COMP0130: ROBOT VISION AND NAVIGATION

Coursework 03:

Evaluation of ORB-SLAM2

Jian Zhou

21082500

ucabj42@ucl.ac.uk

Xianjian Bai

21135608

ucabxb1@ucl.ac.uk

Hanpeng Li

22072479

ucabhl9@ucl.ac.uk

April 24, 2023

Question 1

1.a ORB-SLAM2 uses an architecture which consists of a front end and a backend. Explain the role of each of these two main components, and how they interact with one another.

ORB-SLAM2 is a sophisticated SLAM system that operates on a two-tiered architecture, comprising a front-end and a back end, each fulfilling essential roles within the overall framework.

The front-end primarily processes raw sensor data, such as images or depth maps, and executes feature extraction utilizing the highly efficient ORB (Oriented FAST and Rotated BRIEF) technique. This component carries out frame-to-frame tracking by matching the extracted features from the current frame with those from the previous frame or local map points. Such matching is vital for accurately estimating camera poses. Additionally, the front-end is tasked with selecting keyframes based on strategic criteria, including scene coverage, feature abundance, and motion dynamics, to ensure a compact and informative map representation. These selected key frames are the key to the interaction between the front end and the back end.

Conversely, the back end focuses on optimizing and maintaining both the map and the camera trajectory. It receives keyframes and preliminary estimates from the front-end and performs local mapping by generating new map points through the triangulation of matched feature pairs in keyframes. The back end further refines local map optimization by conducting local bundle adjustment, which fine-tunes camera poses and 3D map point positions to minimize reprojection errors. Loop closure detection and correction are also managed by the back end, utilizing the Bag of Words (BoW) method to identify potential loop closures between keyframes. Once a loop closure is detected, the back end introduces constraints between interconnected keyframes, constructs a pose graph, and optimizes it to rectify accumulated drift. Moreover, the back end oversees map point management, eliminating redundant or outlier points and fusing duplicates to enhance overall map quality.

1.b ORB-SLAM2 uses keyframe-based representation of the map. What is the keyframebased representation? What are the advantages of using keyframes versus other approaches?

Keyframes are a subset of the frames taken from cameras. These keyframes are selected by the front-end threads under the consideration of the amount of camera motion, the number of features obtained in last interval and the distance to previous keyframe.
Using keyframes has multiple advantages versus other approaches like Dense SLAM, Direct SLAM and Grid-based SLAM. The most significant one is that it can considerably reduce the computation effort of the system and reduce the usage of memory, compared with Dense and Direct SLAM. Secondly, it is more robust as keyframe representation select based on new information and this can limit the effect of validation of camera. Moreover, as it only claims a new key frame when there is enough information, it can make the robot rover in the environment for a longtime without memory overflow or exponential growth in computational demand.

1.c. Run ORB-SLAM2 on the *nostructure_texture_near_withloop* dataset.

Present the results graphs you get in both cases, and explain why the two differ from one another.

Evo_ape is a command that calculates the absolute pose error. APE is a measure of the distance of the estimated camera pose compared to ground truth. From figure 2b, we can observe the two similar circles in the map although they are in different poses. It can be deduced that the although the APE is large, the performance of the SLAM is not as worse as the APE indicate.

After an alignment of the map, two circles overlap with each other fairly well (observed from Figure 2). And the APE number has decreased by two orders of magnitude.

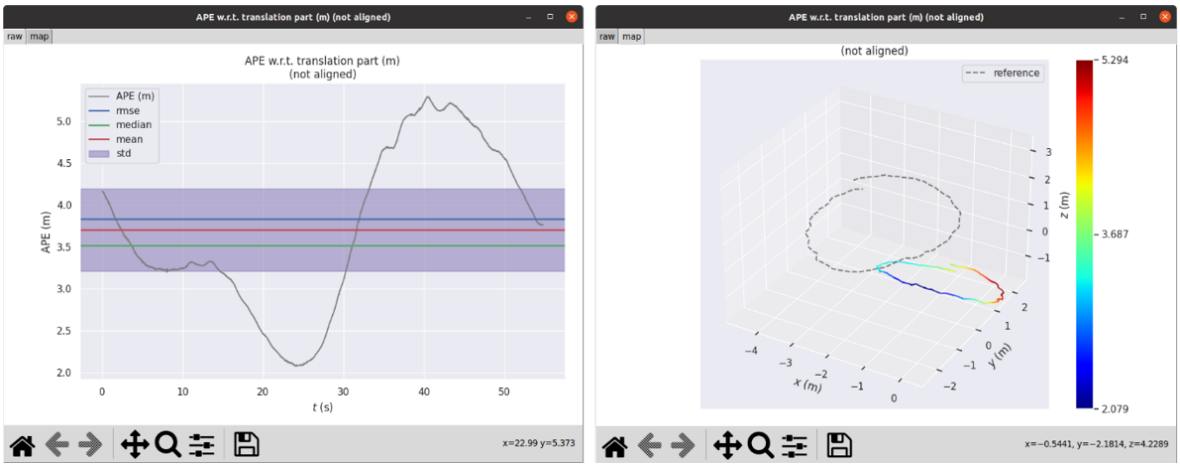

Figure 1: Results of unaligned conditions

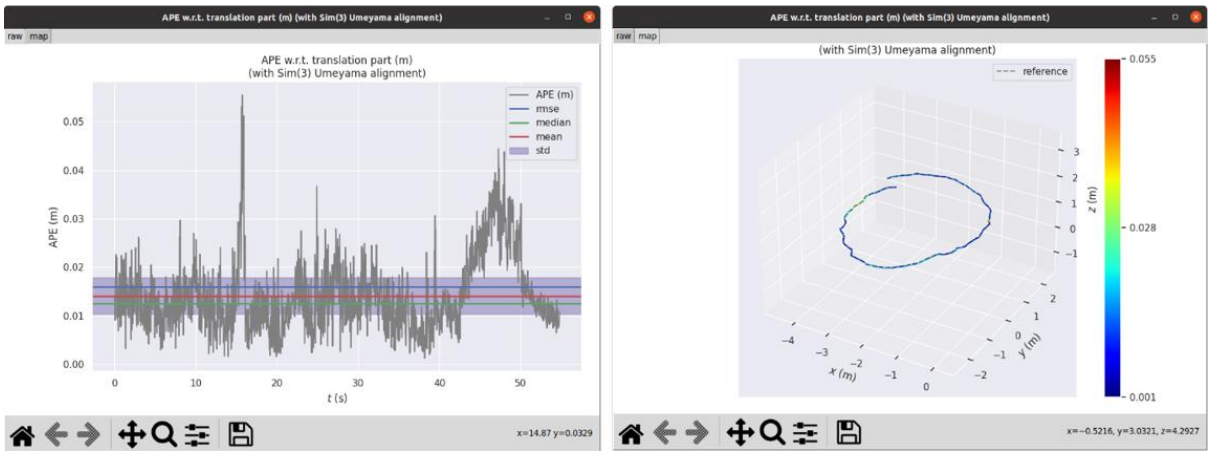

Figure 2: Results of aligned conditions

When is the alignment process likely to fail?

In this case, the Umeyama alignment is used for aligning two point sets using the SVD. It will compute the centroids of both point sets and the covariance of them. Then the optimal rotation matrix is calculated using two rotation matrices from the SVD and scaling factor that minimizes the difference between the two point sets. In the end the optimal rotation matrix is applied to one of the point set.

After understanding the methods of alignment, we can infer that the alignment would fail when the two point sets have too many outliers. The alignment process is not likely to be significantly affected by a few outliers. However, the approach can not bear when there are a large number of outliers.
Another possible situation is when the two point set have large different scales or orientations, where it can be challenging to align them accurately.

It is also possible to fail when the density of the two sets of points is too low. If the point sets have limited resolution or density, it can be challenging to find enough matching points to perform an accurate alignment. The algorithm may not be able to detect and match enough corresponding points in the two point sets, resulting in an inaccurate or failed alignment.

Question 2

2.a Briefly describe how the monocular initialization algorithm works. Why does the algorithm attempt two different models (fundamental matrix and homograpy)? Why is initializing with a monocular camera significantly harder than using stereo or RGBD cameras?

