Mini-project: A visual odometry pipeline

Boyan Duan (19-952-514), Mengqi Wang (19-754-027), Ming Yi (17-734-427)

1 Overview

This report summarizes the workflow and building blocks of a monocular visual odometry pipeline. The pipeline is composed of two main parts: an initialization module and a continuous VO module. At last, we will discuss different feature detectors as an additional feature.

2 Initialization

Two view geometry is used to estimate the relative pose between two selected frames, and triangulate the point cloud of landmarks from tracked keypoints. The following procedure is discussed using the parking dataset.

We proceed as follows:

1. Frame 1 and frame 5 are selected due to optimal keypoint correspondence.
2. Keypoints in frame 1 are detected using Harris detector. For keypoint tracking, we used the Matlab class *vision.PointTracker*, which using the Kanade-Lucas-Tomasi (KLT), feature-tracking algorithm. To remove ouliers in keypoint correspondences, Matlab function *estimateGeometricTransform* is part of *vision.PointTracker* class. Additionally, manually selection including removing points with negative depth and depth larger than 15 times of the median depth are used to filer out remaining outliers.
3. Estimate the relative pose between two frames and triangulate a point cloud of landmarks for remaining keypoints.

3 Continuous operation

3.1 Associating keypoints to existing landmarks

With the inlier keypoints and their associated landmarks from previous initialization, the next step is to propagate to next frame. Since keyframe selection is an essential step in VO and should always be done before updating the motion [1], we added a threshold for the keyframe selection using rule of the thumb formula:

$$\frac{\text{keyframe distance}}{\text{average depth}} > threshold \tag{1}$$

The threshold is set to be 10% in this setting. So for each new frame following the initialization frames, inlier keypoints are continuously tracked and the ratio of frame distance and average depth of the image is computed to find the next keyframe.

3.2 Estimating the current pose

We use the 2D-3D motion estimation. Given 3D points P in keyframe coordinate, and corresponding 2D pixels p in current frame, we first estimate the pose of the current frame by RANSAC and P3P, which gives the pose R_{CW}, t_{CW} of the keyframe coordinate in current frame coordinate.

However, the results of the RANSAC-P3P algorithm tend to be unstable and inaccurate. R, t always jitters, which results in failure of later estimations of the pose and triangulation. Therefore, we add an extra step with nonlinear refinement to stabilize the result and make estimations more accurate.

We use the Matlab function *lsqnonlin* to minimize the lost function $|q - p|$, where q is the equivalent homogeneous pixels of $K \cdot (R \cdot P + t)$ expending in 1D array. To be more specific, $q = (q_1^T, q_2^T, ...q_n^T)$ and $q_i = (u_i, v_i, 1)^T$ homogeneous to the ith column of $K \cdot (R \cdot P + t)$ and R, t are expressed as variables in screw coordinates.

We can see a significant improvement of predicted trajectory from the two figures below. The predicted trajectory of the parking dataset with nonlinear refinement is basically consistent with the ground truth in the first 30 frames while the predicted trajectory without refinement is far from ground truth.

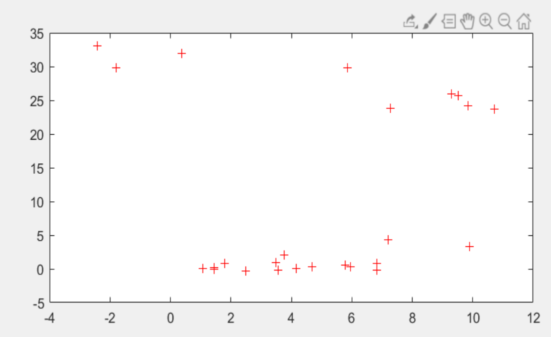

Figure 1: Trajectory before refinement

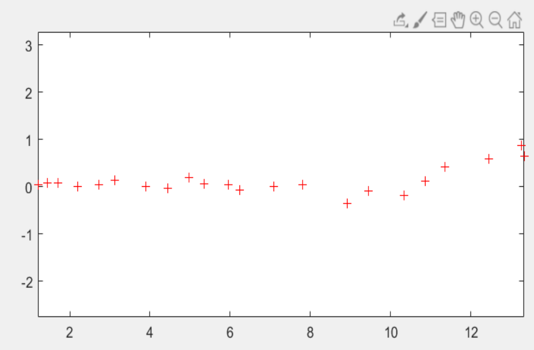

Figure 2: Trajectory after Non-linear refinement

3.3 Triangulating new landmarks

At each frame, we calculate the pose of the frame with respect to the keyframe, if the distance satisfies the condition above, we define the current frame as the new keyframe. And each time we add a new keyframe, we recompute the common features between the new keyframe and get the pixels positions $p1, p2$ in two frames. We also have the pose R_{CW}, t_{CW} of the previous keyframe relative to the current keyframe. Then we can get the 3D position of the common landmarks from triangulation.

