Implementation of a Monocular Visual Odometry Pipeline

Vision Algorithms for Mobile Robotics: Mini Project

Dávid Tokár, *david.tokar@uzh.ch*
Tim Wyss, *tiwyss@ethz.ch*
Alessandro Pietrogiovanna, *apietrogiova@ethz.ch*
Antonio Zecchin, *azecchin@ethz.ch*
Robotics and Perception Group, University of Zurich

January 5, 2026

Abstract

This report presents the development and qualitative evaluation of a monocular visual odometry (VO) pipeline. The system encompasses two-view initialization, KLT-based keypoint tracking, pose estimation via RANSAC-PnP, and landmark triangulation. We evaluate the pipeline on the KITTI, Malaga, and Parking datasets, as well as two custom-recorded sequences. Our results demonstrate locally consistent trajectory estimation and robust landmark management under varying environmental conditions.

1 Introduction

The objective of this project is to implement a monocular VO pipeline with the following steps: initialization of landmarks and poses, tracking of keypoints, estimation of poses using 2D-3D correspondences and triangulation of new landmarks. Various optimization strategies are implemented to combat the accumulation of errors (drift) and to maximize throughput. The pipeline is developed and tested against three provided datasets and is additionally verified on two custom datasets recorded by us.

2 Methodology

The VO pipeline is based on concepts taught in the course "Vision Algorithms for Mobile Robotics" by Prof. Dr. Davide Scaramuzza at the University of Zurich. [1] It is implemented in Python in a complete Docker environment for reproducibility. All code is made publicly available on Github. [2]

2.1 Notation and Design

We denote the sequence of N recorded frames as $\{I^i\}_{i=1}^N$, where I^i represents the i-th frame of the sequence. The 6-DoF pose of the camera in the world frame at the i-th frame is denoted by the transformation matrix $T_{WC}^i \in \mathbb{R}^{4\times 4}$, which maps a point from the camera coordinate system to the world coordinate system.

Our pipeline follows a Markovian design, where the processing of the current frame I^i depends solely on the previous state S^{i-1} and the current image. We define the state of the visual odometry system as the tuple:

$$S^i = (P^i, X^i, C^i, F^i, \mathcal{T}^i) \tag{1}$$

where:

- P^i: The set of currently active 2D keypoints being tracked for pose estimation.
- X^i: The set of 3D landmarks corresponding to P^i, such that $x_k \in X^i$ is the world coordinate of $p_k \in P^i$.
- C^i: A set of candidate 2D keypoints being tracked but not yet triangulated.
- F^i: The pixel coordinates of the candidates C^i when they were first detected (the "anchor" observation).
- $\mathcal{T}^i$: The camera poses T_{WC} corresponding to the frames where candidates F^i were first detected.

2.2 Pipeline Architecture

The pipeline is orchestrated by a central `VoRunner` class, which manages the system state S and executes the processing steps sequentially. The operation is divided into two distinct phases: *Initialization* and *Continuous Operation.*

2.2.1 Initialization

This phase executes once at the beginning of the sequence. It is responsible for bootstrapping the system by establishing the first set of 3D landmarks and the initial metric scale. The runner processes incoming frames to detect features and tracks them until sufficient geometric parallax (which is chosen as a tuning parameter) is observed. Once a valid baseline is established, a two-view reconstruction is performed to populate the initial state $S^0 = (P^0, X^0)$, after which the system transitions to continuous mode.

2.2.2 Continuous Operation

For every subsequent frame, the pipeline executes the following steps in order:

1. **Image Preprocessing:** The raw image is undistorted using pre-computed lookup maps to remove lens distortion. Optionally, contrast enhancing and filtering methods can be applied to improve feature distinctiveness.

2. **Keypoint Tracking:** Existing keypoints (P) and candidates (C) are tracked from I^{i-1} to I^i using KLT optical flow.

3. **Pose Estimation:** The current camera pose T^i_{WC} is estimated using the tracked correspondences (P^i, X^{i-1}).

4. **Triangulation:** Candidates (C) that have sufficient parallax relative to their first observation $(F, \mathcal{T})$ are triangulated and promoted to landmarks (X, P).

5. **Local Bundle Adjustment:** A sliding window optimization refines the recent trajectory and landmark positions to reduce drift.

