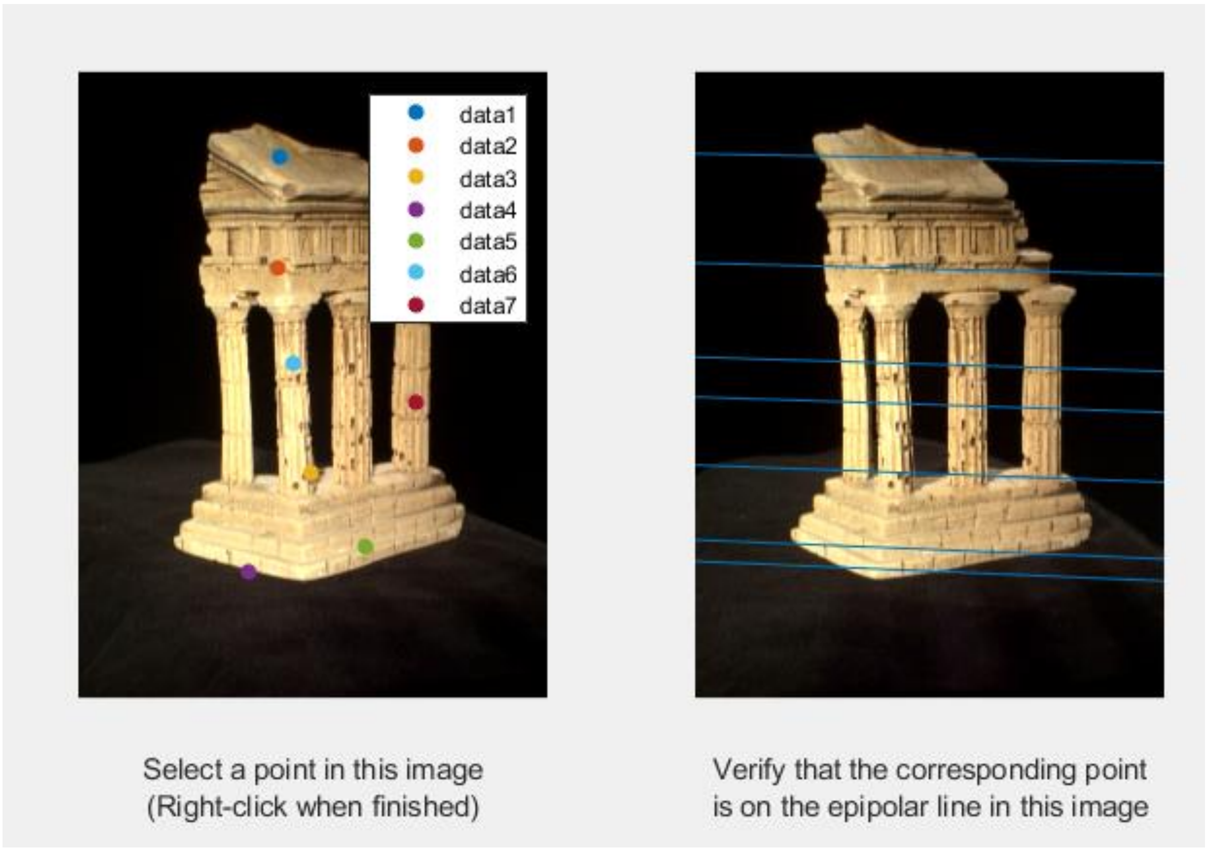
3.1.1) Implement the eight-point algorithm

