3DCV Homework 2

R13522804 張祐誠

Problem 1 — Structure-from-Motion and Mesh Reconstruction

1.1 COLMAP Pipeline

The goal of this step was to reconstruct the 3D scene from a sequence of images by estimating camera poses and triangulating 3D points. I used COLMAP's Structure-from-Motion (SfM) pipeline consisting of feature extraction, feature matching, sparse reconstruction, and image undistortion.

Step 1: Extracted frames from the video at 4 fps to create roughly 200 images for reconstruction.

ffmpeg -i IMG_0810.mp4 -vf fps=4 images/frame_%04d.jpg

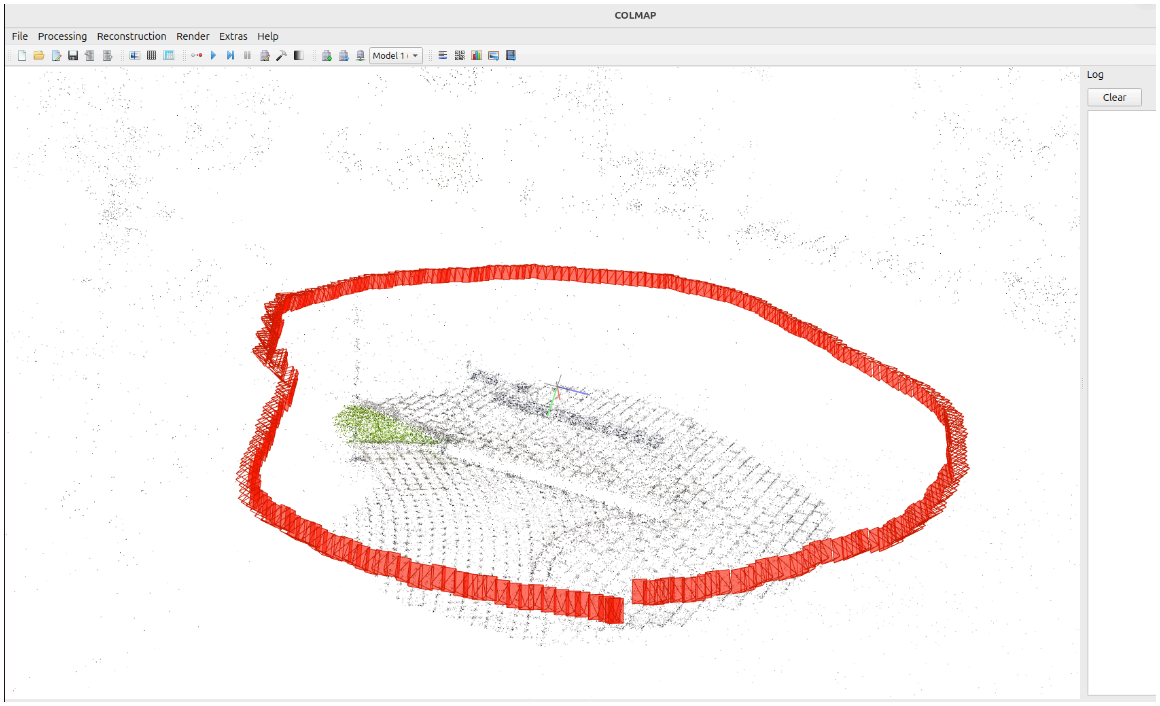
Step 2: Feature Extraction

colmap feature_extractor --database_path database.db --image_path ./images

Step 3: Feature Matching
colmap sequential_matcher --database_path database.db

Step 4: Sparse Reconstruction (Mapping)
colmap mapper --database_path database.db --output_path sparse/

Step 5: Image Undistortion (for dense model input)
colmap image_undistorter --image_path images --input_path sparse/0 --output_path dense/



