VROOM: Visual Robotic Odometry, Only Monocular

Francesco Banelli (24-953-804)
Facundo Garcia Cardenas (24-940-389)
Alessandro Petitti (24-935-751)
Ryan Slocum (24-936-510)

Vision Algorithms for Mobile Robotics
University of Zürich
January 6th, 2025

Abstract. Mobile robots often require the ability to navigate and map complex environments. Among the various sensors available, the camera represents a cost-effective, flexible, and robust option for this purpose. In the course *Vision Algorithms for Mobile Robotics*, we explored the foundational components necessary for constructing a monocular visual odometry (VO) system. Such a system enables a robot to estimate its ego-motion and create a sparse map of its surroundings. This report details the development of our VO pipeline and qualitatively evaluates its performance across the provided datasets.

1 Introduction

Visual odometry (VO) is a critical component for perception and state estimation in mobile robotics. Throughout this course, we have studied various algorithms and techniques underlying monocular and stereo VO, as well as Visual-Inertial Odometry (VIO) systems. For our project, we developed a monocular VO system, adhering closely to the guidelines outlined in the project description. This system enables a robot to estimate its motion and map its environment using only a single camera.

2 Methods

Our VO pipeline is designed based on the framework provided in the course's project statement. The pipeline operates in several stages:

1. **Initialization:** We begin by processing the first two frames of the dataset to establish initial pose estimates and landmark positions.
2. **Keypoint Tracking:** We use the Kanade-Lucas-Tomasi (KLT) optical flow method to track the locations of key points across frames.
3. **Pose and 3D Landmark Estimation:** We employ the RANSAC-based Perspective-n-Point (PnP) algorithm to estimate the camera's current position and the 3D position of the landmarks.

4. **Keypoint Management:** To maintain robust tracking, we periodically refresh the set of keypoints by validating candidate points tracked in the background.

The subsequent subsections provide a detailed explanation of each component of our algorithm.

2.1 Initialization

The initialization phase aims to provide accurate initial estimates of the camera's pose and the positions of landmarks within the environment. To achieve this, we used the first and third frames from the dataset, ensuring a sufficient baseline for reliable pose estimation.

The process begins with the Harris Corner detector [1], which identifies the salient features in the first image that are suitable for tracking. These features are then tracked through the subsequent two frames using KLT optical flow [2]. By establishing a significant baseline between the first and third frames, we can accurately compute the essential matrix using the OpenCV function `findEssentialMat`.

With the essential matrix, we determine the camera pose in the final initialization frame. Subsequently, we triangulate the tracked 2D keypoints to reconstruct their corresponding 3D landmarks. These initial pose estimates and landmark positions are integrated into the continuous operation loop. Here, we apply a Markov chain framework, as outlined in the project statement, to iteratively estimate the camera's subsequent poses and update the positions of keypoints and landmarks based on the current and previous images and the prior state.

2.2 2D-2D Correspondence

With the current and previous images at hand, the next step is to track the 2D keypoints from the previous frame to their corresponding locations in the current frame. We employ KLT optical flow for this purpose, which effectively estimates the new positions of the 2D keypoints. Additionally, the KLT optical flow function `cv2.calcOpticalFlowPyrLK` inherently filters out outliers, ensuring that only high-quality correspondences are retained for further processing.

2.3 2D-3D Correspondence

In this stage, we leverage the current 2D keypoints along with the previously established 3D landmarks to determine the camera's current pose relative to the world frame. We utilize OpenCV's `solvePnPRansac` function, which addresses the Perspective-n-Point (PnP) problem by estimating the camera's rotation and translation vectors. The RANSAC (Random Sample Consensus) component effectively identifies and discards outlier correspondences, enhancing the robustness and accuracy of the pose estimate.

2.4 Candidate Points

As the camera moves and the environment changes, some keypoints may become occluded or drift out of view, leading to the loss of their correspondences. To maintain robust tracking, it is essential to periodically update our set of keypoints. To achieve this, we maintain a separate list of candidate points. Each candidate point comprises the current 2D feature location and metadata about its initial observation, including the 2D position and the camera's pose at the time of the first detection.