6. **Replenishment:** New candidate features are detected in texture-sparse regions of the image to maintain a robust spatial distribution of points for future frames.

2.3 Parameter Tuning and Validation

Given the diverse characteristics of the evaluation datasets which vary in framerate, resolution, and aspect ratio, we adopted a dataset-specific parameter tuning strategy.

To facilitate this calibration and efficiently isolate failure modes, we developed a modular testbench suite for the major pipeline components. This infrastructure allowed us to validate the performance of individual steps, such as tracking stability and triangulation accuracy, in isolation before integrating them. It only works when ground truth data are available and uses them as input values to then return the accuracy of the steps.

Moreover, the pipeline has an optional debug mode that provides detailed insight into the internal processing steps and intermediate results. When enabled, the pipeline generates a structured output directory containing per-frame visualizations of every step.

3 Implementation

This section details the core architecture of our monocular Visual Odometry pipeline. The implementation described here represents a minimal pipeline required for continuous operation, adhering to the fundamental components outlined in the project requirements.

3.1 Pipeline Initialization

The monocular visual odometry pipeline requires a reliable initial structure (a set of 3D landmarks and an initial baseline) to begin the recursive estimation process. Since a single image provides no depth information, the system must be initialized using two-view geometry. We implemented an initialization procedure that automatically selects a valid "keyframe" pair from the incoming video stream based on geometric stability criteria.

The process begins by detecting features in the first frame using the Shi-Tomasi algorithm. To ensure the initial map covers the scene widely, we tile the image into a grid and force feature detection within each tile. We then track these features into subsequent frames using the bidirectional KLT tracker described in more detail in Section 3.2

For every incoming frame, we evaluate its suitability as the second view for initialization. Comparing the pixel movement to a threshold is insufficient, as pure rotation can induce large pixel displacements without providing depth information. Instead, we compute the median parallax angle of all tracked features. We only attempt initialization if the median parallax exceeds a minimal threshold and the features occupy multiple distinct cells in a grid.

Once these conditions are met, we compute the Essential Matrix E using the 5-point algorithm within a RANSAC scheme (`cv2.findEssentialMat`) to reject outlier tracks inconsistent with epipolar geometry. The relative pose $(R, \mathbf{t})$ is recovered by decomposing E, and the correct solution is selected by enforcing the cheirality constraint where points must be in front of both camera poses. Finally, we triangulate the inlier correspondences to form the initial 3D map X^0. As the monocular scale is inherently ambiguous, we normalize the translation vector such that $\|\mathbf{t}\| = 1$. The initial average depth of the scene is calculated and stored to serve as a reference for the scale preservation heuristic used in later triangulation steps.

3.2 Feature Tracking and Association

To propagate the state S^{i-1} to the current frame I^i, we must associate existing landmarks with their new 2D locations. While independent feature matching (e.g., descriptor matching) is a viable strategy, it is computationally expensive and prone to outliers in repetitive textures. Instead, we employ the Kanade-Lucas-Tomasi (KLT) sparse optical flow algorithm to track keypoints P^{i-1} and candidates C^{i-1} directly into the current image.

In our implementation, we utilize a pyramidal implementation of Lucas-Kanade (`cv2.calcOpticalFlowPyrLK`) with tunable parameters. This multi-scale approach allows us to handle larger inter-frame displacements that occur during fast camera motion.

However, KLT tracking relies solely on local intensity consistency, which can lead to drift or false tracks in the presence of occlusion or dynamic objects. To maintain the integrity of the map, we implemented a two-stage filtering process. First, we perform a bidirectional consistency check. We track points forward from I^{i-1} to I^i to obtain estimates $\hat{P}^i$, and then track these estimates backward to I^{i-1} to obtain P'_{i-1}. We discard any track where the tracking error $\|\mathbf{p}_{i-1} - \mathbf{p}'_{i-1}\|_2$ exceeds a pixel threshold.

Second, to filter out dynamic objects that satisfy photometric consistency but violate the static world assumption, we enforce epipolar geometry constraints. Points that deviate too far from their epipolar lines are rejected as outliers. This ensures that only geometrically consistent correspondences are passed to the subsequent pose estimation module.

3.3 Pose Estimation

The primary objective of the pose estimation step is to determine the current camera pose T^i_{WC} by minimizing the reprojection error between the set of tracked 2D keypoints P^i and their corresponding 3D landmarks X^i.

