

Project 5

3D Reconstruction

Name: Muhammad Shahzaib Waseem

Sparse Reconstruction

Eight Point Algorithm

The original F matrix looks like this:

```
>> runner

F =

-0.0000    0.0000   -0.0000
 0.0000    0.0000   -0.0015
-0.0000    0.0015    0.0064
```

Figure 1: Recovered Fundamental Matrix F

The output of `eightpoint.m` looks like this:

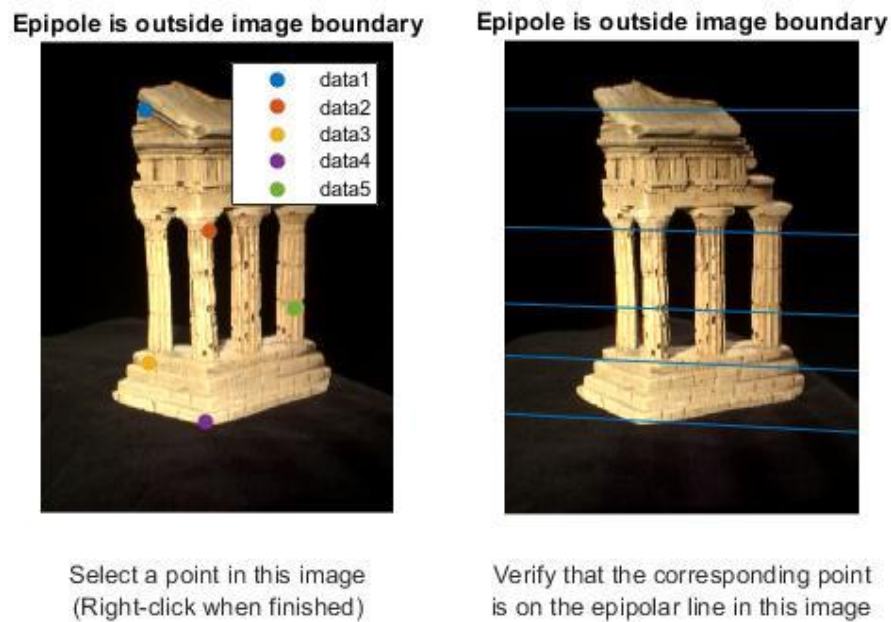


Figure 2: Epipolar Lines

The epipolar correspondences from temple image 1 to temple image 2 look like the following:

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

The screenshot displays the MATLAB R2019a environment. The main window shows a script titled 'epipolarCorrespondence.m'. The script defines two images, 'img1' and 'img2', as grayscale images. It calculates the fundamental matrix 'F' from image 1 to image 2. It then defines the coordinates of points in image 1 and image 2. The script sets the window size, sigma, and loop size. It defines the epipolar lines for a given point in image 1. The script then finds the intersection of the epipolar lines in image 2. The script includes two images of a classical building facade. The first image shows the building with epipolar lines and a point selection interface. The second image shows the building with epipolar lines and a point selection interface. The command window shows the coordinates of the selected point in both images.

```

1 function [pts2] = epipolarCorrespondence(img1, img2, F, pts1)
2 % epipolarCorrespondence:
3 % Args:
4 %   img1: Image 1
5 %   img2: Image 2
6 %   F: Fundamental Matrix from img1 to img2
7 %   pts1: coordinates of points in image 1
8 % Returns:
9 %   pts2: coordinates of points in image 2
10
11 window_size = 17;
12 h = floor(window_size/2);
13 sigma = 5;
14 loop_size = 28;
15
16 img1 = im2double(rgb2gray(img1));
17 img2 = im2double(rgb2gray(img2));
18
19 pts2 = zeros(size(pts1));
20 pts1 = [pts1, ones(size(pts1), 1), 1];
21
22 kernel = special('gaussian', window_size, window_size, sigma);
23 % projections = round(cross(img1_lines, [-epi_line(2), epi_line(1), epi_line(2) * x1 - epi_line(1) * y1]));
24
25 for N=size(pts1, 1)
26     least_error = Inf;
27
28     x1 = pts1(N, 1); y1 = pts1(N, 2);
29     window_img1 = img1(y1-h:y1+h, x1-h:x1+h);
30     epi_line = F * pts1(N, :);
31     epi_line = epi_line / norm(epi_line(2:3));
32
33     projections = round(cross(epi_line, [-epi_line(2), epi_line(1), epi_line(2) * x1 - epi_line(1) * y1]));
34     x2 = projections(1); y2 = projections(2);
35
36     for x=2:loop_size:x2-loop_size
37         for y=2:loop_size:y2-loop_size
38             window_img2 = img2(y-h:y+h, x-h:x+h);
39             % error = sum(abs(window_img1 - window_img2));
40             error = sqrt(sum(kernel .* (window_img1 - window_img2).^2, 'all'));
41             if (error == Least_error)
42                 least_error = error;
43                 pts2(N, :) = [x, y, 1];
44             end
45         end
46     end
47 end

