VAMR Mini Project-A Visual Odometry Pipeline

Jianhao Zheng, Yujie He
{jianhao.zheng, yujie.he}@epfl.ch

I. Introduction

This course project implements a full Visual Odometry (VO) pipeline by integrating the knowledge we learn from the lectures and exercises. The entire pipeline consists of the initialization part and the continuous operation, which is implemented basically by the recommended procedure. We further add a form of sliding window bundle adjustment (BA) to optimize the estimation. Despite the provided three datasets, we also collected our dataset from a parking area at EPFL and regions in the center of Lausanne. Our VO pipeline works in all five datasets. In addition, we compute the quantitative evaluation and benchmark the feature detector methods and VO with/without pose refinement and bundle adjustment. In summary, our main extra feature includes: topsep=0pt,itemsep=-1ex,partopsep=1ex,parsep=1ex

1) Implementation of bundle adjustment and provide both quantitative and qualitative analysis on the improvement from BA.
2) We collect two customized data sequences with careful calibration and image pre-processing.
3) We present a quantitative benchmark over four popular feature detectors, i.e., SURF, BRISK, FAST, and HARRIS.

The performance demonstration videos of our proposed VO pipeline on three provided and two customized data sequences are uploaded to this link.

II. Method

A. Initialization

In the initialization part, we follow the recommended procedure. We first manually choose two distant frames at the beginning to generalize a point cloud of landmarks. For all datasets, the first frame I^1 and the third frame I^3 are selected. We then extract features from the frame I^1 with **SURF** feature detector. These features are tracked in the frame I^3 by **KLT** tracking. Then, we establish key points matching between I^1 and I^3.

With the matched key points, we utilize the eight-point algorithm with **RANSAC** to filter outliers and estimate the relative camera pose between I^1 and I^3. With the knowledge from exercise 6, we can triangulate the matched key points and obtain a landmark point cloud.

Remark 1 One of the extensions that we add is a procedure to filter landmarks based on their depth to the camera of frame I^3. We empirically set a range of depth $\boldsymbol{\mathcal{D}_{valid}} = [\boldsymbol{d_{min}}, \boldsymbol{d_{max}}]$ for each dataset. Every newly triangulated landmark will be translated to the camera coordinate of frame I^3, i.e. $P_{C3} = T_{C3,W} * P_W$. Only those landmarks whose depth is in the given range, $P_{C3}(3) \in \boldsymbol{\mathcal{D}_{valid}}$ will be appended to the initialized point cloud. By adding this procedure, points too far from the camera with high uncertainty are filtered out. Similarly, points that are wrongly triangulated behind the camera frame are deleted to reduce possible errors.

B. Continuous operation

In continuous operation, we implement similar extensions with several of our extensions. To briefly summarize, we first track the key points of the last frame by **KLT**. With the pixel positions of the key points in the current image and their 3D coordinates, we use the **P3P** algorithm with **RANSAC** to filter outliers and estimate the absolute camera pose of the current frame. In order to keep the number of existing landmarks, we extract features from each new frame and append those new features whose distance to any of existing candidates or key points are larger than a given **threshold** $\boldsymbol{r_{\text{th}}}$. The pixel position in the frame which first observe it and the pose matrix of that frame is stored for every new key point. In the future frame, we track the candidates with **KLT**. For those tracked candidates, they will be triangulated and appended to existing landmarks as long as the angle $\alpha(c)$ in figure 1 is greater than a tuned **threshold** $\boldsymbol{\alpha_{\text{th}}}$.

