浙江大學

Studies and Discussions of Special Subjects
Project 3

3D Reconstruction

Zi Jun, Xu
Computer Science 1603, 3160100056
E-mail: 3540166188@qq.com

May 26, 2019

Contents

Chapter 1

Description

One of the major areas of computer vision is 3D reconstruction.
3D reconstruction is the process of capturing the shape and appearance of real objects. The research of 3D reconstruction has always been a difficult goal. Using 3D reconstruction one can determine any object's 3D profile, as well as knowing the 3D coordinate of any point on the profile.
In this project, there are two programming parts: sparse reconstruction and dense reconstruction.
Sparse reconstructions generally contain a number of points, but still manage to describe the objects in question. Dense reconstructions are detailed and fine-grained. In fields like 3D modeling and graphics, extremely accurate dense reconstructions are invaluable when generating 3D models of real world objects and scenes.

Chapter 2

Analysis and comments

2.1 Sparse reconstruction

In this section, a set of function to compute the sparse reconstruction from two sample images of a temple are implemented.
First, estimate the Fundamental matrix, compute point correspondences, then plot the results in 3D.

2.1.1 Implement the eight-point algorithm

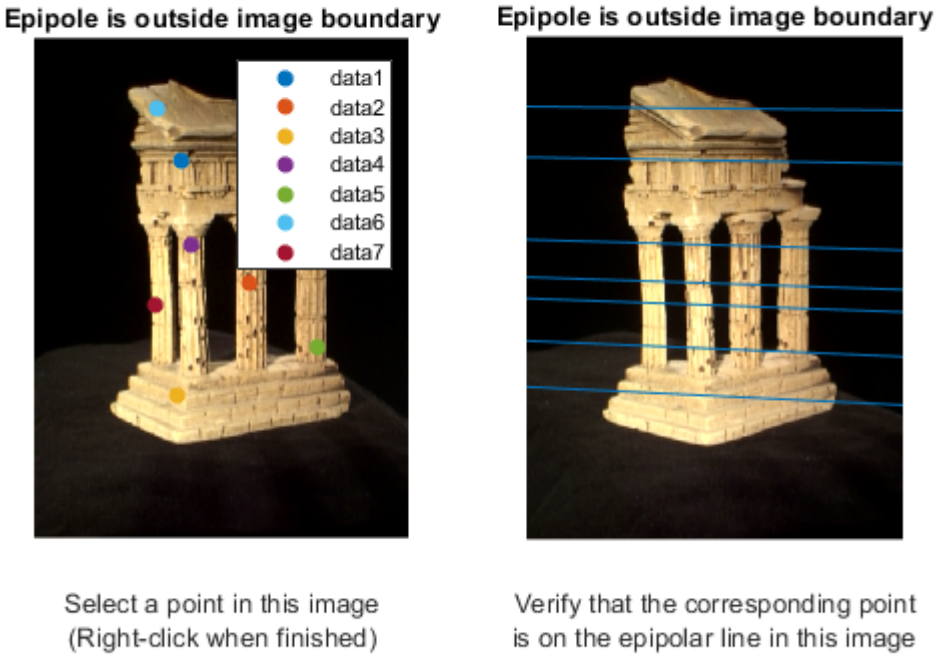
In this part, the eight-point algorithm is implemented to estimate the fundamental matrix.
In implementation, the argument M is removed because the scale is not needed.
Steps:

1. Normalize the two corresponding pairs of point.
2. Use Direct Linear Transformation (DLT).
3. Use Singular Value Decomposition (SVD).

The fundamental matrix and a sample of epipolar lines are shown below and on 2.1.

$$F = \begin{bmatrix} -4.34501133204072e-07 & 2.31522387703940e-05 & 0.000158692389526582 \\ 1.47066904226565e-05 & 5.37276618278489e-07 & -0.223773653509504 \\ -0.00399741961425564 & 0.214825414035401 & 0.951877700008338 \end{bmatrix} \tag{2.1}$$

Figure 2.1: A sample of epipolar lines.



