

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:

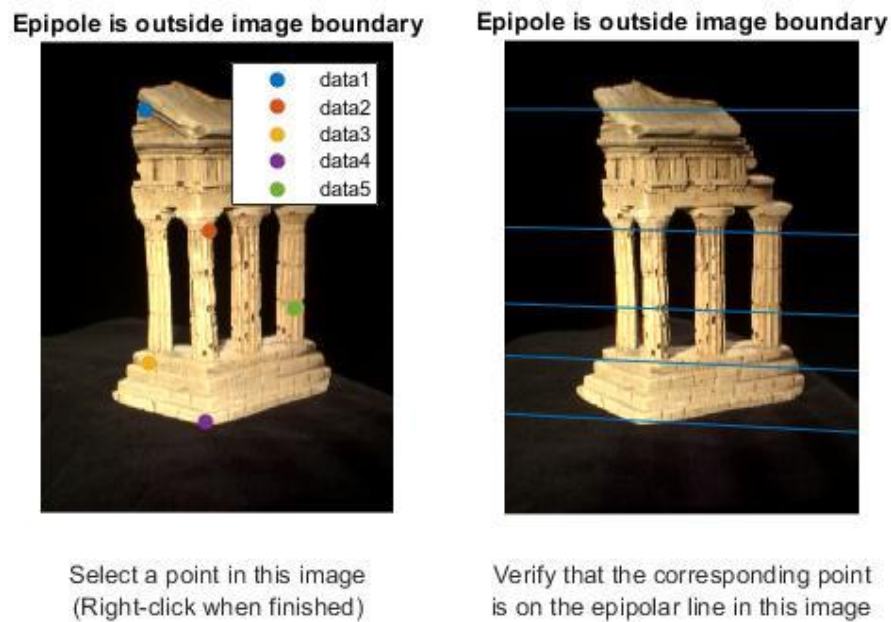


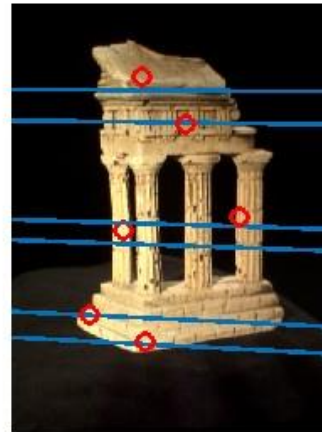
Figure 2: Epipolar Lines

Epipolar correspondences

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



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



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

Figure 3: Epipolar Correspondence between the two temple images

```
function [pts2] = epipolarCorrespondence(in1, in2, F, pts1)
% epipolarCorrespondence
%
% Args:
%   in1: Image 1
%   in2: Image 2
%   F: Fundamental Matrix from in1 to in2
%   pts1: coordinates of points in Image 1
% Returns:
%   pts2: coordinates of points in Image 2
window_size = 17;
h = floor(window_size/2);
sigma = 5;
loop_size = 20;

in1 = im2double(rgb2gray(in1));
in2 = im2double(rgb2gray(in2));
pts2 = zeros(size(pts1));
pts1 = [pts1, ones(size(pts1), 1, 1)];

kernel = special('gaussian', [window_size, window_size], sigma);
% projection = conv2(in1, kernel, 'same'); % pts1, in2, 2) * pts1(1, 1) + epi_line
for N=1:size(pts1, 1)
    least_error = Inf;
    x1 = pts1(N, 1); y1 = pts1(N, 2);
    window_in1 = in1(y1-h:y1+h, x1-h:x1+h);
    epi_line = F * pts1(N, 1:2);
    epi_line = epi_line / norm(epi_line(1:2));
    projections = round(cross(epi_line, [-epi_line(2), epi_line(1), epi_line(1) * x1 - epi_line(2) * y1]));
    x2 = projections(1); y2 = projections(2);
    for x1=x2-loop_size:x2+loop_size
        for y1=y2-loop_size:y2+loop_size
            window_in2 = in2(y1-h:y1+h, x1-h:x1+h);
            % epi_line = conv2(window_in2, kernel, 'same'); % pts1(1, 1) + epi_line
            error = sqrt(sum(kernel .* (window_in1 - window_in2) ^ 2, 'all'));
            if (error < least_error)
                least_error = error;
                pts2(N, :) = [x1 y1];
            end
        end
    end
end

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

coordsIN1 =

215 3601 129 5052
160 0812 360 7094
117 2060 471 2053
225 3450 328 6393
263 1178 177 2539
220 3053 508 0837

coordsIN2 =

196 118
167 339
117 464
348 318
261 181
208 505

>> []
```

