ETH zürich

A Monocular Visual Odometry Pipeline in Python

Ferranti Alessandro, Jensen David, Pirini Alessandro, Rubini Matteo

ROBOTICS & PERCEPTION GROUP

February 20, 2026

Abstract

Visual Odometry (VO) is the process of estimating the position and orientation of an agent by analyzing the changes that motion induces in a sequence of camera images. This report presents a monocular VO pipeline implemented in Python, featuring 3D landmark initialization, keypoint tracking, and continuous pose estimation. The system has been evaluated on three public datasets: KITTI, Malaga, and parking, as well as a custom-recorded sequence.

Contents

1 Introduction

The objective of this project is to develop a robust monocular Visual Odometry (VO) pipeline capable of estimating the pose of a camera through various environments. Monocular VO presents the challenge of scale ambiguity, where the trajectory and scene geometry can only be recovered up to a relative scale factor, something that must always be remembered.
The implementation can be separated into four main parts:

- Initial 3D landmark bootstrapping
- Keypoint tracking across successive frames
- Pose estimation through 2D-3D correspondences
- Triangulation of new landmarks to maintain continuous operation

The pipeline was validated on the KITTI, Malaga, and parking datasets, focusing on achieving local consistency and minimizing drift (Qualitative Evaluation-Results). The system was also tested on a custom-recorded dataset to demonstrate its flexibility across different hardware (since it was recorded with a phone) and lighting conditions. The complete source code, including detailed setup instructions and dependency requirements, is available in the attached repository. The implementation provided is largely based on functions provided by `the OpenCV` library [1].

2 Theoretical Approach

The pipeline adheres to a Markovian process framework in which the state of the system S^i in frame i contains all the information necessary to process the subsequent frame I^i, without requiring a global history. The camera pose is defined as $T^i_{WC} \in SE(3)$, representing the transformation from the camera to the world frame.
The state is defined as the tuple $S^i = \{P^i, X^i, C^i, F^i, \mathcal{T}^i\}$, where:

- $P^i \in \mathbb{R}^{2\times K}$: The set of K 2D keypoints currently being tracked.
- $X^i \in \mathbb{R}^{3\times K}$: 3D world coordinates of landmarks corresponding to P^i.
- $C^i \in \mathbb{R}^{2\times M}$: Candidate keypoints without associated 3D landmarks.
- $F^i \in \mathbb{R}^{2\times M}$: The 2D coordinates of the first observation for each candidate in C^i.
- $\mathcal{T}^i \in \mathbb{R}^{12\times M}$: Reshaped $SE(3)$ camera poses (stored as vectors) corresponding to the first observation of each candidate.

The state transition is governed by the function $[S^i, T^i_{WC}] = \text{processFrame}(I^i, I^{i-1}, S^{i-1})$, which estimates the pose and updates the state with active keypoints and landmarks. New landmarks are promoted from the candidate set C^i to the active set P^i only when the bearing angle α between the current view and the initial view $\tau \in \mathcal{T}$ provides sufficient baseline for triangulation.

3 Pipeline structure

On a high level, there are two main building blocks that make up the pipeline:

1. **Initialization**: This component is run just once to prepare the pipeline for continuous operation. Two sufficiently distant (heuristic) frames are selected, features are extracted using the Shi-Tomasi corner detection algorithm (3.1), KLT is used to establish keypoint correspondences (3.2), and the 8-Point RANSAC Algorithm implemented by is used to find the homography between the two selected frames (3.3). With the inlier keypoint correspondences from RANSAC and the associated homography, an initial 3D point cloud can be triangulated (3.4), which is used to initialize the state (3.5).
2. **Continuous Operation**: This is the core process of the pipeline, and it is run continuously after initialization. First, current and candidate keypoints are tracked forward one frame (3.6); then, the current camera pose is estimated using PnP (3.7); after this, another triangulation to add new landmarks from the current candidates set is attempted (3.8); finally, new features are extracted (3.9) and added if they are not duplicates (3.10).To evaluate performance during operation, plots are updated real-time during execution (3.11).

3.1 Feature Extraction at Initialization

A Shi-Tomasi corner detector `cv2.goodFeaturesToTrack` is used for initial feature extraction from the first bootstrap key-frame. These features are consequently tracked forward to the second bootstrap key-frame (3.2) and used for localization (3.3). The Shi-Tomasi detector was selected instead of the Harris detector is due to its higher accuracy and relatively small increase in computational cost.
To achieve a more uniform distribution of features detected in the images, the frame was divided into a grid, and features were extracted from each sub-section.

