LAB-1

Muhammad Faran Akram	**u1999088**
Solomon Chibuzo Nwafor	**u1999124**

Questions:

Q1: Will the distribution of points in the image affect the accuracy in the computation? (step 6)
Ans: Yes, the distribution of points in the image affects the accuracy of the computation, especially if the points are not well distributed. If the points are collinear or coplanar, it limits the depth information and leads to inaccuracies in the projection matrix estimation. A good distribution of points across the 3D space is necessary to ensure accurate calibration.

Q2: How did the intrinsic parameters change? (step 8)
Ans: The focal length of noisy intrinsic parameters has decreased significantly. There is significant change in principle points and skew parameter is non zero due to noise.

Q3: Why is the axis skew parameter γ different from zero? (step 8)
Ans: As we have added noise in our 2d points therefore the skew parameter γ is not zero anymore. The skew reflects the misalignment between the image axes due to noise.

Part:1:

Step:1:
Description:
We initialized the given intrinsic and extrinsic parameters

Step 2: The projection matrix P
With the intrinsic parameter, Identity matrix, and the transformation matrix (cTw), we computed the projection matrix, P. To get the 3D projection on the camera plane, we divided the last row of P by P, hence, `projection_params`. The result is shown below.

Code:

```
% The projection matrix P
P = K*[1 0 0 0;0 1 0 0;0 0 1 0]*cTw;
%Division of each element of the P with the 3,4 element of the matrix, P
projection_params = P/P(3,4)
```

Result:

```
projection_params = [-0.3324   0.0624   -0.2662   363.5215
                    -0.1207   0.4903    0.1028   298.6679
                     6.3661   0.00039  -0.0005   1]
```

Step 3: Define a set of 3D points in the range [-480:480; -480:480; -480:480]
We generated a set of 3D 6 points using the range, and the result is given below.
p3d = -480+rand([3,6])*(480-(-480))

p4d = [p3d; ones(1, 6)];

Result:

```
p4d = [
[ 107.3489,  282.6464, -313.1714,  252.4739,  454.3469, 41.5587,    33.5065, -218.4274, -473.9290, -203.4846,
-140.0203,   49.8494,   252.3665, -350.6375],
 [ -360.9460,  -87.7183,  424.8824,  242.7074, -441.5125, -447.4775, 159.8961,  431.6822, -384.6453,  -88.6832,
 408.7763, -306.2738,   471.1649,  230.4429],
 [ 249.1483,  426.0456,  -33.1299,  461.9615, -464.6676,  151.1853, -56.3129,  219.1387,  126.4246,  270.2699,  211.6340,
-86.7923,  -409.2937,  320.4301],
 [  1.0000,    1.0000,    1.0000,    1.0000,    1.0000,    1.0000,  1.0000,    1.0000,    1.0000,    1.0000,    1.0000,    1.0000,
    1.0000,    1.0000]
]
```

Step 4: `The projection of the 3D 6 points on the image plane by using P`

From step 3, we created a function that reformulates the 3D projection we obtained in step 3 to 2D by dividing the first two rows of the projection matrix (u and v) by w.

Code:

```
norm_projection_2d_6 = points_2d_gen(p4d_6, projection_params);
```

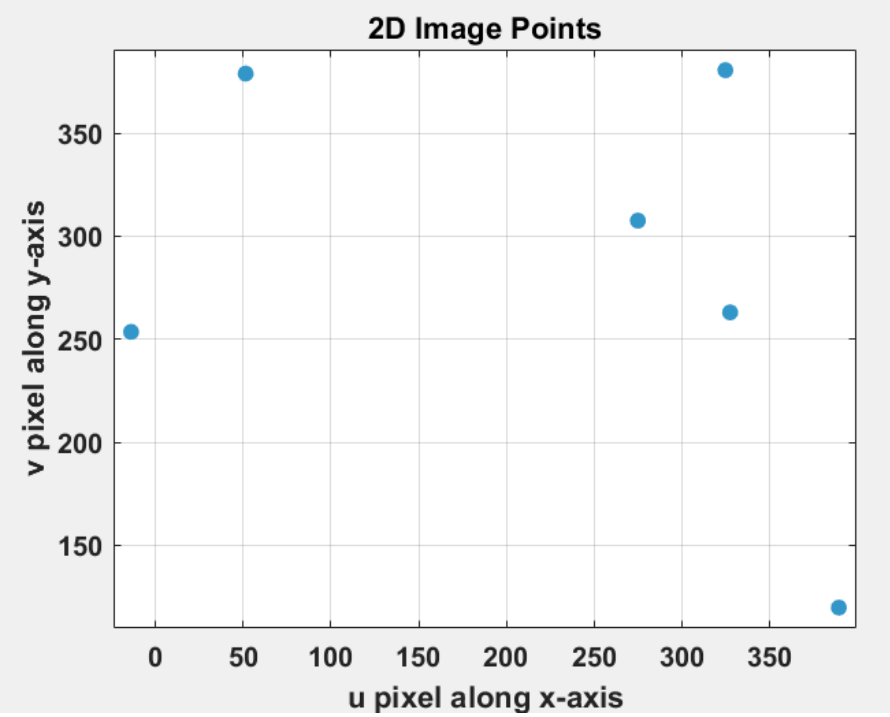
Results:

```
norm_projection_2d_6 = [
119.790771096182      307.571525275868      253.578429072301      380.568329138211      263.046066466280
378.925715183132
389.131832073085      274.708191849124      -13.7905694461522     324.529680216974      327.222049317614
51.4197257826262 ]
```

