

! This Assignment uses 1 free late day

CMPT 762 Assignment 5

301474102 Sihui Wang

1. Sparse Reconstruction

1.1 Implementation of the eight-point algorithm

Eight-point algorithm is implemented in `eightpoint.m`.

The test code is implemented in another matlab file, `testF.m`. It computes the fundamental matrix between the image pair 'im1.png' and 'im2.png'. It also calls to run the code for epipolar line visualization in `displayEpipolarF.m`.

1.1.1 Recovered F

Here is the screenshot of the command window:

```
Command Window
>> testF
-0.0000    0.0000   -0.0000
 0.0000   -0.0000   -0.0021
-0.0000    0.0020    0.0083
```

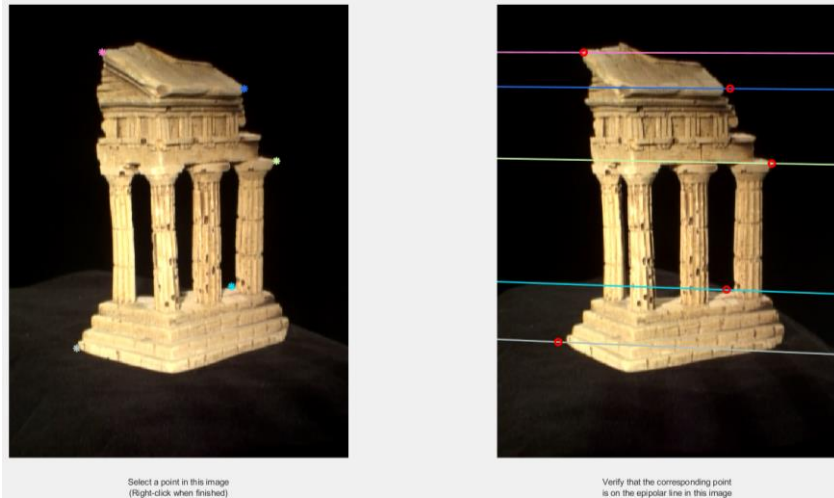
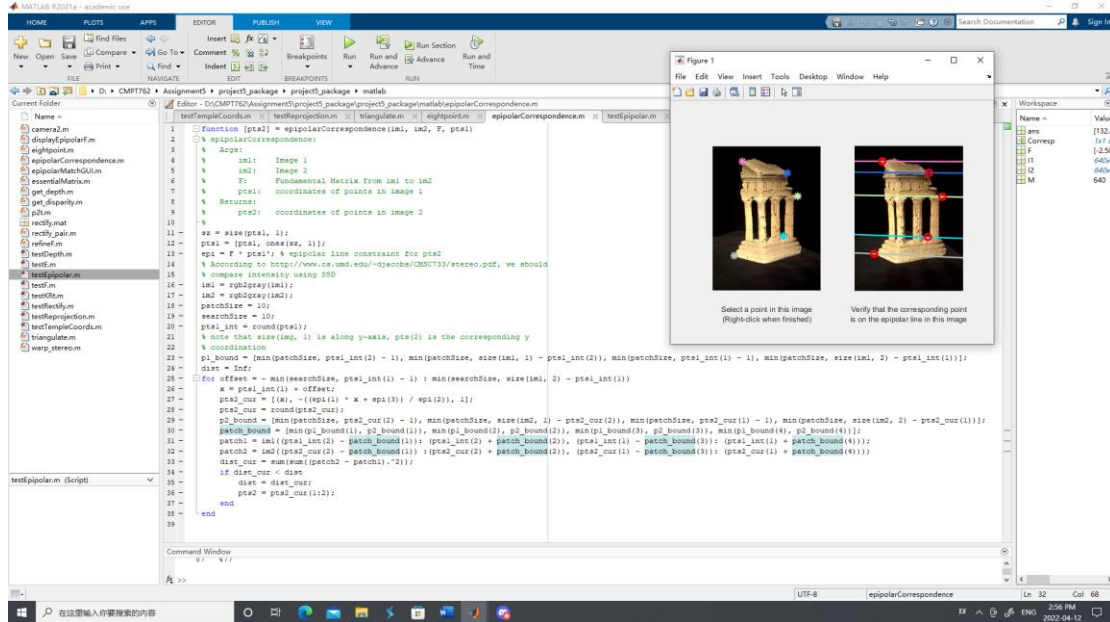
1.1.2 Visualization of Epipolar Lines:



1.2 Find Epipolar Correspondence

1.2.1 Implementation of Matching Points by Epipolar Correspondence

The matching algorithm is implemented in `epipolarCorrespondence.m`, and `testEpipolar.m` is the testing code. The following is the screenshot of the implementation with the results of the testing code.



