COLMAP & Camera Relocalization

A homework for 3D Computer Vision and Deep Learning Applications Course

2025-Oct

HW2

By: Mohammadreza Kamrani – ID: D13949003

Contents

1. Introduction

The field of computer vision seeks to enable machines to perceive and understand the visual world. A fundamental aspect of this understanding is inferring 3D structure from 2D images and determining the precise position and orientation (pose) from which these images were captured. This homework provides practical experience in two deeply interconnected areas central to 3D computer vision: Structure-from-Motion (SfM) and Camera Relocalization.

1.1 Structure-from-Motion (SfM) with COLMAP

Structure-from-Motion is a photogrammetric technique that simultaneously recovers the three-dimensional geometry of a scene (the "Structure") and the camera positions (the "Motion") from a collection of two-dimensional images. The process typically involves several key steps:

Feature Detection and Description: Identifying keypoints (corners, blobs and …) in each image and describing their local appearance using descriptors (SIFT, ORB, …).

Feature Matching: Finding correspondences between keypoints in different images that project to the same 3D point in the world.

Sparse Reconstruction: Estimating initial camera poses and a sparse 3D point cloud through triangulation.

Bundle Adjustment: An optimization step that refines the 3D point positions and camera parameters to minimize the overall reprojection error, resulting in a consistent and accurate reconstruction.

In this assignment, we utilize COLMAP, a state-of-the-art, open-source SfM. COLMAP automates the complex workflow of SfM, allowing us to reconstruct a 3D model of a scene on a video.

I the first part we reconstruct 3D model form a recorded video related to the Academia Sinica campus. In the second part we use provided 3D model for the video frames of the NTU entrance to perform camera relocalization and then add a virtual object (a cube) to video.

We used Google Colab and google drive for programming and data storage and some commands takes a noticeable time so sometimes the sessions restarted and we continued with the last saved results.

2. Q1 – 3D Model

For this part we recorded a short video from the entrance of Academia Sinica. This video is in 36 seconds with fps=30 and res=1280*720.

Figure 1 – One sample frame of video

Then we make Feature Extraction to find all keypoints and their features in all frames. The result is a **"database.db"** file which contains the keypoints, scale, orientation and a 128-dimentions vector for each keypoints in each image.

The next step is finding match points in different images to get ready for 3D points cloud generation.

If we want to find math points between all images, it generates more accurate and dense points for 3D points cloud and more accurate positions of cameras but it needs a huge amount of time and processing budget. So, we decided to compare 5 sequential frames and it will lead to a sparse points cloud but the concept for both results is the same.

When we have two projected points from a unique 3D point in real world we can estimate the point in the 3D dimensions. This method is called Triangulation. A 2D coordination X_2 is a null vector of the other match point X_1. So, we have $X_1 E X_2 = 0$ where E is essential matrix that is equal to [t]xR and R and t can be calculated by decomposing of E. The result of manipulation of $X_1 E X_2 = 0$ is a 9 dimensions vector which includes E elements and when we want to use SVD we need at least 5 points. The vectored E matrix is the V vector of SVD related to the least eigen value. After reshaping E to standard 3x3 matrix and decomposing t and R we will have 4 answers that by chirality test we can reach to the correct t and R that generate 3D points in front of camera. Actually, we can use more points to eliminate noise.

The projection matrix H is the combination of [R t] and by its inverse matrix we can calculate 3D common point. Moreover, the relative position and rotation of two camera is clear now and if we suppose a camera as a reference, we know the other camera position and angles related to the other camera.

For viewing of 3D point we couldn't use Google Colab because open3d library cannot make a graph window in Jupyter environment. So, the final part is converting this 3D points cloud to a PLY model that can be imported to the software for 3D view of points cloud and converting point cloud to mesh view. For view we use COLMAP GUI software and Meshlab software for generating mesh view from points cloud.

Our points cloud is not so dense so mesh results is not completely satisfying but if we increase sequence number for match finding we may generate more 3D points but it needs more time and process power. In the Figure 2 we see output result of Meshlab after cleaning outliers and cast away points.

