A visual odometry pipeline

Markos Gkozntaris, Gerasimos Maltezos, Nishendra Singh
mgkozntaris@student.ethz.ch, gmaltezos@student.ethz.ch, snishendra@student.ethz.ch

***Abstract*—This report provides an overview of the visual odometry pipeline implemented using the most essential features: initialization of the process with two frames using the Harris/ORB detectors and first triangulation of 3D landmarks, keypoint tracking between consecutive frames using KLT, pose estimation using established 2D - 3D correspondences and triangulation of new landmarks.**

I. Introduction

In robotics and computer vision, visual odometry is the process of determining the position and orientation of a robot by analyzing the associated camera images. It has been used in a wide variety of robotic applications, such as on the Mars Exploration Rovers. It is crucial for flying, walking and underwater robots and it has many advantages compared to other localization methods (not affected by wheel slippage, small relative position error e.t.c.)

This report is presenting a visual odometry pipeline for three different dataset: KITTI [1], Malaga and parking, achieving mostly a consistent local trajectory tracking. In more detail, the triangulation of new points was based on two main criteria: whether the baseline was appropriate, which was checked using an alpha angle and whether there were enough features between the two consecutive frames. Features were extracted using Harris detector or ORB, while the tracking between two different frames is done using KLT. As discussed in the course, these are some of the most well known and efficient detectors, which work exceptionally well in VO, whereas the KLT algorithm is much faster than traditional tracking algorithms, because it examines far fewer potential matches between the images. Furthermore the pose is estimated using P3P and outliers are being found using a variation of RANSAC used by Matlab, the MSAC algorithm.

The code is implemented using Matlab, where the Computer Vision Toolbox was heavily used as well as implemented function from the exercise session of course Vision Algorithms for mobile robots of UZH. [2]

II. Algorithms review

A. Feature detection

In the presented code, there were two main feature detectors used, the Harris feature detector and the ORB feature detector. Even though the second managed to calculate more features, the trajectory produced proved to be less optimal compared to when using Harris. In our implementation different thresholds are used. It seems that the different rules of thumb tried could not make it as good and consistent, compared to using Harris detector. ORB was tried out as a computationally-efficient replacement of SIFT that has similar matching performance, is less affected by image noise, rotation invariant and is capable of being used for real-time performance. [3] However it unfortunately proved that more tuning was required. This malfunction of the ORB detector was especially apparent on the parking dataset. When using the Harris detector the global path seemed to be tracked more consistently.

B. Feature tracking

In order to track features KLT was used. Very briefly, the KLT algorithm estimates the parameters of the transformation of a patch of the feature, which minimize the Sum of Squared Differences (SSD) between the reference patch and warped one. The KLT follows the Gauss-Newton method for minimization, so it applies a first order approximation of the warp and attempts to minimize the SSD iteratively. It is categorized as a direct method, as it used intensity values to extract results, thus making it more accurate and more robust to motion blur and weak texture.

C. Outlier removal

Considering that our camera is calibrated we can use P3P, while removing outliers using MSAC. P3P was used over a method such as DLT because it is more robust to noise, it is more accurate and more efficient. The output of this algorithm is than refined using non-linear optimization by minimizing the sum of squared reprojection errors. The reprojection error was used over other error metrics as it shows the highest accuracy. On the other hand, MSAC was used as it proved to be more robust and not sensitive to initial conditions, nor local maxima, compared to other methods such as EM. Also, it can cope with a higher percentage of outliers compared to GNC, even if a bit slower. Considering that the point cloud can be used to match geometrically 3D features, where only distances and no intensities are available the percentage of outliers will increase significantly. However, we must state that MSAC is non-deterministic so it is not possible to re-implement exactly the results presented on Sec. IV (should be close enough though).

III. Visual Odometry Pipeline

The pipeline is separated into two main parts, Bootstraping and Continuous Operation, which will be analyzed in the subsections below.

A. Bootstraping

In the bootstraping (initialization) phase, two frames are being used in order to initialize our pipeline. At first, features are being detected, then these features are matched to the

second frame, Fig. 1, then from the feature correspondences the 3D landmarks are being calculated and lastly, the initial pose estimate (R, T) is being produced. To be exact, as proposed in the statement, we did not use the first two frames, but rather the first and third frame, so there is a sufficient baseline between them to be able to track points close to the camera and to avoid large depth errors, but in the same time not very far apart, so we are sure that many point correspondences actually exist.