```

Figure 1

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

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

```

coordsIM1 =
215.5601 129.5852
166.8812 360.7894
117.5508 471.2853
225.3458 228.0353
263.1178 177.2339
226.3953 586.9617

coordsIM2 =
196 118
167 339
117 464
168 318
261 181
208 565

```

The similarity metric I employed for this task was Euclidean distance, which is given by:

$$error(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

or

```
error = sqrt(sum(kernel .* (window_im1 - window_im2).^2, "all"))
```

This will take a subset from both images and subtract them, square the residual, and sum all the values in the kernel and take the root of the resulting value. The window size for this is set to be 17 pixels.

Explanation

The algorithm failed when there are a lot of points (or windows) which are similar along the epipolar line, as it is visible from the [Image 3](#) that the purple point lies on the stairs which can be repeated multiple times throughout the image, so this makes the algorithm fail for these certain cases.

Compute the Essential Matrix (E)

The essential matrix, E, looks like this:

```
>> runner

E =

   -0.0025    0.4070    0.0476
    0.1863    0.0127   -2.2833
    0.0076    2.3114    0.0026
```

Figure 5: Essential Matrix E

Triangulation

Explanation

I ran a loop for all the four candidate extrinsic matrices (P2) and ran the whole algorithm and measured the positive depth pixels (pixels which are in front of the camera and not behind). The best performing matrix was saved and used.

Errors

The two reprojection errors are: reproj_1 = **0.2783** and reproj_2 = **0.2711** using the points provided in the `templeCoords.mat` file.

If I calculate the reprojection error using the points provided by `someCorresp.mat` I get errors equal to: reproj_1 = **0.2744** and reproj_2 = **0.2670**.

