Assignment 1: Camera Calibration

Linjun Wang(5214513), Zhenyu Liu(5386586), Ziyan Wu(5360684)

1. Result Show

All result files are stored in our Github repository:
https://github.com/Ziyan-Wu/geo1016

1.1 our data

Our point pairs:

```
1 0 11<=>357 609  11 0 1<=>733 461  11 1 0<=>738 421  1 11 0<=>425 174
0 11 4<=>337 231  0 4 11<=>292 523  4 0 11<=>425 630  11 0 4<=>701 520
11 4 0<=>724 352  4 11 0<=>494 170  1 0 1 <=>499 429   2 0 2<=>508 447
```

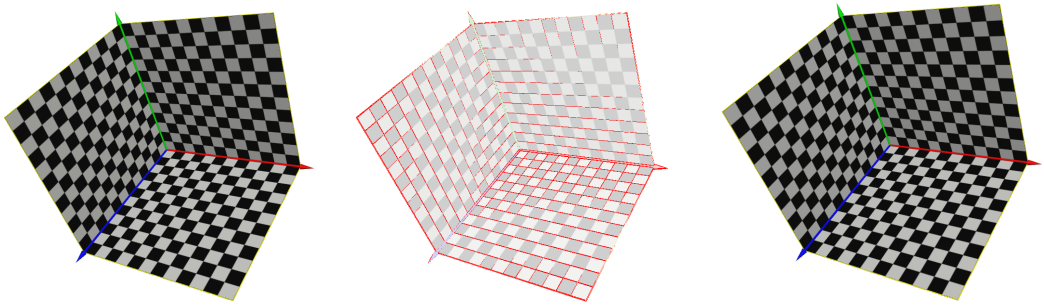


Figure1: Comparison between our input image and our result image

1.2 provided data

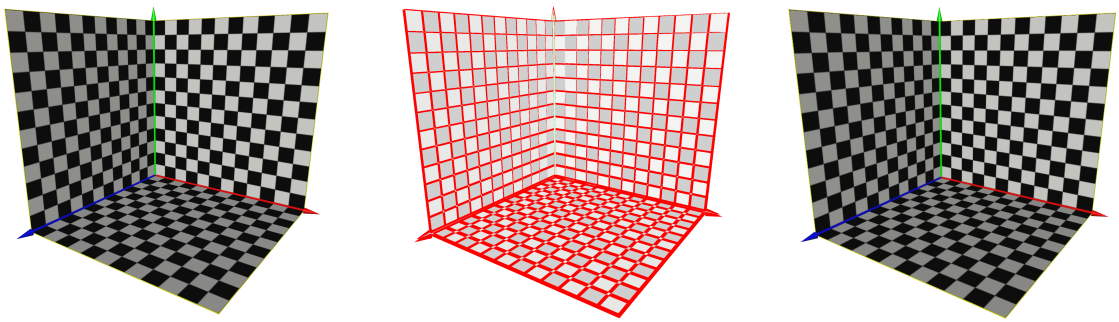


Figure2: Comparison between input image and result image(6 points)

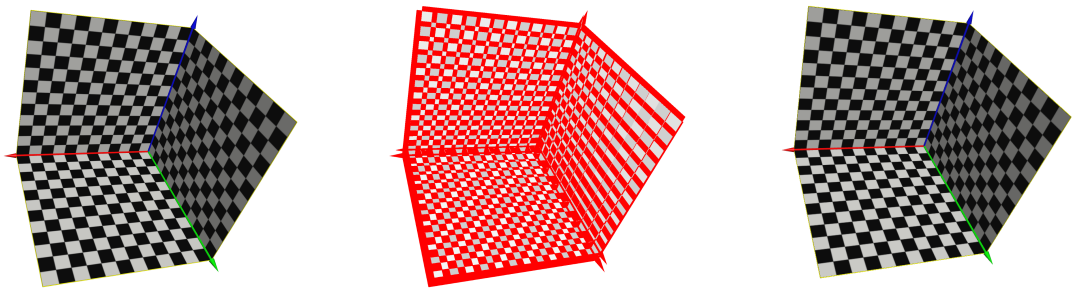


Figure3: Comparison between input image and result image (9 points)

2.Verify Intermediate Result

We mainly use the properties of the image coordinates and matrix to perform some calculations to determine whether the intermediate value is correct. In this section, we use the provided sample data (6 points) as an example to show the results of verification. The provided data (6 points) used as an example to illustrate this part.