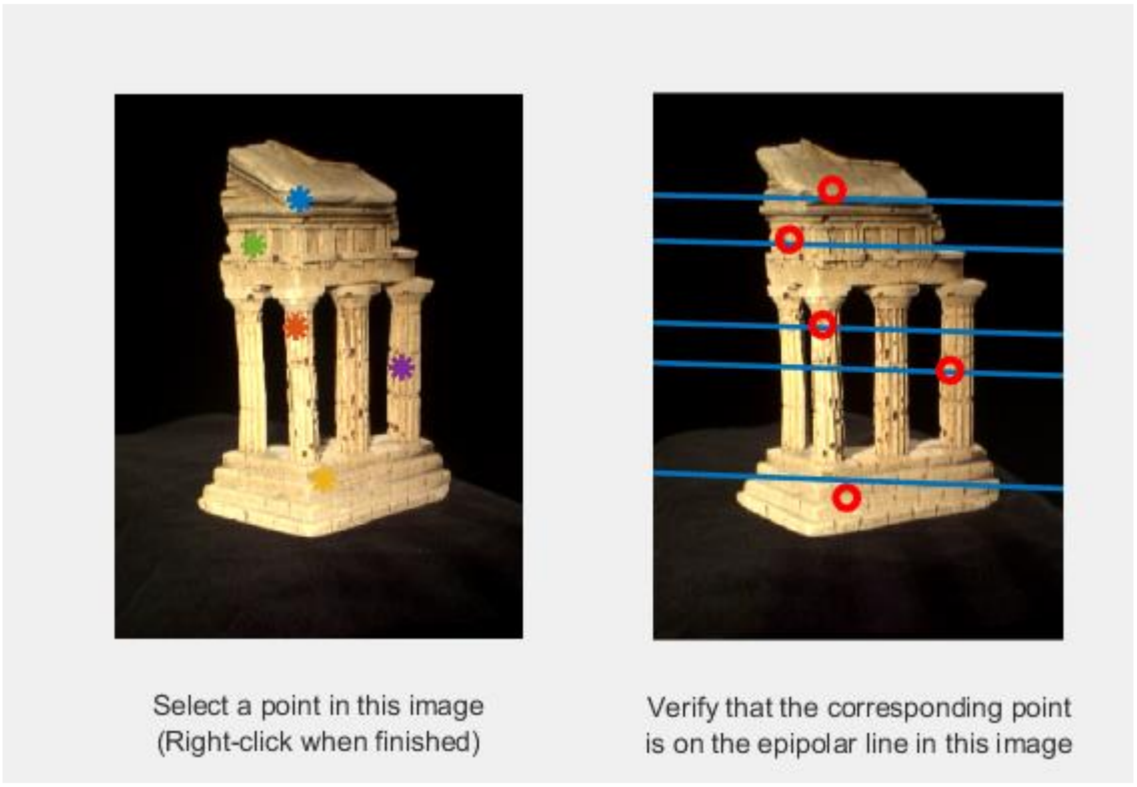
- The eight-point algorithm has been achieved by estimating the Fundamental matrix using the set of more than 8 corresponding image points.
- U has been constructed and SVD has been performed. The fundamental matrix F has been calculated by reshaping to a 3*3 matrix.
- The fundamental matrix F has been unnormalized.


data1
data2
data3
data4
data5
data6
data7
Select a point in this image
(Right-click when finished)
Verify that the corresponding point
is on the epipolar line in this image

- Figure 1: -Visualizations of some epipolar lines shown as per the eight point algorithm.

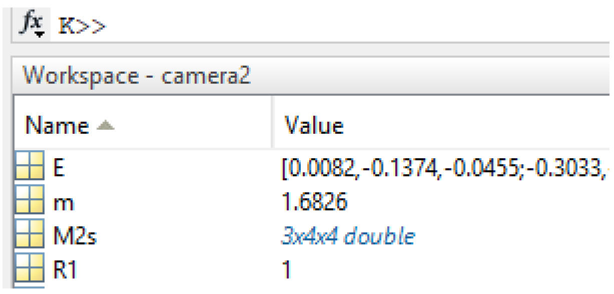
3.1.2) Find epipolar correspondences: -

Point pairs are identified and calculated. Using the fundamental matrix calculated, we estimate the corresponding epipolar line which are in image 1 to the point in image 2. By creating a small window of size w, it is now matched on the first image with the second image. Have called the epipolarMatchGUI method for implementing the epipolarCorrespondance and found the corners and dots as expected.


Select a point in this image
(Right-click when finished)
Verify that the corresponding point
is on the epipolar line in this image

- Figure 2:- Showing the epipolar correspondence points.

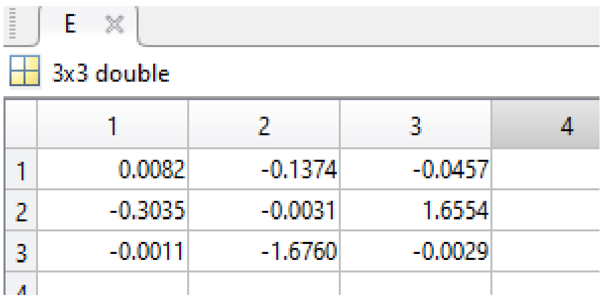
By computing the det() function for all the rotation matrices, it has been found that R1 contains the best solution among all the 4. det(R1)=1



```
fx K>>
Workspace - camera2
Name    Value
E       [0.0082,-0.1374,-0.0455;-0.3033,
m       1.6826
M2s     3x4x4 double
R1      1
```

