Mini-project: A Visual Odometry Pipeline

Vision Algorithms for Mobile Robotics

January 8, 2022

Authors:

Lei Li leilil@student.ethz.ch
Rong Zou ronzou@student.ethz.ch
Yang Miao yamiao@student.ethz.ch
Zhizheng Liu zhiliu@student.ethz.ch

1 Introduction

The implementation details, additional features and result analysis of the mini-project: a visual odometry pipeline of the course *Vision Algorithms for Mobile Robotics* is presented in this report. In particular, the deviations and extensions between our implementation and the recommended pipeline are briefly described in Section 1, after a succinct account of the milestones. In Section 2, details of how the visual odometry pipeline was implemented are depicted, including bootstrapping and continuous operation steps. As some interesting additional features, both local optimization (sliding-window bundle adjustment) and global optimization (loop detection and loop correction) were performed in our pipeline, which alleviate the scale drift and actually make the VO pipeline a rudimentary VSLAM pipeline. More information about the additional features is in Section 3. The feasibility of the proposed pipeline was verified with three datasets, namely, the parking dataset, the KITTI dataset and the Malaga dataset, and the results and analyses can be seen in Section 4. The videos of our pipeline running on these datasets can be seen on YouTube. The link is provided in Appendix.

1.1 Milestones

We divided the whole mini-project into seven phases and set corresponding milestones as follows.

(1). Task Comprehension
(2). VO Front End – Initialization
(3). VO Front End – Continuous Operation
(4). VO Back End – Local Optimization
(5). Toward VSLAM – Loop Detection
(6). Toward VSLAM – Loop Correction
(7). Result Analysis

In the task comprehension phase, we discussed how to co-develop the code, clarified misconceptions about the recommended pipeline, and decided what additional features we were interested in to add to our implementation. Milestones (2) and (3) are about achieving the minimum requirements of the task, which is implementing the front end of a visual odometry. In stage (4) – (6) we were committed to adding some interesting additional features to the pipeline using what we had learned in the course. Summary and analysis were carried out in the final stage. Besides, fine-tuning ran through the entire project.

1.2 Deviations and Extensions from the Recommendations

The pipeline we implemented is quite similar to the recommendations. In general, the

organizational structure and the function of each component of the proposed pipeline are consistent with the recommended one, but at the implementation level we made some modifications and extensions, as shown below.

(1). Selection of Bootstrapping Frames

In the recommended initialization step, two frames are manually selected for triangulating initial landmarks. In our implementation, instead of two views, a consecutive of m frames at the beginning of the dataset were used for bootstrapping, and only those 3D points whose projections appear on every image in the bootstrapping set can be selected as candidate landmarks. In practice we used $m = 5$ for all datasets.

(2). Generation of Initial Landmarks

While generating initial landmarks, we extended the recommendation which only performs pure triangulation. Specifically, we kept a triangulated 3D point as landmark only if it satisfies the following conditions:

(a). it is can be tracked for a consecutive of m frames, as mentioned above;

(b). it is located in front of the cameras;

(c). its average reprojection error is less than a threshold;

(d). the ratio of baseline to its depth, expressed as parallax, is larger than a threshold.

We found that this type of initialization made the pipeline more robust, compared to manually selecting two views and triangulating initial landmarks.

(3). Tracking of Keypoints

Instead of detecting Harris corners as keypoints and using KLT algorithm for keypoint tracking as recommended, we finally decided to use feature matching method by extracting SURF keypoints. Two pipelines, one with KLT and another using feature matching were implemented and finetuned, and it turned out that the one with feature matching performed better than its counterpart with KLT which sometimes lost tracking or cannot find sufficient keypoints, as shown in Fig. 1.2.1. Due to the limited time, the one with KLT was not further developed.

(4). Triangulation of New Landmarks

Another difference between our pipeline and the recommendation is how new landmarks are added. In the recommended version, keypoints are tracked across multiple consecutive frames for triangulating new landmarks. In fact, we first implemented this function, adding new landmarks under the same constraints mentioned in generating initial landmarks, but the results were not so satisfactory.

Therefore, we tried the keyframe-based method, in which features were detected, extracted and matched between keyframes for triangulation. Specifically, a keyframe selection mechanism is applied to each input frame, and when a frame is judged as a keyframe, new landmarks will be triangulated using keypoints in the current and previous keyframes. The keyframe-based method showed better performance than the recommendation in our experiments.