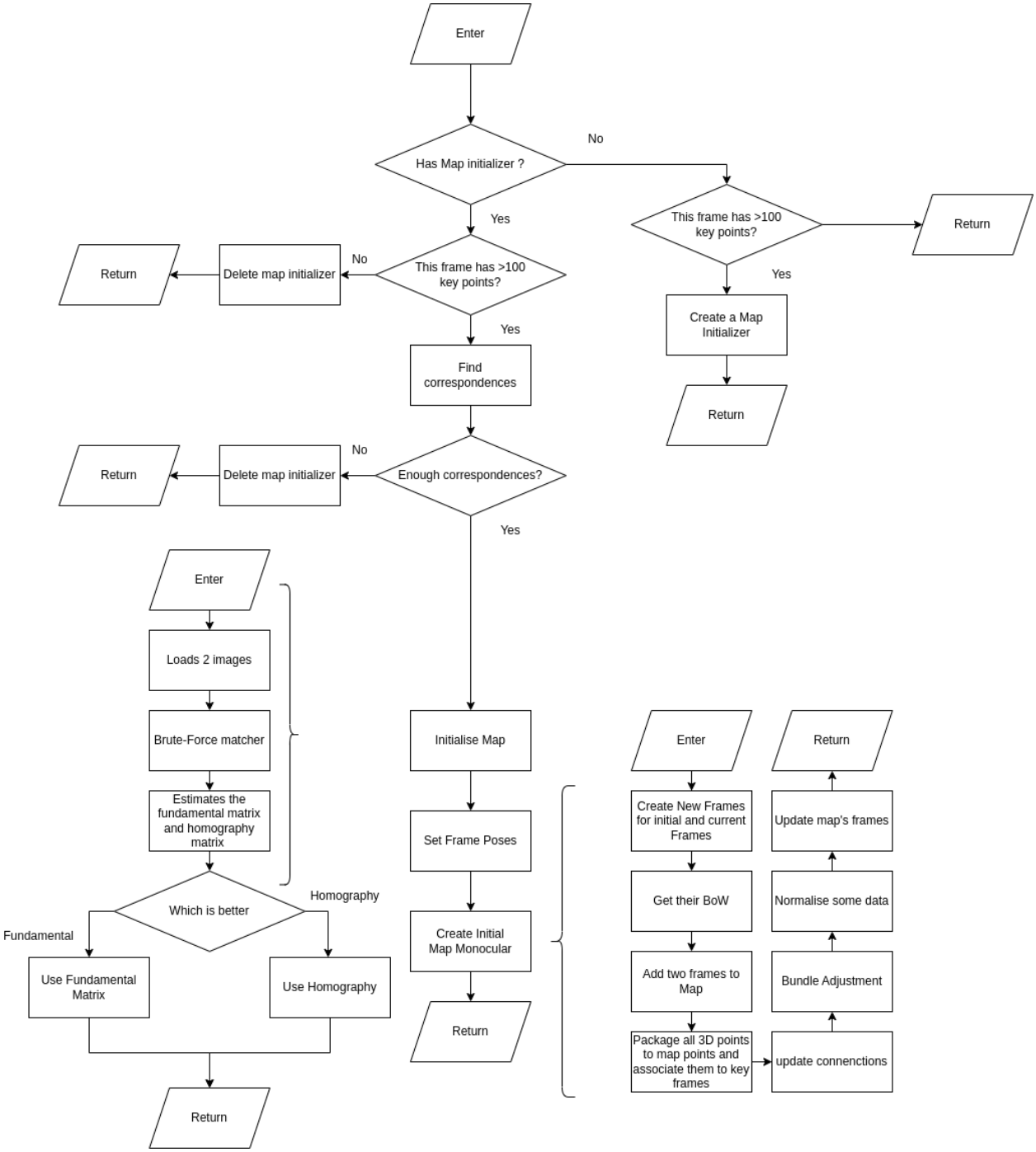

Figure 3: Flowchart of monocular initialization algorithm

The monocular total initialization algorithm is used to estimate the initial pose and position of the image captured by a single monocular camera. Generally, it detects and matches features in the latest two frames of the newcoming images and estimate pose between two frames as the initial pose of the system.
The overall flowchart of monocular initialization algorithm is shown in Figure 3. Most of the codes are doing supportive work and the most critical part of the function is “initialize map”. In specific, the subfunction loads two images from the camera and extracts features using the ORB detector and descriptor. It then matches the features between the two images using the Brute-Force matcher and estimates the fundamental matrix and homography matrix using the RANSAC algorithm. The program then triangulates 3D points using the estimated fundamental matrix and homography matrix and estimates the camera pose using the PnP algorithm. Finally, it checks if the homography pose is better than the fundamental pose and uses the better estimate for the camera pose initialization.
There are two main factors that make monocular camera is significantly harder than using stereo or RGBD cameras. The first one is monocular camera cannot directly capture 3D information which make it difficult to obtain 3D position of the features in the image. The second one is the scale of the image is unknown. Without it, the absolute scale of the scene cannot be determined directly.

2.b Modify ORB-SLAM2 to compute the statistics of how many frames ORB-SLAM2 requires for the tracker to first successfully initialize and populate a map with land-marks for each run. Describe what modifications you made and what information you collected.

The modification produces two text outputs, one from initializer.cpp and the other from tracker.cpp.

```
// Q2b
// This shows an example of how to log data; modify
  stringstream result;
  result << "\n mnID:"<<CurrentFrame.mnId<<"\n"
         << "Current size is"<< CurrentFrame.mvKeys.size()<<"\n"
        <<" mTimeStam: "<< CurrentFrame.mTimeStamp<<"\n"
        << " RH of this frame"<<RH<<"\n"
        << endl;
logger.log(result.str());
```

Figure 4: Modification of Initializer.cpp

```
void Tracking::MonocularInitialization() {
    // ZJA: Q2b
    counter++;
    stringstream result;
    result << "Counter is now " << counter<<"\n"
           << "current size is"<< mCurrentFrame.mvKeys.size()<<"\n"
           << "mnID:"<<mCurrentFrame.mnId<<"\n"
           <<" mTimeStam: "<< mCurrentFrame.mTimeStamp<<"\n"
           << endl;

    logger1.log(result.str());
```

Figure 5: Modification of Tracker.cpp

As shown in Figure 4, I collected mnID, keypoint size, timestamp, and RH in initializer.cpp. Similarly, I collected similar things in Tracker.cpp. The reason for collecting similar things in Tracker.cpp is to see if their values are consistent.

2.c Use your code to examine how the initializer behaves for both *rgbd_dataset_freiburg3_nostructure_texture_near_withloop* and *rgbd_dataset_freiburg1_xyz*. Do you think one dataset could be more challenging than another for initialization? Explain your reasoning.

```
52,Counter is now 53
current size is1994
mnID:52
 mTimeStam: 1.34184e+09

53,Counter is now 54
current size is1991
mnID:53
 mTimeStam: 1.34184e+09

54,Counter is now 55
current size is1987
mnID:54
 mTimeStam: 1.34184e+09

55,Counter is now 56
current size is1994
mnID:55
 mTimeStam: 1.34184e+09
```

a. Dataset 1

```
227,Counter is now 228
current size is2004
mnID:6
 mTimeStam: 1.30503e+09

228,Counter is now 229
current size is2004
mnID:7
 mTimeStam: 1.30503e+09

229,Counter is now 230
current size is2006
mnID:8
 mTimeStam: 1.30503e+09

230,Counter is now 231
current size is2001
mnID:9
 mTimeStam: 1.30503e+09
```

b.Dataset2

Figure 6: Last few lines of output from the Tracker.cpp of two datasets

The first dataset has coherent data that stops at the 56th frame. On the other hand, the second dataset has non-coherent data, and the mnId resets to zero multiple times. Although the key points in each frame appears to be similar in both datasets, the second dataset shows map restarts multiple times due to mnId resets. While the map of the second dataset was quickly initialized and generated after the last restart, overall, the second dataset still runs slower which stops at around the 231st frame.

Furthermore, it was observed that the number of outputs from initializer is less than the number of outputs from tracker. Since the key point size in both files is much greater than 100, it is inferred that in some frames of the second dataset, the number of connections is less than the set value of 100. This may be due to significant camera shake.

In summary, the second dataset poses two challenges, the first being frame loss, and the second being significant camera shake. These are the two reasons that make it more challenging than the first dataset.

Question 3

3.a Where in the code is the factor graph for the tracking front end defined? Draw the structure of the factor graph which is used to estimate the tracker pose. Your description should also identify the C++ classes of any edges and vertices used. It is not necessary to write the equations for the vertices and edges.

The front-end of the ORB-SLAM system is mainly responsible for extracting image features and calculating the camera's pose. In the code, the part of the front-end that applies the factor graph to estimate the pose is defined in the function ***PoseOptimization*** of class ***Optimizer***. The structure of the factor graph has been shown below. In the figure, the vertex x_i represents the pose of camera in the corresponding frame. On the other hand, the m^imeans the position of the landmarks. As for the edges in the figure, the edge between camera poses represents the motion constraint between adjacent frames, while the edge between camera and landmark represents the measured distance between the camera and the landmark in the corresponding frame. More specifically, the class of frame vertex and its related edge is ***g2o::VertexSE3Expmap***, whereas the map point vertex have two categories according to the type of sensor. For the mono, class ***g2o::EdgeSE3ProjectXYZOnlyPose*** were used while for the stereo class g***2o::EdgeStereoSE3ProjectXYZOnlyPose*** was used.

