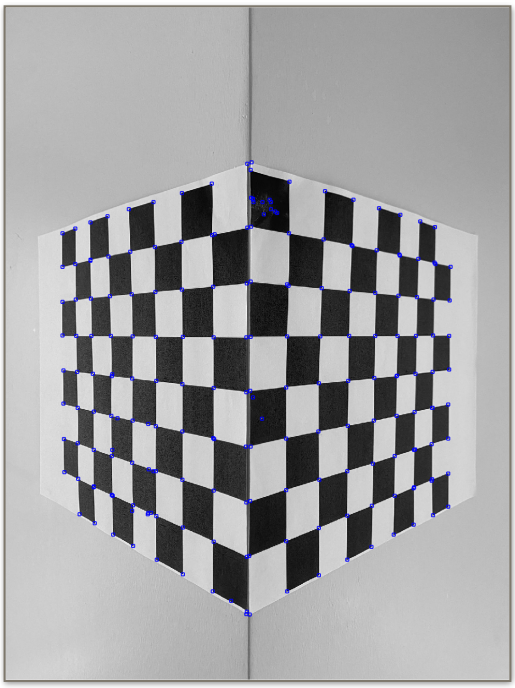
As always, I will start with reading the input image and then converting it to a gray-scaled version. After these operations, I will use ***C = corner(I, 'method')*** built-in MATLAB function which detects corners in image `I` using the specified `method` and returns their coordinates in matrix `C`. Then, I will plot the original image and plot the Harris corners in order to visualize those. Here are the bunch of codes to perform this operations and output figure.

```
img = imread("object.jpeg");

%convert to gray-scale
[~,~,ch] = size(img);
if (ch==3)
    img = rgb2gray(img);
end

img_corners = corner(img, 'harris');

figure;
imshow(img);
hold on;
plot(img_corners(:,1), img_corners(:,2), 's', 'MarkerSize', 4, 'MarkerEdgeColor', 'blue', 'LineWidth', 1);
```



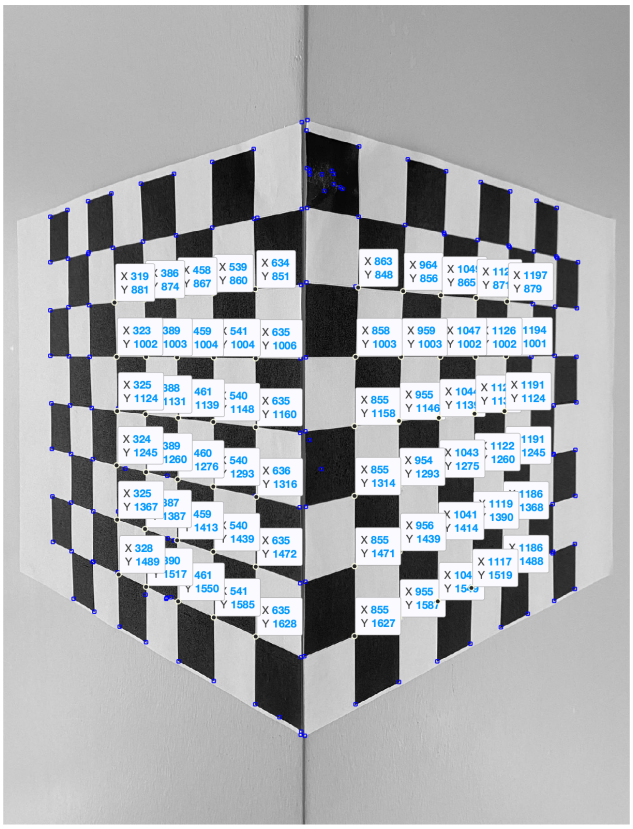
Next step is to pick and keep 30 corner point coordinates from both planes. In other words, in total 60 distinct corner point coordinates will be kept in order to be used to get camera projection matrix later.

```
%% insert image points
%first 30 image coordinates on the right plane
Image_cord_Harris = [855; 1627; 1];
new_corner = [955 ; 1587; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
new_corner = [1042 ; 1549; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
new_corner = [1117 ; 1519; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
new_corner = [1186 ; 1488; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
new_corner = [855 ; 1471; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
new_corner = [956 ; 1439; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
new_corner = [1041 ; 1414; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
new_corner = [1119 ; 1390; 1];
Image_cord_Harris = [Image_cord_Harris new_corner];
```

These are only 9 distinct corners that was put into **Image_cord_Harris** 3x60 array of vector of corner coordinates that was picked manually from the Harris corner detection method output. Below, the variable **Image_cord_Harris**'s elements are shown.