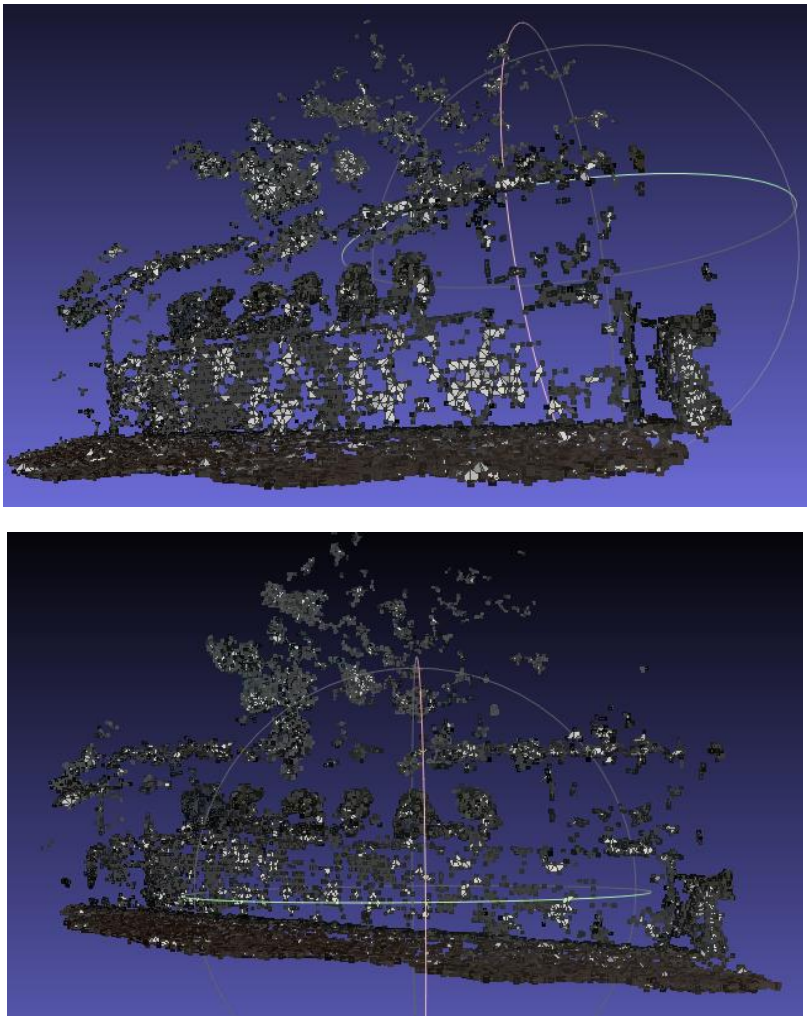

Figure 2 – Mesh views of model in Meshlab software

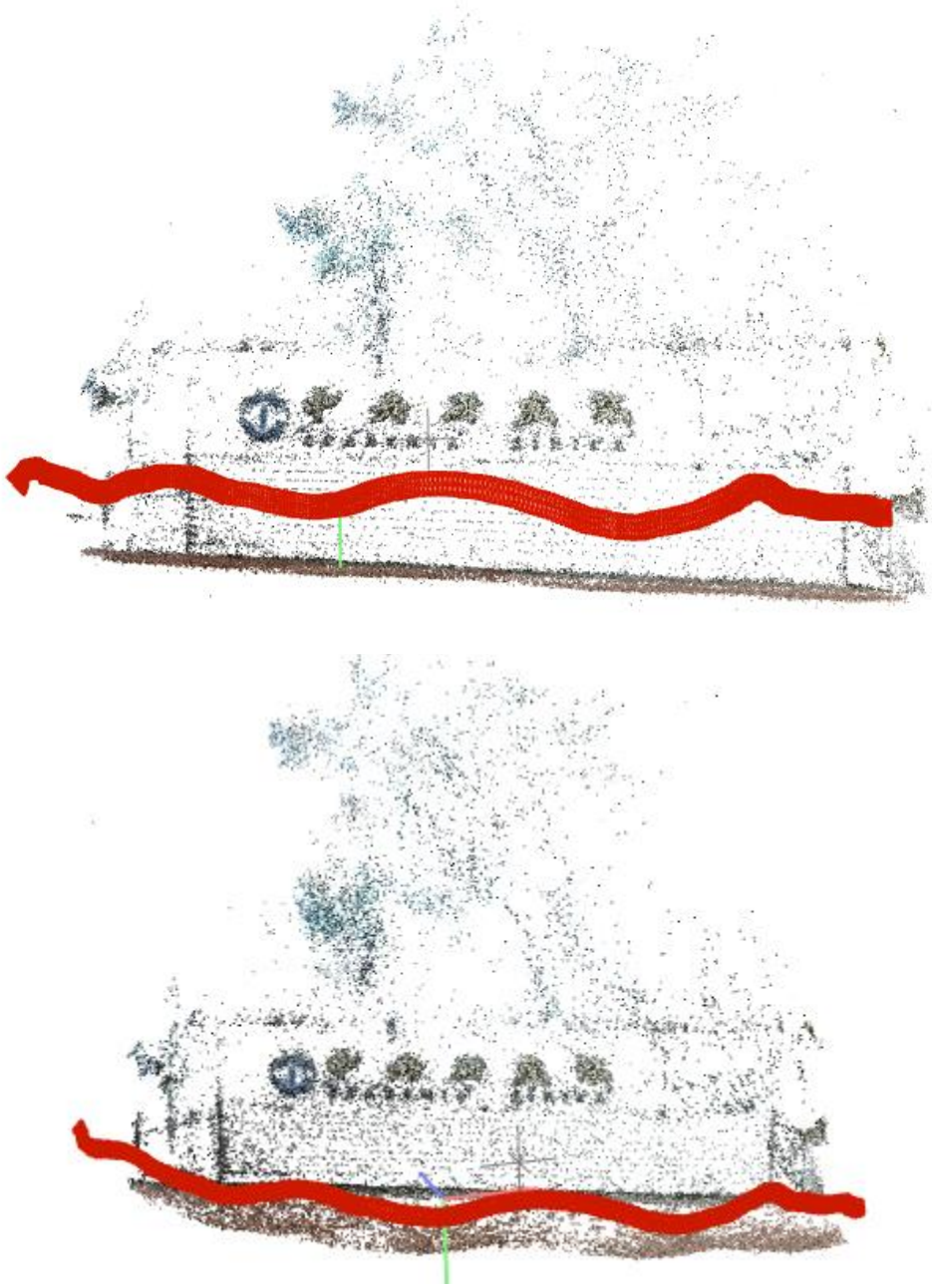

Figure 3 – 3D views of points cloud in COLMAP GUI software

In the recorded video, some notebook cells executes faster because some tables and variables have already calculated. Some functions like mapper in COLMAP takes a lot of time so we paused the recording to decrease the video size.

3. Q 2: Camera Localization

In this part we want to extract camera positions of camera in frames of a video.

The frames are related to the NTU entrance and we have 293 frames which 163 frames are for training and 130 frames are for validation of our estimation. The goal is estimation of location and rotation of the camera in each frame and finally with this information we add a virtual object in each frame and make a new AR video.

The procedure for this question is like this:

1- Select an image from validation dataset
2- Find match points in this image with images in the training set.
3- Remove the wrong matches with Lowe Ratio Test or RANSAC algorithm
4- Find correspondence 3D point of a match point from train table
5- Use 3 points from the precious part to solve P3P problem or PnP problem for more than 3 points.
6- With R and t output of the precious part and given K we can find the location and rotation of the camera.

First, we used cv2.solvePnP instead of cv2.solvePnPRansac to check if the result accuracy is enough or not. Because cv2.solvePnPRansac is much slower than cv2.solvePnP alforithm.

With the results and the ground truth in the image.pkl table we calculate median error of our estimations.

