ROB 498/599 3D Robot Perception: HW1

Term: Fall 2023
Instructor: Jason J. Corso, EECS, University of Michigan
Due Date: 2nd October 2023
Version: 1 (September 6, 2023)

Constraints: This is an open-world homework assignment. You can search for resources on the web; talk to your friends about the assignment; do anything you would do in a real-world setting; but you must do your own work; you may not copy from anyone or any LLM.

Goals: Implement what you have learnt in lectures about camera calibration, and multi-view geometry to prepare a sparse 3D reconstruction of a structure with the given data

General Instructions

This course requires use of a modern set of programming tools, with which you may not be acquainted. The professor and the GSIs are here to help you, but you want to be in the habit of self-learning for this type of thing as it will be required both in future studies and when you are working a job.

This is a list of tools and where you can find information/documentation. I have done everything I can to make this list comprehensive, but something may have been missed.

- **git**: This is the basic tool used in contemporary software development for version control. Documentation: `https://git-scm.com/docs`.
- **python**: The modern *language of the realm* when it comes to computer vision, machine learning, deep learning and data science. Learn it, use it. I recommend Python 3 over Python 2 since Python 2 is now deprecated (and Python 3 has support for types, which improves its power). Python virtual environments are recommended for this course. Documentation: `https://docs.python.org/3/`.
- **numpy**: The basic numerical and linear algebra package for python. Documentation: `https://docs.scipy.org/doc/numpy-1.15.1/user/`.
- **matplotlib**: Plotting tools for python, integrated with scipy and numpy. Documentation: `https://matplotlib.org/users/index.html`.
- **opencv**: A basic computer vision library that has core functionality needed for some of our work in the course. In some cases, we will reimplement methods from opencv for the pedagogical value, in others we may use methods out of the box. OpenCV is written in C++/CUDA and has python bindings. Documentation: `https://docs.opencv.org/3.4/`.

Doubts and Discussions: We have added a new chat group on Perusall for doubts on the assingment. Kindly direct all your questions in that group.

Submission Instructions: Submit a `.zip` file containing your write-up and a folder `scripts` with all the python scripts on Canvas.

Problem 1 (20): Camera Calibration

i. In our recent class discussions, we covered the intriguing topic of camera calibration, specifically the process of finding camera projection matrices for a set of 2D image points and their corresponding 3D world coordinates. This concept is essential in computer vision, as it allows us to establish a mathematical relationship between the physical world and the images captured by a camera.

In this problem you are given a set of 2D points (x,y) and their corresponding 3D points(X,Y,Z) . Your task is to calculate the projection matrix. **Report your Projection Matrix in the write-up**. Recall:

$$p = MP \tag{1}$$

$$\begin{bmatrix} s*x \\ s*y \\ s \end{bmatrix} = \begin{bmatrix} -M_1- \\ -M_2- \\ -M_3- \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2}$$

Hint: Formulate the problem as (3) and estimate the parameters using SVD under the constraint $\|x\|_2 = 1$.

$$Ax = 0 \tag{3}$$

ii. Now that you have successfully derived the camera projection matrix (M) in the previous section, the next step is to evaluate the accuracy of this matrix in estimating the 2D location of 3D points on the image plane. We can achieve this by computing the

re-projection error, a crucial metric that quantifies how well our camera projection model aligns with the observed 2D image points. Your error should be **less than 0.01**.The error is given by (4). **Report your Reprojection Error in your write-up**

$$\frac{1}{N}\sum_{i=1}^{N}\|\text{proj}(MX_i)-p_i\|_2 \tag{4}$$

Sparse 3D Reconstruction

In the next few questions, we are going to try and get a sparse 3D reconstruction of a Temple by using the given data. The images are from the Middlebury Vision Datasets. We will first find the fundamental matrix and then use this matrix to calculate epipolar correspondences of feature points in image 1 to that of image 2. Once we have these points, we will run a triangulation algorithm and estimate the 3D location of feature points, reconstructing the object by its mapped features. The next set of questions will use `sparse_reconstruction.py`

Problem 2 (20): Estimation of Fundamental Matrix

In this question, we are going to estimate the Fundamental Matrix by implementing the 8 point algorithm. Complete the function `compute_fundamental_matrix(pts1, pts2, scale)`. Here, `pts1` and `pts2` are $N \times 2$ matrices which correspond to the (x, y) coordinates in image1 and image2 respectively and `scale` is the maximum of image height and width.