3.2 Initial Tracking

The Kanade-Lucas-Tomasi (KLT) optical flow algorithm implemented by `cv2.calcOpticalFlowPyrLK` is used to track the initial features forward to the next bootstrap key-frame. Only features from the first bootstrap key-frame that are successfully tracked forward are used in the following steps.

3.3 Relative Pose Estimation

Having established reliable 2D-2D correspondences between the two bootstrap key-frames, the relative motion of the camera can be estimated leveraging epipolar geometry. In particular, the fundamental matrix is estimated using RANSAC -`cv2.findFundamentalMat`- so that it is robust to outliers. The outlier 2D-2D keypoint correspondences are discarded.
The essential matrix is then computed with the formula

$$E = K^T F K, \tag{1}$$

where K is the known camera intrinsic calibration matrix. This matrix is given for the three standard datasets and is estimated using a standard calibration process for the custom dataset.
The relative rotation R and translation t can then be extracted from the essential matrix.

3.4 Initial Point Cloud Generation

By letting the pose of the first key-frame be the origin of the world frame, the poses, keypoint correspondences, and camera intrinsics are known for the two bootstrap key-frames. With this information, the 3D coordinates in the world frame remain the only unknown in the perspective projection equation. This is used to build a system of equations, and least squares is used internally to estimate the 3D landmark for each 2D-2D keypoint correspondence, using `cv2.triangulatePoints`.

3.5 Build the State

With the 2D keypoints and 3D landmarks, a dictionary representing the state is initialized according to the form introduced in Theoretical Approach.
The dimensions of the matrices slightly differ from what was outlined in the previous section in order to match the default inputs and outputs of the `openCV` functions used.
Furthermore, if sliding window bundle adjustment feature is being used (4.1), we expand the state so that it includes (in the following win_size is the size of the refinement window we are using):

- An array for unique keypoint-landmark mapping: $ids^i \in \mathbb{R}^{1\times points}$.
- $P\_history^i \in (\mathbb{R}^{1\times 2\times K}, \mathbb{R}^{1\times points})$ is the element (tuple) of a *deque* of length win_size that keeps track of the last win_size-keypoints and their landmarks via the associated id.
- $pose\_history^i \in \mathbb{R}^{3\times 4}$ is the element of a *deque* of length win_size that stores the last win_size-camera poses for BA refinement.

3.6 Tracking Forward

This is the first task in the continuous operation loop and tracks the keypoints (`S["P"]`) from the previous frame to the current frame. The set of keypoints is updated with the 2D locations of the successfully tracked keypoints, and all 3D landmarks (`S["X"]`) that are no longer active are removed.
The current keypoint candidates are tracked forward in the same way, using KLT, and the state (`S[C]`, `S[`$\mathcal{T}$`]`, and `S[F]`, `S[ids]`) is updated accordingly.

3.7 Estimate Pose

The rotation and translation vector **from the world to camera** frame are estimated using `cv2.solvePnPRansac`. We utilize *P3P* with a re-projection error of 5.0, a 99% confidence, and a maximum iteration count of 100, unless otherwise mentioned.
Once the rotation vector is obtained, it must be converted into a rotation matrix used to build a 3×4 camera to world transformation. The current 2D keypoints and corresponding 3D landmarks are updated with the inliers so as to exclude the outliers in subsequent iterations.

3.8 Triangulation of new Landmarks

As the camera moves away from the bootstrapped 3D point cloud, new 3D landmarks must be added to ensure there are sufficient 2D-3D correspondences for pose estimation. Therefore, at each step, new features are extracted and triangulated. A challenge in the latter is finding a good criterion to avoid adding new landmarks when the baseline length is not large enough to guarantee an accurate estimation of the landmark location. This is addressed by computing the bearing angle between camera rays and checking if it exceeds a threshold α_t. For faster operation, all candidates found at the same camera pose T_{CW} are grouped, and triangulation is performed in one shot for each unique pose. In addition, a cheirality check is added from both camera views, so that negative depth points do not corrupt the pose estimation of the camera.