1.2.2 Similarity Metric

According to <http://www.cs.umd.edu/~djacobsc/CMSC733/stereo.pdf>, page 11, we use the SSD distance between two patches of images as the similarity metric.

$$SSD = \sum_{i=-i_1}^{i_2} \sum_{j=-j_1}^{j_2} (I_1(x+i, y+j) - I_2(x'+i, y'+j))^2$$

Here I_1 and I_2 are the intensity values for the grayscale image 1 and image 2. (x, y) and (x', y') are the *candidate* matching points. $i \in [-i_1, i_2]$ and $j \in [-j_1, j_2]$ determine the patch size. By default, I set $i_1 = i_2 = j_1 = j_2 = 10$.

Sometimes we might need to put *additional constraints* on the bounds of i and j so that $x + i$ and $y + j$ are within the boundaries of both images, $[1, w] \times [1, h]$. To guarantee that, i_1 should satisfy:

$$\begin{aligned} i_1 &\leq 10 \\ x - i_1 &\geq 1 \\ x' - i_1 &\geq 1 \end{aligned}$$

So, we set i_1 as follows:

$$i_1 = \min \{10, x - 1, x' - 1\}$$

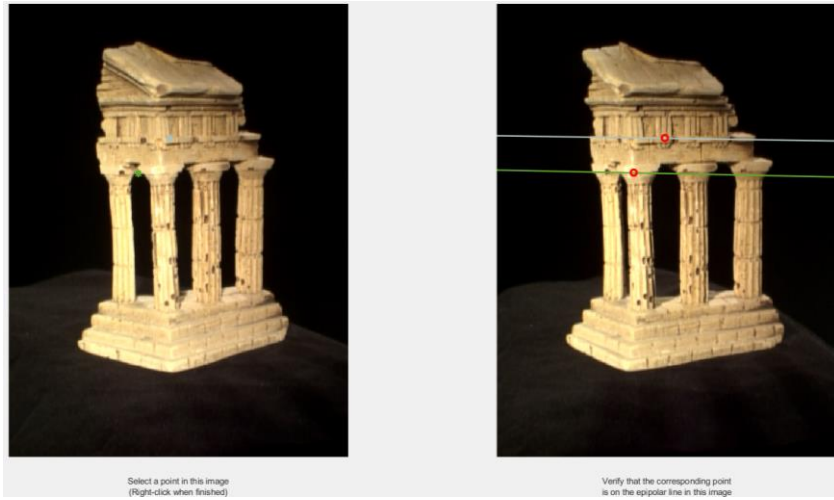
Similarly, we set i_2, j_1, j_2 as follows:

$$i_2 = \min \{10, w - x, w - x'\}$$

$$j_1 = \min \{10, y - 1, y' - 1\}$$

$$j_2 = \min \{10, h - y, h - y'\}$$

1.2.3 Analysis of Failing Cases



From the above images we can see that sometimes the method cannot accurately produce the matching pairs along the epipolar lines. Here are some potential reasons.

Repetitive textures and textureless areas: In the above image, the algorithm made a wrong matching for the gray point. The reason seems to be that the algorithm might have difficulties to identify what is the best matching if there are repetitive textures along the epipolar lines, or if the area is just textureless. This algorithm measures similarity by computing the distance of intensity around the points, which, to some degree, implicitly assumes that each point has a unique intensity ‘fingerprint’ in its neighborhood. However, images with repetitive textures might violate this assumption and lead to failure of this matching algorithm; In textureless areas, all points just look alike, which adds to the difficulties for the algorithm to recognize the matching point.