1. Use the data from `data/corresp_subset.npz`
2. Normalize the data points by scale factor and implement the 8 point algorithm to estimate the F matrix
3. Recall that F is a rank 2 matrix as the epipolar lines we estimate using F intersect at a point. This means that there exists a valid null space for F. The F matrix you estimate from 8 point algorithm would not necessarily be a rank 2 matrix. Verify this and get the closest rank 2 approximation of F using SVD.
4. Now that you have a rank 2 F, de-normalize the matrix. Recall if $p^T F p = 0$, $p_{norm} = Tp$ then $F_{\text{denorm}} = T^T F T$
5. There is a helper function called `epipolar_lines_GUI_tool(I1, I2, F)` in `helper_functions.py` that can be used to visualize epipolar lines using the fundamental matrix. Verify your code by visualizing the epipolar lines and ensure that the lines pass through the points selected in image 1. **Attach a screenshot of the result of the GUI as shown in fig(1)**

Problem 3 (20): Find Epipolar Correspondences

Your task is to find epipolar correspondences between feature points in image 1 and image 2 using the given fundamental matrix F. Complete the function `compute_epipolar_correspondences(img1,img2, F, pts1)`

1. Use the data from `data/temple_coordsnpz`
2. Estimate an epipolar line corresponding to a feature point by using the fundamental matrix. The points on this line in image2 are the candidate points for our correspondences.
3. Select a small window around these points and calculate the similarity between this window and a target window around the feature point in image1. You can use the manhattan distance as a similarity metric.
4. There is a helper function in the file `helper_functions.py` called `epipolar_correspondences_GUI_tool(I1, I2, F)` that can be used to verify the correspondences. They need not be perfect but should be able to capture major features. **Attach a screenshot result of the GUI in your write up**

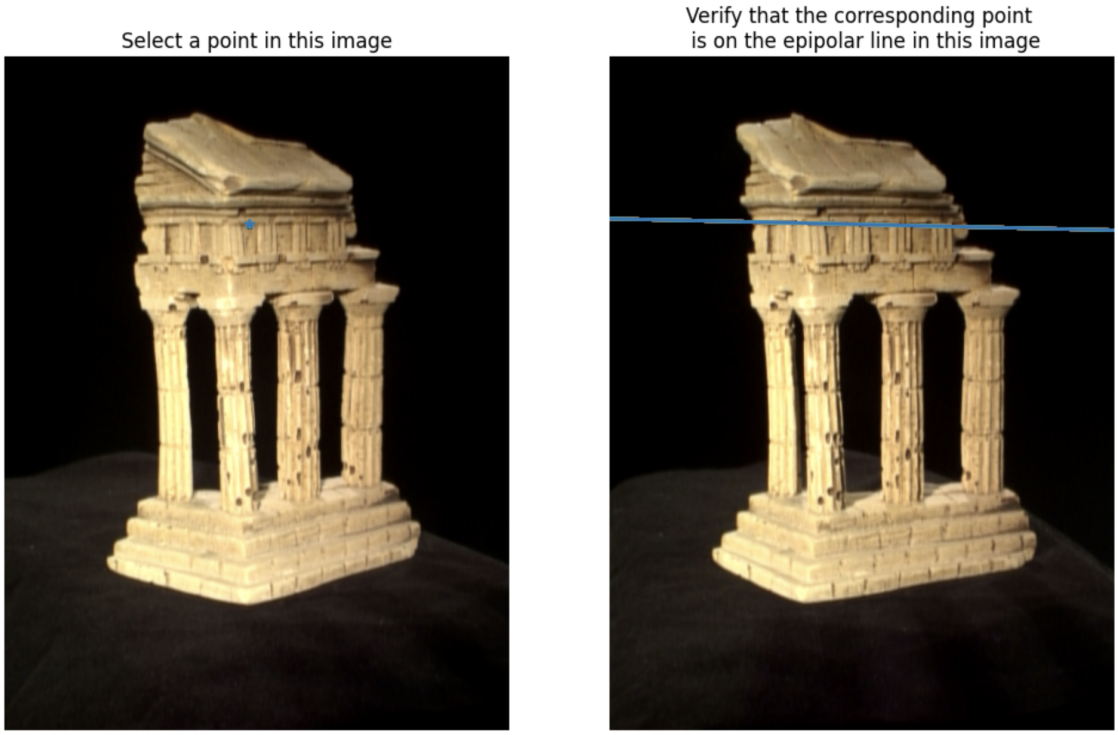

Figure 1: Sample image for GUI output for epipolar lines

Problem 4 (10): Compute the Essential Matrix

Complete the function `compute_essential_matrix(K1, K2, F)` that returns an essential matrix. K1 and K2 are the intrinsic matrices of camera1 and camera2. The intrinsics are stored in the file `data/intrinsics.npz`

Problem 5 (20): Triangulation

Now that we have the epipolar correspondences and the essential matrix, we can compute the triangulation and generate a point cloud of the structure. Complete the function `triangulate_points(E, pts1_ep, pts2_ep)` where E is the essential matrix, pts1_ep is the feature points in image1, pts2_ep is the epipolar correspondence computed from the previous question. Ensure that this function returns a 3D point cloud in the shape of $N \times 3$.