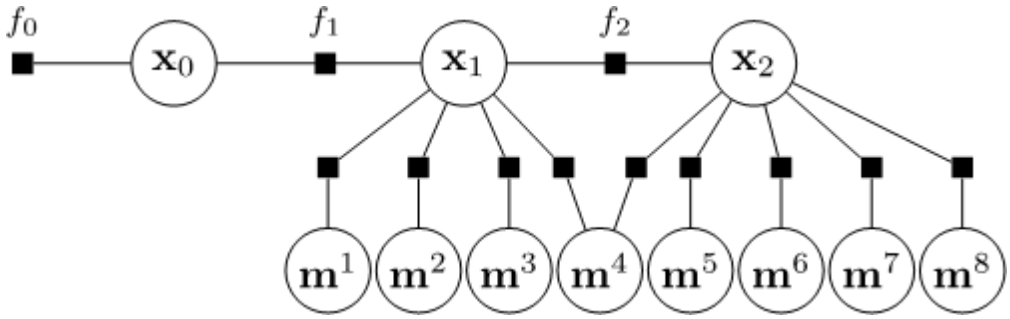

Figure 7: factor graph for pose estimation

3.b One parameter to control is the number of features extracted per frame. ORBSLAM, by default, chooses 2000 features per frame.

Explore how changing this number influences the performance on the KITT/07 run.

In this section we will discuss how changing the number of extracted feature points per frame will affect the performance of the ORB-SLAM system. The following data are obtained experimentally on the KITT/07 dataset. The first and most intuitive thing is the increase of the system running speed. The following table records the median tracking time and mean tracking time of the ORB-SLAM system when the number of extracted points per frame is 1500, 2000, and 2500, respectively. Observing the data in the table, we can see that as the number of extracted feature points per frame increases, it leads to an increase in the amount of operations the system has to perform during tracking, and the time to track and locate increases as a result.

Table 1: tracking time with different feature number

Number of Features	median tracking time(s)	mean tracking time(s)
1500	0.0256377	0.0292371
2000	0.0285062	0.0317123
2500	0.0306112	0.0331462

As for accuracy, as expected, the mean absolute pose error (APE) of the SLAM system output decreases as the number of feature points increases and its sum of squares error decreases simultaneously. The following table and figures record the performance of the ORB-SLAM2 system on the KITT/07 dataset when the number of feature points extracted in each frame is 1500, 2000, and 2500, respectively. Since the system uses Random sample consensus (RANSAC) to estimate the camera pose, the output is different each time with the same parameters. The following table records the data with better performance in several experiments.

Table 2: error data with different feature number

Number of Features	mean	median	rmse	sse	std
1500	4.580128	4.730126	4.755606	24786.9054	1.27993
2000	3.285337	3.194989	3.508539	13491.59177	1.231425
2500	2.800445	2.644904	2.990491	9810.508849	1.049066

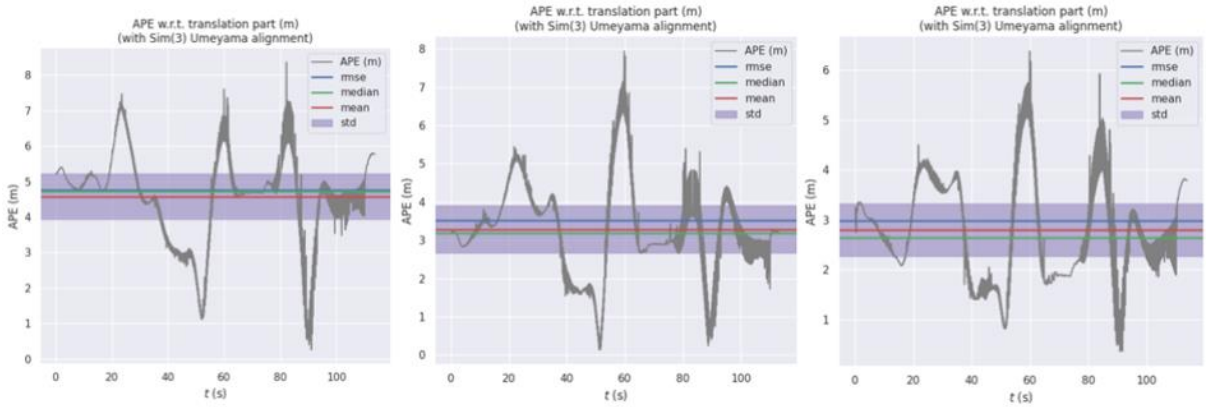

Figure 8: error plot with different feature number

To summarize, when choosing the number of features extracted per frame, we need to consider both efficiency and performance tradeoffs. When there are too few features, the accuracy and stability of the system may be reduced, while when there are too many features, more computation time may reduce the immediacy of the system.

Do you notice any areas which are particularly challenging for ORB-SLAM as you change the number of features? If so, where are they and what do you think makes these areas so challenging?

It is worth mentioning that, observing the figure below, it can be found that during the process of changing the number of features, the straight sections of the route are more affected than the turning sections. The main reason for this phenomenon is that in the straight road section, the feature points tend to be concentrated on both sides of the road. And when the number of feature points is small and the corner points are concentrated, it is easy to have uneven distribution of feature points, which in turn leads to the decrease of the accuracy of system positioning.

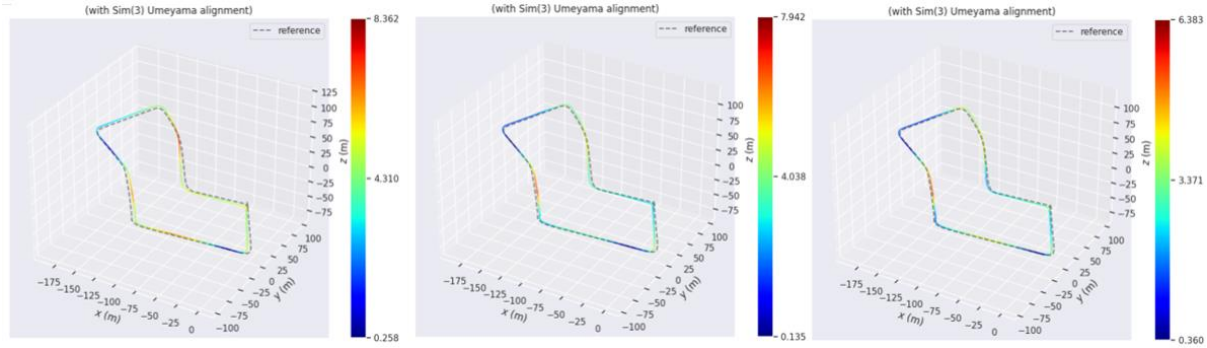

Figure 9: trajectory with different feature number(1500, 2000, 2500)

As an example, the following two figures show the distribution of map points selected by the system for each frame with 1500 and 2000 feature points in close locations. By comparison, when the parameter is 1500, the map points are concentrated on the buildings on both sides of the road, while when the parameter is 2000, the map points are more evenly distributed in this frame, which means that more comprehensive information is provided and the accuracy of localization is improved.

Figure 10: Frame with 1500 features per frame

Figure 11: Frame with 2000 features per frame

3.c How are both the outlier approaches used in the tracking part of ORB-SLAM, and what purpose do they serve in each?

The OBR-SLAM system uses the PnP RANSAC method for solving camera poses from images. RANSAC helps the PnP model to filter the observed data and exclude outliers to improve the accuracy and robustness of the model. Based on this, reprojection error is introduced to further strengthen the outlier rejection. After each iteration of RANSAC, inliers will be filtered twice, and data points with reprojection error exceeding the threshold will be classified as outliers.

3.d Modify the ORB-SLAM2 code so that you can disable the reprojection-based outlier rejection system.

In the code, we can disable the reprojection-based outlier rejection system by blocking the statement that compares the reprejection error to the corresponding threshold. Specifically, change **if (error2 < mvMaxError[i])** to **if (true)** in the function ***CheckInlier*** in class ***PnPsolver***. Next, we will conduct experiments on three datasets KITTI/07(K/7), TUM/rgbd_dataset_freiburg3_long_office_household (T/office), TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop (T/texture) to analyze the impact of disabling outliers.

K/7

Table 3: error number comparison with outlier rejection system on K/7

K/7	mean	median	rmse	sse	std
able outliers	3.149265	3.039445	3.335849	12196.1645	1.100007
disable outliers	4.254359	4.295284	4.438422	21590.7464	1.264918

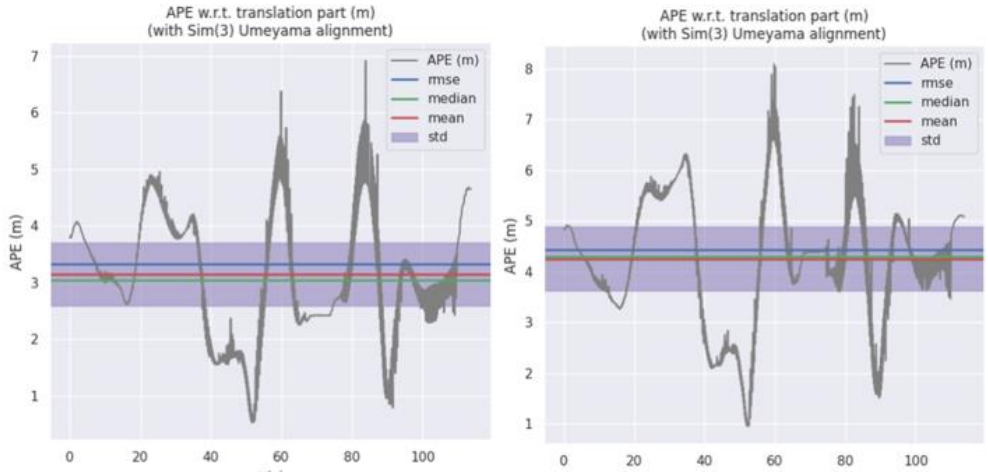

Figure 12: error plot with outlier rejection system abele and disable on K/7

Looking at the data in the table above, we can see that disabling reprojection-based outlier rejection clearly results in a degradation of system performance. The mean error increases from 3.15 to 4.25 and std increases from 1.10 to 1.26 for the same other parameters. The main reason for this phenomenon is that more feature points containing noise are used to estimate the model parameters, resulting in a decrease in model accuracy and robustness. The comparison of the trajectories in the figure below highlights this point even more.

