ETH Zürich

# Visual Odometry Pipeline

---


| *Authors:* | *Student Number:* |
|---|---|
| David Renggli | 13-928-569 |
| Johann Diep | 12-927-125 |
| Timo Strässle | 10-933-042 |


January 7, 2018

# Contents

# 1 Introduction

The proposed monocular visual odometry pipeline takes a set of consecutive gray-scale images as input and estimates the absolute camera position with respect to the world frame (origin defined at the cameras initial position). To perform this, keypoints are continuously tracked and 3D landmarks triangulated. The pipeline is tested with four different datasets: Parking, KITTI, Malaga and a custom dataset recorded with a mobile phone.

# 2 Visual Odometry Pipeline

Our visual odometry pipeline consists of two main elements: the initialization and the continuous part. The work flow is briefly presented here. Details and used functions are described in section 3.

## 2.1 Initialization

First, we have to initialize our pipeline. We take two non adjacent frames (frame 1 and 3) to ensure a large enough baseline between the camera frames. For frame 1 we calculate the harris scores and based on them, we select a number of keypoints. These features are then tracked to the 3rd frame using the Lucas-Kanade-Tracker (KLT), resulting in keypoint correspondences between the two initialization frames. As a next step, RANSAC is applied to remove outliers in the correspondences. Based on the inlier correspondences, the relative camera pose between the two frames is estimated. Lastly, a point cloud of 3D landmarks is triangulated using the inlier correspondences and the camera pose. With this data, the state S (see section 3.1) for the continuous part is initialized.

## 2.2 Continuous Operation

After the first initialization frames, we want to estimate the pose of the camera continuously. Therefore we continuously track the existing keypoints between adjacent frames using KLT and remove outliers with RANSAC. Since we are loosing some keypoints over time (failed to track, defined as outlier or out of image range), we also have to continuously add new keypoints. For every new frame, we get new candidate keypoints from harris scores. They are then tracked between adjacent frames the

same way. Once they reach a large enough baseline relative to their first observation, we use them to triangulate new 3D landmarks. This threshold is defined by the bearing angle. After every iteration, the state S is updated. This way, we add new keypoints (and 3D landmarks) in every cycle to ensure a continuously working pipeline.

# 3 Pipeline Modules

The visual odometry pipeline was implemented in MATLAB 2017a. The code consists of the main file, the pipeline module functions and small helper functions (for matrix calculations, plotting etc.). Tunable parameters are defined directly in the main function. The fundamental pipeline modules are listed in this section while a graphical illustration of the continuous work flow is found in figure 1.

## 3.1 State Definition

The state S includes all relevant parameters that are processed and saved during continuous operation. It is updated with every iteration and includes the following elements:

- **S.P**: Currently used keypoints in actual frame [2 x $numKeypoints$]
- **S.X**: 3D Landmarks to current keypoints P [3 x $numKeypoints$]
- **S.T**: History of absolute camera poses world to $C_i$ [16 x $numRuns$]
- **S.C**: Candidate keypoints in current frame coordinates [2 x $numCandidates$]
- **S.F**: Coordinates of candidate keypoints at first observation [2 x $numCandidates$]
- **S.TC**: Absolute camera pose world to $C_i$ for each element in S.F at first observation [16 x $numCandadates$]

## 3.2 Tunable Parameters

The pipeline includes some tunable parameters that influence the results a lot. They are summarized here together with the chosen default values.

- **Bearing angle limit**: Defines when a triangulation between the first observation of a candidate keypoint and its newest position is qualitatively good

enough. This limit is implemented dynamically. As soon as the 3D landmarks in the state S are critically low (below 100 points), we need to triangulate new ones urgently. Therefore, the limit gets lowered for this iteration. (Default values: 0.5° for normal and 0.1° for lowered case)

- ***KeypointPixelDiff***: Defines the minimum distance in x and y coordinates between two keypoints, in order to be regarded as two non redundant keypoints. (Default value: 2 pixels in each direction)
- **Number of keypoints *numKeypoints***: Defines number of new keypoints obtained from harris scores for initialization and new candidates. Default value is set to 400.
- **Allowed RANSAC error *ransacLimit***: Needed for the eightPointRansac function. (Default value $ransacLimit = 0.1$)

## 3.3 Find new Keypoints

We need new keypoints during initialization and also for generation of new candidate keypoints. The functions harris() and selectKeypoints() are used. To avoid redundancy between the new candidate keypoints and the existing ones, the coordinates in x and y direction have do differ by more than *KeypointPixelDiff*.