1.2 Mesh Construction

1. Input: COLMAP output (points3D.bin) or an existing .ply point cloud

2. Preprocessing: voxel downsampling + normal estimation

3. Mesh reconstruction:

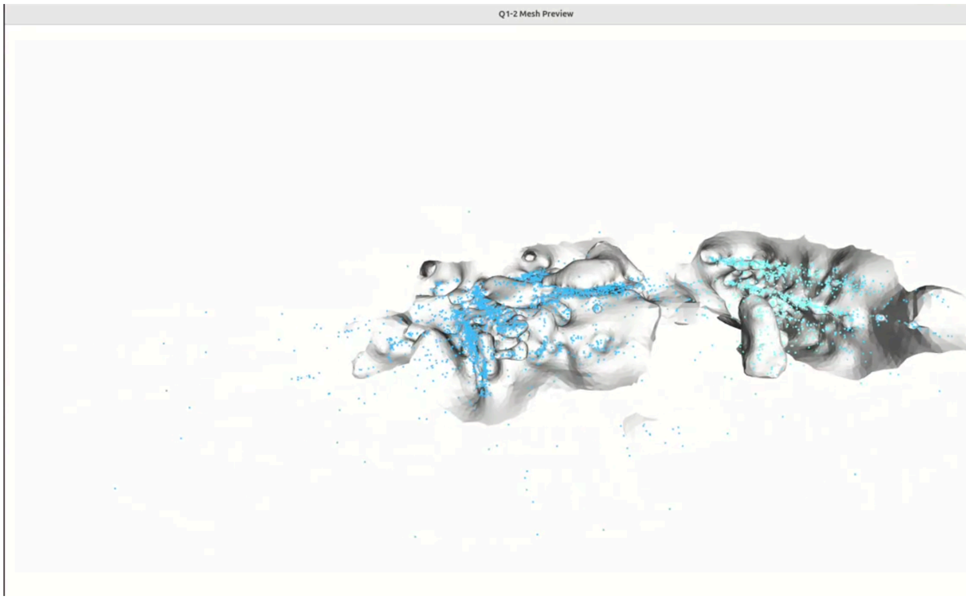
- --method poisson → smoother, watertight mesh
- --method bpa → preserves edges, more raw look

4. Post-processing: trimming low-density regions, smoothing, simplification

5. Output: final mesh file and Open3D visualization window

Execution command:

python mesh_model.py --colmap_model_dir sparse/0 --method poisson --poisson_depth 10 --density_trim 0.03 --target_tris 200000 --out mesh_poisson_clean.ply



Video link: https://youtu.be/8v_artxrPxc

Problem 2 — Camera Relocalization and AR Rendering

2.1 Pose Estimation (Q2-1 Step 1)

Explain the two solvers: OpenCV baseline (PnP) and P3P / P3P-Refine methods. Provide pseudo-code and algorithm descriptions.

Pseudo-code:

[OpenCV Baseline (BFMatcher + solvePnPRansac + iterative refine)]

```
Input: kp_query (Nx2), desc_query (NxD),
    kp_model (Mx3), desc_model (MxD),
    intrinsics K, distortion D
1) Matches ← BFMatcher.L2.knnMatch(desc_query, desc_model, k=2)
2) Good ← { m | m.distance < 0.75 * n.distance }     # Lowe ratio
3) If |Good| < 6 → return failure
4) pts2d ← [ kp_query[m.queryIdx] for m in Good ]
  pts3d ← [ kp_model[m.trainIdx] for m in Good ]
5) (ok, rvec, tvec, inliers) ← solvePnPRansac(
    pts3d, pts2d, K, D,
    method=EPNP, reprojErr=6.0, iters=300, conf=0.999)
6) If not ok or |inliers| < 6 → return failure
7) Use only inliers:
  in2d ← pts2d[inliers], in3d ← pts3d[inliers]
  (ok, rvec, tvec) ← solvePnP(
    in3d, in2d, K, D, rvec, tvec,
    useExtrinsicGuess=True, method=ITERATIVE)
8) Return (ok, rvec, tvec, inliers)
```