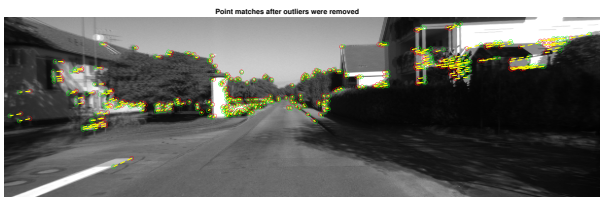

Fig. 1: Point matches after outlier removal

B. Continuous Operation

After bootstraping, the continuous operation takes place. In this stage, we take consecutive frames, we track the features that exist between the previous and the present frame and we again basically estimate the new poses of the camera. Although this is very intuitive and easy to understand, just using the preexisting from the initial bootstraping features is not enough. That is because, every time the camera moves on, some of the features get lost, so the keypoints are being reduced with every step. In order to combat that reduction, continuously detecting new features would be really computationally consuming. So, two main methods are being used. First of all, during the tracking phase, not all features are being used to triangulate new landmarks (because some are considered invalid by the KLT algorithm). These, features are being saved in a DataStruct along with the actual keypoints, and in every step, they are being tested in order to find out if they could be used to triangulate new landmarks. The condition for adding new keypoints is determied by calculation the alpha angle between the bearing vectors between the candidate keypoints that are being tracked and the frame where they were first seen. The threshold value for alpha was tuned manually.

When this is not enough, and the candidate, as well as the actual keypoints are being reduced under a threshold, then the only solution is to add more keypoints. This contains redetecting and rematching features between the current and the previous frame.

C. Bundle Adjust

For optimization purposes bundle adjustment was used over another optimizer such as pose-graph optimization. Bundle adjustment takes into account the pointcloud in addition to the camera poses, which makes it more accurate, although more computationally expensive. Moreover, in order to reduce the computational cost we are using a windowSize = 10. The sliding window takes into account a limited number of previous frames while optimization, and is thus faster to run. Even BA slows down the pipeline, it was our choice was to go with it, as are running the pipeline on a PC were the computational capacity allows it compared to more limited computational resources, like a mobile phone. Furthermore, when using Harris features the number of triangulated points is not very high (usually around 200), so this works in our favours in this case, considering that the complexity of bundle adjustment is cubicly depended on it.

IV. Results

The created trajectory of the 3 datasets are presented on Fig. 2, 3 and 5. A consistent local trajectory tracking is observed. More specifically for the KITTI dataset the local trajectory was observed for around 260 frames, for the Malaga dataset it was observed for around 700 frames, while for the parking dataset it was observed for the complete set of images, which is 598 frames. Note that for the KITTI and Malaga datasets a global trajectory can not be achieved consistently no matter the combination of parameters tried. These datasets proved to be the most challenging between the three. However it must be stated that they follow the ground truth for the first frames, which include a turn to the right and navigation through the first alley, Fig. 2, or a turn to the left respectively, Fig. 4, as shown in the figures. More specifically, for the Malaga set, as seen at the video [4], the pipeline fails when some keypoints are still tracked, even when the vehicle has actually surpassed them. Maybe, this is a glitch of the algorithm, because we already do a check to keep only the points which are still in the front side of the camera. On the other hand the parking dataset seems to be consistent during the whole trajectory, using both Harris, Fig. 5, and ORB algorithm, Fig. 6. Especially using ORB, the final succeeded drift, seems to be reduced drastically. It should be however noted that while using ORB a larger local drift was apparent when the landmarks were mostly in the background (case when no cars are very close). However a loop closure is not implemented yet.

It is also worth mentioning that it was observed that when using ORB detector in the Malaga dataset the turn was not correctly calculated. That might be because the number of landmarks in that case is more than 1000, so usually the candidate keypoints are in the background (further away in real space) and they do not change drastically during the turn. This makes the camera pose calculations be less accurate, as the algorithm is consistently getting a large number of landmarks assuming it turns very slightly.

In this case we applied P3P RANSAC for localization and updated the keyframe when 20% of the initial number of points were observed. This is a typical rule of thumb used which seems to produce good results.

