# CS 83: Computer Vision

# 3D Reconstruction

Amittai Siavava

02/18/2024

**Abstract**

This project implements different algorithms for 3D reconstruction of a scene given two camera perspectives (images) of that scene by finding correspondences between the two images and using triangulation to find the 3D coordinates of the points in the scene.

**Credit Statement**

I discussed ideas with **Ivy (Aiwei) Zhang** and **Angelic McPherson**. However, the code and writeup are entirely my own, with reference to class notes.

Contents

## 1. Sparse Reconstruction

In this section, we implement various algorithms to generate a sparse reconstruction of a scene from two sample images, using two images of a temple as a reference.

### 1.1. **Eight Point Algorithm.**

**1.** Recovered Fundamental Matrix:

$$F = \begin{bmatrix} -1.12822743 \times 10^{-9} & 1.23273153 \times 10^{-7} & -6.24786160 \times 10^{-6} \\ 6.41080466 \times 10^{-8} & 1.04807659 \times 10^{-10} & -1.11138892 \times 10^{-3} \\ -1.31615524 \times 10^{-5} & 1.06851400 \times 10^{-3} & 4.47839257 \times 10^{-3} \end{bmatrix}$$

(see Figure 7 for raw matrix)

**2.** Epipolar Line Visualizations:

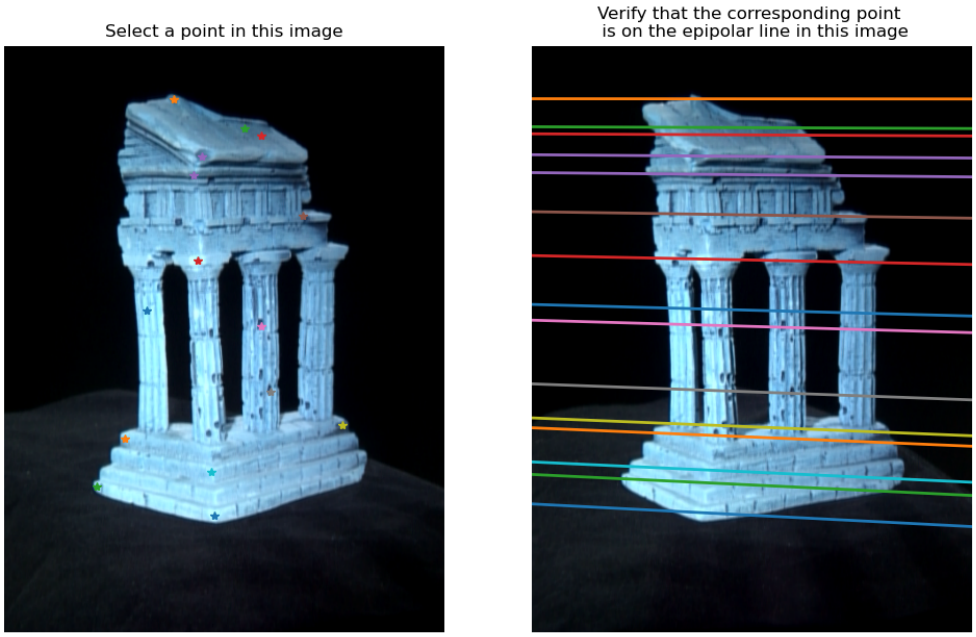



Figure 1. Epipolar lines

1.2. **Epipolar Correspondences.**

**1.** Epipolar Correspondences

To measure similarity, I used a window around each two points being compared, one from the first image and one from the second image. I then computed the sum of squared differences (SSD) between the two windows and used this as a measure of similarity. I experimented with different window sizes; size 5 did not seem to work well especially for non-corner points. Size 20 worked admissibly well although, as demonstrated below, still fails for points that are entirely in the dark background.

