16-720A Computer Vision: Homework 4

3D Reconstruction

Instructors: Srinivasa Narasimhan & David Held
TAs: Chen-Hsuan Lin, Brian Okorn, Yifan Xing, Sree Harsha Kalli,
Siddarth Malreddy, Prakhar Pradeep Naval, Khushi Gupta, Shangxuan Wu,
Bala Siva Jujjavarapu, Jingyan Wang, Yashasvi Agrawal

Due: November 14, 2017, 11:59 PM

Make sure to start early!

Part I

Theory

Before implementing our own 3D reconstruction, let's take a look at some simple theory questions that may arise. The answers to the below questions should be relatively short, consisting of a few lines of math and text (maybe a diagram if it helps your understanding).

Q1.1 **(5 points)** Suppose two cameras fixate on a point P (see Figure 1) in space such that their principal axes intersect at that point. Show that if the image coordinates are normalized so that the coordinate origin $(0, 0)$ coincides with the principal point, the $\mathbf{F}_{33}$ element of the fundamental matrix is zero.

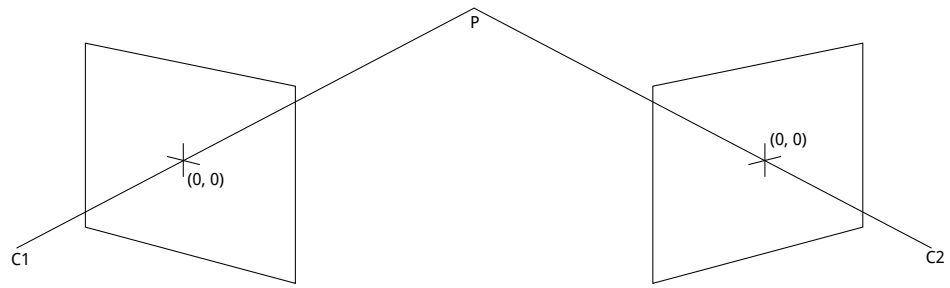

Figure 1: Figure for Q1.1. $C1$ and $C2$ are the optical centers. The principal axes intersect at point P.

Q1.2 **(5 points)** Consider the case of two cameras viewing an object such that the second camera differs from the first by a *pure translation* that is parallel to the x-axis. Show that the epipolar lines in the two cameras are also parallel to the x-axis. Backup your argument with relevant equations.

Q1.3 **(5 points)** Suppose we have an inertial sensor which gives us the accurate positions (R_i and t_i, where R is the rotation matrix and t is corresponding translation vector) of the robot at time i. What will be the effective rotation (R_{rel}) and translation (t_{rel}) between two frames at different time stamps? Suppose the camera intrinsics (K) are known, express the essential matrix (E) and the fundamental matrix (F) in terms of K, R_{rel} and t_{rel}.

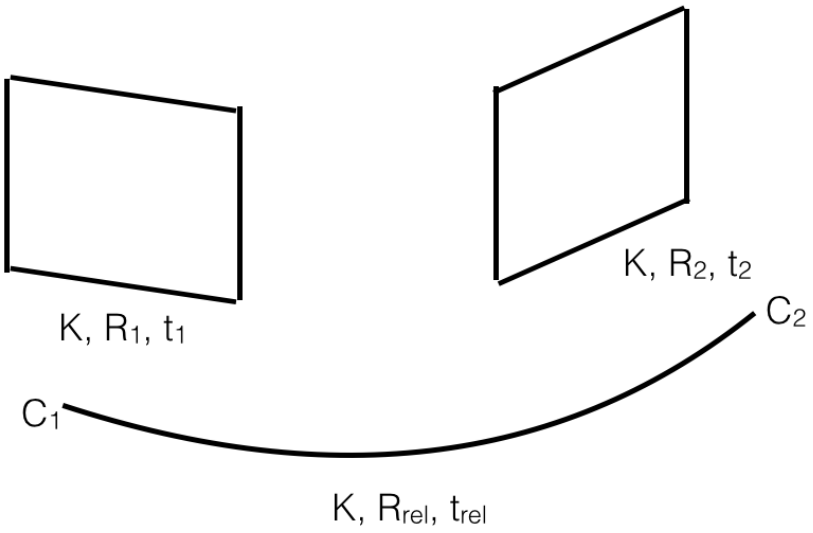

Figure 2: Figure for Q1.3. $C1$ and $C2$ are the optical centers. The rotation and the translation is obtained using inertial sensors. R_{rel} and t_{rel} are the relative rotation and translation between two frames.