Moreover, we tried to implement P3P and RANSAC algorithm from scratch and compare the results. First, we tried P3P implementation by p3p_solver function and then implemented RANSAC concept by choosing 3 random match points and generate R and t and select the best R and t with minimum error during several iterations.

With RANSAC optimization the results got better but still it was not like OpenCV implementation for P3P/PnP even when we increased the iteration to 1000. We think that OpenCV also use some heuristic algorithms for increasing the precision.

By OpenCV.solvePnP method we got very good results and the Median error for the validation images was as following:

Median translation error: 0.0000

Median rotation error: 0.0001 degrees

So, we didn't need to add RANSAC for better results.

4. Conclusion and Note on Implementation

In conclusion, this homework gave us a better understanding of 3D reconstruction and camera relocalization techniques. Implementing COLMAP helped us see how sparse point clouds and feature matching can be used to recover camera poses, while the camera relocalization exercises, including P3P and RANSAC, provided hands-on experience in estimating precise camera positions from 2D-3D correspondences. Comparing our custom implementation with OpenCV's solvePnP also highlighted the challenges in robust pose estimation and the importance of selecting

accurate correspondences. Overall, this assignment strengthened our grasp of geometric computer vision concepts and their practical applications in structure-from-motion pipelines.

For the implementation of this assignment, we used **ChatGPT** to accelerate coding and debugging, but all underlying concepts were fully understood, and the language model served only to speed up development.