	1	2	3	4	5	6	7	8
1	855	955	1042	1117	1186	855	956	1(
2	1627	1587	1549	1519	1488	1471	1439	1.
3	1	1	1	1	1	1	1	

Note that, although there are only 7 homogenous image coordinates is shown, there are 60 distinct image coordinates exist in this variable and can be seen below and also in the data provided in the homework folder.



These clicked coordinates are the corner points which were chosen to be used in the calibration process.

Later, I will enter the world coordinates which correspond to real world coordinates with respect to origin that is picked the most bottom middle corner. Here is the code for do that:

```
%% insert world frames
World_cord = [];
for z=1:6
    for x=1:5
        new_corner = [x*36 ; 0; z*36; 1];
        World_cord = [World_cord new_corner];
    end
end


for z=1:6
    for y=1:5
        new_corner = [0 ; y*36; z*36; 1];
        World_cord = [World_cord new_corner];
    end
end
```

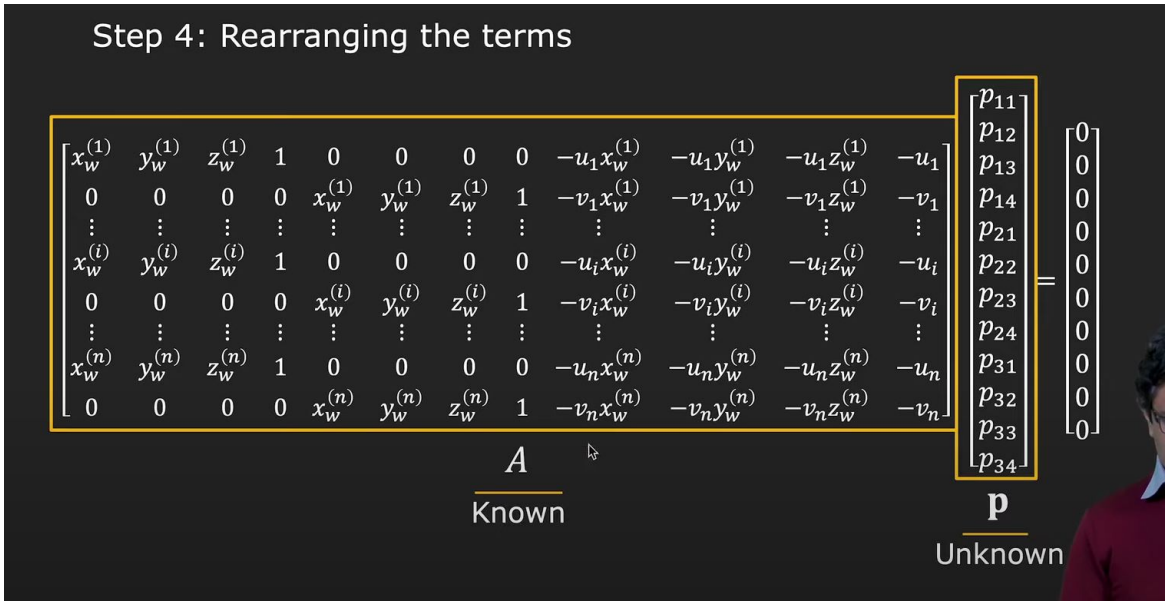
Note that, since the real edge lengths of each squares in the calibration object is 3.6cm, i.e. 36 mm; I have set the world coordinates accordingly.

Word_cord variable is 4x60 array of vectors which keeps the homogenous world frame coordinates correspond to image frame coordinates that were chosen before respectively. I.e. **Word_cord[10]** corresponds to **Image_cord_Harris[10].**

	1	2	3	4	5	6	7	8
1	36	72	108	144	180	36	72	
2	0	0	0	0	0	0	0	
3	36	36	36	36	36	72	72	
4	1	1	1	1	1	1	1	

Here are the first 7 word coordinates, and again; all 60 distinct world coordinates can be seen within the Matlab folder provided in the homework folder.

Now, we are ready to compute the projection matrix since all we need in order to that process exist in our database.



This screenshot is taken from a YouTube video called ‘Camera Calibration ‘from ‘First Principles of Computer Vision’ YouTube channel. Here, we see

that the matrix A should be formed in order to proceed in the process of calculating the projection matrix. Then, let's do that.

```
%% creating matrix A
A = [];
for k = 1:60
    A = [A; transpose(World_cord(:,k)) 0 0 0 0 -Image_cord_Harris(1,k)*transpose(World_cord(:,k));
        0 0 0 0 transpose(World_cord(:,k)) -Image_cord_Harris(2,k)*transpose(World_cord(:,k))];
end
```

For each 60 corner coordinate that are kept in both **Image_cord_Harris** and **Word_cord** variables, we will perform the code inside the for loop. At the end, what we have is 120x12 sized matrix with variable name of **A**.

