Project 3

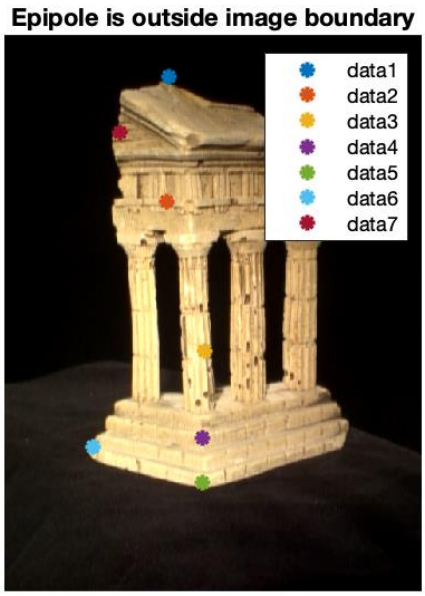
3D Reconstruction

Computer Vision - CMPT 762

3.1 Sparse reconstruction

3.1.1 Implement the eight point algorithm

Here is the result for eightpoint.m by using function displayEpipolarF.m



The recovered F is:

F =

0.0000	-0.0000	-0.0000
-0.0000	-0.0000	0.0005
0.0000	-0.0005	-0.0021

I used scaling to get this result. We can also normalize points by distance. Here you can see F by normalizing:

F =

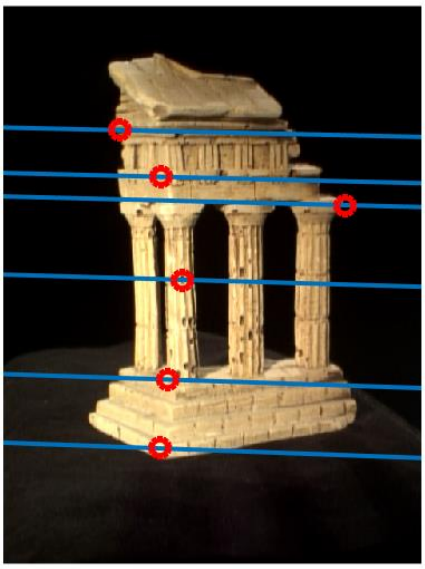
-0.0000	0.0000	-0.0000
0.0000	-0.0000	-0.0007
-0.0000	0.0007	0.0028

As the results are similar to each other, I've used just scaling for the rest of the report.

3.1.2 Find epipolar correspondences

Here is the result for epipolarCorrespondence.m

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



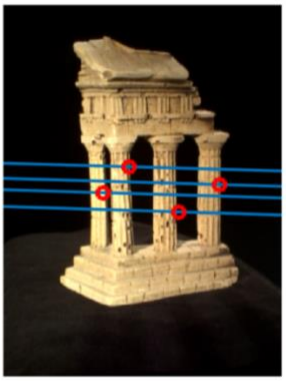
Verify that the corresponding point
is on the epipolar line in this image

We used the Manhattan distance with a kernel of size 35.

```
dist(j) = mean(mean(mean(abs(im1_patch - im2_patch))));
```

Here you can see failure cases if we use kernel size 25. It is hard for the algorithm to distinguish between different pillars. I think the problem is with the similarity that we used here. Because it only checks the vicinity of the pixel. We can alleviate this problem by increasing kernel size when we want to compute distance. Or, we can use other complex similarity measures.

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

Here is the estimated E matrix for the temple image pair by running essentialMatrix.m

