Project Report

Visual Odometry Pipeline

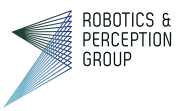
Marius Grimm, Amadeus Oertel, Toni Rosiñol

Titus Cieslewski, Henri Rebecq
Adviser

Prof. Dr. Davide Scaramuzza
Professor

Robotics and Perception Group
University of Zurich & Swiss Federal Institute of Technology Zurich (ETH)

2017-01



Contents

List of Figures

1 Introduction

As part of the Vision Algorithms for Mobile Robotics course at ETH & UZH in Zurich, we developed a Visual-Odometry pipeline that is briefly summarized in this report. Videos of our code showing its performance on the three provided datasets:

1. KITTI
2. Malaga_07
3. Parking

They can be found here https://www.dropbox.com/sh/jvh6bk42ok4fu2e/AADP0zjf7AOGADLhL-1Z81gna?dl=0.
The code has been developed in Matlab, versions R2016b / R2014b.

2 Initialisation

2.1 Monocular Initialisation

The monocular initialisation is a key module in the Visual-Odometry pipeline. It is ordered in the following way:

1. Find the 2D-2D correspondences between the chosen first two images.
2. Apply the 8-point Algorithm to estimate the Fundamental matrix combined with running RANSAC.
3. Check the validity of the estimated Fundamental matrix by either comparing the reprojection error or the distance to the epipolar line.
4. RANSAC returns a set of inliers of the current 2D-2D correspondences.
5. With the new set of inliers we can re-evaluate the final Essential Matrix E estimate.
6. The Essential Matrix E can then be decomposed into two rotation and two translation hypotheses for the pose of the first camera frame. This gives in total a set of four camera motion possibilities.
7. These hypotheses have to be disambiguated to choose the right camera rotation and translation.

Figure 1: 2D-2D Correspondences with KLT.

We have implemented two different approaches for finding the 2D-2D correspondences:

1. Exercise implementation of Harris descriptor and detector.
2. Implementation of the Kanade-Lucas Tracker (KLT).

An examplary output of the 2D-2D module can be seen in Figure 1. The correspondences between two images are indicated.

Due to efficiency reasons we decided to use the epipolar line distance as a measure to validate the estimated Fundamental Matrices. In theory, the reprojection error would have yielded better results (sacrificing efficiency); however, in our case the differences were neglectable.

3 Continuous Operation

The continuous operation part of the project performs basically three tasks:

1. Updates the correspondences 2D-3D to the new frame using KLT.
2. Localizes using the new correspondences by applying P3P plus RANSAC.
3. Stores new candidate keypoints and triangulates new landmarks, also with KLT.

The first part is done by using KLT as a feature tracker. The features that are correctly tracked are then used to estimate the new pose of the camera by running P3P together with RANSAC. This last step will also provide us with a list of inliers and outliers. This mask will let us discard the outliers while updating the 2D-3D correspondences with only inliers.

Next, the last observed keypoints that were in the candidates bin are also tracked in the new frame with KLT. If the tracking is successful, then they have to pass a triangulability check to see if they will result in a good triangulation. For this, we simply compute the angle between the bearing vectors of the current keypoint and the first time we saw it. If the triangulability test does not pass, the keypoint is simply stored for further processing in next frame. If it does pass the test, then the keypoint is used together with the current transform to triangulate a new landmark.

The landmark is then checked if it falls in front of the camera to verify that it is valid. Before, we also used to check if the reprojection error of this landmark was good, but now we just triangulate using

