University of Girona

Visual Perception

Lab 1

Camera Calibration

Author:
Di MENG

Supervisor:
Pro. Joaquim Salvi

March 13, 2017

1 Introduction

The goal of this lab is to calibrate a simulated camera. Based on the given parameters of camera, we compute the transformation to gain the 2d points in image plane from random 3d points in the world coordinate. Using the 2d and 3d points, we could calibrate the camera by method of Hall and method of Faugeras-Toscani. In addition, check and compare the accuracy against the increase of noise in the image points by two different methods.

2 Part 1

2.1 Step 1: Define the intrinsic parameters and extrinsic parameters with the following values

This initial step consists on declaration and initialization of the variables:

```
%% Parameter preparation
au=557.0943; av=712.9824; u0=326.3819; v0=298.6679;
f=80 ;
Tx=100 ; Ty=0 ; Tz=1500;
Phix=0.8*pi/2; Phiy=-1.8*pi/2; Phix1=pi/5;
```

2.2 Step 2: Get the intrinsic and extrinsic transformation matrices

To obtain intrinsic matrix, we only need the initial variables au,av,u0,v0.

To obtain extrinsic matrix, it is necessary to get the basic rotation and translation matrices firstly. And then to form extrinsic matrix.

```
%% Intrinsic and extrinsic matrix
in_mat = [au, 0, u0, 0; 0, av, v0, 0; 0, 0, 1, 0]; % intrinsic matrix
Rot_x = [1, 0, 0; 0, -cos(Phix), sin(Phix); 0, sin(Phix), cos(Phix)];
Rot_y = [cos(Phiy), 0, sin(Phiy); 0, 1, 0; -sin(Phiy), 0, cos(Phiy)];
Rot_x1 = [1, 0, 0; 0, -cos(Phix1), sin(Phix1); 0, sin(Phix1), cos(Phix1)];

cRw = Rot_x * Rot_y * Rot_x1; % rotation matrix
cTw = [Tx; Ty; Tz];           % tranlation matrix
mat_0 = [0, 0, 0];
ex_mat = [cRw, cTw; mat_0, 1];% extrinsic matrix
```

The results of intrinsic and extrinsic matrices are as follows:

$$\textbf{Intrinsic matrix :} \quad \begin{matrix} 557.0943 & 0 & 326.3819 & 0 \\ 0 & 712.9824 & 298.6679 & 0 \\ 0 & 0 & 1.0000 & 0 \end{matrix}$$

$$\textbf{Extrinsic matrix :} \quad 1.0e+03 * \begin{pmatrix} -0.0010 & -0.0002 & -0.0003 & 0.1000 \\ 0.0003 & -0.0003 & -0.0009 & 0 \\ 0.0001 & -0.0009 & 0.0003 & 1.5000 \\ 0 & 0 & 0 & 0.0010 \end{pmatrix}$$

2.3 Step 3: Define a set of 3D points in the rang [-480:480;-480:480;-480:480].

Before defining the 3d points, it is convenient to define a variable of the numbers of points and initialize it with 6. And then generate a set of 3d points in the range [-480:480;-480:480;-480:480];

```
%% Generate 3d points
pt_set_3d = rand(pt_num,3)*960-480; % range[-480:480;-480:480;-480:480]
ones_pt_num = ones(pt_num,1);
pt_set_3d = [pt_set_3d ones_pt_num];
disp('3d points : ');
disp(pt_set_3d);
```

When the number of points is 6, the random 3d points are as follows:

		X	Y	Z	1
	1:	-235.1087	-346.9205	-246.2160	1.0000
	2:	5.7188	-336.6778	412.0931	1.0000
3d points :	3:	191.1137	-232.7921	-144.0156	1.0000
	4:	375.2671	327.0886	-291.2686	1.0000
	5:	440.9198	-235.8891	-238.9595	1.0000
	6:	45.3269	301.7134	111.4029	1.0000