E =

0.0040	-0.0433	-0.0192
-0.1498	-0.0009	0.7264
0.0019	-0.7352	-0.0008

3.1.4 Implement triangulation

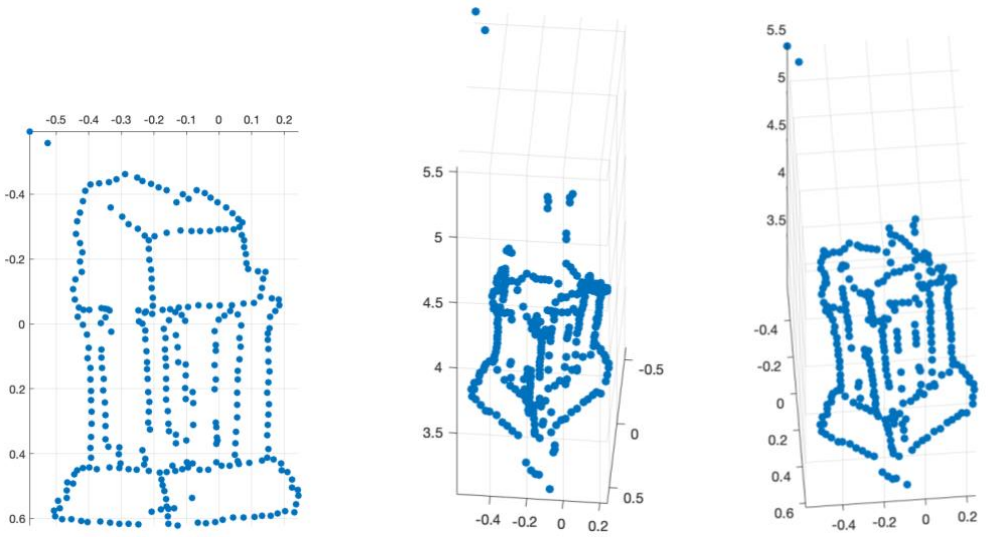
Here we get 4 projection matrix candidates for camera2 from the essential matrix. The correct configuration is the one for which most of the 3D points are in front of both cameras (positive depth). So, we count the number of points with positive z coordinates for all points that are projected into 3D from their corresponding 2D image point. The maximum of these is the correct extrinsic matrix.

Reprojection error for pts1 and pts2:
error1 = [0.5022, 0.5022, 0.5021, 0.5021]
error2 = [0.5130, 0.5130, 0.5128, 0.5128]
As you can see the reprojection error is less than 1 pixel.

Here you can see the number of positive z: [90 198 0 288]. So, we choose 4th projection matrix.

3.1.5 Write a test script that uses templeCoords

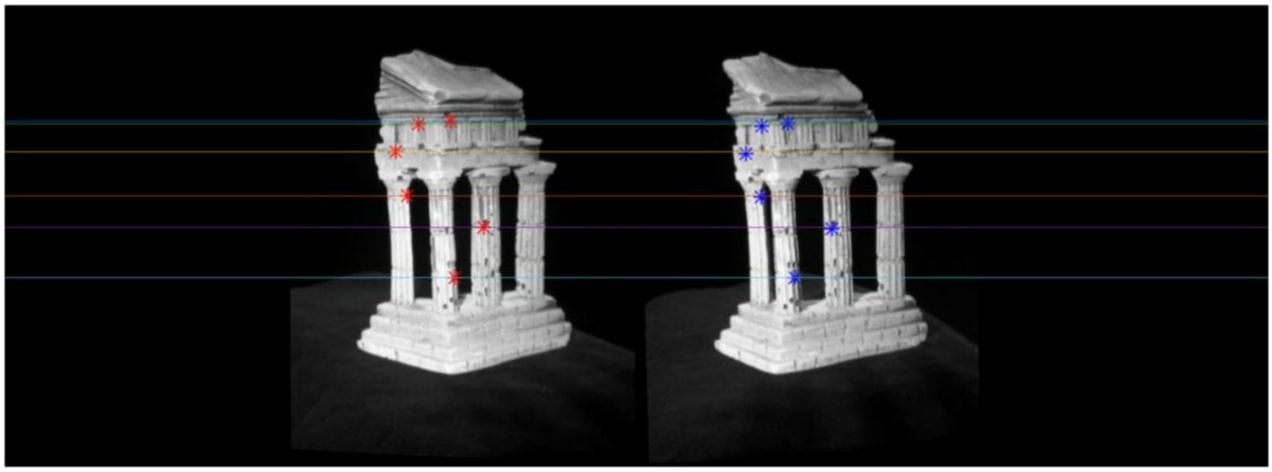
Here is the result for triangulate.m from 3 differnet angles by running testTempleCoords.m.



3.2 Dense reconstruction

3.2.1 Image rectification

Here is the result for rectify_pair.m by running testRectify.m.