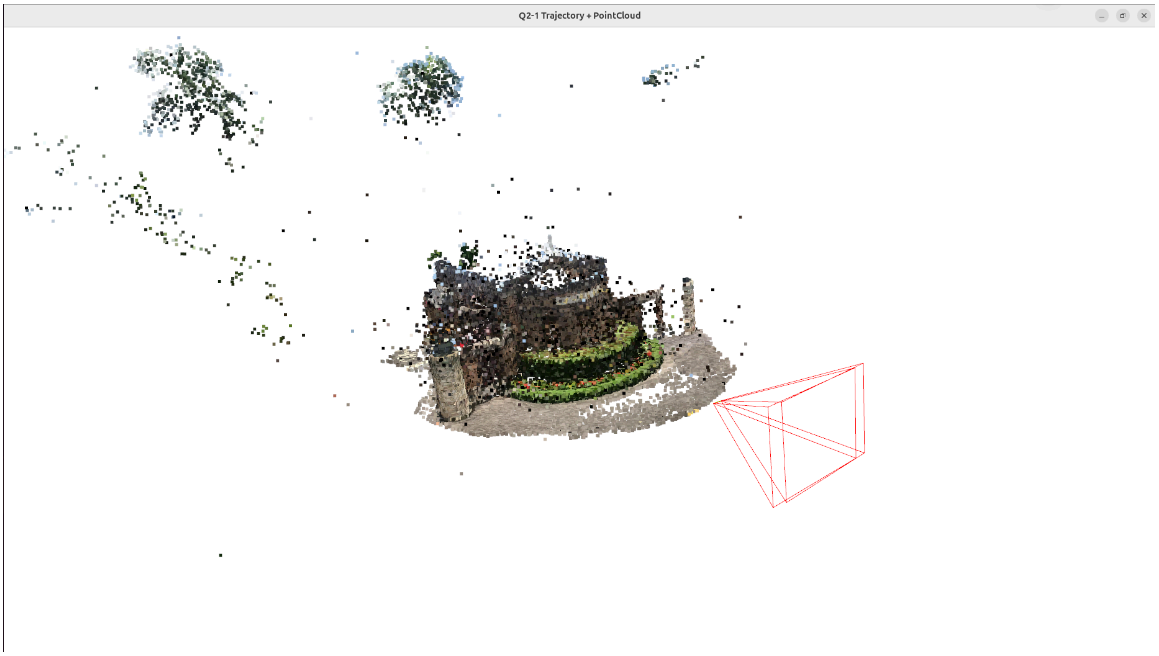
```
for each RANSAC iteration:
  pick 3 matches
  solve P3P numerically
  project points → count inliers
keep best (R,t) and refine using all inliers
```

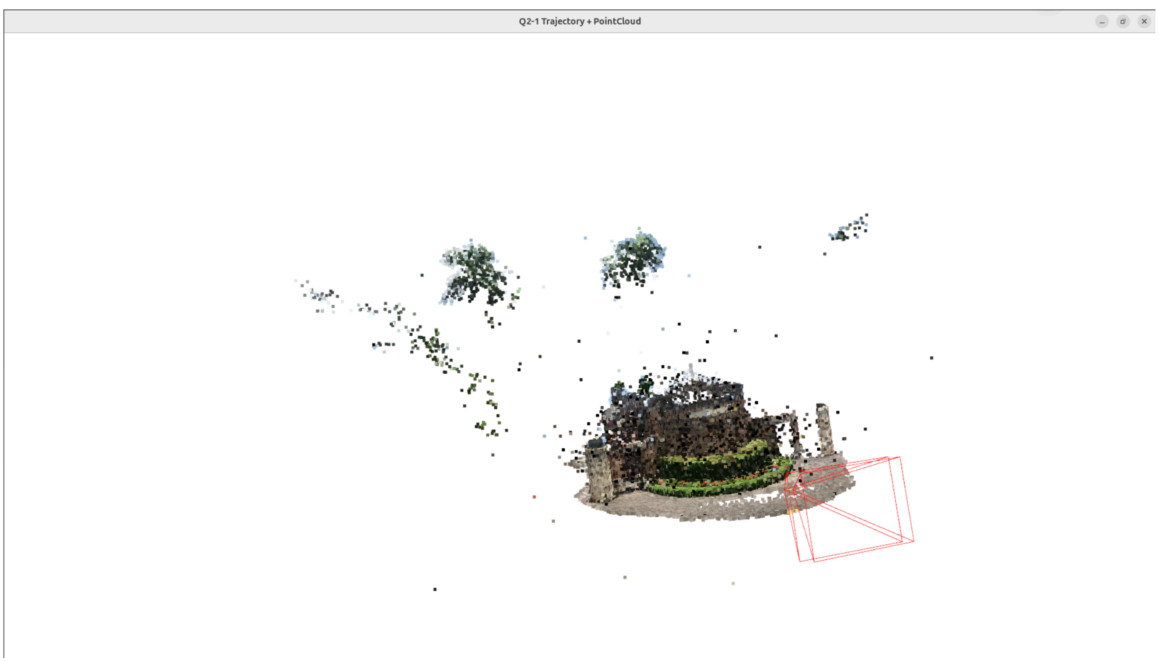
[P3P + RANSAC]

