EECS 442 Computer Vision: Homework 6

Instructions

- This homework is **due at 11:59:59 p.m. on Wednesday, December 11th, 2019**.
- The submission includes two parts:
  1. **To Gradescope:** a `pdf` file as your write-up, including your answers to all the questions and key choices you made for solving problems. You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: https://combinepdf.com/. **Please mark where each question is on gradescope.**
  2. **To Canvas:** a `zip` file including all your code.
- The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.

Python Environment We are using Python 3.7 for this course. You can find references for the Python standard library here: https://docs.python.org/3.7/library/index.html. We will make use of the following packages extensively in this course:

- Numpy (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html).
- OpenCV (https://opencv.org/). Especially, we're using OpenCV 3.4 in this homework. To install it, run `conda install -c menpo opencv`.
- Open3D (http://www.open3d.org/). We're using the latest Open3D to process point cloud. To install it, run `conda install -c open3d-admin open3d` or `pip install open3d-python` instead. Alternatively, you could not use our provided visualization code, and visualize the point cloud using matplotlib's 3D scatterplot.

1 Camera Calibration [20 pts]

The goal is to compute the projection matrix P that goes from world 3D coordinates to 2D image coordinates. Recall that using homogeneous coordinates the equation for moving from 3D world to 2D camera coordinates is:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \equiv P \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

In part 1, you're given corresponding point locations in `pts2d-norm-pic.txt` and `pts3d-norm.txt`, which corresponds to a camera projection matrix. **Solve** the projection matrix P and **include** it in your report.

2 Estimation of the Fundamental Matrix [40 pts]

The next part of this project is estimating the mapping of points in one image to lines in another by means of the fundamental matrix. This will require you to use similar methods to those in part 1. You'll work on the Wizarding Temple dataset, which is shown in Figure 1.

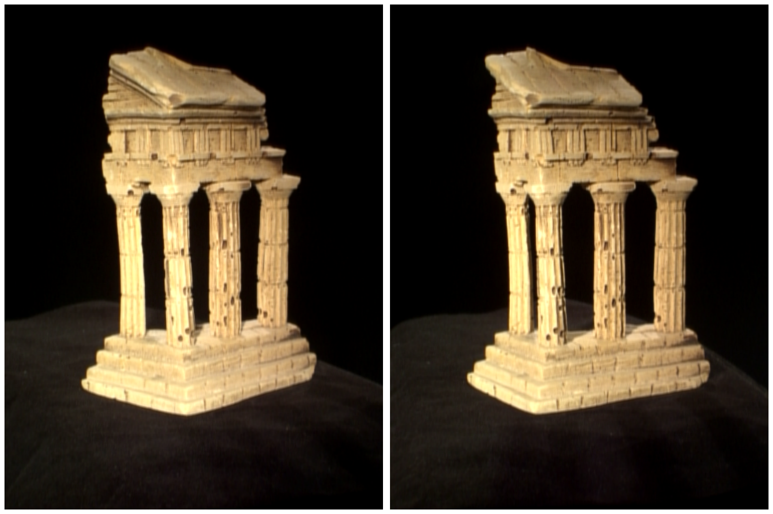

Figure 1: Wizarding Temple Dataset.

Recall that the definition of the Fundamental Matrix is:

$$\begin{pmatrix} u' & v' & 1 \end{pmatrix} \begin{pmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = 0$$

Note: the fundamental matrix is sometimes defined as the transpose of the above matrix with the left and right image points swapped. Both are valid fundamental matrices, but the visualization functions in the starter code assume you use the above form.

And another way of writing this matrix equations is:

$$\begin{pmatrix} u' & v' & 1 \end{pmatrix} \begin{pmatrix} f_{11}u + f_{12}v + f_{13} \\ f_{21}u + f_{22}v + f_{23} \\ f_{31}u + f_{32}v + f_{33} \end{pmatrix} = 0$$

Which is the same as:

$$f_{11}uu' + f_{12}vu' + f_{13}u' + f_{21}uv' + f_{22}vv' + f_{23}v' + f_{31}u + f_{32}v + f_{33} = 0$$

Given corresponding points you get one equation per point pair. Therefore, you can solve this with 8 or more points by constructing a system

$$Af = 0$$

where

$$f = [f_{11}, f_{12}, f_{13}, f_{21}, f_{22}, f_{23}, f_{31}, f_{32}, f_{33}]^T$$

and you can use SVD to solve it.