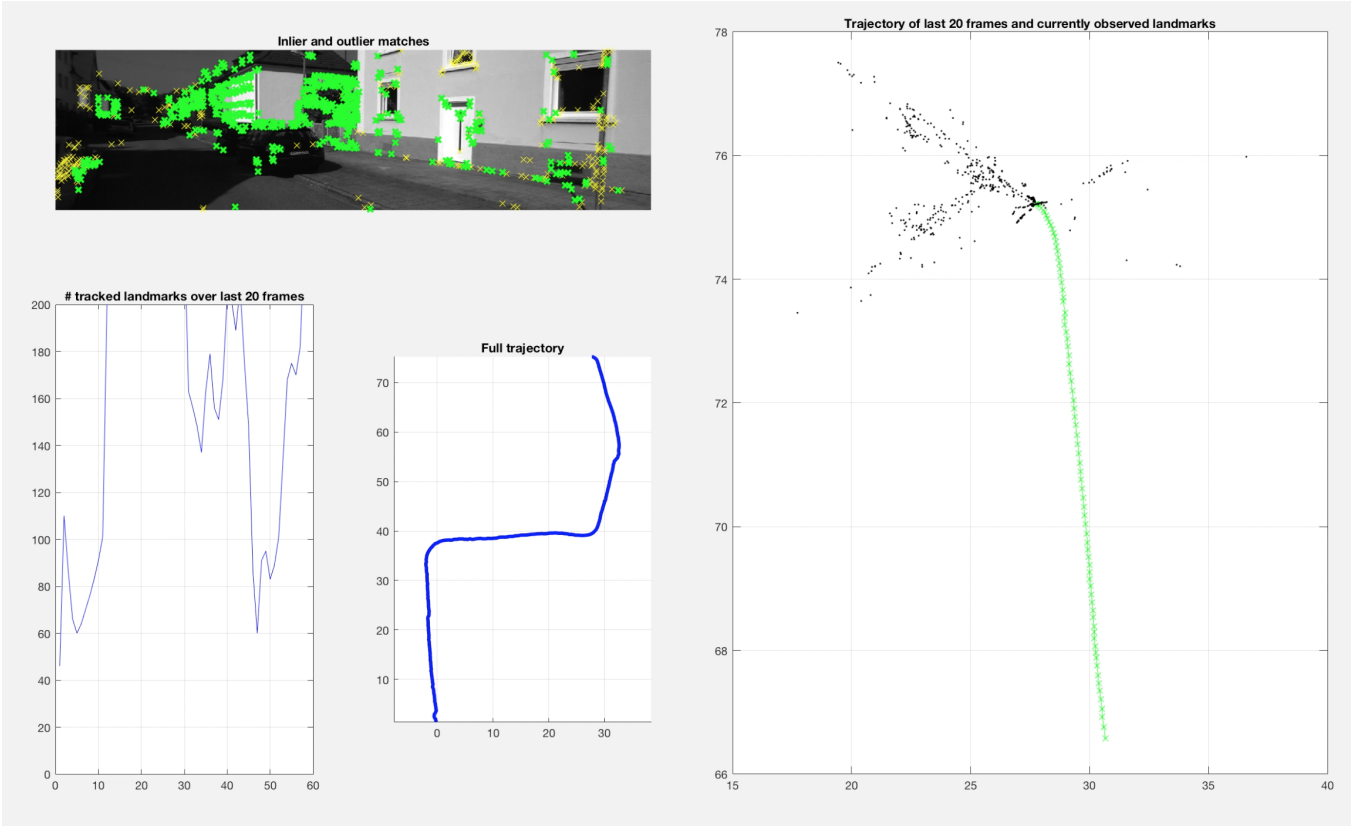


Figure 2: Trajectory with Landmarks and Keypoints.

the function used in the exercises (more information in the Quantitative Analysis Section 4.6.2). Finally we update the set of 2D-3D correspondences with this new landmark. In case the triangulation did not result in a satisfactory landmark, it gets discarded together with its corresponding keypoint.

4 Bonus Features

4.1 Plotting

The source code provides extensive plottings to help debug the pipeline. These are enabled by using the *debug_with_figures* flags on the different modules of the pipeline.

Moreover, in order to properly visualize the logic of the pipeline, we show a figure with the main output of the odometry estimation.
At the top left of the mentioned figure, it is shown the current frame that is being processed. On top of it we plot the keypoints triangulated in green, while in yellow we plot the ones that have not yet been triangulated but are potentially going to be triangulated.
The figures below the image present both the number of landmarks that are being tracked (left image) and the global trajectory estimated by the algorithm.
Finally, the figure on the right shows the last poses of the camera together with the last triangulated landmarks.

