Vision Algorithms for Mobile Robotics
(Mini-) Project

Yvain de Viragh
Marc Ochsner
Nico van Duijn

06.01.2016

1 Introduction

In the process of this course, we implemented several building blocks for a visual odometry pipeline during the exercises. As a mini-project, our task was to put the pieces together and implement a full pipeline. In doing so, we were using some of the work from the exercises as well as some MATLAB functions from the Computer Vision Toolbox. Our final pipeline was tested on three different datasets and showed good performance up to a scale factor.

2 Pipeline Structure

2.1 Initialization

To initialize the pipeline, we use the normalized 8-point algorithm with RANSAC. We use only the monocular parts of the datasets in order to keep the pipeline as general as possible. Initially, a set of harris corners is detected, described and matched on the first two images of the dataset. Then the 8-point algorithm in conjunction with RANSAC returns an estimation of the fundamental matrix of the camera. In order to quantify the accuracy of the current guess of F during the RANSAC algorithm, we use the point-to-epipolar-line-distance.

After RANSAC has determined the largest set of inliers, the 8-point algorithm is run again on the entire set of inliers. We then use the obtained fundamental matrix to triangulate a set of landmarks using all inlier matches from the harris corner detector. The set of landmarks and key points is now used to initialize the state.

2.2 Lukas-Kanade Tracker

Matching harris corners is computationally very costly, as the matching procedure increases quadratically with the number of features. Therefore we decided to use the Lucas-Kanade Tracker on the key points from the previous frame and determine their location in the current frame.

2.3 Feature discard

Features that could not be tracked with the Lucas+Kanade Tracker or were determined to be outliers by the P3P algorithm are removed by incrementing their corresponding value in a voting array. Once a feature receives more than a certain threshold of votes, it is discarded. This ensures that temporary occlusions caused by moving objects in a scene do not cause complete loss tracking and therefore improves robustness.

2.4 P3P + RANSAC

After the location of the features in the new frame has been found, we use the P3P algorithm in a RANSAC loop in order to calculate the current pose of the camera. Again, we increment the "discard" voting array for all key points that have been determined to be outliers by RANSAC.

2.5 Feature Extraction

In order to have a good number of landmarks throughout the pipeline, we continuously extract new features. To do so, we extract the harris corners in the current frame, assert that they are a certain minimum distance away from the current key points and include them in our state as candidate key points. Along with the key points, we store the pose during their first respective observation.

2.6 Linear Triangulation

In every iteration, we examine our candidate key points in order to determine whether they can be found in the current frame. If the angle between the bearing vectors of the current observation of the landmark and the first observation is large enough, we triangulate its position and add the feature to the key points in our state.

2.7 Bonus Features

2.7.1 Structure-Only Bundle Adjustment

In order to increase the accuracy of the pose estimation and to combat scale drift, we run structure-only bundle adjustment on all landmarks which have already been observed a certain number of times. While this should not be as effective as full bundle adjustment, it is computationally significantly cheaper.
Our approach is based on linear triangulation, but instead of only considering two point correspondences takes in account multiple ones. I.e., given n point correspondences we have the following system of equations:

$$\underbrace{\begin{bmatrix} \vec{p}_1^{\times} \boldsymbol{M}_1 \\ \vdots \\ \vec{p}_n^{\times} \boldsymbol{M}_n \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} {}^W\vec{r}_{WP} \\ 1 \end{bmatrix}}_{\vec{x}} = 0_{6\times 1} \quad \Rightarrow \quad \boldsymbol{A}\vec{x} = 0$$

where $\boldsymbol{M}_i$ denotes the projection matrix of frame/image i. This linear-least squares problem is solved same as in the case for 2 point correspondences. Crucial for this approach to work correctly is to not include any outliers. We therefore only take into account those observations where the reprojection error given the unadjusted landmark is sufficiently small.