## 3.4 KLT

In order to establish keypoint correspondences in the initialization as well as continuous part, we use the MATLAB class vision.PointTracker, which is applying KLT. This allows tracking the points of S.P, S.C and the new candidate keypoints from one image to the next.

## 3.5 RANSAC

To remove outliers in keypoint correspondences, we implemented the function eightPointRansac(). Using random samples of eight points, it estimates the essential matrix that results in the maximum amount of inliers. The errors are calculated as the shortest distance from the query keypoint to the epipolar line. All points with error smaller than the threshold *ransacLimit* are considered inliers. The set of inliers and the essential matrix are returned.

## 3.6 Find Camera Pose

The essential matrix is used to calculate the relative pose between two consecutive frames. The functions decomposeEssentialMatrix() and disambiguateRelativePose() are used to obtain the extrinsic parameters R [3 x 3] and t [3 x 1]. In every iteration, we also calculate the camera pose with respect to the world frame (located at first camera pose) for the triangulation step.

## 3.7 Linear Triangulation

The function linearTriangulation() calculates 3D landmarks based on point correspondences and absolute camera poses of two frames with respect to the world frame. During initialization, correspondences between frame 1 and 3 are used. In the continuous part we use the correspondence between keypoints and their first observation as soon as the bearing angle between them exceeds the bearing angle limit.

## 3.8 Update State

In every iteration we update the state S with the following operations:

- **S.P**: only keep points tracked by KLT and defined as inliers by RANSAC, add points from S.C used for triangulation of new landmarks in this iteration.
- **S.X**: only keep landmarks corresponding to a keypoint in S.P, add new triangulated landmarks from candidate keypoints (between S.F and S.C).
- **S.T**: add the current absolute camera pose to S.T.
- **S.C**: only keep points tracked by KLT and defined as inliers by RANSAC, add new non redundant tracked inlier keypoints of the current frame, remove the points used for new triangulation.
- **S.F**: change accordingly to updates in S.C.
- **S.TC**: change accordingly to updates in S.C.