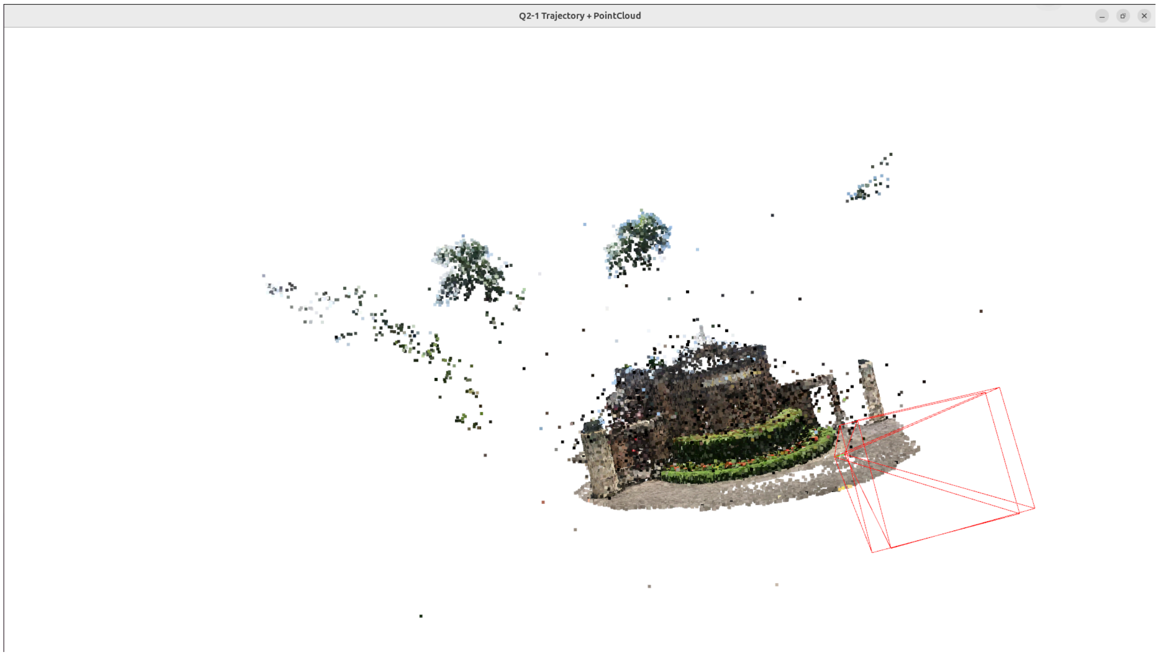
```
Input: kp_query, desc_query, kp_model, desc_model, K

1) Matches ← BFMatcher.L2.knnMatch(desc_query, desc_model, k=2)

2) Good ← ratio test 0.75; if |Good| < 6 → failure

3) uv  ← [ kp_query[m.queryIdx] ]      # 2D pixels

  Pw  ← [ kp_model[m.trainIdx] ]      # 3D points

  N   ← len(Pw)

4) bestInl ← None; bestModel ← None

5) trials ← 0; maxTrials ← 2000

6) while trials < maxTrials:

   a) idx ← random 3 unique indices

   b) For each (R,t) in P3P_NUMERIC(Pw[idx], uv[idx], K):

      i)  uv_hat, z ← project_pixels(K, R, t, Pw)

      ii) err ← ||uv_hat - uv|| per point

      iii) inl ← { i | err[i] < 4.0  and  z[i] > 0 }

      iv) If |inl| > |bestInl|:

           bestInl ← inl; bestModel ← (R,t)

           # update adaptive RANSAC maxTrials

           w ← |inl|/N; eps ← 1 - w^3

           maxTrials ← min(2000, log(1-0.999)/log(eps) + 1)

   c) trials ← trials + 1

7) If bestModel is None or |bestInl| < 3 → failure
```