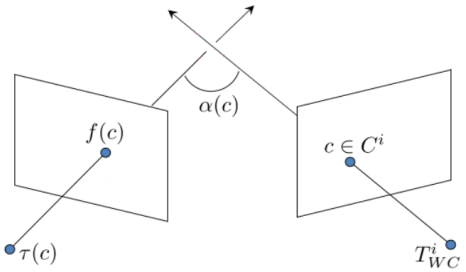


Figure 1: Illustration of the angle $\alpha(c)$. **Directly copied from project statement.**

Remark 2 One extension we add after estimating the current pose by P3P is to refine the estimated camera pose by minimizing the reprojection error:

$$\underset{T_j \in \text{SE}(3)}{\arg\min} \sum_{i=1}^{n_j} \|K * T_j * \bar{P}_i - \bar{p}_i\|^2 \tag{1}$$

where T_j is the translation matrix of frame j, K is the intrinsic matrix, n_j is the number of key points tracked, $\bar{P}_i = [P_i^T,\ 1]^T$ is the homogeneous 3D coordinate of landmark i and $\bar{p}_i = [p_i^T,\ 1]^T$ is the homogeneous pixel coordinate.

We optimize the equation 2 by the MATLAB function **lsqnonlin** and set the camera pose estimated by the P3P algorithm with RANSAC as the initial guess. A quantitative comparison on this extension can be found in section IV.

Remark 3 Note that we also apply the depth-based filtering procedure when we triangulate a new candidate key points in the continuous operation.

Remark 4 Since the way to calculate the angle $\alpha(c)$ in figure 1 not provided in the statement, we briefly describe how we calculate this angle. For a candidate point i We basically have the pixel coordinate in the current frame p_c^i and that in the first detected frame p_f^i, the pose of the current frame $T_{W,C}$ and that of the first frame $T_{W,F}$. We can get the translation between the first and current frame $T_{C,F} = {T_{W,C}}^{-1} * T_{W,F}$. We can further obtained the relative rotation matrix by $R_{C,F} = T_{C,F}(1:3, 1:3)$. Then, the angle can be computed as follow:

$$\alpha^i(c) = arccos(\frac{\bar{P}_C^{i,F} * \bar{P}_C^{i,C}}{\|\bar{P}_C^{i,F}\| * \|\bar{P}_C^{i,C}\|}) \tag{2}$$

where $\bar{P}_C^{i,F} = R_{C,F} * K^{-1} * [p_f^i,\ 1]^T$ is the normalized homogeneous pixel coordinates at the first detected frame (expressed in current frame coordinate) and $\bar{P}_C^{i,C} = K^{-1} * [p_c^i,\ 1]^T$ is that at the current frame.

C. Bundle adjustment*

We define k to be the number of key frames we optimize in BA and we sample the key frames with constant distance k_d since most of our test data set is in almost constant velocity. For each frame i we have the camera pose $T_{i,W}$ and the observation consisting of number of key points detected in this frame, pixel coordinates of key points and key points indices in point cloud $O_i = [k_i, (p_i^1)^T, ..., (p_i^{k_i})^T, l_i^1, ..., l_i^{k_i}]^T$ (same notation as that in exercise 9 is used). We also has the 3D coordinate of the landmarks $\mathcal{P} = [P_1^T, ..., P_m^T]^T$.

Instead of doing BA every time a new frame comes, we do BA **only when a new key frame is reached**. Assuming the index of the first key frame is idx, we have the following frames to be optimized $\mathcal{F} = \{\boldsymbol{F_{idx}}, F_{idx+1}, \cdots, F_{idx+k_d}, \boldsymbol{F_{idx+k_d+1}}, F_{idx+k_d+2}, \cdots, \boldsymbol{F_{idx+(k-1)*(k_d+1)}}\}$ where the key frame is in bolded. We first optimize the key frames and the 3D coordinates of key points detected in at least two key frames by the reprojection error:

$$\underset{T_0,...,T_{f(k-1)},\mathcal{P}_{\text{valid}}}{\arg\min} \sum_{i=0}^{k-1} \sum_{j=1}^{k_{f(i)}} \|K * T_{f(i)} * \bar{P}_{l_{f(i)}^j} - \bar{p}_{f(i)}^j\|^2 \tag{3}$$

where $f(i) = idx + i * (k_d + 1)$ is the index of the ith key frame and the $\mathcal{P}_{\text{valid}}$ is the set of 3D coordinates of key points detected in at least two key frames. Again, $\bar{x} = [x^T,\ 1]^T$ means the homogeneous form of vector x. We implement this optimization by organizing the variables in the same form as that in exercise 9 and directly use **the *runBA* function we implemented in exe 9**.