3.1.3) Write a function to compute the essential matrix: -

Using the intrinsic matrix values K1 and K2 for both the images and with the calculated fundamental matrix F, essential matrix E has been calculated. The 3*3 essential matrix value received is as follows: -



E

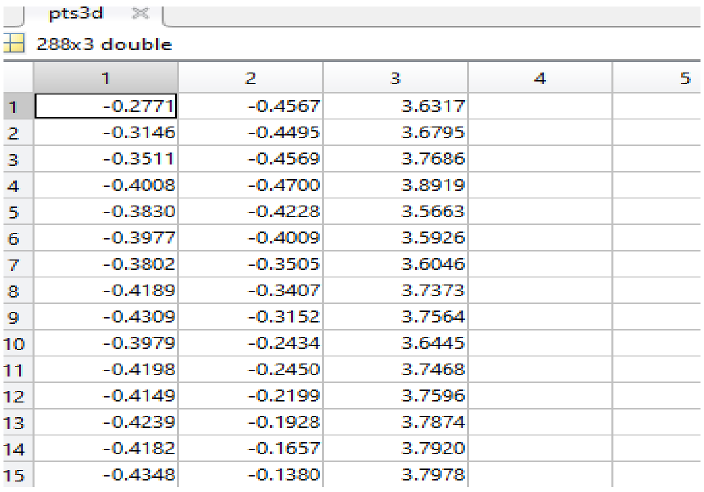
3x3 double

	1	2	3	4
1	0.0082	-0.1374	-0.0457	
2	-0.3035	-0.0031	1.6554	
3	-0.0011	-1.6760	-0.0029	

- Figure 3 :- Showing the Essential Matrix E value

3.1.4) Implement triangulation: -

Triangulation has been implemented using the pts1 and pts2 which are N*2 matrices with the 2D image coordinates and with P1 and P2 values which are the 3*4 projection matrices. The correct pts3d which N*3 matrix has been generated successfully.



pts3d

288x3 double

	1	2	3	4	5
1	-0.2771	-0.4567	3.6317		
2	-0.3146	-0.4495	3.6795		
3	-0.3511	-0.4569	3.7686		
4	-0.4008	-0.4700	3.8919		
5	-0.3830	-0.4228	3.5663		
6	-0.3977	-0.4009	3.5926		
7	-0.3802	-0.3505	3.6046		
8	-0.4189	-0.3407	3.7373		
9	-0.4309	-0.3152	3.7564		
10	-0.3979	-0.2434	3.6445		
11	-0.4198	-0.2450	3.7468		
12	-0.4149	-0.2199	3.7596		
13	-0.4239	-0.1928	3.7874		
14	-0.4182	-0.1657	3.7920		
15	-0.4348	-0.1380	3.7978		

Figure 4: - pts3d values for N rows. First 15 has been shown here.

The re projection error has been calculated by computing the Euclidean error between the projected 2D points and pts1.