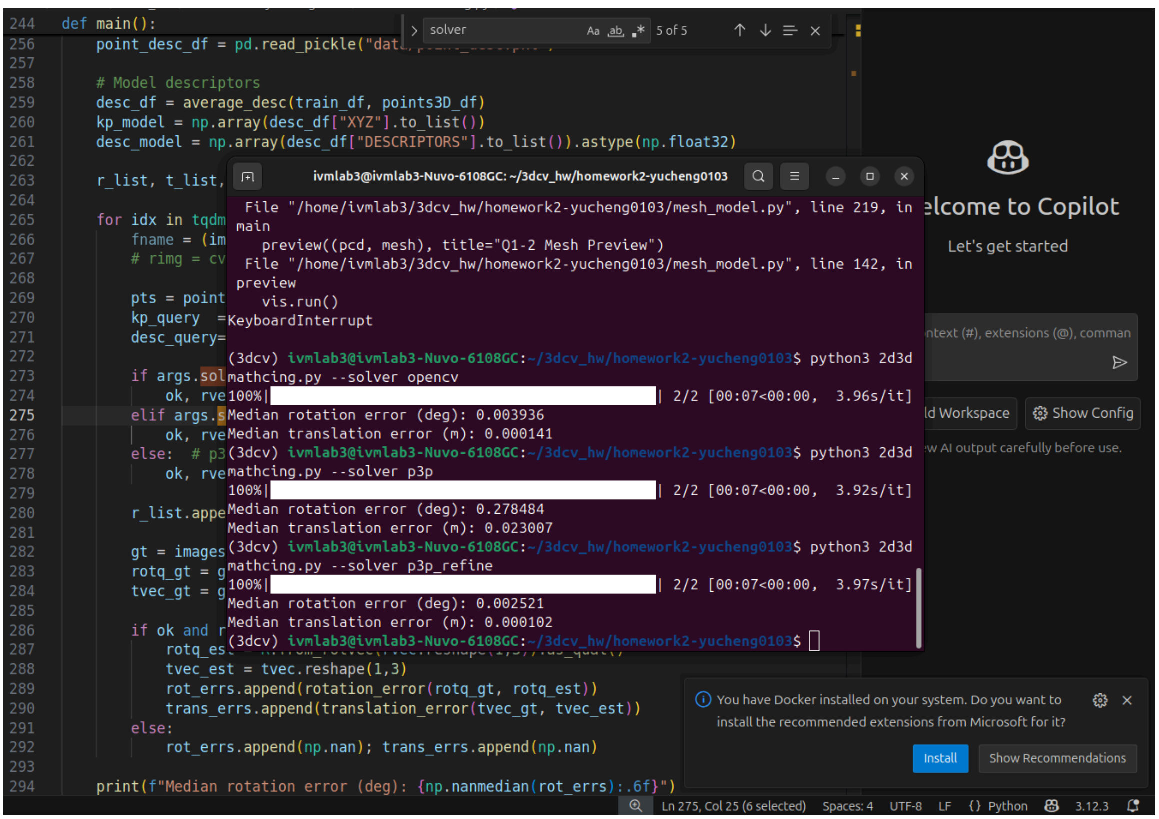
8) rvec ⟵ Rodrigues(bestModel.R); tvec ⟵ bestModel.t

