# CSE803 HW6

Javen W. Zamojcin
zamojci1@msu.edu

2024, December 10

## 1 Question 1: Camera Calibration

```
P = [[-2.16139798e-02 -1.99419043e-01 7.24945907e-01 7.02678089e-02]
    [-1.47733446e-02 6.40074852e-02 -2.53073904e-01 -7.28805413e-04]
    [ 2.58411448e-03 -2.38552804e-01 5.51366021e-01 1.00000000e+00]]
```

I solved the projection matrix by first building the following matrix of equations from every pair of correspondence points, and then solving using SVD. This projection matrix was not normalized.

```
A = [[x, y, z, 1, 0, 0, 0, 0, -u * x, -u * y, -u * z]
    [0, 0, 0, 0, x, y, z, 1, -v * x, -v * y, -v * z]]
```

## 2 Question 2: Estimation of the Fundamental Matrix

```
F = [[-1.90213021e-07 6.43692991e-06 -1.74540062e-03]
    [ 1.01351546e-05 2.35169599e-07 9.27190726e-02]
    [ 1.00421527e-04 -9.91702304e-02 1.00000000e+00]]
```

To solve the fundamental matrix, I first normalized all the points before building the correspondence matrix. The points were normalized such that the mean is 0 and the average distance to the origin is 1, through the use of a normalization matrix. After normalizing, the correspondence matrix A was built with the following equations for each normalized point pair, then solved through SVD. But to further improve accuracy, I enforced the rank-2 constraint by setting the smallest singular value of the solved normalized matrix to 0, then de-normalizing by applying the inverse of the normalization matrix. Additionally, the de-normalized fundamental matrix was scaled by a factor of 2 for further normalization. See Figure 1.

```
A = [[x * u, x * v, x, y * u, y * v, y, u, v, 1]]
```

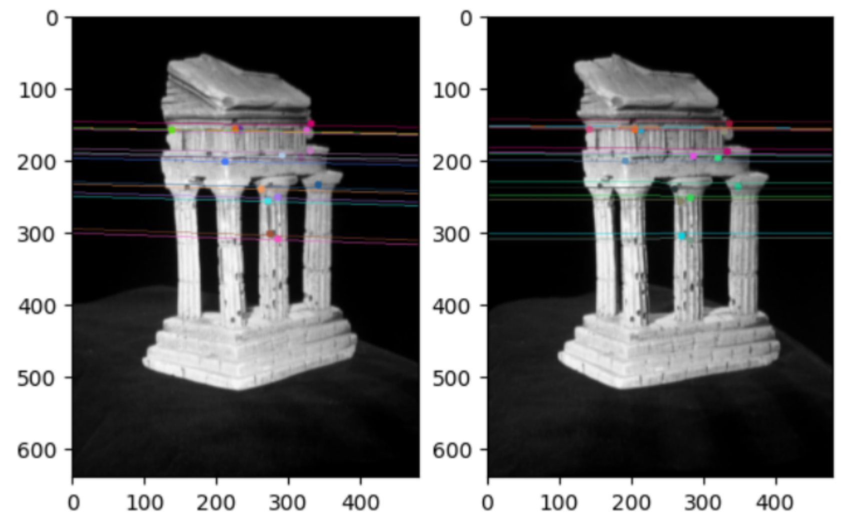

Figure 1: Q2 Epipolar Lines

## 3 Question 3: Triangulation

```
F = [[ 1.03799909 -22.70011437 3.49454518]
     [ 27.768498 -1.17112061 -3.96320412]
     [ -6.63163724 -0.05314547 1.]]

E = [[ 1.03799909e+00 -2.26182934e+01 1.05837760e+04]
     [ 2.78689497e+01 -1.17112061e+00 -1.41830870e+04]
     [ 4.71139019e+00 -4.68670778e+00 -1.88219195e+02]]

P1 = [[1.5204e+03 0.0000e+00 3.0230e+02 0.0000e+00]
      [0.0000e+00 1.5259e+03 2.4690e+02 0.0000e+00]
      [0.0000e+00 0.0000e+00 1.0000e+00 0.0000e+00]]


P2 = [[ 6.18350790e+02 -6.62762912e+02 -1.25753293e+03 -1.42306004e-02]
      [-8.32973510e+02 8.98469160e+02 -9.42464174e+02 2.31875569e-01]
      [ 7.02614732e-01 6.67447327e-01 -2.46671043e-01 9.68699686e-01]]
```

I first loaded the temple points and applied the same normalization methods as described in the previous problem. To extract the essential matrix, I computed the following equation:

$$E = K_2^T * F * K_1^{-1} \tag{1}$$

After decomposing the essential matrix, I calculated the count of positive depth points for each of the four poses and found the pose `(R1, -t)` to produce the

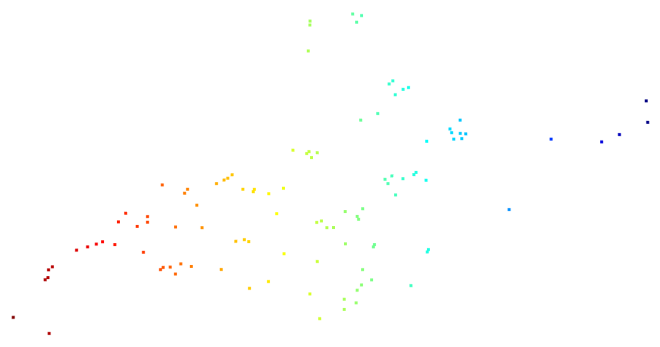

Figure 2: Q3 Point Visualization Angle-1

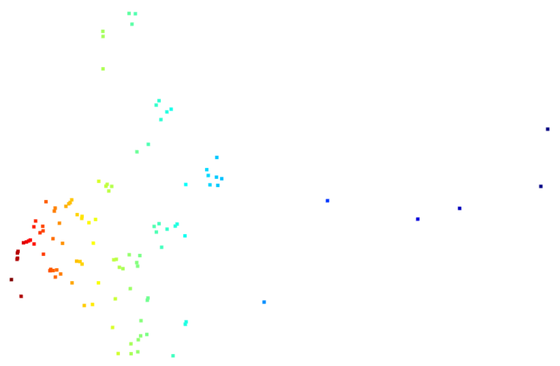

Figure 3: Q3 Point Visualization Angle-2

most positive points of 57. The projection matrices were then calculated by concatenating the rotation and translation matrices appropriately.

To Triangulate the points, I used the `cv2.triangulatePoints(P1, P2, pts1.T, pts2.T)` function call which produced the homogeneous points, which I converted to the 3D points by dividing the first three column values by the 4th column value with an additional small epsilon value added to avoid dividing by zero. After triangulating, the 3D points were then visualized using a custom `open3d` point cloud function and records from three different viewing angles. See figures 2 through 4.

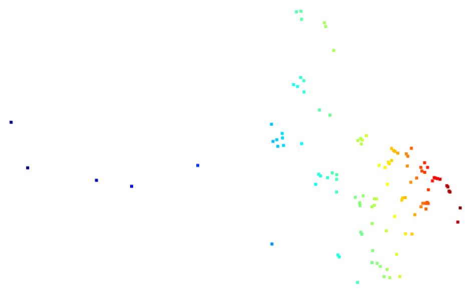

Figure 4: Q3 Point Visualization Angle-3