4.2 Automatic Selection of Frames for Initialisation

We have further implemented the Bonus Feature "Automatic Selection of Frames for Initialisation". The initialisation will try to find correspondences between different frames in the beginning of the datasets. If a pair of initialisation candidates has enough correspondence inliers the automatic selection tries to find the best pair which minimizes:

1. The average reprojection error and
2. the average epipolar line distance.

Once the minimizing pair was found, the monocular initialisation is recomputed with the new initialisation set of images.

4.3 Relocalisation

It can always happen in VO-pipeline that the errors introduced are getting too big and thus, localization is lost. In such a case, we need to be able to recover from the lost localization to further track the camera motion.
We have implemented a relocalisation state in the pipeline which covers this case. If the number of matched keypoints between two frames tends towards zero the track is lost. We reinitialise the track with two closeby frames with the help of the monocular initialisation.
However, the problem is that after reinitialisation also the scale gets reset. To counteract this we first need to recover the approximate scale where the track last ended. This scale is then reused after relocalisation.

4.4 Full Bundle Adjustment

In our VO pipeline, we implemented two different approaches of bundle adjustment in order to refine the obtained pose estimates and landmark positions. The user can choose between different methods by setting the flag BA_ to either Offline, Online, or None. BA_ set to Offline will perform full offline bundle adjustment to simultaneously optimize both, the trajectory of estimated poses as well as the 3D positions of triangulated landmarks, that have been recorded over a previously set number of frames.
This approach is based on the implementation provided with exercise 9.

If *BA_* is set to Online full online bundle adjustment on a certain number of (key-)frames is performed to update the current pose estimate and the landmark positions observed in the last m frames. However, as can be seen in *Video 5 Online BA*, the code is not working properly in the sense of meaningfully improving the current pose estimate. The idea is that the poses, triangulated landmarks, and observed keypoints of the last m frames are stored and can be passed in the format introduced in exercise 9 to Matlabs *lsqnonlin* solver which then iteratively adjusts the last m poses and the positions of landmarks observed in those m frames by minimizing the reprojection error.