Since the input correspondences from the tracking step inevitably contain outliers, we utilize the P3P algorithm within a RANSAC scheme (`cv2.solvePnPRansac`) to robustly estimate the initial pose.

However, RANSAC often produces a sub-optimal estimate because it fits the model to a minimal subset of points (3 or 4). To obtain a smoother and more accurate trajectory, we implemented a custom non-linear refinement step following the initial estimation. We perform a motion-only bundle adjustment, where the 3D landmarks are held fixed, and only the 6-DoF camera pose is optimized using the Levenberg-Marquardt algorithm (`scipy.optimize.least_squares`). To mitigate the influence of high-residual inliers that survived the RANSAC stage, we utilize a robust Huber loss function rather than a standard squared error.

3.4 Map Expansion and Maintenance

As the camera moves, existing landmarks eventually leave the field of view, necessitating the continuous triangulation of new map points to prevent tracking failure. To achieve this, we maintain a persistent set of candidate keypoints C, tracking them alongside the active landmarks until they exhibit sufficient parallax for reliable depth estimation.

Geometrically, a 3D point $\mathbf{X}$ is triangulated by finding the intersection of two bearing vectors originating from camera poses T^{first}_{WC} (where the feature was first observed) and T^{curr}_{WC} (the current frame). While any two views theoretically suffice, small baselines or degenerate motion (e.g., pure rotation) can result in large depth uncertainties. To prevent the corruption of the map with unreliable points, we implemented a filter.

First, we calculate the angle α between the two bearing vectors for every candidate. We only attempt triangulation if the pixel displacement and the parallax angle α are sufficient, the former serving primarily as a pre-filter for the computationally more expensive calculation of the parallax angle. Second, after computing the 3D coordinates, we enforce cheirality constraints to ensure the point lies in front of both cameras. Finally, we apply a depth filter, discarding points closer than the minimum depth or exceeding a maximum depth.

To reduce scale drift, we also enforce a constant scale heuristic during this step. If the initial average scene depth is known from bootstrapping, we rescale newly triangulated clusters to match this average depth relative to the camera center, as described in more detail in Section 4.2.

3.5 Feature Replenishment and Distribution

As the camera explores the environment, tracked landmarks inevitably drift out of the field of view or fail to track due to occlusion and perspective changes. To ensure the pose estimation problem remains well-constrained, the system must continuously replenish the set of active keypoints. Simply detecting the strongest features in the image is insufficient, as corner detectors tend to cluster points in high-contrast regions (e.g., foliage or textured pavements), leaving other areas of the image empty. This uneven distribution can lead to poor conditioning of the PnP problem and increased drift.

To combat this, we implemented a replenishment strategy that uses a grid-based "bucketing" approach. We divide the image into a grid and calculate the current density of active landmarks (both tracked P and candidates C) within each cell. New feature detection is only triggered if the total number of active points drops below a specific threshold.

During the detection phase, we first construct a mask to suppress regions surrounding existing keypoints, preventing the detection of redundant features. We then employ the Shi-Tomasi corner detector (`cv2.goodFeaturesToTrack`) to extract a large pool of potential candidates globally.

We then sort the candidates by their "corner response" score and assign them to their respective grid cells. We populate the candidate list by iterating through the cells and adding the highest-quality candidates only until the cell reaches its specific density cap. This ensures a uniform distribution of features across the image plane, improving the stability of the estimated motion.

4 Additional Features

While the baseline pipeline described in Section 3 provides a functional estimate of the camera trajectory, monocular systems are inherently prone to drift and scale ambiguity over long sequences. To address these limitations and satisfy the requirements for advanced feature implementation, we extended the system with several robustness modules that go beyond the minimal project specifications.

This section details these enhancements, focusing on two primary contributions: a Local Bundle Adjustment (BA) backend for reducing trajectory jitter, and a heuristic scale preservation strategy to mitigate monocular scale drift. Additionally, we discuss specific algorithmic deviations where our implementation diverges from the recommended project guidelines to achieve higher accuracy and stability.

4.1 Local Bundle Adjustment

While the frame-to-frame pose estimation ensures local consistency, errors inevitably accumulate over time, leading to significant trajectory drift. To mitigate this, we implemented a Local Bundle Adjustment (BA) module that refines the camera poses and 3D landmark positions over a sliding window of the previous frames.