1. Assume camera1 is your origin and so the extrinsic matrix for camera1 is the identity matrix with 0 translation. Compute the extrinsic matrix of camera 2 from the essential matrix using texttttcv2.decomposeEssentialMat(). Note that this returns two Rotation and one translation matrix that would give you 4 possible extrinsic matrices: $[R1|t], [R2|t], [R1|-t], [R2|-t]$. Choose the combination that results in majority of the points being in front of the camera.
2. Find the 3D point cloud using triangulation. Recall that we have 2 projection matrices and hence two projection equations to estimate the points in 3D space. Construct the problem as eq (3) and solve using SVD under the constraint $\|x\|_2 = 1$. **Calculate the reprojection error and mention it in your write-up**
3. Get the 3D point cloud from cv2.triangulate as well to compare your result. **Calculate the reprojection error and mention it in your write-up. Is there a difference in the output? Why do you think so?**

Problem 6 (10): Connecting the dots and visualizing the point cloud

Congratulations! You now have everything needed to compute a sparse 3D construction of the temple! Complete the main function in `sparse_reconstruction.py`

1. Get the Fundamental Matrix using the data from `data/some_corresp.npz`
2. Get the epipolar correspondences of data points from `data/temple_coords.npz`
3. Get the Essential Matrix
4. Perform Triangulation and get the 3D point clouds

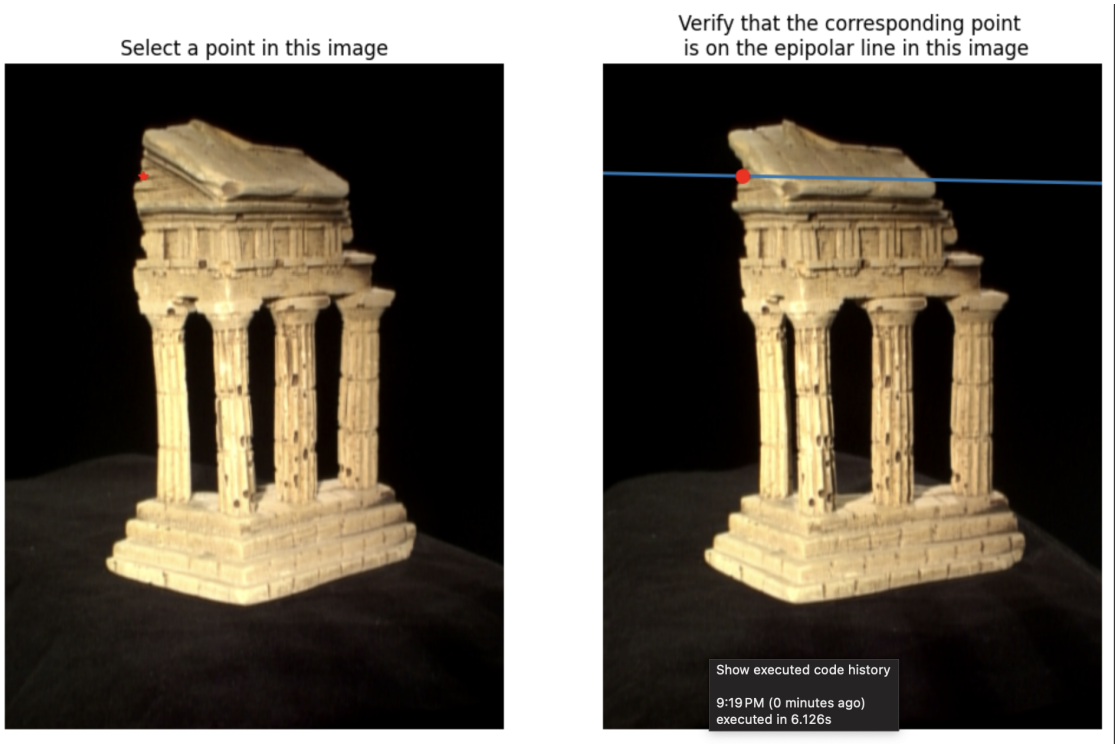

Figure 2: Sample image for GUI output for epipolar correspondences

5. Complete the function `visualize(point_cloud)` to visualize the 3D point cloud using `matplotlib` or `plotly`

Attach image of reconstructed object from both sets of point clouds in your writeup

