CSE 491 Introduction to Computer Vision: Project 4
3D Reconstruction

Instructor: Vishnu Boddeti
Due Date: Fri April 27, 2018 23:59:59

Instructions

1. **Integrity and collaboration:** Students are encouraged to work in groups but each student must submit their own work. If you work as a group, include the names of your collaborators in your write up. Code should **NOT** be shared or copied. Please **DO NOT** use external code unless permitted. Plagiarism is strongly prohibited and may lead to failure of this course.

2. If you have any question, please look at Google Classroom first. Other students may have encountered the same problem, and is solved already. If not, post your question there. We will respond as soon as possible.

3. **Write-up:** Items to be included in the write-up are mentioned in each question, and summarized in the Writeup section. Please note that we **DO NOT** accept handwritten scans for your write-up in this assignment. Please type your answers to theory questions and discussions for experiments electronically.

4. **Handout:** The handout zip file contains 3 items. `hw4.pdf` is the assignment handout. `data` contains 2 temple image files from the Middlebury MVS Temple Data set, as well as 3 `npz` files. `python` contains 1 script that you will make use of in this project.

5. **Submission:** Your submission for this assignment should be a pdf file `<msu-id.pdf>` and a zip file, `<msu-id>.zip`, composed of your write-up, your PYTHON implementations (including helper functions), and your implementations, results for extra credit (optional).

   Your final upload should have the files arranged in this layout:

   - `<msu-id>.pdf`
   - `<msu-id>.zip`
     - `python`
       - `camera2.py` (*provided*)
       - `eightpoint.py` (Q1.2)
       - `epipolarCorrespondence.py` (Q1.2)
       - `essentialMatrix.py` (Q1.3)

    - `testTempleCoords.py` (Q1.5)
    - `triangulate.py` (Q1.4)
    - *Any other helper functions you need*
- ec
    - `runEC.py` (Q?X, *required if EC is attempted*)
    - `ransacEightpoint.py` (Q1X, *optional*)

Please make sure you do follow the submission rule mentioned above before uploading your solution to Google Classroom. Assignments that violate this submission rule will be **penalized 10% of the total score**.

Overview

One of the major areas of computer vision is 3D reconstruction. Given several 2D images of an environment, can we recover the 3D structure of the environment, as well as the position of the camera/robot? This has many uses in robotics and autonomous systems, as understanding the 3D structure of the environment is crucial to navigation. You don't want your robot constantly bumping into walls, or running over human beings!

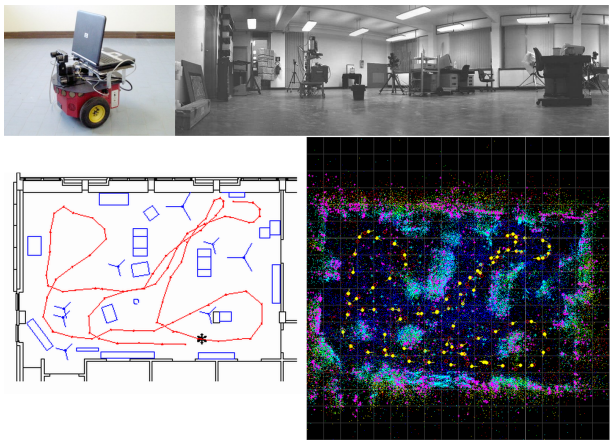

Figure 1: Example of a robot using SLAM, a 3D reconstruction and localization algorithm

You will be writing a set of functions to generate a sparse point cloud for some test images we have provided to you. The test images are 2 renderings of a temple from two different angles. We have also provided you with a `npz` file containing good point correspondences between the two images. You will first write a function that computes the fundamental matrix between the two images. Then write a function that uses the epipolar constraint to find more point matches between the two images. Finally, you will write a function that will triangulate the 3D points for each pair of 2D point correspondences.

