Computer Vision
Homework 2: Structure from Motion (SfM)

Due on Wednesday, October 12, 2022

1 Introduction

In this homework, you will implement several crucial steps of structure from motion (SfM) pipeline. Given several images of an object, SfM is to estimate the camera poses and 3D point cloud of all images. For convenience, we already provide sample images and the correspondence matches. Before doing this homework, you need to get familar with the following CV knowledges:

1. **Camera Projection Model.** The camera intrinsic matrix and homogeneous coordinates.
2. ***SE*(3) Transformations.** The rotation, translation, and transformation matrix.
3. **Python and Numpy Stuffs.**

To write this homework, please modify **main.py** in the provided zip file.

2 Download and Installation

You can download source codes from here.
Please use the following command to install required python libraries.

```
conda create -n cvhw2 python=3.9
conda activate cvhw2
pip install -r requirements.txt
conda install -c anaconda pyqt
```

3 Problems

For each question, the provided python script has the corresponding placeholders. Please read the comments in the source code for more information.

3.1 Camera Pose from Essential Matrix

Given corresponding matches and essential matrix of two images, we need to estimate the camera pose, i.e., rotation (R) and translation (T). However, we will have four answers of camera poses from a single essential matrix. We need to use the corresponding matches to decide which one is correct.

Step 1. To infer the rotation (R), we first apply singular value decomposition (SVD) to our essential matrix, i.e., $E = UDV^T$. We first define $E = MQ$ where $M = UZU^T$ and $Q = UWV^T$ or UW^TV^T, and

$$Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

We can simply compute rotation as $R = \det(Q) \cdot Q$. For applying SVD, you can directly use **np.linalg.svd**.

Step 2. Given $E = U\Sigma V^T$, translation T is either u_3 or $-u_3$, where u_3 is the third column vector of U.

Step 3. Implement this in **estimate_initial_RT()**. We provide the correct R and T to help you debug. Please note that you need to return the four combinations of your estimated R and T abovementioned.

3.2 Linear 3D Points Estimation

Before deciding which one of our estimated poses is correct, we need to compute the 3D locations of the corresponding matches, and we will use direct linear transformation (DLT) to solve this problem.

Notations. For each image i, we have $p_i = \alpha M_i P$, where α is the homogeneous scale, $P = (X, Y, Z, W)$ is a 3D point in homogeneous coordinates, $p_i = (u, v, 1)$ is the homogeneous image coordinate of the point, and M_i is the projection matrix.

Explanations. Since α is a scalar, p_i and $M_i P$ should be parallel. In other words:

$$p_i \times M_i P = 0 \ ,$$
$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \times \begin{bmatrix} M_i^1 P \\ M_i^2 P \\ M_i^3 P \end{bmatrix} = 0 \ ,$$

Where M_i^{1-3} indicate different rows of M_i. After recalling vector cross product operation in your high school, the above equation can be rewritten as:

$$\begin{bmatrix} vM_i^3 - M_i^2 \\ M_i^1 - uM_i^3 \\ uM_i^2 - vM_i^1 \end{bmatrix} \cdot P = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad .$$

Since $uM_i^2 - vM_i^1$ is just a linear combination of $vM_i^3 - M_i^2$ and $M_i^1 - uM_i^3$, it is useless for us. Hence, we have two linear independent equations for each image.
When we have multiple images, we can establish the matrix of DLT as:

$$\begin{bmatrix} v_1 M_1^3 - M_1^2 \\ M_1^1 - u_1 M_1^3 \\ \vdots \\ v_n M_n^3 - M_n^2 \\ M_n^1 - u_n M_n^3 \end{bmatrix} \cdot P = 0.$$

Then, we can solve P by SVD. Please implement this in **linear_estimate_3d_point()**. We provide answers to help you debug.

3.3 Non-Linear 3D Points Estimation

Since the linear estimation from SVD is not robust to noise, we need to further apply non-linear optimization to our estimated points. In this question, you need to implement Gauss-Newton approach.

