Project 4

3D Reconstruction

Wenxiang He

301417521

Due date: 23:59 Tuesday 12/5th (2023)

(1 Free late-day used, total 0 days remain)

Undergraduate

3.1 Sparse reconstruction

3.1.1 Implement the eight point algorithm (2 pts)

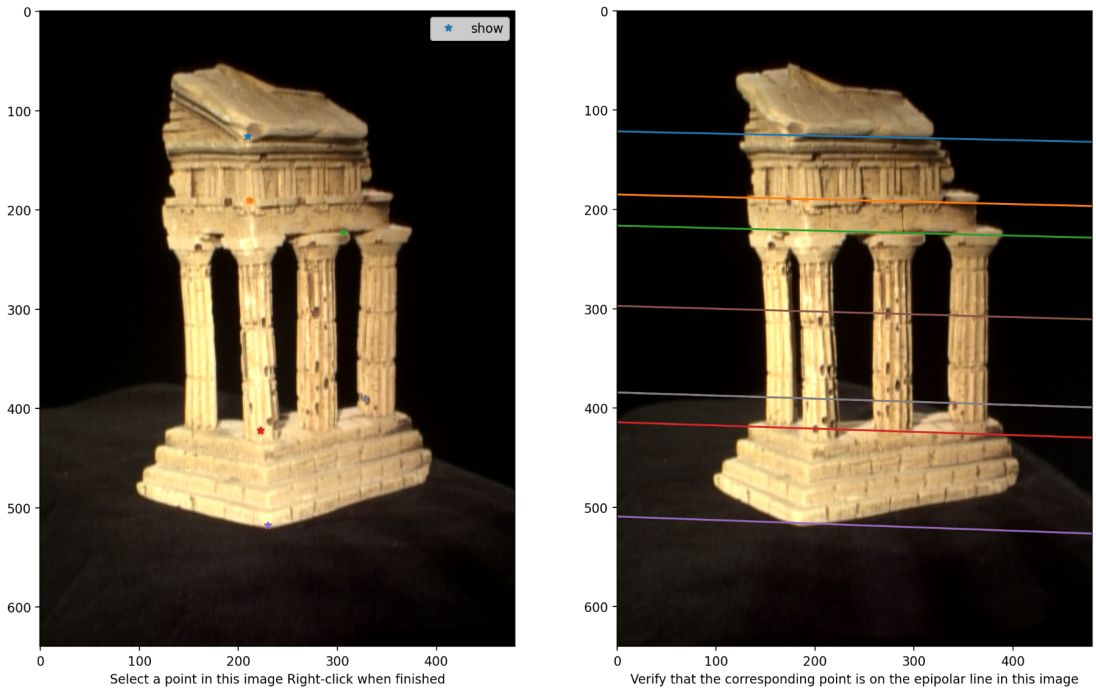
The recovered fundamental matrix F I get is:

```
[[ 1.77324726e-09 -1.70876561e-08 -8.77582202e-06]
 [-6.63558809e-08 -4.05296608e-10  4.95571220e-04]
 [ 1.69307908e-05 -4.75954394e-04 -2.06189981e-03]]
```

After retain six decimal places of precision:

```
[[ 0.000000   -0.000000   -0.000009]
 [-0.000000   -0.000000    0.000496]
 [ 0.000017   -0.000476   -0.002062]]
```

Here is the visualization of epipolar lines:



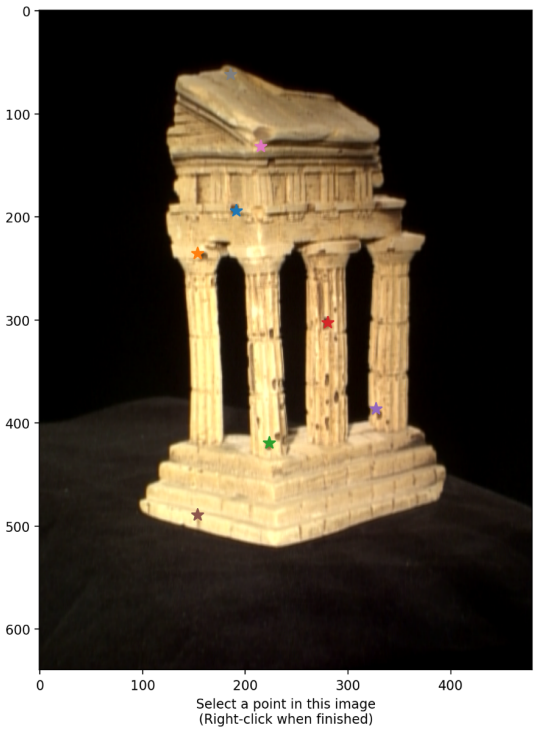
3.1.2 Find epipolar correspondences (2 pts)