We formulate this as a non-linear least squares optimization problem. Let $\mathcal{F}_W$ be the set of frames in the current window and $\mathcal{L}_W$ be the set of landmarks observed by at least two frames within this window. We seek to minimize the total robust reprojection error:

$$\min_{\{\mathbf{T}_i\},\{\mathbf{X}_j\}} \sum_{i\in\mathcal{F}_W} \sum_{j\in\mathcal{L}_W} \nu_{ij} \cdot \rho\left(\|\mathbf{p}_{ij} - \pi(\mathbf{T}_i\mathbf{X}_j)\|^2\right) \tag{2}$$

where $\mathbf{T}_i$ is the camera pose of frame i, $\mathbf{X}_j$ is the position of landmark j, ν_{ij} is a visibility indicator, and $\rho(\cdot)$ is the Huber robust loss function. To provide a stable reference for the optimization window, we fix the pose of the oldest frame (the "anchor frame") during the optimization.

Solving this optimization efficiently is critical for real-time performance. Relying on finite differences to approximate the Jacobian matrix is computationally prohibitive given the high dimensionality of the parameter space. Instead, we derived and implemented the *analytical Jacobian*. We compute the exact partial derivatives of the residuals with respect to the camera parameters and the 3D point coordinates. We exploit the specific block-sparsity structure of the BA problem where residuals only depend on one camera and one point to construct a sparse Jacobian matrix (`scipy.sparse.coo_matrix`). This is passed directly to the `scipy.optimize.least_squares` solver using the Trust Region Reflective (`trf`) algorithm, resulting in a significant speedup compared to numerical differentiation.

Furthermore, when a frame leaves the optimization window, its refined pose is cached and used as the linearization point for the next window's initialization. This prevents the discarding of information gained from previous optimization steps and improves the convergence speed of subsequent windows.

4.2 Scale Drift Mitigation

Monocular visual odometry suffers from an inherent scale ambiguity, where the metric scale of the scene can drift over time due to accumulation of small errors in translation estimation and the resulting inaccurate triangulation of new landmarks. To combat this, we implemented a heuristic scale preservation strategy during the triangulation step. During the initialization phase, we compute the average depth of the scene relative to the first camera frame. In subsequent frames, whenever a new cluster of landmarks is triangulated, we calculate the average depth of these new points relative to the current camera center and apply a scaling factor to the new cluster to realign it with the initialization scale. This approach assumes that the average distance to the scene remains roughly constant, as it will incorrectly rescale the map if the scene changes from a narrow to an open area or vice-versa. Using a dynamic rescale factor based only on the previous frames or using the height of the camera above the ground plane as reference would serve as more robust alternatives.

4.3 Real-Time Performance Optimization

To ensure the pipeline operates in real-time, meaning without the need of buffering images, we implemented comprehensive performance optimizations.

Across the entire pipeline, we replaced iterative loops with fully vectorized NumPy operations and ensured consistent use of datatypes to eliminate unnecessary recasts.

For the computationally intensive Bundle Adjustment, we implemented the analytical Jacobian for the reprojection error, eliminating the latency associated with numerical differentiation, and exploited the problem's **sparsity** using `scipy.sparse` matrix structures to minimize solver overhead.

4.4 Algorithmic Deviations

We introduced two significant deviations from the recommendations in the project statement to improve system robustness:

Non-Linear Pose Refinement

The project guidelines suggest refining the P3P pose estimate using a Direct Linear Transform (DLT) on all inliers. However, DLT minimizes an algebraic error that does not necessarily correspond to the optimal geometric solution. Instead, we implemented a motion-only bundle adjustment (detailed in Section 3.3). By minimizing the robust reprojection error directly using the Levenberg-Marquardt algorithm, we achieve tighter alignment and reduced trajectory jitter compared to the linear algebraic solution.

Spatially Uniform Replenishment

Standard feature detection often clusters points in high-contrast areas, leaving low-texture regions unconstrained. Deviating from the standard global detection approach, we implemented a grid-based "bucketing" strategy (Section 3.5). We divide the image into cells and only replenish features in specific cells where the density of active tracks falls below a threshold. This ensures a uniform spatial distribution of landmarks regardless of the mask radius we chose.