After this step, we will optimize the translation of the normal frames among the key frames, i.e. $\{F_{idx+1}, \cdots, F_{idx+k_d}, F_{idx+k_d+2}, \cdots, F_{idx+(k-1)*(k_d+1)-1}\}$, by reprojection error with the optimized 3D coordinate of landmarks. The formulation for this step is similar to equation 2.

Remark 5 Our BA approach succeeds in the parking dataset and improves the performance of the VO pipeline. More results and comparison can be found in section IV. However, we find that when implementing this in the Kitti dataset, the *lsqnonlin* function takes much time to compute and outputs unreasonable results in some frames after 50 frames. Even though it improves the estimation in the first 50 frames, it can not be implemented in the full Kitti dataset. We conjecture this is because the Kitti dataset is more challenging than the parking dataset. Thus, the reprojection error is harder to be optimized.

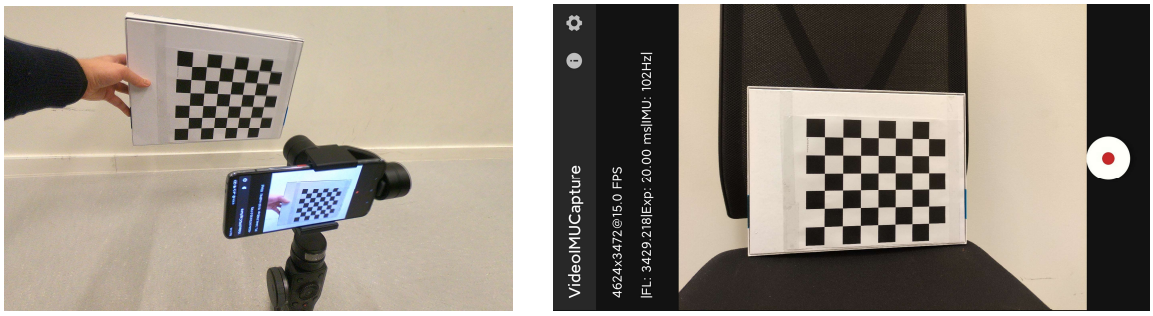
III. Customized Dataset*

For this project, two customized data sequences are compiled to evaluate the proposed method in diversified scenrios, i.e., `epfl_parking` and `lausanne_center_nav`.

A. Data collection

1) **`epfl_parking`** sequence is recorded with the open-source toolkit for a Android phone with a wide-angle camera len. To avoid motion blur, the Zhiyun Smooth 4 gimbal is used to mount the phone with 3 DOF stabilization as shown in figure 2. The original data is recorded at about 15.0 frames per second (FPS) with 4624x3475 pixels. The detailed camera calibration procedure can be referred to Section III-B, and images are resized and downsampled to about 3 FPS.
2) **`lausanne_center_nav`** sequence is recorded with RealSense Depth Camera D435 mounted on a mobile robot in Lausanne center street. The original data is recorded at about 14 FPS with 640x480 pixels. Because the camera is pre-calibrated, rectified images and camera parameters are extracted from recorded rosbag and downsampled to about 5 FPS.

Exemplary challenging factors in the sequences are shown in figure 3. For the image from `epfl_parking` sequence, illumination has great changes when the view from the entrance to the parking lot inside, leading the key points dropping from 600 to 150. In terms of



(a) Recording calibration sequence with checkerboard when camera mounted on the gimbal

(b) The screenshot of the VideoIMUCapture app. The data is recorded with 4624x3472 pixels @15.0 FPS

Figure 2: Camera calibration with checkerboard

Figure 3: Challenging factors in customized data sequences: illumination variation from the entrance into the parking lot (left, No. 26 frame in `epfl_parking`) and large moving object in the field of view (right, No. 57 frame in `lausanne_center_nav`)

`lausanne_center_nav` sequence, some moving pedestrians from frame No.50 to No.60 occlude large areas of the image, which poses great challenges to keypoint matching.