Notice that without a more sophisticated technique a closed loop trajectory (for example for the Malaga dataset) is very hard to be achieved, as accumulation of error over time is a characteristic of a visual odometry system. Usually in such cases we need to have a visual odometry algorithm to recognize a previosuly visited place (a frame that is quite close

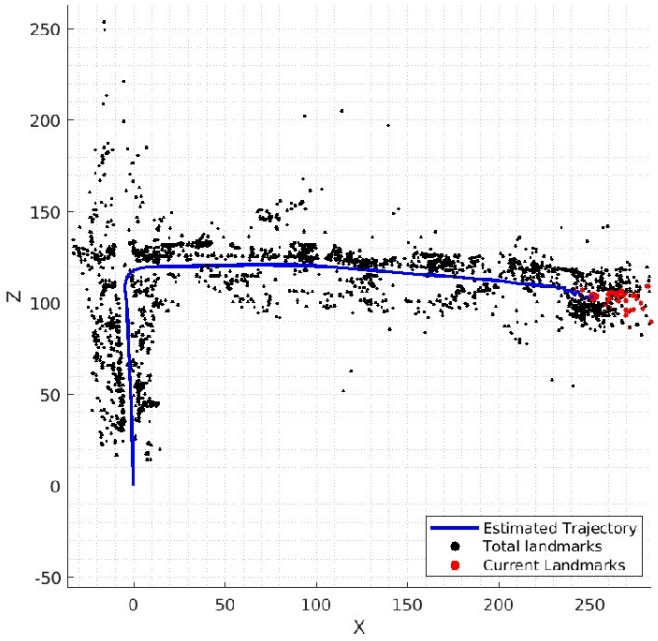

Fig. 2: Trajectory and 3D point cloud calculated for the KITTI dataset [5].

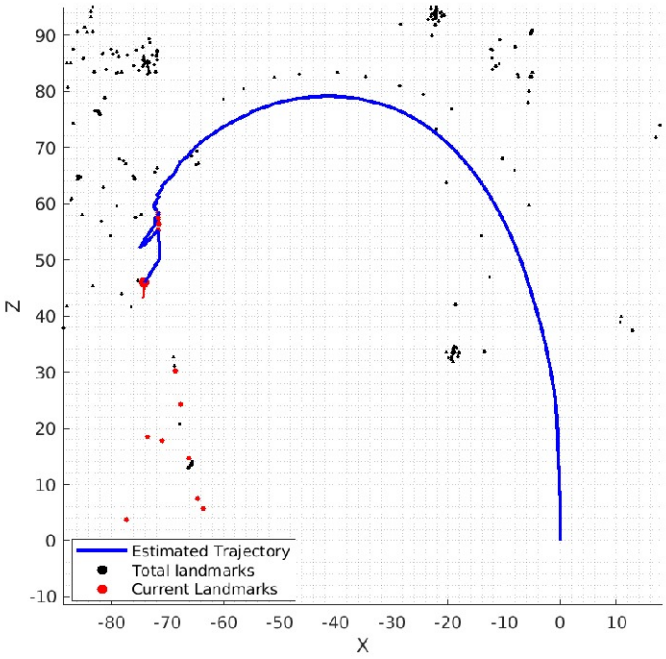

Fig. 4: Trajectory and 3D point cloud calculated for the Malaga dataset (left turn) using Harris [4].

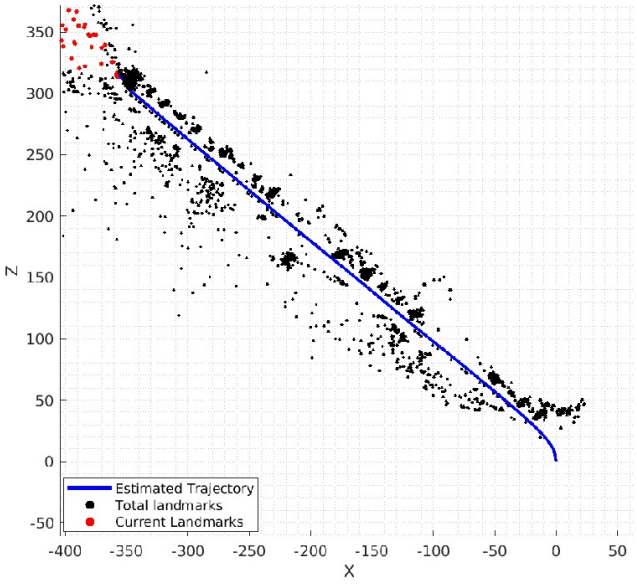

Fig. 3: Trajectory and 3D point cloud calculated for the Malaga dataset (beginning and straight line) using Harris [4].