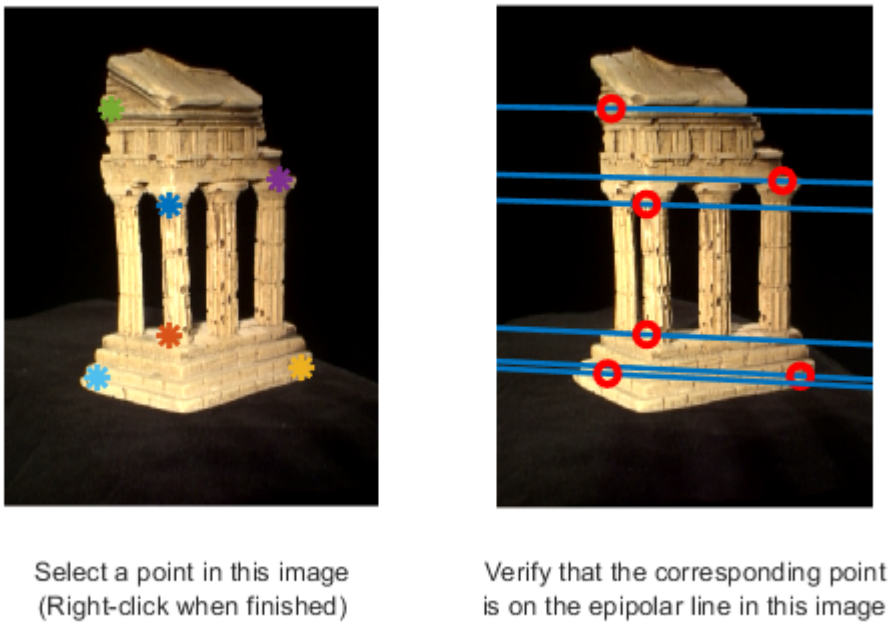
2.1.2 Find epipolar correspondences

In this part, a function to find corresponding pairs of points is implemented. Steps:

1. Find the boundary points one each epipolar line.
2. Use Bresenham's line algorithm to find the points on each epipolar line.
3. Remove the boundary pairs of points from fixed kernel size.
4. Use Euclidean error function to find the minimal error and take the pairs of points as correspondences.

A sample of epipolar correspondences is shown on 2.2.

Figure 2.2: Epipolar correspondences.



2.1.3 Compute the essential matrix

Simply, from the formula, the essential matrix is computed from transpose of camera matrix 1 times fundamental matrix times camera matrix 2. The essential matrix is shown below:

$$E = \begin{bmatrix} -1.00439984105285 & 53.7126929328672 & 8.73154650299580 \\ 34.1192035277824 & 1.25097919489515 & -334.469481708257 \\ -0.757367999883989 & 338.684852110018 & 0.400887172110174 \end{bmatrix} \quad (2.2)$$

2.1.4 Implement triangulation

Estimate the 3D positions of points from 2D correspondence and compute the reprojection error.
In this project, the way to determine the correctness of correspondences in the four possibilities is to check if all of the points are in front of two cameras (positive depth). The projection matrix 2, rotation matrix 2, and transition 2 are shown below:

$$P2 = \begin{bmatrix} 1391.85671613790 & -36.5982809138707 & 681.475597491190 & -1472.42554129443 \\ -28.2826639025924 & 1525.25965669119 & 249.196062354189 & -0.369174228672691 \\ -0.256102567844053 & -0.000825074203291159 & 0.966649261105724 & 0.158687677644781 \end{bmatrix} \quad (2.3)$$

$$R2 = \begin{bmatrix} 0.966378350729093 & -0.0239074220473110 & 0.256010387314988 \\ 0.0228988642906277 & 0.999713836266964 & 0.00692008602465366 \\ -0.256102567844053 & -0.000825074203291159 & 0.966649261105724 \end{bmatrix} \tag{2.4}$$

$$t2 = \begin{bmatrix} -1.00761210707263 \\ -0.00213155151786689 \\ -0.102794398189249 \end{bmatrix} \tag{2.5}$$

2.1.5 Plot the correspondences

There are three images of different angles of the final reconstruction shown below:

Figure 2.3: Final reconstruction (Angle 1).