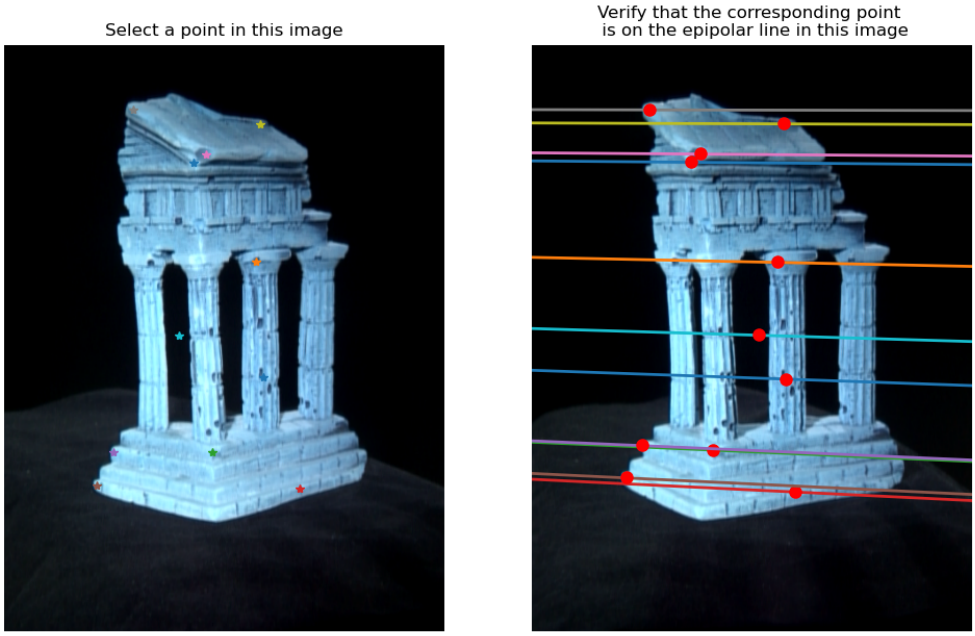



FIGURE 2. Epipolar lines

**2.** Failure Cases

The matching algorithm fails when there aren't enough unique features to distinguish a point from other points on the same epipolar line. A simple example is clicking in the black region in the image. A more interesting example is the point on the red line, which gets mismatched to a wrong point on the epipolar line. One potential solution can be to make the window larger, but that eventually makes the algorithm slower. Maybe combining that with some dynamic programming (instead of recomputing the window every time, maintain the sliding values and only update the edges of the window when moving to the next point on the epipolar line).

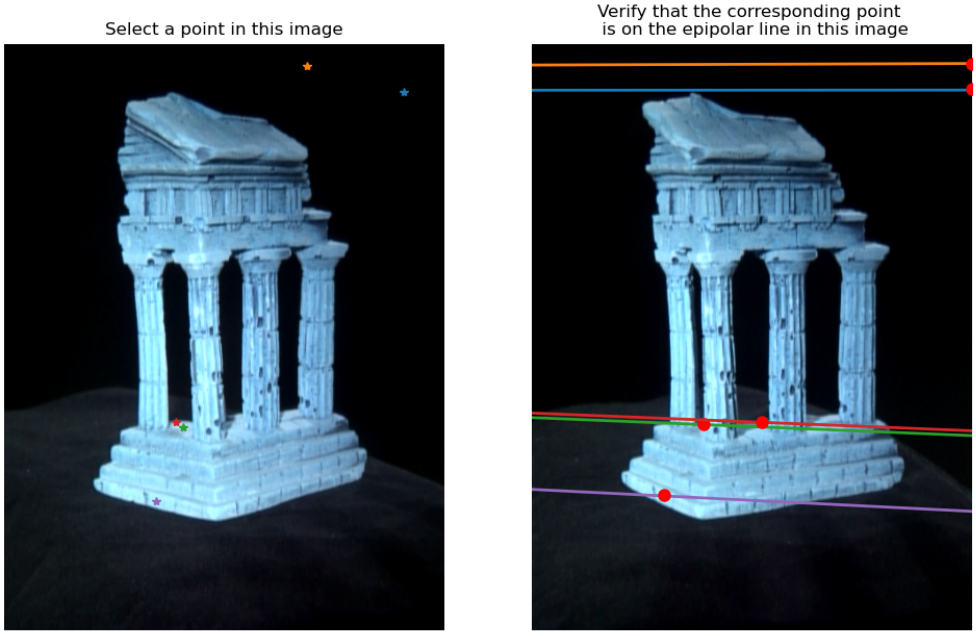



FIGURE 3. Failure case

## 1.3. **Essential Matrix.**

**1.** Recovered Essential Matrix

$$E = \begin{bmatrix} -4.90914507 \times 10^{-1} & 3.96179658 & 2.19734674 \\ 2.12129669 & -1.48056692 \times 10^{-1} & -9.42668155 \\ 6.10057409 \times 10^{-2} & 9.54224292 & 1.09934587 \times 10^{-1} \end{bmatrix}$$

(see Figure 8 for raw matrix)

1.4. **Triangulation.**

**1.** Determining The Correct P-Matrix

As mentioned in the hint, the correct configuration of the P-matrix should have the most points in front of the camera.