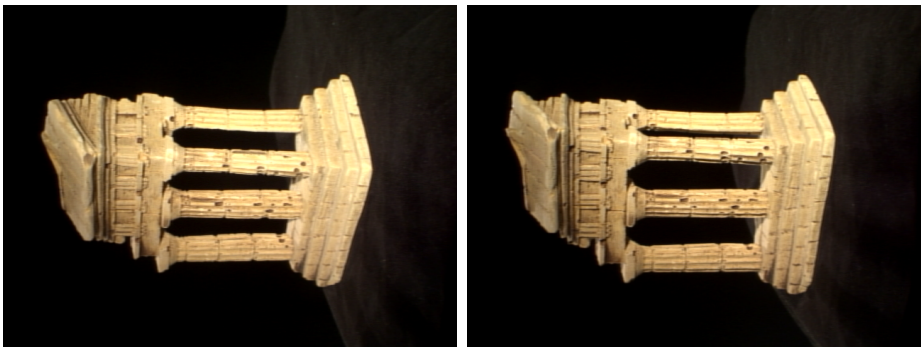

Figure 2: The two temple images we have provided to you

We have provided you with a few helpful `npz` files. `someCorresps.npz` contains good point correspondences. You will use this to compute the Fundamental matrix. `intrinsics.npz` contains the intrinsic camera matrices, which you will need to compute the full camera projection matrices. Finally `templeCoords.npz` contains some points on the first image that should be easy to localize in the second image.

For 3D reconstruction, multiple images are required, because without two images with a large overlapping portion the problem is mathematically underspecified. It is for this same reason biologists suppose that humans, and other predatory animals such as eagles and dogs, have two front facing eyes. Hunters need to be able to discern depth when chasing their prey. On the other hand herbivores, such as deer and squirrels, have their eyes position on the sides of their head, sacrificing most of their depth perception for a larger field of view. The whole problem of 3D reconstruction is inspired by the fact that humans and many other animals rely on dome degree of depth perception when navigating and interacting with their environment. Giving autonomous systems this information is very useful.

Theory

1. Triangulation (5 Points)

Structured light is a general way of retrieving 3D structure from a stationary camera. One very interesting way to do this is by using a light source and a thin shadow (maybe cast by a pencil). See http://www.vision.caltech.edu/bouguetj/ICCV98/ for an example.

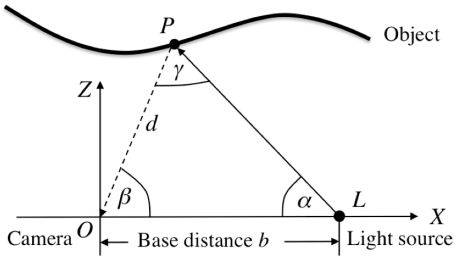

Figure 3: Camera and light source triangle diagram

This scheme uses a stationary lamp, and a stationary camera. Then a thin wand is waved above the object, and a shadow plane is cast. From the camera view, you know the angle to a point on the object is β, and from the position of the wand, you know the angle at which the shadow is cast from the light source is α. You also know how far the camera is from the light source, so you know the baseline distance b.

Given this information, how can you recover the position of the point in the scene?

2. Pure Rotation (5 Points)

Suppose I wanted to get a 3D Reconstruction of a mountain. Would I be able to do this by standing in one place at the base of the mountain, and capturing multiple images by only rotating the camera view?

Programming

1 Sparse Reconstruction

In this section, you will be writing a set of function to compute the sparse reconstruction from two sample images of a temple. You will first estimate the Fundamental matrix, compute point correspondences, then plot the results in 3D.

It may be helpful to read through **Q1.5** right now. In **Q1.5** we ask you to write a testing script that will run your whole pipeline. It will be easier to start that now and add to it as you complete each of the questions one after the other.

Q1.1 Implement the eight point algorithm: (20 Points)

In this question, you're going to use the eight point algorithm which is covered in class to estimate the fundamental matrix. Please use the point correspondences provided in `someCorresp.npz`.

Write a function with the following signature:

```
function F = eightpoint(x1, x2, M)
```

Where `x1` and `x2` are $N \times 2$ matrices corresponding to the (x,y) coordinates of the N points in the first and second image respectively. `M` is a scale parameter.

- You should scale the data by dividing each coordinate by M (the maximum of the image's width and height). After computing F, you will have to "unscale" the fundamental matrix like what you did in `project 3`.
- You must enforce the rank 2 constraint on F before unscaling. Recall that a valid F matrix will have all epipolar lines intersect at a certain point, meaning that there exists a non-trivial null space for F. In general, with real points, the SVD solution for F will not come with this condition. To enforce the rank 2 condition, decompose F with SVD to get the three matrices U, S, V such that $F = USV^T$. Then force the matrix to be rank 2 by setting the smallest singular value in S to zero, giving you a new S'. Now compute the proper fundamental matrix with $F' = US'V^T$.
- 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, you can select a point in one image, estimate the epipolar line in the other image and plot it in the other image.