5 Results

The complete videos of the results are available on Polybox [3] and on YouTube [4].

5.1 Public datasets

5.1.1 Parking

In the parking dataset, having a lot of varying depths, the scale mitigation was turned off. This helped to avoid unwanted drift and to eventually get a consistent trajectory of the entire movement (Figure 1). It can be noticed here that the pipeline initializes faster than the other datsets (after 3 frames against the 7 of kitti) and it is due to the purely horizontal movement that allows to achieve a good parallax angle faster.

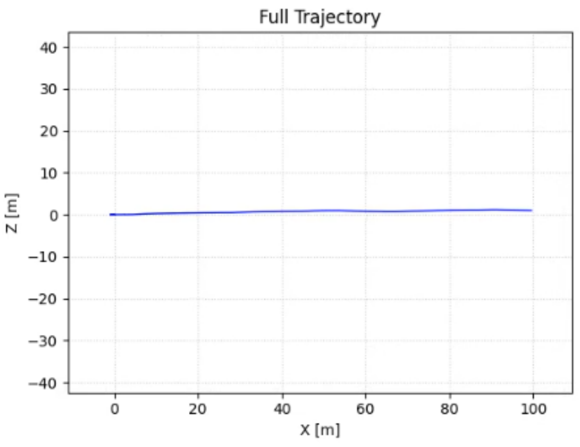

Figure 1: Full trajectory of the Parking dataset

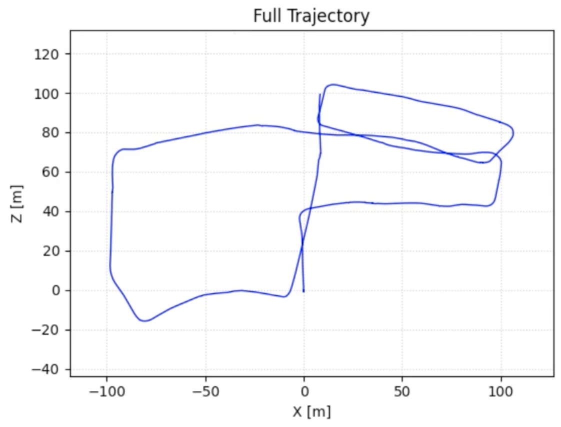

Figure 2: Full trajectory of the Kitti dataset

5.1.2 Kitti 05

After tuning the parameters of the pipeline, we achieved a good global trajectory with minimal scale drift, as shown in Figure 2. The scale drift mitigation plays a crucial role here, since without it we noticed a locally consistent trajectory until around 800 frames followed by a scale collapse that barely allowed trajectory tracking. The trajectory would eventually stay still because of the too little scale.

The achieved scale is still smaller than the ground truth absolute scale, but its amplitude stays constant. In order to achieve an absolute scale correctness, an implementation of a detection of a known object would be necessary to extract the right dimensions.

Bundle Adjustment increases the local consistency of the trajectory, allowing a globally consistent tracking.

It can be noted that all the moving cars in this dataset were correctly rejected and the tracked keypoints on moving object were considered as outliers, therefore without violating the "static world" assumption.

5.1.3 Malaga

The trajectory obtained for Malaga, shown in Figure 3, is locally consistent. Global consistency is not optimal since we observed a wide left turn where the ground truth trajectory is straight. This behavior was observed after the use of Scale Drift Mitigation on this dataset. However, without the use of this feature, the scale collapsed regularly after the first U turn (see Figure 4). Therefore, we decided to keep Scale Drift Mitigation active, thus avoiding early scale collapse but sacrificing global consistency.