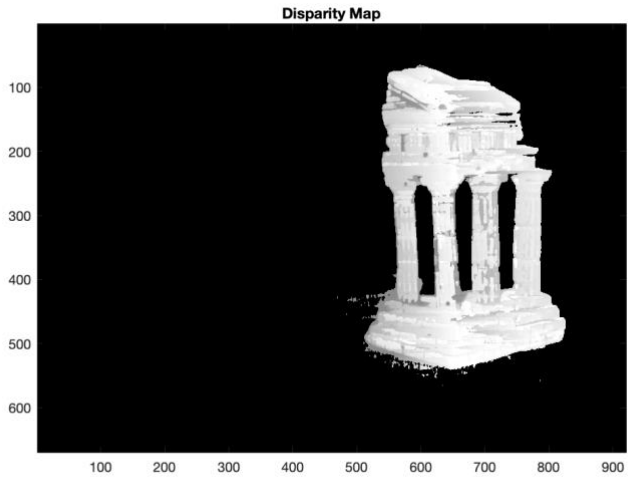


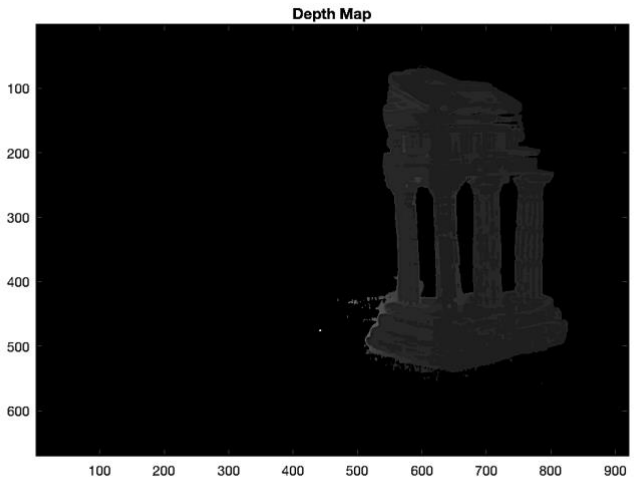
3.2.2 Dense window matching to find per pixel density

You can see the result for get_disparity.m in the next part.

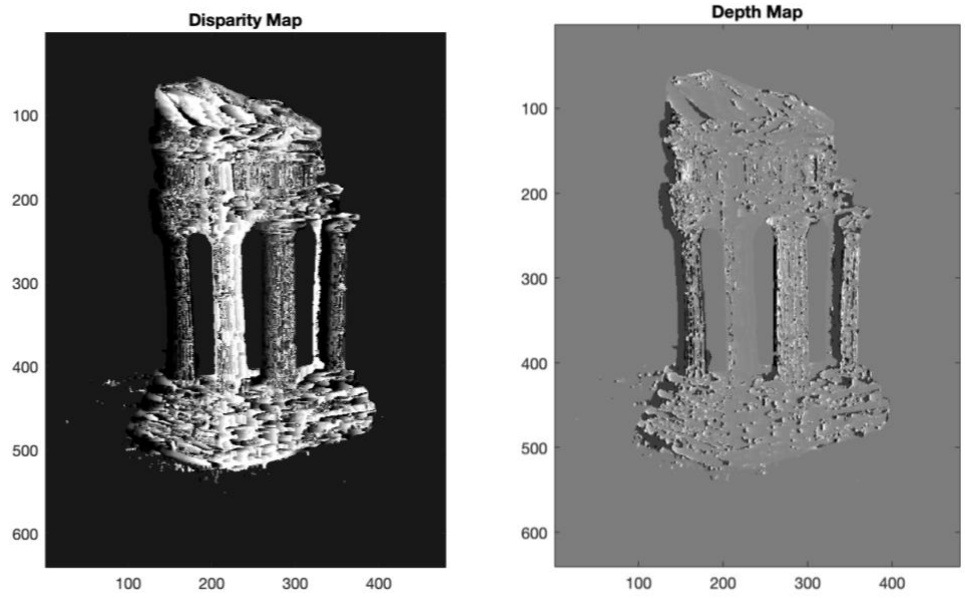
3.2.3 Depth map

Here is the result for get_depth.m and get_disparity.m by running testDepth.m after rectifying.