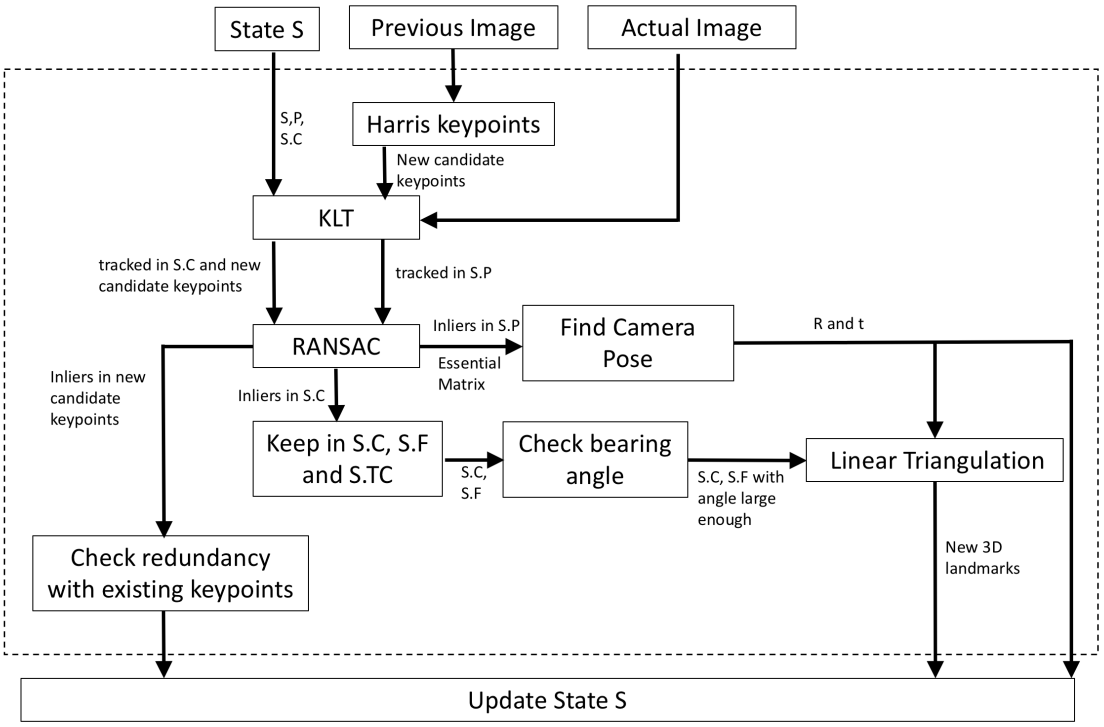


Figure 1: Continuous operation flow chart.

# 4 Custom Dataset

In order to extend the applicability of the framework to self recorded datasets, a smartphone's camera was used to record a dataset for camera calibration and another one for verification of the VO pipeline.

## 4.1 Camera Calibration

The camera calibration was done using the calibration toolbox of MATLAB[1]. The intrinsic parameters of the camera were determined with the known correspondences between pixel coordinates and world points on a set of checkerboard gray scale images, which were taken at various positions and orientations. Due to some scaling during the printing, the size of each square on the checkerboard does not equal the default size used by the program and must be changed in the click_calib.m file (parameters dX_default and dY_default) beforehand.

[1] `http://www.vision.caltech.edu/bouguetj/calib_doc/`

### 4.2 Data

The dataset shows a straight walk in a corridor and contains 425 frames, taken by a handhold smartphone (iPhone 8 with a 12 megapixel camera[2]). The surrounding was chosen to have enough texture and lighting. Images were downscaled to 1/4 of the native resolution to achieve an acceptable computation time.

## 5 Results and Discussion

### 5.1 Key Parameters

Table 1 shows the parameters that have worked for the different datasets. However, they were not optimized for performance. The main parameter that improved trajectory error was RANSAC tolerance. It had to be set to a small value in order to improve trajectory. Although we used RANSAC, there were still triangulated landmarks behind the camera. They can be reduced by optimizing the RANSAC tolerance parameter. The bearing angle was the main parameter that lead to avoidance of crashing due to a low number of landmarks.

Table 1: Key parameters chosen for each dataset

| | **Parking** | **KITTI** | **Malaga** | **Custom** |
|---|---|---|---|---|
| Bearing angle normal[deg] | 0.5 | 2 | 2 | 2 |
| Bearing angle lowered [deg] | 0.1 | 0.1 | 0.1 | 0.1 |
| KeypointPixelDiff [pixel] | 2 | 2 | 2 | 2 |
| Number of keypoints | 400 | 400 | 400 | 400 |
| RANSAC max. epipolar line dist. | 0.1 | 0.1 | 0.1 | 0.1 |

### 5.2 Runtime Characteristics

The evaluation of the final pipeline on the different datasets were performed on a Intel Core i7-4800MQ CPU @2.7GHz. On the KITTI dataset with image resolutions of 1241x376, the calculations during continuous operation took approximately 2.35 seconds per image with the parameters specified above.

[2] `https://www.apple.com/iphone-8/specs/`

Too many landmarks and keypoints in the state slowed down the performance a lot. Additionally, it was observed that the Malaga dataset led to the largest number of lowered bearing angle usage (when less then 100 landmarks are in the state). Especially when entering the roundabout, a lot of keypoints (on the tree and the transmission tower) disappear at the same time. Therefore, new landmarks have to be triangulated immediately to avoid crashing.

## 5.3 Comparison to Ground Truth

For the parking data set we see that the straight direction of movement is estimated correctly locally with some scale drift that can not be avoided using only local computations[3] (see figure 2).
When comparing the results on the KITTI dataset to the ground truth, it can be seen that the angles when changing direction are modeled pretty accurately, while linear distances get distorted (scaled). This is probably due to the fact that too many keypoints are far away from the camera, which leads to large errors in z direction during triangulation[4] (see figure 3).
The Malaga data set has no ground truth data available, however ideally it should be a closed loop. We see that the first 180° curve is estimated correctly, while the second is a bit worse. This is probably due to scale drift that occurs when turning. Combined with the error in linear distance, the trajectory is no longer a closed curve[5] (see figure 4).
The results of our custom dataset are satisfying. The straight direction of movement was estimated correctly and no large outliers in trajectory were observed[6] (figure 5).

