VISION ALGORITHMS FOR MOBILE ROBOTICS - HS 2016

VO PROJECT REPORT

Daniel Gehrig: 12-922-738
Raffael Theiler: 10-928-893
Nicolas Küchler: 14-712-129
Cyrill Halter: 13-928-171

Contents

1. Introduction

1.1 Monocular Visual Odometry

The following report describes a completely monocular visual odometry (VO) pipeline. It relies on the input of a single visual sensor to estimate a trajectory which is valid up to an unknown scale factor.

1.2 Overview

Before starting our VO pipeline by executing the file *main.m*, the tester may choose the data set and whether or not bundle adjustment and/or trajectory alignment to the ground truth should be performed. These parameters can be adjusted in the *Configuration Section* and are summarized in Table 1.1. Once the pipeline is running, an overview detailing the various operation parameters can be seen. A typical overview is shown in figure 1.1.

Table 1.1: Summary of control parameters

Parameters	Possible values	Description
dataset_id	0, 1, 2, 3	chooses between KITTI (0), Malaga (1), parking (2) and Vespa (3) data set.
bundle_adjustment	true, false	turns on bundle adjustment
align_to_ground_truth	true, false	turns on alignment to ground truth

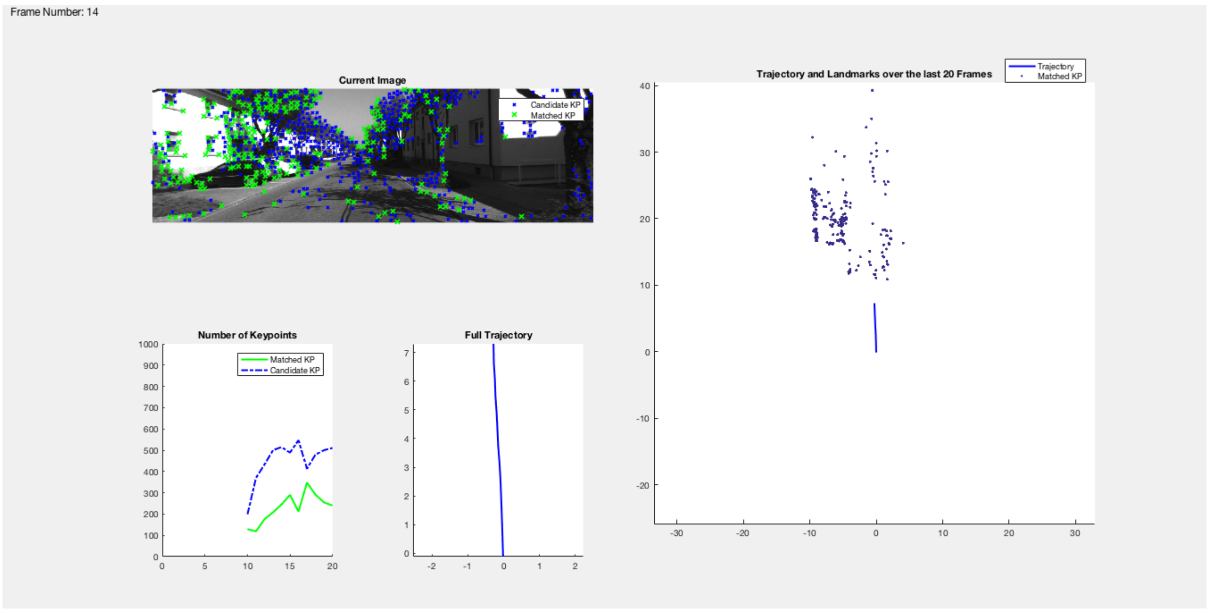


Figure 1.1: Overview of the operation parameters of the VO pipeline. The current frame can be seen in the top left with the matched key points in green and candidate key points in blue. The bottom left plot shows the number of tracked candidates and matched key points over the last 20 frames while the bottom middle plot shows the full estimated trajectory. The right plot shows the trajectory and tracked landmarks from the last 20 frames.