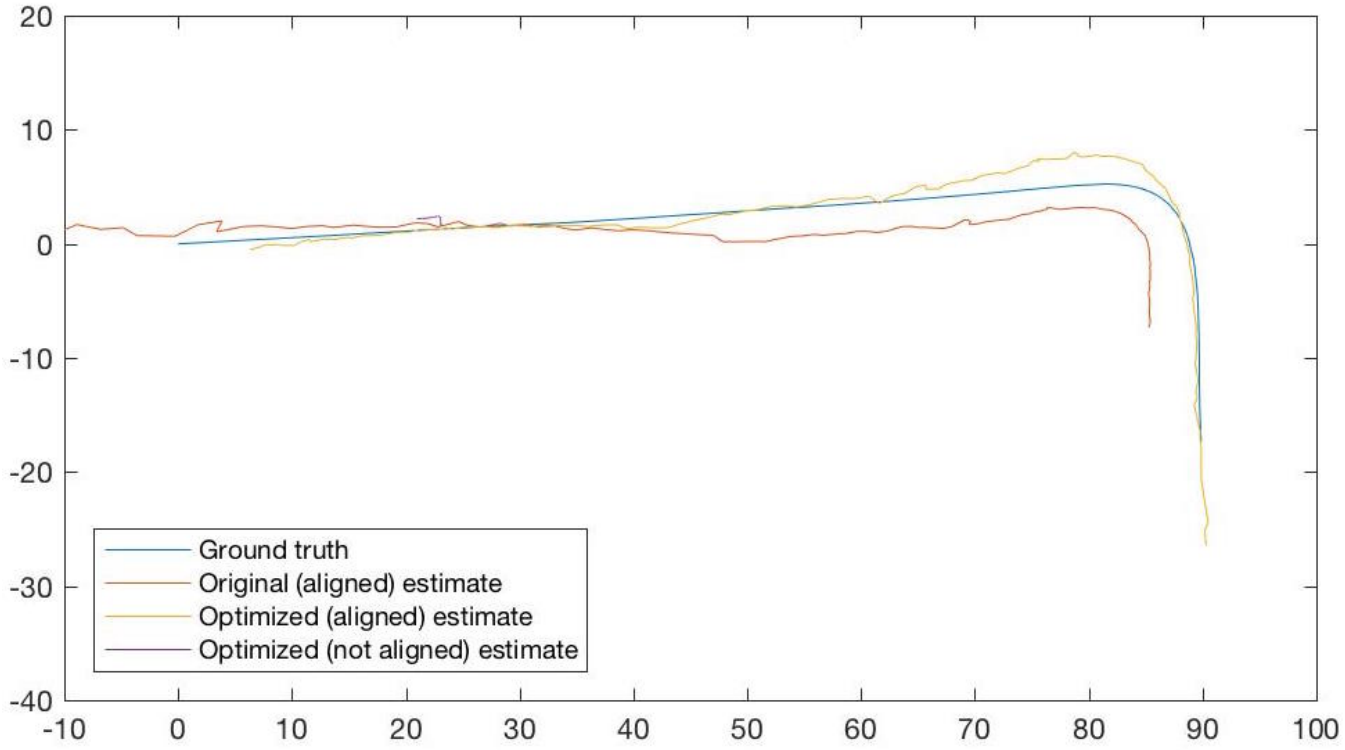


Figure 3: This Figure shows the original estimated poses (orange) for the first 150 frames of the KITTI dataset. To account for the scale ambiguity of monocular VO, the obtained trajectory has been aligned to the ground truth by means of a similarity transformation.

4.5 Calibrated Smartphone Camera and Own Dataset

In order to assess the robustness of our VO pipeline, we decided to record our own dataset. For this endeavour, we printed a checkerboard to calibrate the camera of an smartphone. We then used the calibrated camera to record the dataset. In order to achieve good results for the calibration, we made sure that we followed the following procedure:

1. Set focus and exposure mode of the smartphone to fixed.
2. Ensure that the checkerboard is over a planar surface and presents no light reflections.
3. Take frames of the checkerboard from sufficiently different angles and distances.

Once the calibration dataset recorded, we proceeded to find the intrinsics of the camera. As a first approach we used the Matlab Toolbox. Unfortunately, we could not get good calibration results. This was in part due to the fact that the interface requires you to click on the corners of the checkerboard in the image, and therefore limits the amount of images you can process in a given time. We also used OpenCV to retrieve the calibration matrix of the camera. Nonetheless, we found that the most user-friendly calibration tool was the one offered as a Matlab app named Camera Calibrator.

Using this last tool, we managed to retrieve approximately the same intrinsic parameters independently of the calibration images. Once calibrated, we processed the images to get a gray-scaled,

Figure 4: Block Matching Tracker.

resized and undistorted set of images. Then we fed the new dataset to our VO pipeline and checked the results with our ground truth. The ground truth was taken approximately with no special instrumentation other than a meter.

We encountered many problems while recording the dataset, to name a few: Motion blur when moving, changes of illumination on the scene, maintaining a constant focus, transferring images, having enough features on the scene, etc.

In the end, we found that monocular initialisation was having difficulties to output a sufficient amount of 2D-3D correspondences. This led the continuous operation part to stop after two frames on average. The reprojection errors on the triangulated keypoints were of the order of magnitude of tens of pixels, depending on parameters and bootstrap_frames.

In the end, we learned about different tools available for camera calibration and about the difficulties of recording a satisfiable dataset for visual odometry. Or seen from another perspective, we learned about the limits of our VO pipeline.

Our calibration files and images of the dataset are provided under the folder smartphone.

4.6 Quantitative Analysis

4.6.1 Keypoint Tracking via Block Matching versus KLT

Our first approach to keypoint tracking was to implement a block matching algorithm. After describing the keypoints by a block of pixels around it we simply computed the sum of squared differences to match different blocks. This approach, while simplistic, performed reasonably well and we were able to actually localize reliably. As seen in this Figure 4, Block Matching gives overall good results but fails to track some keypoints.
Nonetheless, knowing that the range of movement is constrained in our datasets (small translational displacements) we thought that it would be appropriate to use a more performant tracker in this situations such as the KLT algorithm.