3.9 Feature Extraction

As in the bootstrapping phase, Shi-Tomasi corners are extracted from sections of the image in order to avoid having all the keypoints concentrated in high-uncertainty areas - i.e. in the center of the image - as the error on the estimation grows quadratically with depth. Current and candidate keypoints are masked out so duplicate features are not extracted. In addition, the same number of features are extracted from each subsection of the frame to avoid a dense group of features dominating pose estimates. A key change that improved the performance of the algorithm - mainly in the *Malaga* dataset - was to set an absolute threshold on the minimum eigenvalue intensity when finding the candidate keypoints. A map of minimum eigenvalues for the image is extracted, and then for each region of interest, the maximum is compared to an absolute threshold, and if it is less, the region is skipped. This filtering step rejected most of the candidates that were consistently found in sky regions in the image. This was fundamental for the correct operation of the algorithm, because even with RANSAC, some of the points in the sky were still selected as inliers, injecting a large uncertainty in the pose estimation step.

3.10 Adding New Features

Once features have been extracted from the current frame, they are compared with the current and candidate keypoint locations. Only features with a squared distance from the nearest existing keypoint greater than α_{new} pixels are added to avoid adding duplicate keypoints. Then each unique feature and the current camera pose are added to the list of candidates $S[C]$ and the list of first observations $S[\mathcal{T}]$.

3.11 Plotting

The following are plotted continuously as frames are processed:

1. Image frame
   - Candidate keypoints are tracked in **red** with the corresponding tracking arrows from frame I_{i-1} to frame I_i
   - Inliers for PnP RANSAC in **green** with the corresponding tracking arrows from frame I_{i-1} to frame I_i
2. Number of keypoints being tracked and PnP inliers displayed on a graph that varies over time
3. The pose in the last 20 frames, with the triangulated landmarks X in the x-z plane
4. The global trajectory plotted in the x-z plane

4 Additional Features

In addition to the basic pipeline, two other features were implemented:

- **Sliding Window Bundle Adjustment**(4.1): mitigates drift during operation.
- **Scale estimate from ground points**(4.2): attempts to immediately estimate the absolute scale of the data set by computing the estimated camera height.

4.1 Sliding Window Bundle Adjustment

To compensate for the accumulation of drift inherent in monocular Visual Odometry, a **Sliding Window Bundle Adjustment (BA)** local optimization function was implemented. BA jointly refines both the camera poses and the 3D landmark positions over a fixed-size temporal window to avoid adding too much computational cost.

The primary aim of this feature is to ensure **local multi-view consistency**. Reconsidering the observations of the same landmarks from multiple previous frames stored in the `pose_history` and `P_history` deques, the system achieves the following:

- **Drift Reduction**: Correcting small errors in pose estimation that would otherwise accumulate quadratically over the trajectory.

- **Geometry Refinement**: Improving the 3D coordinates of the landmarks in X^i as more viewpoints become available, providing a more stable reference for subsequent P3P iterations.

The function minimizes the **Total Reprojection Error** within a window of size N (specified by `window_size` in our `VO_Params`). The cost function J is defined as:

$$J(\theta, X) = \sum_{i=1}^{N} \sum_{j \in \mathcal{V}_i} \rho\left(p_{i,j} - \pi(K, T_i, c_j)\right) \tag{2}$$

Where:

- T_i and c_j are the camera poses and 3D landmarks to be optimized.

- $\pi(\cdot)$ is the perspective projection function that maps a 3D point to 2D pixel coordinates using intrinsics K.

- $p_{i,j}$ is the actual KLT-tracked observation of landmark j in the frame i.

- $\rho(\cdot)$ represents the **Huber Loss** function, utilized via `scipy.optimize.least_squares`[2] to make optimization more robust.

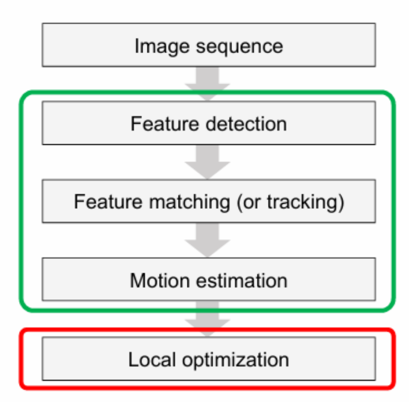


Figure 1: The VO pipeline scheme featuring local optimization via Bundle Adjustment.

For a more efficient algorithm, a **Jacobian sparsity mask** generated by the helper function `get_jac_sparsity()` is used, according to what is suggested in the Scipy Cookbook [3]. After refinement, the current pose and the set of landmarks X^i are updated in the state S^i with optimized values, so that the motion of the camera can be better estimated. With this feature implemented, the last piece (**local optimization**) of the VO pipeline scheme shown here *fig:1* is added, and, in general, it improves the output trajectory, if compared to the ground truth.
Unless otherwise stated, the window length of refinement is considered to be 10.

