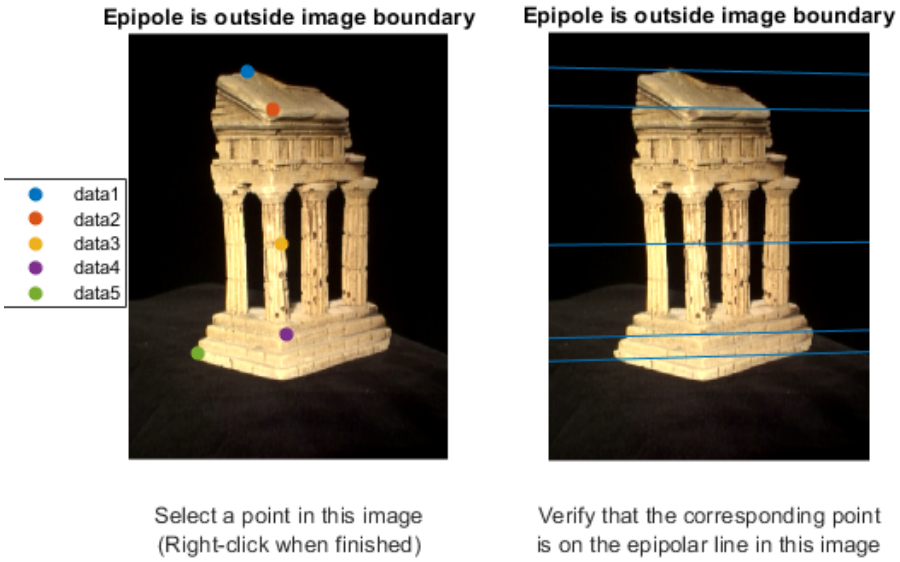
CMPT412 Project 5
Marco Hiu Yeung Lai
301356237

3.1.1 Implement the eight point algorithm

The function inputs two sets of corresponding points, a scale parameter, and outputs the estimated fundamental matrix. Then it normalizes the points, enforces the rank 2 constraints, and refines the solution using local minimization. Below are the epipolar lines:



Corresponding recovered matrix F:

F

3x3 double

	1	2	3
1	3.5644e-09	-1.3083e-07	3.0478e-05
2	-5.9213e-08	-1.3110e-09	-0.0011
3	-1.6503e-05	0.0011	-0.0042

```
>> test
    0.0000   -0.0000    0.0000
   -0.0000   -0.0000   -0.0011
   -0.0000    0.0011   -0.0042
```

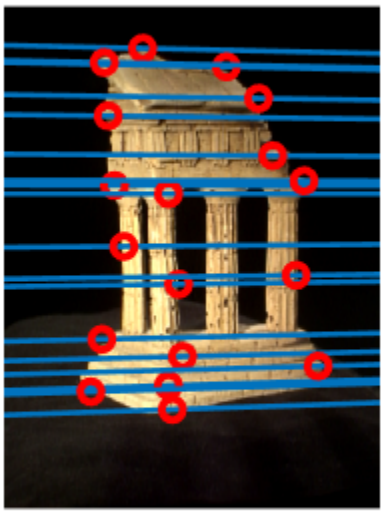
3.1.2 Find epipolar correspondences

For each point in pts1, the algorithm operates on a sliding window of size windowSize. Using the epipolarLine function, it first computes the epipolar line associated with each point. The intersection of this line with the image plane of image 2 is then computed to obtain candidate points.

The algorithm computes a similarity score for each candidate point by comparing the window around the corresponding point in image 1 with the window around the candidate point in image 2. The similarity score is defined as the sum of the absolute differences between the two windows' corresponding pixel values.

The algorithm chooses the candidate point with the highest similarity score to be the corresponding point in image 2 for each point in pts1. pts2 represents the final corresponding points. Below shows the output:

Select a point in this image
(Right-click when finished)



Verify that the corresponding point
is on the epipolar line in this image

3.1.3 Write a function to compute the essential matrix

For this part, the essential matrix is just computed from the fundamental matrix and intrinsic camera matrix formula

```
E = K2' * F * K1;
```

Essential matrix
disp(E)

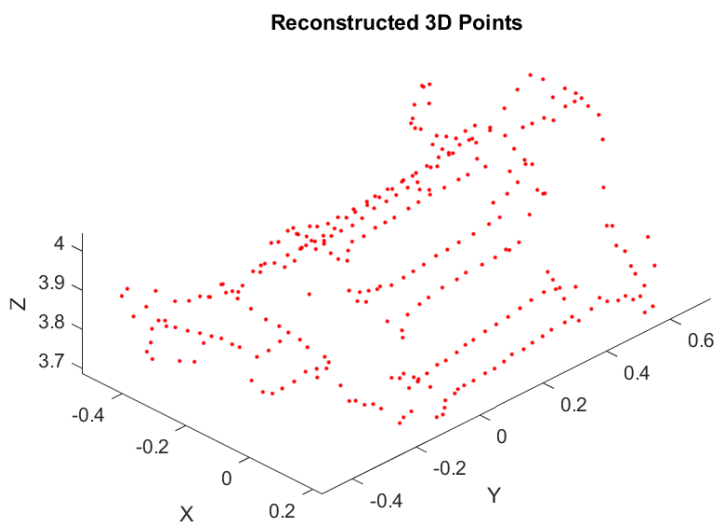
```
    0.0082   -0.3035   -0.0011
   -0.1374   -0.0031   -1.6760
   -0.0457    1.6554   -0.0029
```

