Universität Zürich[UZH]

Vision Algorithms for Mobile Robotics

Mini Project: Monocular Visual Odometry Pipeline

Group Members

Irem Kaftan

Cafer Mertcan Akcay

January 9, 2021

1 Introduction

The purpose of this mini project is to implement a monocular visual odometry pipeline which performs landmark initialization, keypoint tracking, pose estimation, and triangulation of new landmarks. We successfully implemented a pipeline which satisfies the requirements of the project and demonstrated our pipeline on the KITTI, Malaga, and parking datasets. As an additional feature, we implemented 2-view bundle adjustment in order to refine the camera pose and the positions of 3D landmarks. We also recorded a small dataset with a mobile phone and another dataset from the video game Minecraft in order to test our pipeline with different environments. In the following sections, the methodology will be explained in detail and the results will be discussed.

2 Methodology

2.1 Bootstrapping

In the beginning, an initial set of landmarks is required to estimate the pose of the camera for subsequent frames which is achieved by bootstrapping. Two frames from the beginning of each dataset are selected to bootstrap landmarks (these frames differ based on the dataset). It is important that these frames are not consecutive since small baselines lead to large depth uncertainty but they should also not be so far apart since some of the keypoints are lost as the camera moves. The bootstrapping frames are chosen by trial and error for each dataset so that the initial landmarks are reliable. The procedure of bootstrapping consists of four main operations: feature detection, feature matching, relative pose estimation, and triangulation of landmarks. The Harris features are found for the first frame by using MATLAB's *detectHarrisFeatures()*. They are then tracked in the second bootstrapping frame by using Lucas-Kanade Tracker (KLT) which is implemented by using MATLAB's *vision.PointTracker*. Using the tracked points from the two frames, the fundamental matrix and the inlier points are found by using MATLAB's *estimateFundamentalMatrix()*. The relative camera pose is then computed by using MATLAB's *relativeCameraPose()* function. The first frame is defined as the world reference frame (i.e. rotation = identity matrix, translation = 0) so that the subsequent frames are referenced with respect to the initial position. Finally, a set of initial landmarks are created from the keypoints (i.e. tracked features) by using MATLAB's *triangulate()*. The block diagram of the bootstrapping part is shown in Figure 1 which summarizes the main operations that are explained above.

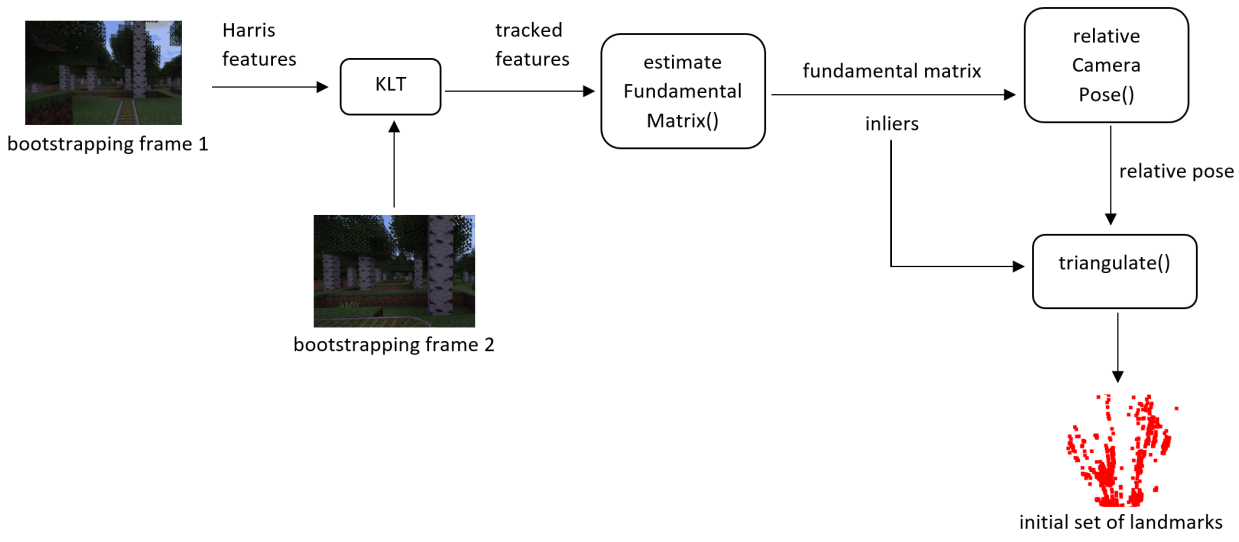


Figure 1: Block diagram of bootstrapping.