The process of managing candidate points mirrors that of the keypoint tracking pipeline. Initially, we employ the Harris Corner detector to identify additional salient features within the current frame. These new features are added to the candidate list for potential tracking. Subsequently, we apply optical flow to track these candidate points across frames. For each candidate point, we assess whether it satisfies a baseline threshold, specifically checking if there is a sufficient angular difference between its initial observation and its current position. Points that meet this criterion are triangulated to determine their 3D locations and are then promoted to the main keypoint list for ongoing tracking in subsequent frames.

2.5 Key Deviations from Statement Recommendations

Because we found that the pipeline laid out in the project statement was generally sufficient to achieve our goal of locally-consistent tracking, we did not make any large changes to the recommended structure. We did, however, find that we needed more filters than simply a threshold angle α between candidate points. In addition, we filtered candidate points to ensure that they did not land too near the currently tracked keypoints or the other tracked candidate points. In the parameter tuning process we also realized that we needed to have both sparse features to ensure robustness and high quality features to guarantee accuracy. We took care of this by finding a way to alternate between different parameters on the Harris Detector to ensure we were extracting different types of features along the way.

3 Development

3.1 Roadmap

To begin the project, we developed a flowchart (see Figure 1) to visualize the Markov-chain process outlined in the project statement. This visualization enabled us to decompose the overall task into manageable blocks, facilitating a systematic approach to development.

Our initial focus was on implementing classes responsible for processing video frames and visualizing the results. This foundational work allowed us to effectively debug and validate individual stages of the pipeline as we progressed. Following this, we constructed a basic version of the VO pipeline, establishing

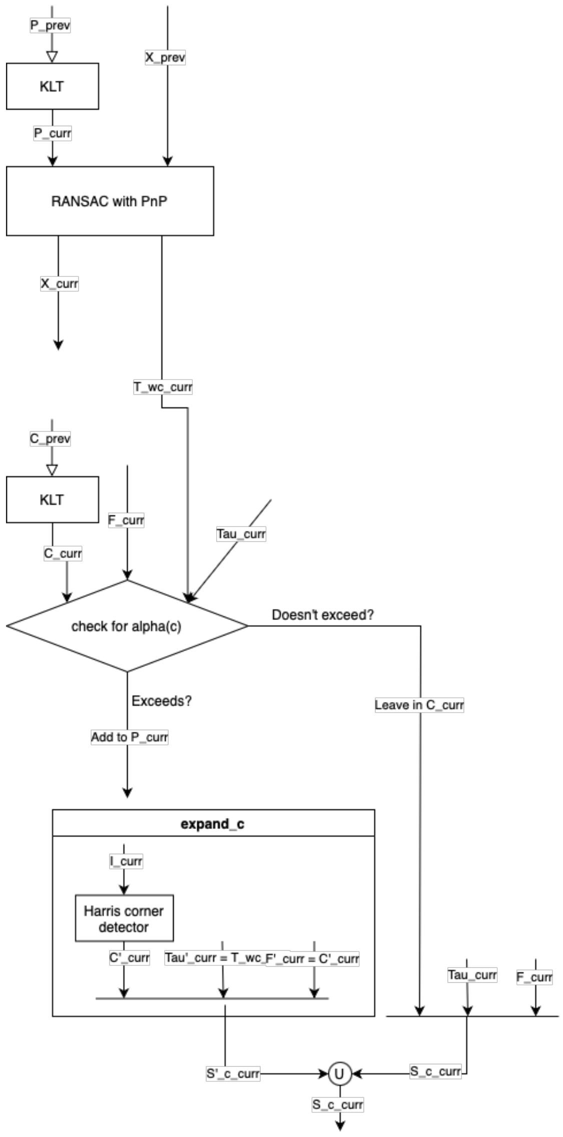

Fig. 1. Visual representation of our algorithm, as adapted from the project statement

the core loop structure necessary for sequential processing. This preliminary implementation served as a testing ground for integrating and refining the pipeline's individual components.