After triangulation, we remove the points that are not ideal. The instruction mentioned to take only the points which forms angle α with two camera center that is large enough. We use a simplified version of that, which simply verify that the distance of the points with the camera center is not too large relative to the distance between two camera center.

We also considered the option of the non-linear approximation of the triangulation to minimize the re-projection error. However, it doesn't improve the performance much, and it is also very time consuming, since there are thousands of variables at each triangulation (the number of variables is 3 times the number of points).

4 Results

We give four plots as results during the pipeline:

1. An image of the tracked and matched points.
2. the number of tracked and matched points.
3. the global trajectory of the predicted pose.
4. the local predicted trajectory.

The first image is given by plotting the current and previous position of the keypoints on the image. In the thrid and forth image, we only consider the trajectory on the xz plane, since the car is moving on the plane in all datasets, and the x,z direction is fixed. In each frame we calculate the relative pose of it from the keyframe given by $R_1 = R_{CW}^T, t_1 = -R_{CW}^T \cdot t_{CW}$. and then the relative pose to initial frame $R = R_k \cdot R_1, t = t_k + R_k \cdot t_1$, where R_k, t_k are the pose of the keyframe relative to initial frame. In the third image we plot all the position of the frames in initial coordinate. In forth image, we plot the position of adjacent 20 frames and the position of landmarks in keyframe with respect to the initial frame. An example of plotting is shown below.

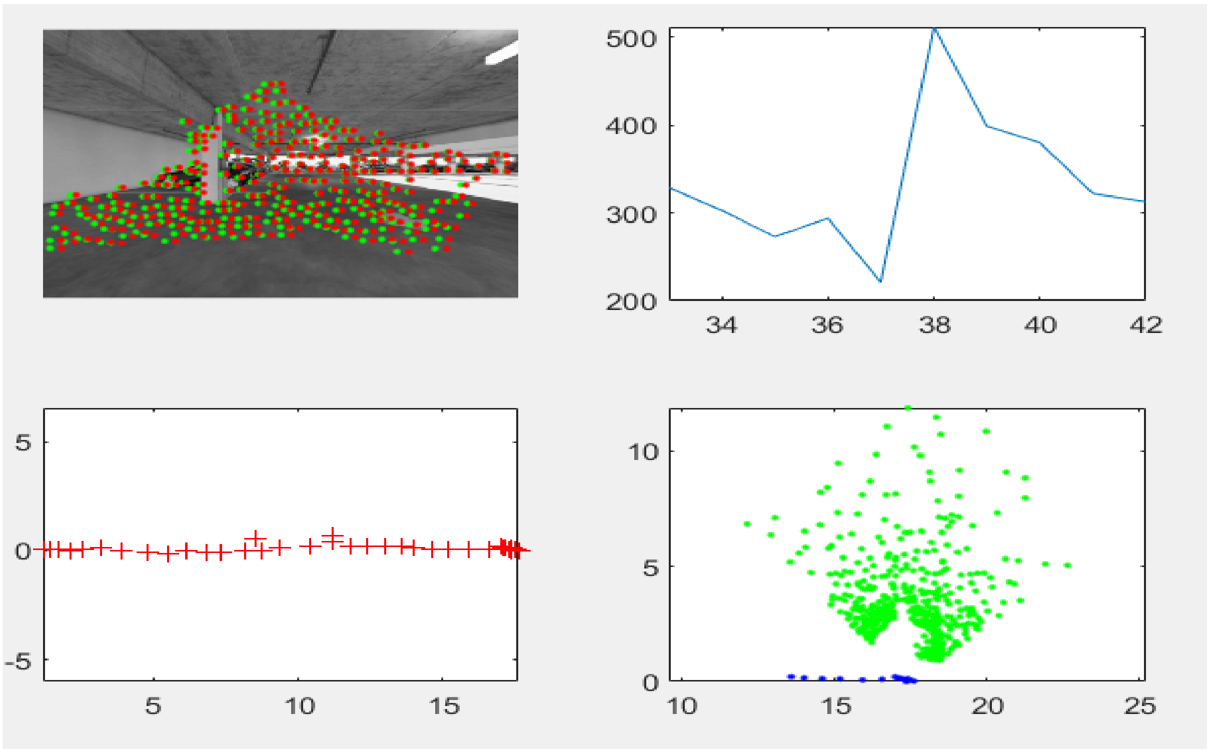

Figure 3: Example of plotting

5 Additional Feature

Finally we explore the use of different features. We find that the different features give similar results if the number of feature points is large enough.