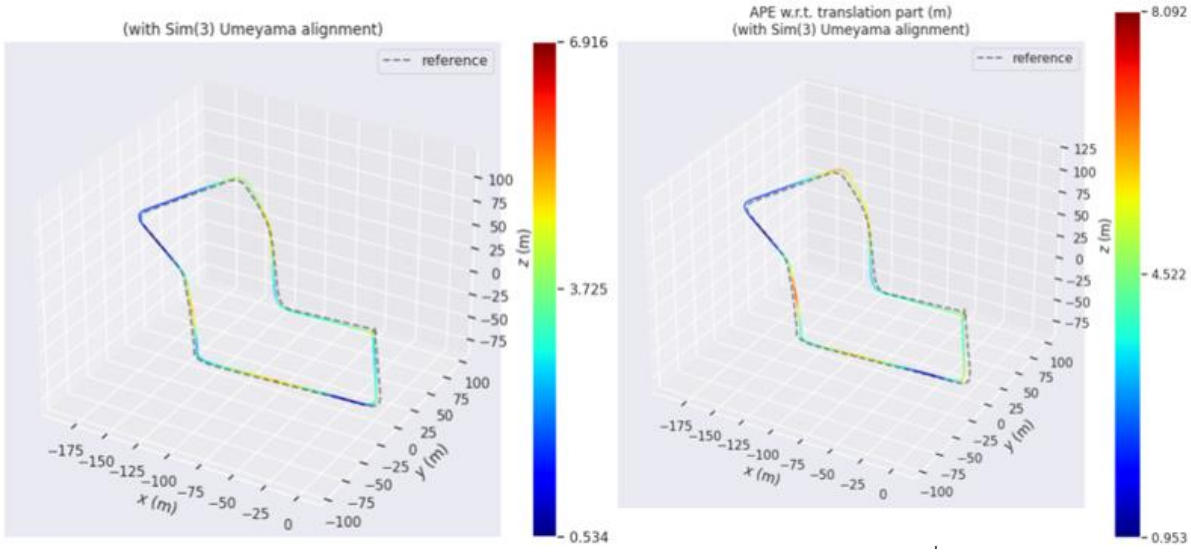

Figure 13: trajectory with outlier rejection system abele and disable on K/7

T/office

Table 4: error number comparison with outlier rejection system on T/office

T/office	mean	median	rmse	sse	std
able outliers	0.010967	0.010015	0.012507	0.399374	0.006014
disable outliers	0.038032	0.031766	0.043073	4.74575	0.020219

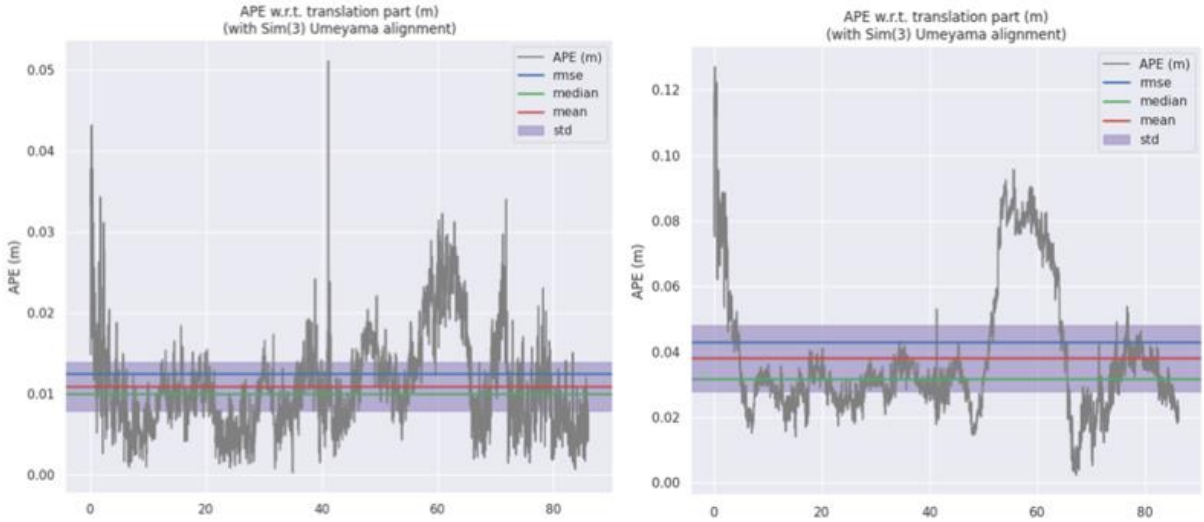

Figure 14: error plot with outlier rejection system abele and disable on T/office

The experiments obtained on the T/office dataset are similar to those obtained on K/7, and both show that outliers rejection can improve the performance of the ORB-SLAM system. This improvement is particularly evident on the T/office dataset. Comparing the experimental data before and after disabling, its average error is reduced by 71%, which is a significant performance improvement.

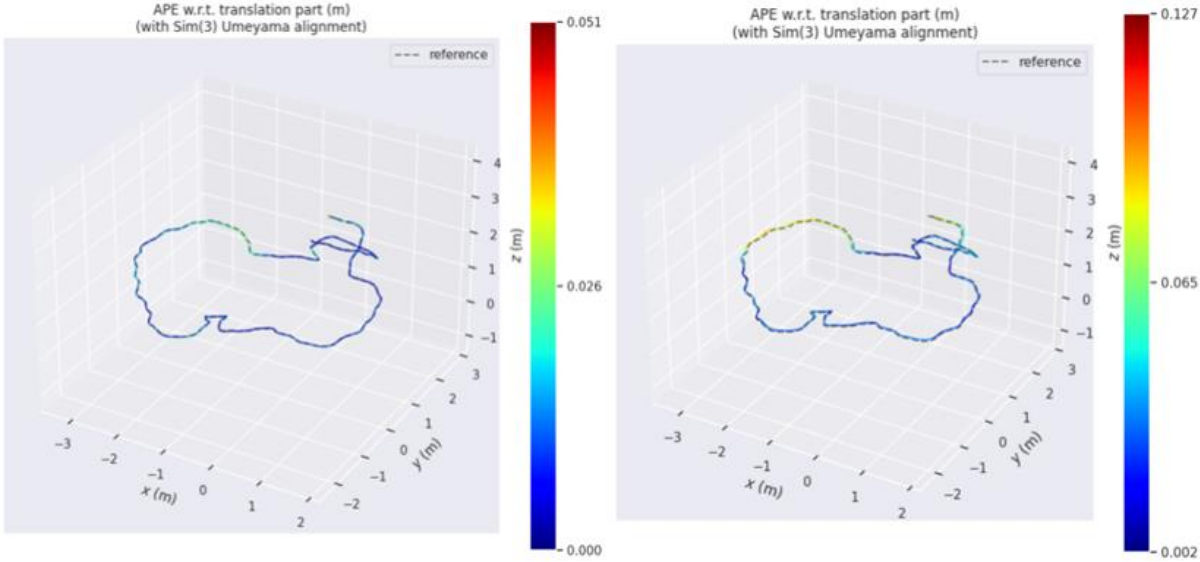

Figure 15: trajectory with outlier rejection system abele and disable on T/office

T/texture

Table 5: error number comparison with outlier rejection system on T/texture

T/texture	mean	median	rmse	sse	std
able outliers	0.012596	0.011597	0.013965	0.31769	0.00603
disable outliers	0.011889	0.010967	0.013353	0.29008	0.006078