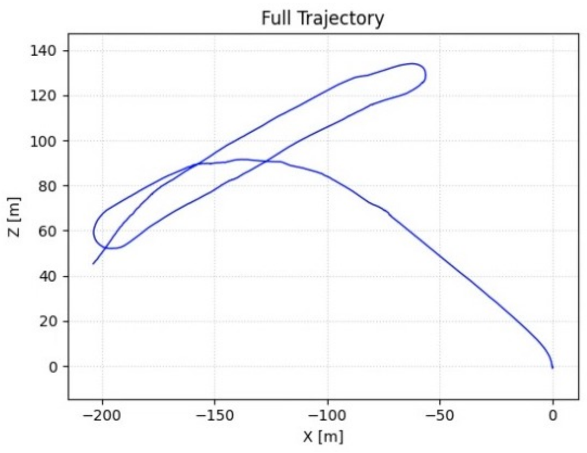

Figure 3: Full trajectory of the Malaga dataset with Scale Drift Mitigation

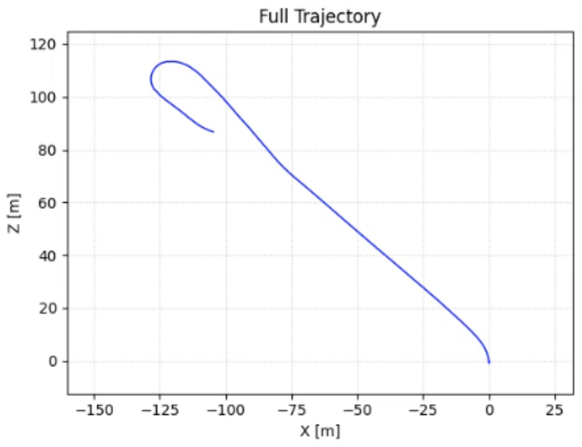

Figure 4: Full trajectory of the Malaga dataset without Scale Drift Mitigation, exhibiting scale collapse

5.2 Personal datasets

Both personal datasets were capture on an iPhone 14 and use the same set of tuning parameters. One dataset was taken on the polybahn, while the other around the block in Sonneggstrasse. The dataset are then exported at a resolution of 768 x 432 pixels and different FPS.

The results on both datasets show a good local and global consistency without the need to manually tune parameters between datasets, if these are taken with the same camera.

5.2.1 Polybahn

The right curve done by the descending wagon is clear and defined. The tracking and RANSAC algorithm correctly rejected the vast majority of the candidates landmark on the ascending wagon. Towards the end, there is a slight right turn that deviates from the physical trajectory. This could arise from the use of Scale Drift Mitigation, similar to what happens in Malaga, since the depth of the scene

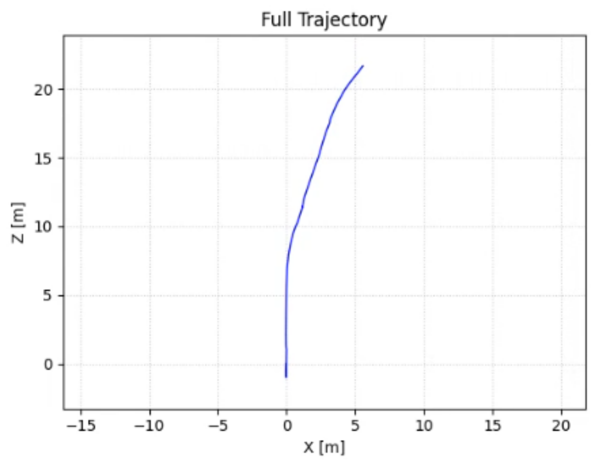

Figure 5: Full trajectory of the Polibahn descent dataset

suddenly changes.

5.2.2 Sonneggstrasse block

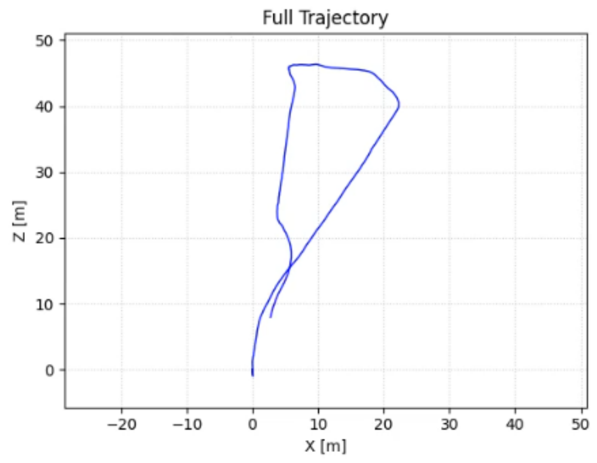

Figure 6: Full trajectory of the walk around the block in Sonneggstrasse dataset

The walk around the block results in a close trajectory even without the implementation of loop closure code, demonstrating the robustness of the scale mitigation. Moreover, bundle adjustment helps to refine the trajectory even with sharp turns like the one on the top-left of the trajectory map.