[3] https://youtu.be/A010dO0C8xs/
[4] https://youtu.be/6Qhjw7qiLLA
[5] https://youtu.be/YIcQiRIm7ns
[6] https://youtu.be/kWW-mAln7Wc

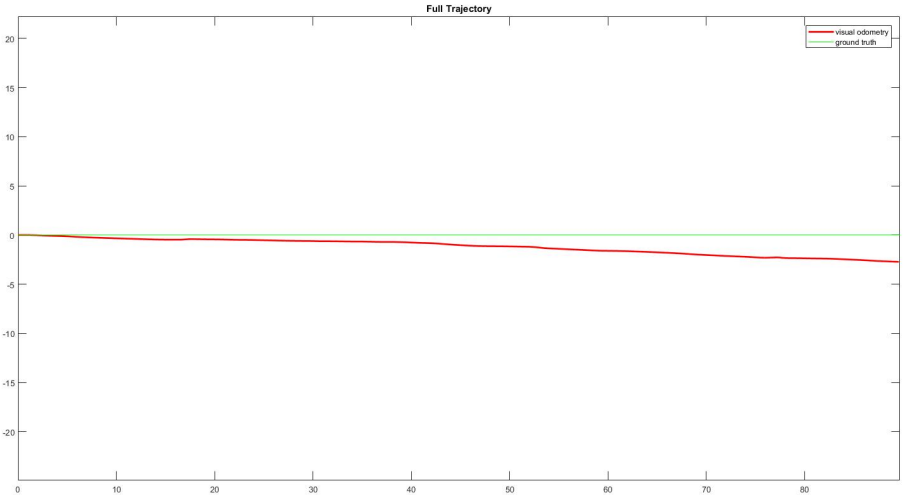


Figure 2: Parking dataset, ground truth in green and VO estimation in red

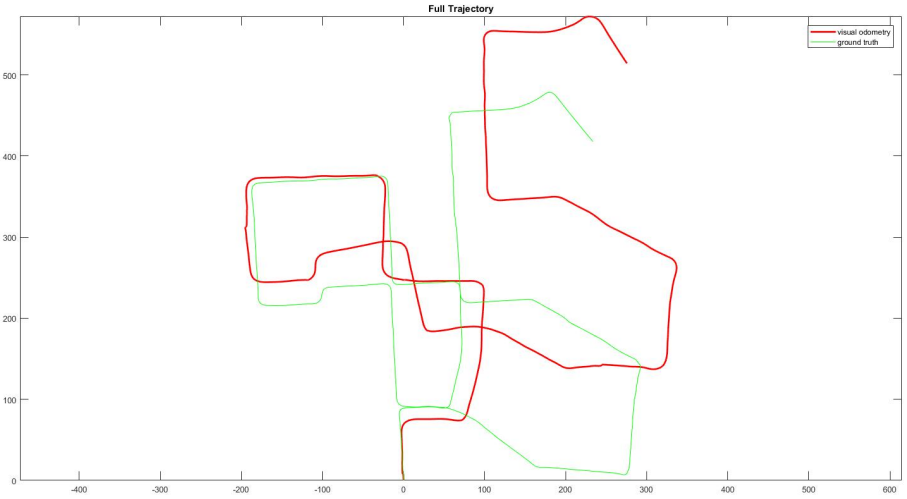


Figure 3: KITTI dataset, ground truth in green and VO estimation in red

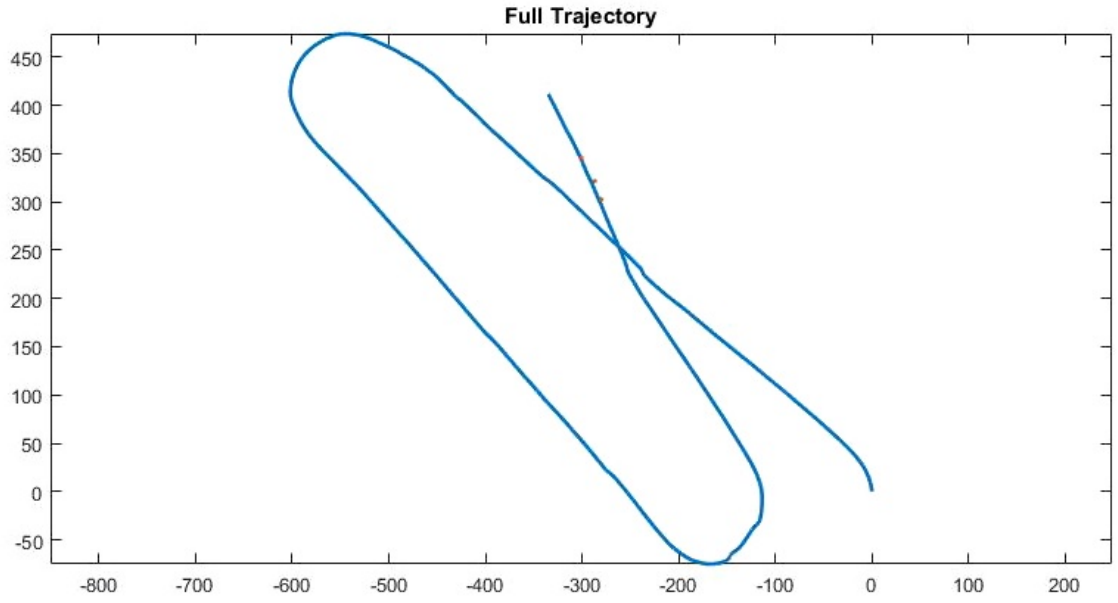


Figure 4: Malaga dataset, without ground truth information

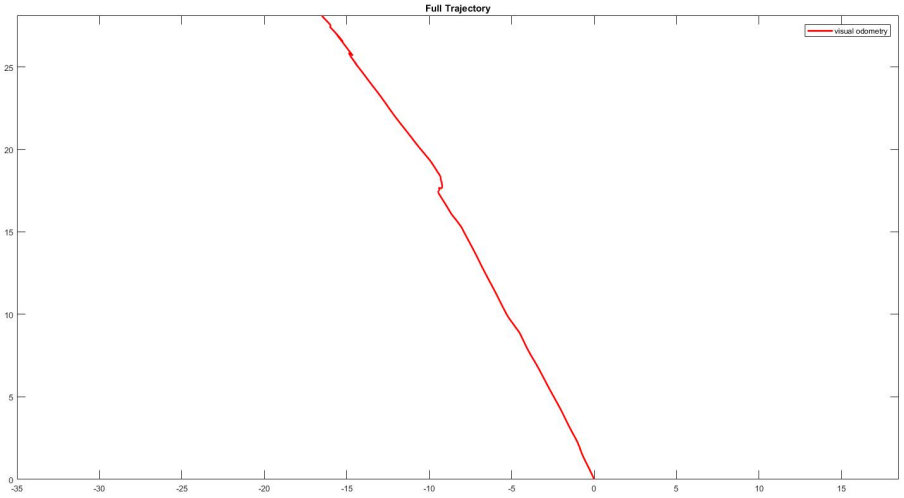


Figure 5: Custom dataset, without ground truth information

# 6 Conclusion

The results of our visual odometry pipeline were satisfying on simple datasets (parking and our custom dataset) as well as on more difficult ones (KITTI). We observed the trade off between computational cost and performance by varying our key parameters. The number of triangulated 3D landmarks in the state was proportional to computation time. Some of the parameters (e.g. the RANSAC epipolar line error threshold) are very sensitive to changes and influence the results a lot. For optimal results, they would have to be tuned much more. Furthermore, we observed some inconsistency in the estimation of the length of a linear path. This is probably due to the fact that a lot of features are far away from and straight ahead of the camera, which was not explicitly counteracted, apart from a threshold for maximum distance. Especially on the Malaga dataset, this effect is observed strongly. Additional information like GPS or stereo images could counteract this issue, and keypoints should be chosen more selectively depending on the distance and local keypoint density. For an improvement on trajectory estimation, graph optimization like bundle adjustment could be applied. On datasets which contain loops (such as Malaga), loop detection would also lead to big improvements.