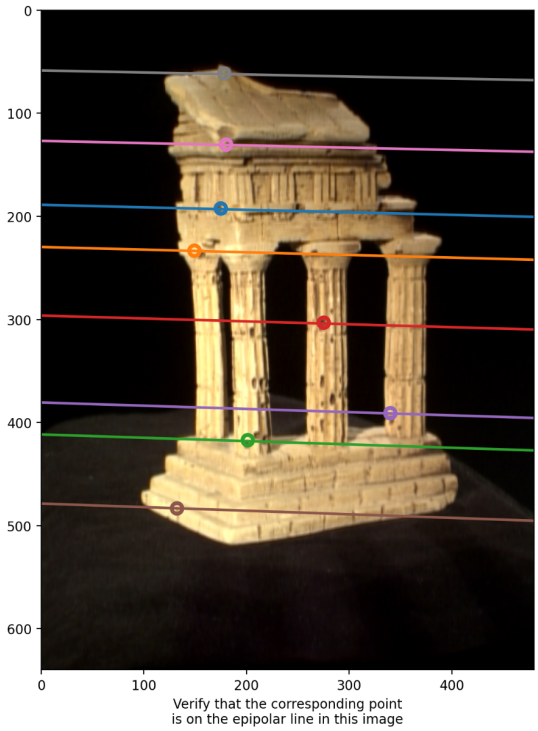
In the epipolarCorrespondence function, I use the Normalized Cross-Correlation (NCC) as the similarity metric for matching points across two images.

To compute the NCC, I first extract windows around the feature points in the first image and their corresponding epipolar lines in the second image. Then, I normalize these windows by subtracting the mean and dividing by the standard deviation to account for any intensity differences.

After normalization, I compute the NCC score by taking the mean of the element-wise product of the two windows. This score represents the similarity between the two patches, where a higher value indicates a better match. I iterate through all possible matching points along the epipolar line in the second image and select the point with the highest NCC score as the corresponding point. By utilizing the NCC, I can ensure that the matching is more reliable and accurate.

For my test cases, there are no cases where matching algorithm consistently fails,





3.1.3 Write a function to compute the essential matrix (2 pts)

The estimated essential matrix E I got is:

$$\begin{bmatrix} 0.00409907 & -0.03964299 & -0.01894139 \\ -0.15394421 & -0.00094368 & 0.72542881 \\ 0.00165055 & -0.73429419 & -0.00084402 \end{bmatrix}$$

After retain six decimal places of precision:

$$\begin{bmatrix} 0.004099 & -0.039643 & -0.018941 \\ -0.153944 & -0.000944 & 0.725429 \\ 0.001651 & -0.734294 & -0.000844 \end{bmatrix}$$

3.1.4 Implement triangulation (2 pts)

After triangulating the points using different candidate matrices, I checked the z coordinates of resulting 3D points, which represent the depth relative to the camera.

The correct external matrix must satisfy the physical constraint that all trigonometric points should be in front of both cameras, which means that when viewed from the perspective of each camera, their z coordinates must be positive. Therefore, the correct external matrix is one that produces only positive z coordinates for all triangulated 3D points.

In my implementation, I run a loop over all the candidate P2 matrices, triangulating each matrix, and checking the resulting 3D points. I use my trigonometric function that uses singular value decomposition (SVD) to find the least square solution of a homogeneous system consisting of the corresponding points in the two images. Once I have found the candidate matrices with all positive z coordinates, I got the correct projection matrix P2, and use for later extract R, t matrices.

