EECS 442 Computer Vision: Homework 5

Instructions

- This homework is due **due at 5:00:00 p.m. on April 17, 2023**.
- The submission includes two parts:
  1. **To Canvas:** submit a `zip` file of all of your code.
     We have indicated questions where you have to do something in code in red.
     We have indicated questions where we will definitely use an autograder in purple.
     Please be especially careful on the autograded assignments to follow the instructions. Don't swap the order of arguments and do not return extra values.
     If we're talking about autograding a filename, we will be pulling out these files with a script. Please be careful about the name.
     Your zip file should contain a single directory which has the same name as your uniqname. If I (David, uniqname `fouhey`) were submitting my code, the zip file should contain a single folder `fouhey/` containing all required files.
     What should I submit? At the end of the homework, there is a canvas submission checklist provided. We provide a script that validates the submission format here. If we don't ask you for it, you don't need to submit it; while you should clean up the directory, don't panic about having an extra file or two.
  2. **To Gradescope:** submit a `pdf` file as your write-up, including your answers to all the questions and key choices you made.
     We have indicated questions where you have to do something in the report in blue.
     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/.
     The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.

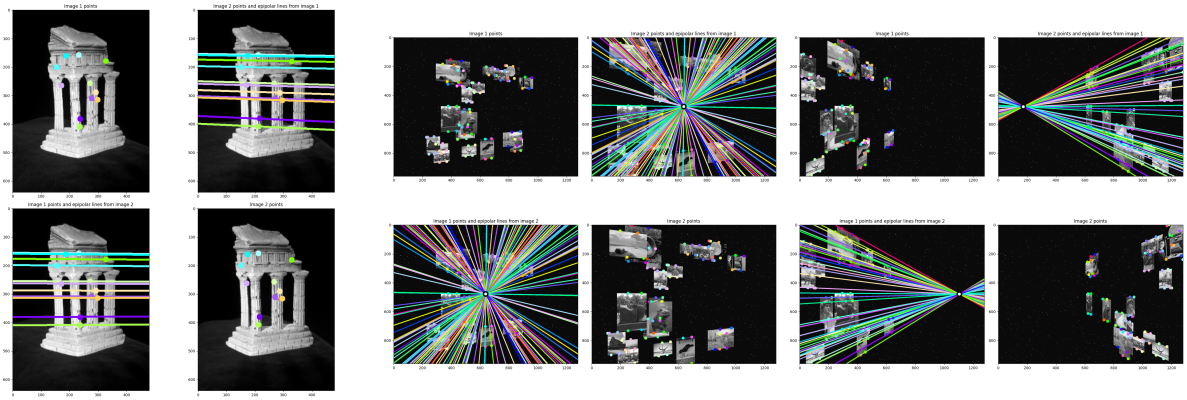
Python Environment

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. To make your life easier, we **recommend** you to install the latest Anaconda for Python 3.7 (https://www.anaconda.com/download/). This is a Python package manager that includes most of the modules you need for this course.

We will make use of the following packages extensively in this course:

- Numpy (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html).
- Matplotlib (http://matplotlib.org/users/pyplot_tutorial.html).
- OpenCV (https://opencv.org/).

1 Camera Calibration



Temple zrtrans reallyInwards

Figure 1: Epipolar lines for some of the datasets.

Task 1: Estimating M [35 points]

We will give you a set of 3D points $\{\mathbf{X}_i\}_i$ and corresponding 2D points $\{\mathbf{p}_i\}_i$. The goal is to compute the projection matrix $\mathbf{M}$ that maps from world 3D coordinates to 2D image coordinates. Recall that

$$\mathbf{p} \equiv \mathbf{MX}, \tag{1}$$

and (see foreword) by deriving an optimization problem. The script `task1.py` shows you how to load the data. The data we want you to use is in `task1/`, but we show you how to use data from Task 2 and 3 as well. **Credit:** The data from task 1 and an early version of the problem comes from James Hays's Georgia Tech CS 6476.

(a) (15 points) **Fill in `find_projection` in `task1.py`.**

(b) (5 points) **Report M** for the data in `task1/`.

(c) (10 points) **Fill in `compute_distance` in `task1.py`.** In this question, you need to compute the average distance in the image plane (i.e., pixel locations) between the homogeneous points $\mathbf{MX}_i$ and 2D image coordinates $\mathbf{p}_i$, or

$$\frac{1}{N}\sum_{i}^{N} ||\text{proj}(\mathbf{MX}_i) - \mathbf{p}_i||_2. \tag{2}$$

where $\text{proj}([x, y, w]) = [x/w, y/w]$. The distance quantifies how well the projection maps the points $\mathbf{X}_i$ to $\mathbf{p}_i$. You should use `find_projection` from part a). Note: You should feel good about the distance if it is **less than 0.01** for the given sample data. If you plug in different data, this threshold will of course vary.

(d) (5 points) **Describe what relationship, if any, there is between Equation 2 as above and Equation 6 in the HW5 Notes** Note that the points we've given you are well-described by a linear projection – there's no noise in the measurements – but in practice, there will be an error that has to minimize. Both equations represent objectives that could be used. If they are the same, show it; if they are not the same, report which one makes more sense to minimize. Things to consider include whether the equations directly represent anything meaningful.