5.3 Performances

At the end of every pipeline execution, a file containing timing statistics for each step is generated, including:

- Mean execution time (ms)
- Standard deviation (ms)
- Minimum and maximum execution times (ms)
- Processed frames per second (FPS)

The performance was evaluated on an Acer Aspire 7, running Windows 11 with 32Gb RAM and Intel Core i7-10750H (2.60GHz).

5.3.1 Parking

Table 1 shows the performances achieved on the parking garage dataset. The average processing speed was 20 FPS, which should be sufficient to process the images without buffering them. We didn't find any official data on the dataset frame rate, but this consideration follows the low speed observed during the motion.

Table 1: Performance Metrics for Parking

	mean	std	min	max	FPS
1. Prepro	0.62	0.09	0.44	1.03	1621.76
2. Track	5.81	2.09	2.43	18.8	172.19
3. PoseEst	5.22	1.68	2.61	28.65	191.7
4. Triang	4.81	1.93	0.82	12.97	207.92
5. LocalBA	27.07	38.14	0.01	274.89	36.93
6. Replen	5.93	1.09	3.6	21.87	168.62
Total	49.48	37.95	17.13	295.53	20.21

5.3.2 Kitti

Table 2 shows the performances achieved on the Kitti 05 dataset. The average processing speed was around 15 FPS, which is above the 10 FPS at which the dataset was recorded. Therefore, the pipeline is able to process the images without buffering them

Table 2: Performance Metrics for Kitti

	mean	std	min	max	FPS
1. Prepro	0.76	0.1	0.61	1.52	1311.56
2. Track	6.91	2.62	3.76	25.17	144.72
3. PoseEst	28.74	26.65	3.04	97.6	34.79
4. Triang	9.24	4.32	0.6	28.48	108.28
5. LocalBA	11.41	4.42	0.02	37.95	87.63
6. Replen	8.82	1.13	5.66	20.75	113.39
Total	65.91	25.83	28.11	173.26	15.17

5.3.3 Malaga

The Malaga Urban dataset was recorded at 20 FPS and our pipeline reaches 17 FPS, as shown in Table 3. However, deactivating Bundle Adjustement will raise the mean FPS count to 23 FPS, thus making the pipeline run without the need of buffering images. To maintain consistency with the other results, we decided to run and present the tests with the local BA activated.

Table 3: Performance Metrics for Malaga

	mean	std	min	max	FPS
1. Prepro	0.78	0.06	0.66	1.43	1279.6
2. Track	7.91	2.67	4.21	22.84	126.38
3. PoseEst	7.41	6.02	2.7	50.2	134.99
4. Triang	19.13	5.13	6.3	44.86	52.26
5. LocalBA	15.06	6.07	0.02	53.39	66.41
6. Replen	8.23	1.48	4.97	19.04	121.45
Total	58.55	11.0	33.8	131.47	17.08

5.3.4 Polybahn

The Polybahn dataset was exported at 10 FPS and our pipeline performed at 15 FPS, as shown in Table 4, thus eliminating the need of image buffering.

Table 4: Performance Metrics for Polybahn

	mean	std	min	max	FPS
1. Prepro	0.59	0.07	0.43	1.04	1706.42
2. Track	6.79	2.45	3.79	18.04	147.37
3. PoseEst	7.95	6.13	2.82	75.68	125.79
4. Triang	18.93	5.91	5.05	40.23	52.83
5. LocalBA	26.18	12.9	0.02	86.05	38.19
6. Replen	5.6	2.11	0.0	12.44	178.62
Total	66.05	13.64	35.67	128.85	15.14

5.3.5 Sonneggstrasse block

As shown in Table 5, our pipeline reaches a performance of almost 15 FPS on this database, which was recorded at 5 FPS. Therefore, the frame-rate was sufficient to eliminate image buffering.

Table 5: Performance Metrics for Sonneggstrasse block

	mean	std	min	max	FPS
1. Prepro	0.59	0.09	0.44	2.27	1683.65
2. Track	8.11	2.36	4.42	20.61	123.23
3. PoseEst	11.93	10.28	4.09	71.57	83.79
4. Triang	22.95	4.69	8.24	39.79	43.57
5. LocalBA	16.92	7.29	0.02	62.11	59.09
6. Replen	6.95	0.89	0.00	14.23	143.96
Total	67.50	12.92	33.37	129.91	14.82

6 Discussion

We noticed that the use of bundle adjustment can cause the trajectory to shift backwards in order to minimize the reprojection error, thus causing sometimes some jitter in the trajectory.

