CS4277/CS5477 Assignment 3: Structure From Motion and Bundle Adjustment

1. Overview

In this assignment, you will perform structure from motion and bundle adjustment to simultaneously obtain extrinsic camera parameters i.e. poses and perform 3D reconstruction.

References: Lecture 10.

Honour Code. This coding assignment constitutes **10%** of your final grade in CS4277/CS5477. Note that plagiarism will not be condoned! You may discuss with your classmates and check the internet for references, but you MUST NOT submit code/report that is copied directly from other sources!

2. Submission Instructions

Items to be submitted:

- **Source code (code/).** This is where you fill in all your code.
  - preprocess.py
  - sfm.py
  - bundle_adjustment.py
- **Results (predictions/).** Upload your results.
  - mini-temple/
    - results/
      - no-bundle-adjustment/ # all contents
        - all-extrinsic.json # all camera poses
        - points3d.npy # 3d reconstructed points
        - registration-trajectory.txt # order of registration
        - correspondences2d3d.json # all 2d-3d correspondences
      - bundle-adjustment/ # same as above
  - temple/
    - results/
      - no-bundle-adjustment/ # same as above.
- **Report (report.pdf).** This should describe your implementation and be no more than three pages. You may include diagrams/results of your implementation.

Please clearly indicate your name and student number (the one that looks like A1234567X) in the report as well as the top of your source code. Zip the files together and name it in the following format: **A1234567X_lab3.zip** (replace with your student number). Opening the zip file should show similar to lab1.

Submit your assignment by **5 April 2022, 2359HRS** to LumiNUS. 25% of the total score will be deducted for each day of late submission.

3. GETTING STARTED

This assignment as well as the subsequent ones require Python 3.7, or later. You need certain python packages, which can be installed using the following command:

```
pip install -r requirements.txt
```

If you have any issues with the installation, please post them in the forum, so that other students or the instructors can help accordingly.

4. PREPROCESSING: SCENE GRAPH

In structure from motion, we first begin with finding 2d point correspondences between pairs of images to build a scene graph. To find point correspondences, we perform the following steps:

1. Detect SIFT features within each image as key points.
2. Match detected key points across images through using feature matching of SIFT descriptors.
3. Filter out noisy matches using geometric verification. We will use RANSAC to robustly identify and filter outlier matches.
4. Finally, we will build a scene graph. Scene graphs have images as nodes. Edges are added between images that have sufficient inlier matches from step 3.

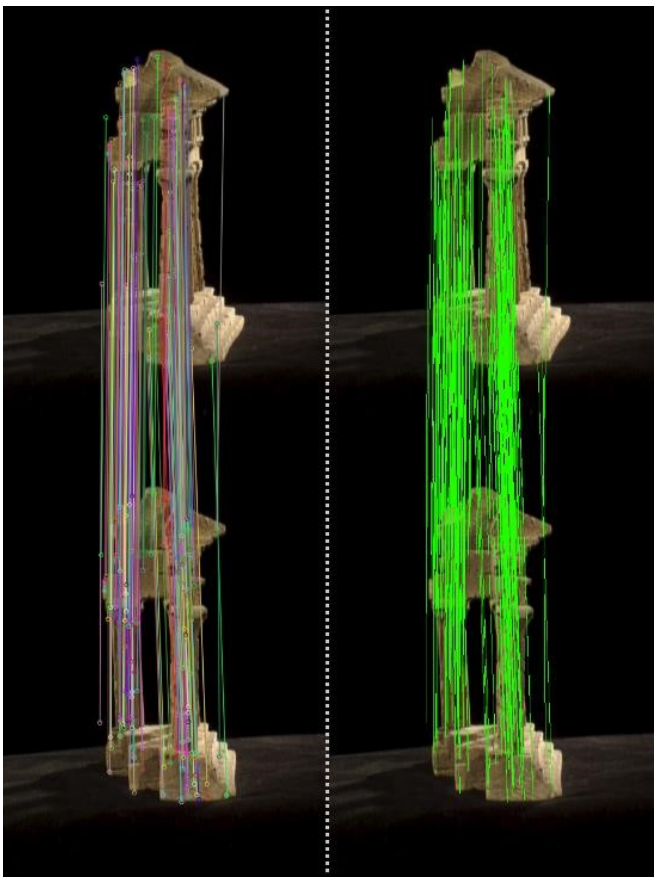

Figure 1: SIFT feature matching (left); RANSAC inlier matches (right)