2 Estimation of the Fundamental Matrix and Reconstruction

Data: we give you a series of datasets that are nicely bundled in the folder `task23/`. Each dataset contains two images `img1.png` and `img2.png` and a numpy file `data.npz` containing a whole bunch of variables. The script `task23.py` shows how to load the data.

Credit: `temple` comes from Middlebury's Multiview Stereo dataset. The images shown in the synthetic images are described in HW1's credits.

Task 2: Estimating F [35 points]

(a) (15 points) **Fill in** `find_fundamental_matrix` in `task23.py`. You should implement the eight-point algorithm. Remember to normalize the data and to reduce the rank of $\mathbf{F}$. For normalization, you can scale the image size and center the data at 0.

(b) (10 points) **Fill in** `compute_epipoles`**.** This should return the homogeneous coordinates of the epipoles – remember they can be infinitely far away!

(c) (5 points) **Show epipolar lines for** `temple`**,** `reallyInwards`**, and another dataset of your choice.**

(d) (5 points) **Report the epipoles for** `reallyInwards` **and** `xtrans`.

Task 3: Triangulating X [30 points]

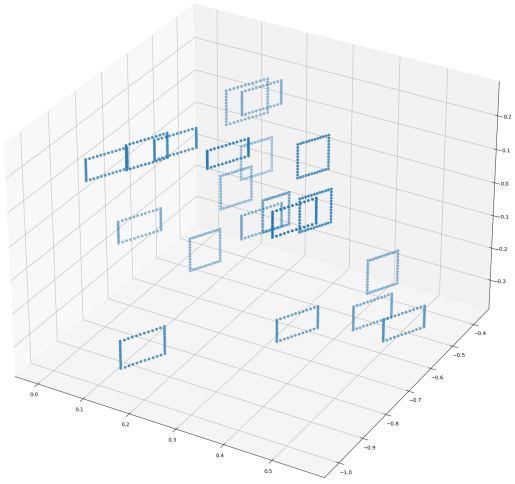


Figure 2: Visualizations of **reallyInwards** reconstructions

The next step is extracting 3D points from 2D points and camera matrices, which is called triangulation. Let $\mathbf{X}$ be a point in 3D.

$$\mathbf{p} = \mathbf{M}_1\mathbf{X} \quad \mathbf{p}' = \mathbf{M}_2\mathbf{X} \tag{3}$$

Triangulation solves for $\mathbf{X}$ given $\mathbf{p}, \mathbf{p}', \mathbf{M}_1, \mathbf{M}_2$. We'll use OpenCV's algorithms to do this.

(a) (5 points) **Compute the Essential Matrix E for the Fundamental Matrix F.** You should do this for the dataset `reallyInwards`. Recall that

$$\mathbf{F} = \mathbf{K}'^{-T}\mathbf{E}\mathbf{K}^{-1} \tag{4}$$

and that $\mathbf{K}, \mathbf{K}'$ are always invertible (for reasonable cameras), so you can compute $\mathbf{E}$ straightforwardly.

(b) (15 points) **Fill in `find_triangulation` in `task23.py`.**

The first camera's projection matrix is $\mathbf{K}[\mathbf{I}, \mathbf{0}]$. The second camera's projection matrix can be obtained by decomposing $\mathbf{E}$ into a rotation and translation via `cv2.decomposeEssentialMat`. (Note: $\mathbf{E}$ can be obtained using the formula from part a) This function returns two matrices $\mathbf{R}_1$ and $\mathbf{R}_2$ and a translation $\mathbf{t}$. The four possible camera matrices for $\mathbf{M}_2$ are:

$$\mathbf{M}_2^1 = \mathbf{K}'[\mathbf{R}_1, \mathbf{t}], \quad \mathbf{M}_2^2 = \mathbf{K}'[\mathbf{R}_1, -\mathbf{t}], \quad \mathbf{M}_2^3 = \mathbf{K}'[\mathbf{R}_2, \mathbf{t}], \quad \mathbf{M}_2^4 = \mathbf{K}'[\mathbf{R}_2, -\mathbf{t}] \tag{5}$$

You can identify which projection is correct by picking the one for which the most 3D points are in front of both cameras. You'll have to run `cv2.triangulatePoints` in order to do this. Once you have the 3D point $\mathbf{X}$ and a potential camera matrix $\mathbf{M}_2$, you can compute $\mathbf{M}_2\mathbf{X}$ and then check the last entry w of the resulting homogeneous coordinate $[u, v, w] = \mathbf{M}_2\mathbf{X}$. The reason why this works is that the extrinsics put the 3D point in the camera's frame, where $z < 0$ is behind the camera, and the last row of $\mathbf{K}$ is $[0, 0, 1]$ so this does not change things.

Then, triangulate the 2D points using `cv2.triangulatePoints` using the $\mathbf{M}_2$ that makes the most points positive.

Note: you can approach this in two ways: (1) try all the $\mathbf{M}_2$ and figure out which one puts the most points in front of the camera; then do the triangulation to get the final set of points; or (2) try all the $\mathbf{M}_2$ options and keep track of both the "best" set of points and the $\mathbf{M}_2$, and then return "the best" set of points. Either is fine.

(c) (10 points) **Put a visualization of the point cloud for `reallyInwards` in your report.** You can use `visualize_pcd` in `utils.py` or implement your own.

References and Credits

- Temple dataset used in Tasks 2 and 3: http://vision.middlebury.edu/mview/data/.
- 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.