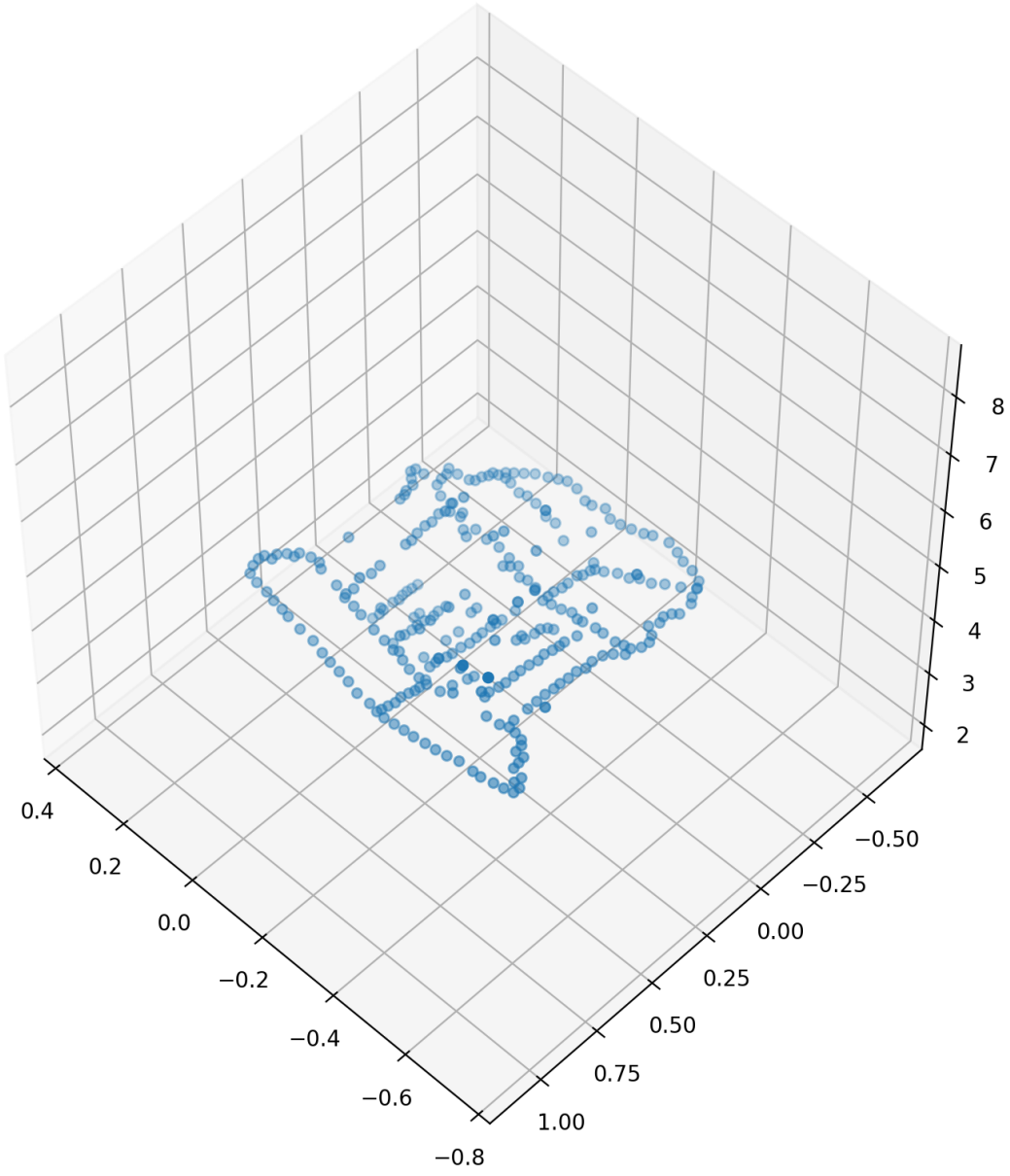
Here is the reprojection error I got:

Re-projection Error of pts1 and pts2: 0.4979656728055408 0.5024626266460707

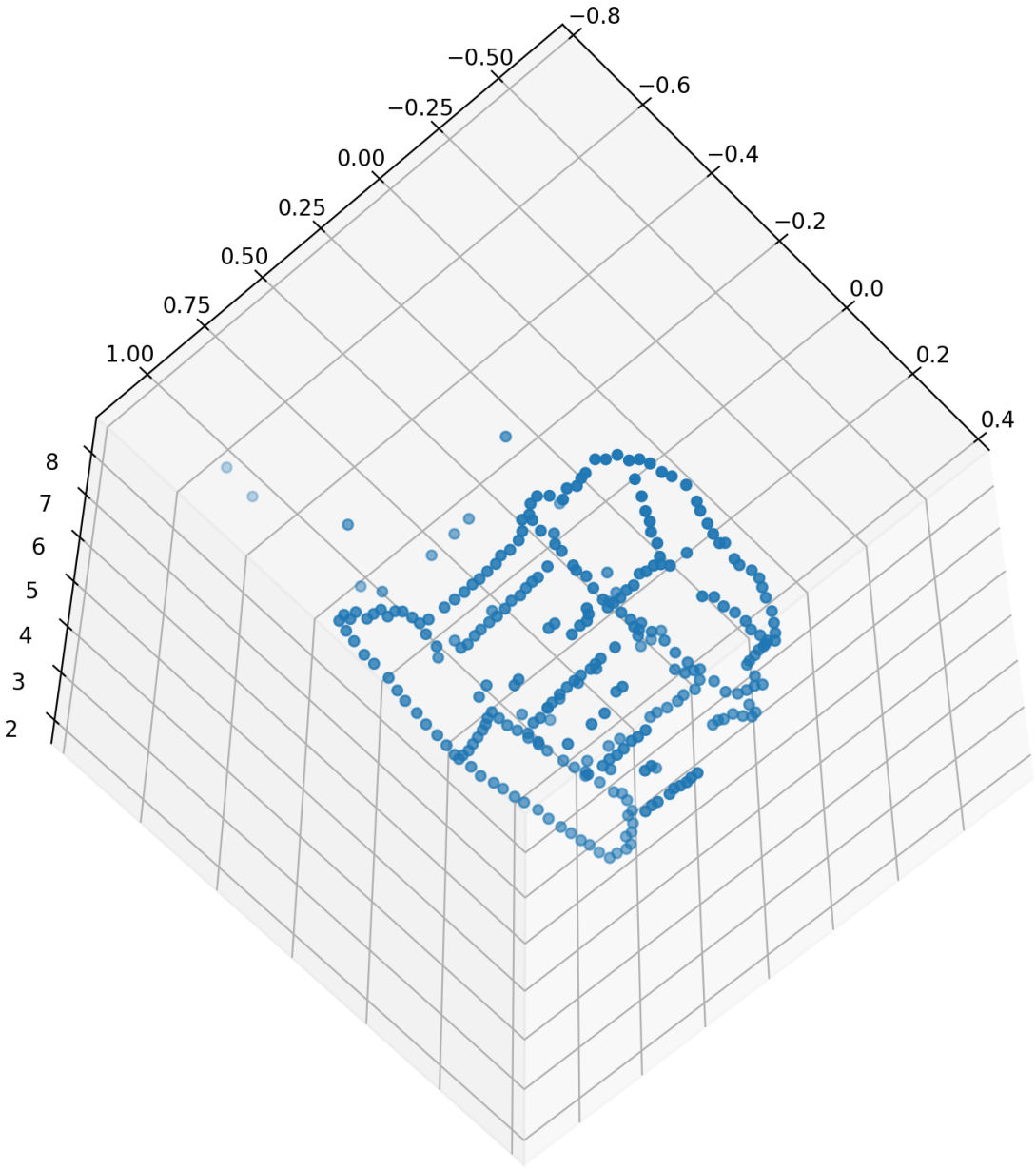
3.1.5 Write a test script that uses templeCoords (2 pts)

