CMPT 412

Project 3

3D Reconstruction

(Using 1 free late day)

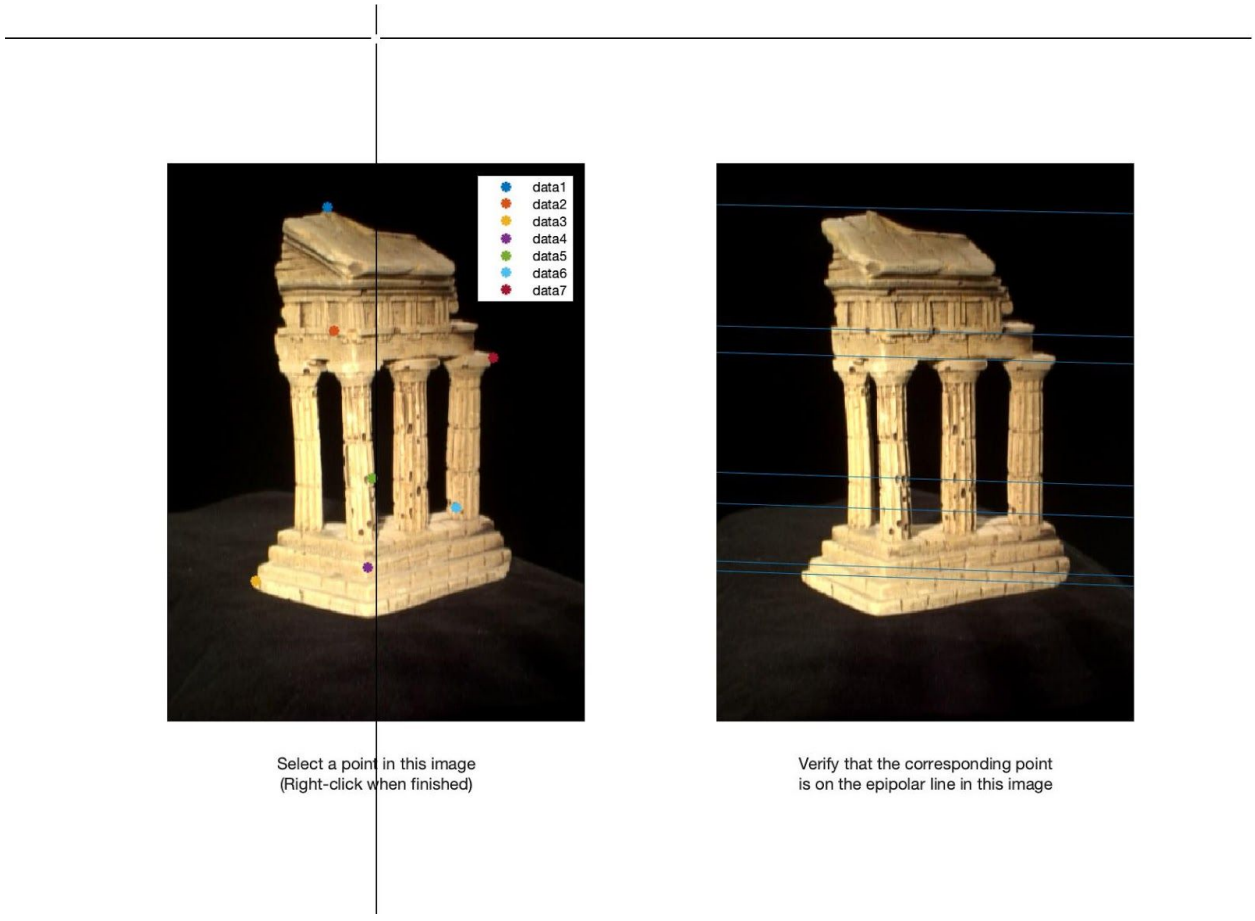
3.1.1 Eight Point Algorithm

In this part, by simply following the steps provided, I was able to implement the eight point algorithm and estimate the fundamental matrix. The fundamental matrix estimated is below (the 0 values are very small and only displayed as 0 but not actually 0):