2.7.2 Re-initialization

We found that for large camera motion, the Lucas-Kanade Tracker would often fail to find the old key points in the new frame. This would mean that we have to either use brute-force matching on all key points, and remove many outliers, which would be very costly. Therefore we decided to re-initialize the entire pipeline with the 8-point algorithm if the number of successfully tracked key points drops below a certain threshold. This is slow and costly, but yields very good results, along with very accurate landmarks which can now be used for a number of frames. Furthermore we found that this procedure alleviates scale drift to a large extent. We achieved good results with a minimum keypoint number of 30.
During our evaluation, we also tested a procedure where the pipeline regularly re-initializes after a set number of frames. This yields very good results, but slows down the pipeline significantly.

2.7.3 Custom Dataset Testing

As an additional bonus feature we made a video with a mobile phone camera. We used the MATLAB toolbox for camera calibration and image undistortion.

3 Parameters and Tuning

3.1 RANSAC Iterations

In the initialization procedure, RANSAC is run in conjunction with the 8-point algorithm. This means we need RANSAC to run long enough such that we have a reasonable chance of selecting 8 inliers in our dataset. By plotting the number of inliers over lots of iterations, we determined that 1000 iterations is a good number to use.

We still found that there are occasionally outlier-matches in this procedure. They are very rare and can only be explained by the case that two falsely matching key points happen to lie on the same epipolar line or very close to it. But with a large number of inliers their effect has shown to be insignificant.
During the P3P pose estimation, we require only 3 points to obtain a pose estimate. Therefore the chances of selecting a sample entirely consisting of inliers is much larger and we found that 80 RANSAC iterations are usually sufficient. However, we received bad results for the parking dataset and chose to increase the number of RANSAC iterations to 500 and received much better results.

3.2 Harris Patch Size

During the exercises, we use a harris patch radius of 9 pixels. This showed to be a good compromise for our implementation, as larger patches significantly slowed down the pipeline without adding significant improvements in accuracy.

3.3 Minimum Angle in Linear Triangulation

When triangulating new landmarks, we ensure that the landmark is accurately calculated by checking for the angle between the initial and the current observation. The angle depends on the relative motion of the camera as well as the distance to the landmark. Very large angles proved to be strongly correlated with bad matches as well. We found that by setting a maximum angle, we were able to remove many outliers and achieve much better landmark triangulation. For the KITTI dataset we found that a minimum angle of 1 degree and a maximum angle of 1.8 degrees yield good results. For the "parking" dataset we found that many landmarks are much closer to the camera, so we increased the maximum angle to 6 degrees.

4 Results and Discussion

In our pipeline evaluation, we tested the three datasets that were provided with the assignment, as well as our own custom dataset. We also tested with and without bundle adjustment in order to show its effectiveness. Furthermore, we enabled and disabled the continuous re-initialization to illustrate how it alleviates scale drift problems
Our pipeline was primarily tested and tuned for the KITTI dataset. Figure X shows our custom Z/X plot of the path estimated by our VO pipeline in red, as well as the ground truth provided as part of the dataset in blue. The top image shows the current frame with the successfully tracked keypoints highlighted with green lines, as well as the candidate keypoints in blue. The red marks are newly added landmarks. The bottom figure shows the current position with a red cross and the triangulated landmarks with green crosses.

4.1 KITTI Dataset Comparison

To compare the performance of our various implementations, we decided to use the KITTI dataset as a benchmark. In this section we will compare four different runs that showcase various trade-offs. First, we discuss our full integration including bundle adjustment and periodic re-initialization. Then we compare it to previous versions of our pipeline, namely with deactivated bundle adjustment, deactivated re-initialization and a combination of the two.

4.1.1 Bundle Adjustment, Periodic Re-initialization

4.1.2 No Bundle Adjustment, No Periodic Re-initialization

4.1.3 Bundle Adjustment, No Periodic Re-initialization

4.1.4 No Bundle Adjustment, Periodic Re-initialization