2.2 Continuous Operation

Continuous operation works in a Markovian way: the properties of each state are estimated using only the properties of the previous state except for triangulation of new landmarks which uses information from more than one state ago. This Markovian structure reduces the complexity of the algorithm but sacrifices some information from previous states. The procedure of continuous operation consists of three main components. The first one is estimating the current camera pose using the set of landmarks. For this purpose, the keypoints from the previous frame are tracked using KLT. The current camera pose is then estimated by using MATLAB's *estimateWorldCameraPose()* which filters out the outliers by using MSAC which is a derivative of the RANSAC algorithm and outputs the current camera pose by using the P3P algorithm.

The second main component is triangulation of new landmarks because the landmarks eventually go out of field of view. For this purpose, Harris corners are found in each frame. These corners are then added to the candidate keypoints if they survive after filtering based on their distance to the already existing keypoints and candidates. The filtering is applied because there is mostly a cluster of features rather than single distinct features. When a new candidate keypoint is discovered, the camera pose at the first instance of that keypoint is stored together with the image coordinates it is first seen at. At each frame, the candidate keypoints from the previous frames are tracked using KLT. The successfully tracked candidates are then triangulated using their first image coordinates, the camera pose where they are first seen at, their current image coordinates, and the current camera pose. The angle between the rays from the two camera positions (the first and the current) towards the triangulated landmarks is calculated as in Eq. 1.

$$\alpha = \arccos \frac{ray0 \cdot ray1}{\|ray0\| \|ray1\|} \tag{1}$$

In Eq. 1, ray0 corresponds to the ray from the camera pose where the candidate keypoint is first seen at towards the triangulated landmark and ray1 corresponds to the ray from the current camera pose towards the triangulated landmark. If α is larger than a predetermined threshold, triangulated landmarks are added to the set of landmarks. Otherwise, they are discarded and the candidates wait for the following frames. This operation is required to ensure that there is enough baseline between the two frames used for triangulation so that there is smaller depth uncertainty.

The last component of the continuous operation is two-view bundle adjustment. Two-view version is used because it fits the Markovian structure and has little complexity. Tracked keypoints are stored as *pointTrack* objects at each frame transition and the refined poses and 3D landmark positions are computed by using MATLAB's *bundleAdjustment()*. At the end of processing each frame, refined poses and refined landmarks are passed to the next frame. The block diagram of the continuous operation part is shown in Figure 2 which summarizes the main components that are explained above.