F =

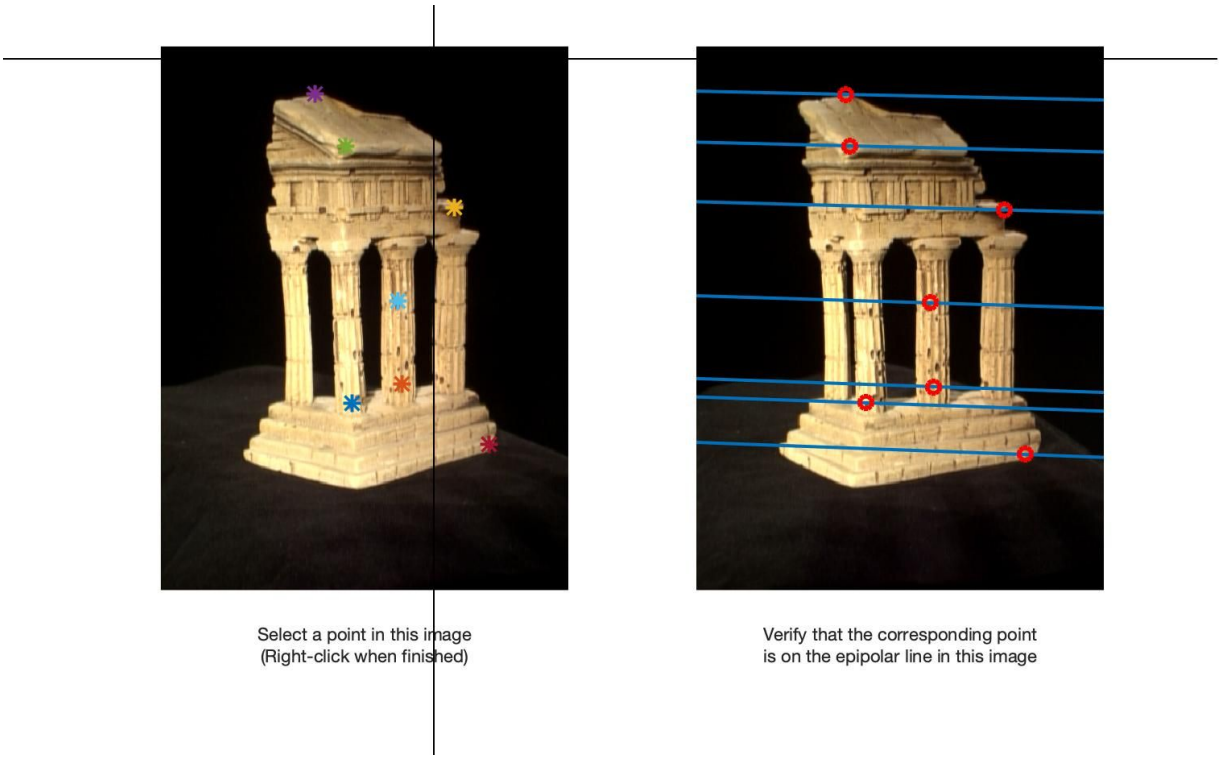
0.0000	-0.0000	-0.0000
-0.0000	-0.0000	0.0005
0.0000	-0.0005	-0.0018

Below is a screenshot of the usage of displayEpipolarF function. A few points are selected in the left image and the corresponding epipolar lines are shown in the right image. (Please ignore the cross pointer which happens to be in the screenshot).