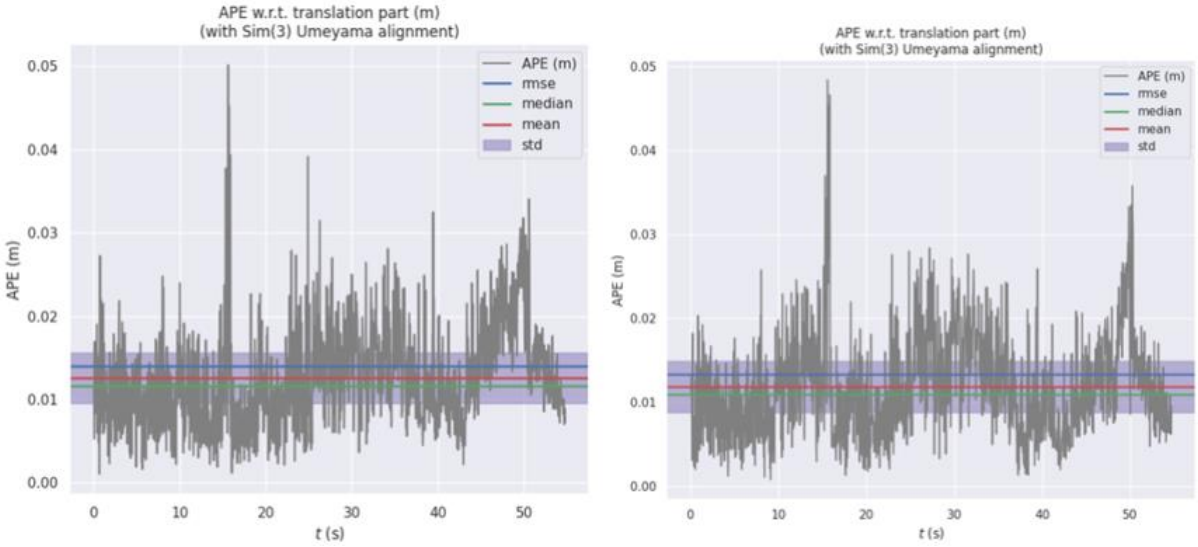

Figure 16: error plot with outlier rejection system abele and disable on T/texture

On the T/texture dataset, similar results to the first two sets of experiments were not obtained. On the contrary, the mean error even decreased from 0.0126 to 0.0119 after disabling outliers rejection, but overall both performed similarly. There are several possible reasons for such experimental results. First of all, this dataset has a simple scenario and was able to make accurate predictions before outliers rejection was enabled. Therefore, the room for improvement is limited after enabling. Second, the parameters of the reprojection-based outlier rejection system might not appropriate in the T/texture scenario, and it is possible that some of the available feature points are classified as outliers, resulting in less information available. This explains why there is a rise in error after outliers rejection is enabled.

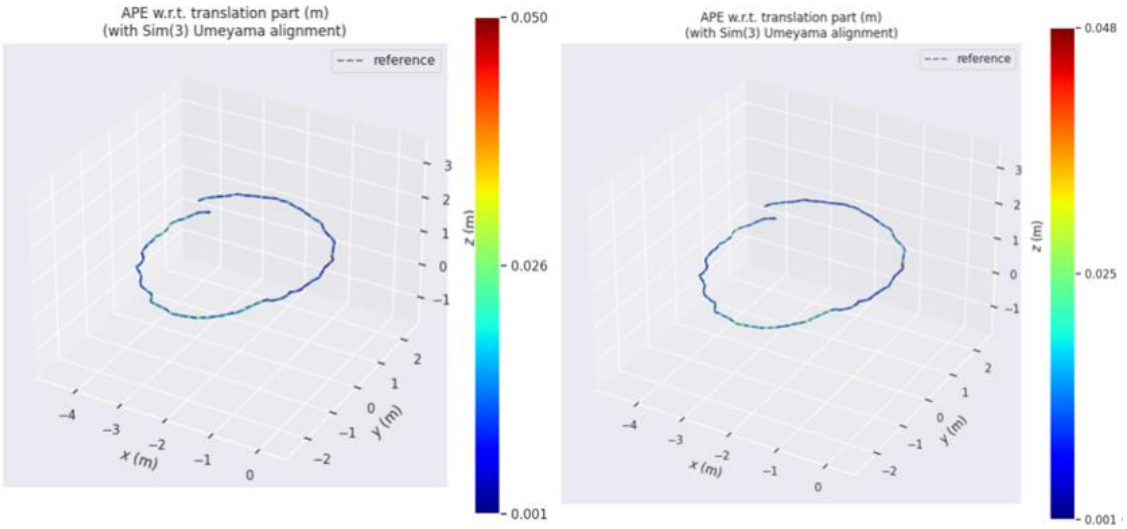

Figure 17: trajectory with outlier rejection system abele and disable on T/texture

Question 4

4.a What are Sim3 and SE3? Why does ORB-SLAM use Sim3 to close loops for monocular SLAM, but uses SE3 for stereo and RGB?

Sim3 and SE3 are distinct geometric transformations widely employed in robotics and computer vision fields, particularly within the context of SLAM (Simultaneous Localization and Mapping) systems. SE3, or the Special Euclidean Group, represents rigid body transformations in 3D space, possessing 6 degrees of freedom which consist of 3 for translation (x, y, z) and 3 for rotation (roll, pitch, yaw). In SLAM applications, SE3 transformations are typically harnessed to express motion between frames or the pose of a camera within a coordinate system. Importantly, SE3 transformations preserve relative distances and angles between points, and can be represented as a 4x4 homogeneous transformation matrix. In contrast, Sim3, or the Similarity Transformation Group, encompasses similarity transformations that incorporate SE3 transformations along with an additional scale factor, resulting in a total of 7 degrees of freedom, including 3 for translation, 3 for rotation, and 1 for scale.

In monocular SLAM, depth data is not directly available from the sensor, causing an indeterminate scale factor for the reconstructed map. Upon detecting a loop closure, the system must compensate for scale drift in addition to translation and rotation errors. To address this challenge, Sim3 transformations are employed, as they encapsulate the scale factor and facilitate the correction of scale drift during loop closure events.

Conversely, stereo and RGB-D cameras supply direct depth information, thereby enabling map reconstruction at a known scale. As a result, when a loop closure is detected, there is no scale drift to rectify; the system only needs to account for translation and rotation errors. SE3 transformations are thus suitable in this context, as they address the 6 degrees of freedom required for translation and rotation corrections.

In summary, ORB-SLAM adopts Sim3 transformations for loop closure in monocular SLAM to accommodate the indeterminate scale factor, whereas SE3 transformations are employed for stereo and RGB-D SLAM, where depth information is directly obtainable and the scale is known.

4.b Where in the code is the loop closer implemented? Why is visual appearance rather than geometry used to detect loops?

Where in the code is the loop closer implemented?
In file LoopClosing.cc the loop closer related functions are defined.

What are the main steps which are applied?
The main steps involved in the loop closure system are:
a) Loop candidate detection: When a new frame is added to the map, the system checks whether the current frame has enough matching features with any previous keyframes in the map. If enough matches are found, the system identifies the candidate keyframes that form the loop.
b) Loop verification: To verify whether the loop candidate is valid, the system performs geometric verification by computing the fundamental matrix or essential matrix between the loop candidate and the current frame. The system also checks whether the candidate keyframe is consistent with the current frame in terms of visual words and BoW representation.
c) Similarity transformation estimation: If the loop candidate is verified, the system calculates the similarity transformation (Sim3) matrix between the current frame and the loop candidate keyframe. The Sim3 matrix includes rotation, translation, and scale factors.
d) Scale correction: The system extracts the scale factor from the Sim3 transformation and corrects the scale drift by adjusting the map points and camera trajectory accordingly.
e) Global optimization: The system performs global optimization, such as graph optimization or pose graph optimization, to refine the map and camera trajectory based on the corrected scale factor.

Why is visual appearance rather than geometry used to detect loops?
The loop closure system relies on visual appearance rather than geometry to detect loops because visual appearance is more robust to changes in lighting, texture, and viewpoint, which are common in real-world scenes. In contrast, geometric features may be affected by occlusions, noise, and incorrect feature matching, which can lead to false positives or false negatives in loop detection.

4.c For each run, identify when the loop closure event occurs, and record the scale change which must happen.

For each run, identify when the loop closure event occurs,
Here we are investigating the loop closure of three datasets

The loop closure event occurs when the camera returns to the scenes that they have passed through. In our experiment, the keyframes that they are detected a loop closure are the 505th frame, 492th frame and 382th frame respectively, corresponding to the “07”, “long_office_household” and “nostructure_texture_nearwithloop” datasets.

record the scale change which must happen.

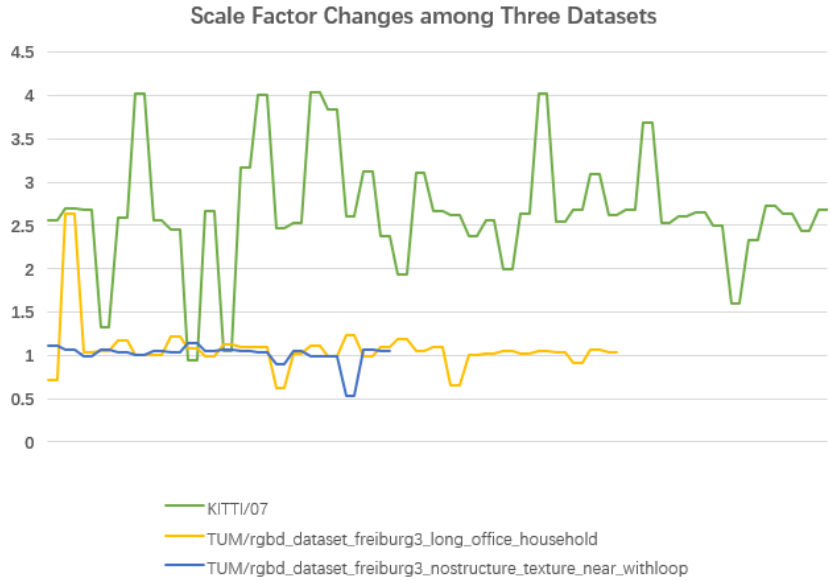

Figure 18: scale factor change of three datasets

The scale factor changes of three datasets are shown above.