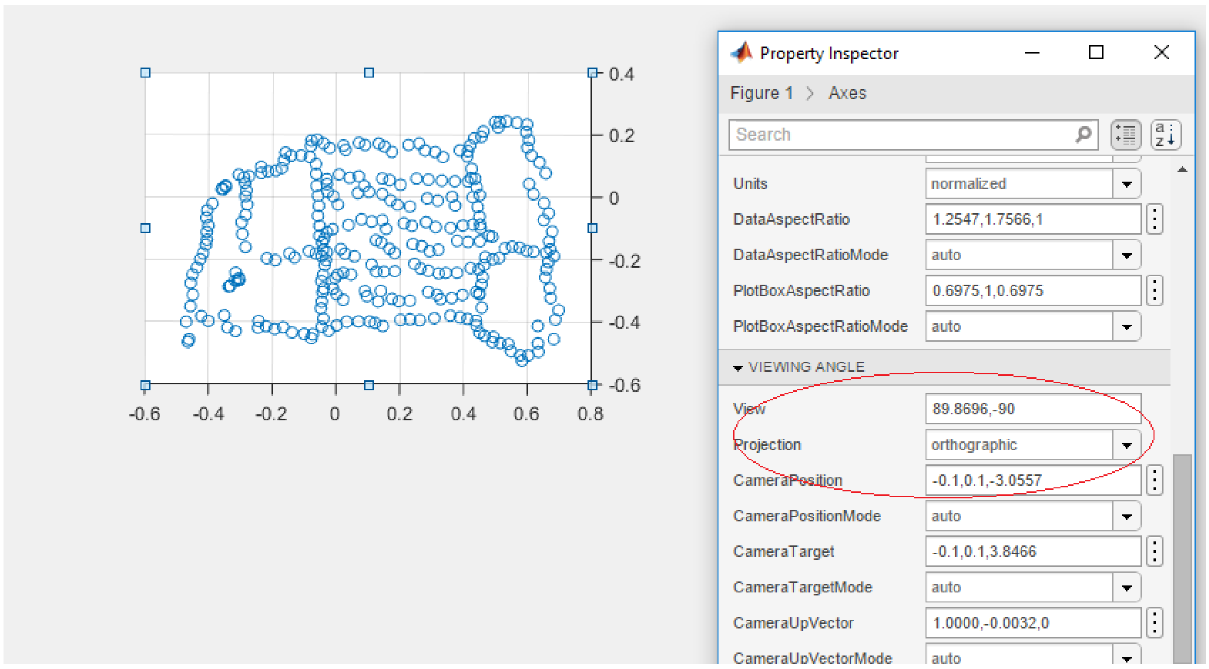
```
0.4536    0.4158    0.4115    0.4080    0.2297
0.0415    0.0524    0.0589    0.0661    0.0576
0.0000    0.0000    0.0000    0.0000    0.0000
```

Figure 5: - Reprojection error for each pixel clearly says that it’s less than 1 pixel.

3.1.5) Write a test script that uses templeCoords

testTempleCoords.m file has been executed for generating a 3D reconstruction of the temple images provided. After the implementation, I have shown three the below point correspondences of the temple reconstruction shown with different angles.

Sample 1: -

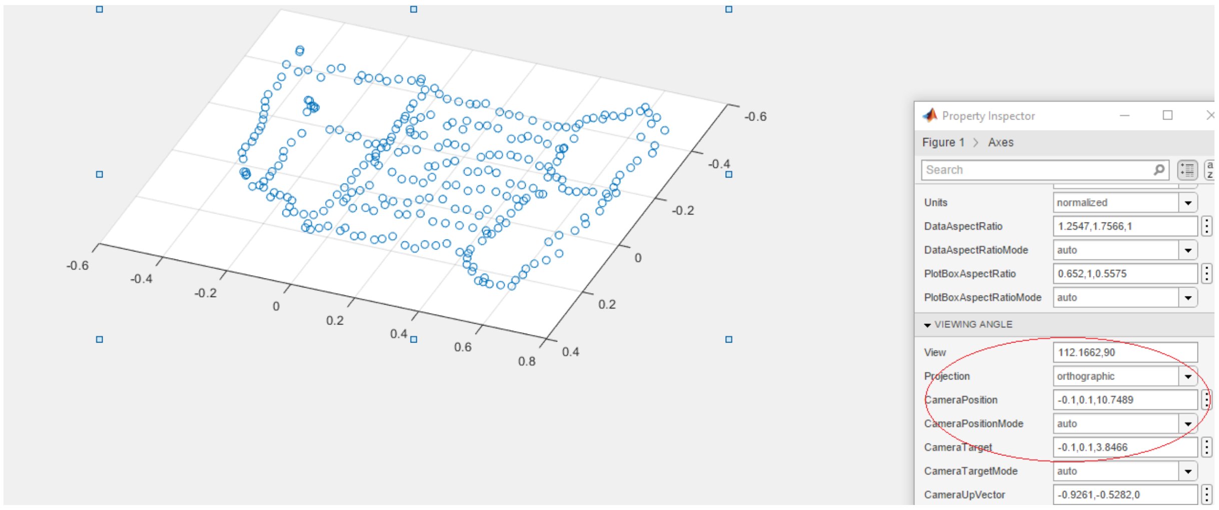


Property Inspector

Figure 1 > Axes

Units	normalized
DataAspectRatio	1.2547,1.7566,1
DataAspectRatioMode	auto
PlotBoxAspectRatio	0.6975,1,0.6975
PlotBoxAspectRatioMode	auto
VIEWING ANGLE	
View	89.8696,-90
Projection	orthographic
CameraPosition	-0.1,0.1,-3.0557
CameraPositionMode	auto
CameraTarget	-0.1,0.1,3.8466
CameraTargetMode	auto
CameraUpVector	1.0000,-0.0032,0
CameraUpVectorMode	auto