Intensity and Shape difference due to the Change of Viewpoint: In the above image, the algorithm made a wrong matching for the green point. One of the reasons for this failure is that the black area around the green point in the left image shrinks a lot in the right image, and the algorithm fails to recognize their resemblance. Generally speaking, the object’s intensity might change between the two views, which makes it difficult for the algorithm to detect the matching point by calculating the similarity of intensity. In addition, the object’s shape might change between the two views, meanwhile the algorithm always computes the similarity using the windows of the same size, which also might lead to inaccurate matching results.

1.3 Write a Function to Compute the Essential Matrix

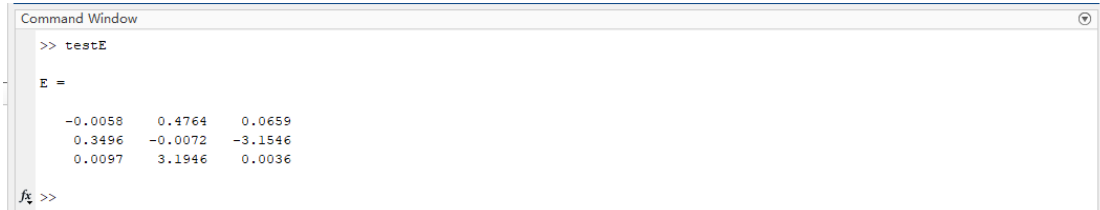
The essential matrix is obtained by the following formula:

$$E = K_2^T F K_1$$

The algorithm for essential matrix estimation is implemented in `essentialMatrix.m`.

The test code is implemented in another matlab file, `testE.m`. It computes the essential matrix between the image pair 'im1.png' and 'im2.png'.

The estimated essential matrix between 'im1.png' and 'im2.png' is:



```
Command Window
>> testE
E =
    -0.0058    0.4764    0.0659
    0.3496   -0.0072   -3.1546
    0.0097    3.1946    0.0036
fx >>
```

Note: Both essential matrices E and fundamental matrices F are of rank 2. E has 6 degrees of freedom, while F has 7 degrees of freedom.

This means that both E and F are *not uniquely decided*. For example, if we use different scale factors for normalization in the eight-point algorithm, we will obtain different results for F and E . However, this difference won't affect the visualization results.

1.4 Implement Triangulation

1.4.1 Deciding Extrinsic Matrix

First, use `camera2.m` to compute camera2's 4 potential extrinsic matrices: $E_{21}, E_{22}, E_{23}, E_{24}$.

Then, use camera2's intrinsic matrix K_2 to left-multiply $E_{2i} (i = 1, 2, 3, 4)$ to obtain 4 candidates of projection matrices, $P_{2i} = K_2 E_{2i} (i = 1, 2, 3, 4)$.

Use camera1's projection matrix, P_1 , camera2's each candidate projection matrix, $P_{2i} (i = 1, 2, 3, 4)$, and the matching pairs of points in the 2 images to triangulate and obtain the reconstructed 3D points.

Find out which candidate projection matrix has the largest number of reconstructed 3D points of positive depth. If there is a unique one candidate projection matrix P_{2i} who has the largest number of reconstructed 3D points of positive depth, then that candidate matrix P_{2i} will be chosen as the true projection matrix, and the corresponding extrinsic matrix E_{2i} should be the correct extrinsic matrix.

If there are more than one candidate projection matrices who equally have the largest number of reconstructed 3D points of positive depth, then we calculate the reprojection errors for each of these candidates.

Among all candidate projection matrices who equally have the largest number of reconstructed 3D points of positive depth, the candidate projection matrix P_{2i} who has the smallest reprojection error should be chosen as the true projection matrix, and the corresponding extrinsic matrix E_{2i} should be the correct extrinsic matrix.

1.4.2 Re-projection Error

Note: Changing normalization methods, or changing the parameters for normalization methods, will affect the re-projection error.

To get a lower re-projection error, in 1.4.2 I used a normalization method which is different from the one I used in 1.1.

The eight-point algorithm with different normalization techniques is implemented in

eightpoint2.m. The method is based on CMPT 732's tutorial: [Assignment 2 - 8 Point Algorithm + RANSAC - Google Docs](#).

Here is a brief introduction of the implementation in eightpoint2.m.

For `pts1` and `pts2`, I calculated their centroid (x, y) , then I calculated d , which is all points' mean distance to the centroid.