In the chart we can find the kitti_07 dataset exhibits the largest fluctuation in its curve and deviates the most from the preset value of 1.2, indicating the greatest scale factor variation and the largest discrepancy from the preset value. The TUM/rgbd_dataset_freiburg3_long_office_household dataset comes in second in terms of these characteristics. The smallest variation and the least deviation from the preset value are found in the TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop dataset. This dataset is the closest to the preset value and displays the least amount of change.

What factors do you think influence the size of the scale factor?

When tracking features between two frames, ORB-SLAM2 estimates the scale factor by triangulating the matched feature points in 3D space. The scale factor is then computed as the ratio of the distances between the camera and the 3D point in the current and previous frames.

Several factors may influence the size of the scale factor during loop closure:

Firstly, the scale factor is directly related to the distance between the camera and the scene. As the camera moves closer to or farther away from the scene, the scale factor changes accordingly. Also, if the scene changes, such as the introduction or removal of objects, the scale factor can be influenced. This is because the depth of the 3D points in the scene relative to the camera can change.

However, the most significant change could happen when there are changes in the camera's intrinsic parameters, such as focal length or principal point. Other peripheral factors could be the changes in lighting conditions, as these can affect the accuracy of the feature detection and matching process.

4.d Propose a way to disable the loop closure system.

We add a code “return false” at the function **bool LoopClosing::DetectLoop()** at line 96 to disable the loop closure system as below figure shown.

```
94  bool LoopClosing::DetectLoop() {
95    {
96          return false;
97      unique_lock<mutex> lock(mMutexLoopQueue);
98      mpCurrentKF = mlpLoopKeyFrameQueue.front();
99      mlpLoopKeyFrameQueue.pop_front();
100     // Avoid that a keyframe can be erased while it is being process by this
101     // thread
102     mpCurrentKF->SetNotErase();
103   }
```

Figure 19: disable the loop closure system in code

4.e Present the results of disabling the loop closure on each of the datasets above. Describe the behaviours you observe. How do these relate to the the results you saw in part c.?

Present the results of disabling the loop closure on each of the datasets above.

(1)Results of disabling the loop closure on KITTI/07

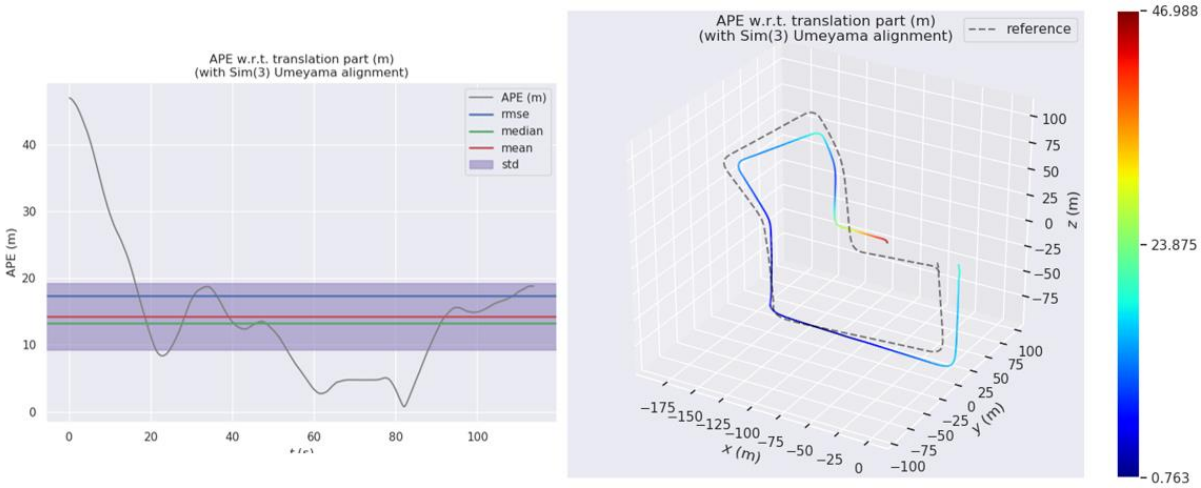

Figure 20: Absolute pose error between the results on KITTI/07 disabling the loop closure system and groundtruth.

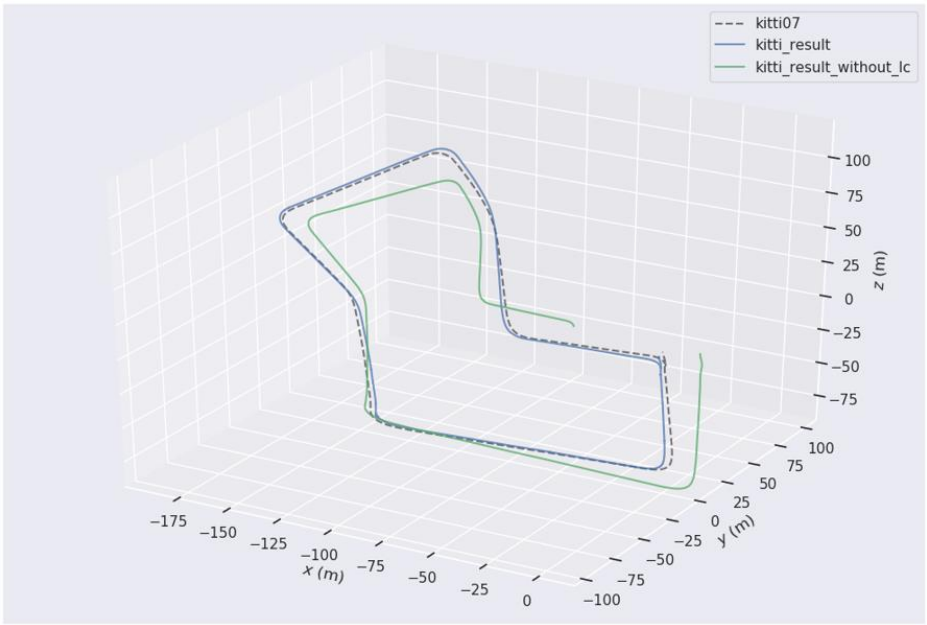

Figure 21: The trajectories of the kitti_07 data results with and without LoopClosing system compared with groundtruth.

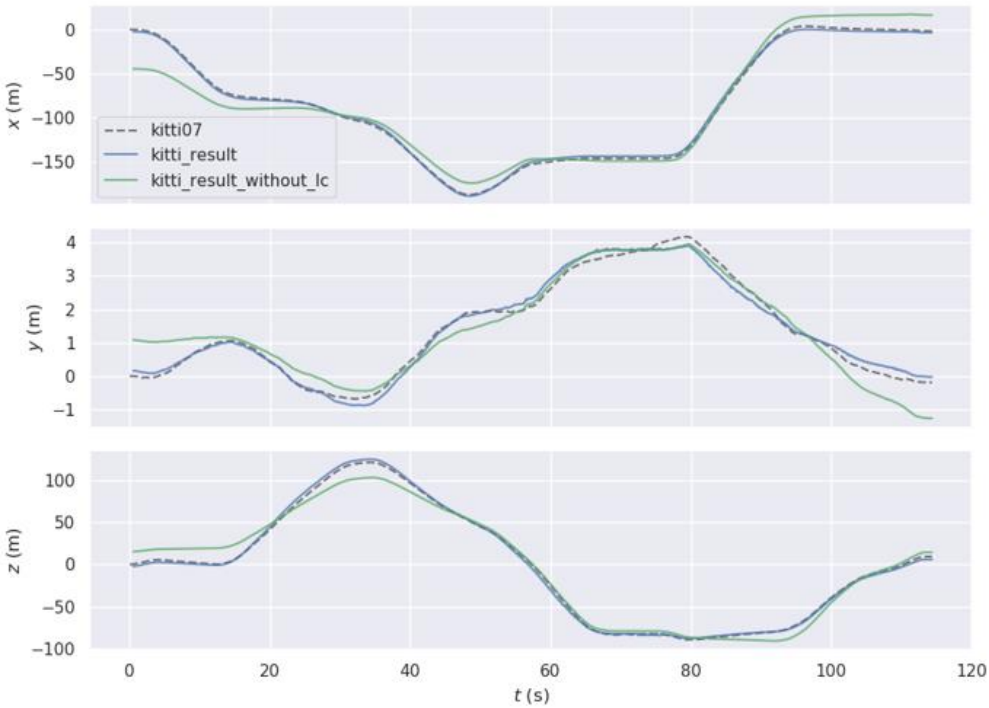

Figure 22: The trajectories of the kitti_07 data results with and without LoopClosing system compared with groundtruth in xyz view.