Sample 2: -

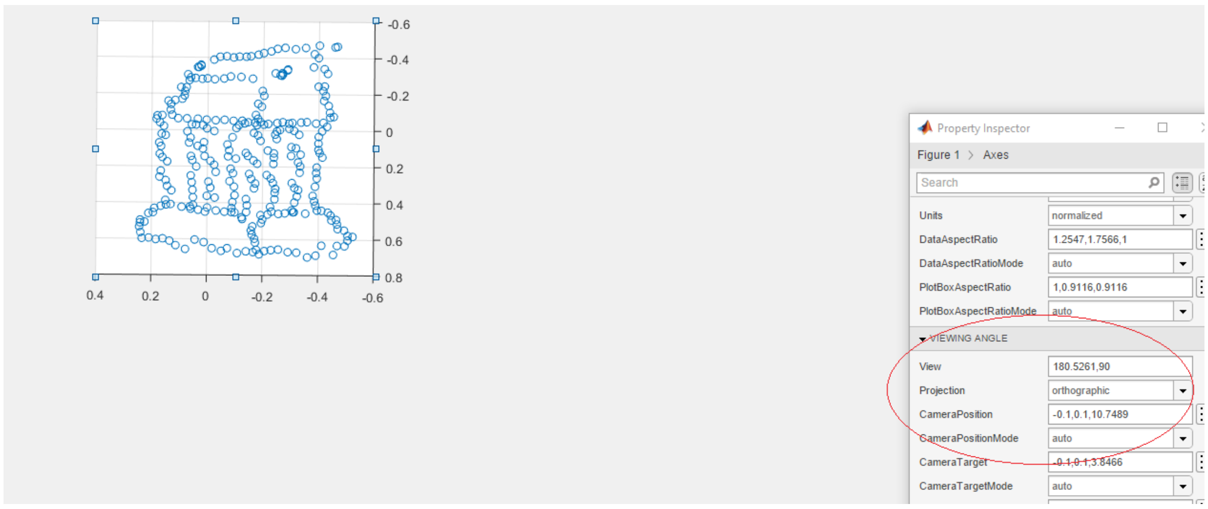


Property Inspector

Figure 1 > Axes

Units	normalized
DataAspectRatio	1.2547,1.7566,1
DataAspectRatioMode	auto
PlotBoxAspectRatio	0.652,1,0.5575
PlotBoxAspectRatioMode	auto
VIEWING ANGLE	
View	112.1662,90
Projection	orthographic
CameraPosition	-0.1,0.1,10.7489
CameraPositionMode	auto
CameraTarget	-0.1,0.1,3.8466
CameraTargetMode	auto
CameraUpVector	-0.9261,-0.5282,0

Sample 3: -



Units	normalized
DataAspectRatio	1.2547,1.7566,1
DataAspectRatioMode	auto
PlotBoxAspectRatio	1,0.9116,0.9116
PlotBoxAspectRatioMode	auto
VIEWING ANGLE	
View	180.5261,90
Projection	orthographic
CameraPosition	-0.1,0.1,10.7489
CameraPositionMode	auto
CameraTarget	-0.1,0.1,3.8466
CameraTargetMode	auto

3.2.1) Image rectification

testRectify.m file has been executed successfully. The rotation and translation values are used from the extrinsics.mat file which I have saved in the ../data path for passing the value in rectify_pair.m.

- The optical centres c1 and c2 are calculated from each camera.
- The rotational matrices r1,r2,r3 has calculated respectively.
- The new intrinsic parameter and the translation is identified.
- The rectification matrix for both the camera's are acquired.