9) Return (True, rvec, tvec, bestInl)






Q2-1 Trajectory + PointCloud

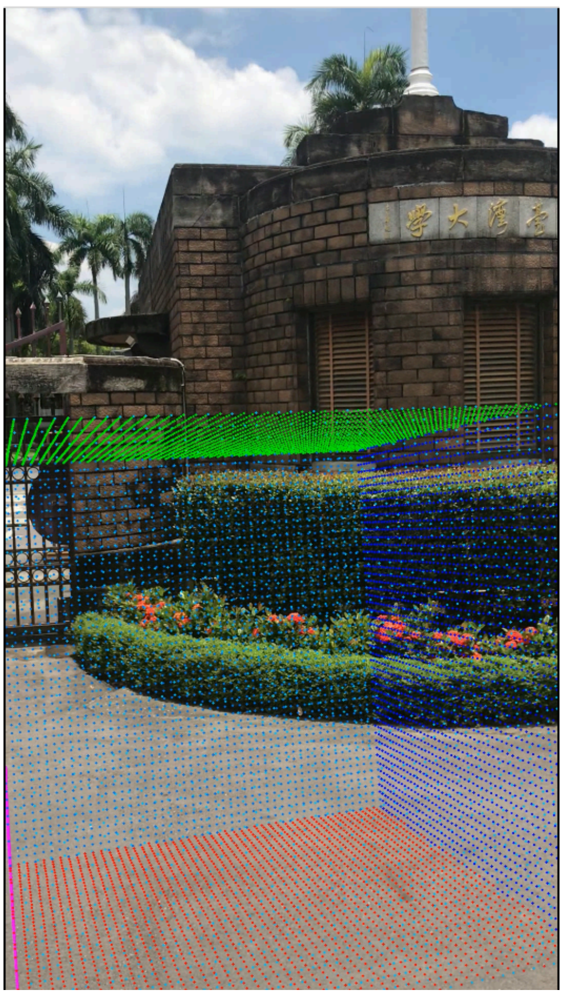

def main():
point_desc_df = pd.read_pickle("data/
# Model descriptors
desc_df = average_desc(train_df, points3D_df)
kp_model = np.array(desc_df["XYZ"].to_list())
desc_model = np.array(desc_df["DESCRIPTORS"].to_list()).astype(np.float32)
r_list, t_list,
for idx in tqdm
fname = (im
# rimg = cv
pts = point
kp_query =
desc_query=
if args.sol
ok, rve
elif args.s
ok, rve
else: # p3
ok, rve
r_list.appe
gt = images
rotq_gt = g
tvec_gt = g
if ok and r
rotq_es
tvec_est = tvec.reshape(1,3)
rot_errs.append(rotation_error(rotq_gt, rotq_est))
trans_errs.append(translation_error(tvec_gt, tvec_est))
else:
rot_errs.append(np.nan); trans_errs.append(np.nan)
print(f"Median rotation error (deg): {np.nanmedian(rot_errs):.6f}")
solver
5 of 5
ivmlab3@ivmlab3-Nuvo-6108GC: ~/3dcv_hw/homework2-yucheng0103
File "/home/ivmlab3/3dcv_hw/homework2-yucheng0103/mesh_model.py", line 219, in main
preview((pcd, mesh), title="Q1-2 Mesh Preview")
File "/home/ivmlab3/3dcv_hw/homework2-yucheng0103/mesh_model.py", line 142, in preview
vis.run()
KeyboardInterrupt
(3dcv) ivmlab3@ivmlab3-Nuvo-6108GC:~/3dcv_hw/homework2-yucheng0103$ python3 2d3d mathcing.py --solver opencv
100%| | 2/2 [00:07<00:00, 3.96s/it]
Median rotation error (deg): 0.003936
Median translation error (m): 0.000141
(3dcv) ivmlab3@ivmlab3-Nuvo-6108GC:~/3dcv_hw/homework2-yucheng0103$ python3 2d3d mathcing.py --solver p3p
100%| | 2/2 [00:07<00:00, 3.92s/it]
Median rotation error (deg): 0.278484
Median translation error (m): 0.023007
(3dcv) ivmlab3@ivmlab3-Nuvo-6108GC:~/3dcv_hw/homework2-yucheng0103$ python3 2d3d mathcing.py --solver p3p_refine
100%| | 2/2 [00:07<00:00, 3.97s/it]
Median rotation error (deg): 0.002521
Median translation error (m): 0.000102
(3dcv) ivmlab3@ivmlab3-Nuvo-6108GC:~/3dcv_hw/homework2-yucheng0103$
elcome to Copilot
Let's get started
ntext (#), extensions (@), comman
ld Workspace
Show Config
w AI output carefully before use.
You have Docker installed on your system. Do you want to install the recommended extensions from Microsoft for it?
Install
Show Recommendations
Ln 275, Col 25 (6 selected)
Spaces: 4
UTF-8
LF
{} Python
3.12.3