**Algorithm 1** Counting Invalid Points

**function** COUNT_INVALID($X[1..n]$)
    $c \leftarrow 0$
    **for** $i \leftarrow 1$ to $n$ **do**
        **if** $X[i][3] < 0$ **then**
            $c \leftarrow c + 1$
        **end if**
    **end for**
    **return** $c$
**end function**

**Algorithm 2** Determining the Correct P-Matrix

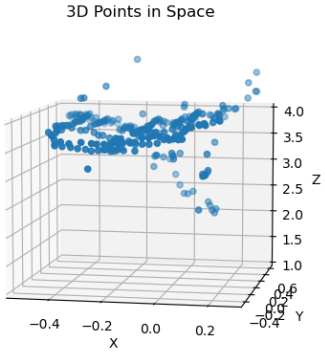
$P_2[1 \,..\, 4] \leftarrow \mathbf{camera2}(E)$
**for** $i \leftarrow 1$ to $4$ **do**
    $X_i \leftarrow \text{TRIANGULATE}(P_1, \text{points}_1, P_2[i], \text{points}_2)$
    $A[i] \leftarrow \text{COUNT\_INVALID}(X_i)$
**end for**
**return** $P_2[\mathbf{argmin}(A)]$

**2.** Reprojection Error

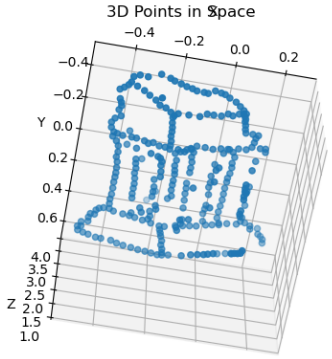
$\text{Points}_1$ Reprojection Error = 0.8521671561856665

$\text{Points}_2$ Reprojection Error = 0.8630693907867739

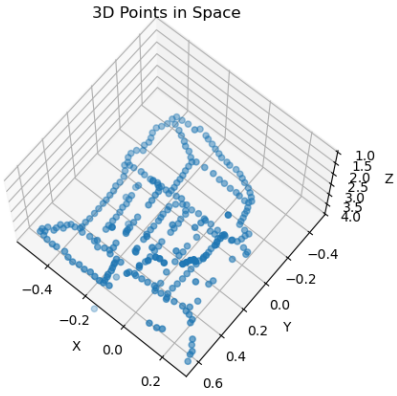
1.5. **Test Script.** Examples of generated sparse reconstructions are shown below.




Perspective 1




Perspective 2




Perspective 3

## 2. Dense Reconstruction

*In applications such as 3D modelling, 3D printing, and AR/VR, a sparse model is not enough. When users are viewing the reconstruction, it is much more pleasing to deal with a dense reconstruction. To do this, it is helpful to rectify the images to make matching easier.*

2.1. **Image Rectification.** Initially, I would get an awkwardly-cropped image when I tried to rectify the images. I discovered that setting the $M$ scaling factor passed to `eight_point` to 1 fixes the issue. The sparse projections looks unaffected, but I saw other students mention on Slack that they ran into the same issue.