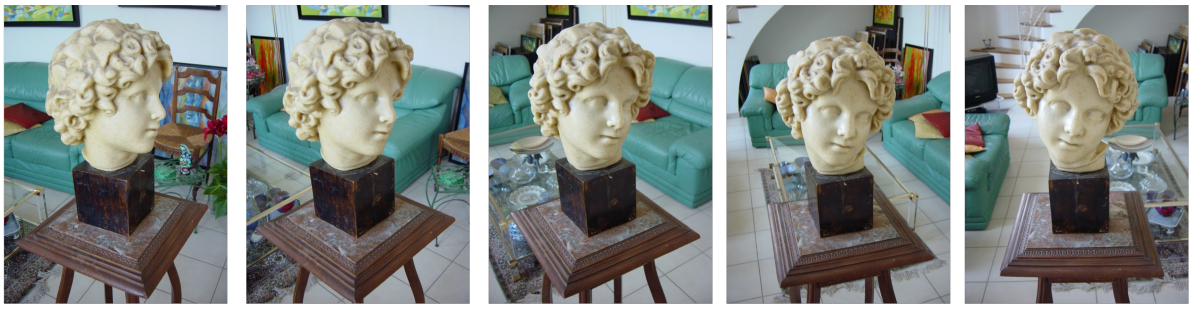
Reprojection error and Jacobian. Given n images and each image is denoted as i, we compute $y = M_i P$ where M_i is projection matrix and $P = [X, Y, Z, 1]$ is the 3D location of a point. The projected image coordinate of P is then:

$$p_i' = \begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{y_3} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \ .$$

Given the ground truth projected image coordinate p_i, the reprojection error is:

$$e_i = p_i' - p_i \ ,$$

(a)



(b)

Figure 1: (a) All sample images. (b) The 3D point cloud visualization.

and the Jacobian matrix is:

$$J = \begin{bmatrix} \frac{\partial e_1}{\partial X_1} & \frac{\partial e_1}{\partial Y_1} & \frac{\partial e_1}{\partial Z_1} \\ & \vdots & \\ \frac{\partial e_{2n}}{\partial X_n} & \frac{\partial e_{2n}}{\partial Y_n} & \frac{\partial e_{2n}}{\partial Z_n} \end{bmatrix} .$$

Notice that we have $2n$ reprojection errors (i.e. e_{2n}) since the reprojection error consists of values for both u and v.
Please implement this in **reprojection_error()** and **jacobian()**. We also provide the answer for you.

Optmization. Implement Gauss-Newton optimization after you finish the above questions. You can use your linearly estimated points as your initial guess and run the optimization for 10 iterations. The iteration of Gauss-Newton optimization can be simply implemented by:

$$P = P - (J^T J)^{-1} J^T e \ ,$$

where J and e are Jacobian and reprojection error. Please implement this in **nonlinear_estimate_3d_point()**, and make sure that the final reprojection error is lower than the one of linearly estimated points.

3.4 Decide the Correct RT

Now let's find out which RT is the correct one. You can do this by

1. First, compute the 3D locations of every pairs of corresponding matches for each estimated R and T.
2. For each R, T, you can just rotate and translate all 3D points. The correct R, T is the one in which there are most projected 3D points having positive z-coordinate for both images.

Please implement this in **estimate_RT_from_E()**
If all problems above are correctly implemented, our python script will continue to run a simple SfM pipeline for all sample images. The final estimated 3D point cloud will be stored in **results.npy**. You can then type **"python PtsVisualizer/visualize.py results.py"** to see the results. The correct results should be like Figure 1.

3.5 Write Your Report

Write down your insights and results of all problems in a short report. The file name should be **report.pdf**.

4 Submission

Please compress all source codes and report in a single zip file, and upload it to eLearn system. The name should be **HW2_YOURID.zip**.

Acknowledgements. This homework is modified from https://web.stanford.edu/class/cs231a/.