Navigation Mapping and Localization Algorithms
Homework 2

1. Introduction

Visual Odometry is a technique in robotics where the motion of the camera is estimated using only visual input (images in our case). In this homework, we implemented a **monocular** visual odometry pipeline, meaning all the estimations are done using a single camera with no depth sensors or other helping factors.

2. Problem Definition

1. **Input:** A sequence of monocular RGB images with timestamps.
2. **Output:** An estimated camera trajectory over time, represented as a sequence of camera poses.
3. **Constraints:**
    a. No depth information
    b. No camera intrinsics matrix (K) given.
    c. No IMU or additional sensors
    d. Scale is unknown
    e. Motion is estimated frame to frame
4. **Evaluation:**

    Although not given, we compared to ground truth (after SIM(3) alignment with scale correction) using:

    - **Relative Pose Error (RPE)**
    - **Absolute Pose Error (APE)**

3. System Overview

The visual odometry pipeline follows these main steps:

1) Load image frames
2) Preprocess the images and apply filtering
3) Extract visual features (matching using ORB)
4) Match features between consecutive frames
5) Estimate Epipolar Geometry
6) Recover Relative Camera Pose
7) Validate Pose using Sanity Checks
8) Chain poses using sanity checks
9) Export trajectory for evaluation
10) Apply trajectory **smoothing**

4. Feature Extraction and Preprocessing

4.1 ORB Features

ORB features are used for both keypoint detection and description. ORB was chosen because it's fast, robust, and suitable for real-time systems. It also produces binary descriptors which makes the matching process efficient.

4.2 CLAHE Preprocessing

We observed that the input images contain varying lighting conditions and contrast levels. To address this issue, we evaluated the effect of applying a CLAHE filter before feature extraction.

An evaluation test was performed to measure the number of inliers obtained with and without CLAHE. The result showed the applying CLAHE leads to an approximately 15% increase in inliers across the image sequence.

Other filters

No additional image filters were used beside CLAHE. Similar evaluation tests were conducted for other common filters, such as **Gaussian Blur** and **Bilateral Filtering.**

These tests showed that applying such filters reduced the inlier count by approximately 15%. This behavior is also visually expected, since the input images are already somewhat blurry, and additional smoothing degrades the feature quality even further.

For this reason, no extra preprocessing filtering was applied in the final pipeline.

5. Feature Matching

Feature matching is performed using:

1) K-Nearest Neighbors where k = 2.
2) Lowe's Ratio Test

Only matches that pass the ratio test are kept. Frames with too few matches are rejected early to avoid unstable pose estimation.

We do **not** use symmetric matching or optical flow, to keep the pipeline simple and focused two-view geometry.

6. Epipolar Geometry and Pose Estimation

6.1 Essential Matrix Estimation

The Essential matrix is estimated using **USAC_MAGSAC**, which is robust to outliers and better than other options such as RANSAC. A minimum inlier threshold is enforced to accept the estimation.

6.2 Pose Recovery

The relative camera pose (rotation and translation direction) is recovered using $cv2.recoverPose$.

Since monocular pose recovery can return physically incorrect solutions, we applied sanity checks:

1) **Rotation matrix determinant check**
2) **Finite value check**
3) **Rotation angle limit (flip guard), set to 60 degrees.**

If the pose fails validation, it is rejected.

6.3 Fundamental Matrix Fallback

If the pose recovery from the Essential matrix fail, a Fundamental matrix fallback is used. The Fundamental matrix is estimated and then converted into an Essential matrix using camera intrinsics. This increases the robustness in difficult frames.

7. Pose Changing and Trajectory Construction

Each valid relative pose is converted into a homogeneous transformation matrix and chained using matrix multiplication:

$$T_{world}^{curr} = T_{world}^{prev} \times T_{prev}^{curr}$$

The result is a continuous camera trajectory in an arbitrary scale.

Frames where pose estimation fails are skipped, which avoids inserting incorrect motion into the trajectory.

8. Trajectory Smoothing

To reduce high-frequency noise in the estimated trajectory, we used **Savitzky-Golay smoothing** only to the **translation** components.

An important design decision:

1) Smoothing is **applied only** during post-processing.
2) Raw poses are kept unchanged for evaluation. Two outputs are given.

This allows us to compare raw and smoothed trajectories without affecting the VO pipeline itself.

9. Evaluation Methodology

The estimated trajectory is saved in TUM formatted and evaluated using the **evo** toolkit.

Before evaluation:

1) SIM(3) alignment is applied
2) Scale correction is enabled

The following metrics are reported:

1) RPE (Rotation and Translation)
2) APE (Translation)

Results are reported for:

1) Raw Trajectory
2) Smoothed Trajectory
3) With and without pose safeguards.

Doing these different tests allowed us to evaluate our pipeline better. For instance, when doing the Rotation test for degrees, we found the we have a very high error. Thus, we applied the safeguard.

Although we weren't given ground truth, we found ground truth for the dataset and applied our tests. We also tested our pipeline on **different** datasets and compared to their ground truths.

10. Results and Discussion

Applying the rotation flip safeguard significantly reduced larger rotation errors and eliminated catastrophic pose flips. Trajectory smoothing slightly improved translation metrics and improved the visual consistency by a lot.

Rotation metrics were mostly unaffected by smoothing, as expected, since rotations were **not** modified.

Overall, the pipeline produces stable and reasonable results given that we are working with a monocular VO system without any extra sensor information or bundle adjustments.

11. Conclusion

In this homework, we implemented a complete monocular visual odometry pipeline using classical computer vision techniques. Using careful pose validation, fallback strategies, preprocessing and post-processing, we achieved good stable trajectories within a reasonable accuracy despite the limitations of the monocular vision and the given inputs.