Q1.4 **(10 points)** Suppose that a camera views an object and its reflection in a plane mirror. Show that this situation is equivalent to having two images of the object which are related by a skew-symmetric fundamental matrix. You may assume that the object is flat, meaning that all points on the object are of equal distance to the mirror (*Hint:* draw the relevant vectors to understand the relationships between the camera, the object and its reflected image.)

Part II
Practice

1 Overview

In this part you will begin by implementing the two different methods seen in class to estimate the fundamental matrix from corresponding points in two images (Section 2). Then, given the fundamental matrix and calibrated intrinsics (which will be provided) you will compute the essential matrix and use this to compute a 3D metric reconstruction from 2D correspondences using triangulation (Section 3). Finally, you will implement a method to automatically match points taking advantage of epipolar constraints and make a 3D visualization of the results (Section 4).

2 Fundamental matrix estimation

In this section you will explore different methods of estimating the fundamental matrix given a pair of images. In the `data/` directory, you will find two images (see Figure 3) from the Middlebury multi-

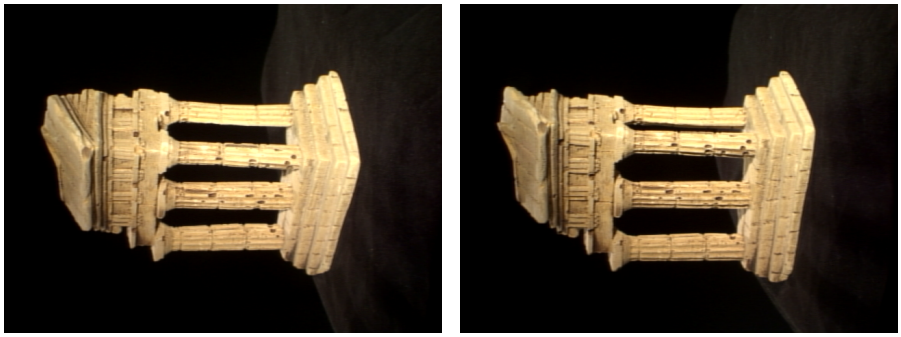

Figure 3: `Temple` images for this assignment

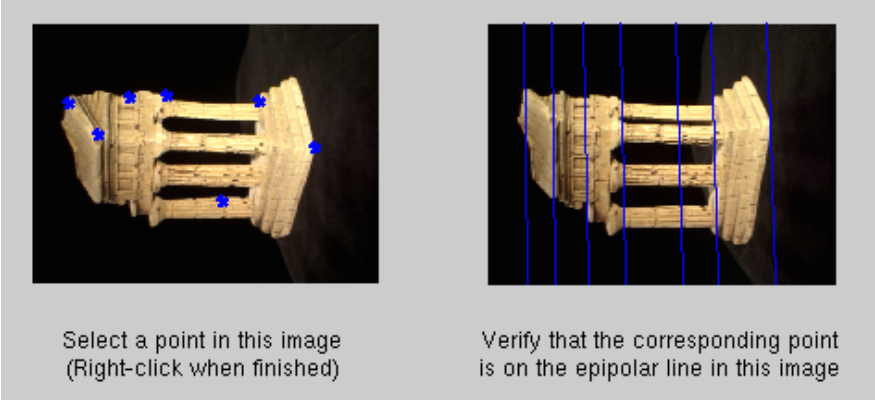

Figure 4: `displayEpipolarF.m` creates a GUI for visualizing epipolar lines

view dataset[1], which is used to evaluate the performance of modern 3D reconstruction algorithms.

The Eight Point Algorithm

The 8-point algorithm (discussed in class, and outlined in Section 10.1 of Forsyth & Ponce) is arguably the simplest method for estimating the fundamental matrix. For this section, you may manually select point correspondences in an image pair using `cpselect`, or use provided correspondences you can find in `data/some_corresp.mat`.

Q2.1 **(10 points)** Submit a function with the following signature for this portion of the assignment:

```
function F = eightpoint(pts1, pts2, M);
```

where `pts1` and `pts2` are $N \times 2$ matrices corresponding to the (x, y) coordinates of the N points in the first and second image repectively (the format returned by `cpselect`). `M` is a scale parameter.

- You should scale the data as was discussed in class, by dividing each coordinate by M (the maximum of the image's width and height). After computing $\mathbf{F}$, you will have to "unscale" the fundamental matrix.

[1] `http://vision.middlebury.edu/mview/data/`

Hint: If $x_{normalized} = Tx$, then $F_{unnormalized} = T^T \mathbf{F} T$.
You must enforce the singularity condition of the **F** before unscaling.

