University of Zurich

Vision Algorithms for Mobile Robotics

Monocular Visual Odometry

A Real-Time Implementation with Bundle Adjustment Refinement

Group Members

Matteo Cascone
Luca Gandolfi
Angelo Stefanini
Pietro Venè

Fall Semester 2025

Contents

Abstract

Visual odometry is a key capability in modern robotics. Drones, exploration rovers, and underwater robots often operate in environments where conventional navigation systems are unreliable or unavailable: while wheel encoders may suffer from slippage, and GPS signals can be degraded or entirely absent, a single onboard camera is still enough to successfully localize the vehicle.
In this mini-project, we present a simple yet effective monocular visual odometry pipeline capable of estimating the camera pose trajectory while simultaneously reconstructing a sparse 3D point cloud of the observed environment. Despite the inherent scale ambiguity of monocular setups, the proposed approach shows excellent local consistency and achieves satisfactory accuracy, especially thanks to the refinement performed with bundle adjustment.

1 Introduction

In the following sections, we present our monocular visual odometry pipeline and its main building blocks, with emphasis on a clean, modular implementation designed for clarity and collaborative development. The system includes a unified visualization interface for monitoring run-time performance, which can be disabled via the `visualize_frames` flag to improve efficiency. We then evaluate the pipeline on three public datasets and on a self-recorded sequence, discussing dataset-specific challenges and optimal parameter settings.

2 Building blocks of the pipeline

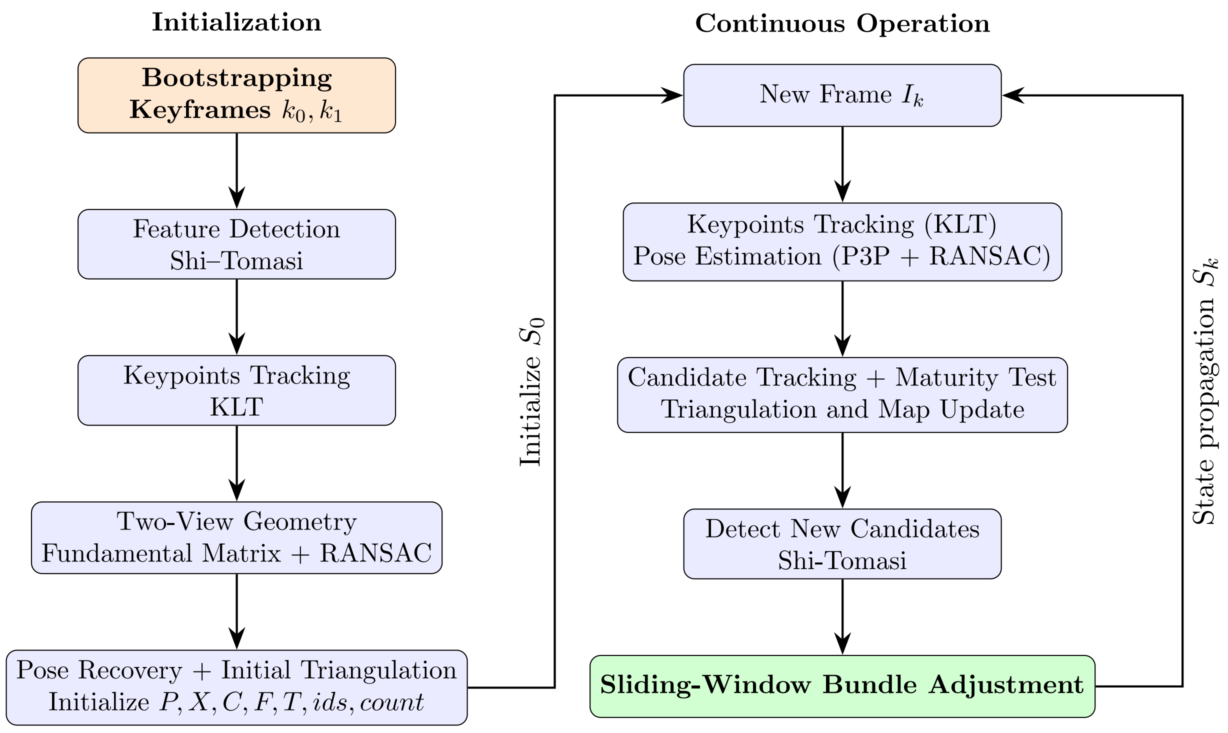

Initialization
Bootstrapping Keyframes k_0, k_1
Feature Detection Shi–Tomasi
Keypoints Tracking KLT
Two-View Geometry Fundamental Matrix + RANSAC
Pose Recovery + Initial Triangulation Initialize $P, X, C, F, T, ids, count$
Initialize S_0
Continuous Operation
New Frame I_k
Keypoints Tracking (KLT) Pose Estimation (P3P + RANSAC)
Candidate Tracking + Maturity Test Triangulation and Map Update
Detect New Candidates Shi-Tomasi
Sliding-Window Bundle Adjustment
State propagation S_k