In your write-up: Please include your recovered F and the visualization of some epipolar lines.

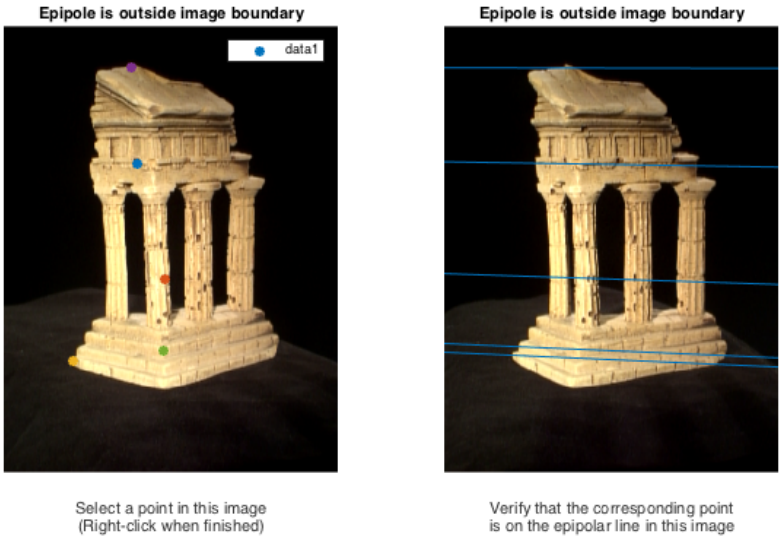

Figure 4: Example epipolar lines visualization

Q1.2 Find epipolar correspondences: (30 Points)

To reconstruct a 3D scene with a pair of stereo images, we need to find many point pairs. A point pair is two points in each image that correspond to the same 3D scene point. With enough of these pairs, when we plot the resulting 3D points, we will have a rough outline of the 3D object. You found point pairs in the previous homework using feature detectors and feature descriptors, and testing a point in one image with every single point in the other image. But here we can use the fundamental matrix to greatly simplify this search.

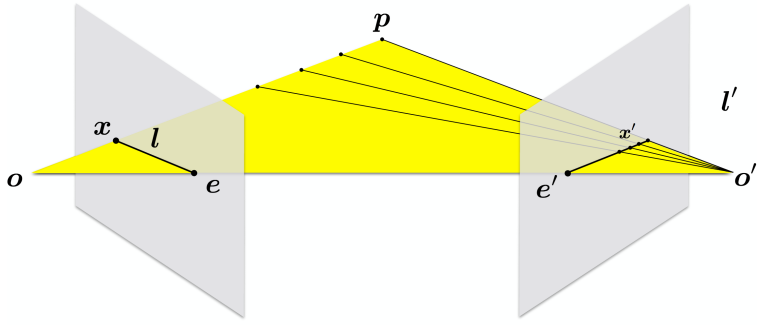

Figure 5: Epipolar Geometry (source Wikipedia)

Recall from class that given a point in one image (the left view in Figure 6), it could lie anywhere along the line from the point to the camera center. This line, along with the camera center of a second image (the right view in Figure 6) forms a plane that intersects with the image plane of the second camera. This results in a line in the second image which describes all the possible locations that the point in the first image may be found in the second image. This is the epipolar line, and we only need to search along this line to find a match for a point found in the first image.

Write a function with the following signature:

```
function x2 = epipolarCorrespondence(im1, im2, F, x1)
```

Where `im1` and `im2` are the two images in the stereo pair. `F` is the fundamental matrix computed for the two images using your eightpoint function. `x1` is an $N \times 2$ matrix containing the (x,y) points in the first image. Your function should return `x2`, an $N \times 2$ matrix, which contains the corresponding points in the second image.

To match one point in image 1, compute a small window of size w around the point. Then compare this target window to various windows of candidate points in the second image. For the images we gave you, simple Euclidean distance or Manhattan distance should suffice. Manhatten distance was not covered in class. Consider Googling it. Candidate points in the second image should only come from points that lie on (or very close to) the epipolar line created by the target point. Select the candidate point with the smallest window distance to the target point.