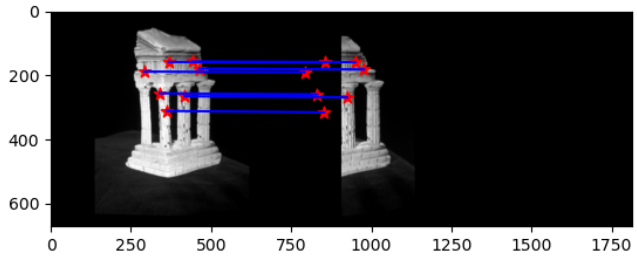



Figure 4. Dense Reconstruction, $M = \mathbf{max}(I_x, I_y)$

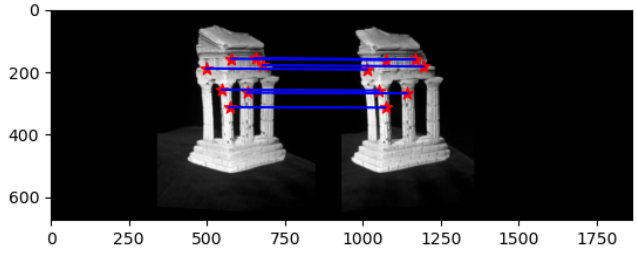



Figure 5. Dense Reconstruction, $M = 1$

2.2. **Disparity Map and Depth Map.** Computed disparity and depth maps

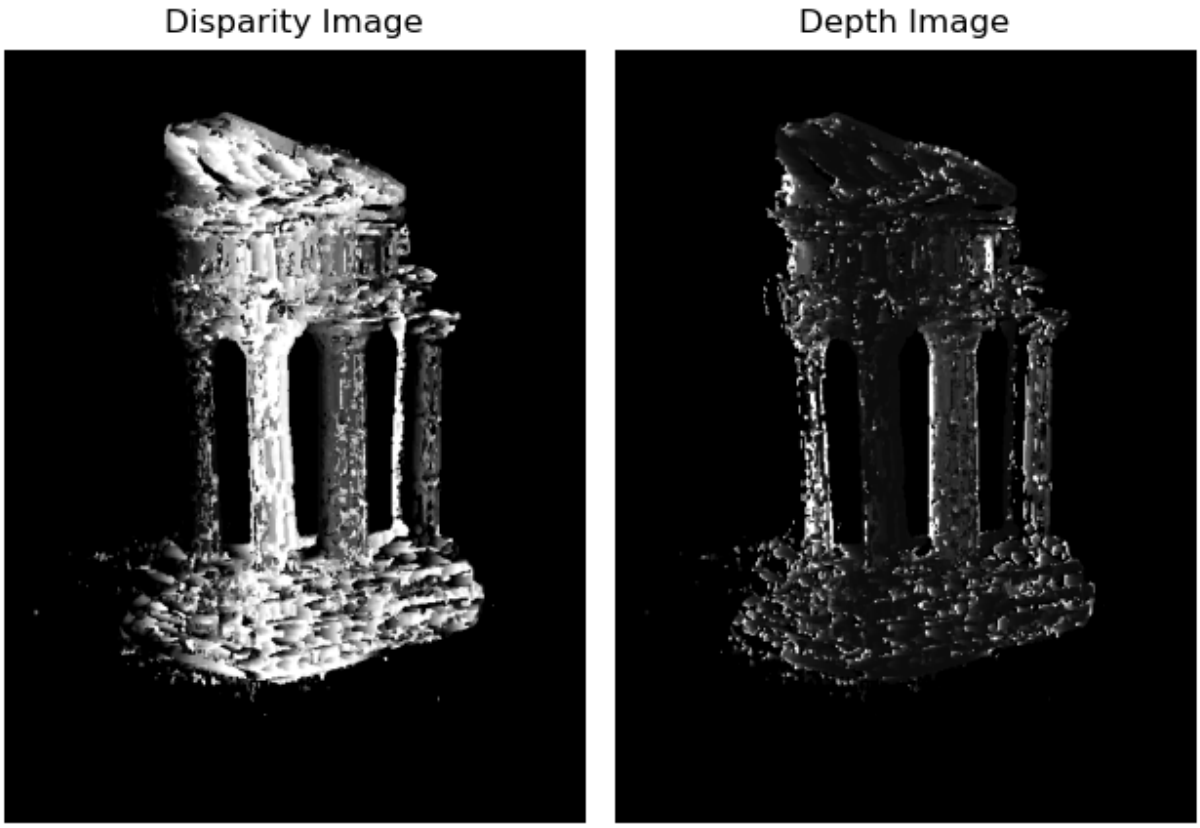



FIGURE 6. Disparity and Depth Maps

## 3. Pose Estimation

3.1. **Camera Matrix Estimation.** Pose estimation errors

Estimated camera matrix

$$P = \begin{bmatrix} 1.19121690 \times 10^{-1} & 5.45061053 \times 10^{-1} & 1.90356266 \times 10^{-1} & -9.32574074 \times 10^{-2} \\ -3.72738881 \times 10^{-1} & 1.96903454 \times 10^{-1} & -4.69365814 \times 10^{-1} & 4.95756458 \times 10^{-1} \\ -2.82243252 \times 10^{-4} & -1.29896978 \times 10^{-4} & 6.42031422 \times 10^{-5} & 1.39340650 \times 10^{-3} \end{bmatrix}$$

| **Metric** | **Value** |
|---|---|
| Reprojection Error with clean 2D points | $1.5101900847116 98 \times 10^{-10}$ |