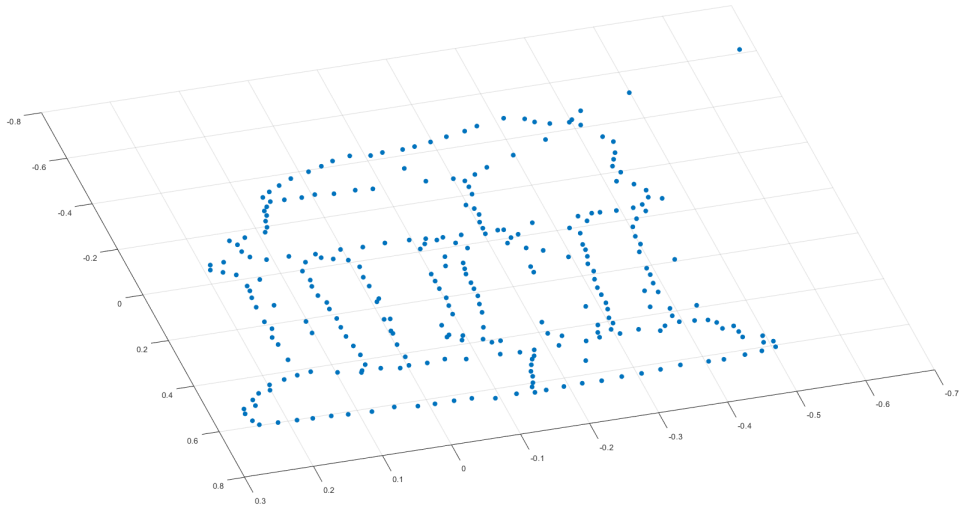


Figure 2.4: Final reconstruction (Angle 2).

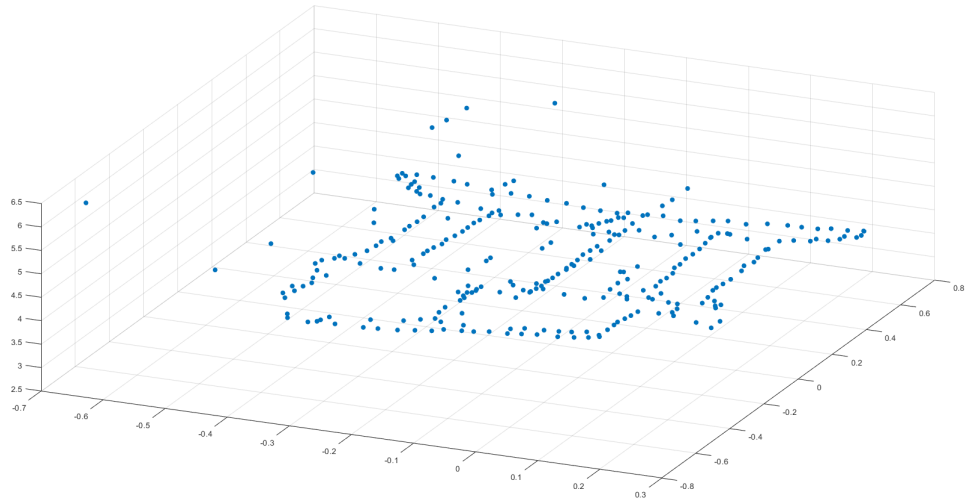
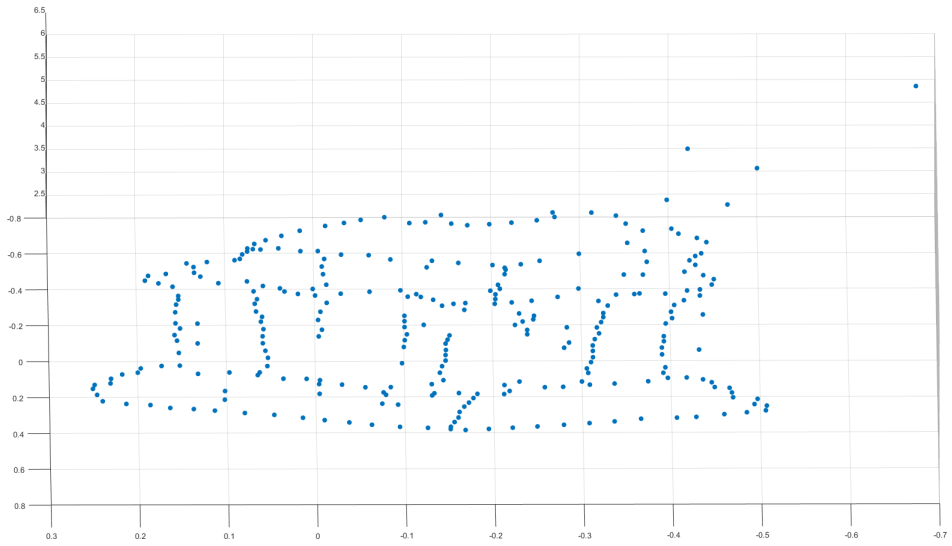


Figure 2.5: Final reconstruction (Angle 3).



2.2 Pose estimation

In this section, a set of functions to estimate both the intrinsic and extrinsic parameters of camera given 2D point x on image and their corresponding 3D points X are implemented.