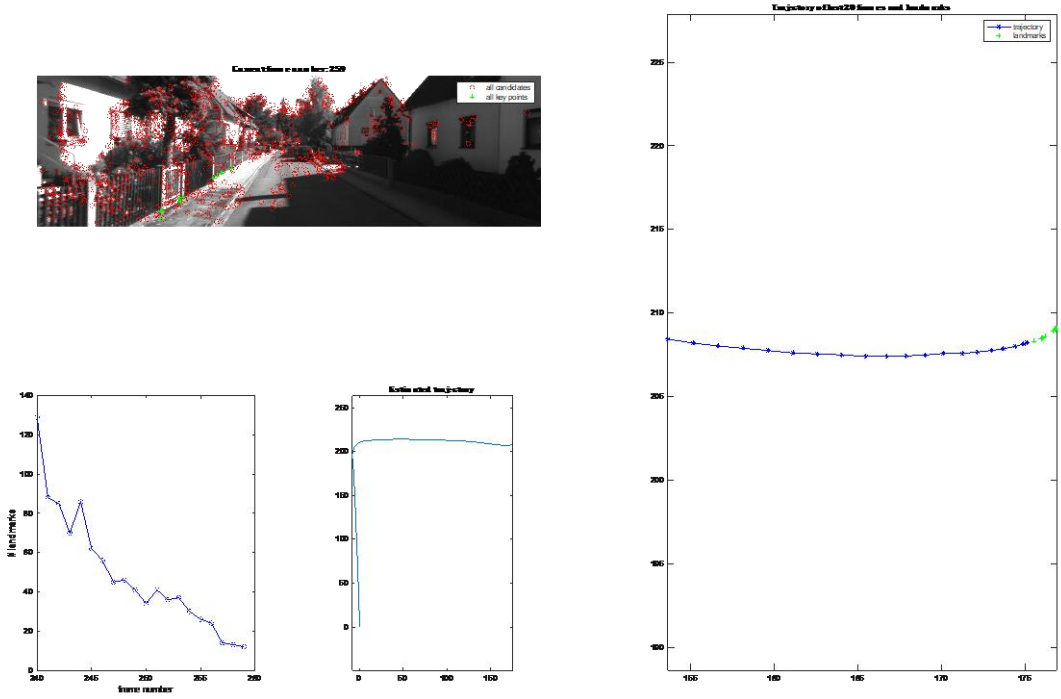

Fig. 1.2.1. Failure case of KLT. The subgraph in the lower-left corner shows that the number of tracked landmarks plunged to a level where the pipeline stopped working. The subgraph in the upper-left corner shows that hundreds of candidate features (red) were detected in the current frame whereas only a small fraction of them were successfully tracked (green).

(5). Maintaining A Global Pose Graph

Since we implemented local and global optimization in our pipeline, the Markovian style update which stores features and landmarks for computing the pose of the current frame as described in the recommendation was modified. Specifically, we maintained a global pose graph, where each vertex represents a keyframe with its pose and detected keypoints, and each edge represents the feature correspondences between two keyframes.

This means that when processing a new frame and performing local and global optimization, we re-select keypoint correspondences from the pose graph and triangulate world points. Compared to maintaining a set of landmarks and remove or append points, we found our method more flexible, because the locations of world points change frequently with optimization.

(6). Local and Global Optimization

Extended from the recommended pipeline, bundle adjustment for locally optimizing camera poses and landmark positions, as well as loop detection and correction for global trajectory refinement were added in our implementation. More details are in Section 3.

2 Visual Odometry Implementation

2.1 Initialization

In the bootstrapping step, an initial 3D point cloud was established using the first few frames. The initialization can be roughly divided into four sub-steps.

Firstly, a certain number of frames, denoted by $\{I_1, \dots, I_m\}$, at the head of the dataset were selected as the bootstrapping set. By our definition, all frames in the bootstrapping set are keyframes. For every frame in this set, the following tasks were performed:

(a). Keypoint detection: detecting SURF features and selecting a subset of keypoints that are uniformly distributed throughout the frame;

(b). Keypoint description: extracting descriptors of the detected keypoints;

(c). Keypoint matching: establishing 2D to 2D keypoint correspondences between every two adjacent frames, and hence a set of *n* candidate landmarks $\{\hat{L}_1, \dots, \hat{L}_n\}$ that appear in every frame across the entire bootstrapping set can be tracked. Each candidate landmark corresponds to *m* keypoints, where each of the *m* keypoints comes from one frame in $\{I_1, \dots, I_m\}$ respectively. For instance, candidate landmark $\hat{L}_i$ corresponds to keypoints $\{\hat{p}_1^i, \dots, \hat{p}_m^i\}$.