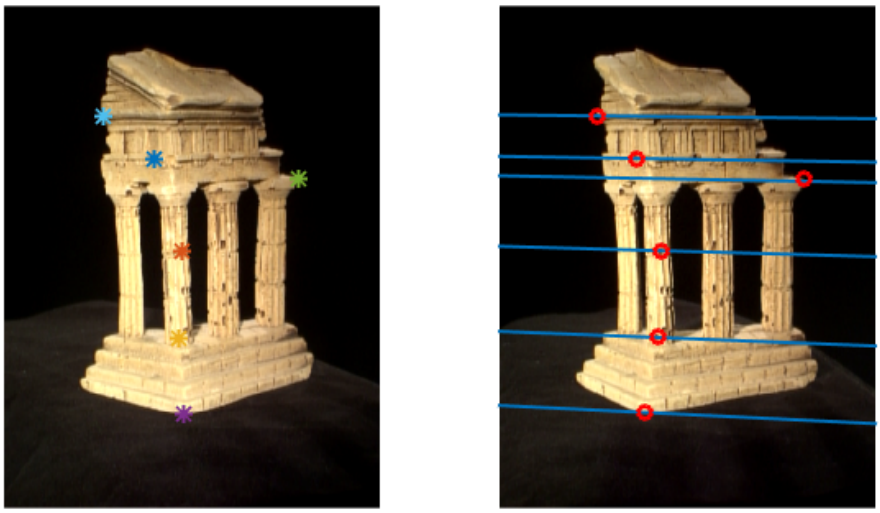

Figure 6: Epipolar Match visualization. A few errors are alright, but it should get most easy points correct (corners, dots, etc...)

In your write-up: Include a visualization of the the epipolar matches by running your implementation of `epipolarCorrespondence`. Mention the window size and distance metric you decided to use. Also comment on any cases where your matching algorithm consistently fails, and why you might think this is.

Q1.3 Write a function to compute the essential matrix: (5 Points)

In order to get the full camera projection matrices we need to compute the Essential matrix. So far, we have only been using the Fundamental matrix.

Write a function with the following signature:

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

Where `F` is the Fundamental matrix computed between two images, `K1` and `K2` are the intrinsic camera matrices for the first and second image respectively (contained in `intrinsics.npz`). `E` is the computed essential matrix. The intrinsic camera parameters are typically acquired through camera calibration.

Refer to the class slides for the relationship between the Fundamental matrix and the Essential matrix.

In your write-up: Write your estimated `E` matrix for the temple image pair we gave you.

Q1.4 Implement triangulation: (20 Points)

Write a function to triangulate pairs of 2D points in the images to a set of 3D points with the signature:

```
function X = triangulate(P1, x1, P2, x2)
```

Where `x1` and `x2` are the $N \times 2$ matrices with the 2D image coordinates and `X` is an $N \times 3$ matrix with the corresponding 3D points (in all cases, one point per row). `P1` and `P2` are the 3×4 camera projection matrices. Remember that you will need to multiply the given intrinsic matrices with your solution for the extrinsic camera matrices to obtain the final camera projection matrices. For `P1` you can assume no rotation or translation, so the extrinsic matrix is just $[\mathbf{I}|0]$. For `P2`, pass the essential matrix to the provided `camera2.py` function to get four possible extrinsic matrices. You will need to determine which of these is the correct one to use (see hint in **Q5**).

Refer to the class slides for one possible triangulation algorithm that uses SVD. Once you have it implemented, check the performance by looking at the re-projection error:

$$R = \sum_i \left\| x1_i - \hat{x1}_i \right\|^2 + \left\| x2_i - \hat{x2}_i \right\|^2$$

Where $\hat{x1}_i = P_1 * X^T$ and $\hat{x2}_i = P_2 * X^T$

Q1.5 Write a test script that uses `templeCoords`: (15 Points)

You now have all the pieces you need to generate a full 3D reconstruction. Write a test script `testTempleCoords.py` that does the following:

1. Load the two images and the point correspondences from `someCorresp.npz`
2. Run `eightpoint` to compute the fundamental matrix `F`
3. Load the points in image 1 contained in `templeCoords.npz` and run your `epipolarCorrespondences` on them to get the corresponding points in image 2
4. Load `intrinsics.npz` and compute the essential matrix `E`.
5. Compute the first camera projection matrix $\mathbf{P}_1$ and use `camera2.py` to compute the four candidates for $\mathbf{P}_2$
6. Run your `triangulate` function using the four sets of camera matrix candidates, the points from `templeCoords.npz` and their computed correspondences.
7. Figure out the correct $\mathbf{P}_2$ and the corresponding 3D points.