3.1.2 Epipolar Correspondences

In this part, by multiplying the fundamental matrix by the points in left image for which we want to find correspondence in the right image, we retrieve a line equation. This line equation can be used to figure out what pixels in the right image to check. I then create a matrix with all the “interesting points” in the right image that I want to check. Then by shifting a window of size 11 across the epipolar line, I was able to accurately find the correspondence for each desired point in the left image using the Euclidean distance to the template from the left image. Different window sizes were tried, however, 11 delivered the best results in my case. Smaller windows would sometimes fail to select the correct point. A couple of points seemed to always fail no matter the window size. For instance, a point on the stairs that looks very uniform and similar to other points in the neighbourhood. This is likely due to the fact that Euclidean distance is not a perfect metric for such a task. However, obvious points had 100% success rate using the method described above. Below is a screenshot of the epipolarMatchGui running on my implementation. (Again please don’t mind the cross. For some reason it was there again when I took the screenshot)



3.1.3 Essential Matrix

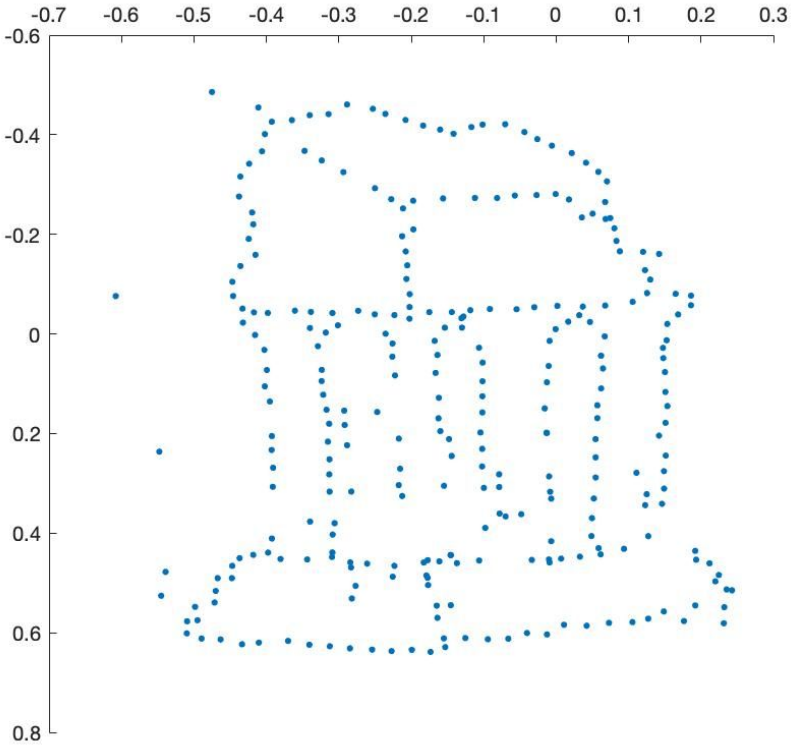
In this part, we simply use the F matrix and the intrinsic parameters to compute the essential matrix using this formula: E = transpose(K2) * F * K1;