1.3 ADDITIONAL WORK

The implementation of the VO pipeline involved the tuning of several parameters which were vital to its stable performance. This was done by performing a quantitative simulation of the pipeline on the KITTI data set in order to find the settings that lead to the best trajectory estimate. For this a performance metric was defined to quantify the quality of the trajectories. A reasonable function was found by calculating the average trajectory deviation from the ground truth after alignment. More on this analysis can be found in section 3.3.
For the purpose of testing these ideal parameters we recorded an own video which we used to validate the existing VO pipeline. This is an important test because it guards agains over-fitting our parameters to the benchmark data sets. More information on the data set can be found in section 3.1.
Unfortunately scale drift is inevitable in a VO pipeline due to errors in the pose and landmark estimates. To combat this drift multiple views can be refined using bundle adjustment (BA). We integrated a form of sliding window and overlapping bundle adjustment into the existing pipeline in order to refine both structure and motion. The adjusted estimates are significantly more accurate but take longer to compute. For a more detailed discussion see section 3.2.

2. Functionality

The following points summarize the pipeline's functionality. Topics that are documented further in the code will be denoted by [⊟ filename] with the respective file name.

2.1 Bootstrapping

To initialize the pipeline, an initial point cloud is triangulated between two key frames (frame 1 and 3) and the respective camera homographies are estimated. Using the *Harris corner detector*, key points are extracted from images 1 and 3 and matched (using the SSD of the corresponding descriptor patches). In the following step, the camera homography of frame 3 is estimated by calculating and decomposing the essential matrix using the *eight-point algorithm* (since K is known). For robustness, *RANSAC* is performed with a reprojection error tolerance of 5 and 500 iterations. The resulting inliers are then used to calculate the final landmarks and pose [⊟ initializePointCloudMono.m] .

2.2 Frame Processing

Subsequent frames are processed in a Markovian fashion, meaning that information from the previous frame is sufficient to compute all necessary variables of the next frame [⊟ processFrame.m] .
When processing a new frame, key points (matched with landmarks) are tracked from the previous image to the next image using the MATLAB implementation of a *Kanade-Lucas-Tomasi tracker (KLT)* [⊟ propagageState.m] in a first step. Due to the large displacement of key points across images, 3 pyramidal levels and a patch size of 13 x 13 pixels is used.
In a second step, the new camera homography is computed using the new 2D-3D correspondences between matched key points and landmarks. This can be computed efficiently using the *P3P algorithm* in conjunction with *RANSAC* [⊟ localizationRANSAC.m] . Using only three points allows for few iteration cycles (500 cycles with a pixel tolerance of 5). Outliers and key points that are discarded by the *KLT* tracker are removed together with their corresponding landmarks.
Next, new landmarks are triangulated from candidate key point tracks which have been tracked over several frames. Tracks are discarded if they are lost by the *KLT* tracker. Triangulation is performed between the track end point and start point. New landmarks are triangulated asynchronously as soon as the angle between the bearing vectors at the track start point and end point exceeds 3 degrees. New landmarks are discarded if their reprojection error exceeds 5 pixels or if they are triangulated behind the camera [⊟ tryTriangulate.m] . However, if the number of mathed key points drops below 100 then these landmarks are no longer discarded.
This candidate loss through tracking and triangulation means that new candidate key points must be added continuously. 150 new candidate key points are extracted from each frame using a Harris corner detector. To achieve a more robust behavior, the *Harris score* is suppressed around existing candidate key points before feature extraction [⊟ suppressExistingMatches.m] . This ensures that the same key points are not added multiple times and enables a more uniform distribution of key points across the frame.

In order to make our VO pipeline more robust to scale drift, we added an adaptive bearing angle threshold that depends on the number of currently matched key points and defines the angle necessary for candidates to be added as key points, the threshold being lower when less key points are currently active.

3. Additional Features

3.1 Data Set

The additional data set was recorded in the 7th district of Zürich using a *Huawei Nexus 6p* smartphone strapped to the luggage rack of a *Vespa* motorcycle. Compared to the data sets provided to us it includes some additional difficulties that are handled by our VO implementation with varying success:

1. *Additional Camera Movement:* Due to the size and weight of the motorcycle, the camera is affected by a much larger amount of vibration originating from the motor. The same factor also causes the vehicle to be subject to a lot more squat (leaning backwards on acceleration) and dive (leaning forward on breaking) than the cars used in the KITTI and the MALAGA data sets. This, together with leaning into turns, results in more camera rotation along the pitch and roll axes.
2. *Traffic:* Another factor that adds to the complexity of our data set is traffic. Both pedestrian and vehicle traffic were present during the recording which includes two pedestrians walking by directly in front of the camera at some point. Additionally, two stops at red lights were made.
3. *Unknowns:* Preprocessing performed automatically on the video recording by the smartphone (e.g. image stabilization or autofocusing) adds an unpredictable factor that may affect the behavior of our VO pipeline.