	1	2	3	4	5	6	7
1	36	0	36	1	0	0	0
2	0	0	0	0	36	0	36
3	72	0	36	1	0	0	0
4	0	0	0	0	72	0	36
5	108	0	36	1	0	0	0
6	0	0	0	0	108	0	36
7	144	0	36	1	0	0	0
8	0	0	0	0	144	0	36
9	180	0	36	1	0	0	0
10	0	0	0	0	180	0	36
11	36	0	72	1	0	0	0
12	0	0	0	0	36	0	72
13	72	0	72	1	0	0	0
14	0	0	0	0	72	0	72
15	108	0	72	1	0	0	0
16	0	0	0	0	108	0	72
17	144	0	72	1	0	0	0
18	0	0	0	0	144	0	72
19	180	0	72	1	0	0	0
20	0	0	0	0	180	0	72
21	36	0	108	1	0	0	0
22	0	0	0	0	36	0	108
23	72	0	108	1	0	0	0
24	0	0	0	0	72	0	108
25	108	0	108	1	0	0	0

This is some portion of this matrix. Now, let's calculate the projection matrix. Solution of this matrix equation can be reached by calculating the eigenvector corresponding two smallest eigenvalue of transpose(A)*A. So;

```
%% eigenvector
[v,d] = eigs(transpose(A)*A);
projection_matrix = v(:,6);
```

Where v is eigenvectors and d is eigenvalues corresponding to those eigenvectors. By applying v(:,6) I obtain the eigenvector corresponding to the smallest eigenvalue. Now, I have projection matrix estimation and it is shown as; Which corresponds to p11, p12, till p34 of the projection matrix.

	1
1	0.4729
2	-0.6653
3	-0.1651
4	-0.0014
5	-0.4485
6	0.2079
7	-0.2492
8	-0.0017
9	6.5688e-05
10	1.6807e-05
11	-2.3491e-04
12	-0.0035

Next step is to extract intrinsic and extrinsic parameter from this camera projection matrix. How are we going to to that process is to utilize MATLAB built-in **[Q,R] = qr(B**) QR decomposition function which performs a QR decomposition on `m`-by-`n` matrix `B` such that `B = Q*R`. The factor `R` is an `m`-by-`n` upper-triangular matrix, and the factor `Q` is an `m`-by-`m` orthogonal matrix. In our case, since B = KR where K is upper triangular intrinsic matrix and R is orthonormal rotation matrix, we should apply qr decomposition to the inverse of B matrix. Here are the codes to get K and R matrix values.

```
%% intrinsic and extrinsic values
B = [projection_matrix(1,1) projection_matrix(2,1) projection_matrix(3,1);
    projection_matrix(5,1) projection_matrix(6,1) projection_matrix(7,1);
    projection_matrix(9,1) projection_matrix(10,1) projection_matrix(11,1)];
[R_inv,K_inv] = qr(A^(-1));

R = R_inv^(-1);
K = K_inv^(-1);
temp = [projection_matrix(4,1); projection_matrix(8,1);projection_matrix(12,1)];
t = (K^(-1))*temp;
```

Note that, B is created by projection matrix's some portion. And therefore K and R are calculated. And at the end, t, which is translation vector is calculated by using K value and the remaining portion of projection matrix.

$$\begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = KR$$

Here all the intrinsic and extrinsic parameters:

K :

	1	2	3
1	-2.3166e-05	-2.3549e-05	-4.2224e-04
2	0	7.4238e-05	-5.4716e-04
3	0	0	1.0000

From K, we can easily say that effective focal lengths are fx = K[1,1], fy = K[2,2], Ox = K[1,3], Oy = K[2,3], and K[2,2] is the skew factor.

R:

	1	2	3
1	-0.4929	-0.7158	0.4946
2	0.7084	-0.6602	-0.2495
3	0.5051	0.2274	0.8326

T:

	1
1	-0.0087
2	-5.2666e-04
3	0.0063

CONCLUSION

Basically, different from Caltech ToolBox implementation; I have used only one 3D calibration object on order to calibrate my camera and obtain intrinsic and extrinsic parameters of it. Here, in this report, you have seen that the processes made are half manual and half using functions. Also, note that when I have gone over the Caltech Toolbox, the processes were less time-consuming since extracting the corner points and calibrating the camera were done by the toolbox itself rather then entering the coordinates on both image and world coordinates entered manually by me. As a result, I might have use Caltech Calibration ToolBox since there are also other kinds of utilities than we can use as well such as getting undistorted images and so on.