Figure 1: Overview of the proposed monocular visual odometry pipeline with two-view initialization, iterative pose estimation, and local Bundle Adjustment.

2.1 Bootstrapping

The initialization of the pipeline consists in defining the world reference frame and generating the initial 3D point cloud.

This process is carried out using two selected frames, referred to as **keyframes**. The first keyframe corresponds to the first frame of the dataset, while the choice of the second one is context-dependent. This is due to the fact that depth estimation uncertainty decreases as the baseline increases:

$$\Delta Z = \frac{Z^2}{bf}\Delta D.$$

To avoid unreliable landmark triangulation, frames are skipped until the keyframe-to-keyframe baseline becomes sufficiently large. This condition is empirically verified using:

$$\frac{b}{Z_{\text{av}}} = \frac{1}{10}.$$

Once the two keyframes have been selected, the next step is to extract the 2D–2D correspondences required for two-view Structure-from-Motion. This is achieved by first detecting salient points on the initial keyframe. A good trade-off between accuracy and computational efficiency is provided by the Shi–Tomasi corner detector, implemented in OpenCV as `cv2.goodFeaturesToTrack` [1].

The next step consists in finding the correspondences in the second keyframe for the keypoints detected in the first one. This can be achieved either through feature-based indirect methods, which rely on descriptors and matching algorithms, or through pixel-intensity based direct methods, which minimize the photometric error. Since the latter are faster and less prone to outliers, we adopt a direct approach, using OpenCV's KLT tracker `cv2.calcOpticalFlowPyrLK` [2].

Exploiting the obtained 2D–2D correspondences, we proceed with two-view Structure-from-Motion. By fixing the world reference frame to coincide with the pose of the first keyframe, the goal is to estimate the pose of the second one, defined by a rotation matrix `R` and a translation vector `T`.

These quantities are encoded in the **essential matrix**, defined as:

$$E = [T]_\times R,$$

which is included in the definition of the epipolar constraint:

$$\bar{p}_2^\top E \bar{p}_1 = 0.$$

To strictly exploit the **8-point algorithm**, instead of the essential we estimated the **fundamental matrix** using OpenCV's `cv2.findFundamentalMat` [3], which implements **RANSAC** for outlier rejection and solves the epipolar constraint expressed as:

$$p_2^\top F p_1 = 0.$$

After recovering E as:

$$E = K^T F K$$

the essential matrix is then decomposed into `R` and `T` via `cv2.recoverPose` [4], which resolves the four-fold ambiguity by enforcing the **cheirality constraint**, ensuring that the reconstructed 3D points lie in front of both cameras.

At this point, knowing all the required variables, we can finally generate the 3D point cloud by triangulating the corresponding image points, using OpenCV's `cv2.triangulatePoints` [5].

Lastly, a crucial step is the state initialization. Given the Markovian nature of the proposed pipeline, particular care must be taken at this stage, as each iteration of the algorithm depends entirely on the state produced by the previous one. The state is defined by the sets P, X, C, F, T, *ids* and *count*.

In the bootstrapping phase, these quantities are initialized as follows:

- $P \in R^{2\times N}$ — image coordinates of keypoints in the second keyframe that are classified as inliers by `cv2.findFundamentalMat` and `cv2.recoverPose`, both of which perform outlier rejection.
- $X \in R^{3\times N}$ — 3D landmarks triangulated from the keypoints in P.
- $C \in R^{2\times M}$ — set of candidate keypoints. It is obtained by running `cv2.goodFeaturesToTrack` on the current keyframe and extracting features that are not already present in P. This is enforced by providing the detector with a mask of occupied pixels.
- $F \in R^{2\times M}$ — image coordinates of the candidate keypoints at the time of their first detection. At the end of the bootstrapping phase, F coincides with C.
- $T \in R^{12\times M}$ — camera poses associated with each candidate keypoint at first detection, represented as unrolled 3×4 transformation matrices. In this case, each column corresponds to the pose of the second keyframe.
- $ids \in R^{1\times M}$ — id tracker for every 3D Landmark (used in Bundle Adjustment).
- $count \in R^{1\times M}$ — number of consecutive frames in which the landmark was tracked successfully (used in Bundle Adjustment).