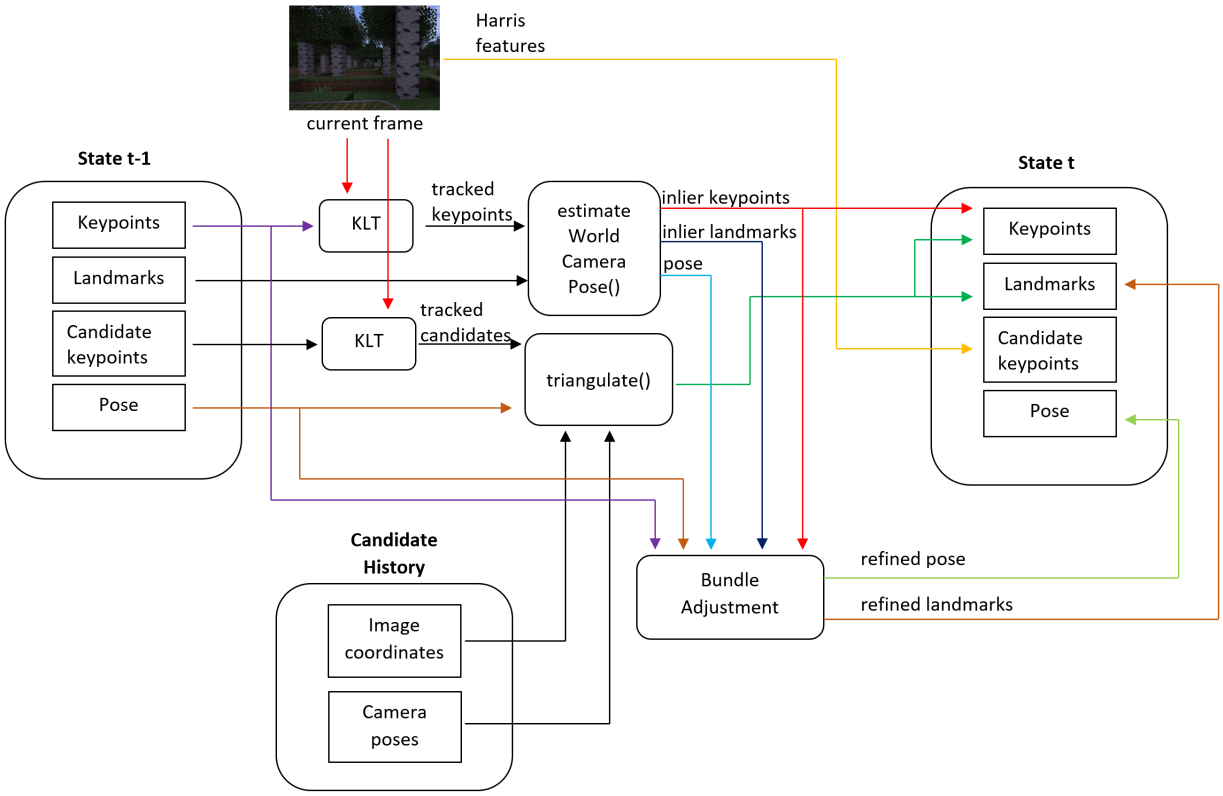


Figure 2: Block diagram of continuous operation.

3 Results

3.1 KITTI, Malaga, and Parking Datasets

The pipeline is first applied to the KITTI, Malaga, and parking datasets and the results can be seen in this playlist. Please note that the figures are saved as images first and then the video is created from images to have a constant 2 fps. The actual runtime is similar to videos (1-2 fps) but the time between the frames is not uniform.

Overall, despite having some noisy parts, local trajectory (trajectory of last 20 frames and landmarks) is accurate in the KITTI, Malaga, and parking datasets. However, in the KITTI and Malaga datasets, global trajectory is accurate until the pipeline loses scale. At this point, the scale between the consecutive frames becomes so small that the global trajectory updates become unnoticeable. In the parking dataset, the global trajectory is also accurate because it does not lose scale. This is because the motion in the parking dataset is constant in one direction and the lighting and the environment are also constant which are suitable conditions for the VO pipeline.

The trajectory update while the car is making a turn in frame 447 (from the initial parts of the KITTI dataset) can be observed in Figure 3 and the trajectory update while the car is making a turn in frame 2427 (from the later parts of the KITTI dataset) can be observed in Figure 4. Note that the last 20 frames in Figure 3 span an interval between 180 and 190 in x-axis while the last frames in Figure 4 span an interval between 85.0072 and 85.0075 in x-axis. These trajectories are both locally correct but their scale ratio is about 10^5.