Throughout development, extensive debugging was essential. We addressed various issues, such as incorrect assumptions about input and output data types. Once the pipeline demonstrated reasonable behavior on simpler datasets, we proceeded to fine-tune parameters to enhance keypoint detection and ensure that the predicted trajectories aligned with our expectations.

3.2 Problems faced during development

During the bootstrapping phase on the "Parking" dataset, we observed that very few keypoints were detected, particularly during the transition from an outdoor to an indoor view. This issue was traced back to the parameters of the OpenCV's `cv2.goodFeaturesToTrack()` function, which we used for corner detection. Specifically, the function's default settings were too restrictive, limiting the number of detected features in certain frames. As noted in the official OpenCV documentation, the function discards any corner whose minimal eigenvalue falls below the highest cornerness response function, scaled by a parameter called QUALITYLEVEL. This meant that the extremely strong edges between the inside and outside of the scene, which had steep gradients, effectively set the threshold too high for detecting inside corners. By adaptively tuning the detector parameters for each analyzed frame (e.g., maximum number of features and minimum quality threshold), we were able to improve the detection rate.

Furthermore, achieving a uniform distribution of high-quality keypoints across the image proved challenging. Often, the Harris corners we detected and added to the candidate set were concentrated in two or three highly salient regions. This clustering significantly compromised the overall tracking quality, as it reduced the diversity of viewpoints and increased the likelihood of losing track of keypoints in less prominent areas. To mitigate this, we tweaked the Harris corner detector parameters to enforce a more even spatial distribution of detected keypoints, thereby enhancing the robustness of our VO pipeline. In addition, lowering the minimum feature quality allowed the detection of features in less salient areas of the image. These tweaks allowed us to collect much more diverse candidate points, leading to better tracking over time.

Finally, we noticed that the requirements of each dataset differed significantly from one another. For example, the parking dataset needed to deal with outdoor to indoor transitions, while the Kitti dataset presented several turns. This issue necessitated using a different parameter set for each dataset, managed by a dictionary of parameters. On top of that, the requirements for the Harris corner detector and KLT for initialization and ongoing operation were also different. Working with a parameter dictionary facilitated testing different configurations for each dataset, such as comparing dense and sparse feature tracking, diverse quality thresholds, and baseline requirements, among other things.

4 Results

To evaluate the performance of our VO pipeline, we tested it using three distinct self-driving datasets: the KITTI dataset [3], the Parking dataset, and the Malaga dataset [4]. Each dataset presents unique challenges, allowing us to assess the robustness and adaptability of our system under varying conditions.

4.1 KITTI

The behavior on KITTI results satisfactory in maintaining very good local consistency all the way through two curves, as shown in Fig. 2. The system loses sense of scale after around 600 frames due to the prevalence of stationary features and much of the view being covered by the shadow of a house. We concluded that further parameter tuning could improve the performance in this scenario.

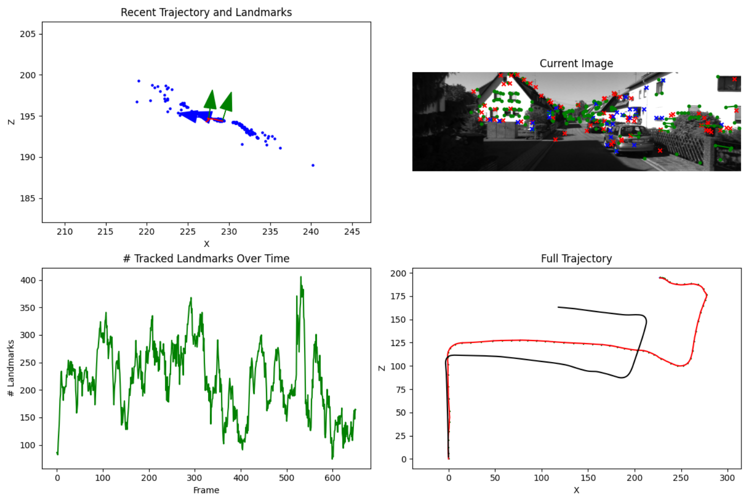

Fig. 2. Final trajectory of Kitti dataset (up to 650 frames).