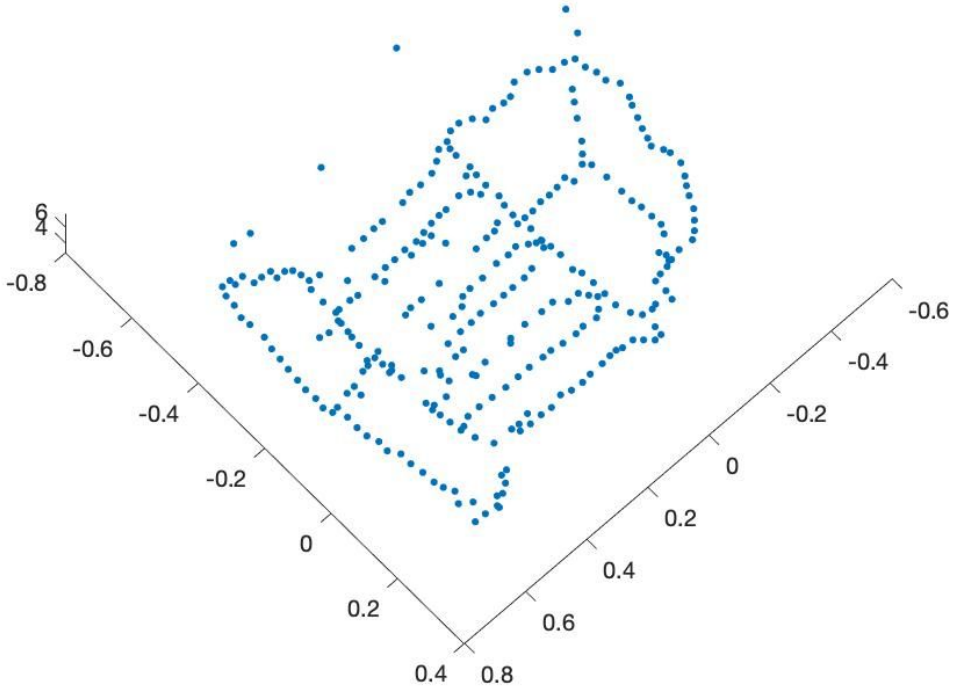
3.1.4 Triangulation

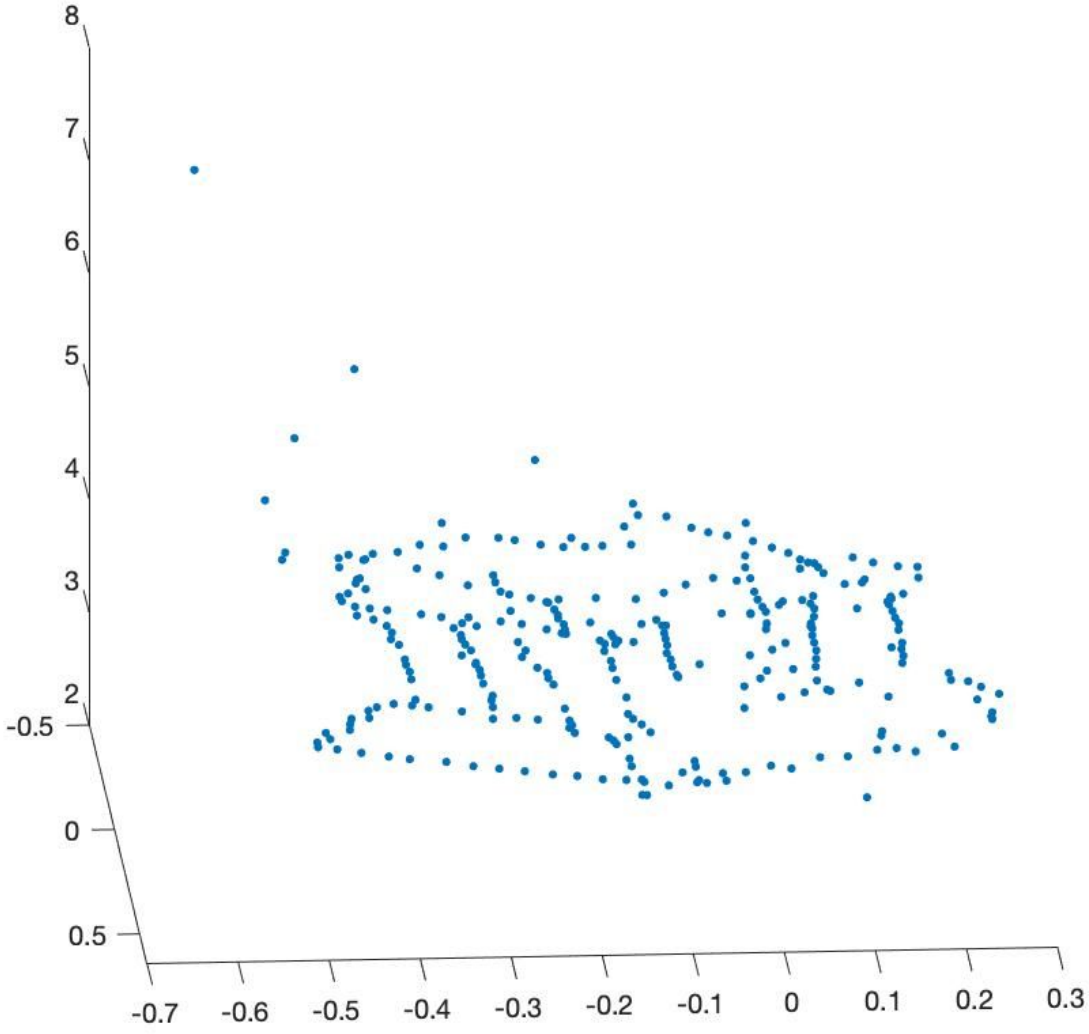
In this part, I was able to triangulate the 2D points to 3D and reproject back to 2D with a reprojection error of 0.62 pixels for left image reprojection and 0.63 for right image reprojection (this was done using Euclidean distance to true points from reprojected points). Moreover, when deciding which extrinsic matrix to choose out of the 4 available ones, positive depth test was performed. By putting an if condition in my test script which will only pass if all the reprojected values of the form [u v w]' have a positive w. If one is negative, it means it is not in front of the camera and must be discarded. My code only passed on the 3rd extrinsic matrix. All the procedures described in this section are done in the testTempleCoords script and not in the triangulation function. The triangulate function simply assumes the right extrinsic is passed to it and also does not check for reprojection errors. All of that is done in the test script testTempleCoords.