2.1 Check the result of Matrix P

We outputted Matrix P to verify if its element values are correct:

```
the correspondences loaded from the file are:
        0: (2 4 0) <-> (1021 648)
        1: (6 9 0) <-> (1282 308)
        2: (5 0 10) <-> (734 1226)
        3: (3 0 12) <-> (459 1250)
        4: (0 4 9) <-> (458 766)
        5: (0 10 7) <-> (550 233)
TODO: I am going to implement calibration() ...
P:
size: 12 by 12
2       4       0       1       0       0       0       0       -2042   -4084   0       -1021
0       0       0       0       2       4       0       1       -1296   -2592   0       -648
6       9       0       1       0       0       0       0       -7692   -11538  0       -1282
0       0       0       0       6       9       0       1       -1848   -2772   0       -308
5       0       10      1       0       0       0       0       -3670   0       -7340   -734
0       0       0       0       5       0       10      1       -6130   0       -12260  -1226
3       0       12      1       0       0       0       0       -1377   0       -5508   -459
0       0       0       0       3       0       12      1       -3750   0       -15000  -1250
0       4       9       1       0       0       0       0       0       -1832   -4122   -458
0       0       0       0       0       4       9       1       0       -3064   -6894   -766
0       10      7       1       0       0       0       0       0       -5500   -3850   -550
0       0       0       0       0       10      7       1       0       -2330   -1631   -233
```

Figure4: Output of point pairs and matrix P

2.2 Check result of SVD decomposition

- First check U, U is orthogonal, so $U \cdot U^T$ must be identity.

```
std::cout << "U * U^T: \n" << U * transpose(U) << std::endl;
```

```
U*U^T:
size: 12 by 12
1       0       0       0       0       0       0       0       0       0       0       0
0       1       0       0       0       0       0       0       0       0       0       0
0       0       1       0       0       0       0       0       0       0       0       0
0       0       0       1       0       0       0       0       0       0       0       0
0       0       0       0       1       0       0       0       0       0       0       0
0       0       0       0       0       1       0       0       0       0       0       0
0       0       0       0       0       0       1       0       0       0       0       0
0       0       0       0       0       0       0       1       0       0       0       0
0       0       0       0       0       0       0       0       1       0       0       0
0       0       0       0       0       0       0       0       0       1       0       0
0       0       0       0       0       0       0       0       0       0       1       0
0       0       0       0       0       0       0       0       0       0       0       1
```

Figure5: Output of $U \cdot U^T$

- Second check V, V is orthogonal, so $V \cdot V^T$ must be identity.

```
std::cout << "V: \n" << V * transpose(V)<< std::endl;
```

```
V*V^T:
size: 12 by 12
1       0       0       0       0       0       0       0       0       0       0       0
0       1       0       0       0       0       0       0       0       0       0       0
0       0       1       0       0       0       0       0       0       0       0       0
0       0       0       1       0       0       0       0       0       0       0       0
0       0       0       0       1       0       0       0       0       0       0       0
0       0       0       0       0       1       0       0       0       0       0       0
0       0       0       0       0       0       1       0       0       0       0       0
0       0       0       0       0       0       0       1       0       0       0       0
0       0       0       0       0       0       0       0       1       0       0       0
0       0       0       0       0       0       0       0       0       1       0       0
0       0       0       0       0       0       0       0       0       0       1       0
0       0       0       0       0       0       0       0       0       0       0       1
```

Figure6: Output of $V \cdot V^T$

- Third, check S, S must be a diagonal matrix.

```
std::cout << "S: \n" << S << std::endl;
```

```
S:
size: 12 by 12
25017.6 0       0       0       0       0       0       0       0       0       0       0
0       15349.6 0       0       0       0       0       0       0       0       0       0
0       0       5982.09 0       0       0       0       0       0       0       0       0
0       0       0       694.773 0       0       0       0       0       0       0       0
0       0       0       0       20.8112 0       0       0       0       0       0       0
0       0       0       0       0       13.8309 0       0       0       0       0       0
0       0       0       0       0       0       7.13171 0       0       0       0       0
0       0       0       0       0       0       0       3.73541 0       0       0       0
0       0       0       0       0       0       0       0       2.38433 0       0       0
0       0       0       0       0       0       0       0       0       1.07406 0       0
0       0       0       0       0       0       0       0       0       0       0.296941        0
0       0       0       0       0       0       0       0       0       0       0       6.65542e-05
```