Then, the relative pose between every two consecutive frames was estimated by solving the essential matrix based on their keypoint correspondences, so as to obtain the absolute pose of the camera corresponding to each frame, denoted by $\{C_1, \dots, C_m\}$.

After that, positions of candidate landmarks $\{\hat{L}_i\}$ were triangulated from keypoint coordinates $\{\hat{p}_1^i, \dots, \hat{p}_m^i\}$ and absolute camera poses $\{C_i\}$, and only those points with small reprojection errors and large parallaxes between the first view I_1 and the last view I_m were retained as real landmarks (as described in Section 1.2 (2)), which formed the real landmark set $\{L_1, \dots, L_r\}$. A real landmark L_i corresponds to keypoints $\{p_1^i, \dots, p_m^i\}$.

Finally, bundle adjustment was performed in the bootstrapping set to further improve the estimation quality of the camara poses and landmark positions.

2.2 Continuous Operation

The initial keypoints and landmarks are used to bootstrap the whole pipeline. In continuous operation, the trajectory of the camera is incrementally estimated and new keyframes are selected which provide new landmarks. The process of continuous operation can be divided into four parts.

Firstly, for every input image, the keypoint detection and description procedures same

with those in the initialization stage were done, and a set of 2D-2D keypoint correspondences between the current frame I_{cur} and the previous keyframe I_k were established, denoted by $\{(p_{cur}^i, p_k^i)\}$. If there exists a 2D-3D correspondence between a matched keypoint of the previous keyframe and a landmark, a 2D-3D correspondence between the matched keypoint of the current frame and the same landmark can be established, i.e., from (p_k^i, L_i) to (p_{cur}^i, L_i). The landmarks were triangulated using two recent keyframes, except the initial landmarks which were provided by the bootstrapping step.

Based on the established image-to-world correspondences, the current absolute camera pose was estimated using the P3P algorithm.

Then, a keyframe selection mechanism was used to judge whether the current frame is a keyframe. Specifically, when the number of 2D-3D correspondences of the current frame is less than a threshold, we need to add a keyframe to guarantee that P3P can function normally. Besides, when the tracked keypoints of the current frame from the previous keyframe is not enough, and there are a sufficient number of non-keyframes between them, the current frame will also be judged as a keyframe.

Finally, sliding-window bundle adjustment will be applied to optimizing camera poses for the recent *s* keyframes, under the condition that the current frame is a keyframe.

3 Interesting Additional Features

3.1 Motivation

To achieve higher accuracy in camara pose estimation, and also locate landmarks with higher quality, we were well-motivated to perform local optimization in our pipeline using bundle adjustment.

In addition, observing that there exists loops in the ground truth path of the KITTI dataset and the quality of the global trajectory estimated by the basic VO was not ideal, we decided to implement additional functions of loop detection and loop correction to improve the global consistency of the estimated trajectory.

3.2 Local Optimization

Local optimization plays a significant role in combating scale drift. In the initialization stage, bundle adjustment was applied to the entire bootstrapping set. All initial landmarks and camera poses were optimized to minimize the reprojection error.

By contrast, in the continuous operation step, sliding-window bundle adjustment with

a window size *s* was performed every *k* keyframes, and only for keyframes, due to limited computational power. For every *k* keyframes, after the camera pose of the latest keyframe was calculated using world points triangulated by previous keyframes, camera poses of the recent *s* keyframes were adjusted to optimize the reprojection error. We first triangulated all possible 3D points among the *s* keyframes and retained those with a reprojection error less than a threshold, then the camera poses of the recent *s* keyframes were updated using bundle adjustment. The optimized camera poses would be used for triangulating new landmarks.

3.3 Loop Detection

The purpose of loop closure is to eliminate the accumulated error and correct the global motion trajectory. The main task of loop detection is to perform large-scale place recognition to determine if the current location has been visited before, and if it is the case, re-localization and global optimization will then be performed.