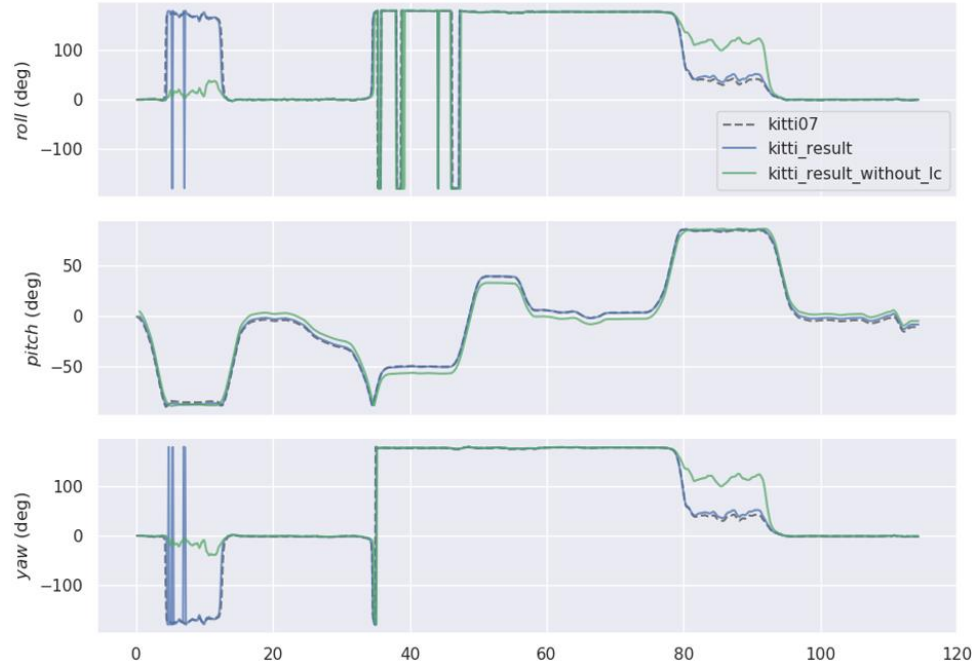

Figure 23: The trajectories of the kitti_07 data results with and without LoopClosing system compared with groundtruth in rpy view.

(2)Results of disabling the loop closure on TUM/rgbd_dataset_freiburg3_long_office_household

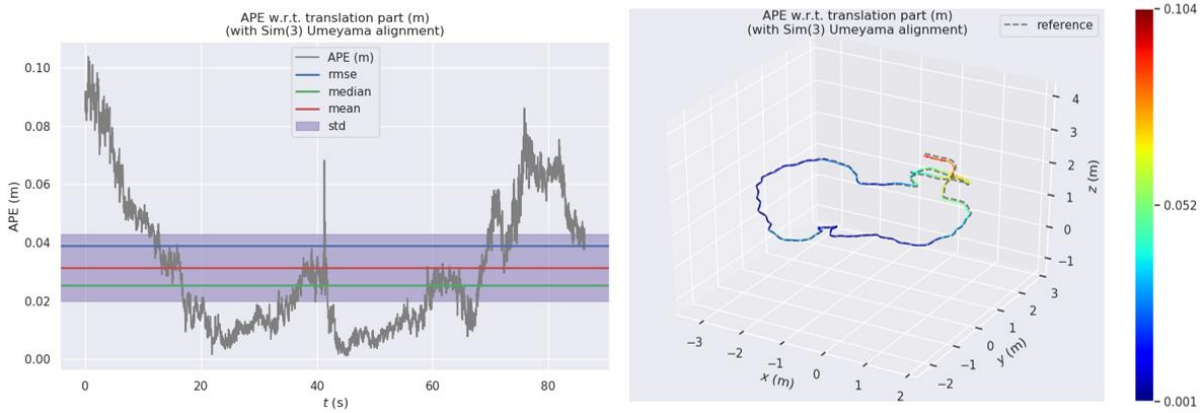

Figure 24: Absolute pose error between the results on TUM/rgbd_dataset_freiburg3_long_office_household disabling the loop closure system and groundtruth.

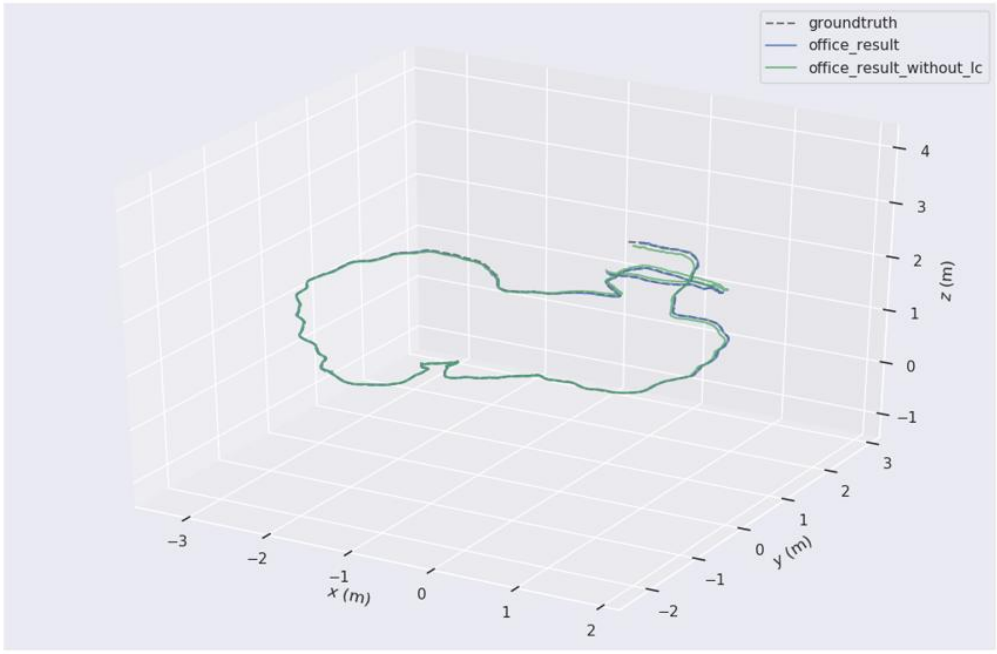

Figure 25: The trajectories of the TUM/rgbd_dataset_freiburg3_long_office_household data results with and without LoopClosing system compared with groundtruth

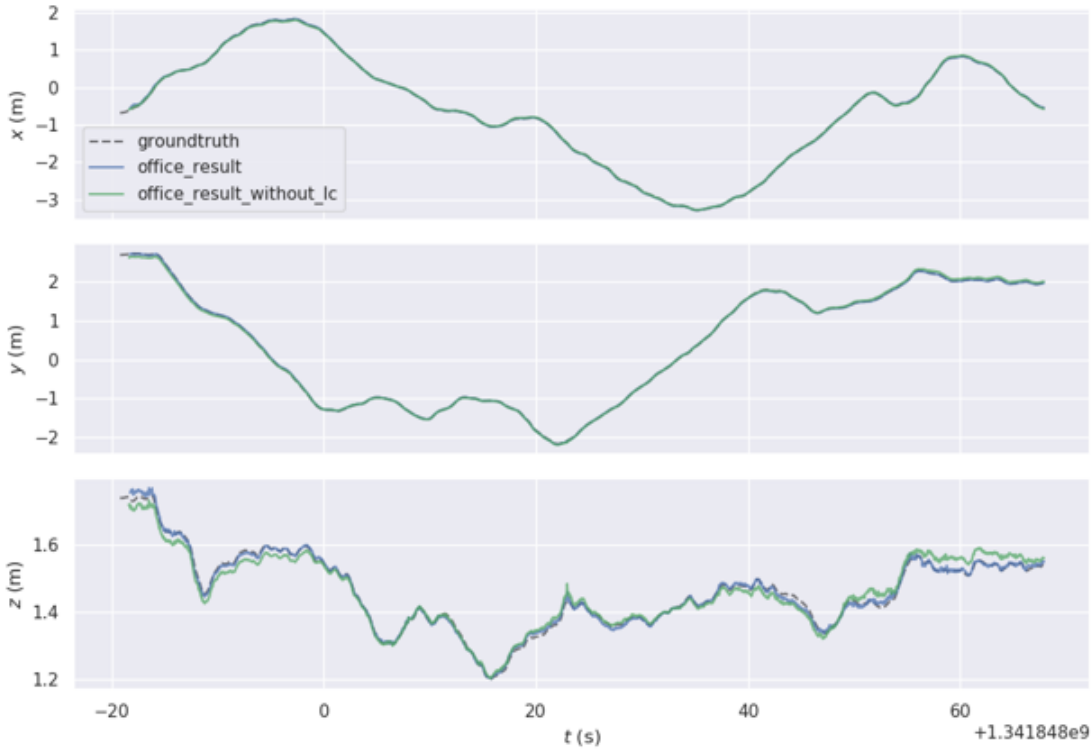

Figure 26: The trajectories of the TUM/rgbd_dataset_freiburg3_long_office_household data results with and without LoopClosing system in xyz view compared with groundtruth