4.2 Scale estimate from ground points

This feature tries to estimate the absolute scale of the dataset at bootstrap time; two fundamental assumptions have been made:

- The dataset was recorded through a camera in a car, and thus **on a road**.

- The camera is at a height of 1.65 [m] from the ground, which corresponds to what KITTI used [4] and is close to the average car height, thus it is a good estimate for other datasets as well.

The algorithm tries to estimate the correct absolute scale of the dataset by computing the ratio between the assumed camera height and the estimated one from the model of a plane fit from initially triangulated points. The ground plane is estimated as follows:

Algorithm 1 Absolute Scale Estimation via Ground Plane Fitting

```
Require: Initial monocular image sequence, expected ground normal $\mathbf{n}_{\text{exp}}$, thresholds $\tau$, $\tau_{\text{angle}}$
Ensure: Absolute scale factor for monocular VO
1: Extract dense and potentially noisy visual features in the three central-lower regions of the image to
   obtain an initial road hypothesis.
2: Track the extracted features across the first frames to obtain sufficient parallax.
3: Triangulate tracked features to generate a 3D point cloud.
4: Estimate the ground plane using RANSAC:
     • Randomly sample three points $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$.
     • Compute the plane normal
         $$\mathbf{n} = \frac{(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)}{\|(\mathbf{p}_2 - \mathbf{p}_1) \times (\mathbf{p}_3 - \mathbf{p}_1)\|}$$
       and offset
         $$d = -\mathbf{n}^\top \mathbf{p}_1.$$
     • Classify a point $\mathbf{x}_i$ as an inlier if
         $$|\mathbf{n}^\top \mathbf{x}_i + d| < \tau.$$
     • Enforce approximate horizontality via
         $$|\mathbf{n}^\top \mathbf{n}_{\text{exp}}| > \tau_{\text{angle}}.$$
5: Select the plane model with the largest number of inliers.
6: Refine the plane parameters by solving a least-squares optimization over all inliers.
7: Compute the camera height as the perpendicular distance from the camera center to the refined plane
   we have just obtained.
8: Compute the absolute scale as the ratio between the known camera height and the estimated one.
9: return Absolute scale factor for the monocular VO pipeline.
```

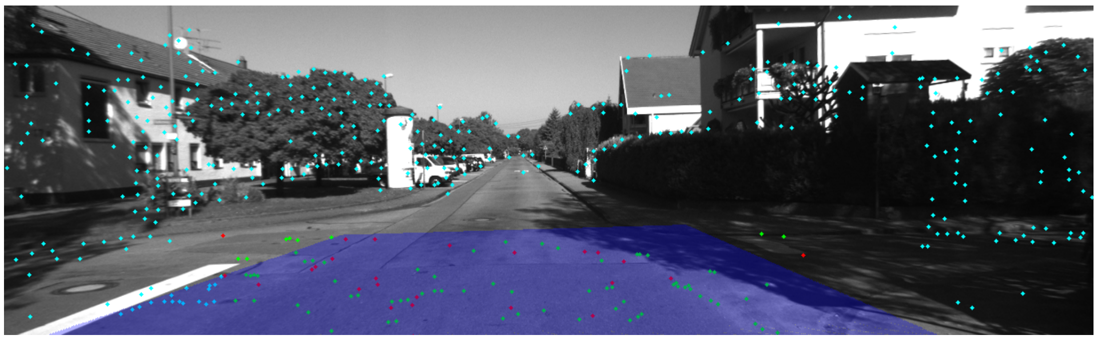


Figure 2: KITTI: keypoints in light blue, ground inliers in green, outliers in red, plane fitted in blue

The approach has been mainly tested on the KITTI and Parking datasets, because it could be compared with the ground truth values for the found trajectory. In the parking dataset, the approach demonstrates robustness to outliers due to high dynamic range in the picture.