Then, I constructed a transformation matrix T :

$$T = \begin{pmatrix} \frac{\sqrt{2}}{d} & 0 & -x \cdot \frac{\sqrt{2}}{d} \\ 0 & \frac{\sqrt{2}}{d} & -y \cdot \frac{\sqrt{2}}{d} \\ 0 & 0 & 1 \end{pmatrix}$$

To normalize the points, I transform the homogeneous coordinates, `pts1` and `pts2`, by the following transformation:

```
pts1 = pts1 * T';  
pts2 = pts2 * T';
```

After obtaining the fundamental matrix F with the normalized points, I de-normalize F by the following transformation:

$$F = T' * F * T;$$

The test code is in `testReprojection.m`.

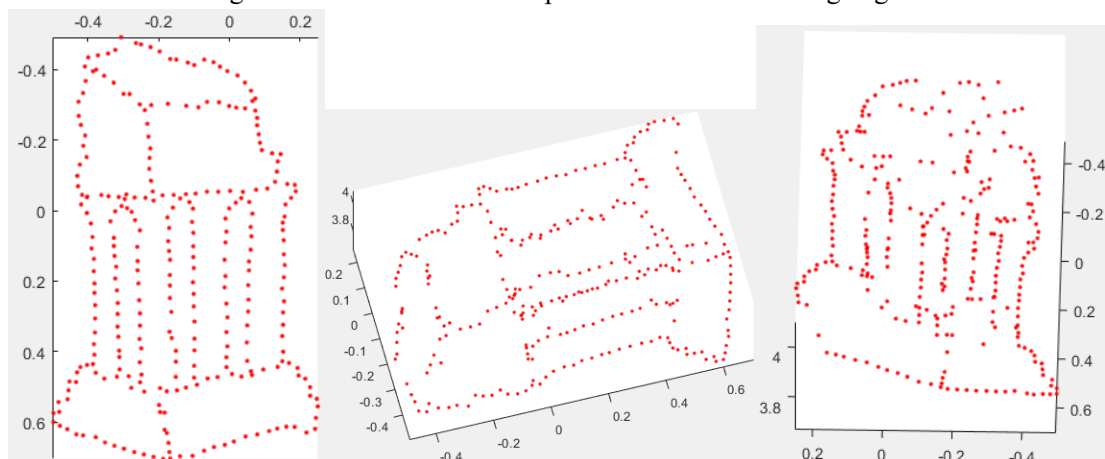
The re-projection error between `pts1` and the projection of the reconstructed 3D points is: **0.839748**;

The re-projection error between `pts2` and the projection of the reconstructed 3D points is: **0.826902**.

```
Command Window  
>> testReprojection  
Reprojection error for image 1: 0.839748  
Reprojection error for image 2: 0.826902  
fx >>
```

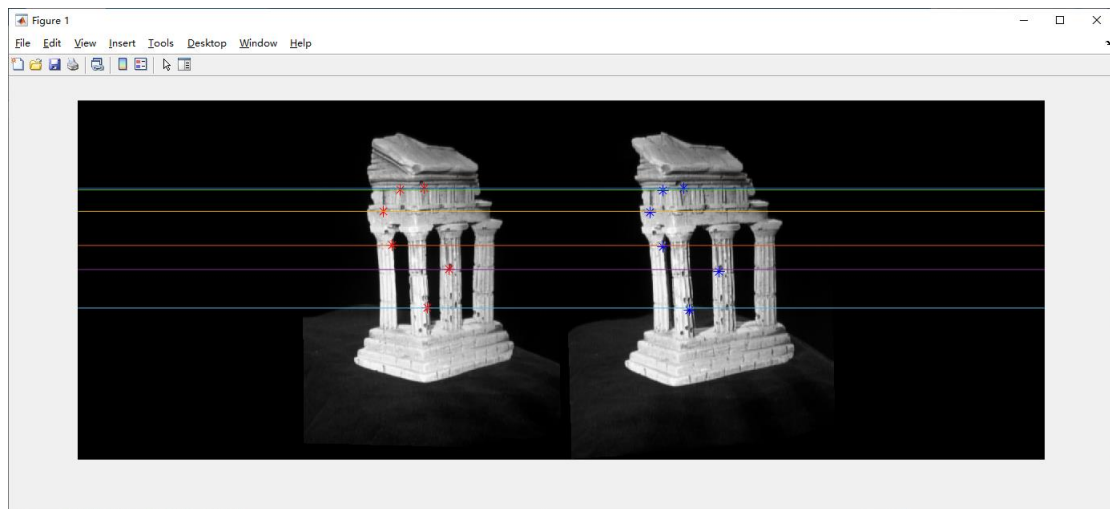
1.5 Write a Test Script that Uses templeCoords

Here are 3 images of the reconstructed temple from different viewing angles.