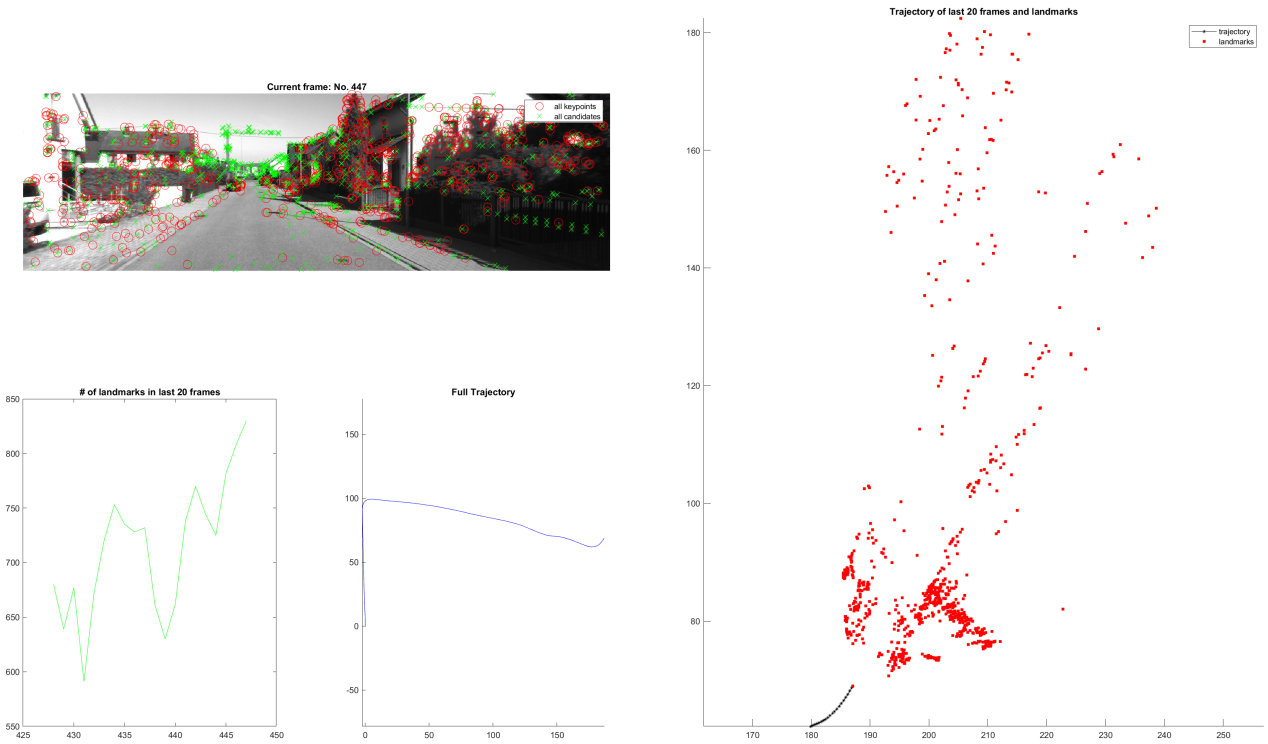


Figure 3: Trajectory update in frame 447 of the KITTI dataset.