   Hint: You'll get 4 projection matrix candidates for camera2 from the essential matrix. The correct configuration is the one for which most of the 3D points are in front of both cameras (positive depth).
8. Use matplotlib's `scatter` function to plot these point correspondences on screen

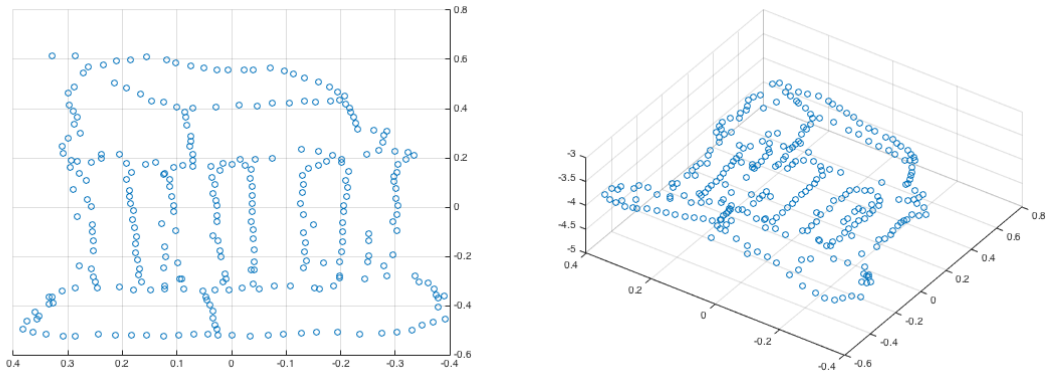

Figure 7: Sample Reconstructions

We will use your test script to run your code, so be sure it runs smoothly. In particular, use relative paths to load files, not absolute paths.

In your write-up: Include 3 images of your final reconstruction of the templeCoords points, from different angles.

Extra credit

If you attempt any of the extra credit parts, include a script `runEC.py` that will run your extra credit code, and generate any visualization plots. If you make use of any of the non-extra credit code, either copy that code into the `ec` directory, or set up the matlab path appropriately.

Q1X Implement RANSAC for fundamental matrix estimation (15 Points)

In real world application, correspondences will be noisy. As such, RANSAC is usually used for fundamental matrix estimation. Write a function with the following signature:

```
function [F] = ransacEightpoint(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 respectively. `M` is a scale parameter. You can make use of your `eightpoint.py` solution. We have provided some noisy correspondences in `some_corresp_noisy.npz`.

Compute the noisy point correspondences by using any pair of feature detector and descriptor you have learned about (whether in class or not).

In your write-up: Include the visualization of eipoplar lines using the fundamental matrix estimated from eightpoint and ransac using the noisy correspondences, and briefly analyze the results. Explain the error metric you used, how you decided which points were inliers.

Q2X Dense visualization for temple images (20 Points)

Instead of using the matplotlib built-in `scatter`, there are methods that give better dense 3D visualization. Please try some 3D visualization method other than `scatter`. **This video** shows an example of dense 3D points visulization. Feel free to use third party code, and it is not necessary to include it in your submission.

In your write-up: Include figures of of your visualization result and a brief explanation about they were obtained.

Writeup Summary

Theory Answer to both theory questions

Q1.1 Please include your recovered F and the visualization of the epipolar lines.

Q1.2 Include a screen shot of `epipolarMatchGui` running with your implementation of `epipolarCorrespondence`. Mention the window size and distance metric you decided to use. Also comment on any cases where your matching algorithm consistently fails, and why you might think this is.

Q3 Write your estimated `E` matrix for the temple image pair we gave you.

Q1.5 Include 3 images of your final reconstruction of the templeCoords points, from different angles.

Q1x Include visualization of of epipolar lines using the fundamental matrix estimated with and without ransac method, and analyze the results. Explain the error metric you used, how you decided which points were inliers.

Q2x Include figures of of your visualization result and a brief explanation about they were obtained.