All together

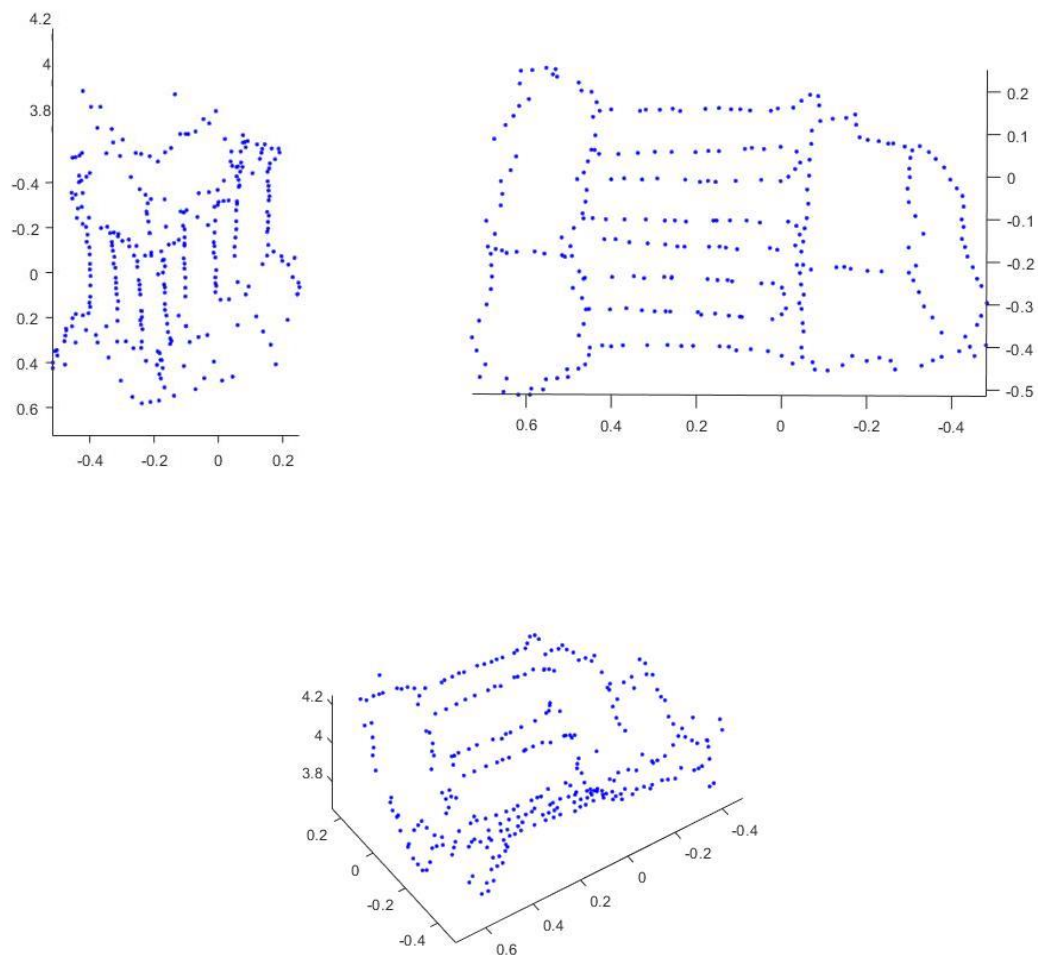


Figure 6: Sparse reconstruction of templeCoords

Dense Reconstruction

Image rectification

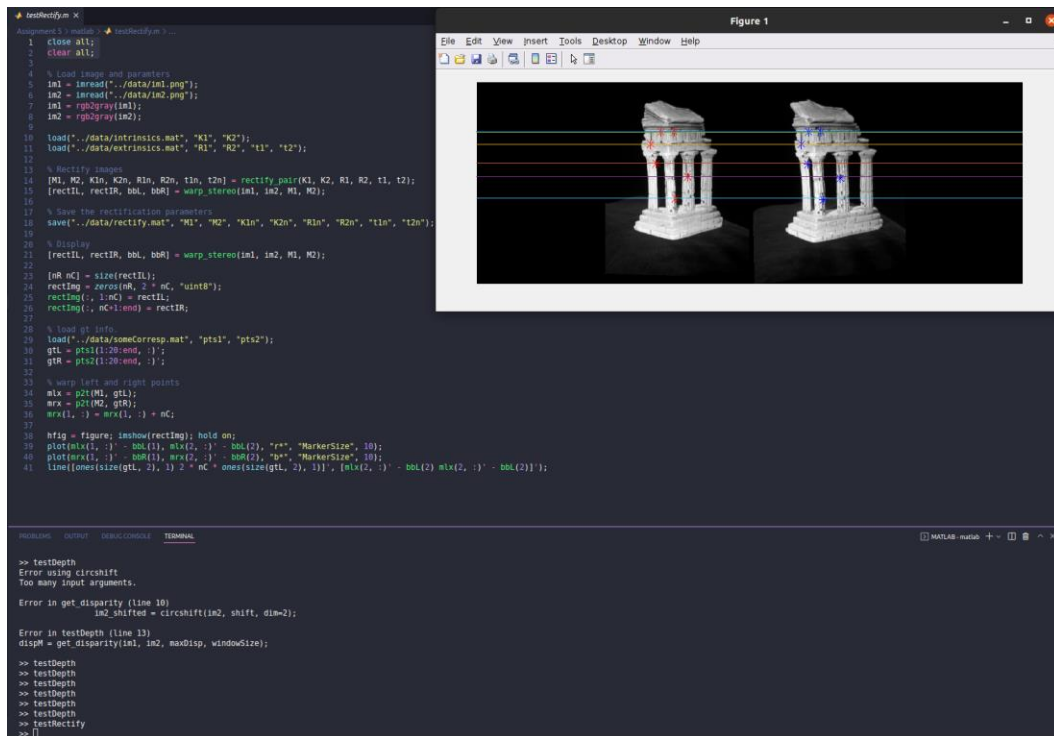


Figure 7: Screenshot of the whole screen while running testRectify.m

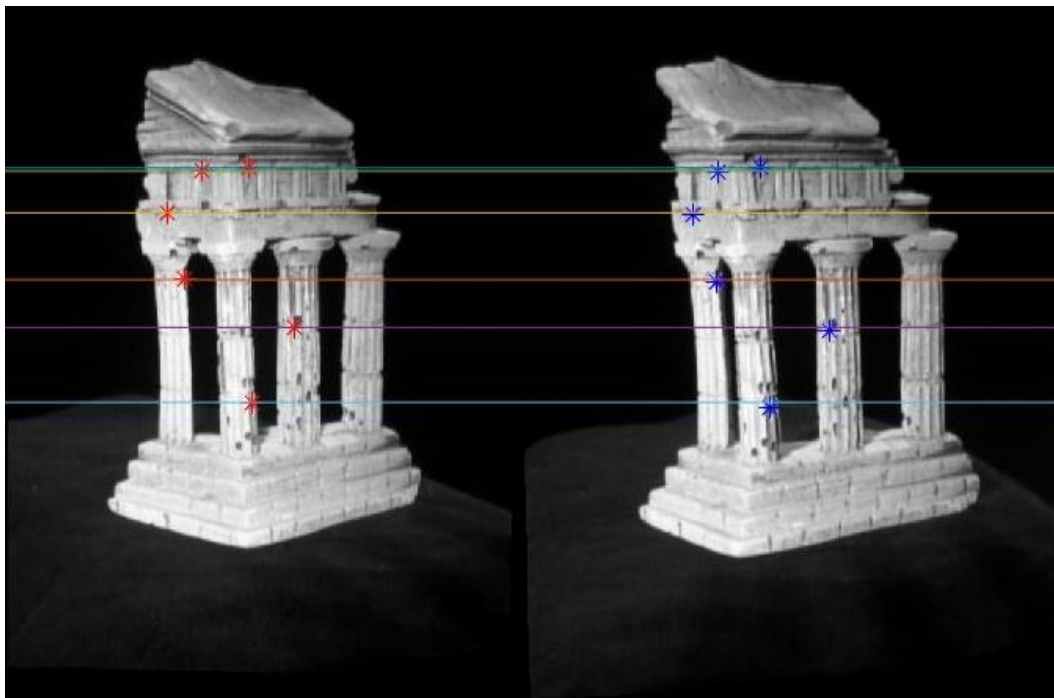


Figure 8: Rectify Image (Enlarged Image)

Dense window matching



Figure 9: Disparity Map

Depth map



Figure 11: Depth Map

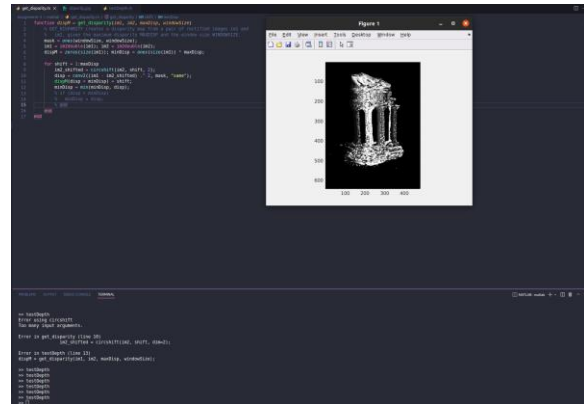


Figure 10: Screenshot of the whole screen

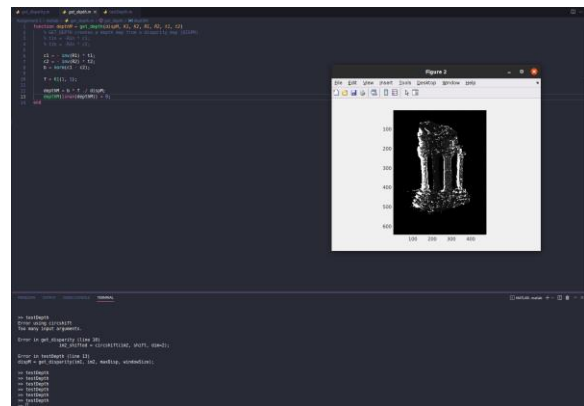


Figure 12: Screenshot of the whole screen

Pose Estimation

Estimate camera matrix P