In practice however T is not saved in this way: for efficiency reasons we save T in *T_cw_array* for every frame and for every candidate point we save the id of the frame where it first appeared in *frame_id*, so that with *frame_id* we can recover the pose.

2.2 Camera pose trajectory estimation

After the successful initialization of the pipeline, the algorithm enters its iterative phase, in which the same set of operations is executed for each incoming frame. The first operation is **localization**, namely the estimation of the 6-DoF camera pose of the current frame.

As discussed previously, localization relies entirely on the state propagated from the previous frame. From this point onward, quantities associated with the current frame are denoted by the subscript $_k$, while those from the previous frame are denoted by $_{k-1}$.

The keypoints in P_{k-1} are first tracked to the current frame using `cv2.calcOpticalFlowPyrLK`, yielding a temporary set P'_k. The corresponding 3D landmarks are then retrieved from X_{k-1}, forming the set X'_k, which defines the 3D–2D correspondences required for pose estimation. Localization is performed using OpenCV's `cv2.solvePnPRansac` [6], which estimates the camera pose with respect to the world frame using the **P3P** formulation and applies **RANSAC** for inlier selection. As a result, additional outliers are rejected, yielding the refined inlier sets P''_k and X''_k.

2.3 3D point cloud extension

Due to the Markovian formulation, the set of 3D landmarks is incrementally updated at each frame by exploiting the candidate sets C_{k-1}, together with their associated auxiliary sets F_{k-1} and T_{k-1}. These candidates are first tracked to the current frame using `cv2.calcOpticalFlowPyrLK`, yielding the temporary set C'_k, from which the updated auxiliary sets F'_k and T'_k are derived.

As in the bootstrapping phase, reliable triangulation requires a sufficiently large baseline between the frame of first observation and the current one. Candidate readiness is evaluated using the criterion proposed in the project statement: a candidate is considered *mature* if the angle between the bearing vectors at first observation and at the current frame exceeds a predefined threshold. During parameter tuning, this threshold was empirically adjusted for each dataset.

Mature candidates are then triangulated using `cv2.triangulatePoints`. Their image coordinates and corresponding 3D points are appended to the keypoint and landmark sets, yielding P_k and X_k. Finally, the triangulated candidates are removed from the candidate pools, resulting in the updated sets C''_k, F''_k, and T''_k.

2.4 Candidate set update

After localizing the current frame and triangulating mature candidates, the candidate set is augmented with newly detected features. As in the bootstrapping phase, a mask excluding points already in P_k and C''_k is provided to `cv2.goodFeaturesToTrack`. The detected keypoints are added to the candidate set, yielding C_k and F_k, while the current camera pose is appended to the pose set, forming T_k. This completes the state update for the current frame.

2.5 Additional features

2.5.1 Local optimization with Bundle Adjustment

To jointly refine camera poses and 3D landmark positions, **Bundle Adjustment (BA)** minimizes the reprojection error through a nonlinear least-squares formulation:

$$\{P^i, C_1, \ldots, C_{N_{\text{frames}}}\} = \underset{P^i, C_1, \ldots, C_{N_{\text{frames}}}}{\arg\min} \sum_{k=1}^{N_{\text{frames}}} \sum_{i=1}^{N_{\text{landmarks}}} w_i^k \, \rho\Big(p_i^k - \pi(P^i, K, C_k)\Big),$$

where $P^i \in R^3$ denotes the i-th 3D landmark, C_k the camera pose at frame k, and $p_i^k \in R^2$ its observed image location. The projection function $\pi(\cdot)$ uses the intrinsic calibration matrix K. Residuals are weighted by w_i^k, chosen inversely proportional to the landmark depth to improve numerical conditioning, and the Huber loss $\rho(\cdot)$ is adopted for robustness against outliers.

Bundle Adjustment is performed locally using a *sliding-window* strategy to ensure computational efficiency and real-time feasibility. The window includes a fixed number of recent frames, controlled by `buffer_dim` $= 5$, and optimization is executed at each frame. To stabilize the problem, a gauge-fixing strategy [7] is applied by keeping the first `n_fix_ba` $= 1$ (or 2) camera poses in the window fixed, while optimizing the remaining ones. Camera poses are initialized from PnP with RANSAC estimates, while 3D landmarks are initialized from the current map state.