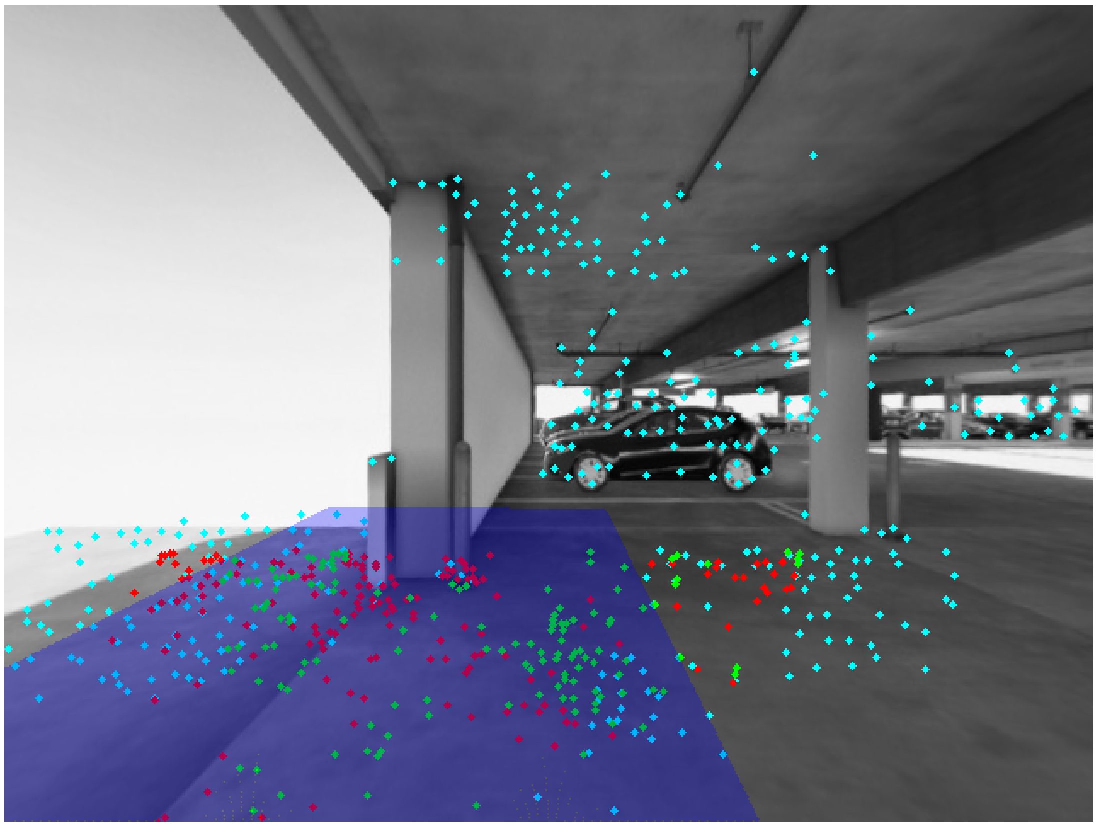
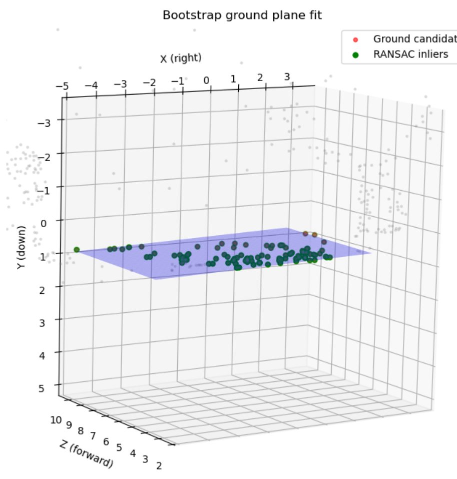
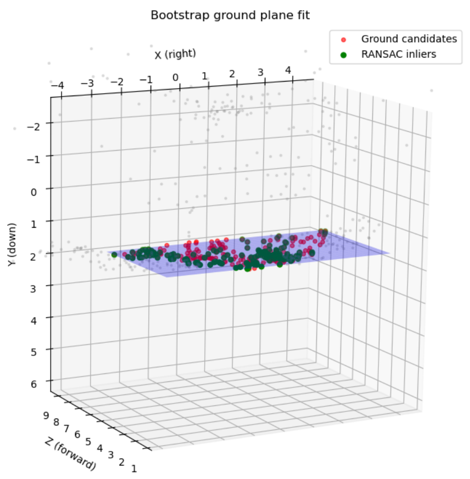


Figure 3: Parking: keypoints in light blue, ground inliers in green, outliers in red, plane fitted in blue



(a) KITTI 3D ground reconstruction



(b) Parking 3D ground reconstruction

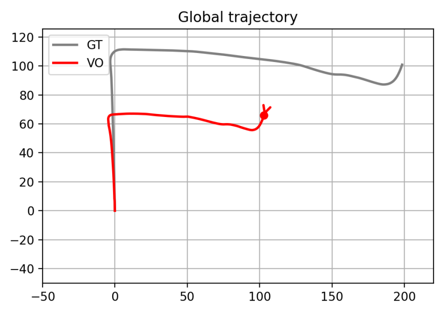
Figure 4: 3D point cloud: ground inliers in green, outliers in red, fitted plane in blue