We implemented loop detection by training offline bag-of-words and then retrieving images using the bag-of-words model. In accordance with the feature descriptors used in the front-end, here SURF features were also used to implement bag-of-words training and retrieval. We used the *bagOfFeatures* provided in MATLAB2021 to extract visual words on the SURF features extracted from the KITTI dataset. Since KITTI is the only dataset that has loops in the three datasets of this project, our bag-of-words was trained only on the KITTI dataset. Features were extracted from each image and mapped to the descriptor space. Then, the descriptor space was hierarchically clustered into *K* clusters. The hierarchical clustering method clustered features into *b* branches and *l* levels to reduce the time of performing feature queries during image retrieval. The more branches and levels, the higher accuracy of the feature query, but in the meantime the more time required for querying. Therefore, it is necessary to keep a balance between precision and efficiency. We tried different settings of *[b, l]*, such as [4,12], [5,10] and [6,10]. In practice, we achieved good results using 5 branches and 10 levels.

The loop detection was performed only on keyframes in order to reduce time cost and false matches. However, if we only use the bag-of-words model for loop detection, a large number of false matches will occur because there are too many features that are similar across scenes. Therefore, we drew lessons from loop detection in ORB-SLAM to perform sequence matching, which contains the following steps:

(1). First, the similarity scores between the current keyframe and its strongly connected keyframes were calculated. The minimum similarity score was used as a baseline to find candidate keyframes that are visually similar to, but not connected to the current keyframe.

(2). Then, for the current keyframe, we eliminated the local keyframes connected to it and then iterated through all other keyframes to find the ones that have the same words with it. At the end of this step we will obtain a set of candidate keyframes.

(3). After that, for each candidate keyframe, a group was formed, which contains

the top 10 keyframes that are closest to it. Then for each group, a similarity score was evaluated. A group can only be retained when its score is higher than a threshold, and the candidate keyframes corresponding to the retained groups were selected as 'loop keyframes'. In practice we used 0.75 times the highest similarity score among all groups as the threshold.

(4). Finally, we performed the consistency check. If more than 3 loop keyframes are consecutive, a loop is considered to be detected.

After these steps, we will obtain a set of 'loop keyframes' with good accuracy, but there may still be some false detections, so we need to perform further verification to double check whether there are enough point matches between the obtained loop keyframes and the current keyframe.

3.4 Global Optimization

The goal of global optimization is to enable the pipeline to correct the trajectory when loops are detected and improve global consistency of the estimated trajectory. The key ideas are as follows.

Once the loop detector detects a loop and returns multiple potential 'loop keyframes' retrieved from previous keyframes, the middle frame of the retrieved keyframes, denoted by I_l, is used to do loop correction.

Firstly, to reject false detections, we double checked that the current query frame I_c and the retrieved frame I_l indeed contain the same scene. Instead of pure feature matching between I_c and I_l, we checked whether re-localization of I_c is possible. Concretely, triangulation between I_l and its previous keyframe I_{l-1} was performed to get world points, and then world-to-image correspondences were obtained for P3P to get the relative pose of I_c. If the number of inliers is below a threshold, the detected loop will be considered a false detection, otherwise an edge between I_c and I_l will be added to the global pose graph and global pose optimization will then be performed.

For the global optimization task, we used the existing function in MATLAB 2021. In order to alleviate the effect of scale drift on pose estimation, we came up with several ideas which indeed improved the performance of our system:

(1). Firstly, for the edges of pose graph, we used not only rigid-body transformation constraint but also 3d similarity transformation (SIM(3)). For every 5 keyframes, one edge of SIM(3) constraint was added to connect two keyframes. It was also used in connecting detected loops. By using SIM(3), the scale changes can be handled in global optimization and the adverse influence of scale drift can then be reduced.

(2). Another adjustment we made was based on the observation that sometimes the scales in the bootstrap frames were largely different from that of subsequent frames due to scale ambiguity in initialization. So we also applied SIM(3) constraint instead of

rigid-body constraint for the bootstrapping frames when doing optimization.

(3). We found that once a loop is detected, many loop will be detected along the way. As global optimization is computationally very expensive, and experiments show that it is neither necessary nor beneficial to continuously perform global optimization, we only do it after a certain number of frames has passed since the last loop-triggered optimization, in practice we set the number as 20. Experiment showed that doing so made the system more stable.

4 Results and Analyses

4.1 Basic VO

The performance of the basic VO without local and global optimization was very unsatisfactory. The following figures shows two trajectories estimated by the basic VO for the parking and the KITTI datasets, respectively. Apparently, there are huge deviations between the estimated ones and the ground truths. In sharp contrast, we will see that the scale drift would be much smaller and the trajectory would be much closer to the ground truths after adding local optimization.