Only a subset of landmarks is included in the Bundle Adjustment to control computational complexity. Only landmarks observed within the sliding window are considered and filtered by a valid depth range `z_threshold_ba` $= [1.0, 100.0]$ and a minimum track length of three frames; newly triangulated points are excluded by construction. If the number of eligible landmarks exceeds `max_num_ba_points` $= 100$, a random subset of at most `max_num_ba_points` points is selected.

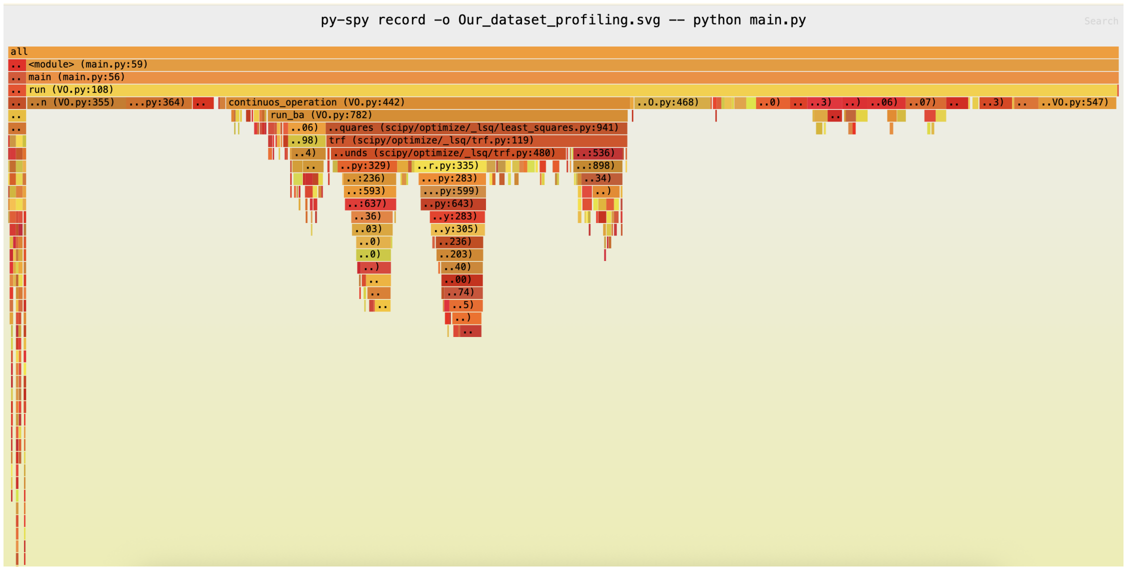
The resulting nonlinear least-squares problem is solved using `scipy.optimize.least_squares` [8] with the *Trust Region Reflective* (TRF) algorithm [9]. A sparse Jacobian structure is explicitly provided, exploiting the fact that each reprojection residual depends only on a single camera pose and a single 3D point; residuals associated with fixed camera poses therefore depend only on landmark parameters, further reducing the effective problem dimensionality. Convergence is controlled through tolerances on the cost function, parameter updates, and gradient norm (`ba_tol` $= 10^{-2}$), as well as a maximum number of function evaluations (`max_nfev` $= 50$). After convergence, the optimized camera poses and 3D landmark positions are written back to the sliding-window buffer and the global map, respectively. Bundle Adjustment is computationally expensive: thus, tuning parameters was crucial to balance the trade-off between trajectory accuracy and computational efficiency, as it will be discussed later.

2.5.2 Real-time implementation

To achieve real-time performance, the code was profiled using `py-spy` [10]. The resulting flame graphs identified the most computationally expensive components of the pipeline, which were subsequently optimized in an iterative manner. For example, the computation of the angles used to assess candidate maturity was refactored and fully vectorized, significantly reducing its contribution to the overall runtime. As examples of other tricks, we precompute the inverse of K just once at the beginning of the algorithm, and we save at every frame not only the pose T_{cw} in *T_cw_array* but also it's inverse T_{wc} in *T_wc_array*, so that we don't have to recompute it every time.

A further major optimization concerned the enforcement of a minimum spatial distance between candidate points. In the initial implementation, this constraint was verified through pairwise distance checks between all candidates, which, despite vectorization, introduced a substantial computational overhead. This procedure was replaced with a more efficient image-based strategy: exclusion regions with a radius equal to the *Minimum Distance* are drawn onto a binary mask, and new candidate points are extracted directly from the remaining valid regions. This approach eliminates explicit pairwise comparisons and considerably accelerates this stage of the pipeline.