5 Qualitative Evaluation-Results

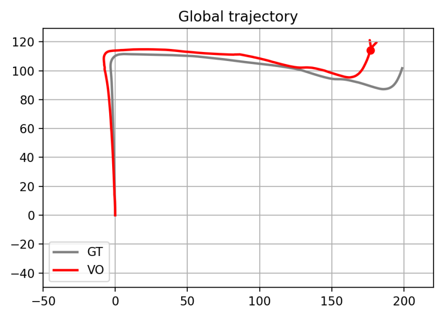
This section evaluates the pipeline's performance across three standard benchmarks and a custom sequence, focusing on local consistency and robustness. KITTI, MALAGA and CUSTOM datasets were run on an `intel core ultra 9 285H, 32 GB RAM`. The top clock frequency is 5.1 GHz, but the one measured while running the process was around a mean of 2GHz. The Parking dataset was executed on a `MacBook Pro` featuring an `Apple M1` chip with `8 CPU cores` (4 performance and 4 efficiency cores), for a total of `8 logical threads`, and `8 GB` of unified memory. Due to dynamic frequency scaling, the CPU does not expose a fixed clock frequency; however, the maximum frequency is approximately `3.2 GHz`. The frame rate is displayed on top of the image frame in the visualization: without sliding window BA we were able to run real time (25 Hz).

5.1 KITTI Dataset

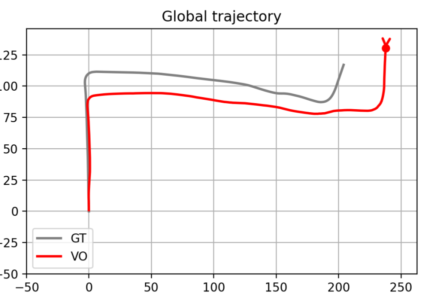
The KITTI dataset shows a series of frames in an outdoor street environment, where enough texture is present. The ground truth motion is generally straight with well-recognizable rotations of the camera.



(a) Trajectory with BA



(b) With BA and scale adjustment



(c) Trajectory with MonoVO

Figure 5: Evaluation on the KITTI dataset. The stable number of PnP inliers over time supports smooth local pose estimation, while global drift accumulates over long trajectories, as expected in monocular VO.

A Shi-Tomasi corner detector is used with a moderate quality threshold (`qualityLevel=0.01` and `maxCorners=100`) to get a stable number of keypoints, using a 21x21 window and 2 KLT pyramid levels.

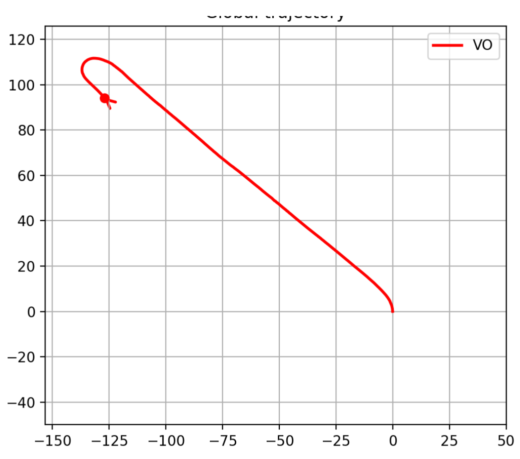
Behaviour and limitations. In this dataset, using PnP with RANSAC (with `reprojectionError=5` and `confidence=0.9`), the number of inliers in the frames is stable and we get a smooth frame to frame movement.

As expected for a monocular VO pipeline, scale drift affects global consistency. The KITTI dataset provides a textured outdoor environment where the pipeline achieved stable pose estimation during straight segments. Using a moderate Shi-Tomasi quality threshold and a 21×21 KLT window, the system produced smooth transitions. However, rapid camera rotations caused temporary drops in 2D-3D correspondences, leading to reduced stability in those segments. While the trajectory is locally consistent, typical monocular scale drift accumulates over long distances without global optimization.