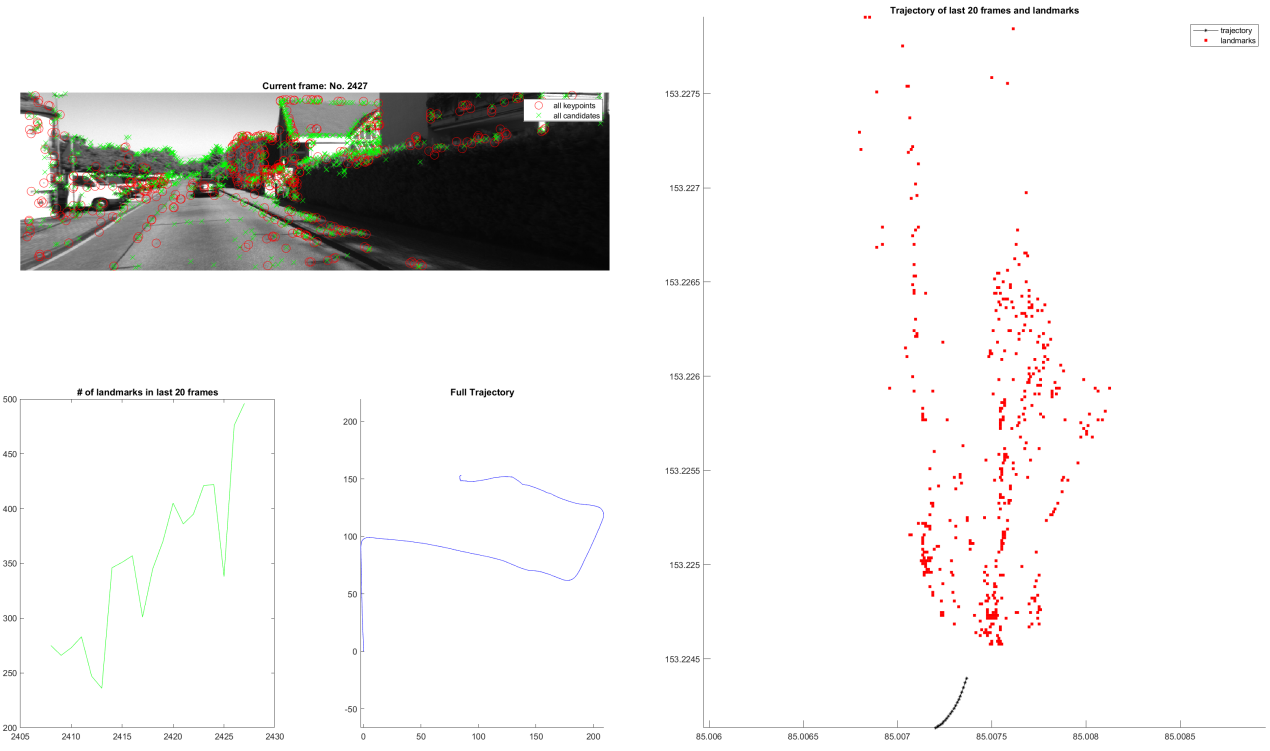


Figure 4: Trajectory update in frame 2427 of the KITTI dataset.

While analyzing the results, we wanted to discover what causes this significant scale drift. We know that it is inevitable to have scale drift in monocular vision but there were frames with drastic changes in scale. We realized that this happens at instances where there are sudden illumination or viewpoint changes which violate the assumptions behind the algorithms used in our pipeline. For example, KLT assumes that the brightness does not change much between consecutive frames. At those frames, large amount of landmarks from the previous frame cannot be tracked due to changes

and they are lost which leads to large scale drift. For example, sun washes out most of the pixels on white houses on the right side of the road in Figure 5. Furthermore, the shadows of the houses on the road lead to frequent changes in brightness. Therefore, this street was one of the places with noisy estimations and the global trajectory was lost there.

Figure 5: Sudden illumination change in frame 628 of the KITTI dataset.

3.2 Custom Datasets

We concluded that the scale is lost significantly in the KITTI and Malaga datasets because of the challenging environments and the large motion between frames. Especially, KLT requires a small motion displacement between frames (1-2 pixels) because it makes a first order approximation. KLT also needs small illumination changes around points as it calculates SSD using the patch around them. Therefore, we wanted to investigate whether we can estimate a global trajectory close to the real trajectory (up to a scale) if these requirements of the algorithms were violated less. In the parking dataset, these are not violated much and the global trajectory is accurate but it does not have different types of motions such as turning right or left. Hence, we wanted to record our own datasets.

3.2.1 Market Dataset

The KITTI and Malaga datasets are recorded on a car and, for that reason, the motion between the consecutive frames is mostly large. This is why we decided to have a dataset recorded while walking. We also wanted to have an environment with relatively constant brightness. Markets are illuminated uniformly and there are interesting trajectories possible so we decided to record a dataset in a market with a smartphone.

We calibrated the smartphone (Samsung Galaxy Note 5) using the app VIZARIO. It uses checkerboard photos to calculate the calibration matrix. We then recorded the video which is in the playlist together with its results. The video is 30 fps but we skip one of each two frames when selecting frames from the dataset because the motion between the consecutive frames are tiny. It estimates the local trajectory okay but its performance is worse than the other datasets. The biggest problem is that the video is not recorded with a stabilizer and the camera moves a lot while walking which causes pixels to move in a motion like pendulum. The motion between the two frames in Figure 6 should be on a straight line (epipoles must be the same) but, as it can be seen, the camera moves sideways due to walking. This leads to large pixel displacement between the frames.

(a) Frame 693 of the market dataset. **(b)** Frame 719 of the market dataset.

Figure 6: Camera swing in the market dataset.

3.2.2 Minecraft Dataset

In order to create the ideal world for our monocular visual odometry pipeline and test its performance in this ideal world, we recorded a dataset from the video game Minecraft. It is a good candidate of this ideal world because the illumination is constant, the motion between the frames is small, and there are no occlusions. We also ensure that the movement is smooth by making the character move in a cart on a railway. We calibrate the virtual camera (i.e. the eye of the player in the game) by using the following information: the principal point is the center of the frame, the field of view (θ) is 105 degrees, and the pixels are square. The dimensions of each frame is 2512x1440 (width x height) pixels so the principal point located at $(1256, 720)$. The focal length in pixels (α) is then calculated as in Eq. 2.