After implementing Bundle Adjustment, the frame rate fell heavily, so balancing efficiency and accuracy became crucial: the greatest impact was obtained by reducing the dimension of the sliding window, the number of the landmarks used in the optimization and



```
py-spy record -o Our_dataset_profiling.svg -- python main.py
```

Figure 2: Flame graph of the algorithm applied to our dataset. Bundle Adjustment accounts for the largest share of the runtime. Among the remaining components, `cv2.imread`, `cv2.calcOpticalFlowPyrLK`, and `cv2.goodFeaturesToTrack` have almost equal contributions, while the rest of the operations individually account for only a negligible fraction of the total runtime.

increasing the tolerance of the `least_squares` function: however, if set too permissive, the Bundle Adjustment effect became negligible. After multiple evaluations, we were able to tune the parameters such that the performance remained optimized while running the Visual Odometry in real time.
As an anticipation of Section 3, Table 1 reports the frame rates achieved by the full pipeline on the evaluated datasets.

Dataset	**Frame Rate [Hz]**	
	Without BA	**With BA**
Parking	86	31
Malaga	69	26
KITTI	59	22
Our dataset	45	30

Table 1: Average processing rate with visualization disabled. Machine: MacBook Air M4 (10 cores), 16GB of RAM

3 Results on datasets

In the following section, we present the results obtained with the proposed pipeline on the different datasets. Trajectory plots are adopted as the primary evaluation tool: whenever ground truth is available, the estimated trajectories are compared against it, while explicitly accounting for the **intrinsic scale ambiguity** of monocular vision. In the absence of ground truth, qualitative indicators such as tracking stability, trajectory smoothness, and local geometric consistency are used to assess performance. Overall, the experimental results show that the selected parameter configuration **generalizes well across all datasets**, without requiring significant dataset-specific tuning.

In addition, we provide a visual comparison to highlight the effectiveness of Bundle Adjustment by contrasting the estimated trajectories obtained with the optimization enabled and disabled. To further demonstrate the real-time performance of the proposed pipeline, we provide recorded videos of the experiments. Direct links to the corresponding YouTube videos for each dataset are reported in Section 7.

3.1 Parking

On the Parking dataset, the pipeline achieves reliable localization throughout the entire sequence, maintaining a PnP inlier ratio of approximately 70–90% despite illumination variations. The estimated trajectory closely follows the ground-truth relative motion, while real-time performance is sustained at approximately **31 Hz** when sliding-window Bundle Adjustment is enabled.

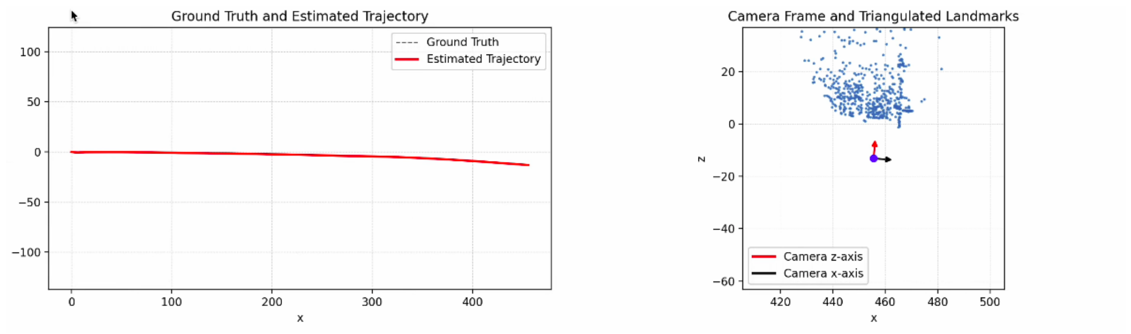


Figure 3: Estimated and ground-truth trajectories with triangulated landmarks for the current frame .

To isolate the effect of local optimization, we also report results obtained with Bundle Adjustment disabled. In this configuration, the system runs at a significantly higher frame rate (**86 Hz**), but exhibits increased drift and reduced trajectory smoothness.