B. Camera calibration & image pre-processing

Two-step calibration is performed for the Android wide-range camera. As shown in figure 2, the 7x6 checkerboard pattern with a square size of 3 cm is used as the calibration pattern. The kalibr [1] toolbox are utilized for camera and IMU calibration. Besides, ffmpeg is used to downsample from 15/30 FPS to about 5 FPS before generating the processed rosbag, which could reduce redundant information in the dataset and reduce the runtime of the calibration.

For camera calibration, The camera system is fixed and the calibration target is moved in front of the cameras to obtain the calibration images. In this step, the pinhole camera model and radial-tangential distortion model are chosen for convenient processing for the VO pipeline and processing images by using OpenCV or MATLAB. For camera-IMU calibration, the checkerboard pattern is fixed, and the camera-imu system is moved in front of the target to excite all IMU axes.

As a result, the maximum error in reprojected pixels is less than 10.0, and the mean reprojection error in pixels is 1.97. For more details, please refer to the calibration folder[2].

After calibration, image resizing and undistortion operations are performed on the original images. Compared to original high-resolution images captured at high frequency, images are subsized and downsampled into similar resolutions as the provided datasets. For more details about data collection and calibration, please refer to the code[3].

C. Discussion

Compared to the established dataset for visual odometry evaluation, our compiled data sequences lack a feasible device to generate ground truth pose for benchmarking. Therefore, only undistorted image sequences and calibrated camera parameters are provided for qualitative evaluation. For future work, the calibrated IMU data can also be used for visual-inertial odometry for better pose estimation.

[1] https://github.com/ethz-asl/kalibr

[2] Link to detailed calibration results

[3] GitHub repository for camera calibration and image preprocessing

IV. Experiments

A. Evaluation metric

We follow the method described in the provided slides[4] to compute the relative error. To briefly summarize, we take different distance $d \in D$. For the frame F_d that has navigated d [m] away from the first frame, we do the trajectory alignment between the pose estimation from F_1 to F_d and the ground truth. Then, we calculate the relative error by:

$$err(d) = |\Delta\hat{t}_{1,d} - \Delta t_{1,d}| \tag{4}$$

where $\Delta\hat{t}_{1,d} = \hat{t}_d - \hat{t}_1$ is the aligned estimated replacement between frame F_d and F_1 while $\Delta t_{1,d} = t_d - t_1$ is the ground truth. Note that $|\cdot|$ is the element-wise absolute value function, not the vector normalization function.

B. Quantitative evaluation

Three quantitative comparisons have been performed for our pipeline. We first compare the performance of our VO with different feature detection methods. The second evaluation is to compare the performance of VO with and without pose refinement as described in **remark 2**. The last one is to compare the result of VO with and without BA. The first two experiments will be conducted on the Kitti dataset since it is the popular benchmark dataset. As stated above, BA cannot work in all the frames of Kitti, we do the last comparison on the parking dataset. We conduct **five experiments** for each single setting and report the mean and std of the quantitative evaluation.

Feature detector benchmark We compare the VO performance when using **SURF**, **BRISK**, **FAST** and **HARRIS** as the feature detector. For a fair comparison, we first tune the parameter of each detector to ensure that the number of key points extracted from the first frame is similar for different detectors. Any other settings are all kept the same. We do the experiments on the Kitti dataset and choose the traveled distance set $D = \{10m, 40m, 90m, 160m, 250m, 360m\}$ to compute the relative distance.

Figure 4 shows the relative error of three axes with different feature detectors. One can observe that all the methods perform close except **BRISK** suffers a bit in the estimation on Z-axis. In general, we find the estimation with **SURF** as the detector has slightly lower error. Moreover, we provide the operation efficiency for our full VO pipeline using different detectors (not just for detecting features). As expected, **FAST** is the quickest one, while **SURF** outperforms the rest two in a substantial margin. Note that we all activate the pose refinement for all the experiments in this part. SURF (W/O PR) is for the incoming part.

Remark 6 Considering the result in this part, we utilize **SURF** as our feature detector in the remaining experiments.