Results without rectifying:



3.3 Pose estimation

3.3.1 Estimate camera matrix P

Here is the result for estimate_pose.m by running testPose.m.

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 2.0380
Pose Error with noisy 2D points is 0.0244

3.3.2 Estimate intrinsic/extrinsic parameters

Here is the result for estimate_params.m by running testKRt.m.

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 0.7848
Rotation Error with clean 2D points is 0.0999
Translation Error with clean 2D points is 0.1824

3.3.3 Project a CAD model to the image

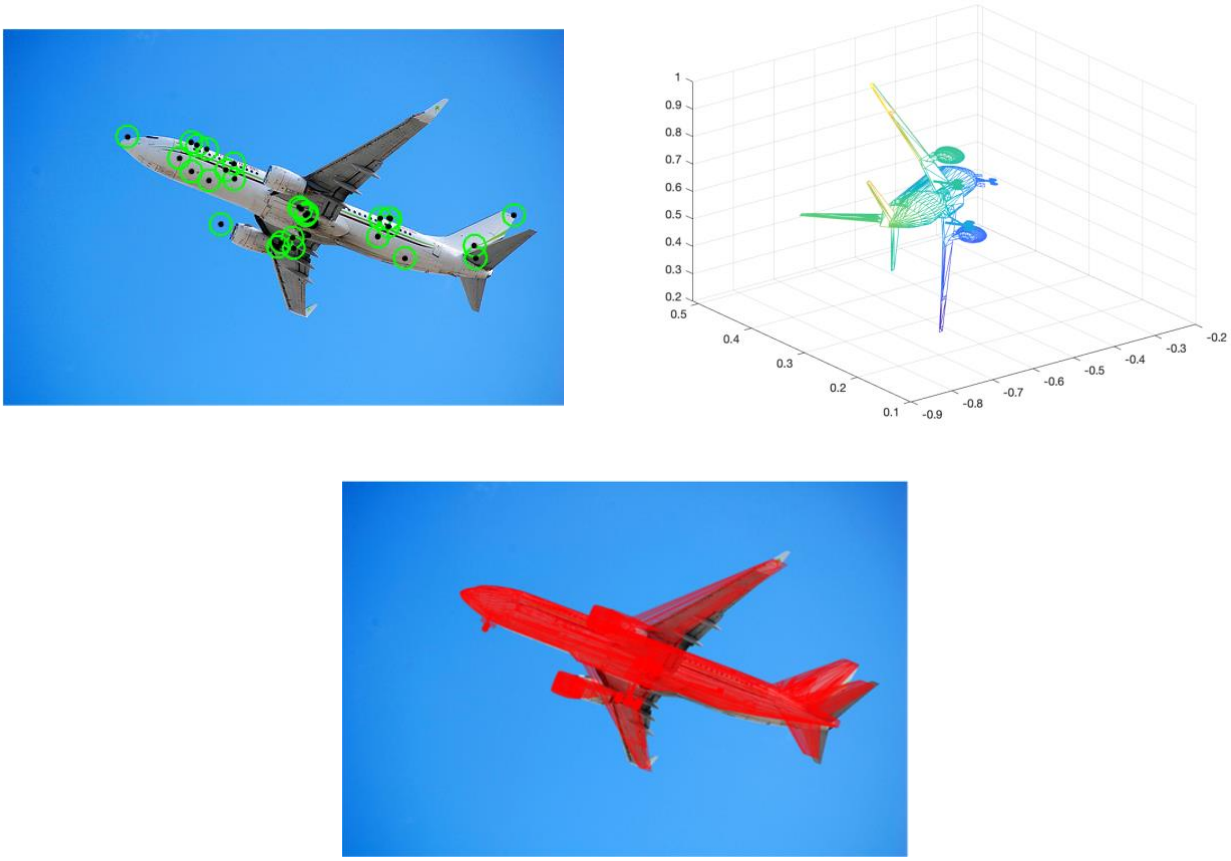


Figure: Project a CAD model back onto the image. Left: the image annotated with given 2D points (blue circle) and projected 3D points (red points). Right: the CAD model rotated by estimated R. Middle: the image overlapping with projected CAD model.