Lab 4: Stereo Visual Odometry

This repository contains the code for Lab 4 of AER1217: Development of Unmanned Aerial Vehicles. The objective of this lab is to implement a stereo camera-based visual odometry (VO) system to estimate a vehicle's pose using images from the KITTI dataset.

Overview

The provided code already includes stereo image preprocessing, feature detection, and feature correspondence matching. So the main tasks for this lab are:

- Outlier rejection using the RANSAC algorithm.
- Pose estimation through scalar-weighted point cloud alignment.

Our Implementation for the `pose_estimation()` function

The only function we need to modify is `pose_estimation()` in `stereo_vo_base.py`. This function takes as input the filtered set of feature correspondences across four stereo frames and computes the relative transformation (rotation and translation) between the current and previous frame.

Here is a summary of `pose_estimation()`:

1. **3D Triangulation**: The function uses the stereo camera model to convert 2D image coordinates into 3D points in both the previous and current frames.

2. **RANSAC for Outlier Rejection**:

    - Randomly selects three pairs of 3D corresponding points.
    - Computes the rigid body transformation (rotation `R` and translation `t`) using SVD.
    - Applies the transformation to all current points and calculates the residuals compared to previous points.
    - Points with residuals under a fixed threshold are considered inliers.
    - Repeats this process multiple times and selects the transformation with the most inliers.

3. **Final Pose Estimation**:

    - With the best set of inliers, re-computes the transformation using a scalar-weighted point cloud alignment:
        - Computes the centroids of both point clouds.
        - Constructs the cross-covariance matrix `W`.
        - Applies SVD on `W` to compute the final rotation matrix.
        - Derives the translation vector from the rotation and centroids.

4. **Output**: The function returns the final rotation matrix `C`, translation vector `r`, and the inlier feature correspondences in the right image.

Notes

- This implementation assumes the KITTI dataset has been preprocessed and images are rectified.

- Keypoints are detected using SIFT and matched using brute-force matching.