4.2 Parking

When applied to the Parking dataset, our VO pipeline effectively handled the transition between exterior and interior views, as shown in Fig. 3. Despite initial challenges with keypoint detection, parameter adjustments enabled consistent tracking and pose estimation. We noted that most of the drift experienced in the trajectory comes from the erroneous scale estimation, which is carried over from the initialization step. As shown in Fig. 4, by initializing the scale factor using the ground-truth data, the system achieves excellent performance on this dataset.

Thus, a stereo setup, an IMU, or any other method to accurately recover scale would significantly improve performance. Lastly, we concluded that the vertical drift experience in the last section of the sequence comes from the sudden shift from nearby features, such as parked cars, to the far-away and scattered features on the garage wall, but the drift fits our expected performance.

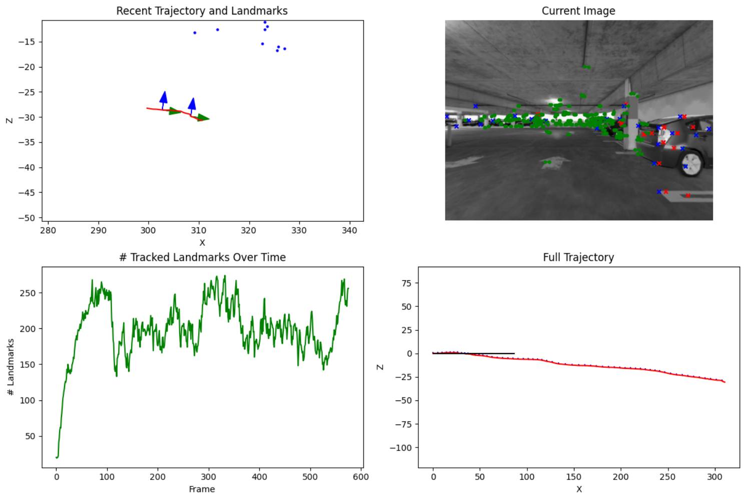

Fig. 3. Final trajectory of Parking dataset.

4.3 Malaga

Even though we don't have access to a ground truth to compare it with, the performance on Malaga is very good, as shown in Fig. 5. It loses some sense of scale after losing sight of a big structure but local coherence is sufficiently preserved. This can be noted in the drop in the number of tracked features and could be improved with further parameter tuning. It's worth noting that the parameters used on KITTI transfered very well to Malaga.

5 Conclusion

In this project, we successfully developed a monocular visual odometry (VO) pipeline capable of estimating ego-motion and constructing a sparse 3D map using only an RGB camera. Our implementation followed a structured approach, encompassing initialization, keypoint tracking, pose estimation, and keypoint management. Testing across the KITTI, Parking, and Malaga datasets demonstrated the pipeline's effectiveness in diverse scenarios, highlighting its robustness and adaptability.