| Pose Error with clean 2D points | $6.782416819715269 \times 10^{-12}$ |
| Reprojection Error with noisy 2D points | 5.056479446706304 |
| Pose Error with noisy 2D points | 1.1255732262731768 |

Table 1. Pose Estimation Errors (see Figure 9)

3.2. **Intrinsic/Extrinsic Parameters Estimation.**

$$K = \begin{bmatrix} -3.91311040 \times 10^{-1} & 2.16322603 \times 10^{-2} & -5.05036684 \times 10^{-1} \\ 0.00000000 \times 10^{0} & -5.79132614 \times 10^{-1} & -3.84387155 \times 10^{-2} \\ 0.00000000 \times 10^{0} & 0.00000000 \times 10^{0} & 4.38131343 \times 10^{-4} \end{bmatrix}$$

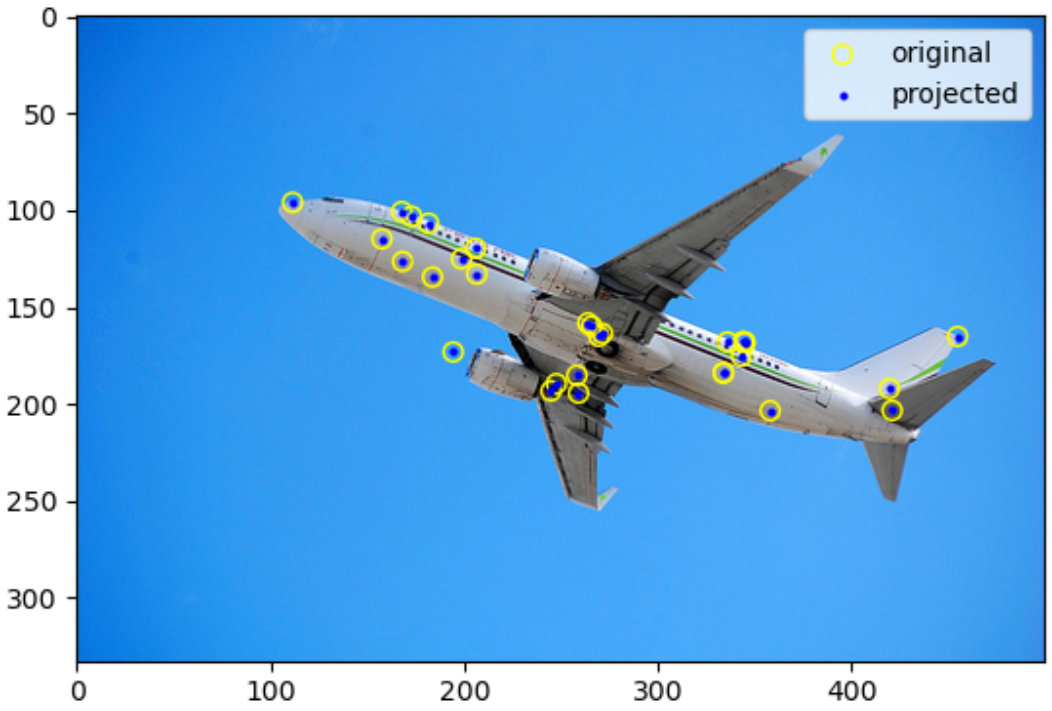
$$R = \begin{bmatrix} -3.04416892 \times 10^{-1} & -9.52538789 \times 10^{-1} & 4.58882152 \times 10^{-4} \\ 9.52538627 \times 10^{-1} & -3.04417134 \times 10^{-1} & -6.10562418 \times 10^{-4} \\ 7.21275976 \times 10^{-4} & 2.51237461 \times 10^{-4} & 9.99999708 \times 10^{-1} \end{bmatrix}$$