2.2.1 Estimate camera matrix

Computes the pose matrix (camera matrix) from given 2D and 3D points.

Figure 2.6: testPose result.

```
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 2.3856
Pose Error with noisy 2D points is 0.1726
```

2.2.2 Estimate intrinsic/extrinsic parameters

Computes the intrinsic K, rotation R and translation t.

Figure 2.7: testKRt result.

```
Intrinsic Error with clean 2D points is 2.0000
Rotation Error with clean 2D points is 2.0000
Translation Error with clean 2D points is 0.1026
------------------------------
Intrinsic Error with clean 2D points is 2.0238
Rotation Error with clean 2D points is 2.0000
Translation Error with clean 2D points is 0.3238
```

2.2.3 Project a CAD model to the image

Figure 2.8: Plot 2D and projected 3D points.

Figure 2.9: CAD model.

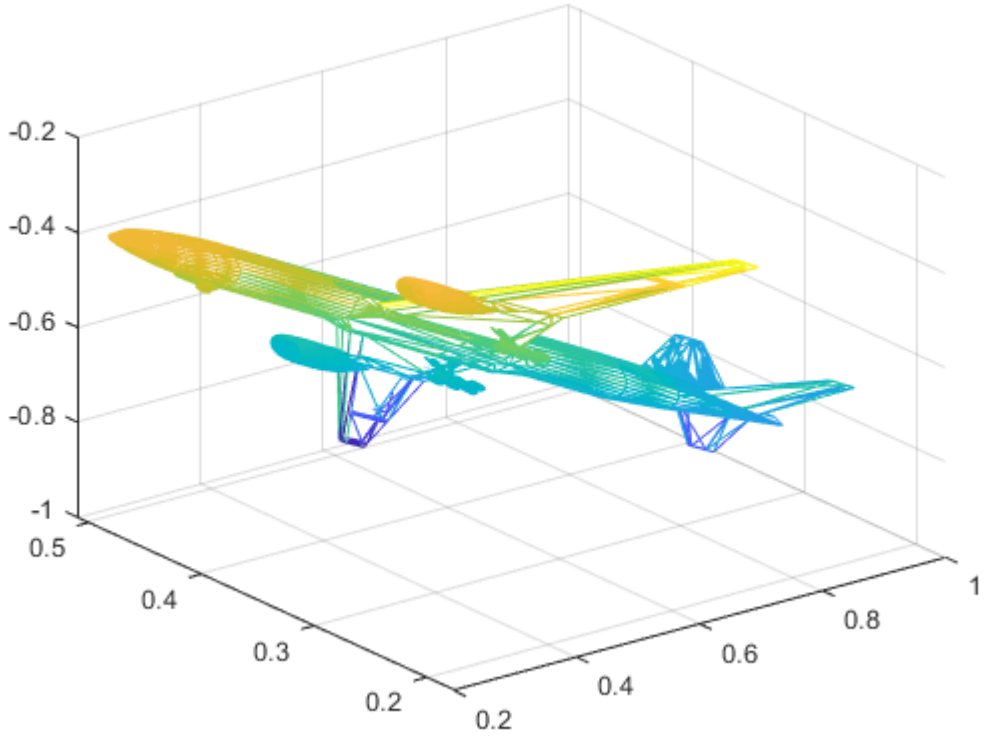


Figure 2.10: Draw the projected CAD model overlapping with the 2D image.

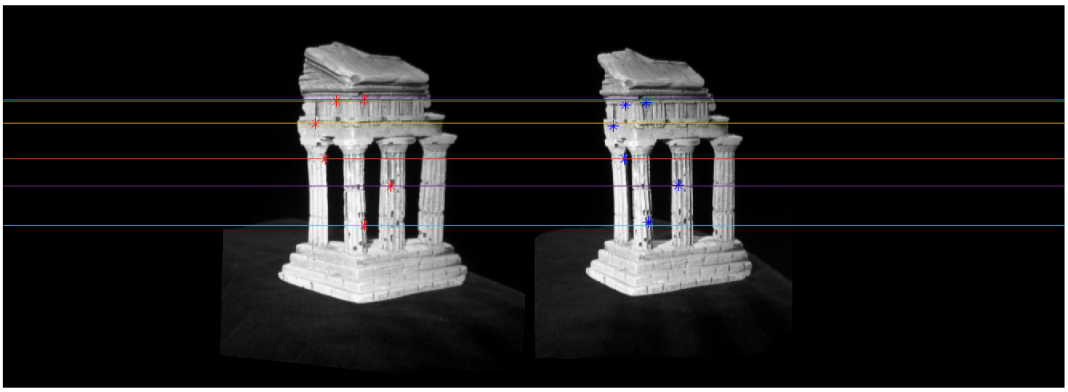
2.3 Dense reconstruction

In this section, a set of functions to perform a dense reconstruction on the temple examples are implemented.

