1 Direct Linear Transform

```
Optimization terminated successfully    (Exit mode 0)
            Current function value: 0.06030438888592402
            Iterations: 4
            Function evaluations: 57
            Gradient evaluations: 4
Reprojection error after optimization: 0.06030438888592402
K=
[[ 2.871e+03 -2.010e+01  1.487e+03]
 [ 0.000e+00  2.876e+03  9.863e+02]
 [ 0.000e+00  0.000e+00  1.000e+00]]
R =
[[-0.776  0.631 -0.013]
 [ 0.313  0.367 -0.876]
 [-0.548 -0.684 -0.482]]
t = [[0.049 0.039 3.452]]
```

Figure 1: Camera Parameters

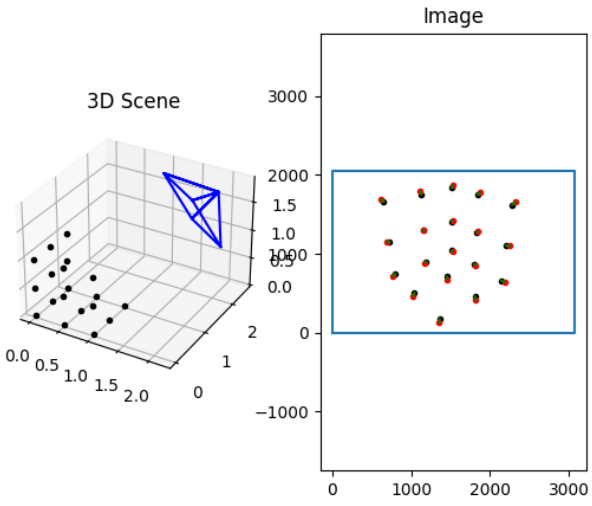

Figure 2: Calibration camera

1.1 Normalize

If we do not normalize the points we can have the following problem: It is enough that the numbers are too small or too big and the algorithm can be numerically unstable.

1.2 DLT

We need 6 points to solve the system. Each point therefore reduces the degree of freedom by two in the system of equations. Hence you can add two constraints per points if the points are not colinear with other points.

1.3 Optimizing reprojection error

The algorithm that reduces reprojection is iterative and therefore reduces the size of the error over time.

1.4 Denormalizing the projection matrix

The matrix is denormalized in the code as shown in the slides. There is nothing special to say for this section.

1.5 Decomposing the projection matrix

Looking at the photo, I see that the error is small. I think we can say that it is acceptable for the following reason. There are in any case some stability errors and the algorithm has to solve some SVD which can cost a little in terms of accuracy.

2 Structure from Motion

2.1 Essential matrix estimation

To build this matrix I used the eight points algorithm which is on wikipedia. Then I used the SVD decomposition to increase the stability of the problem and the problem passes the assertions. So we can say that this part works without problems.

2.2 Point triangulation

The filtering step was not that complicated. I transformed each point using the camera. Then it's a simple filtering step. Each point behind one of the two cameras is not valid.

2.3 Correct decomposition

To find the correct decomposition, simply test the 4 relative poses. My solution worked but I had the cameras looking in the wrong direction. So I changed the identity camera and the

pose works without any problem. Then I visualized a photo to see and everything works without any problem. To find the correct decomposition, simply test the 4 relative poses. My solution worked but I had the cameras looking in the wrong direction. So I changed the image with the identity camera and the pose works without any problem. Afterwards I have

2.4 Absolute pose estimation

There is nothing special for this step

2.4 Map extension

I spent quite a bit of time figuring out how this feature works. I simply filled in the data structure with the points and indexes. I had a lot of trouble with this code. The cameras are not well rendered and it took me a long time to find the right solution. Finally by triangulating the points we see that the points display a fountain. Have a look the last image.

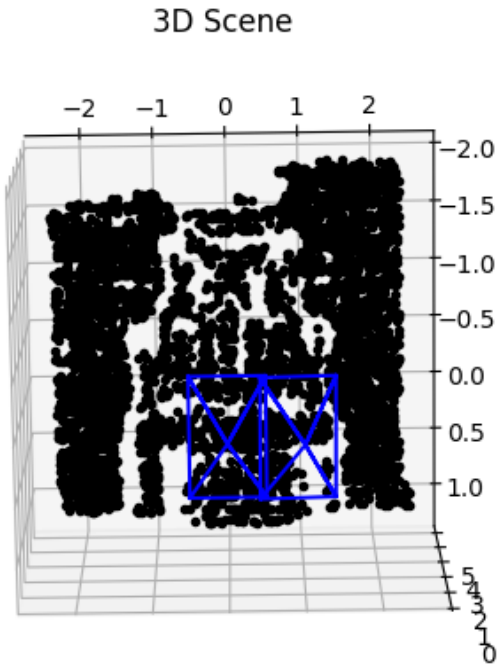

Figure 3: Camera with two poses

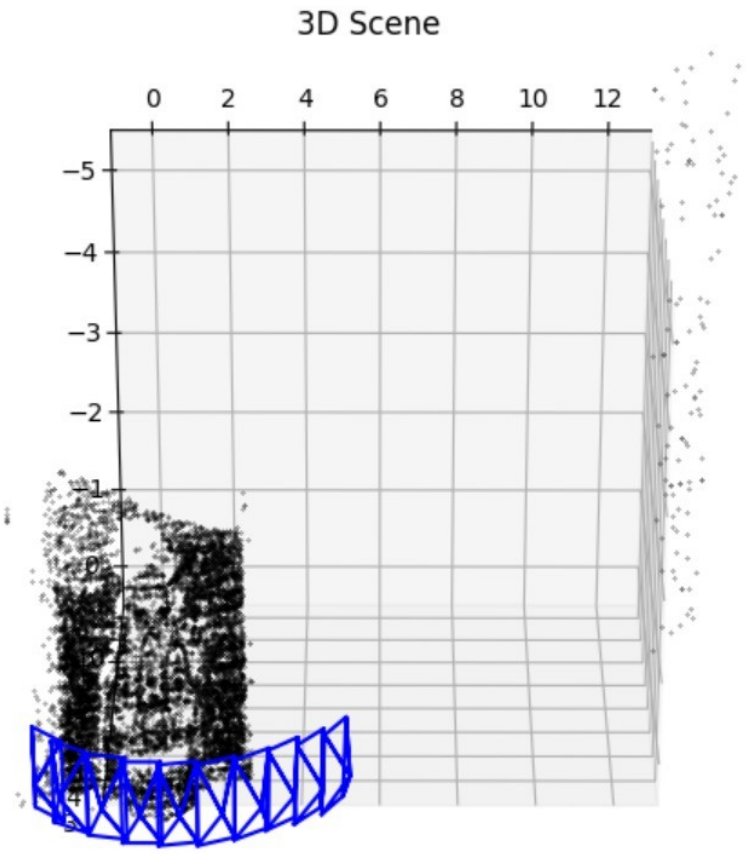

Figure 4: Full SfM