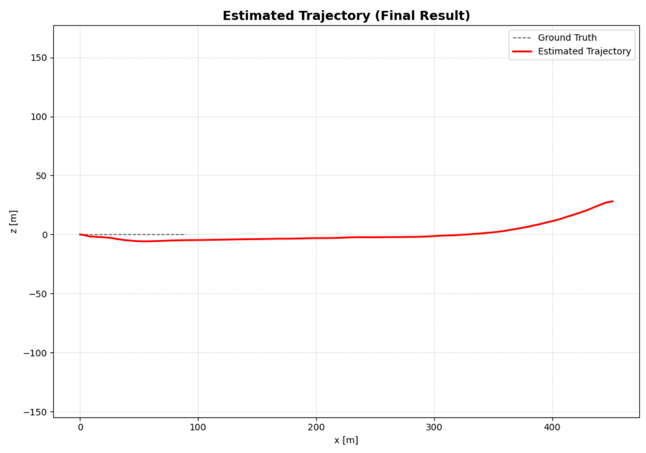
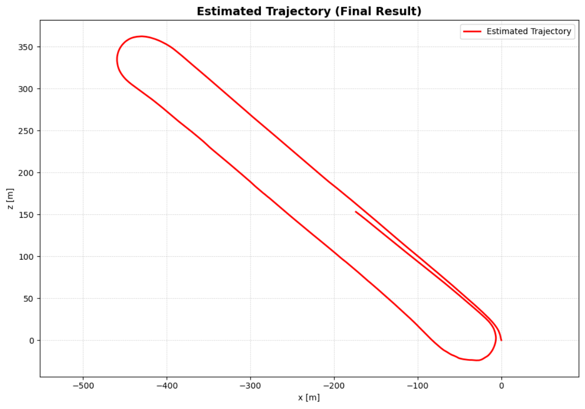


Figure 4: Estimated and ground-truth trajectories on the Parking dataset without BA, showing increased drift despite higher frame rate.

3.2 Malaga

Figure 5 shows the estimated trajectories for the Malaga dataset, obtained using only the rectified left camera images, with and without sliding-window Bundle Adjustment.



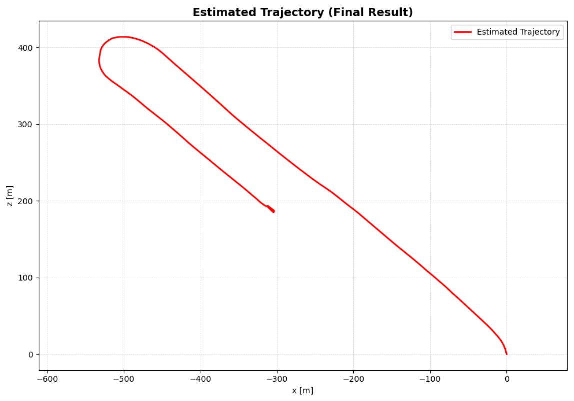
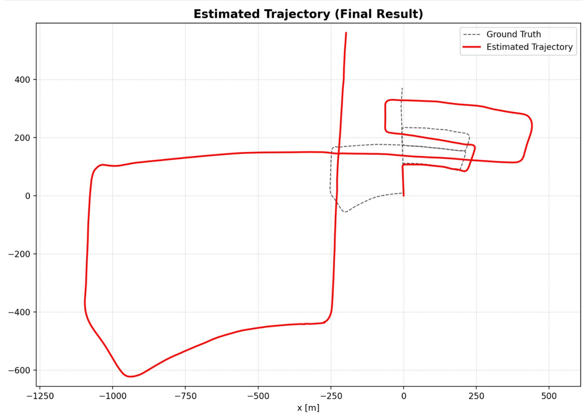


Figure 5: Estimated (final) trajectory and tracked inliers for the Malaga dataset.

Despite the absence of ground truth, the estimated motion remains visually consistent and smooth, and the reconstructed landmarks stay well aligned with the observed camera motion. Enabling Bundle Adjustment further improves trajectory smoothness and local geometric consistency, whereas disabling optimization leads to increased local drift and a noticeable compression of the estimated scale. Overall, these results confirm that the chosen parameter configuration generalizes effectively to complex urban scenarios while maintaining real-time performance (**26 Hz** with Bundle Adjustment).

3.3 KITTI

Figure 6 show the estimated trajectories on the KITTI dataset obtained with and without local Bundle Adjustment. In both cases, the pipeline maintains continuous and stable localization over the entire sequence.