Figure 4: Screenshot with the implementation

Similarity Metric

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, like on the stairs and if the window size is huge in the columns (did not try it though) or the top of the temple or the window like architecture in the top part of the temple 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

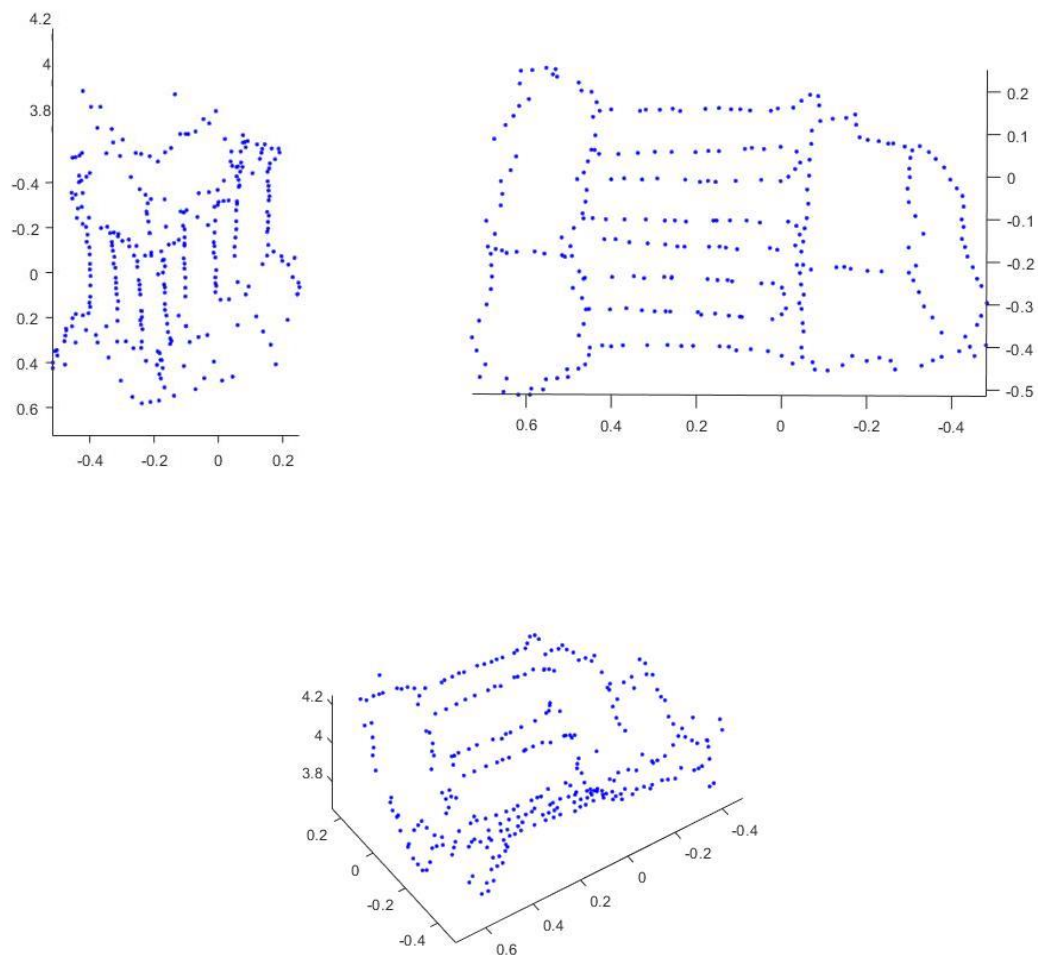