- You may find it helpful to refine the solution by using local minimization. This probably won't fix a completely broken solution, but may make a good solution better by locally minimizing a geometric cost function.
  For this we have provided `refineF` (takes in F and the two sets of points), which you can call from `eightpoint` before unscaling `F`.

- Remember that the x-coordinate of a point in the image is its column entry, and y-coordinate is the row entry. Also note that eight-point is just a figurative name, it just means that you need at least 8 points; your algorithm should use an over-determined system ($N > 8$ points).

- To visualize the correctness of your estimated **F**, use the supplied function in `displayEpipolarF.m` (takes in F, and the two images). This GUI lets you select a point in one of the images and visualize the corresponding epipolar line in the other image (Figure 4).

- **Output:** Save your matrix `F`, scale `M`, 2D points `pts1` and `pts2` to the file `q2_1.mat`.
  In your write up: Write your recovered **F** and include an image of some example output of `displayEpipolarF`.
  Hint: In LaTeX, the `verbatim` environment may be useful for copy-pasting the matrix displayed in Matlab

The Seven Point Algorithm

Q2.2 **(15 points)** Since the fundamental matrix only has seven degrees of freedom, it is possible to calculate **F** using only seven point correspondences. This requires solving a polynomial equation. In the section, you will implement the seven-point algorithm (described in class, and outlined in Section 15.6 of Forsyth and Ponce). Use `cpselect` to manually select 7 points, and use these points recover a fundamental matrix **F**. The function should have the signature:

```
function F = sevenpoint(pts1, pts2, M)
```

where `pts1` and `pts2` are 7×2 matrices containing the correspondences and `M` is the normalizer (use the maximum of the images' height and width), and `F` is a cell array of length either 1 or 3 containing Fundamental matrix/matrices. Use `M` to normalize the point values between $[0, 1]$ and remember to "unnormalize" your computed **F** afterwards.

- Use `displayEpipolarF` to visualize **F** and pick the correct one.

- **Output:** Save your matrix `F`, scale `M`, 2D points `pts1` and `pts2` to the file `q2_2.mat`.
  In your write up: Write your recovered **F** and print an output of `displayEpipolarF`. Also, include an image of some example output of `displayEpipolarF` using the seven point algorithm.

- *Hints:* You can use Matlab's `roots()` and `conv()` The epipolar lines may not match exactly due to imperfectly selected correspondences, and the algorithm is sensitive to small changes in the point correspondences. You may want to try with different sets of matches.

3 Metric Reconstruction

You will compute the camera matrices and triangulate the 2D points to obtain the 3D scene structure. To obtain the Euclidean scene structure, first convert the fundamental matrix $\mathbf{F}$ to an essential matrix $\mathbf{E}$. Examine the lecture notes and the textbook to find out how to do this when the internal camera calibration matrices $\mathbf{K}_1$ and $\mathbf{K}_2$ are known; these are provided in `data/intrinsics.mat`.

Q3.1 **(5 points)** Write a function to compute the essential matrix $\mathbf{E}$ given $\mathbf{F}$ and $\mathbf{K}_1$ and $\mathbf{K}_2$ with the signature:

```
function E = essentialMatrix(F, K1, K2)
```

In your write up: Write your estimated $\mathbf{E}$ using $\mathbf{F}$ from the eight-point algorithm.

Given an essential matrix, it is possible to retrieve the projective camera matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ from it. Assuming $\mathbf{M}_1$ is fixed at $[\mathbf{I}, 0]$, $\mathbf{M}_2$ can be retrieved up to a scale and four-fold rotation ambiguity. For details on recovering $\mathbf{M}_2$, see section 7.2 in Szeliski. We have provided you with the function in `matlab/camera2.m` to recover the four possible $\mathbf{M}_2$ matrices given $\mathbf{E}$ and $\mathbf{K}_2$.

Note: The `M1` and `M2` here are projection matrix of the form: $M_1 = \big[I|0\big]$ and $M_2 = \big[R|t\big]$.

Q3.2 **(10 points)** Using the above, write a function to triangulate a set of 2D coordinates in the image to a set of 3D points with the signature:

```
function [P, err] = triangulate(C1, p1, C2, p2)
```

where `p1` and `p2` are the $N \times 2$ matrices with the 2D image coordinates and `P` is an $N \times 3$ matrix with the corresponding 3D points per row. `C1` and `C2` are the 3×4 camera matrices. Remember that you will need to multiply the given intrinsics matrices with your solution for the canonical camera matrices to obtain the final camera matrices. Various methods exist for triangulation - probably the most familiar for you is based on least squares (see Szeliski Chapter 7 if you want to learn about other methods):

For each point i, we want to solve for 3D coordinates $P_i = \big[x_i, y_i, z_i\big]^T$, such that when they are projected back to the two images, they are close to the original 2D points. To project the 3D coordinates back to 2D images, we first write P_i in homogeneous coordinates, and compute $\mathbf{C}_1 P_i$ and $\mathbf{C}_2 P_i$ to obtain the 2D homogeneous coordinates projected to camera 1 and camera 2, respectively.