Figure7: Output of matrix S

- Forth, check whether the value of $U \cdot S \cdot V^T$ equals P or not, the result should be a 0 Matrix.

```
std::cout << "P - U * S * V^T: \n" << P - U * S * transpose(V) << std::endl;
```

```
P - U * S * V^T:
size: 12 by 12
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0       0       0       0       0       0
```

Figure8: Output of $P - U \cdot S \cdot V^T$

2.3 Check Projection Matrix M

According to formula $Point_{2D} = M \cdot Point_{3D}$, we use M multiple $Point_{3D}$ to check the result whether it equals $Point_{2D}$. It can be seen from table 1 that the calculated pixel coordinate value is very close to the initial value, and the error is basically controlled within 0.1.

```
//check if the M is correct
for (int i = 0; i < points_3d.size(); i++) {
   Matrix<double> pt(4, 1, 0.0)
   pt[0][0] = points_3d[i][0];
   pt[1][0] = points_3d[i][1];
   pt[2][0] = points_3d[i][2];
   pt[3][0] = 1;
   Matrix<double> multi = M * pt;
   std::cout << "\t" << i << ": ("
   << points_3d_[i] << ") <-> (" << multi[0][0] / multi[2][0]
   << " " << multi[1][0] / multi[2][0] << ")" << std::endl;
}
```

Table1: Coordinates differences between input file and our result

x			y		
initial	compute by M	difference	initial	compute by M	difference
1021	1021	0	648	648.012	-0.012
1282	1282.01	-0.01	308	307.999	0.001
734	733.953	0.047	1226	1225.95	0.05
459	459.051	-0.051	1250	1250.07	-0.07
458	457.994	0.006	766	765.949	0.051
550	549.991	0.009	233	233.016	-0.016

3. Determine the sign of ρ

$T = \rho K^{-1} b$, the sign of ρ should be the same as t_z . t_z is positive when the origin of the world coordinate system is in front of the origin of the camera coordinate system. In our case, the origin of the rig is in front of the camera, which means that t_z should be positive, then we can know that the sign of ρ should also be positive.

4.Accuracy

In section 1.2, we found a strange phenomenon that the accuracy of 9 sample points is better than that of 6 points. We have analyzed the following three reasons that affect the accuracy, and applied them to our point pairs to obtain a higher accuracy (see Figure1).

4.1 The point pair matching accuracy

The main reason for the poor accuracy of the 9 points data is the bad matching accuracy of the point pairs, thus introducing errors (see figure 9). That's why even though it has more points, it doesn't have better accuracy.

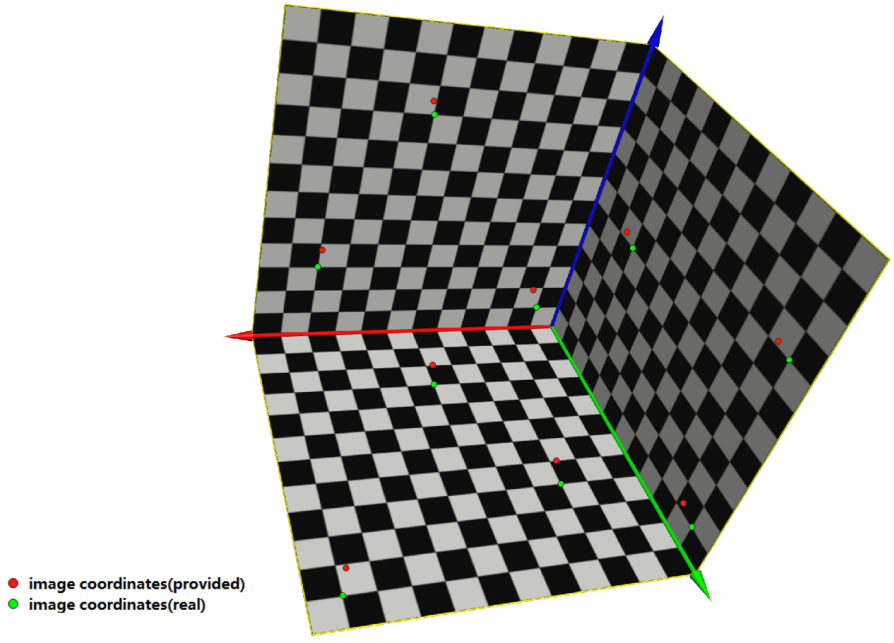


Figure9: Matching error in provided sample data (9 points)