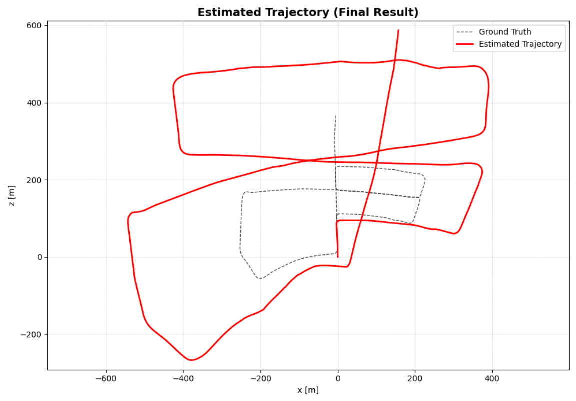


Figure 6: Estimated trajectories with (left) and without sliding-window Bundle Adjustment.

When local optimization is enabled, the estimated trajectory exhibits smoother motion, reduced deviations during turns and changes in direction, and highly accurate **locally consistent** motion estimates. The complete system operates at approximately **22 Hz**, demonstrating real-time performance even with Bundle Adjustment.

3.4 Our Dataset

In addition to the publicly available benchmarks, we evaluated the proposed pipeline on a self-recorded dataset acquired with a handheld camera. The sequence was captured at a resolution of **1280 × 720** pixels and a frame rate of **30 fps**, and is characterized by relatively small inter-frame motion baselines, typical of pedestrian motion.

Camera intrinsic parameters were estimated using the `Camera Calibration Toolbox` [11] through a checkerboard-based calibration procedure. The resulting intrinsic matrix was used consistently throughout the entire pipeline.

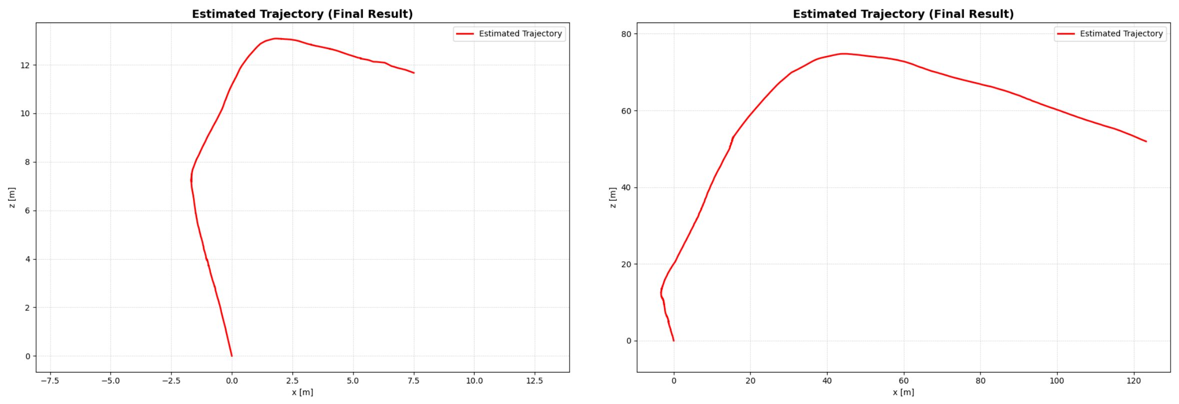


Figure 7: Estimated trajectories on the self-recorded dataset with sliding-window Bundle Adjustment enabled (left) and disabled (right).

Despite the challenging motion characteristics, the pipeline maintains stable tracking and achieves real-time performance, processing an average of approximately **30 frames per second** with BA enabled. As illustrated in Fig. 7, the inclusion of sliding-window BA leads to a noticeably smoother and more geometrically consistent trajectory, effectively reducing local drift and short-term inconsistencies with respect to the solution obtained without optimization. When BA is enabled, the number of tracked keypoints remains consistently higher and the ratio of PnP inliers improves, indicating a better-conditioned pose estimation problem.

Moreover, the reprojection error is usually reduced by one to three orders of magnitude at each optimization step, and convergence is typically reached in a small number of iterations, thanks to the good initialization provided by the PnP solution and the limited size of the sliding window. In more challenging motion segments, the local optimization may be less effective and residual errors can remain higher; nevertheless, Bundle Adjustment helps limit error accumulation and improves the overall consistency of the estimated trajectory.

4 Conclusion

In this project, we developed a complete monocular visual odometry pipeline, from an initial bootstrapping stage to a fully iterative framework capable of estimating the camera trajectory while incrementally building a sparse 3D representation of the environment. The system adopts a modular and lightweight architecture, favoring clarity, ease of inspection, and extensibility.