The VO pipeline is robust to factors *1.* and *3.*. The estimated path remains smooth and retains the overall direction well. Factor *2.* however, is an issue to some degree: While the trajectory is not significantly affected by other vehicles, pedestrians passing in front of the camera lead to the loss of most of the key points causing the detection of erroneous movement.

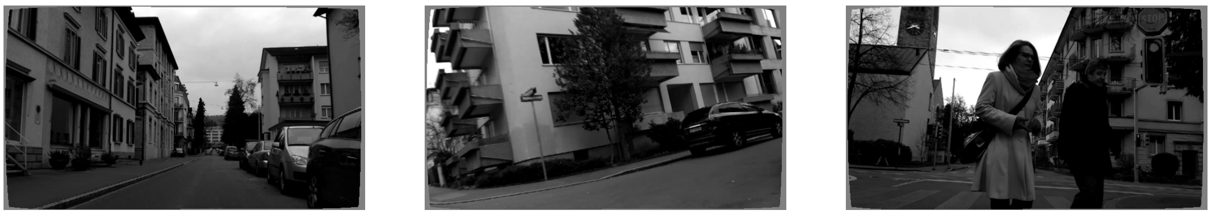
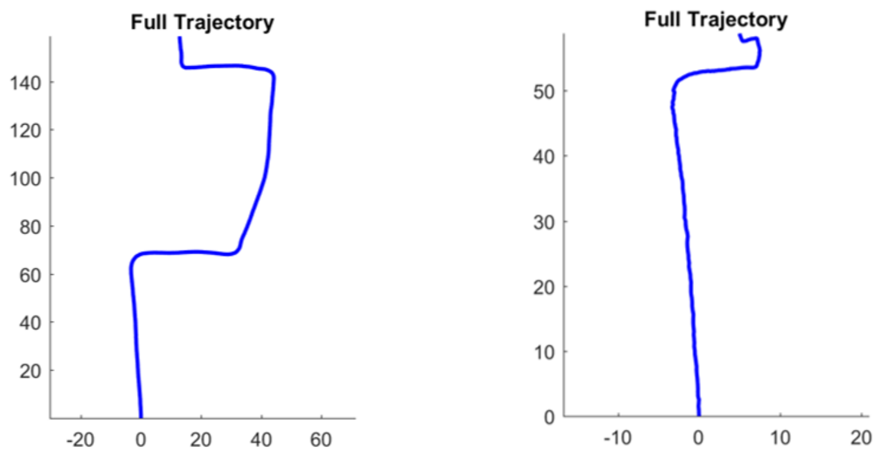

(a) a normal frame **(b)** roll while making turns **(c)** pedestrian traffic

Figure 3.1: Frames from the *Vespa* data set

3.2 Bundle Adjustment

As mentioned in section 1.3, the stand-alone VO pipeline suffers from a significant amount of scale drift. This means that along the trajectory, the average scale of the displacements changes due to numerical errors and noise accumulation. For continuous operation, a form of overlapping windowed BA was implemented to counteract this drift. [main.m, runBA.m] Every 3 frames, the trajectory and landmarks are adjusted and replaced over a segment of the last 5 frames. In order to ensure continuity, the adjusted point cloud and trajectory are constrained such that the last two orientations and locations of the last segment coincide with first two orientations and locations of the newly adjusted segment.

In cases where most key points are lost, BA switches to the *Levenberg-Marquardt algorithm*. In these cases the locations and orientations must be aligned manually.



(a) **(b)**

Figure 3.2: Comparison between bundle adjusted trajectory (a) and non-bundle adjusted trajectory (b) over 650 frames in the KITTI dataset with fixed seed rng(1).

Figure 3.2 shows that BA reduces scale drift significantly, smooths the trajectory and maintains the heading more reliably. These advantages come at the price of computing performance: VO with bundle adjustment uses about 40% more computing time than VO without bundle adjustment.