```
>> cd ../ec
>> 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 2.3516
Pose Error with noisy 2D points is 0.1831
>> 
```

Figure 13: Estimate Camera Matrix P (testPose.m script)

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 0.5802
Rotation Error with clean 2D points is 0.1829
Translation Error with clean 2D points is 0.2097
>> 
```

Figure 14: Estimate intrinsic/extrinsic parameters (testKRt.m script)

Project a CAD model to the image



Figure 15: 3D and 2D projections on the airplane

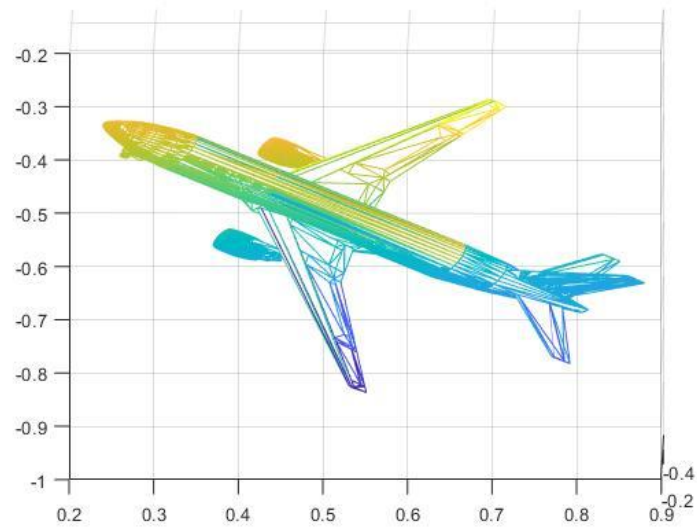


Figure 16: trimesh graph of the CAD model



Figure 17: CAD model overlayed on the airplane image