Across all evaluated datasets, the pipeline demonstrates stable behavior and good local consistency despite the intrinsic limitations of monocular vision. The combination of

KLT-based feature tracking, robust pose estimation via PnP with RANSAC, incremental triangulation, and sliding-window Bundle Adjustment proves effective across different motion patterns and scene configurations, with limited dataset-specific tuning.

Overall, the results indicate that a clean and well-structured monocular visual odometry implementation, supported by robust geometric estimation and local optimization, can achieve accurate real-time performance in practical scenarios. The proposed framework provides a extremely solid basis for possible extensions, such as stereo integration to address scale ambiguity or tightly coupled IMU fusion to further enhance robustness and accuracy.

5 Members contribution

- **Matteo Cascone:** Bundle Adjustment, Parameters Tuning, general work on the pipeline.
- **Luca Gandolfi**: Performance improvement, Bundle Adjustment, general work on the pipeline.
- **Angelo Stefanini**: Set-up, continuous operation, general work on the pipeline.
- **Pietro Venè:** Bootstrapping, plotting, first implementation of localization, general work on the pipeline.

6 YouTube links

- **KITTI:**
  - Live plot: https://youtu.be/C5oUy4Wf8pg
  - Real time: https://youtu.be/xDAxI2Blp0c
  - Real time without BA: https://youtu.be/1oaR3rvHTfM
- **Malaga:**
  - Live plot: https://youtu.be/Yn3_Sx4yHeA
  - Real time:https://youtu.be/eX0SOViOHJY
- **Parking:**
  - Live plot: https://youtu.be/RstgSvBFxi8
  - Real time: https://youtu.be/3hI08cnOMuU
- **Our dataset:**
  - Live plot: https://youtu.be/ECT0WpL7LmY
  - Real time: https://youtu.be/u3qOrVqJXfc

7 GitHub repository

https://github.com/angelostefanini0/monocular_VO

References

[1] OpenCV. “cv2.goodFeaturesToTrack.” Accessed: December 30, 2025. [Online]. Available: https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga1d6bb77486c8f92d79c8793ad995d541.

[2] OpenCV. “cv2.calcOpticalFlowPyrLK.” Accessed: December 30, 2025. [Online]. Available: https://docs.opencv.org/4.x/dc/d6b/group__video__track.html#ga473e4b886d0bcc6b65831eb88ed93323l.

[3] OpenCV. “cv2.findFundamentalMat.” Accessed: December 30, 2025. [Online]. Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga59b0d57f46f8677fb5904294a23d404a.

[4] OpenCV. “cv2.recoverPose.” Accessed: December 30, 2025. [Online]. Available: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#gadb7d2dfcc184c1d2f496d8639f4371c0.

[5] OpenCV. “cv2.triangulatePoints.” Accessed: December 30, 2025. [Online]. Available: https://docs.opencv.org/4.x/d0/dbd/group__triangulation.html.

[6] OpenCV. “cv2.solvePnPRansac.” Accessed: December 30, 2025. [Online]. Available: https://docs-opencv-org.translate.goog/4.x/d9/d0c/group__calib3d.html?_x_tr_sl=en&_x_tr_tl=it&_x_tr_hl=it&_x_tr_pto=sc#ga50620f0e26e02caa2e9adc07b5fbf24e.

[7] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, “Bundle adjustment — a modern synthesis,” *Vision Algorithms: Theory and Practice*, pp. 298–372, 2000, Accessed: 2025-01-04.

[8] P. Virtanen, R. Gommers, T. E. Oliphant, et al., “Scipy 1.0: Fundamental algorithms for scientific computing in python,” *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020, Accessed: 2025-01-04. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html.

[9] M. A. Branch, T. F. Coleman, and Y. Li, “A subspace, interior, and conjugate gradient method for large-scale bound-constrained minimization problems,” *SIAM Journal on Scientific Computing*, vol. 21, no. 1, pp. 1–23, 1999, Accessed: 2025-01-04. [Online]. Available: https://doi.org/10.1137/S1064827595289108.

[10] B. Goregaokar, *Py-spy: Sampling profiler for python programs*, https://github.com/benfred/py-spy, Accessed: 2025-01-04, 2020.

[11] MathWorks. “Camera Calibration Toolbox.” Accessed: December 30, 2025. [Online]. Available: https://www.mathworks.com/help/vision/camera-calibration.html.