In order to have good estimations of the pose and enough landmarks, we need a large number of features. A small number of features may lead to a match only locally and therefore might not give an accurate estimation of the pose. The RANSAC algorithm also requires a sufficient number of common features to yield good results, and the number of features will also diminish over the tracking between two keyframes. Taking Harris detector as an example, in order to have a large number of global distributed features, we can set the nonmaximum supression radius and the number of keypoints large. As shown in the figure, if the number of features are small and distributed only locally, it will lead to a inaccurate estimate of pose. The left figure is set to have 800 features with nonmaximum supression radius 12, and the right figure has 500 features with nonmaximum supression radius 8.

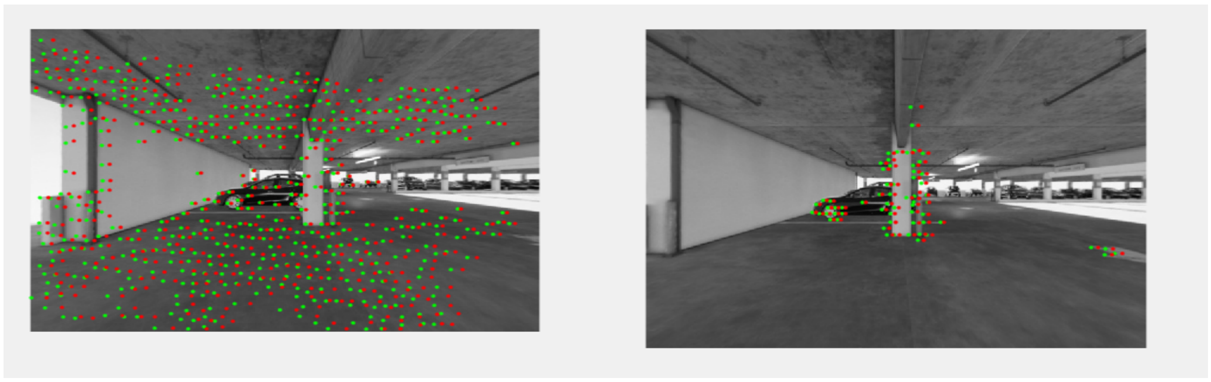

Figure 4: Tracked and matched keypoints.