3 images of your final reconstruction of the templeCoords points, from different angles

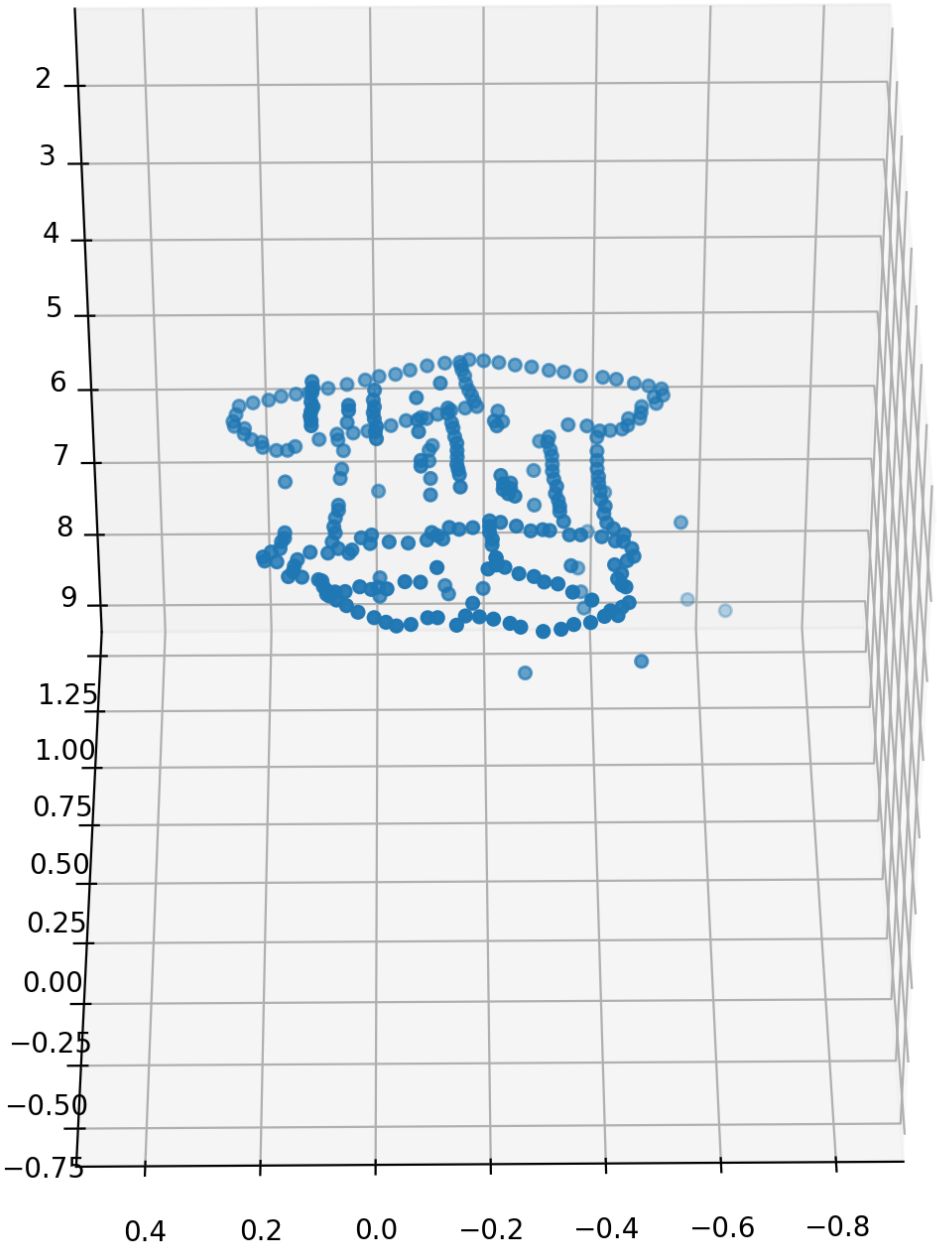
Top view:



Down view:

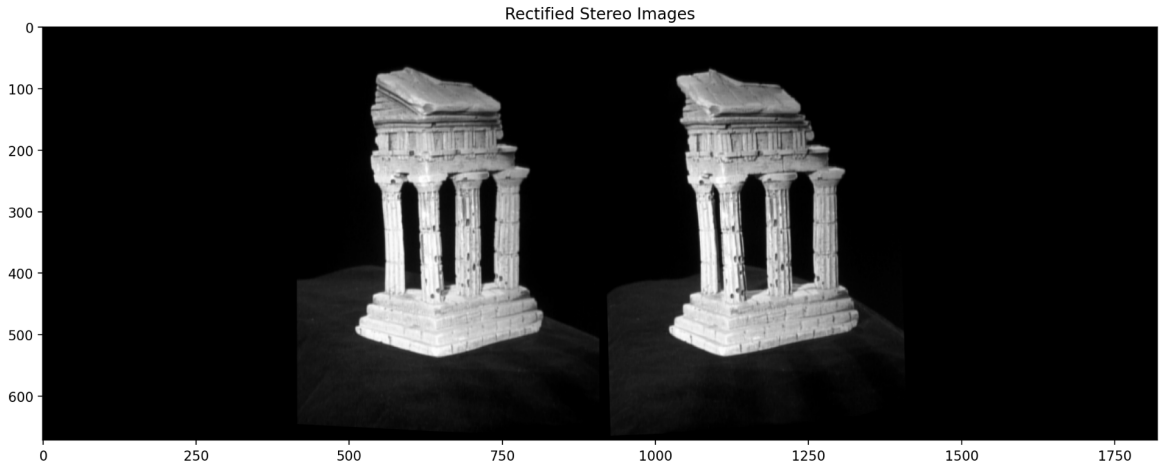


Side view:

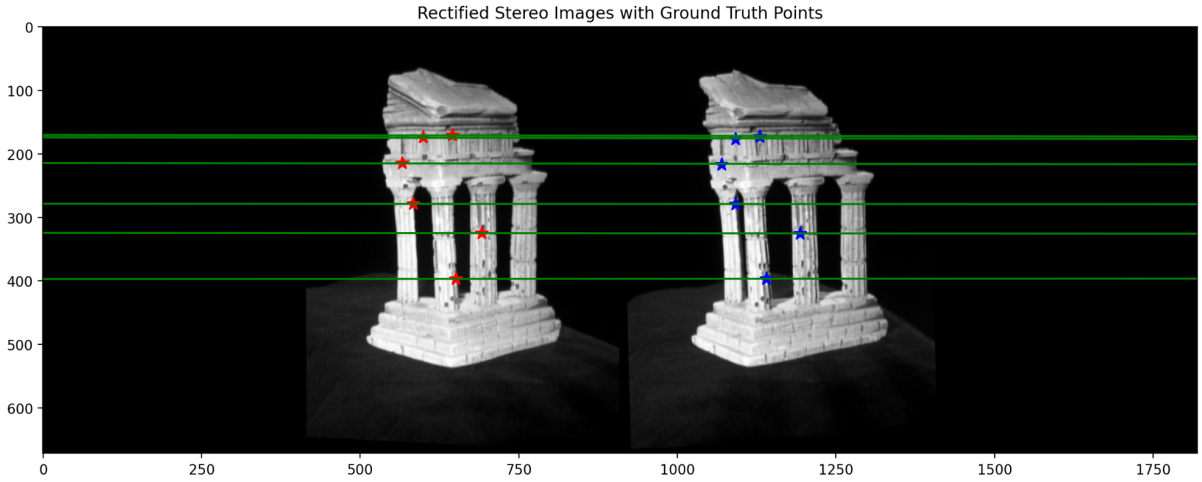


3.2 Dense reconstruction

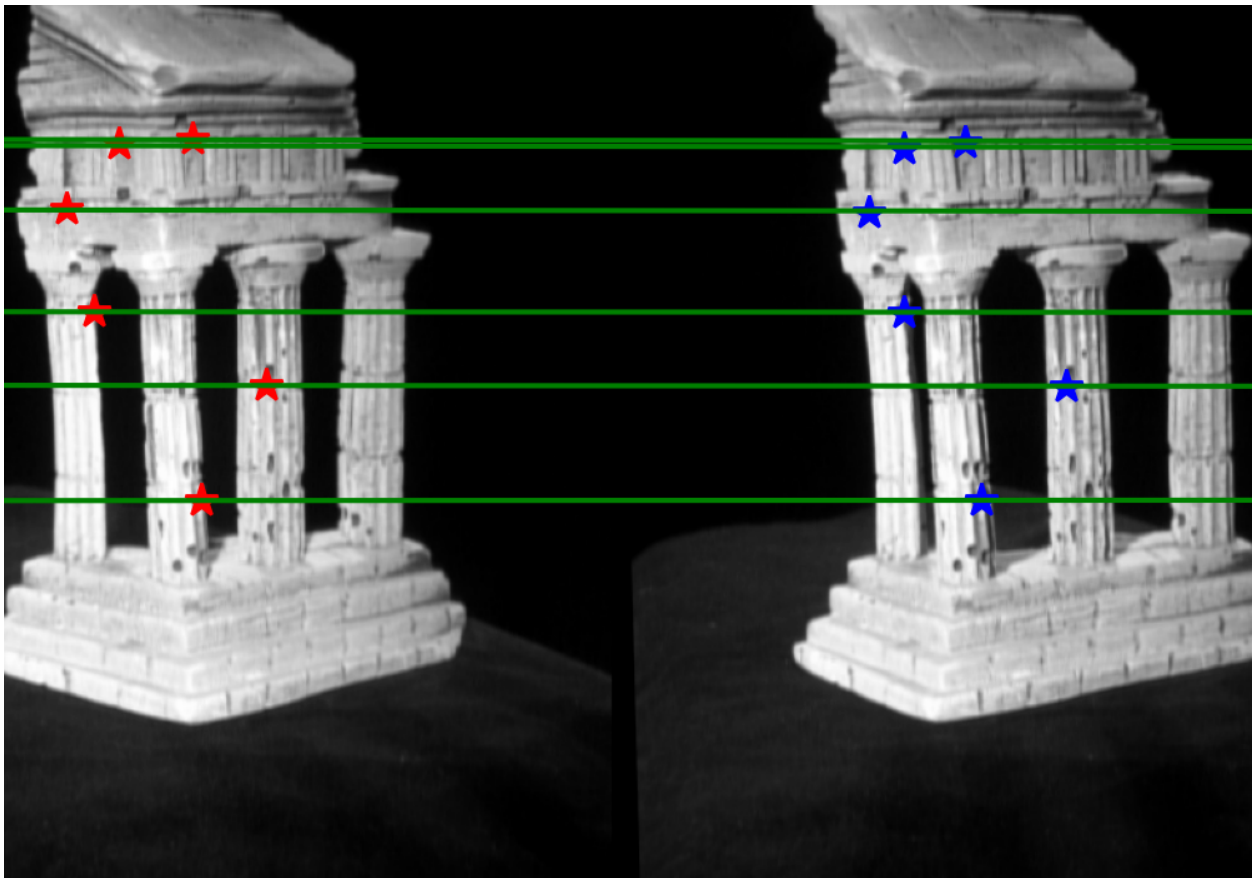
3.2.1 Image rectification (2 pts)



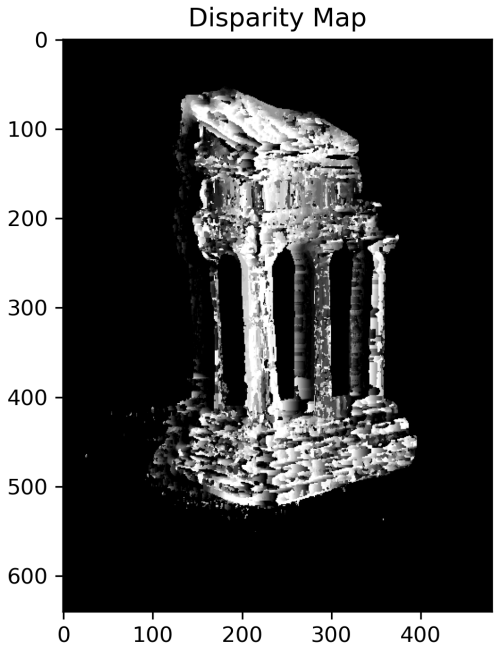
As shown below, the epipolar lines are perfectly horizontal, with corresponding points in both images lying on the same line (also line perfectly across the middle of points).



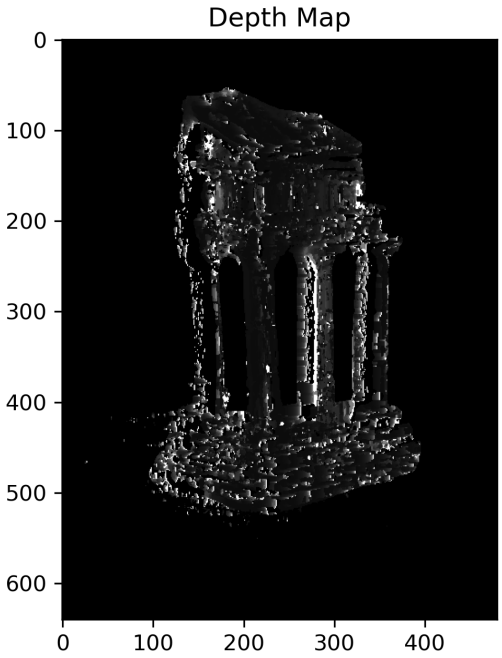
Zoom in for checking:



3.2.2 Dense window matching to find per pixel density (2 pts)



3.2.3 Depth map (2 pts)



3.3 Pose estimation

Output of the script testPose.py:

Reprojected Error with clean 2D points is 0.0000
Pose Error with clean 2D points is 0.0000

Reprojected Error with noisy 2D points is 3.0356
Pose Error with noisy 2D points is 0.1008