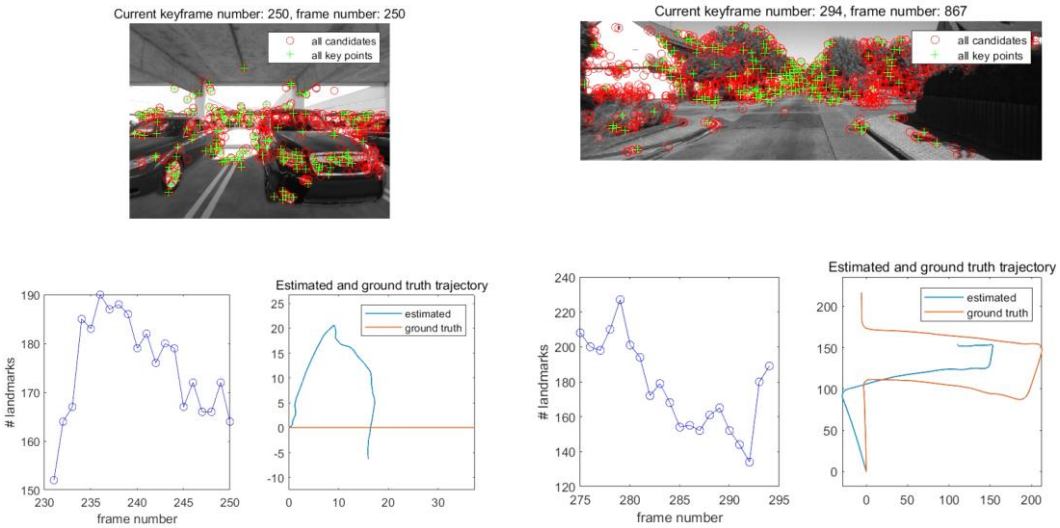

Fig. 4.1.1. The basic VO applied to the parking (left) and the KITTI (right) datasets.

4.2 Standard VO: Basic VO with Local Optimization

The quality of the standard VO pipeline, namely, basic VO with local optimization, was proved to be much better than the basic VO. As can be seen in Fig. 4.2.1, the results of the standard VO pipeline are now comparable to the ground truths. Also, compared with the results given by the basic VO in Fig. 4.1.1, it is obvious that the quality of the VO has been greatly improved after local optimization.

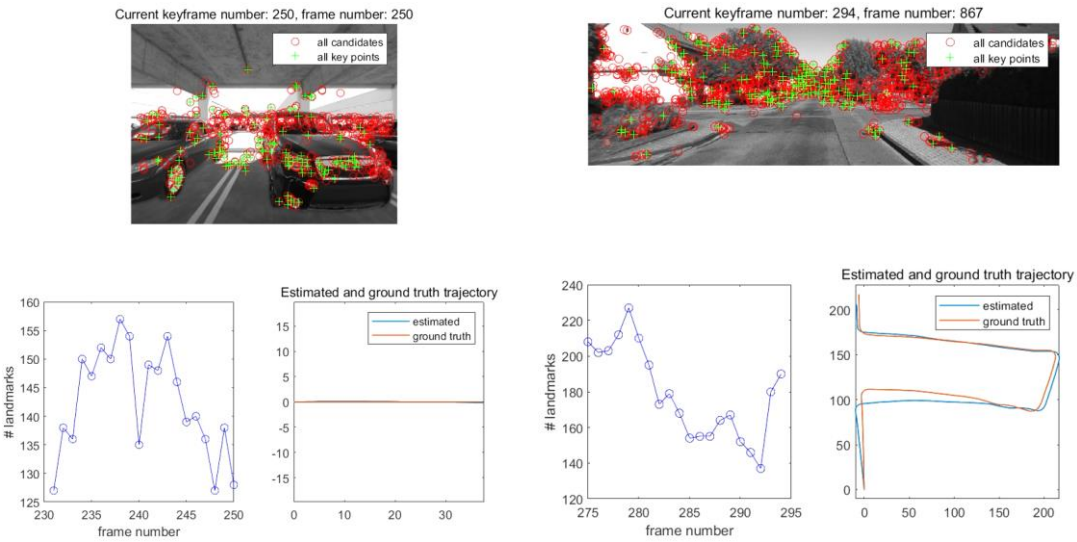

Fig. 4.2.1. The standard VO (basic VO with local optimization) applied to the parking (left) and the KITTI (right) datasets. The frame numbers are the same with those in Fig. 4.1.1 for the convenience of comparison.