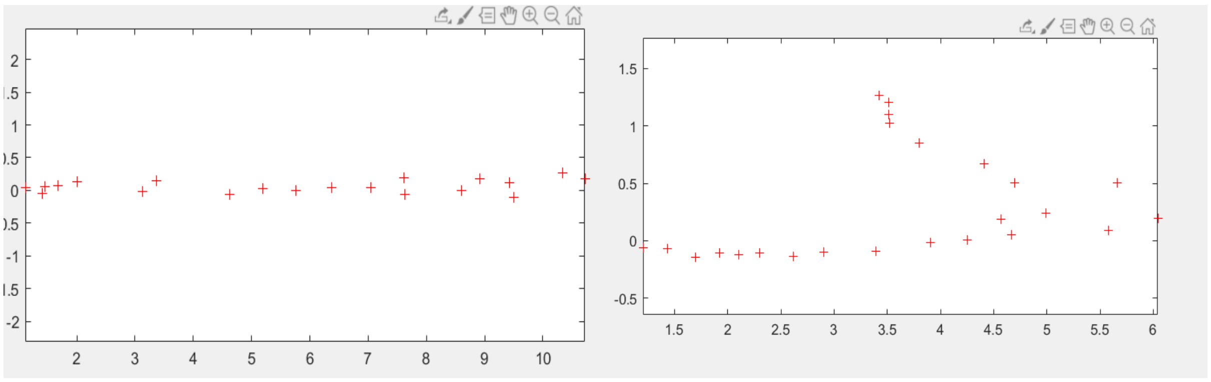

Figure 5: Predicted trajectory of first 30 frames

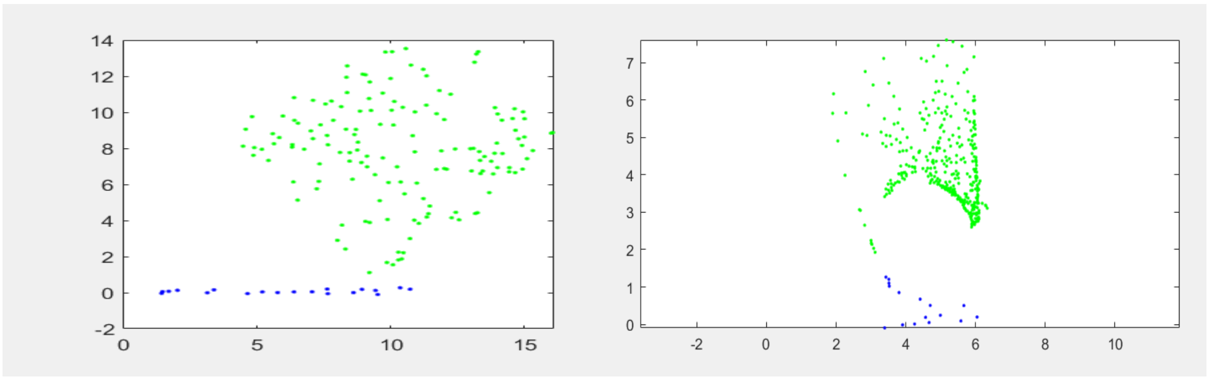

Figure 6: Local trajectory and triangulated landmarks.

Next, we compare different detection methods under the condition of large number of features and matching by tracking. In order to compare different feature detection methods, we test them on the first 100 frames of the parking dataset. We compare the following methods: 1 Harris, 2 SURF, 3 FAST, 4 BRISK, 5 Grid points 6 Random points. The Grid method directly takes a grid of points in keyframes and then do tracking. The random method take random points in keyframes and then do tracking. The Harris and Grid and Random give a fixed number of points in single image while the other methods give different numbers of points in different images.

We propose that the Harris, SURF, Grid and Random method give similar results locally, no matter of corner of blob detector or even random selected points. It is because that we use the same tracking method KLT which will track points taking the global image as a clue, and the position of points doesn't matter much. And if the number of keypoints is large, all methods might have a global distribution of points, and we also filters out outliers and the points that are not ideal in other step. Therefore all methods might give a good approximation of the pose. The FAST,BRISK method gives worse results since they doesn't give sparse features and therefore the tracking and matching is not ideal.

The trajectory comparisons are given below:

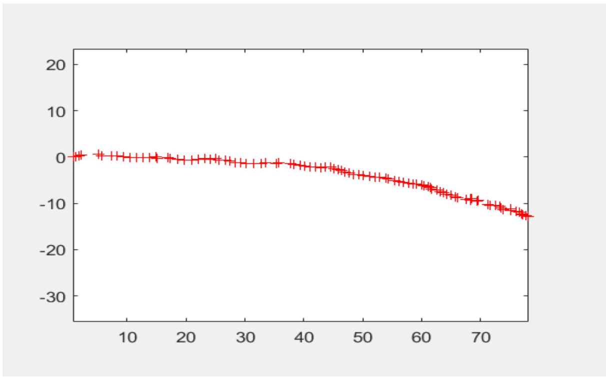

Figure 7: (a)

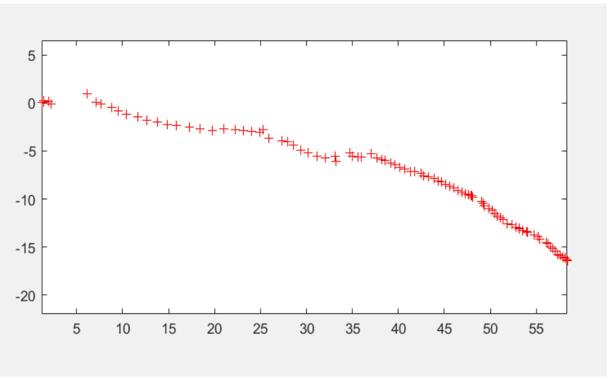

Figure 8: (b)