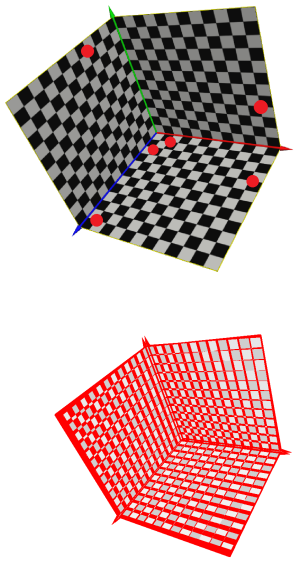
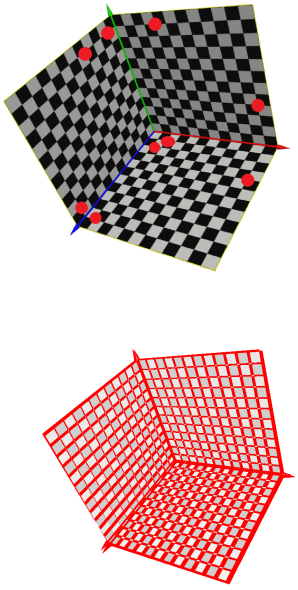
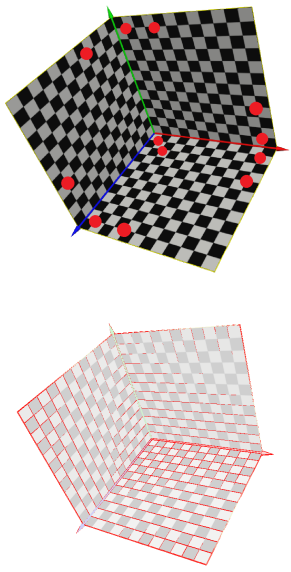
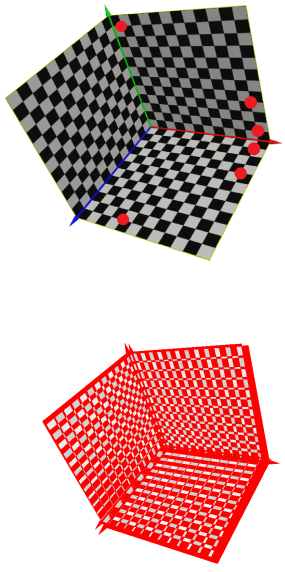
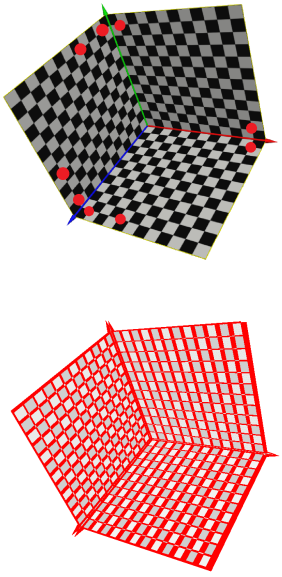
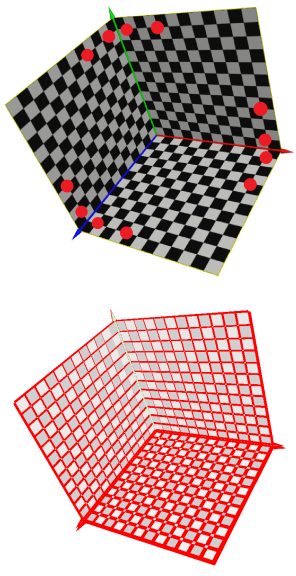
4.2 Number of point pairs

From table2, other things equal, more pairs of points means better accuracy.

4.3 Distribution of sample points

From table2, other things equal, the more uniform distribution of sample points in the image means better accuracy.

Table2: The influence of distribution and number of sample points on accuracy

	6 points	9 points	12 points
uniform distribution			
nonuniform distribution			

Contributions

In order to better understand the camera calibration process, each of us completed the code independently this time, and then combined the good parts of the code into the final result.

Linjun Wang(33%)
-construct the P Matrix
-solve the Matrix M
-result show part in the report
-determine the sign of ρ in the report

Zhenyu Liu(34%)
-collect the 3D-2D corresponding points
-extract intrinsic parameters from M
-the accuracy part in the report

Ziyan Wu(33%)
-extract extrinsic parameters from M
-result show part in the report
-intermediate result verification in the report