Step 5: `The 2D plot of the 6 points`

The 2D points generated in step 4 were plotted as shown in the plot below. As expected, the 6 points were in the bounded range.

Plot:



Step 6: `Estimation of the 3x4 projection parameter matrix using Hall method`

We created a function for the Hall method implementation and found the 11 unknown projection parameters and appended one known parameter to get a 3x4 matrix as shown in the results.

```
projection_2d_Q6=estimate_params(p4d_6, norm_projection_2d_6);
```

Results:

```
projection_2d_Q6 = [-0.332440977389380        0.0623705255265161   -0.266219007970205    363.521519999998
                    -0.120680015546131        0.490343478823164    0.102838669991938    298.667900000005
                     6.36610018750735e-05     0.000397783421925063      -0.000531187415632188          1]
```

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

The projection parameters we got using the Hall method were compared to the projection parameters we obtained in spet 2 using this function: `proj_params_err_1 = mean(sqrt(sum(projection_params - projection_2d_Q6).^2))`. From our observation, the matrix we obtained in Step 6 and the one we obtained in Step 2 are the same with an error of 9.1535e-13, which is very small (not noticeable).

We thereafter called *get_intrinsics_from_proj_matrix* to get the K and cRw for 6 points as:

```
[K_Q6, cRw_Q6] = get_intrinsics_from_proj_matrix(projection_2d_Q6);
```

Results: 9.1535e-13

Part:2:

Step 8: Comparison of a noisy projection matrix you obtain with the one you got in step 6

We added a gaussian noise to the projection parameters obtained with Hall in Step 6 using a *gaussian noise* we created. We then used the estimation function we created in Step 6 to the noisy projection parameters as follows:
projection_noise_param_6 = gaussian_noise(norm_projection_2d_6);
est_projection_noise_Q6 = estimate_params(p4d_6,projection_noise_param_6);

Thereafter, we compared the *est_projection_noise_Q6* and that of Step 6 as:
proj_err_6 = mean(abs(sum(projection_2d_Q6-est_projection_noise_Q6)))

Our projection error obtained was **0.1911**.

```
%Extract the intrinsic parameter matrix K, and the camera rotation matrix %Rw%c%from the camera projection you just computed.
[K_Q6_noise, cRw_Q6_noise] = get_intrinsics_from_proj_matrix(est_projection_noise_Q6);
```

Results:

K_Q6_noise =

Variables - K_Q6_noise

3x3 double

	1	2	3
1	559.1411	-9.4562	308.5645
2	0	711.4413	318.2206
3	0	0	1

cRw_Q6_noise =

Variables - cRw_Q6_noise

3x3 double

	1	2	3
1	-0.9467	-0.1543	-0.2827
2	-0.2965	0.7601	0.5782
3	0.1257	0.6312	-0.7654

est_projection_Q6 =

Variables - est_projection_Q6

3x4 double

	1	2	3	4
1	-0.3324	0.0624	-0.2662	363.5215
2	-0.1207	0.4903	0.1028	298.6679
3	6.3661e-05	3.9778e-04	-5.3119e-04	1

est_projection_noise Q6 =

Variables - est_projection_noise_Q6

3x4 double

	1	2	3	4
1	-0.3248	0.0675	-0.2662	361.5366
2	-0.1138	0.4939	0.1117	299.9205
3	8.3699e-05	4.2033e-04	-5.0968e-04	1

Proj_err_6 = **0.1911**

Step 9: Now compute the 2D points with the projection matrix and compare them to those obtained in step 4

We compared the estimated projection parameters with noise we obtained in Step 8 with the one we obtained in noise free parameters we obtained in Step 6 and got an difference of 0.529076725681595 using this function:
proj_err_p6 = mean(sqrt(sum(norm_projection_2d_6-proj_2d_6Q_noise).^2))
Also, we made a function for computing the following: mean projection error, K_error, and cRw_error (magnitude and angle (phi)).