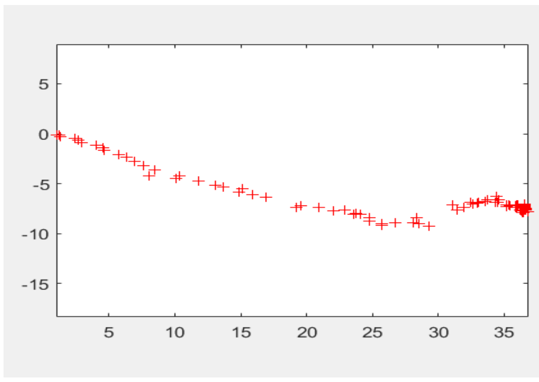
Figure 9: (c)

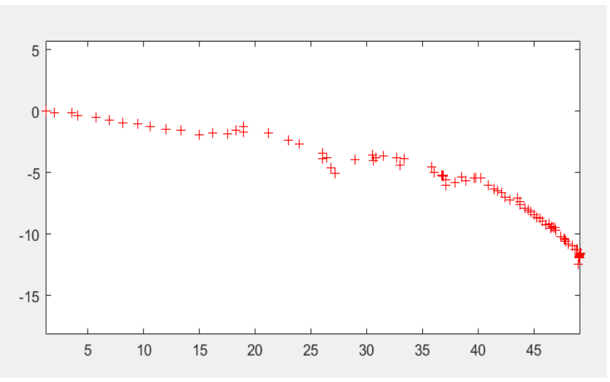
Figure 10: (d)

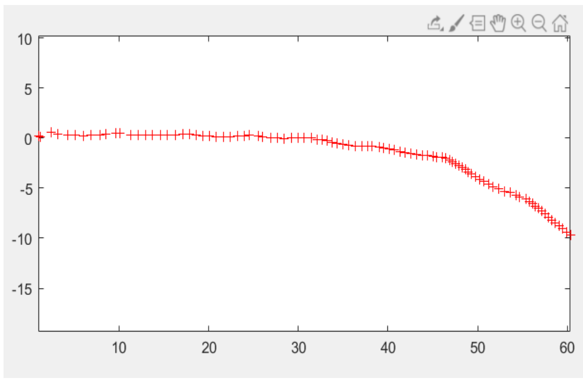
Figure 11: (e)

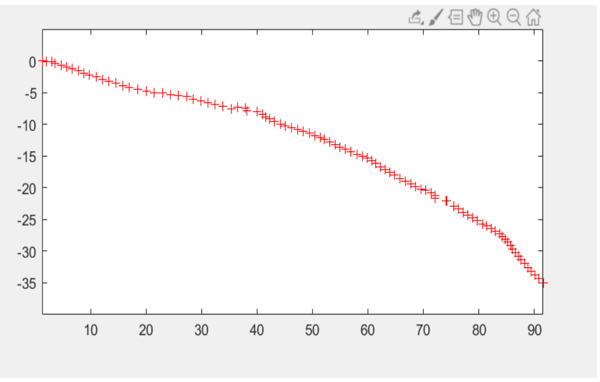
Figure 12: (f)

Figure 13: Trajectory of 6 methods: (a) Harris method; (b) SURF method; (c) FAST method; (d) BRISK method; (e) Grid of points method; (f) random points method.

The ground truth is a evenly distributed horizontal trajectory, from the plotting we find that the Harris, SURF, Grid and Random method basically gives locally good trajectory, while FAST,BRISK has worse performance.

Number of inliers(tracked and matched landmarks) over frames and tracked keypoints between keyframes are shown in table:

method	mean kpts	mean inlier	raito	min inlier	ratio
Harris	758	299	39.4%	78	10.3%
SURF	1814	682	37.6%	377	20.8%
FAST	3947	1792	45.4%	64	3.6%
BRISK	4986	2297	37.8%	392	7.9%
Grid	2825	1389	49.2%	781	27.6%
Random	4548	2224	48.9%	595	13.1%

From the result we conclude that all methods give a good number of inliers in average, while sometimes the FAST and BRISK give less keypoints matching, which may lead to worse results. A possible explanation might be that the FAST and BRISK tend to be dense, and the tracked and matched inliers might be locally distributed which might cause error in further processing. Whereas the other methods are somehow sparsely distributed and tend to yield better results.

References

[1] D. Scaramuzza and F. Fraundorfer. Visual odometry tutorial. *IEEE Robotics Automation Magazine*, 18(4):80–92, Dec 2011.