2.4 Step 4: Compute the projection on the image plane

2d points in the image plane can be obtained by transforming 3d points using transformation matrix which is the result of multiplying intrinsic and extrinsic matrices. The results should be normalized by dividing the third component to gain only x and y values.

```
%% Projection: 2d points
pt_set_2d = zeros(pt_num,3);
for n = 1:pt_num
    pt_set_2d(n,:) = in_mat * ex_mat * [pt_set_3d(n,1); pt_set_3d(n,2);
        pt_set_3d(n,3);pt_set_3d(n,4)];
    pt_set_2d(n,1) = pt_set_2d(n,1)/pt_set_2d(n,3); % normalization
    pt_set_2d(n,2) = pt_set_2d(n,2)/pt_set_2d(n,3);
end
pt_set_2d(:,3) = [];  % delete the 3rd components
disp('2d points :');
disp(pt_set_2d);
```

When the number of points is 6, the projected 2d points are as follows:

		X	Y
	1:	471.0936	403.4304
	2:	341.4340	196.3359
2d points :	3:	325.2381	405.4386
	4:	206.7716	477.3427
	5:	254.8285	473.6992
	6:	314.9543	200.2162

The obtained 2d points are float values, which are with subpixel preci-

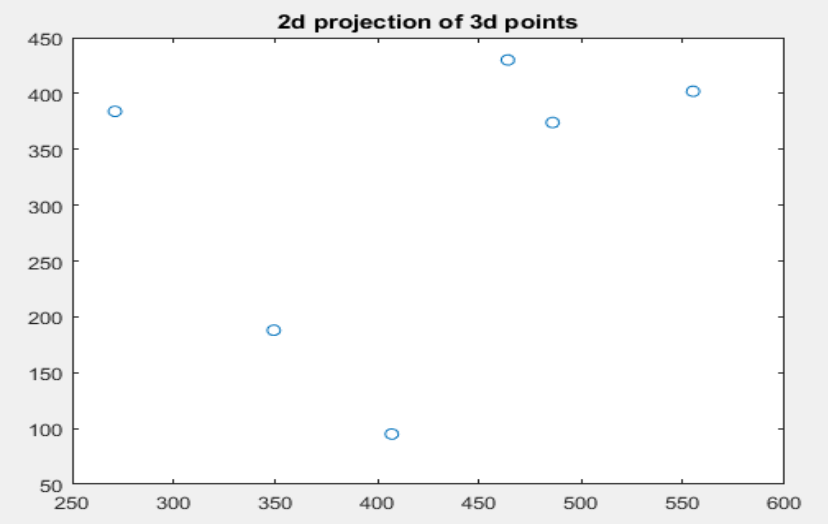
sions.

2.5 Step 5: Draw the 2d points on the image plane

The 2d points we got in the former step are float values. In order to draw them on the image plane, we round them to adjust pixel accuracy before drawing.

```
%% Draw 2d points
pt_set_2d_pxl = round(pt_set_2d);
figure;
plot(pt_set_2d_pxl(:,1),pt_set_2d_pxl(:,2),'o');
title('2d projection of 3d points');
```

And the distribution of points are in the following figure:



From the figure, we can see that the points are well spread. The distribution of points in the image plane will affect the accuracy in the computation. The results would be not accurate or useless if the points are not correctly distributed or in the same position.

2.6 Step 6: Compute the transformation matrix using the method of Hall

From the theory, the model of method of Hall shows that the 2d points are equal to the product of transformation(A) and 3d points. Firstly A matrix should be normalized and fixed by dividing by the end component. After extracting the Q and B vectors, the transformation matrix can be computed as follows:

$$A = (Q^t Q)^{-1} Q^t B$$

```
%% Transformation matrix from Method of Hall
Q = zeros(2*pt_num,11);
B = zeros(2*pt_num,1);
for n = 1:2:(2*pt_num-1)
    Q(n,:) = [pt_set_3d((n+1)/2,1), pt_set_3d((n+1)/2,2), pt_set_3d((n+1)/2,3), ...
              1, 0, 0, 0, 0, (-pt_set_2d((n+1)/2,1)*pt_set_3d((n+1)/2,1)), ...
             (-pt_set_2d((n+1)/2,1)*pt_set_3d((n+1)/2,2)), ...
             (-pt_set_2d((n+1)/2,1)*pt_set_3d((n+1)/2,3))];
    Q(n+1,:) = [0, 0, 0, 0, pt_set_3d((n+1)/2,1), pt_set_3d((n+1)/2,2), ...
               pt_set_3d((n+1)/2,3), 1, (-pt_set_2d((n+1)/2,2)*pt_set_3d((n+1)/2,1)),
               (-pt_set_2d((n+1)/2,2)*pt_set_3d((n+1)/2,2)), ...
               (-pt_set_2d((n+1)/2,2)*pt_set_3d((n+1)/2,3))];
    B(n,:) = pt_set_2d((n+1)/2,1);
    B(n+1,:) = pt_set_2d((n+1)/2,2);
end
A = [inv(Q'*Q)*Q'*B;1];
A = reshape(A, [4,3])';
disp('Transformation matrix from Hall : ');
disp(A);
```