$$t = \begin{bmatrix} 1.38022671 // -4.14750766 // 2.38863304 \end{bmatrix}$$

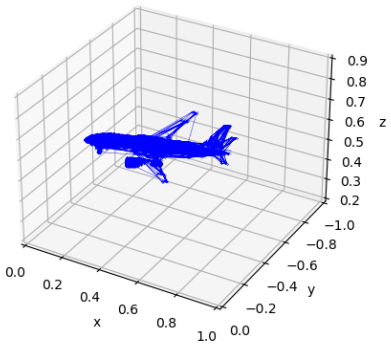
| **Metric** | **Value** |
|---|---|
| Intrinsic Error with clean 2D points | 141.45256375245376 |
| Rotation Error with clean 2D points | 1.8653823929245028 |
| Translation Error with clean 2D points | 3.0626221861668257 |
| Intrinsic Error with noisy 2D points | 141.45251343470733 |
| Rotation Error with noisy 2D points | 1.8735506641560857 |
| Translation Error with noisy 2D points | 4.420673631875621 |

Table 2. Intrinsics/Extrinsics Estimation Errors (see Figure 10)

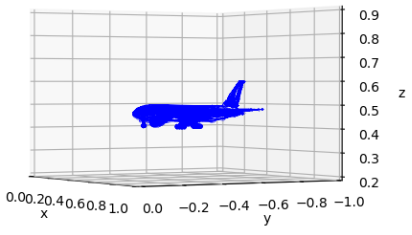
3.3. **CAD Alignment and Projection.** An aeroplane CAD model aligned onto an image using calculated depth information.



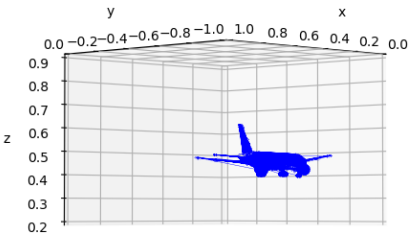

Points Projected onto Image



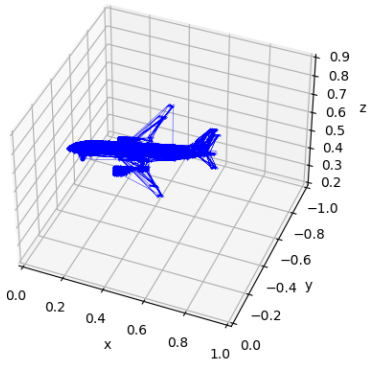

Warped CAD (no rotation)