[4]Workshop report *How to Run Reproduciable Visual SLAM Experiments* by Zichao Zhang, Davide Scaramuzza

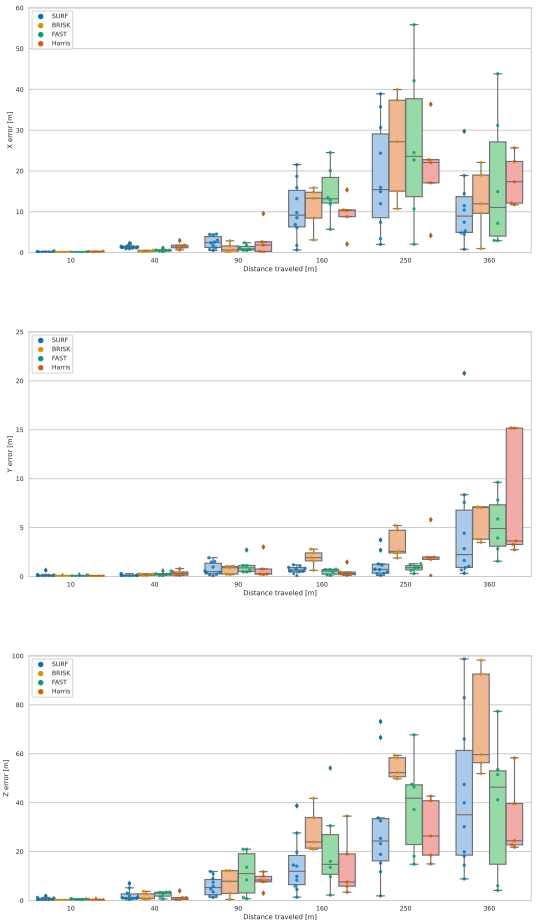


Figure 4: Relative error comparison among the proposed VO pipeline with **different feature detector methods** (Best view in color and zoomed in).

Table I: Operation efficiency with different features on `KITTI seq05` sequence. FPS stands for frame per second.

Methods	SURF	SURF (W/O PR)	BRISK	FAST	Harris
FPS	4.09 ± 0.24	4.11 ± 0.24	3.35 ± 0.02	4.24 ± 0.10	3.81 ± 0.03

Evaluation on pose refinement Figure 5 shows the evaluation on our VO pipeline with and without pose refinement after computing camera pose by P3P with RANSAC. In general, activating pose refinement improves the performance of the full VO pipeline, especially in the Y-axis, which is supposed to be 0 all the time. If we do not activate pose refinement, the estimated pose in the Y-axis starts to have error closer to 10 m. Moreover, the standard deviation of the error along Y and Z axes has been significantly reduced by pose refinement. Thus, the estimation can be more robust and stable if using such refinement.

In table I, one can observe an interesting finding that VO with pose refinement (**SURF**) is a bit faster than that without

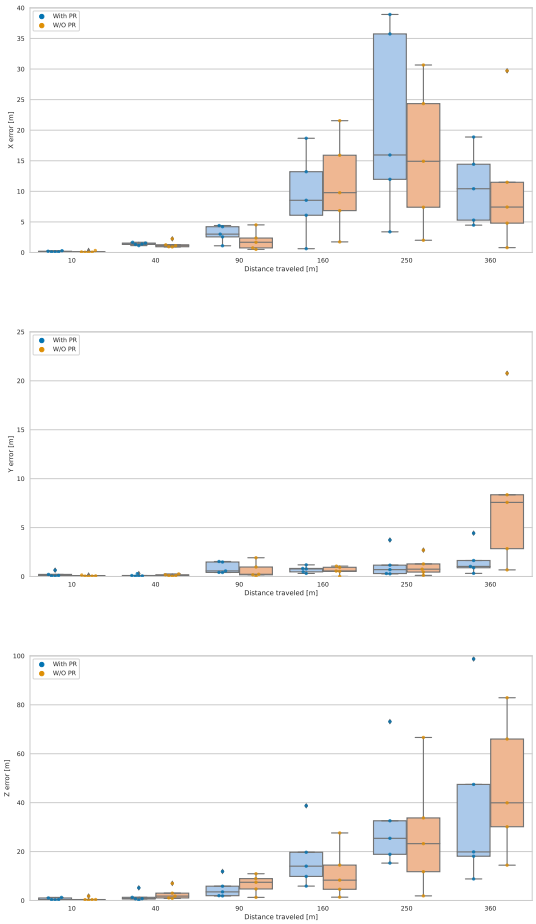


Figure 5: Relative error comparison on the proposed VO pipeline **with and without pose refinement** (Best view in color and zoomed in).