The result of transformation matrix computed by 2d and 3d points is as follows:

Transformation matrix from Method of Hall:

$$\text{A1} = \begin{matrix} -0.3324 & -0.2725 & -0.0229 & 363.5215 \\ 0.1587 & -0.3215 & -0.3702 & 298.6679 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

2.7 Step 7: Compare the matrix obtained in Step 6 to the one defined in step 2.

Based the intrinsic and extrinsic matrices, we can compute the transformation matrix which is the product of intrinsic and extrinsic matrices. The result is as follows:

$$\text{A2} = \begin{matrix} -0.3324 & -0.2725 & -0.0229 & 363.5215 \\ 0.1587 & -0.3215 & -0.3702 & 298.6679 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

The results either from method of Hall or from intrinsic and extrinsic matrices are the same. Because the 2d points we get are using the transformation matrix form intrinsic and extrinsic matrices, and the way to compute method of Hall is actually the opposite procedure.

2.8 Step 8: Add some Gaussian noise to all the 2D points and repeat step6 and step4

Some Gaussian noise are added to the 2d points between the range [-1,+1] pixels for the 95% of points.

```
%% Add noise to 2d points
range_noi = 1;
noise = range_noi*randn(pt_num,2)*0.5;
pt_set_2d_noi = pt_set_2d + noise;
```

Then recompute the transformation matrix using method of Hall, the result is shown below:

$$\text{A\_noise} = \begin{matrix} -0.3518 & -0.2662 & -0.0031 & 362.9981 \\ 0.1517 & -0.3151 & -0.3560 & 295.0917 \\ 0.0000 & -0.0006 & 0.0003 & 1.0000 \end{matrix}$$

And the 2d points obtained by using the transformation matrix above are as follows:

2d points from estimated transformation :

	X	Y
1:	324.1163	343.3607
2:	315.8076	96.6817
3:	287.2427	323.7394
4:	315.0638	329.0324
5:	170.9549	164.6833
6:	212.8384	229.0719

Comparing these 2d points with the ones get in step4, we compute the discrepancy between points.

```
%% Check accuracy computing the discrepancy between points
dist = [];
for n = 1:pt_num
    dist(n) = sqrt((pt_set_2d_n(n,2)-pt_set_2d(n,2))^2 + ...
                   (pt_set_2d_n(n,1)-pt_set_2d(n,1))^2);
end
dist = mean(dist);
disp('Discrepancy between points: ');
disp(dist);
```

The result of discrepancy is 0.5306 approximately. When we increase the number of points, this number will converge and the accuracy will get higher.

2.9 Step 9: Increase the number of 3D points up to 10 points and then up to 50 points and repeat step 8.

When the number of points is 10:

The estimated transformation matrix is:

$$A\_10 = \begin{matrix} -0.3321 & -0.2724 & -0.0284 & 363.7290 \\ 0.1581 & -0.3216 & -0.3729 & 298.8015 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

The discrepancy between points is 0.2571.

When the number of points is 50:

The estimated transformation matrix is:

$$\text{A\_50} = \begin{matrix} -0.3322 & -0.2714 & -0.0232 & 363.4957 \\ 0.1596 & -0.3202 & -0.3704 & 298.4590 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

The discrepancy between points is 0.2353.

As we see, when the number of points is increasing, the estimated matrix is closer to the model transformation matrix and the discrepancy between points is decreasing and approaching to 0.