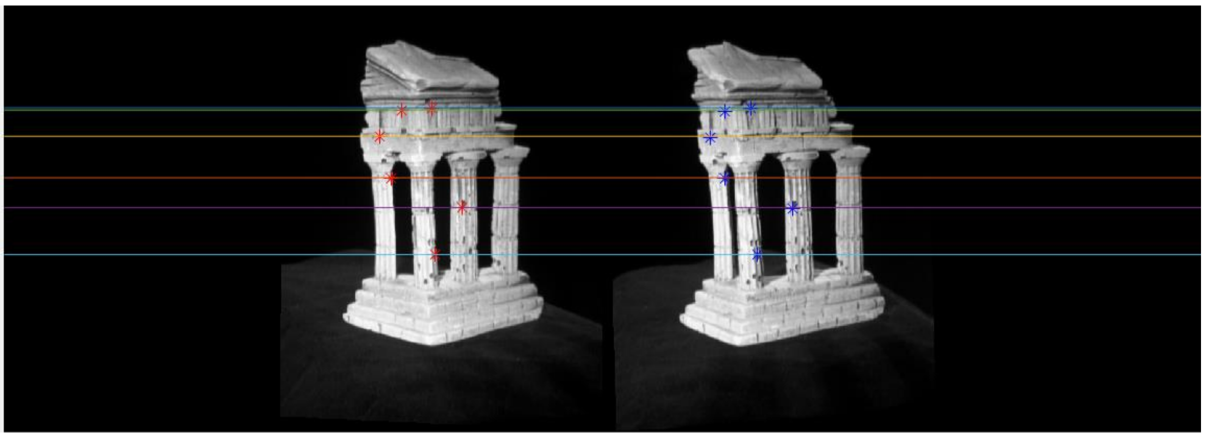


Figure 6: - The results are showing that the epipolar lines are horizontal with the corresponding points in the image lying on the same line.

3.2.2) Dense window matching to find per pixel density

By running testDepth.m the disparity map & depth map are shown as follows: -

The disparity map has been calculated by setting the window size by w = (windowSize-1)/2. The disparity map has been calculated and shown in the below figure.

A disparity map has been created for the temple image. The below figure shows that the disparity map has the same dimension as the images im1 and im2.