Before introducing scale drift mitigation, the pipeline used to come to a stop where all landmarks would collapse close to the camera and pose estimation would fail. Scale Drift Mitigation is not used in the Parking dataset because it caused scale drift and erratic movement. This is probably due to the fast change in scene depth going from the parking door and then passing through the empty space. Scale Drift Mitigation worked well in the section where the camera passes close to the cars, where the scene depth is more homogeneous.

Finally, even if we used a Dockers Container to ensure a common operating system to test and evaluate the pipeline, we noticed some difference in performance when using different machines. Although not affecting the local accuracy significantly, these differences sometimes resulted in a variation in the global consistency of the pipeline. We pinpointed this difference to internal optimization in OpenCV functions that use CPU kernel and therefore cannot reproduce the same exact result in different machines. However, we did not notice major global differences on our own recorded datasets, ensuring consistent results our recorded data.

7 Future Work

While we are proud of the results that we achieved during this project, we recognize that some additions would make this pipeline even more robust and precise. One of these additions would be a proper loop closure algorithm. This optimization would use new observation of features already recognized in the past to refine trajectory and landmarks in between. This would allow to attain a closed trajectory that is not dependent on the right choice of parameters.

Moreover, even if our current Scale Drift Mitigation algorithm performs well in our recorded datasets, it sometimes reduces performances on the public datasets. Since our datasets are recorded with an iPhone, inertial measurements could be extracted and used to perform an optimization where the scale of the trajectory is optimized using these measurements. The pipeline would be therefore for Visual Inertial Odometry.

8 Contributions

The development of the Visual Odometry pipeline was a collaborative effort, with all team members contributing to the system architecture and debugging. The specific responsibilities were distributed as follows.

System Architecture and Bootstrapping: Dávid designed the central `VoRunner` and the Markovian `VoState` data structure, ensuring efficient data flow between modules. He made as great effort to optimize the performance of the pipeline to achieve real-time performance. Antonio, Alessandro, and Tim implemented the `initialization` module, including robust parallax-based keyframe selection and the integration of the 5-point algorithm for bootstrapping the map.

Tracking and State Estimation: Tim focused on the front-end tracking, implementing `keypoint tracking` with bidirectional optical flow and the dynamic grid-based `replenishment` strategy. Alessandro developed the `pose estimation`, specifically the P3P-RANSAC integration and the custom motion-only non-linear refinement described in the deviations section.

Optimization and Mapping: The `triangulation` module, including the scale preservation heuristic and cheirality filters, was implemented by Tim and Alessandro. For the Local Bundle Adjustment, Antonio and Tim worked on implementing and refining logic to ensure real-time performance.

Evaluation and Analysis: Antonio and Alessandro collaborated on the recording and camera calibration for the custom datasets, while all authors contributed equally to the tuning and evaluation of the pipeline.

During this project, generative AI was employed as a coding assistant for error detection, to support debugging and for rapid prototyping of new features. All generated responses were reviewed and manually tested before being integrated into our project. The core logic of the pipeline was developed exclusively by us, using the available material from lectures and exercises, without the use of generative AI.

References

[1] Robotics and P. Group, *Course website*, Accessed: 2026-01-05, University of Zurich. [Online]. Available: `https://rpg.ifi.uzh.ch/teaching.html`.

[2] Dávid Tokár, Tim Wyss, Alessandro Pietrogiovanna, Antonio Zecchin, *Vision_ odmetry_ pipeline*, `https://github.com/DavidTokar12/vision_odometry_pipeline`, GitHub repository. Accessed: Jan. 5, 2026, 2026.

[3] Dávid Tokár, Tim Wyss, Alessandro Pietrogiovanna, Antonio Zecchin, *Results on polybox*. [Online]. Available: `https://u.ethz.ch/ObMXU`.

[4] Dávid Tokár, Tim Wyss, Alessandro Pietrogiovanna, Antonio Zecchin, *Results on youtube*. [Online]. Available: `https://www.youtube.com/playlist?list=PL-waHff4z-BM604fTs38r3McEt11LEMOO`.