2.3 AR Cube Overlay (Q2-2)

The cube is created as a dense set of 3D points sampled on its six faces (unit cube [0,1]^3) with distinct face colors. After closing the window, the script saves:

- `cube_transform_mat.npy` (final transformation matrix)
- `cube_points.npy`, `cube_colors.npy` (transformed cube geometry and colors)

The script reads camera poses from `images.pkl` (rotation quaternions + translation vectors) and camera intrinsics KKK and distortion coefficients. Projected points are drawn as small colored circles on the corresponding image (sorted by depth for correct overlap). Each augmented image is added to a video writer using OpenCV. The final output video (`output.mp4`) shows the cube consistently overlaid on the real scene based on the estimated camera poses.



https://youtu.be/m_fTLb5wMGU

2.4 Discussion

The OpenCV PnP solver achieves more stable and accurate results compared to the hand-implemented P3P + RANSAC method. While the custom implementation can perform well on clean data, it becomes unstable when noise or mismatched features are present, leading to higher rotation and translation errors.

The hand-implemented approach offers greater flexibility and transparency, allowing direct control over parameters and optimization steps. However, it depends heavily on good initialization, is sensitive to noise, and requires manual tuning, which limits its robustness.

In contrast, the OpenCV method is faster, more reliable, and better suited for real-world applications. Its built-in RANSAC efficiently handles outliers and produces consistent results, though it provides less flexibility for customization. Overall, OpenCV's PnP is more practical, while the hand-written version is valuable for understanding the algorithm's internal workings.

Execution Guide

Execution environment:

OS	Ubuntu 24.04 LTS
Python version	3.10 (Anaconda env named `3dcv`)
Major packages	`numpy`, `pandas`, `opencv-python`, `open3d`, `scipy`, `tqdm`
GPU / Driver	NVIDIA RTX 2080 Ti with CUDA support (Open3D used CPU rendering)
Data path	`homework2-yucheng0103/data/` (contains images.pkl, train.pkl, points3D.pkl, point_desc.pkl)

To install dependencies:
conda create -n 3dcv python=3.10

conda activate 3dcv

pip install numpy pandas opencv-python open3d scipy tqdm

Usage:

python3 2d3dmathcing.py --solver opencv

python3 2d3dmathcing.py --solver p3p

python3 2d3dmathcing.py --solver p3p_refine

python3 transform_cube.py

After adjustment and closing the window, the program saves:

cube_transform_mat.npy

cube_points.npy

cube_colors.npy

output.mp4

Acknowledgements & LLM Disclosure

I used ChatGPT (GPT-5) to assist in structuring this report, optimizing the code, bug fixes, and explaining algorithm logic. All experiments and testing were performed by the student.