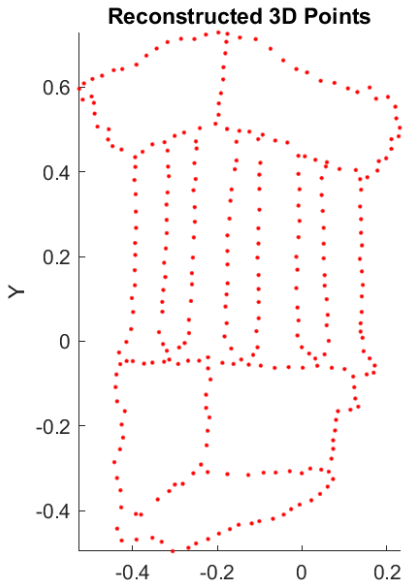
3.1.4 Implement triangulation

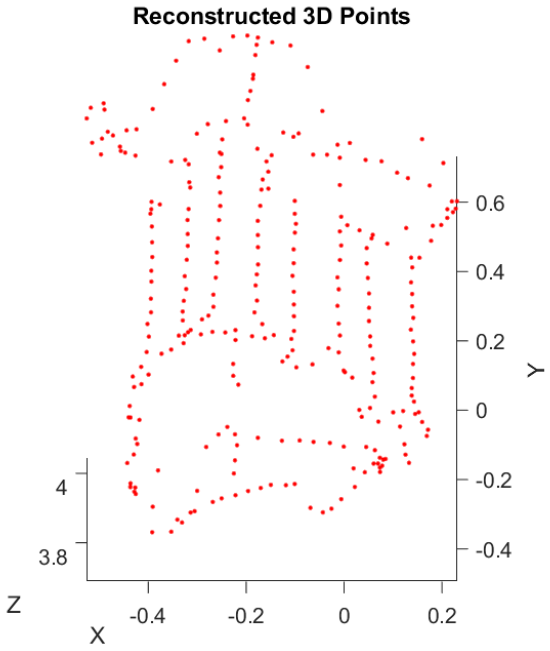
The algorithm extracts the 2D coordinates from each image point (x1, y1, x2, y2) and builds a matrix A from the projection matrices and the 2D points. The last column of its SVD decomposition then gives us the point's 3D position in homogeneous coordinates. The values in the last column are normalized by dividing by the fourth element. The 3D point that results is then added to the pts3d matrix. The resulting matrix pts3d contains the estimated 3D positions of the corresponding points in the two images after iterating through all of the points. The re-projection error is 0.0447 for point 1 and 0.0444 for point 2. For algorithm details please see triangulate.m, re-projection error please refer to testTempleCoords.m.

3.1.5 Write a test script that uses templeCoords

For this part, we first calculate the corresponding point in pts2 using epipolarCorrespondence function. Next, the essential matrix E is computed from the fundamental matrix F. Then the camera matrix cP2 is computed using the essential matrix E and the intrinsic camera matrices K1 and K2. Followed by the triangulate function is used to calculate the 3D positions of the points using the projection matrices P1 and cP2. The algorithm then iterates over the four possible camera matrices in cP2 and selects the one with the smallest cost function, where the cost function is defined as the sum of the Euclidean distances between the original and projected 2D coordinates(re-projection error). The chosen camera matrix is saved as P2, and the 3D coordinates of the points are saved as pts3d. There are 6 images in matlab folder, below are 3 images from different angles:

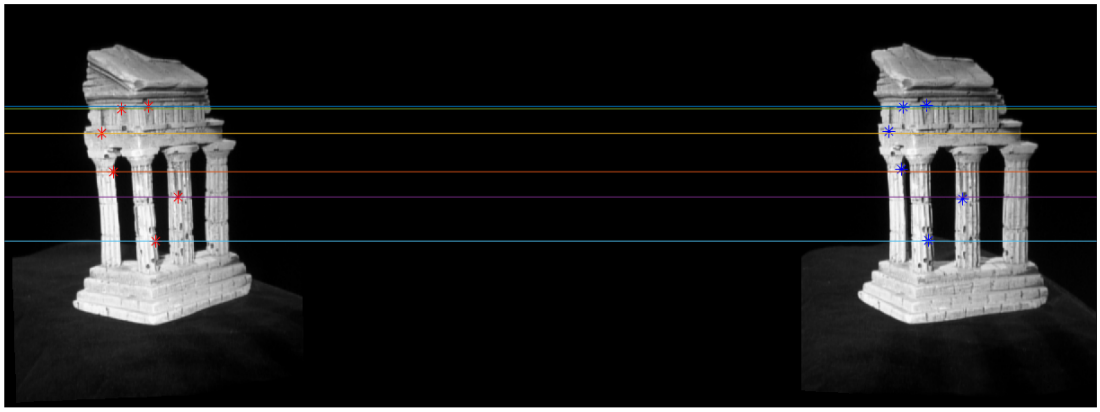

Reconstructed 3D Points
Z
4
3.9
3.8
3.7
X
-0.4
-0.2
0
0.2
Y
-0.4
-0.2
0
0.2
0.4
0.6


Reconstructed 3D Points
Y
0.6
0.4
0.2
0
-0.2
-0.4
-0.4
-0.2
0
0.2


Reconstructed 3D Points
Z
4
3.8
X
-0.4
-0.2
0
0.2
Y
0.6
0.4
0.2
0
-0.2
-0.4

3.2.1 Image rectification

Given left and right camera intrinsic and extrinsic parameters, the rectify pair function computes left and right rectification matrices and updated camera parameters. It goes through a series of steps that include calculating optical centres, new rotation matrices, intrinsic parameters, and translations. The results are perfectly horizontal, with corresponding points in both images lying on the same line.



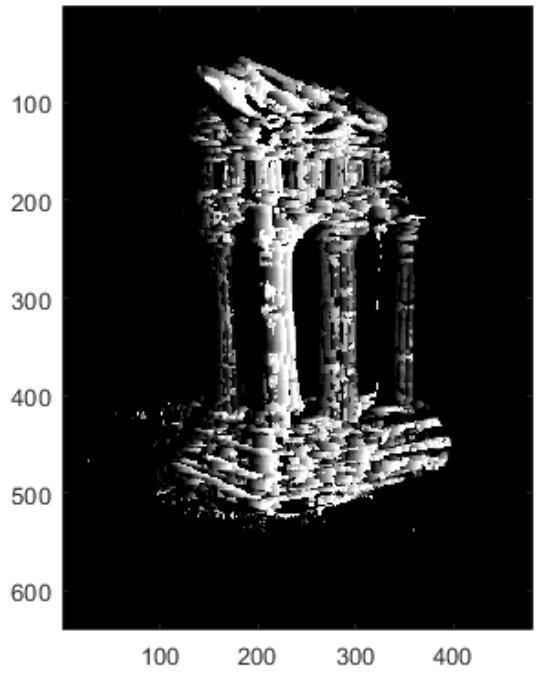
3.2.2 Dense window matching to find per pixel density

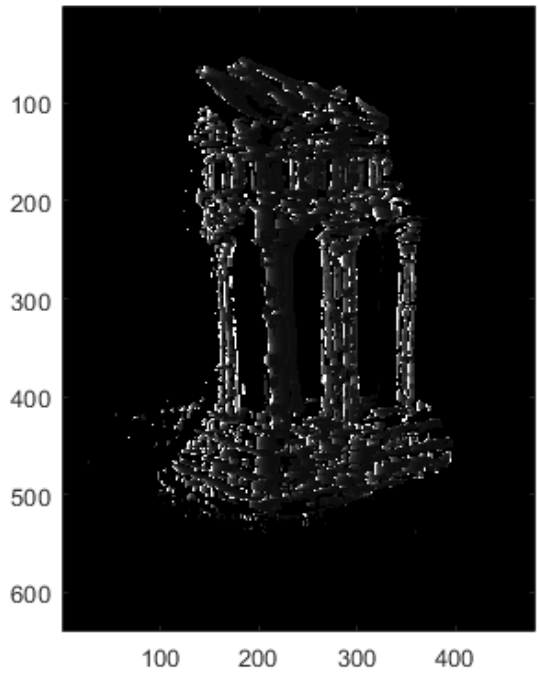
The get_disparity function takes two rectified images and computes a disparity map using dense window matching with a maximum disparity and window size specified. The result is a disparity map with the same dimensions as the input images, calculated with a sum of squares difference distance function.

3.2.3 Depth map

For this part, the get_death function computes the depth map using the formula depthM(y, x) = b × f /dispM(y, x), where b is the baseline and f is the focal length of the camera.

Below are images after rectification. Left: Disparity Right: Depth





3.3.1 Estimate camera matrix P

```
>> testPose
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 1.9430
Pose Error with noisy 2D points is 0.0559
```

3.3.2 Estimate intrinsic/extrinsic parameters

```
>> testKRt
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 0.0000
------------------------------
Intrinsic Error with clean 2D points is 0.5982
Rotation Error with clean 2D points is 0.1173
Translation Error with clean 2D points is 0.2023
```

3.4 Multi-view stereo (BONUS)

SKIPPED