Figure 6: Sparse reconstruction of templeCoords

Dense Reconstruction

Image rectification

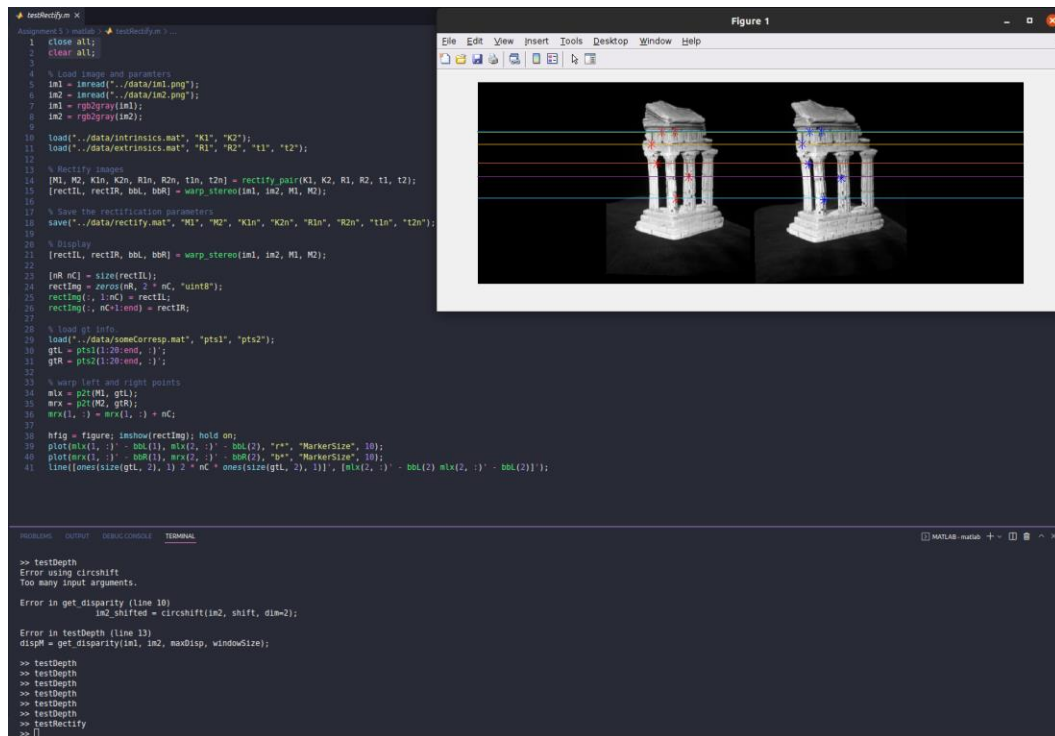


Figure 7: Screenshot of the whole screen while running testRectify.m

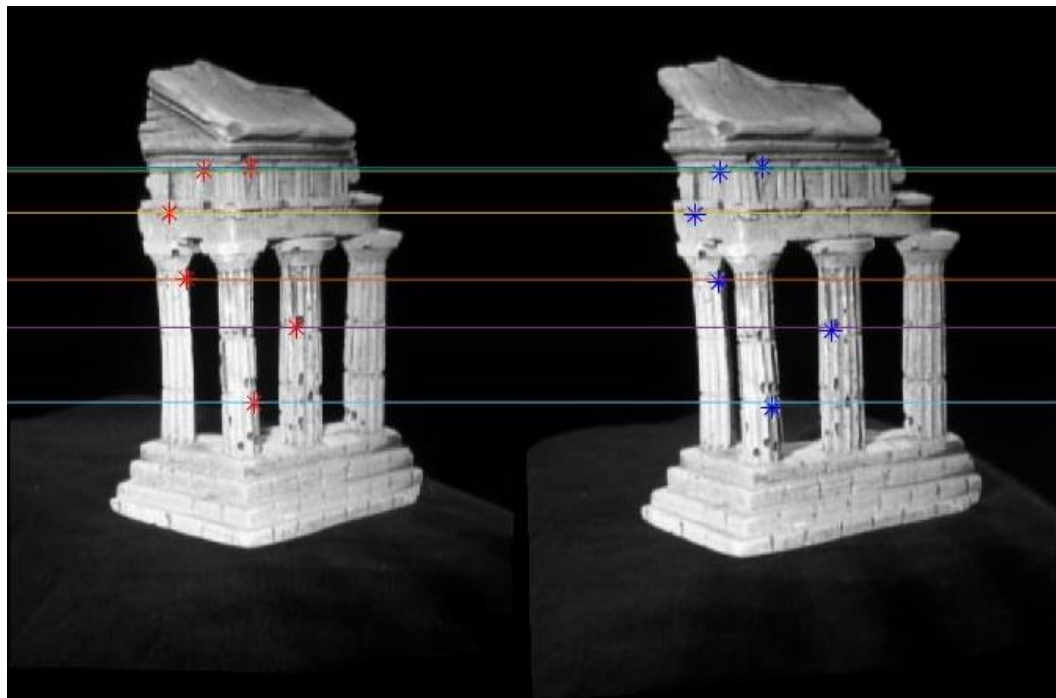


Figure 8: Rectify Image (Enlarged Image)

Dense window matching



Figure 9: Disparity Map

Depth map