2.3.1 Image rectification

Takes left and right camera parameters (intrinsic K, rotation R, translation T) and returns left and right rectification matrices and updated camera parameters. The result is shown on 2.11.

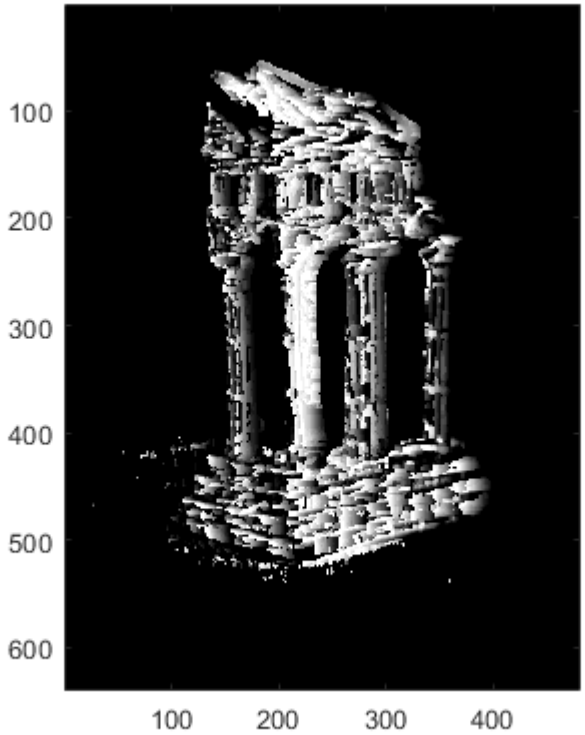
Figure 2.11: testRectify result.



2.3.2 Dense window matching to find per pixel density

Creates a disparity map from a pair of rectified images with given the maximum disparity and the window size.
The result is shown on 2.12.

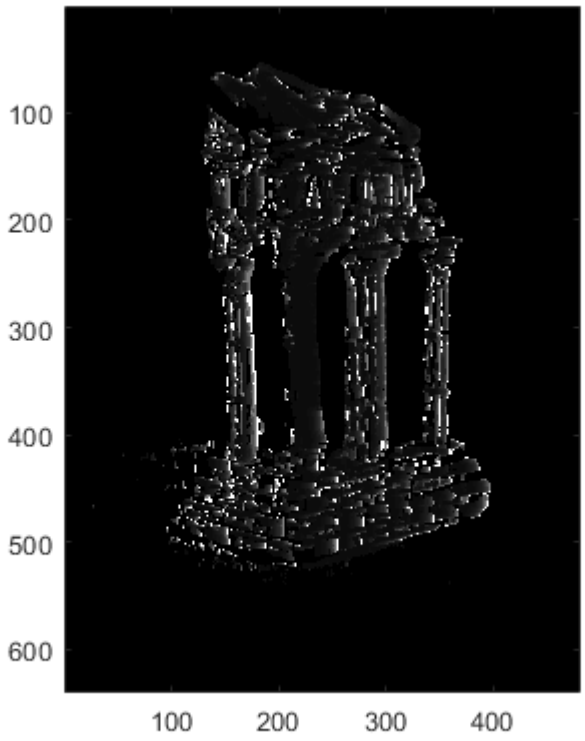
Figure 2.12: Disparity map.



2.3.3 Depth map

Creates a depth map from a disparity map (DISPM).
The result is shown on 2.13.

Figure 2.13: Depth map.



Chapter 3

Conclusion

There are some errors in epipolar correspondences. At first, I use Gaussian filter to blur the first image because I thought that there must be some noise, then compute Euclidean distance between the first and second images. However, I found that the larger size of kernel I used, the higher the error was, and compute the Euclidean without any blur gave the best result, but the error still can be seen clearly.
During the project, I found that I was still not proficient in matrix computation. It took me a long time to think about how to create a needed matrix. Actually, This project took me longer time than last 2 projects.
I had to totally understand the formula, and then started coding, but still encountered tons of problems and they did frustrate me. I'm glad that I could persist in learning new things, and eventually completed this project.