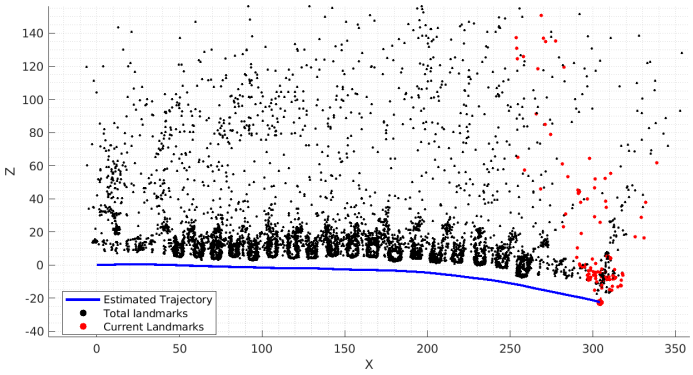

Fig. 5: Trajectory and 3D point cloud calculated for the parking dataset usign Harris[6].

to it as exactly the same frame will not be possible), which can be added as a constraint in the Bundle Adjustment and avoid therefore map duplication. A place recognition algorithm could be Bag of Words. Sometimes, this is even not possible considering that because of the accumulated error an open loop will be constructed.

V. Conclusions

The parking dataset seems to be the most consistent considering that a good local trajectory is generated arguably well while using either Harris or ORB. Furthermore the global trajectory seems to be acceptable. Of course it should be noted that drift was evident, no matter the combinations tried, but that is generally to be expected from a visual odometry pipeline, as the error accumulates over time making the trajectory globally incorrect.

Possible reasons why the VO pipeline failed at the first two datasets could be: the loss of numerous keypoints during the turns, the huge illumination changes and very dark shadows during the turns and the existence of keypoints only on the one side of the image. In order to improve the trajectory of the two first datasets a possible solution would be to reinitialize the pipeline every x (150) number of frames. This would create new 3D point cloud. However, in this case the orientation of the last frame should be kept in order to orient the new calculation accordingly towards the world frame.

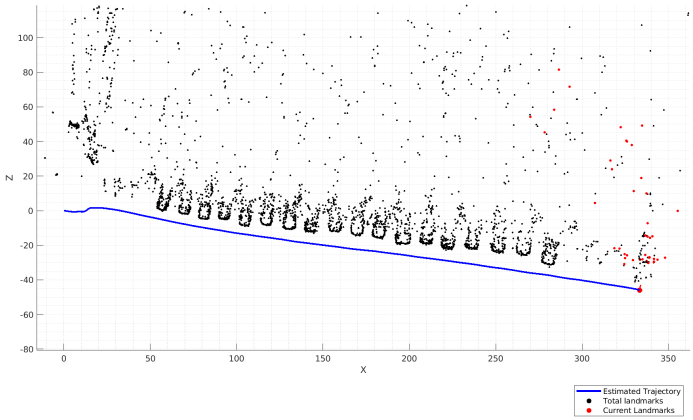

Fig. 6: Trajectory and 3D point cloud calculated for the parking dataset using ORB descriptors.

In the future, when we finally succeed to have local consistency for all three datasets for the whole set of images, the next steps would be, initially to achieve global consistency using an optimization method, like Bundle adjustment, as well as an algorithm for place recognition, like the Bag of Words algorithm. More improvements would consist of using methodologies taken from Machine Learning, which could drastically change the efficiency of our pipeline, trying to test our pipeline in more datasets, inserting and using measurements from IMU for error minimization and generally improving the efficiency of our algorithm.

REFERENCES

[1] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[2] Robotics and P. group. Vision algorithms for mobile robotics. University of Zurich. [Online]. Available: http://rpg.ifi.uzh.ch/teaching.html

[3] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[4] M. Gkozntaris, G. Maltezos, and N. Singh. Vo malaga dataset. ETH Zurich. [Online]. Available: https://youtu.be/VefXI7ZN9pQ

[5] ——. Vo kitti dataset. ETH Zurich. [Online]. Available: https://youtu.be/ZDaJRc1xPr4

[6] ——. Vo parking dataset. ETH Zurich. [Online]. Available: https://youtu.be/WLacxrNTOxU