For each point i, we can write this problem in the following form:

$$\mathbf{A}_i P_i = 0,$$

where $\mathbf{A}_i$ is a 4×4 matrix, and P_i is a 4×1 vector of the 3D coordinates in the homogeneous form. Then, you can obtain the homogeneous least-squares solution (discussed in class) to solve for each P_i.

In your write up: Write down the expression for the matrix $\mathbf{A}_i$.

Once you have implemented triangulation, check the performance by looking at the reprojection error:

$$\texttt{err} = \sum_i \|p_{1i}, \widehat{p_{1i}}\|^2 + \|p_{2i}, \widehat{p_{2i}}\|^2$$

where $\widehat{p_{1i}} = Proj(\mathbf{C}_1, P_i)$ and $\widehat{p_{2i}} = Proj(\mathbf{C}_2, P_i)$.

Note: The `C1` and `C2` here are projection matrix of the form: $\mathbf{C}_1 = \mathbf{K}_1\mathbf{M}_1 = \mathbf{K}_1 \begin{bmatrix} I|0 \end{bmatrix}$ and $\mathbf{C}_2 = \mathbf{K}_2\mathbf{M}_2 = \mathbf{K}_2 \begin{bmatrix} R|t \end{bmatrix}$.

Q3.3 **(10 points)** Write a script `findM2.m` to obtain the correct `M2` from `M2s` by testing the four solutions through triangulations. Use the correspondences from `data/some_corresp.mat`. Save the correct `M2`, the corresponding `C2`, 2D points `p1`, `p2`, and 3D points `P` to `q3_3.mat`.

4 3D Visualization

You will now create a 3D visualization of the temple images. By treating our two images as a stereo-pair, we can triangulate corresponding points in each image, and render their 3D locations.

Q4.1 **(15 points)** Implement a function with the signature:

```
function [x2, y2] = epipolarCorrespondence(im1, im2, F, x1, y1)
```

This function takes in the x and y coordinates of a pixel on `im1` and your fundamental matrix $\mathbf{F}$, and returns the coordinates of the pixel on `im2` which correspond to the input point. The match is obtained by computing the similarity of a small window around the (x_1, y_1) coordinates in `im1` to various windows around possible matches in the `im2` and returning the closest.

Instead of searching for the matching point at every possible location in `im2`, we can use $\mathbf{F}$ and simply search over the set of pixels that lie along the epipolar line (recall that the epipolar line passes through a single point in `im2` which corresponds to the point (x_1, y_1) in `im1`).

There are various possible ways to compute the window similarity. For this assignment, simple methods such as the Euclidean or Manhattan distances between the intensity of the pixels should suffice. See Szeliski Chapter 11, on stereo matching, for a brief overview of these and other methods.
Implementation hints:

- Experiment with various window sizes.
- It may help to use a Gaussian weighting of the window, so that the center has greater influence than the periphery.
- Since the two images only differ by a small amount, it might be beneficial to consider matches for which the distance from (x_1, y_1) to (x_2, y_2) is small.

To help you test your `epipolarCorrespondence`, we have included a GUI:

```
function [coordsIM1, coordsIM2] = epipolarMatchGUI(im1, im2, F)
```

that can be found in `matlab/epipolarMatchGUI.m`.

This script allows you to click on a point in `im1`, and will use your function to display the corresponding point in `im2`. The process repeats until you right-click in the figure, and the sets of matches will be returned. See Figure 5.

It's not necessary for your matcher to get *every* possible point right, but it should get easy points (such as those with distinctive, corner-like windows). It should also be good enough to render an intelligible representation in the next question.
In you write up: Include a screenshot of `epipolarMatchGUI` with some detected correspondences.
Output: Save the matrix `F`, points `pts1` and `pts2` which you used to generate the screenshot to the file `q2_6.mat`.