3.3 Ground Truth Alignment and Error Analysis

For a stable operation of the VO pipeline, optimized parameters had to be chosen to control detection and matching of new key points, triangulation etc. We determined these parameters by running a simulation [⊟ simMain.m, simTaskScheduler.m] on a benchmark data set (*KITTI*) over various parameter choices. The computed trajectories were aligned with the ground truth and total squared error in orientation (angle deviation) and the total deviation in location was determined for the first 500 frames [⊟ alignToGroundTruth.m] . Using these error metrics, optimal parameters could be identified which will be discussed further in section 4. An aligned trajectory can be see in figure 3.3. This trajectory alignment is done automatically if the flag *align_to_ground_truth* is set.

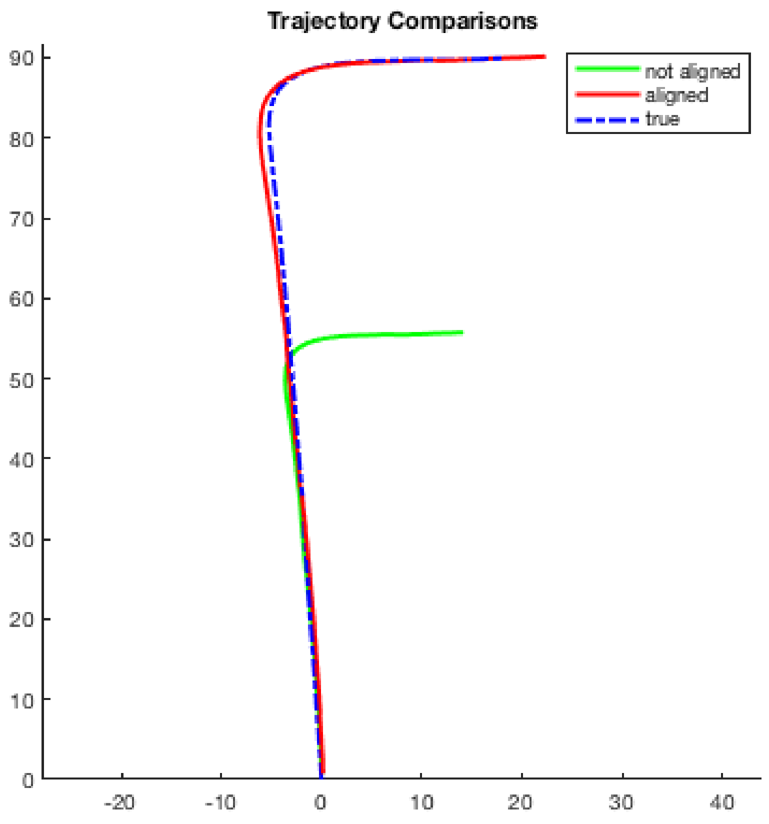


Figure 3.3: Comparison between the non-aligned (green), aligned (red) and true (blue) trajectory. The square deviation of orientation and location between the aligned and true trajectory was chosen as a suitable error metric.

4. Performance

4.1 Error Analysis of Parameters

In order to do a thorough optimization, a total of 15 parameters would have to be varied simultaneously. Brief descriptions of these parameters can be found in *parameters.txt* [⊟ parameters.txt] . Such a large number of variables, however, leads to an intractable search space for optimization. To curb the computational burden we segmented parameter selection into a preliminary simulation step [⊟ simTaskScheduler.m] which identified sensitive and non-sensitive parameters and a fine tuning step which investigated these parameters further. During the analysis the following error metrics were used to compare trajectories. The full analysis was done with fixed angle thresholds and without BA.

$$LE = \min_{R,t,s} \sum_i (sR \cdot p_{est,i} + t - p_{GT,i})^2$$

$$OE = \sum_i \text{rotMatToRotVec}(R^* \cdot R_{est,i} \cdot R_{GT,i}^T)^2$$

Where i is the index along the trajectory, LE is the location error, OE is the orientation error, p_{GT} and R_{GT} are the ground truth location and orientation, p_{est} and R_{est} are the estimated location and orientation and R^* is the argument for R when LE is minimal. The parameters s, R and t describe a generic similarity transformation. The function *rotMatToRotVec* transforms the rotation matrix into a rotation vector of the form $\phi = \theta\mathbf{n}$ [⊟ rotMatToRotVec.m] .