5. STRUCTURE FROM MOTION

In our assignment, we will be implementing incremental Structure from Motion (SfM), where images are iteratively registered. The first pair of images to initialize our incremental structure from motion is important. We select the image that have the highest number of inlier image correspondences in our scene graph. From our initial image pair (I_1,I_2), we:

1. We set I_1 to have camera pose [$\mathbf{I}_3|\mathbf{0}$] where $\mathbf{I}$ is a 3x3 identity matrix and $\mathbf{0}$ 3-dim zero vector, where poses of other cameras are relative to the camera of the first image.
2. Then, we find the essential matrix between the between the image pair using the 5-point algorithm with RANSAC for robust estimation. Note that we can use the 5-point algorithm for filtering out noisy feature matches in pre-processing step 3.
3. We then recover the relative camera pose of the second image from the essential matrix and set the camera pose of I_2 to [$\mathbf{R}|\mathbf{t}$] where $\mathbf{R}$ is the recovered rotation matrix and $\boldsymbol{t}$ is the recovered translation vector.
4. From the recovered poses, we perform triangulation of inlier correspondences to obtain our initial set of reconstructed 3D scene points.

Figure 2: Initial image pair

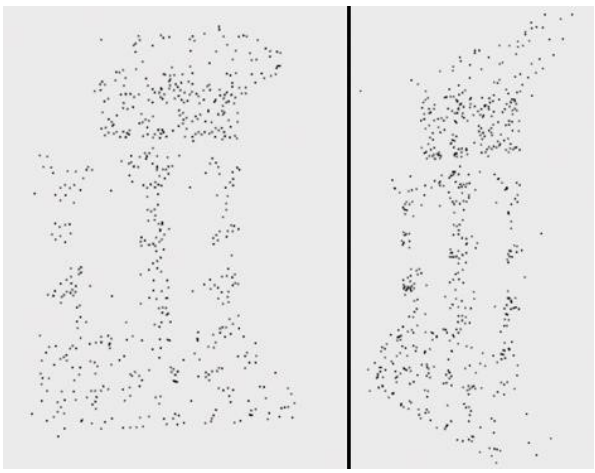

Figure 3: Initial 3D scene points from triangulation of feature correspondences.

Once, we have our initial 3D scene points and initial set of cameras, we will perform iterative registration + 3D reconstruction of the remaining images in the scene graph. We will perform the following steps for each iteration:

1. **Register new image.**
    a. We will find the next image to register by finding the image that has the highest inlier correspondences with any previously registered image. Specifically, we find the next best image pair (I_{new}, I_{reg}) in the scene graph with the highest inlier matches where one of the images in the pair is an unregistered image and the other is a registered image I_{reg}.
    b. Using the 2d-3d correspondence of the registered image with the 3D scene points and 2d-2d correspondences between the image pair, we will find 2d-3d correspondences between the I_{new} and the scene points.
    c. Then, using the 2d-3d correspondences between keypoints in I_{new} and scene points, we will recover the camera pose of I_{new} by solving the Perspective-n-Point problem with RANSAC.
2. **Reconstruct scene points.** From the new image I_{new}, we will add new 3D scene points using triangulation of 2D point correspondences with previously registered images. In this assignment, we will simplify this step by only triangulating with the I_{reg}.
3. **Bundle Adjustment.** We will refine the camera parameters of registered cameras and the 3D scene points using bundle adjustment.

The output of the incremental SfM will be camera poses for each image and the 3D reconstruction of the scene i.e. Figure 4.

Figure 4: Results of incremental SfM (without bundle adjustment)

6. BUNDLE ADJUSTMENT

For bundle adjustment, we will be minimization the reprojection errors between the scene points and their corresponding 2D points using. In this assignment, we will define the reprojection error between the 2D image points x_i and their corresponding 3D point X_i as:

$$\text{Reprojection error} = d(x_i, P(X_i, \mathbf{R}_i, \mathbf{t_i}, \mathbf{K}))$$

where $\mathbf{R_i}$, $\mathbf{t_i}$ and $\mathbf{K_i}$ refers to the camera rotation, translation and intrinsic of the image of the 2D point x_i, and $P(\cdot)$ is the projection operator that projects the 3D scene point back into the image. $d(\cdot)$ refers to the Euclidean distance between the detected 2d image point and the re-projected 3D scene point.