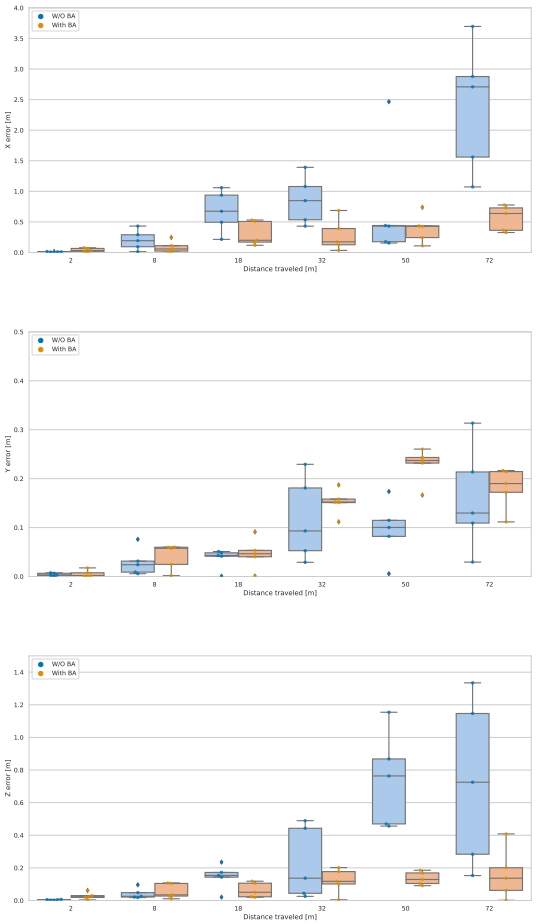


Figure 6: Relative error comparison on VO **with and without bundle adjustment** (Best view in color and zoomed in).

refinement (**SURF (W/O PR)**). We think this is because the computational time cost by pose refinement is trivial, and the fact that VO with refinement is fast in our evaluation is just due to the uncertainty of the measurement.

Evaluation on pose bundle adjustment

For evaluation on the effect of BA, we do the test on the parking dataset since BA cannot continue in the Kitti dataset as stated in section II. The traveled distance set to compute the relative distance is chosen as $D' = \{2m, 8m, 18m, 32m, 50m, 72m\}$ The comparison on whether BA is activated or not can be found in figure 6. The error significantly reduces when BA is utilized. Note that the error of VO with BA in Y-axis is more or less similar to that without BA. However, the Y-axis's error scale is far smaller than the other two. The errors of the two methods in the Y-axis are all acceptable. Thus, we should focus on comparing error on X and Z axes where we can observe the improvement brought by BA.

As we can see in table II, adding BA, however, reduces the computational speed a lot. Whether activating BA in the VO pipeline is a trade-off on the computational power in hand and the accuracy required.

Table II: Operation efficiency with or without bundle adjustment on `parking` sequence

Methods	SURF (With BA)	SURF
FPS	1.68 ± 0.11	4.56 ± 0.04

C. Qualitative evaluation

Key points matching. Figure 7 shows the key points detected in the first frame of the Kitti dataset, while we report the matched positions on the third frame by **KLT**. The result shows that just few points is failed to be tracked. In Figure 9, we can observe the inliners estimated by the eight-point algorithm with **RANSAC**, where one can find

that lots of matched key points in featureless regions, such as shadows and road, are eliminated as outliers.

Figure 7: The keypoints detected in the first frame of `KITTI seq05` sequence by **SURF** detector.