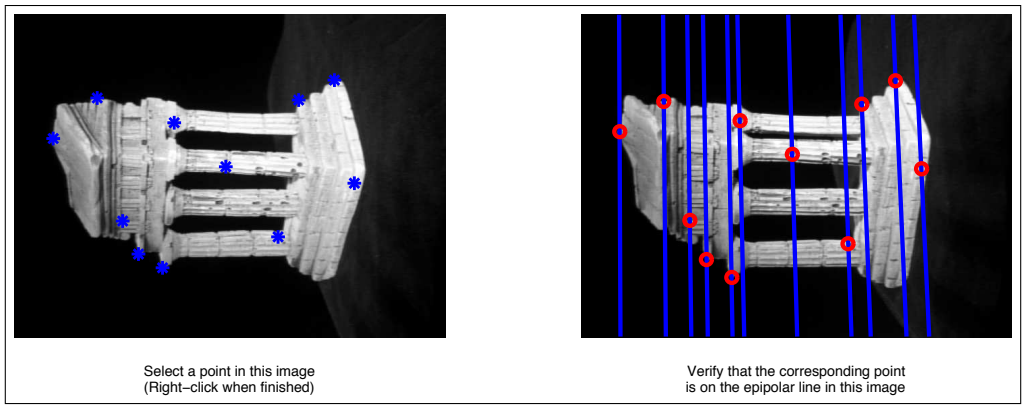

Figure 5: `epipolarMatchGUI` shows the corresponding point found by calling `epipolarCorrespondence`

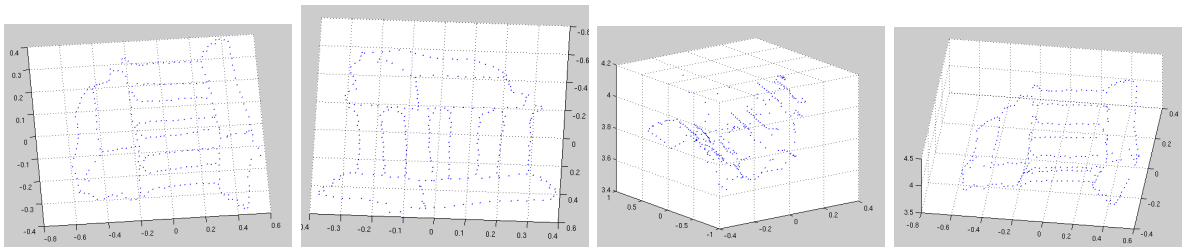

Figure 6: An example point cloud

Q4.2 **(10 points)** Included in this homework is a file `data/templeCoords.mat` which contains 288 hand-selected points from `im1` saved in the variables `x1` and `y1`.

Now, we can determine the 3D location of these point correspondences using the `triangulate` function. These 3D point locations can then plotted using the MATLAB function `scatter3`. Write a script `visualize.m`, which loads the necessary files from `../data/` to generate the 3D reconstruction using `scatter3`. The resulting figure can be rotated using the Rotate 3D tool, which can be accessed through the figure menubar.

In your submission/writeup: Take a few screenshots of the 3D visualization so that the outline of the temple is clearly visible, and include them with your homework submission.

Output: Again, save the matrix `F`, matrices `M1`, `M2`, `C1`, `C2` which you used to generate the screenshots to the file `q4_2.mat`.

5 Deliverables

If your andrew id is `bovik`, your submission should be the writeup `bovik.pdf` and a zip file `bovik.zip`. **Please submit both the pdf and the zip files directly to Canvas.**

We have provided `checkA4Format.m`, which ensures that your functions return the expected output dimensions. This is *not* a correctness test.

The zip file should include the following directory structure:

- ☐ `matlab/eightpoint.m`: implementation of the eight-point algorithm.
- ☐ `matlab/q2_1.mat`: file with output of Q2.1.

- ☐ `matlab/sevenpoint.m`: implementation of the seven-point algorithm.
- ☐ `matlab/q2_2.mat`: file with output of Q2.2.
- ☐ `matlab/essentialMatrix.m`: function to compute the essential matrix.
- ☐ `matlab/triangulate.m`: function to triangulate correspondences.
- ☐ `matlab/findM2.m`: script to compute the correct camera matrix.
- ☐ `matlab/q3_3.mat`: file with output of Q3.3.
- ☐ `matlab/epipolarCorrespondence.m`: function to compute correspondences with epipolar constraints.
- ☐ `matlab/q4_1.mat`: file with output of Q4.1.
- ☐ `matlab/q4_2.mat`: file with output of Q4.2.
- ☐ `matlab/camera2.m`: provided function. Do not change the function implementation.
- ☐ `matlab/epipolarMatchGUI.m`: provided function.
- ☐ `matlab/displayEpipolarF.m`: provided function.
- ☐ `matlab/refineF.m`: provided function. Do not change the function implementation.

We have also provided `checkA4Submission.m`, which verifies your directory structure. Please run this script before your final submission.