Here are detailed instructions:

1. Load corresponding points from `temple.npz`.
2. Implement eight-point algorithm and estimate the fundamental matrix F. **Report** F in your report. Remember to normalize F so that the last entry of F is 1. *Hint: You should normalize the data first. For example, scale the data by dividing each coordinate by the maximum of the images width and height.* You may validate your implementation by comparing against the output of `cv2.findFundamentalMat`, but **you will be graded on your eight-point algorithm.**
3. Show epipolar lines. **Include** the visualization in your report. A sample output is shown as Figure 2. You can call `draw_epipolar(img1, img2, F, pts1, pts2)` in `utils.py` to generate an image like the sample output. Note that you only need to show around 10 points and their corresponding epipolar lines so that we can verify your calculation is correct.

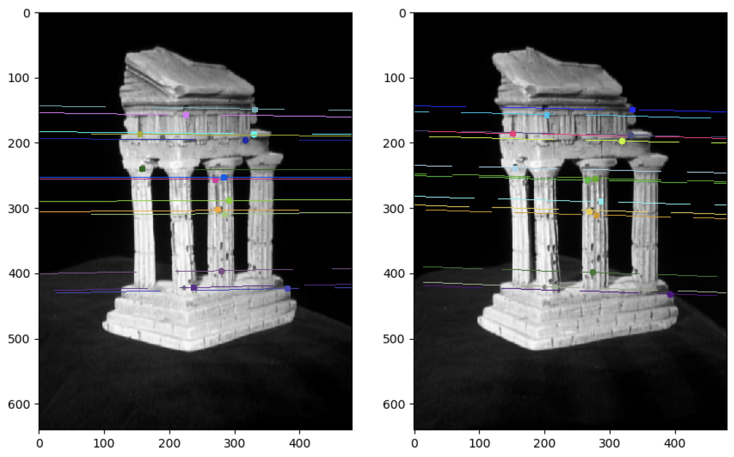

Figure 2: Epipolar lines of Wizarding Temple Dataset.

3 Triangulation [40 pts]

The next step is extracting 3D points from 2D points and camera matrices, which is called triangulation. Let $X = (X_1, X_2, X_3, 1)^T$ be a point in 3D. For two cameras, we have

$$x_1 = P_1 X$$

$$x_2 = P_2 X$$

Triangulation is to solve X given x_1, x_2, P_1, P_2. We'll use Direct Linear Transform (DLT) to perform triangulation, which has already been implemented in OpenCV.

Here are the instructions:

1. Load camera intrinsic matrix K_1 and K_2 from `temple.npz`.

2. Extract the essential matrix E given the fundamental matrix F and intrinsic matrices K_1 and K_2. **Report** E. Recall that

$$F = K_2^{-T} E K_1^{-1}$$

For this question, you may use `cv2.findFundamentalMat` to obtain the Fundamental matrix.

3. Decompose the essential matrix E and get the rotation matrix R and translation t. You can use `cv2.decomposeEssentialMat`. Hint: There are four possible combinations of R and t. The correct configuration is the one for which most of the 3D points are in front of both cameras (positive depth).

4. Determine the camera projection matrices P_1 and P_2 according to the intrinsic and extrinsic matrix $[R|t]$, K_1 and K_2. **Report** P_1 and P_2. You can set

$$P_1 = K_1[I \quad 0]$$

$$P_2 = K_2[R \quad t]$$

5. Triangulate 2D pairs of points to 3D. You can use `cv2.triangulatePoints`.

6. Visualize the point cloud. **Include** the visualization in your report from at least 3 views (so that we can reconstruct it!). A sample output is shown as Figure 3. You may use our provided visualization function, `visualize_pcd` found in `utils.py`, or you can implement your own visualization. One good alternative is matplotlib's 3D scatterplot.

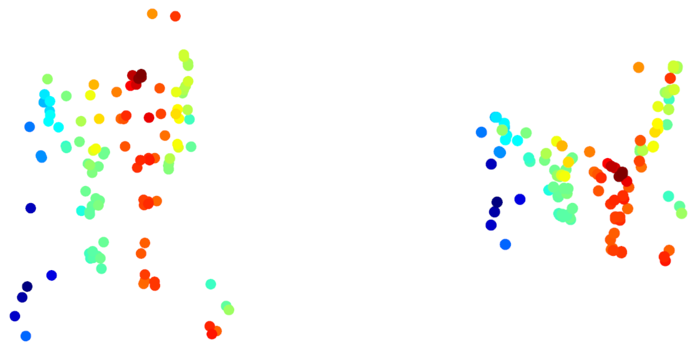

Figure 3: sample output

References

- Temple dataset. http://vision.middlebury.edu/mview/data/.

Acknowledgement

Part of the homework are taken from Georgia Tech CS 6476 by James Hays and CMU 16-385. Please feel free to similarly re-use our problems while similarly crediting us.