Results:

Proj_err_p6 =	**0.529076725681595**

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

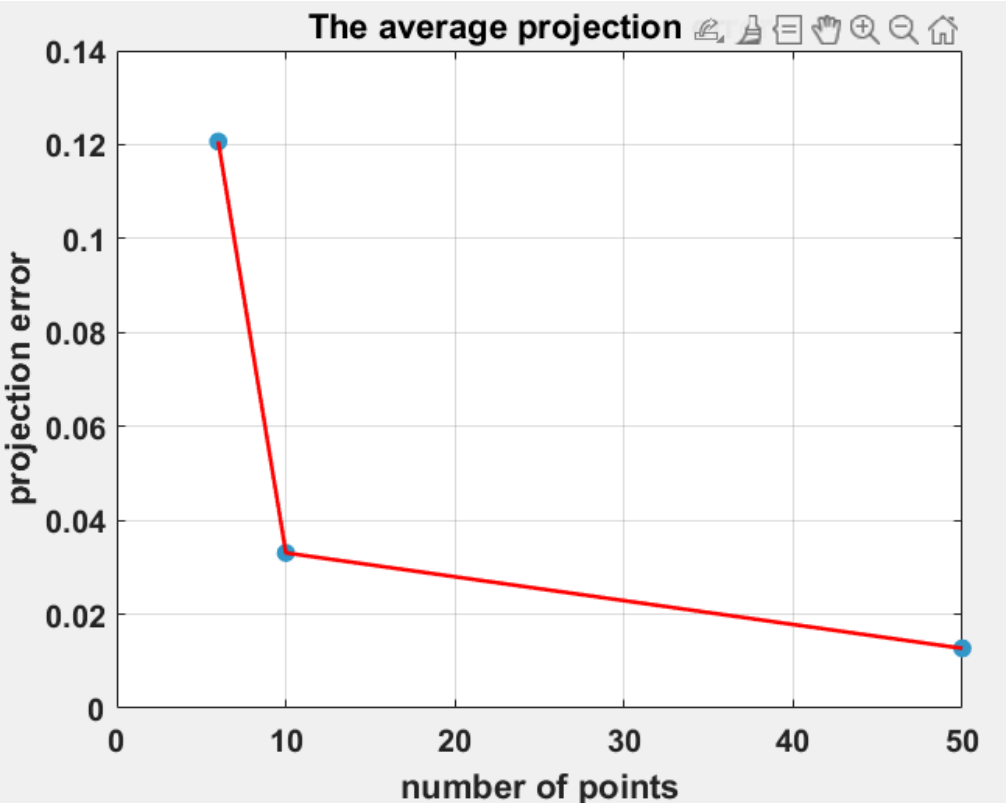
We made a loop that increases the number of the 3D points from 6, 10, and 50 points and computes the projection errors of those point. From the plot, it is evident that the error reduces as we increase the number of points.

```
mean_err = [];
mag_err = [];
phi_err = [];
proj_err = []
points = [6,10,50];
for i =points
    p4d = points_3d_gen(i);
    projections = points_2d_gen(p4d, projection_params);
    est_proj_param_org = estimate_params(p4d,projections)
    projection_noise_param = gaussian_noise(projections);
    est_proj_params_noise = estimate_params(p4d,projection_noise_param)
    [mean_err_Ki, dmag_i, phi_i] = error(est_proj_param_org, est_proj_params_noise);
    proj_err_i = mean(sqrt(sum(est_proj_param_org-est_proj_params_noise).^2))
    mean_err(end+1) = mean_err_Ki;
    mag_err(end+1) = dmag_i;
    phi_err(end+1) = phi_i;
    proj_err(end+1) = proj_err_i;
end
```

Results:

	Points = 6	Points = 10	Points = 50
mean_err =	17.8178	5.0837	1.1729
mag_err =	0.1192	0.0265	0.0033
phi_err =	0.0596	0.0132	0.0016
proj_err =	0.1207	0.0331	0.0127

Plot:


The average projection
projection error
number of points
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
10
20
30
40
50