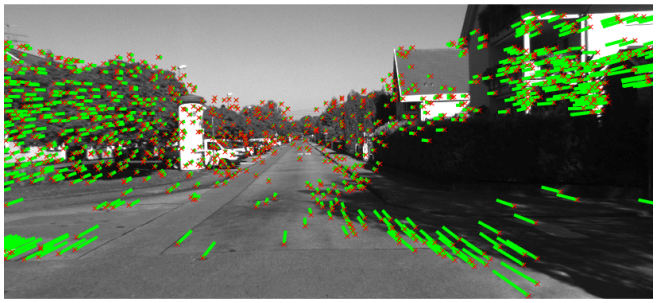


Figure 8: Valid keypoints tracked in the third frame of `KITTI seq05` sequence by **KLT**.

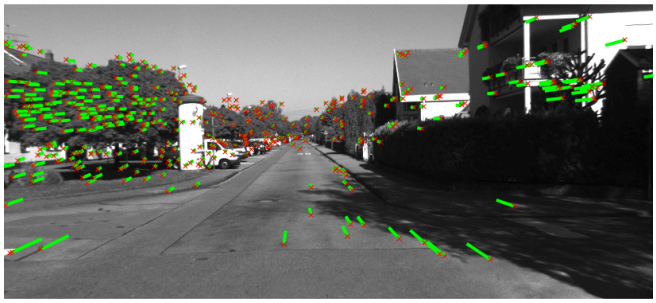


Figure 9: Inlier keypoints remained after **RANSAC**.

Full trajectory. Figure 10 to figure 12 shows the estimated trajectory, aligned trajectory, and the ground truth over 10 m, 160 m, and 360 m away from the first starting points. From figure 10, we can find that our aligned estimation is highly accurate in the local (short) movement. The scale of the movement starts to drift when it moves 160 m far as can be seen in figure 11. However, the rotation is still being estimated accurately as the aligned trajectory is very close to the ground truth in the term of direction. In figure 12, the scale drift degrades the performance furthermore, while the rotation estimation is relatively accurate. Since we only implement VO without other sensor information, the estimation inevitably suffers from scale ambiguity. Thus, the scale drift is reasonable in our case.

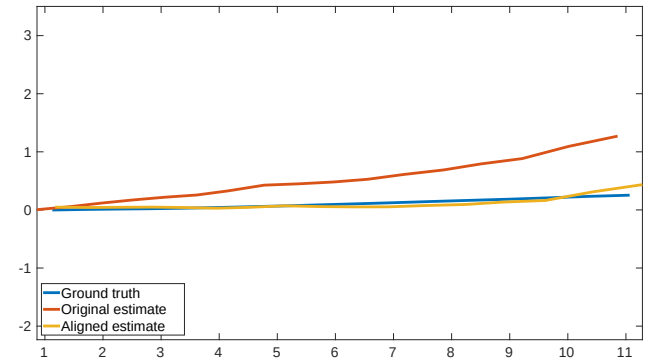


Figure 10: Estimated, aligned and ground truth of trajectory after navigating 10 m since the first frame (Kitti).

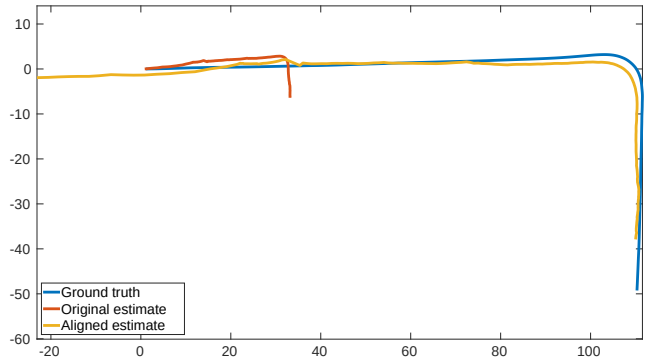


Figure 11: Estimated, aligned and ground truth of trajectory after navigating 160 m since the first frame (Kitti).