3.3.2 Estimate intrinsic/extrinsic parameters (1 pts)

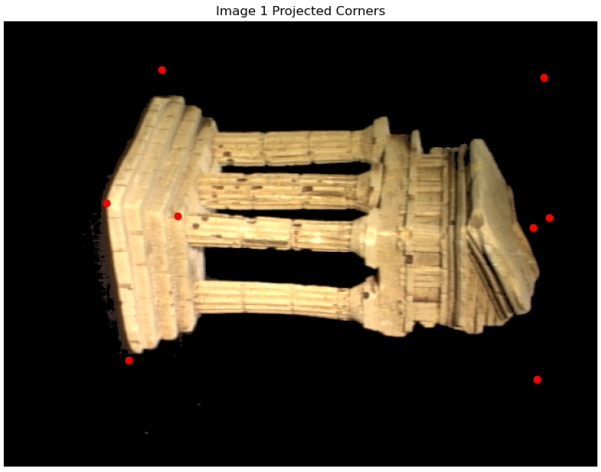
Output of the script testKRt.py

Intrinsic Error with clean 2D points is 0.0000
Rotation Error with clean 2D points is 0.0000
Translation Error with clean 2D points is 1.1164

Intrinsic Error with noisy 2D points is 0.8090
Rotation Error with noisy 2D points is 0.0501
Translation Error with noisy 2D points is 1.6747

3.4 Multi-view stereo

3.4.1 (1 pts)



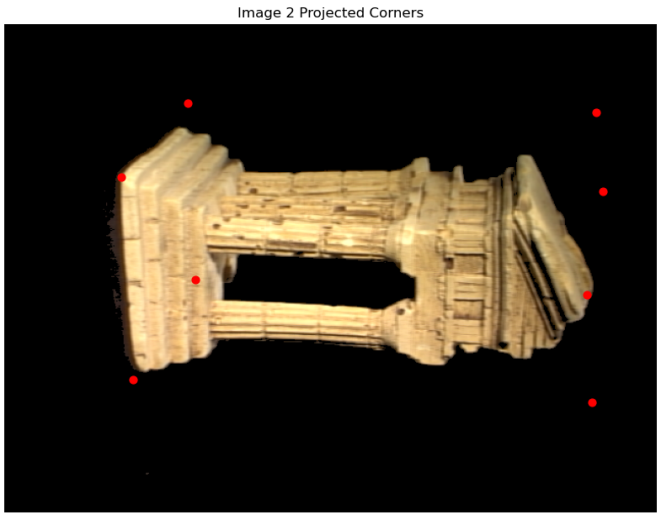


Image 3 Projected Corners

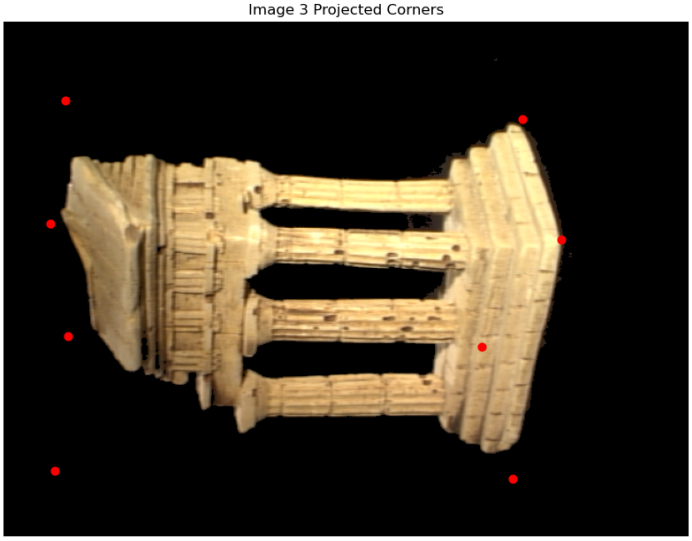
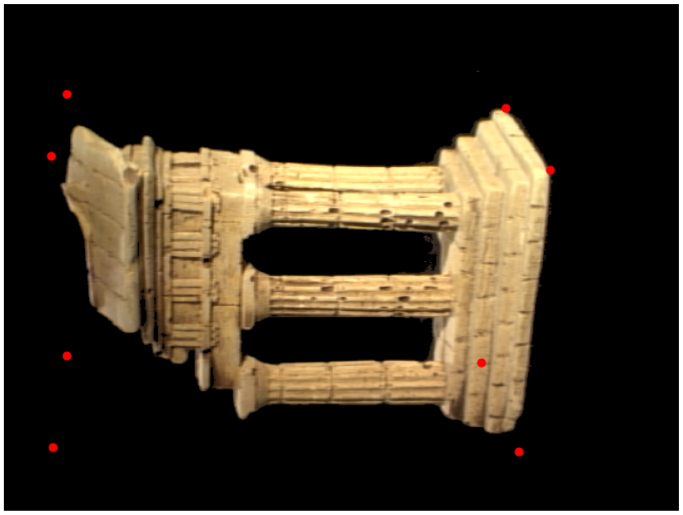