For the Malaga dataset, since there was no ground truth provided, we compared our results with the results of the recommended pipeline found in the YouTube playlist. The comparison is given in Fig. 4.2.2., which shows that our pipeline went even further.

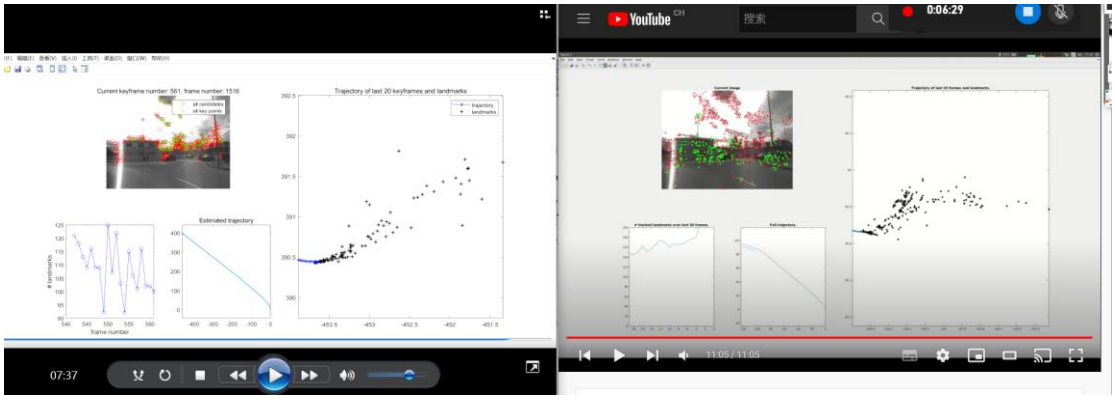

Fig. 4.2.2. Our standard VO (left) and the recommended pipeline (right) applied to the Malaga dataset. The recommended pipeline crashed at this frame, whereas our pipeline can still run and go further.

4.3 Toward VSLAM: Standard VO with Global Optimization

On the basis of local optimization, we upgraded our VO pipeline again by adding the functions of loop detection and global optimization, which actually made our pipeline a rudimentary VSLAM pipeline.

Experiments on the KITTI dataset showed that, the loop detection module worked very well. In other words, loops can always be detected successfully. An example of the work of the loop detector is shown in Fig. 4.3.1.

The loop correction also functioned normally, but the overall effect depends on the accumulated scale drift. If the scale drift of the standard VO is small, the effect of the global optimization will be quite impressive. Figure 4.3.1 and 4.3.2 show the trajectory before and after loop correction. Comparing the two figures we can see that the global optimization works reasonably well. However, it should be noted that when the scale drift is large, loop correction may degrade the performance of the system and sometimes may even crash the pipeline.