3 Part 2

3.1 Step 10: Compute X using the points of step 3 and 4, without noise. Extract the camera parameters from vector X.

Similar to the computation of A matrix, we firstly extract the Q and B vectors which are different from the Q and B in step6. Then we can compute X vector using the equation below:

$$X = (Q^t Q)^{-1} Q^t B$$

The result we get is:

$$\text{X} = \begin{pmatrix} -0.3322 \\ -0.2714 \\ -0.0232 \\ 0.0001 \\ -0.0006 \\ 0.0002 \\ 0.1596 \\ -0.3202 \\ -0.3704 \\ 363.4957 \\ 298.4590 \end{pmatrix}$$

From X vectors, we can extract the camera parameters.

```
%% Extract camera parameters
T1 = X(1:3);  T2 = X(4:6);  T3 = X(7:9);
T1 = T1';     T2 = T2';     T3 = T3';
C1 = X(10);   C2 = X(11);
u0_f = (T1*T2')/(norm(T2)^2);
v0_f = T2*T3'/(norm(T2)^2);
au_f = norm(cross(T1',T2'))/(norm(T2)^2);
av_f = norm(cross(T2',T3'))/(norm(T2)^2);
r1_f = norm(T2)/norm(cross(T1',T2'))*(T1-(T1*T2'/(norm(T2)^2))*T2);
r2_f = norm(T2)/norm(cross(T2',T3'))*(T3-(T2*T3'/(norm(T2)^2))*T2);
r3_f = T2/norm(T2);
tx_f = norm(T2)/norm(cross(T1',T2'))*(C1-T1*T2'/(norm(T2)^2));
ty_f = norm(T2)/norm(cross(T2',T3'))*(C2-T2*T3'/(norm(T2)^2));
tz_f = 1/norm(T2);
```

The parameters we get this step are the same as the ones defined in the first step.

3.2 Step 11: Add Gaussian noise to the 2D points and compare the accuracy with two methods with noise

- **Gaussian noise in range [-1,+1]**

  The estimated transformation matrix from Hall:

$$\text{A\_Hall} = \begin{matrix} -0.3334 & -0.2687 & -0.0234 & 363.4676 \\ 0.1584 & -0.3190 & -0.3706 & 298.3944 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

  The discrepancy between points is 0.4241.

  The estimated transformation matrix from Hall:

$$\text{A\_Faugeras} = \begin{matrix} -0.3334 & -0.2687 & -0.0234 & 363.4676 \\ 0.1586 & -0.3213 & -0.3698 & 302.0927 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

  The discrepancy between points is 3.6403.

- **Gaussian noise in range [-2,+2]**

  The estimated transformation matrix from Hall:

$$\text{A\_Hall} = \begin{matrix} -0.3233 & -0.2741 & -0.0211 & 362.9280 \\ 0.1641 & -0.3247 & -0.3646 & 299.7264 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

  The discrepancy between points is 1.0489.

  The estimated transformation matrix from Hall:

$$\text{A\_Faugeras} = \begin{matrix} -0.3233 & -0.2741 & -0.0211 & 362.9280 \\ 0.1635 & -0.3202 & -0.3661 & 292.6371 \\ 0.0001 & -0.0006 & 0.0002 & 1.0000 \end{matrix}$$

  The discrepancy between points is 6.9728.

- **Gaussian noise in range [-3,+3]**

  The estimated transformation matrix from Hall:

$$\text{A\_Hall} = \begin{matrix} -0.2988 & -0.2849 & -0.0124 & 362.3362 \\ 0.1896 & -0.3316 & -0.3651 & 298.3667 \\ 0.0001 & -0.0007 & 0.0002 & 1.0000 \end{matrix}$$

  The discrepancy between points is 1.0513.

  The estimated transformation matrix from Hall:

$$\text{A\_Faugeras} = \begin{matrix} -0.2988 & -0.2849 & -0.0124 & 362.3362 \\ 0.1882 & -0.3251 & -0.3675 & 288.4770 \\ 0.0001 & -0.0007 & 0.0002 & 1.0000 \end{matrix}$$

  The discrepancy between points is 9.2265.

3.3 Comparison and conclusion

From the results we get above, a summary is concluded.

As for method of Hall, the advantage is that it has high accuracy. The 2d points obtained from the 3d points in the world have less discrepancy with the ones from camera parameters' transformation matrix. Its transformation matrix is closer to the camera default transformation matrix. The disadvantage is that this method can not extract camera parameters, what we can do using this method is to get the whole transformation matrix without any other information.

About method of Faugeras, the advantage is that we can not only get the transformation matrix but also extract camera parameters and intrinsic and extrinsic matrices. More information can be obtained from this method. But on the other hand, the accuracy is relatively lower. It appears more discrepancy between points.

4 Part 3