4.2 Preliminary Simulations

First, a sensible parameter configuration was guessed for each cluster. Then groups were fine tuned either manually or using simulation. Parameters which showed an increase in accuracy (deviation from ground truth, see chapter 3.3) and robustness (successful completion of run with varying other parameters) were selected at each step.

Good Harris parameters (patch size = 9 and $\kappa = 0.08$) where identified with low tracking loss. Moreover, the maximal number of candidates was determined to have low impact above 500. The sensitive parameters and ranges were narrowed down to 5 and are presented in table 4.1.

4.2.1 Parameter Fine Tuning

In this step the above parameters were varied in their ranges, while the other parameters were held constant and the OE and LE were recorded. 1000 different combinations were tested in total and their results are shown in an error histogram in figure 4.1. While the LE exhibits a large variance, the orientation error is much more concentrated in one peak. This shows that the location error is influenced much more by the variation of our parameters so from here on only the LE will be discussed. In table 4.1 the three parameter combinations that lead to the lowest LE can be seen.

To compare the robustness of these candidates it is necessary to see how well the algorithm performs for similar parameter combinations. For this purpose histograms showing the distribution of the 10 best and 200 best results for each parameter were constructed and are shown in figure 4.2 and 4.3. It can be seen that *triangulate_max_repr_error* is very robust as 50% (5 out of 10) of the top 10 share this value. It therefore makes sense that all top 3 candidates share this value. Also with *nonmaximum_suppression_radius* all top 3 share parameters with the majorities from the top 10. This trend is supported further by the top 200 histograms. The parameter *tracker_max_bidirectional_error* is shared by the top 3 and top 10. If we compare *critical_kp* we see that although two of the top three share the same value as the majority in the top 10 the best value is reached with a value of 100. This can be explained by the fact that in the KITTI data set the second curve contains a dark patch which makes it hard to track key points. If there is no safety net many runs will just fail because they have too few landmarks to perform localization. A histogram displaying all failed runs illustrates this point in figure 4.4. It shows that a parameter value of 0 leads to the majority of the exceptions which leads us to believe that this value is not robust although it may be optimal. The final parameter is *add_candidate_each_frame* which is robust to failure for all top 3 although not optimal for the best candidate.

Factoring in all available information, we arrived at the parameter configuration described in table 4.1 under *Chosen*.

Table 4.1: Parameter Fine Tuning: Best three parameter combinations and their respective location error (LE). LE_{max} : 23187, LE_{min}: 6664.9

Parameters	**Range**	**Best**	**2nd Best**	**3rd Best**	**Chosen**
LE		6664	6987	7207	
nonmaximum_suppression_radius	10:2:16	12	10	10	10
add_candidate_each_frame	50:50:200	150	200	200	150
triangulate_max_repr_error	[0.5, 1, 2, 5, 10, 20]	5	5	5	5
critical_kp	[0, 20, 50, 100]	100	0	0	100
tracker_max_bidirectional_error	[2.1, 5, 9999999]	5	2,1	5	5

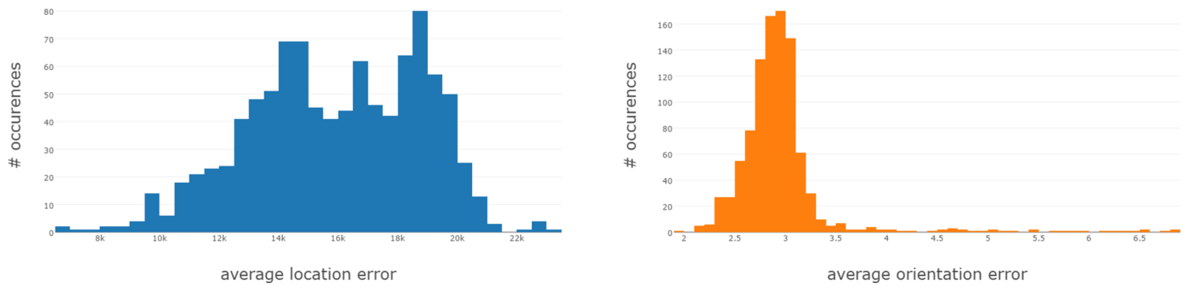


Figure 4.1: Distribution of the location error *LE* and orientation error *OE*. An outlier (23.79) was removed from the orientation errors. OE_{min} : 1.97, OE_{max} : 23.79, OE_{med} : 2.89, LE_{min} : 6664.9, LE_{max} : 23187.5, LE_{med} : 16059.0