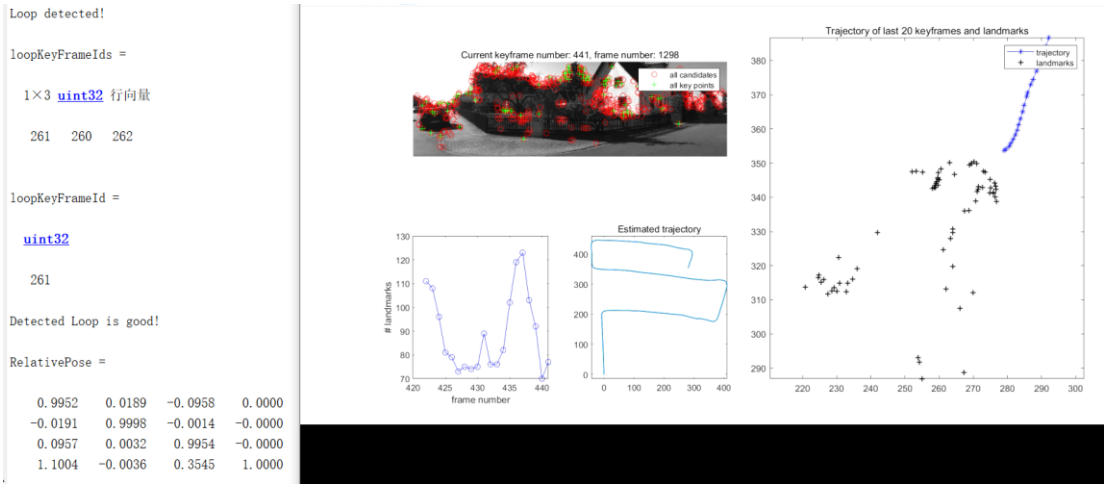

Fig. 4.3.1. The work of the loop detection module of our pipeline. At frame 1298, a loop was detected and the loop detector returned 3 retrieved keyframes, indexed 261, 260 and 262. The middle one, keyframe 261, was selected for double checking the loop detection quality. Here the quality was judged as good, so the relative pose between the current frame and keyframe 261 was calculated and global optimization was then performed. See Fig. 4.3.2 for the result.

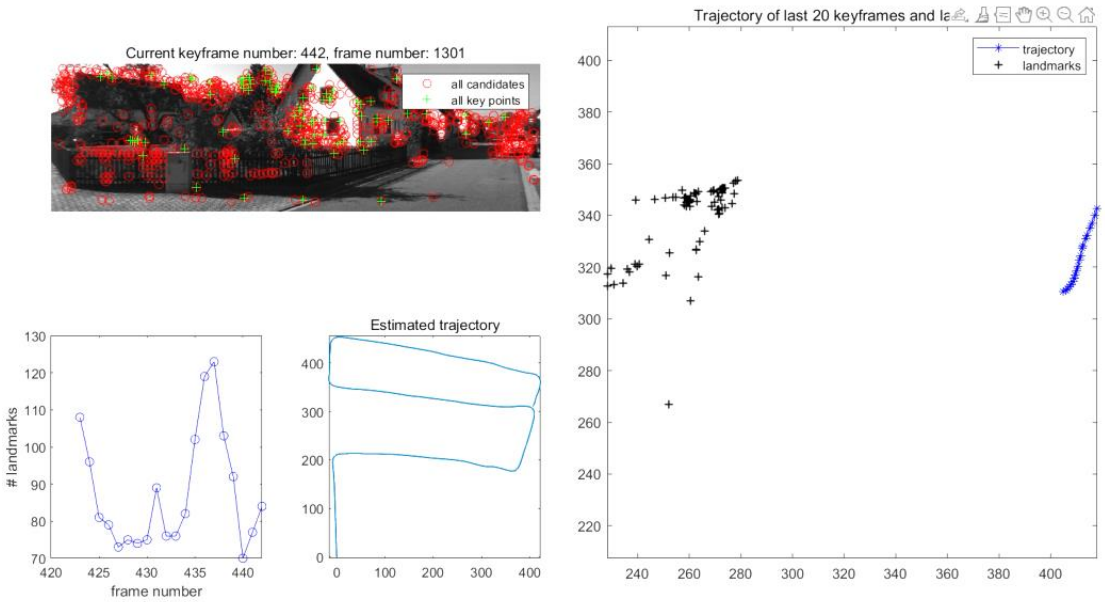

Fig. 4.3.2. The work of the loop correction module of our pipeline. Compared with Fig. 4.3.1, it can be seen that the loop is successfully closed and the quality is reasonable.

The figures below present the results of our standard VO pipeline applied to the KITTI dataset both without and with global optimization. Obviously, the global consistency of the trajectory is well guaranteed in the global optimization case. Therefore, we can conclude that global optimization significantly improves the quality of our pipeline in datasets with loops.