In bundle adjustment, we will minimize the reprojection error with respect to the set of 3D scene points **X** and camera poses **R, t**. Given that bundle adjustment may require longer optimization times, we will perform bundle adjustment only after registering the final image. Additionally, we will only require bundle adjustment on a sparser set of images i.e. mini-temple. From Figure 5, from our initial 3D reconstruction, bundle adjustment refines the edges of our 3D reconstruction.

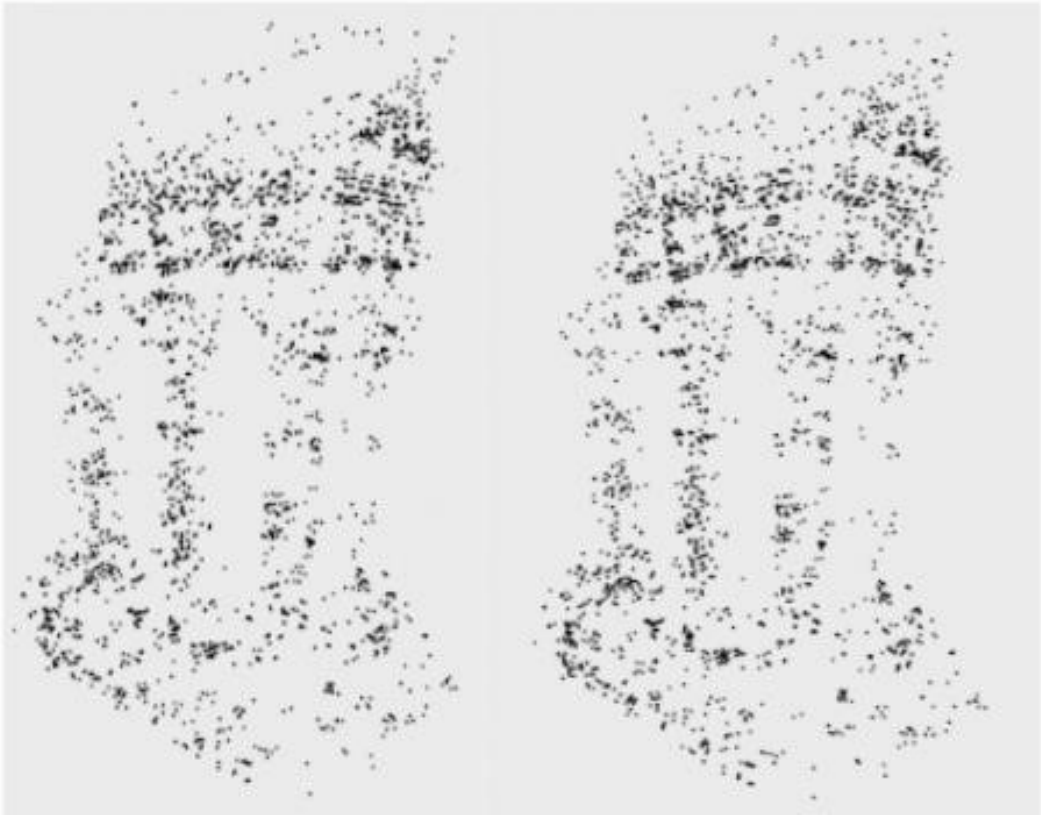

Figure 5: No bundle adjusted (left); bundle adjustment (right).

We can also project the 3D scene points back into the image to visualize the effects of bundle adjustment, we provide an example in Figure 6 where our 2D points are in green, projected 3D points are in red and correspondences between 2D-3D points are in blue. In Figure 6, we see that our bundle adjusted image has less misaligned points (more green points).