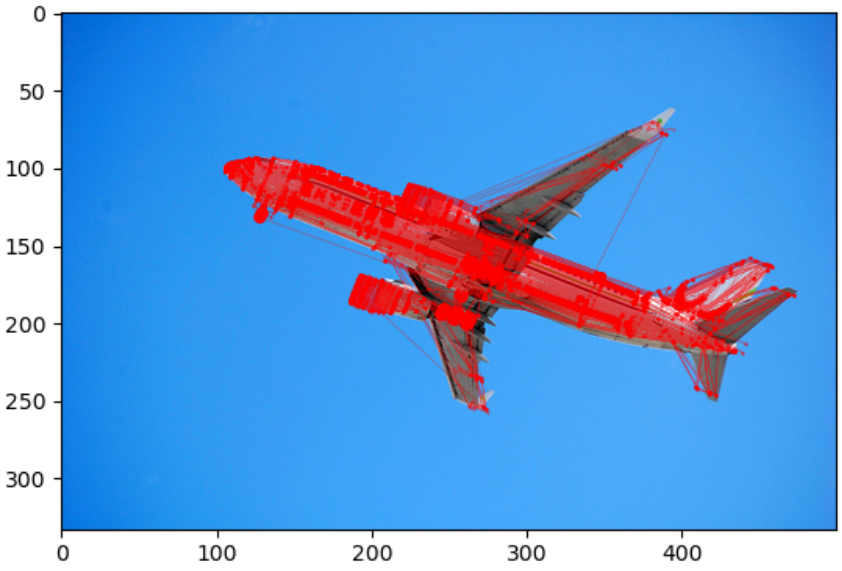

Perspective 2




Perspective 3




Perspective 4




Warped CAD

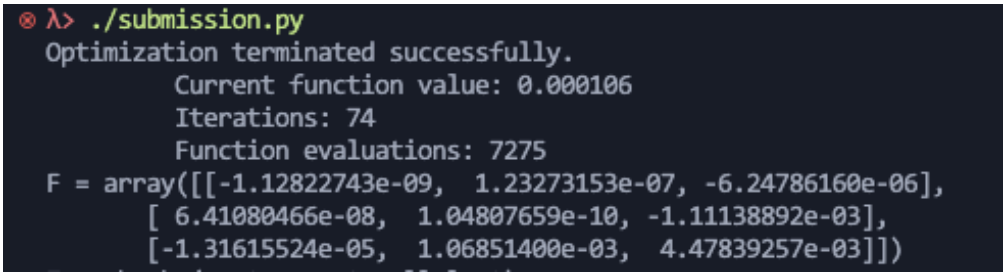



Figure 7. Raw Fundamental Matrix

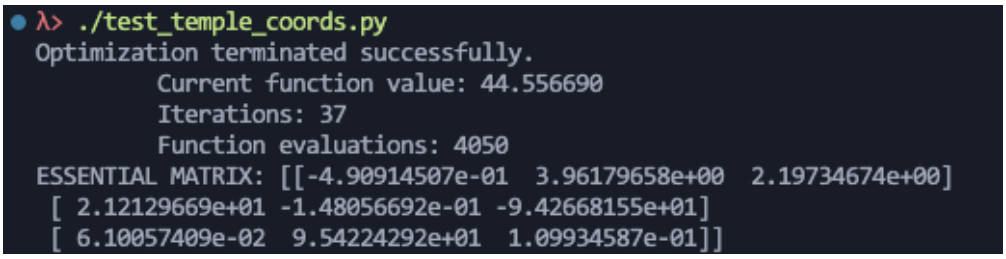



Figure 8. Raw Essential Matrix

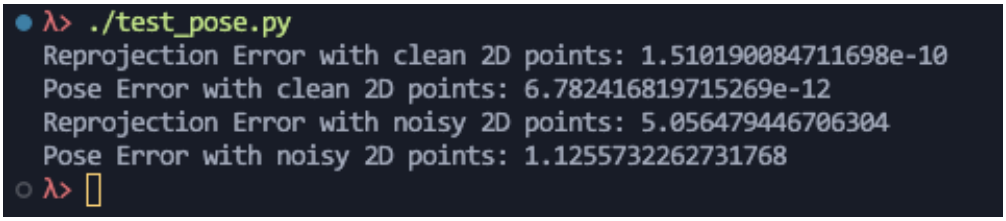



Figure 9. Pose Estimation Errors

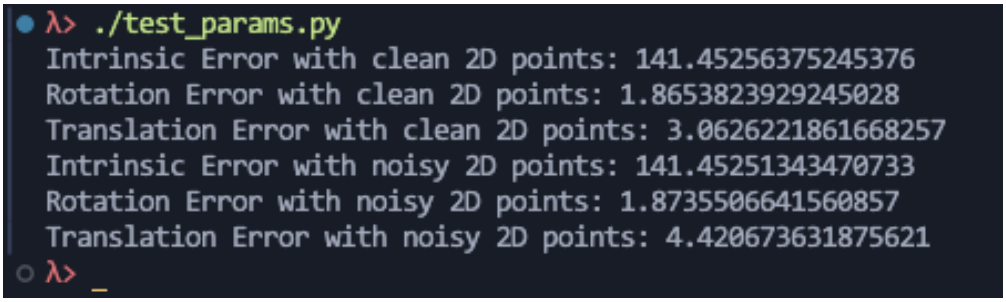



Figure 10. Intrinsics/Extrinsics Estimation Errors