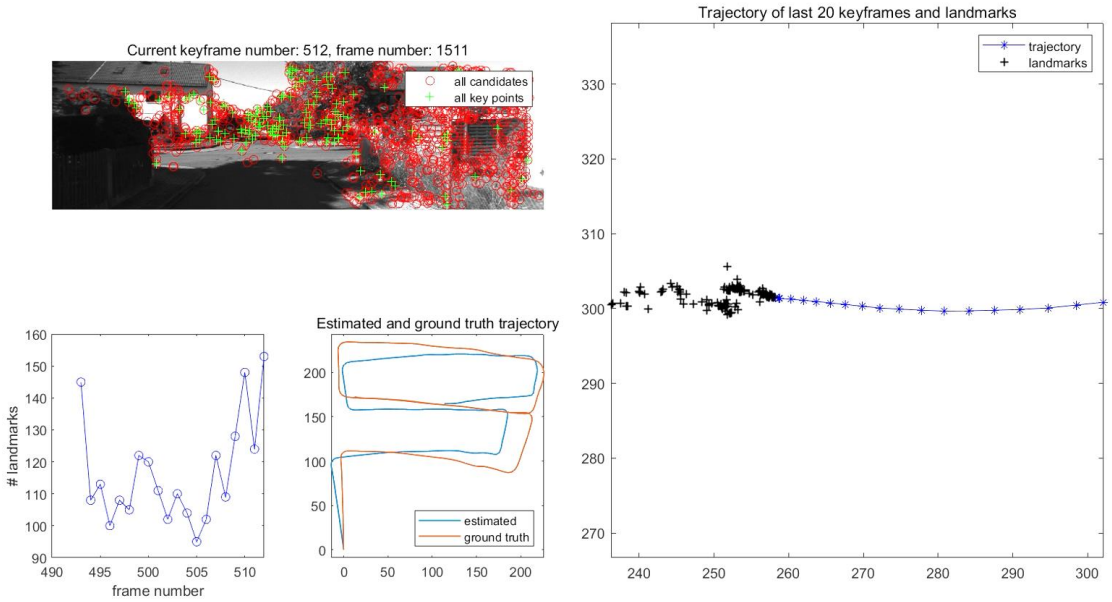

Fig. 4.3.3. Results of our standard VO pipeline applied to the KITTI dataset. Loop is not corrected.

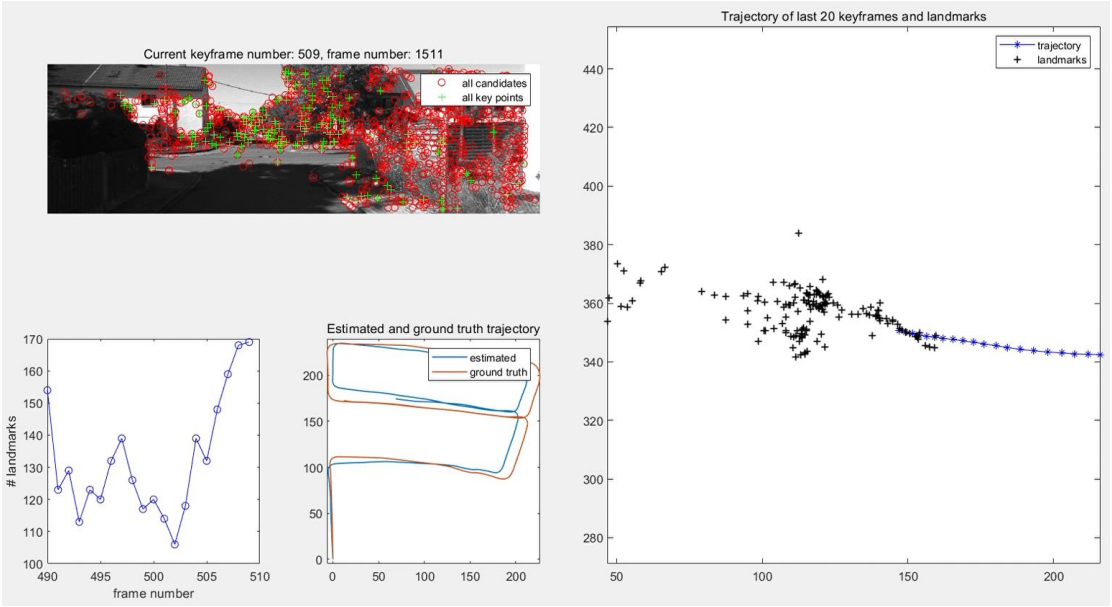

Fig. 4.3.4. Results of our standard VO pipeline with global optimization applied to the KITTI dataset. Loop is detected and corrected with good quality.

Appendix

1. YouTube link for videos
https://youtube.com/playlist?list=PL0_u6UI2Giaq8gtEdg6E5ObpMR-UAU-H5
There should be 4 videos:
(1). Standard VO applied on Parking.
(2). Standard VO without global optimization applied on KITTI.
(3). Standard VO with global optimization applied on KITTI.
(4). Standard VO applied on Malaga.

2. Used pre-written functions:
bundleAdjustment
detectSURFFeatures
extractFeatures
estimateEssentialMatrix
estimateWorldCameraPose
evaluateImageRetrieval
findTracks
matchFeatures
optimizePoses
relativeCameraPose
retrieveImages
selectUniform
triangulateMultiview