3.1.5 testTempleCoords Test Script

In this part, all the sections above come together and we are able to plot the triangulated 3D points. Below are 3 images which show the 3D plot from different angles.

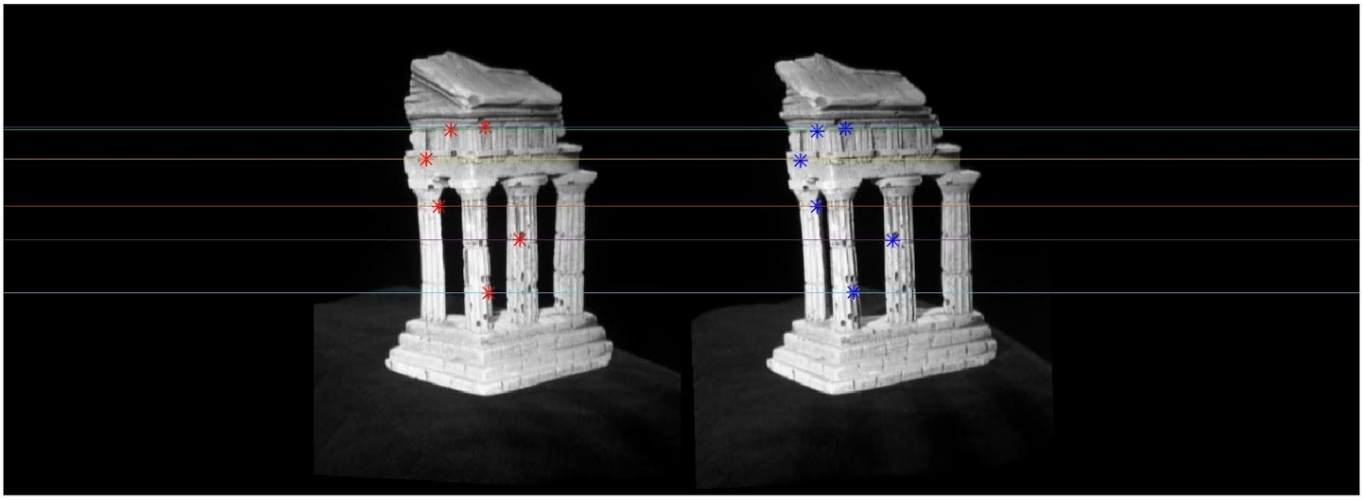

-0.7 -0.6 -0.5 -0.4 -0.3 -0.2 -0.1 0 0.1 0.2 0.3
-0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8


6
4
-0.8 -0.6 -0.4 -0.2 0 0.2 0.4
-0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8