The simple replacement of the Block Matching algorithm by the KLT algorithm offered an overall better performance of the estimation due to its sub-pixel accuracy when tracking keypoints. As

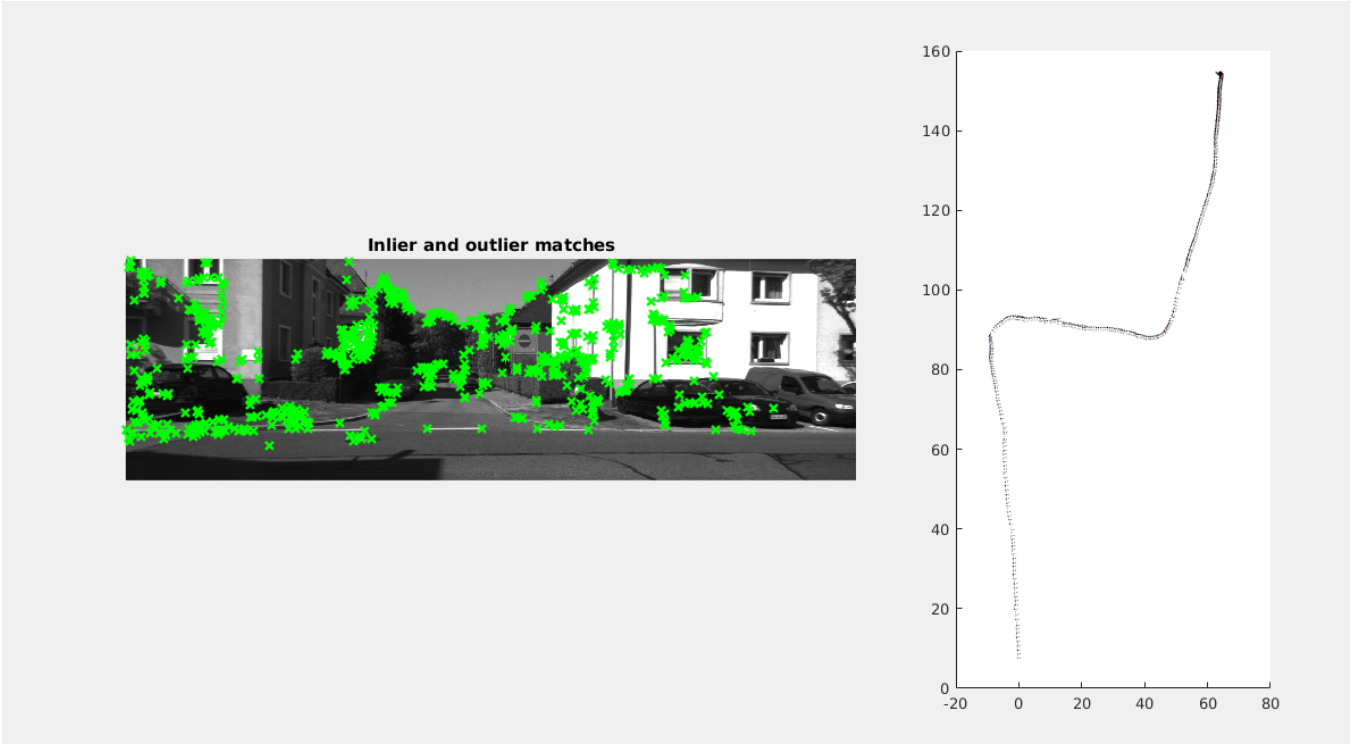


Figure 5: KLT Matching Tracker.

reference, this Figure 5 shows the performance of the KLT algorithm on the estimation of the pose.