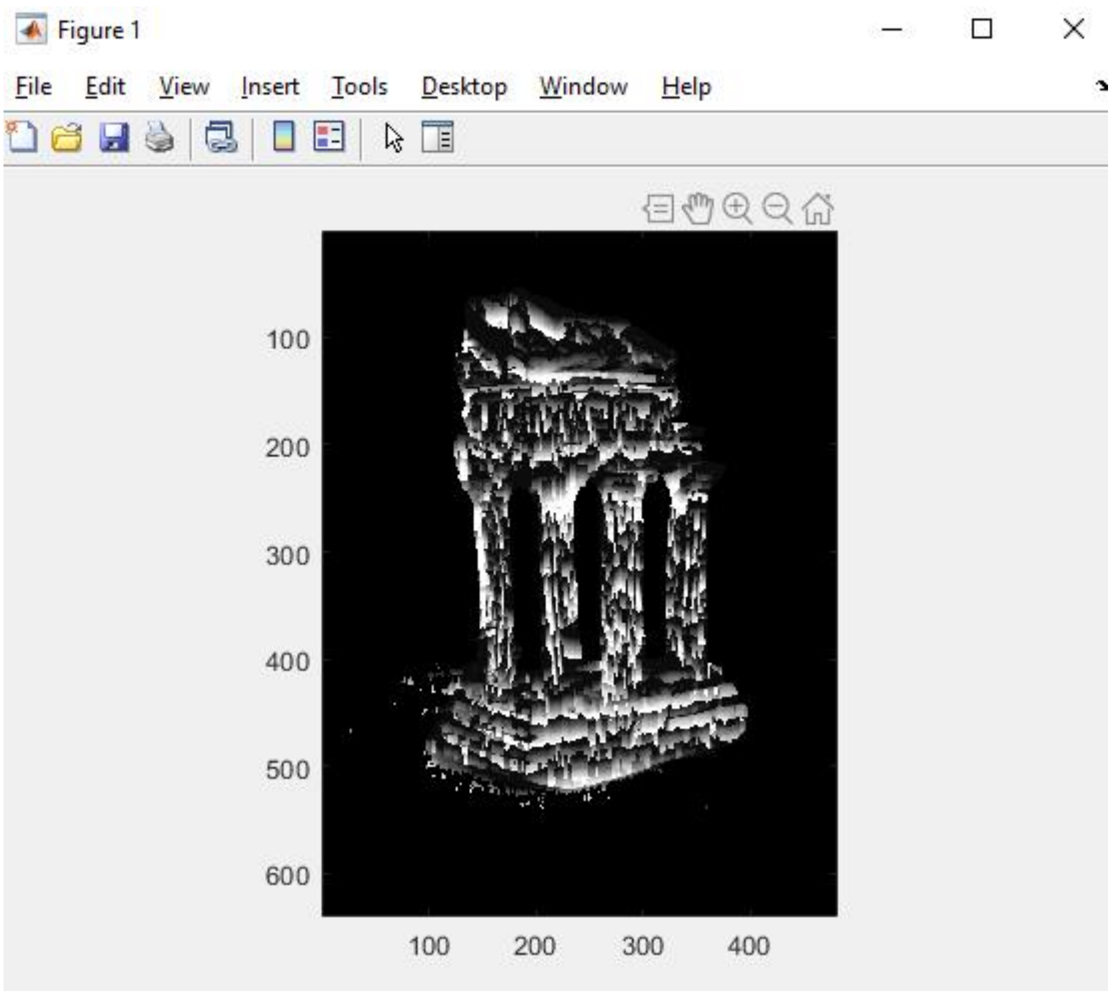


Figure 7: - Figure showing the Disparity map

3.2.3) Depth map

Depth map has been calculated using the identified disparity map using the formula depthM(y, x) = b × f /dispM(y, x). The depth map function first rectifies the images and then computes the depth map.

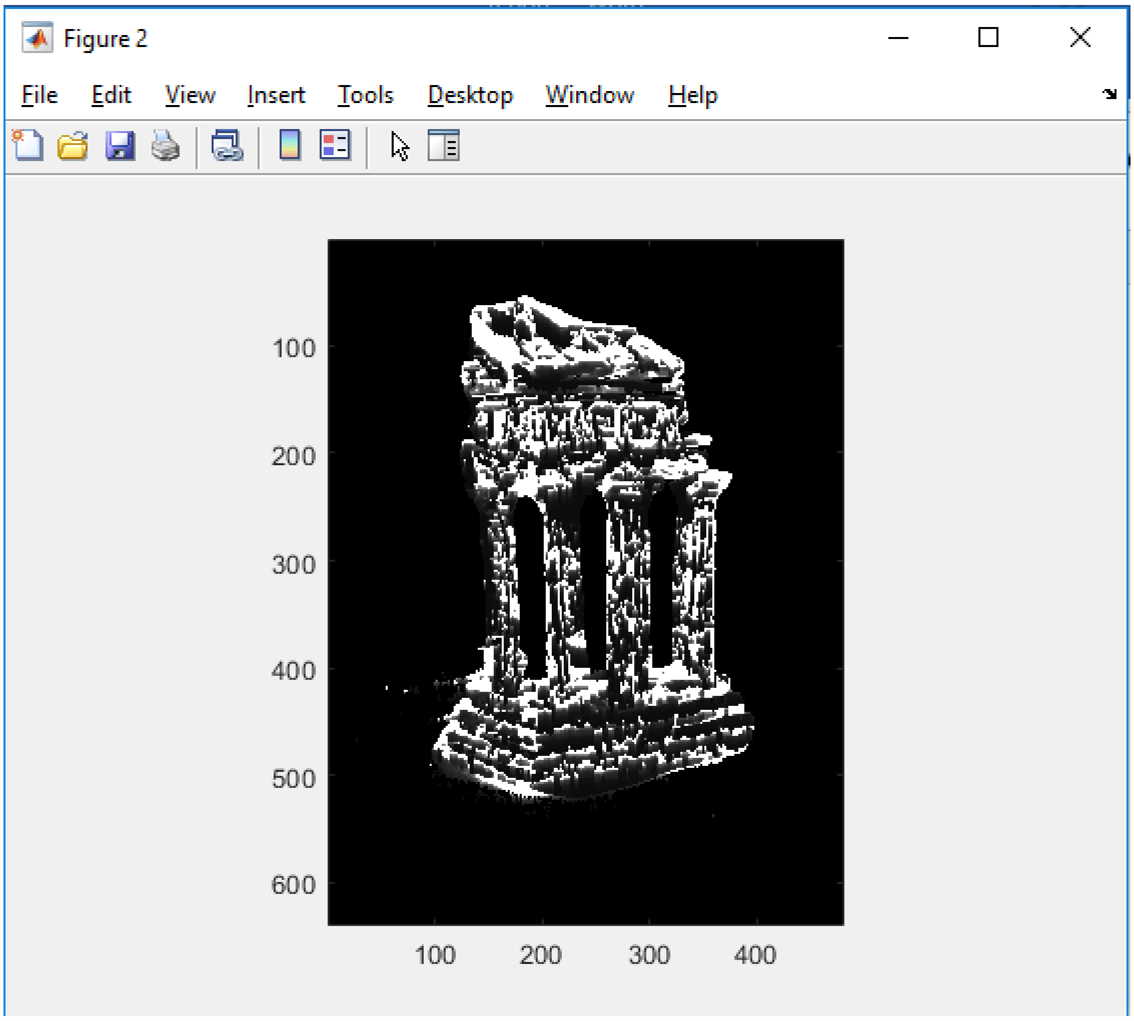


Figure: - Figure showing the Depth map

EXTRA CREDITS

3.3.1 Estimate camera matrix P

The camera matrix has been computed. The reprojected and pose errors with clean and noisy 2D points are calculated and I have provided them below:-

```
>> 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 1.9251
Pose Error with noisy 2D points is 0.0289
```