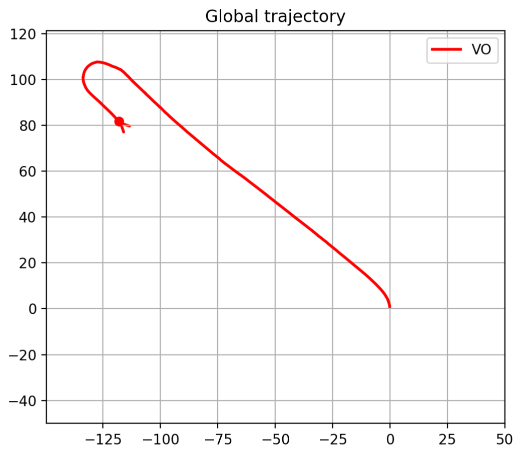
It's important we mention that the screen-cast of KITTI without scale estimation and sliding window bundle adjustment **is not sped up**, and that we did not have enough time to fine-tune the parameters, however it still runs smoothly and in real time.

5.2 Malaga Dataset

This dataset shows urban environment frames with mild texture and abrupt rotations.



(a) Trajectory with BA



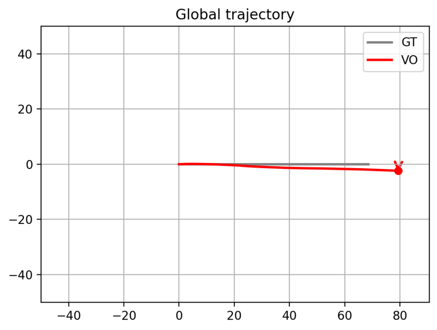
(b) Trajectory with MonoVO

Figure 6: Evaluation on the Malaga dataset. Temporary drops in the number of PnP inliers correspond to wide roundabouts and open areas with limited nearby structure, where reduced parallax impacts landmark triangulation.

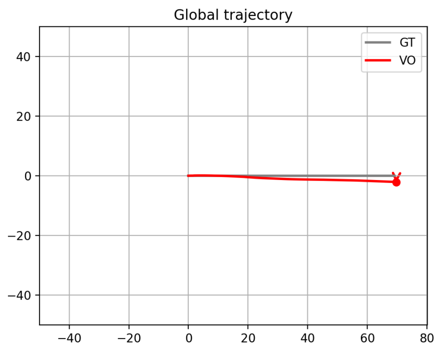
In the pipeline for this dataset, a higher threshold for the Shi-Tomasi corner detector is used to improve the performance for cluttered scenes(`qualityLevel=0.02`).
The images are based in a *crowded* urban setting, so the pipeline utilized a higher corner detection threshold to filter out low-texture regions. The system demonstrated high inlier counts on structured streets, but struggled in wide roundabouts and open areas where the sky dominated the frame, limiting the parallax for triangulation of landmarks. The implementation of an absolute minimum eigenvalue threshold rejected weak candidates in sky regions, ensuring that local motion remained smooth despite challenging lighting and reflections.

5.3 Parking Dataset

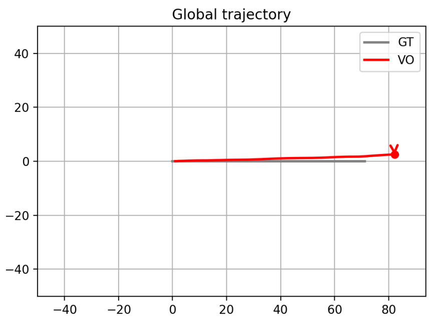
This dataset mainly shows repetitive structures and limited visibility over background objects, and predominantly shows planar motion of the camera.