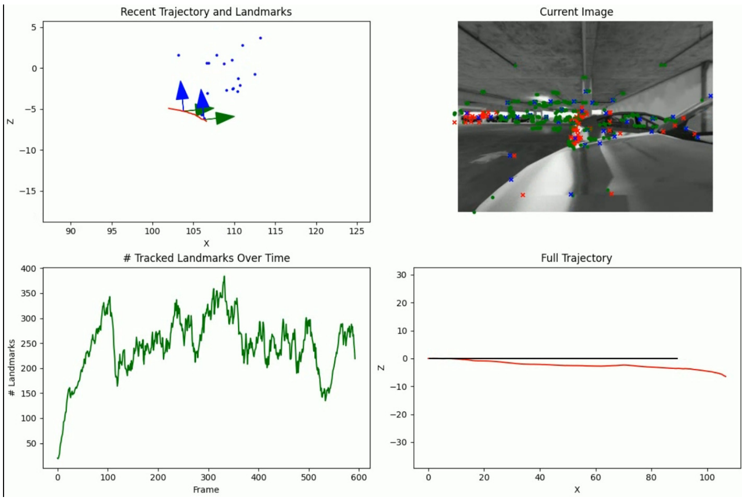

Fig. 4. Parking dataset with ground truth scale initialization.

While the VO system performed well in controlled environments, challenges such as uneven keypoint distribution and sensitivity to parameter settings were encountered. Addressing these issues through improved keypoint detection strategies and adaptive parameter tuning could further enhance performance.

Although our VO system loses tracking in particular situations, its local performance over sequences of a few dozen frames remains appreciable, and the drift is comparable to what would be expected of such a monocular, purely vision system. We believe that with an additional loop-detection and closure component, the drift would certainly be correctable in mobile robotics applications.

Overall, this project provided valuable insights into the practical implementation of visual odometry algorithms and underscored the importance of meticulous system design and parameter optimization in achieving reliable performance in real-world applications.

A Links to Screencasts

KITTI: youtu.be/HQ9_9UxqoJw
Parking: youtu.be/eZebqgfcbBM
Malaga: youtu.be/ppkdubCo8aA

B Github Repository

Repository: https://github.com/fgarciacardenas/vamr-project

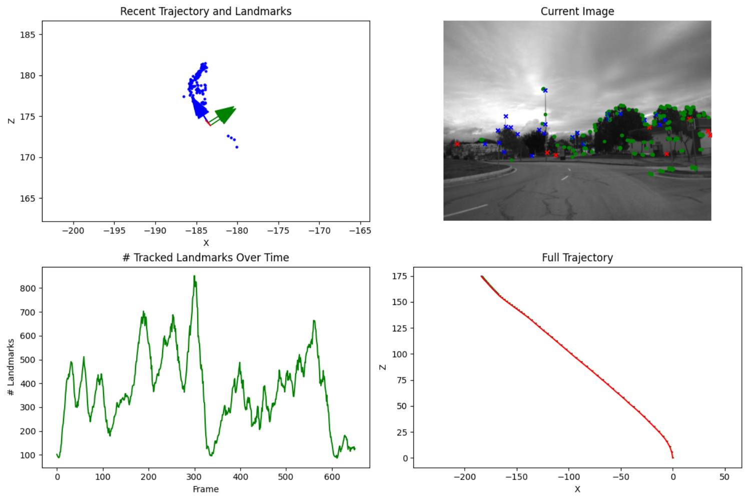

Fig. 5. Final trajectory of Malaga dataset (up to 650 frames).

C Author Contributions

Francesco Banelli: Outlined the visual representation of the algorithm, worked on the first implementation of the pipeline and did the final parameter tuning.
Facundo Garcia Cardenas: Worked on data processing, initialization procedure, parameter tuning, Docker setup, Github setup, and debugging.
Alessandro Petitti: Took care of: defining overall structure, the visualizer code, the bootstrapping code and debugging.
Ryan Slocum: Wrote initial continuous operation pipeline, debugged and tested continuous operation and initialization, wrote report template.

References

1. Harris, C.G., Stephens, M.J.: A combined corner and edge detector. In: Alvey Vision Conference. (1988)
2. Lucas, B., Kanade, T.: An iterative image registration technique with an application to stereo vision (ijcai). Volume 81. (04 1981)
3. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2012)
4. Blanco, J.L., Moreno, F., González-Jiménez, J.: The málaga urban dataset: High-rate stereo and lidar in a realistic urban scenario. International Journal of Robotics Research **33** (02 2014) 207–214