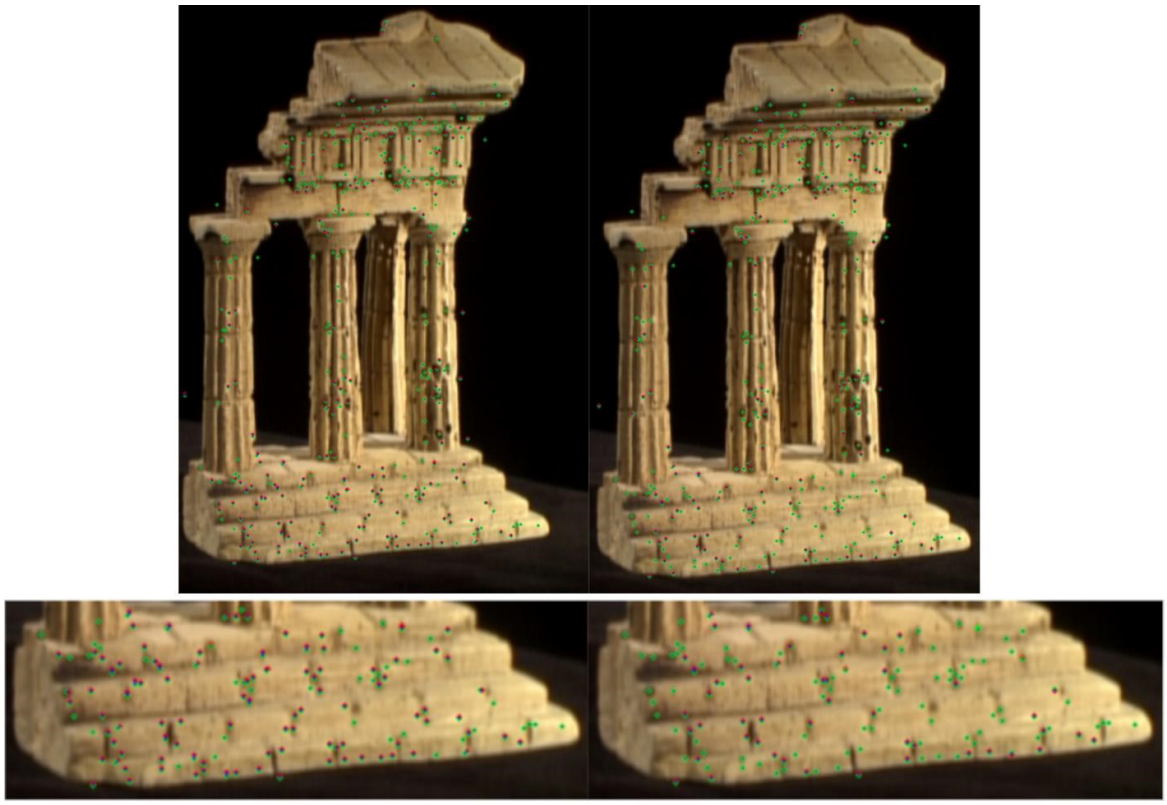

Figure 6: No bundle adjustment (left); Bundle adjustment (right)

7. ASSIGNMENT TASK

Your task in this assignment is to implement the following functions for the algorithm. We will provide the skeleton file with missing code. You are to implement the following:

- preprocess.py [2]:
  - detect_keypoints [0.5]: SIFT keypoint detection
  - create_feature_matches [0.5]: SIFT feature matching + filtering using Lowe ratio.
  - create_ransac_matches [0.5]: RANSAC filtering
  - create_scene_graph [0.5]: scene graph creation
- sfm.py [6]:
  - **Initialization** [1]
    - get_init_image_ids [0.5]: choosing the initial image pair (I_1,I_2)
    - get_init_extrinsics [0.5]: computing the poses of the initial pair from essential matrix.
  - **Incremental SfM** [5]:
    - get_next_pair [0.5]: choosing next image pair (I_{new},I_{reg})
    - solve_pnp [2.5]: RANSAC PnP to solve pose for I_{new}
      - get_reprojection_residuals [2]: returns reprojection error residuals.
      - Miscellaneous code [0.5]
    - add_points3d [2]: add new 3d points from newly registered image
- bundle_adjustment.py [2]:
  - compute_ba_residuals [2]: reprojection residuals with no loops.

Write your code within the block (“”” YOUR CODE HERE “””) and (“”” END YOUR CODE HERE “”””) for each function. Students that do not follow submission instructions will incur a 1-point penalty. Students are strongly encouraged to seek help via the forums.

We also provide the outputs from our implementation in `ta-results/`, which you can compare with using `test.py`. Due to slight implementation differences *etc.*, it is possible that you may not be able to perfectly replicate our outputs. This is fine as long as your implementation achieves the same goal. Having said that, you should be able to closely match our results for:

- preprocess.py
  - bf-match
  - bf-match-images
  - ransac-match
  - ransac-match-images
- sfm.py
  - all-extrinsic.json
  - points3d.npy

You are strongly encouraged to start your implementation early. The README.md will give instructions on how to run the code. You do not need to run bundle adjustment for the temple dataset. The mini-temple dataset is meant for bundle adjustment.