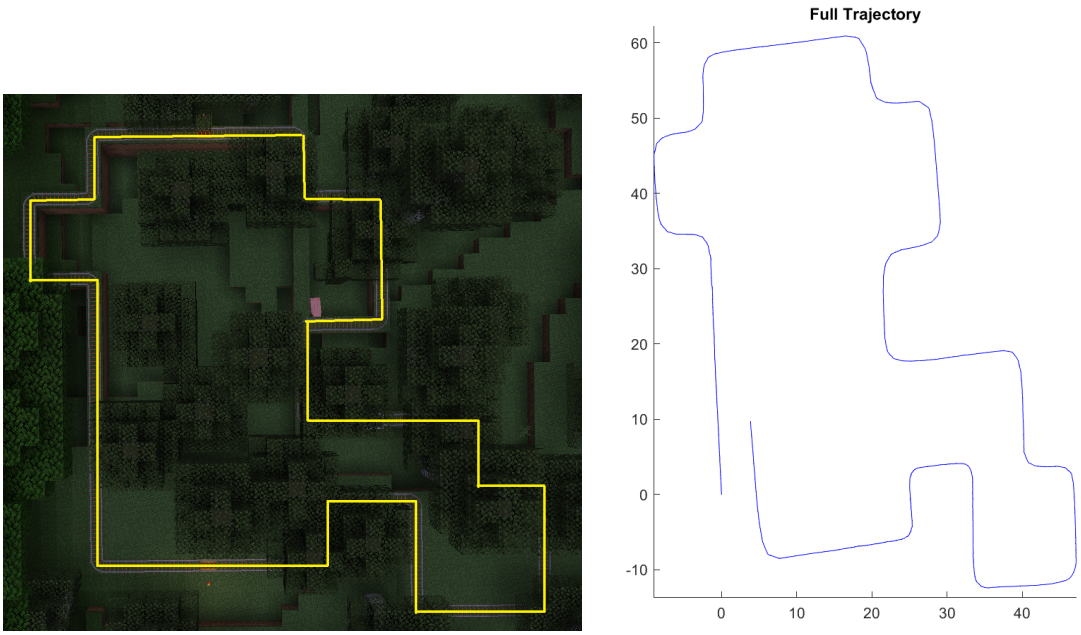
$$\tan\frac{\theta}{2} = \frac{w}{2\alpha} \rightarrow \tan 52.5 = \frac{1256}{\alpha} \rightarrow \alpha = \frac{1256}{\tan 52.5} = 1610 \tag{2}$$

As the pixels are assumed to be square, we have $\alpha_u = \alpha_v = \alpha$. An example frame from the dataset is shown in Figure 7.

Figure 7: Example frame from the Minecraft dataset.

A path with many turning points is prepared in the game in order to test the pipeline. Its results together with the gameplay is in the playlist. In the end, the resulting global trajectory is very close to

the ground truth trajectory as shown in Figure 8. It is still not perfect but, comparing to other datasets, it is much closer to the ground truth trajectory. Hence, it can be concluded that, in an ideal world, monocular visual odometry would estimate the global trajectory more accurately (up to a scale) and the scale drift would be less.



(a) Ground truth trajectory. **(b)** Estimated global trajectory.

Figure 8: Ground truth trajectory and estimated global trajectory.

4 Conclusion

In this mini project, we had an opportunity to observe the concepts that we learned in the lectures. As mentioned in the results part, sudden illumination or viewpoint changes and occlusions decrease the performance of the pipeline drastically since the previously tracked keypoints can vanish from the scene suddenly. We also realized that the correct selection of parameters depends on the dataset and boosts the performance of the algorithms significantly which is the reason why we used different parameters, such as different bootstrapping frames, for each dataset. For example, for the KITTI dataset, we mostly have more keypoints and landmarks compared to other datasets in order to make it more robust to sudden changes. Moreover, we observed the negative effect of a not stabilized camera on the performance of the pipeline when we tested it with our custom market dataset. We also created another custom dataset to mimic ideal world conditions (constant illumination, small motion between consecutive frames, and no occlusions) and found out that the global trajectory is more accurate and there is less scale drift in this case. Overall, we were able to successfully implement a monocular visual odometry pipeline and interpret its performance on different datasets.