Figure 11: Depth Map

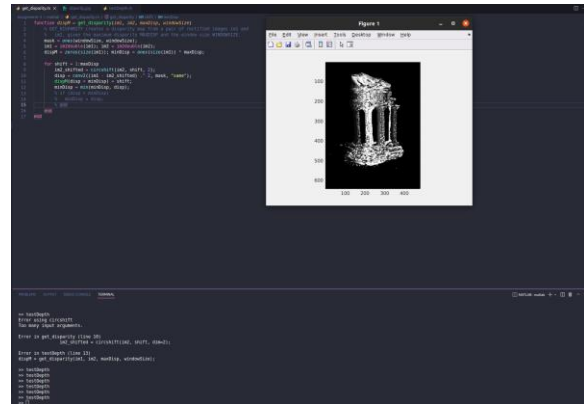


Figure 10: Screenshot of the whole screen

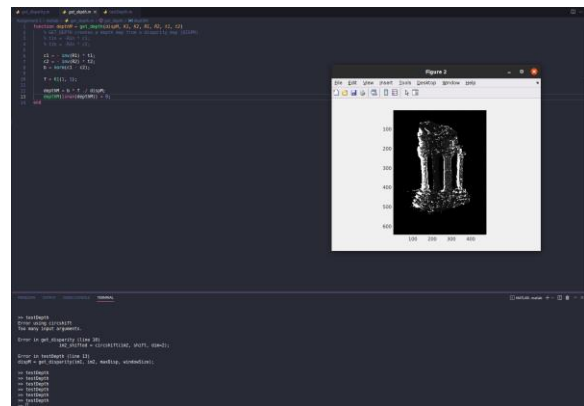


Figure 12: Screenshot of the whole screen

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

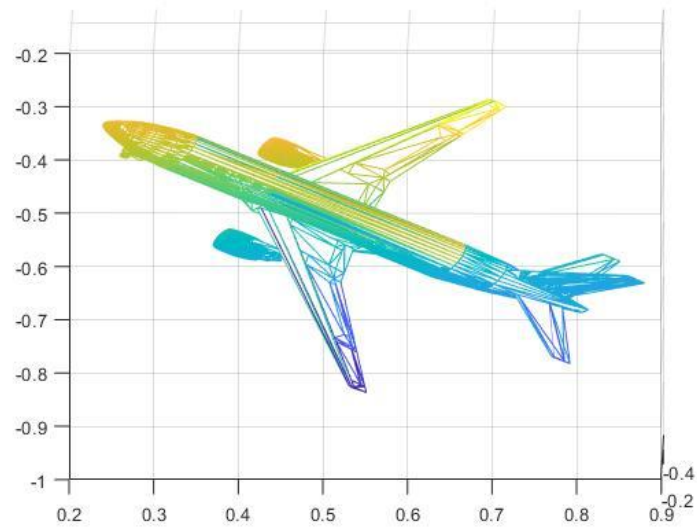


Figure 16: trimesh graph of the CAD model



Figure 17: CAD model overlayed on the airplane image