References and Credits:
Parts of this homework are from David Fouhey's EECS 442 and CMU 16-385

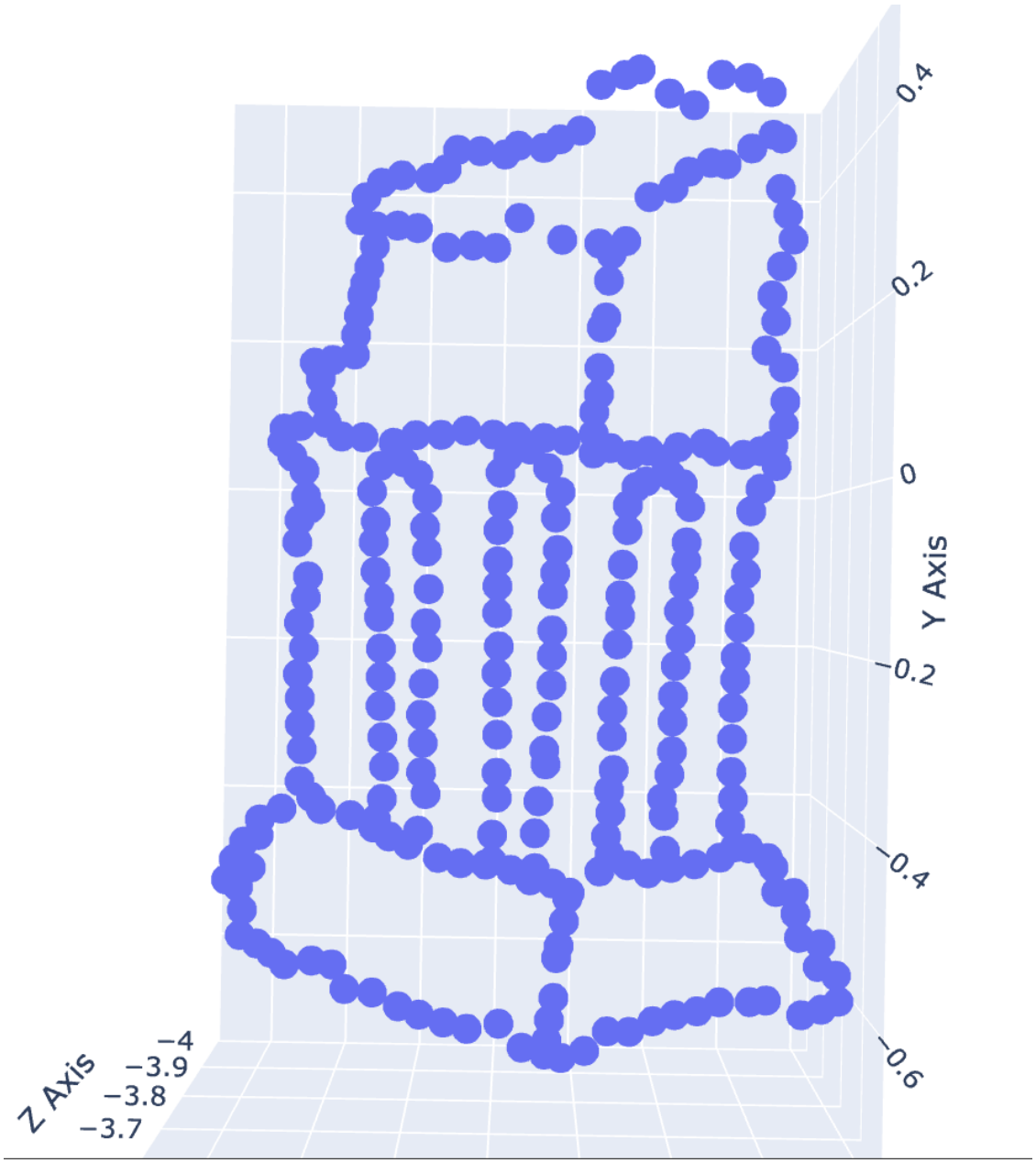

Figure 3: Sparse Reconstruction