Bundle adjustment. We present the aligned estimated trajectory and the ground truth of the parking dataset for VO with and without BA in figure 14 and figure 13. Note that we do not plot the original estimated trajectory to present a clear comparison as the original one may occlude some part of the ground truth. From the comparison, we can see that the estimation by VO with BA outperforms that without BA in a clear margin. The estimation by VO with BA has almost no error compared to the ground truth, while without BA generates a clear curve-like trajectory instead of a straight line. Moreover, the starting point and ending point estimation without BA does not align with the ground truth due to its larger-scale drift. Our qualitative result is also consistent with our findings in the quantitative result.

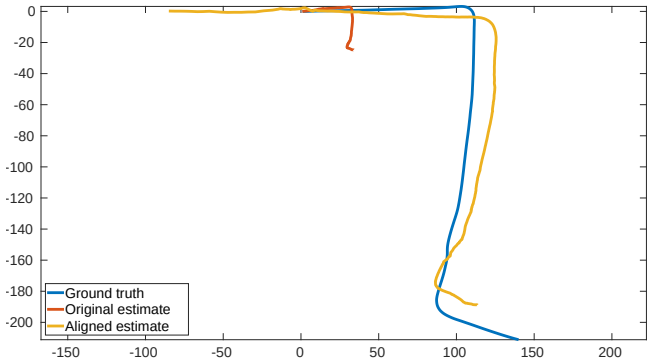


Figure 12: Estimated, aligned and ground truth of trajectory after navigating 360 m since the first frame (Kitti).

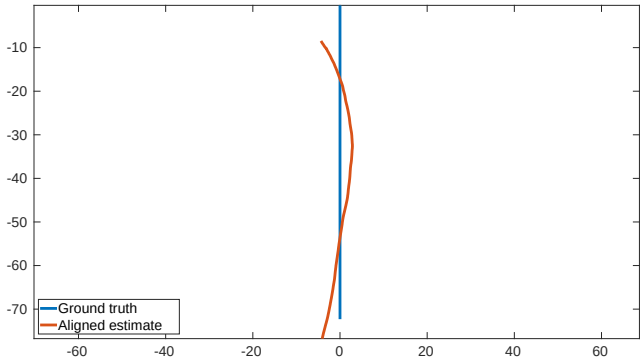


Figure 13: Aligned and ground truth of trajectory after navigating 72 m since the first frame without BA (parking).

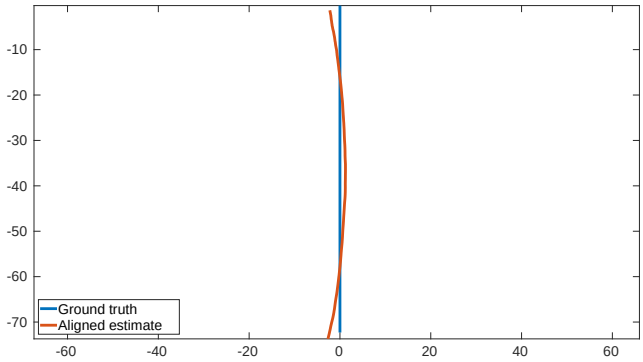


Figure 14: Aligned and ground truth of trajectory after navigating 72 m since the first frame with BA (parking).

Video recording. For the three provided data set and our two customized datasets, we have recorded videos to show the performance of our VO pipeline. These videos can be found in this link.

V. Conclusion and Limitations

In conclusion, we successfully implemented the full VO pipeline based on the recommended procedure. We also added several additional features such as pose refinement, bundle adjustment, customized datasaet, quantitative evaluation, and benchmark test for feature detectors. The most time-consuming part of this project is finding suitable parameters and finding bugs. We find that even many small bugs can accumulate to make the full pipeline performs far worse. It is quite important to ensure each part is in high quality. However, it is still very accomplished for us when seeing our VO generate an accurate estimation. The only pity is that we could not let BA work on the Kitti dataset. Possible future works are trying to figure out why **lsqnonlin** takes so long time in some frames and try the *bundleAdjustment* function from MATLAB.