(a) Trajectory with BA



(b) With BA and scale adjustment



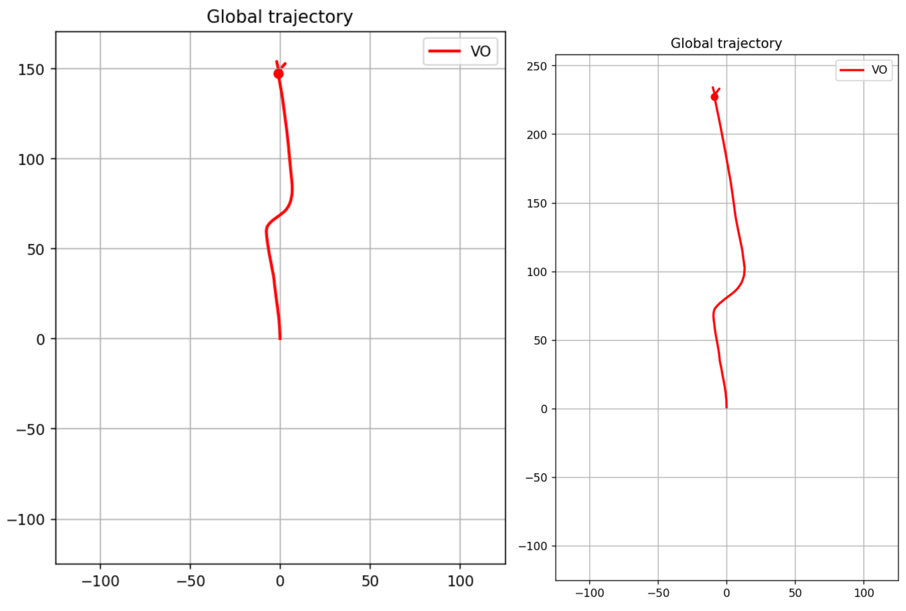
(c) Trajectory with MonoVO

Figure 7: Evaluation on Parking dataset. The sufficient number of inlier leads to local consistency, reconstructing the lateral motion of the camera.

The Parking dataset requires an increased number of features and a more aggressive bearing angle threshold ($\alpha = 0.05$ rad). These adjustments ensure that only landmarks with sufficient baseline are added to the state, preventing ill-conditioned depth estimates. Despite a faster decrease of inliers compared to other datasets, the refined triangulation logic maintained the correct orientation and reconstructed the lateral camera path accurately. For this dataset we are using a 5-frame-long refinement window, a 0.95% confidence and a reprojection error of 2 pixels for RANSAC.

5.4 Custom Dataset

The custom data set was recorded in an urban data set in Rome, under wet conditions (visible in Figure 8), which introduce non-Lambertian reflections and specular trackings. These elements challenge the typical assumptions of feature tracking and could increase the outliers correspondences.



(a) Trajectory with BA (b) Trajectory with MonoVO

Figure 8: Evaluation on the custom dataset recorded under wet conditions. Despite reflections and unstable features caused by rain, the trajectory remains consistent, indicating robustness to moderate real-world visual disturbances.

Recorded under wet conditions in Rome, the custom sequence tested the pipeline's robustness against non-Lambertian reflections and specular tracking. Even with visual disturbances from puddles and wet asphalt, the combination of Shi-Tomasi and KLT tracking maintained a stable number of inliers. The local consistency remained good throughout the sequence, with Bundle Adjustment further smoothing the trajectory by optimizing over a temporal window, demonstrating the system's ability to compensate against real-world disturbances.

Edge Cases

The main degradation is expected in frames with a large portion of the scene represented by reflective surfaces. In these frames, a drop in the number of PnP inliers occurs, which can lead to noisier pose update. However, the local consistency of the reconstructed motion is quite impressive, indicating that, not considering scale drift, the pipeline generalizes well to small real world disturbances. Bundle Adjustment provides a smoother trajectory compared to the simple Monocular VO pipeline: in particular, there are no abrupt changes in the trajectory, which is expected, as it is optimized over a window of frames. On the other hand, scale estimation impact over this dataset cannot be truly quantified due to the lack of the ground truth, so it was not even used in this case.

Author Contributions to the project

The tasks were distributed among team members and completed as follows:

Team Member	**Tasks and Contributions**
Ferranti Alessandro	Developed bootstrap feature extraction (3.1), initial feature tracking across intermediate frames (3.2), and relative pose estimation between bootstrap key-frames (3.3). Acquired and calibrated the custom dataset (5.4).
Jensen David	Developed bootstrapping of initial point cloud (3.4), logic for adding new keypoint candidates in continuous operation (3.10), software organization (Git and Jupyter Notebook), and initial integration and visualization (3.11)
Pirini Alessandro	Developed the triangulation of new landmarks (3.8), feature extraction in continuous operation (3.9); refined the visualization system (3.11) and tuned the parameters for KITTI dataset (5.1). Implemented the additional ground detection and absolute scaling feature (4.2).
Rubini Matteo	Responsible for the initialization of the state (3.5), tracking the keypoints forward (3.6) and estimating the pose during operation (3.7). Tuned the parameters for Parking and Malaga (5.2 - 5.3). Implemented Sliding Bundle Adjustment additional feature(4.1).

References

[1] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[2] V. et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* (2020).

[3] S. C. Contributors. *Bundle Adjustment.* `https://github.com/scipy/scipy-cookbook/blob/main/ipython/bundle_adjustment.ipynb`. Accessed: 28 Dec 2025. n.d.

[4] M. Menze and A. Geiger. "Object Scene Flow for Autonomous Vehicles". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.