Image 4 Projected Corners



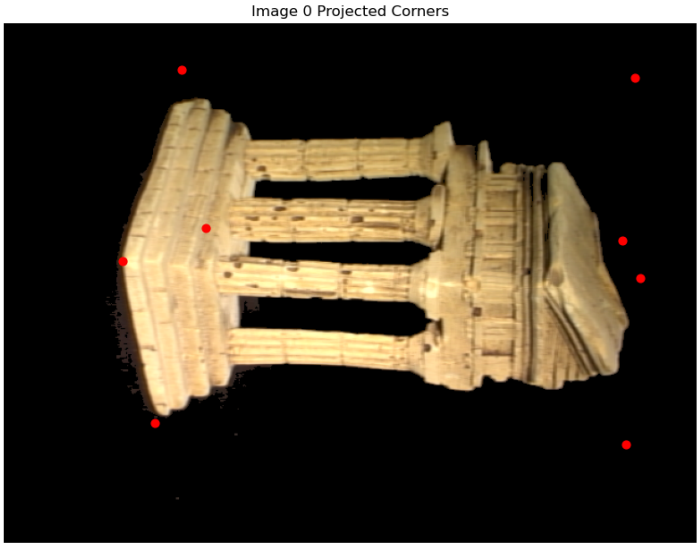
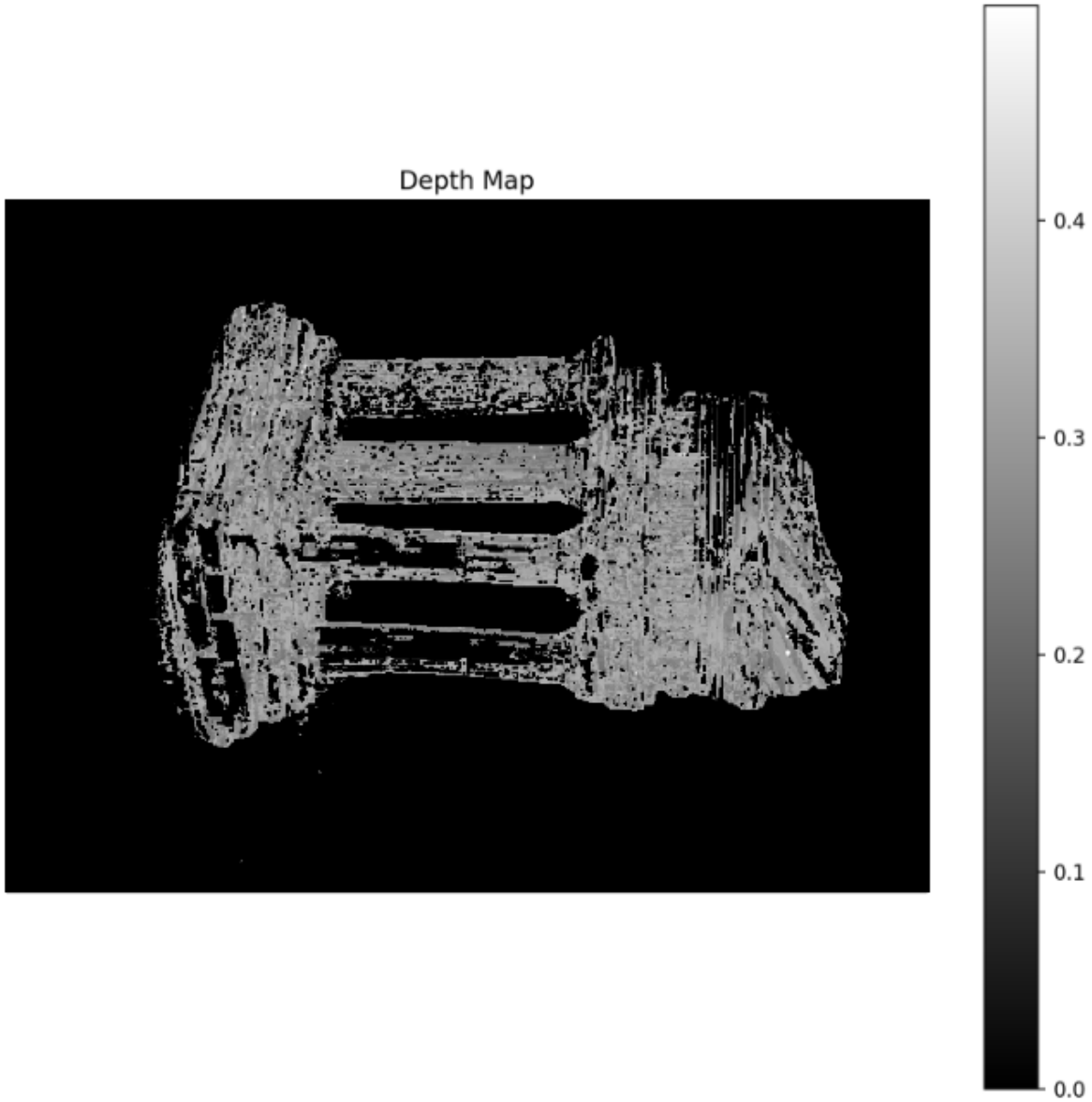


Image 0 Projected Corners

3.4.2 (1 pts)



3.4.3 (1 pts) screenshots of the point cloud with the visualization software