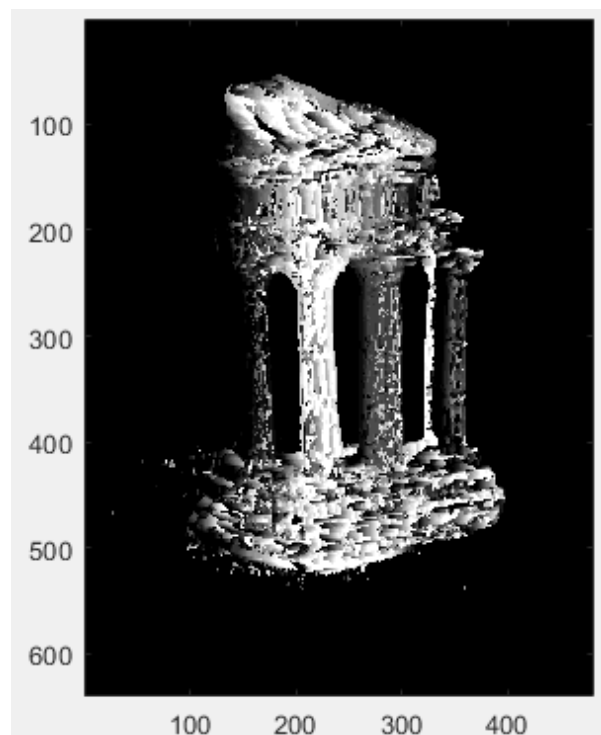


2. Dense Reconstruction

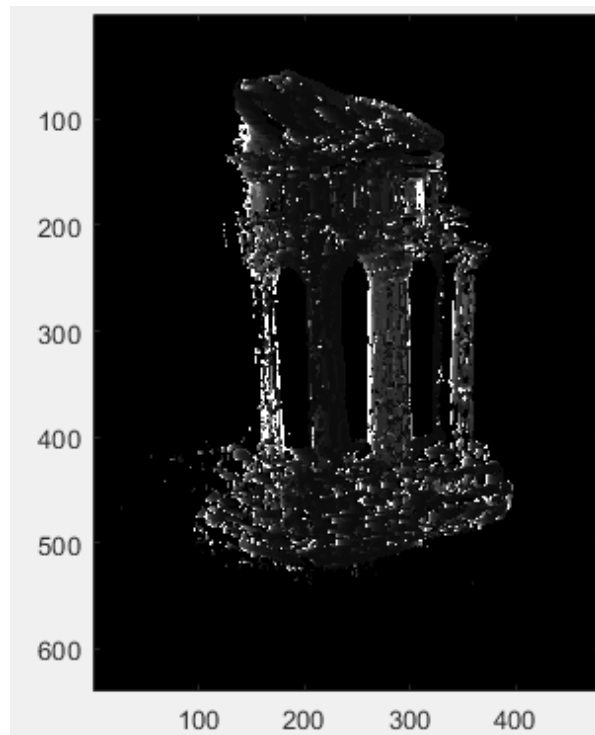
2.1 Image Rectification



2.2 Dense Window Matching to Find Per Pixel Density



2.3 Depth Map



3. Pose Estimation

3.1 Estimate Camera Matrix

```
Command Window
>> 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 1.6374
Pose Error with noisy 2D points is 0.3261
fx >>
```

3.2 Estimate Intrinsic/Extrinsic Parameters

```
Command Window
>> testERt
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 0.8525
Rotation Error with clean 2D points is 0.1619
Translation Error with clean 2D points is 0.2764
fx >>
```

3.3 Project a CAD Model to the Image