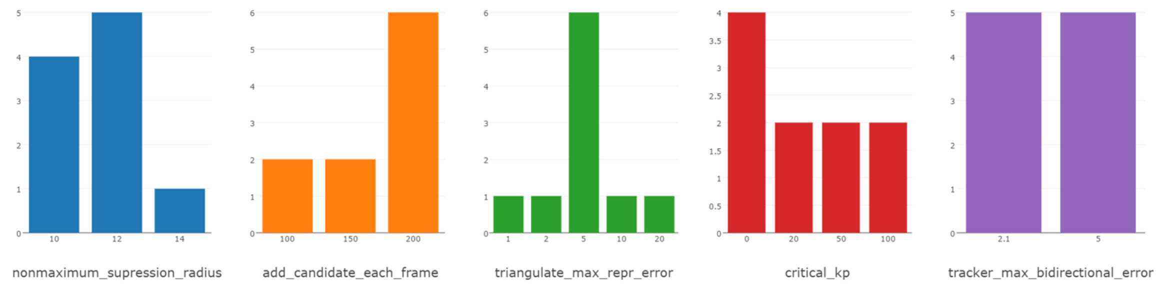


Figure 4.2: Parameter distribution of the best 10 results (15% of the dynamic range).

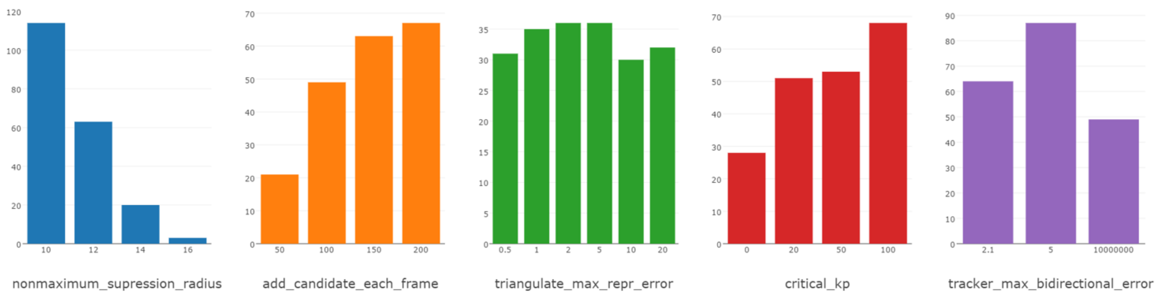


Figure 4.3: Parameter distribution of the best 200 results (40% of the dynamic range).

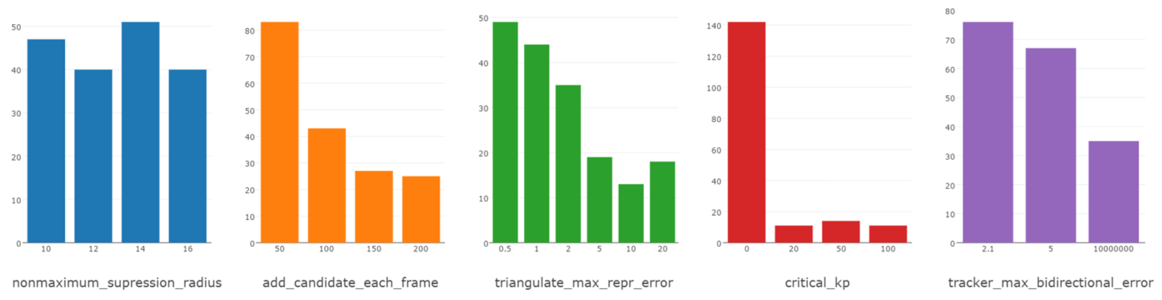


Figure 4.4: Parameter distribution of all runs that terminated due to excessive loss of key points.