4.6.2 Triangulation Functions

During the early development of the pipeline we decided to use the linear triangulation function developed during the exercises. For the first iterations of the pipeline it gave us good results that were satisfactory. Nonetheless, the reprojection error of the newly triangulated keypoints was not taken into account when considering the keypoints.
Since the Matlab implementation of the triangulation function outputs at the same time the reprojection errors of each newly triangulated keypoint we decided to implemented and quantify the performance gain. Overall, linearTriangulation, the function done in the exercises, performs faster than the triangulation function of Matlab. This is comprehensible since the reprojection errors are also computed in the Matlab implementation. However, the few keypoints that had a big reprojection error were actually behind the camera, which they were easy to remove from the correct triangulations.
Finally, we decided to use the function done in the exercises since the increase of speed outperformed the slightly better reprojection errors of the Matlab implementation.

4.6.3 Analysis of performance of different ways to extract Harris features

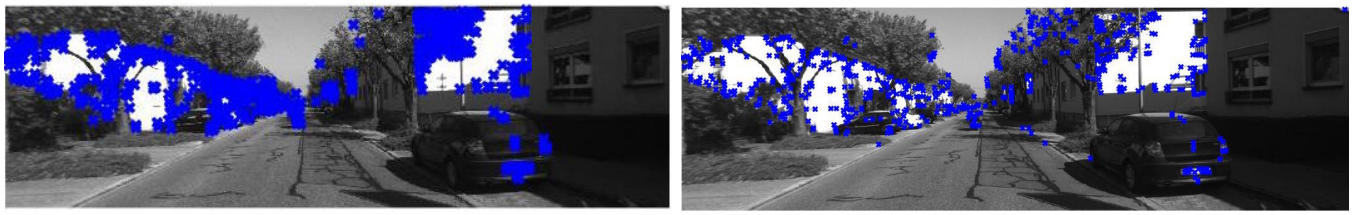
Extracting keypoints from the images is one of the most important parts of a visual odometry pipeline. For that reason, we decided to do a quantitative analysis of the performance of the way we used the Harris detector.

Figure 6: Uniform Harris Implementation.

In our implementation we added four different methods to apply Harris. The first consists in an implementation that we believed could improve the performance of the keypoints extraction. While running the pipeline we noticed that the keypoints were not uniformly detected over the image.
In order to get a more uniform set of keypoints over the image we decided to divide the frame into sub-images. We will refer to them as bins. In each of these bins we run the Matlab implementation of "detectHarrisFeatures".

This provides us with locally optimal corners in the bin. Applying this approach recursively over the bins gives us an homogeneous extraction of features of the image. Figure 7(b) presents the result of this implementation on the first frame of the Kitti dataset.

The other two algorithms that we may use consist in running the Harris corner detector over the whole image, as it is usually done. They differ between them in that one selects the strongest detected keypoints while the other selects them more uniformly. As it can be seen in Figure 7(a) and Figure 7(b), the uniform selection of keypoints performed by Matlab is still not homogeneous over the image. This should not be a problem if the quality of the corners is good.
Nonetheless, we experienced in practice that a mapping of keypoints on different patches of the image gives overall better results and is more robust to occlusions.



(a) Strongest Harris Features.

(b) Uniform Harris Features.

Figure 7: Harris Features.

5 Conclusion & Future Work

As can be seen in the videos provided and in the code, we have implemented a visual odometry pipeline with both stereo and monocular initialisation. Furthermore, we have added features such as automatic selection of initialisation frames, relocalisation and bundle adjustment.
Furthermore, we have tried to calibrate a smartphone camera and recorded our own dataset. This also showed us the difficulties in recording data and the limitations of our VO pipeline.
However, there is also room for improvement. Future work would include a further refinement of the full bundle adjustment online and loop closure.

References

[1] J.-L. Blanco, F.-A. Moreno, and J. González-Jiménez. The málaga urban dataset: High-rate stereo and lidars in a realistic urban scenario. *International Journal of Robotics Research*, 33(2):207–214, 2014.

[2] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[3] D. Scaramuzza and F. Fraundorfer. Visual odometry: Part i - the first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, 18(4), 2011.

[4] D. Scaramuzza and F. Fraundorfer. Visual odometry: Part ii - matching, robustness, optimization, and applications. *IEEE Robotics and Automation Magazine*, 19(2), 2012.