8
7
6
5
4
3
2
-0.5
0
0.5
-0.7
-0.6
-0.5
-0.4
-0.3
-0.2
-0.1
0
0.1
0.2
0.3

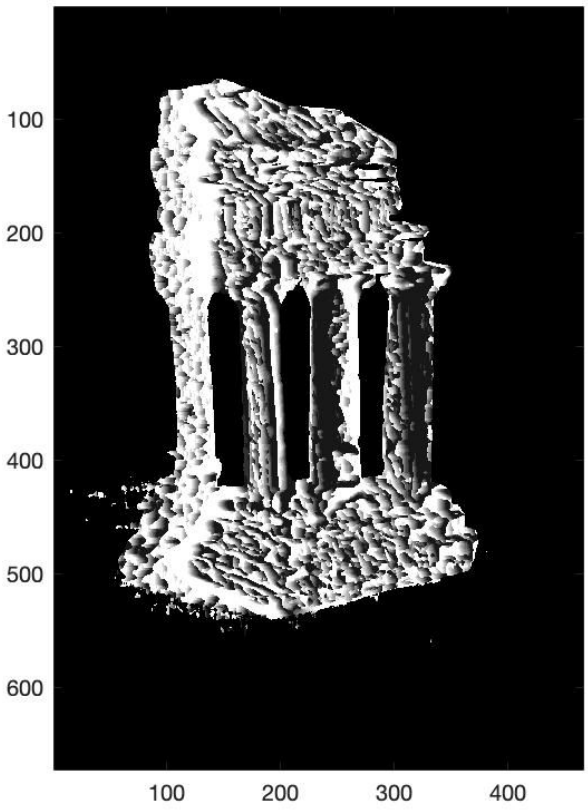
3.2.1 Rectification

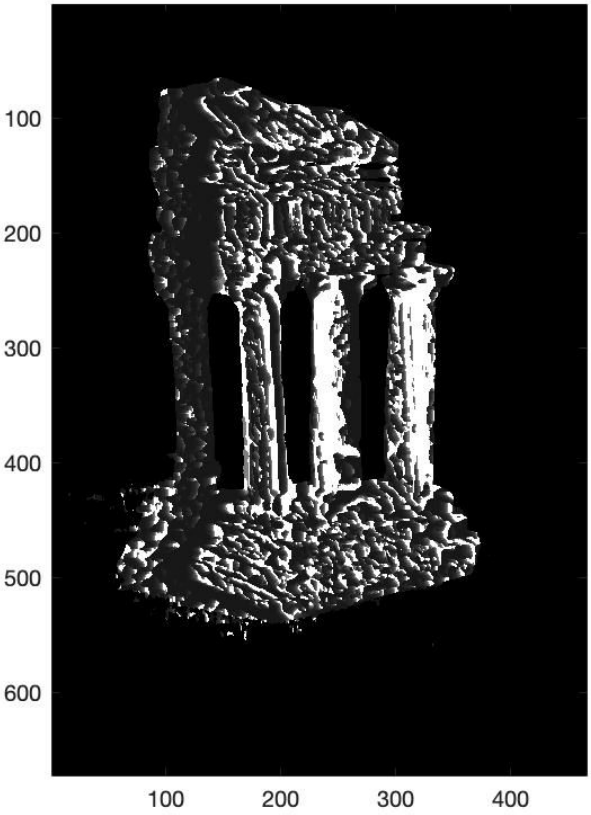
In this part, we rectify the given images to generate perfectly horizontal epipolar lines in order to be able to create a dense depth map through simple 1D search. Below is a screenshot of the results from testRectify using my rectification implementation.



3.2.2 & 3.2.3 Disparity Map & Depth Map

In this part, a disparity map is generated from the rectified temple images. The similarity measure used is Sum of Squared Differences or SSD. By attempting to minimize the SSD across a horizontal line and using a window size of 5 and maximum disparity of 10, the below images are generated. The first one is the disparity map and the second one is the corresponding depth map.


100
200
300
400
500
600
100
200
300
400


100
200
300
400
500
600
100
200
300
400

Below is another example disparity map created using different parameters. Window size of 5 and maximum disparity of 30 was used for disparity map below.