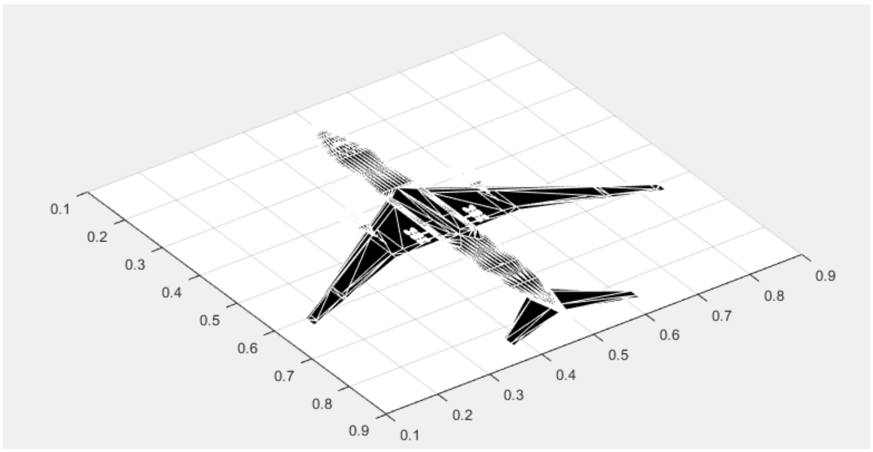
3.3.2 Estimate intrinsic/extrinsic parameters

The intrinsic parameter and the extrinsic parameters K,R, t has been calculated as Intrinsic, Rotation and Translation errors respectively to the clean and adding few noise. The performance is as below:-

```
>> 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 0.6065
Rotation Error with clean 2D points is 0.0364
Translation Error with clean 2D points is 0.3989
```

3.3.3 Project a CAD model to the image

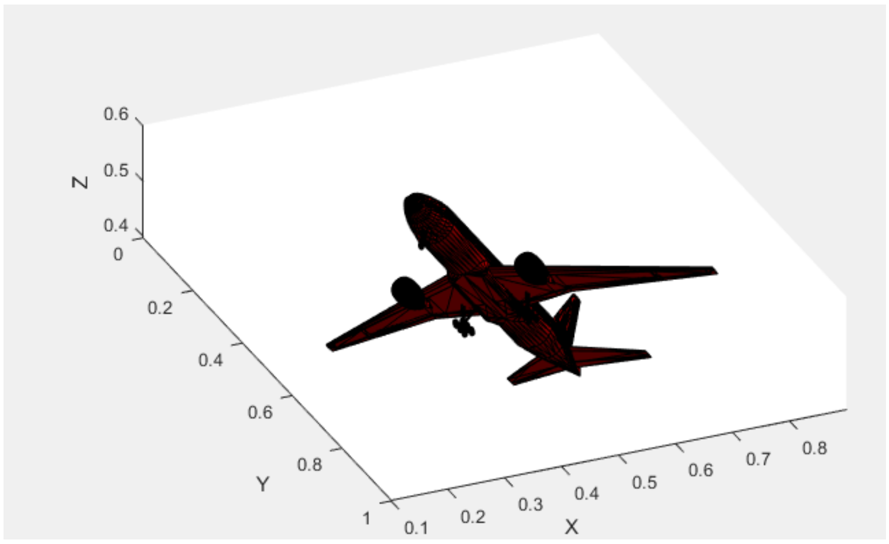
The image image has been loaded on to projectCAD.m .The camera matrix P, intrinsic matrix K, rotation matrix R, and translation t has been calculated by running the estimate_pose.m and estimate_params.m file. With the new projected matrix, 2D points x and the projected 3D points are plotted.



After the patch applied for the CAD model, the figure is displayed as below

Projected CAD Model

With Projected CAD Model using patch, I tried to change the Face Colour property with value to Red.


Z
0.6
0.5
0.4
0
0.2
0.4
0.6
0.8
Y
1
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
X