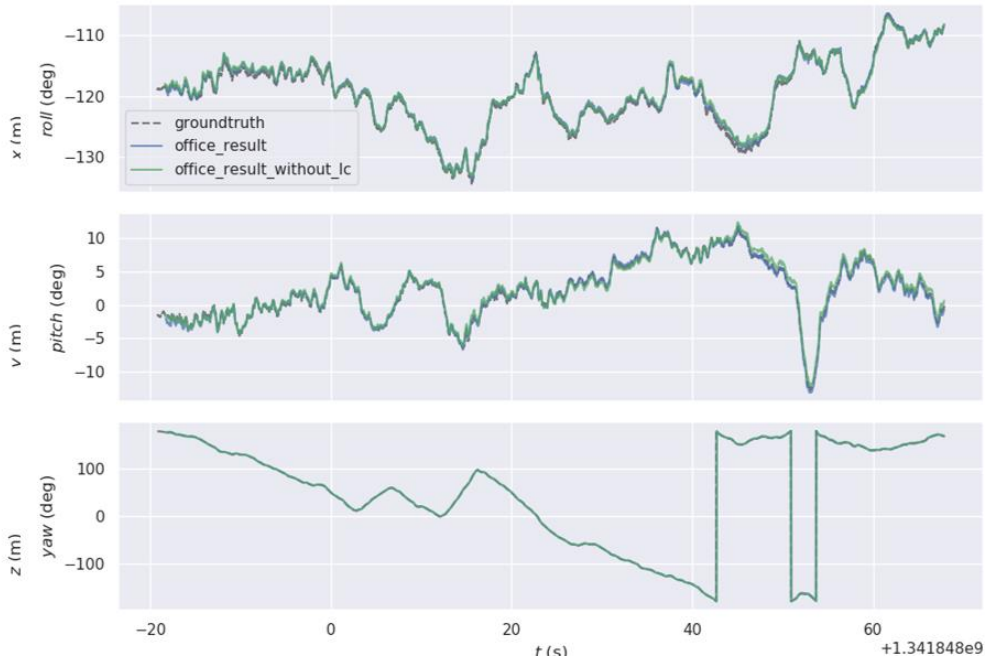

Figure 27: The trajectories of the TUM/rgbd_dataset_freiburg3_long_office_household data results with and without LoopClosing system in rpy view compared with groundtruth

(3)Results of disabling the loop closure on
TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop

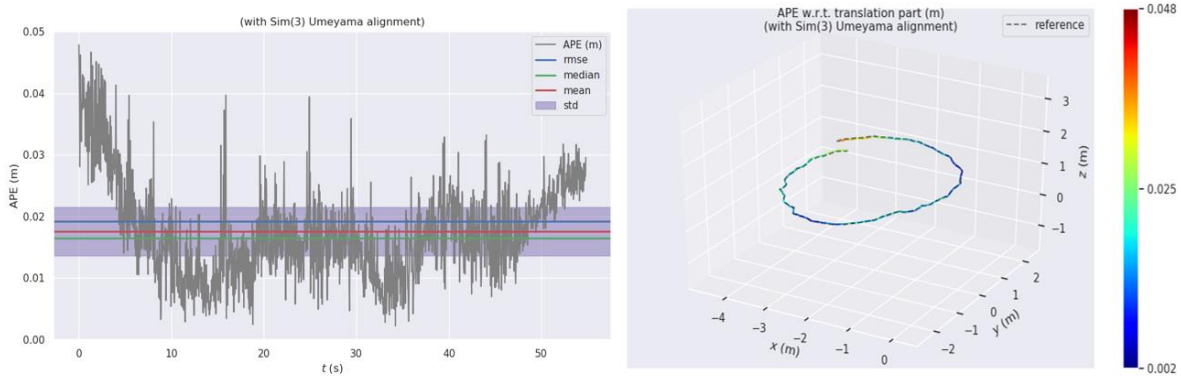

Figure 28: Absolute pose error between the results on TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop disabling the loop closure system and groundtruth

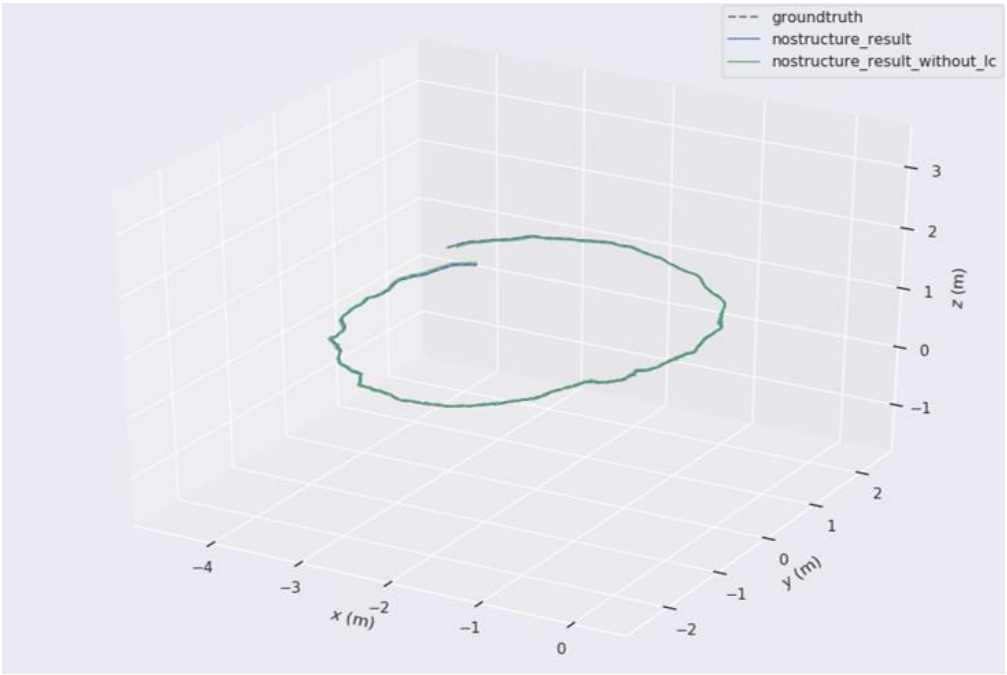

Figure 29: The trajectories of the TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop data results with and without LoopClosing system compared with groundtruth

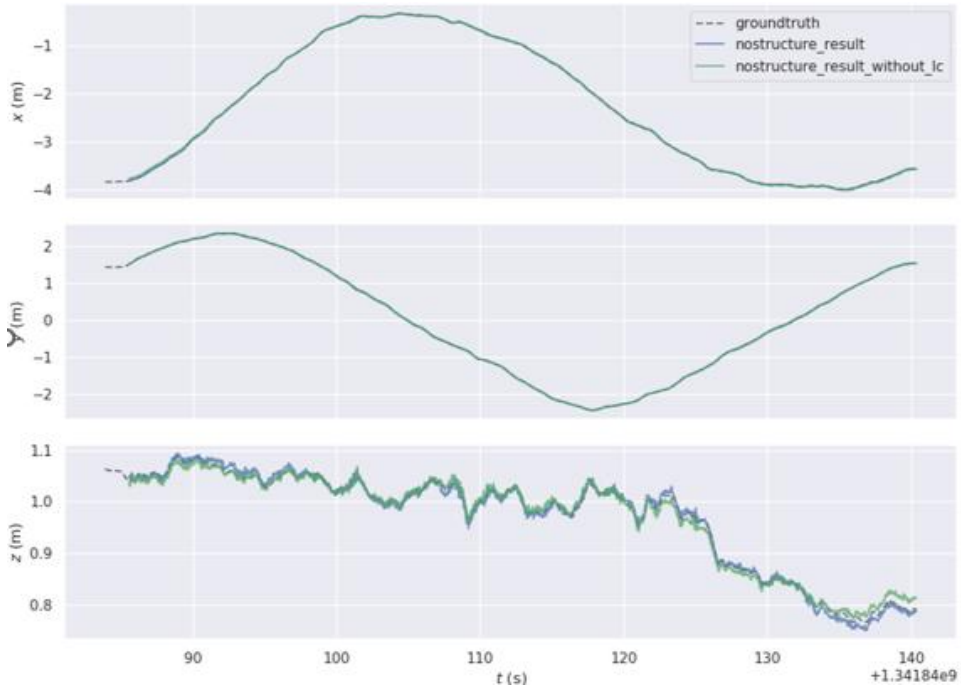

Figure 30: The trajectories of the TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop data results with and without LoopClosing system in xyz view compared with groundtruth

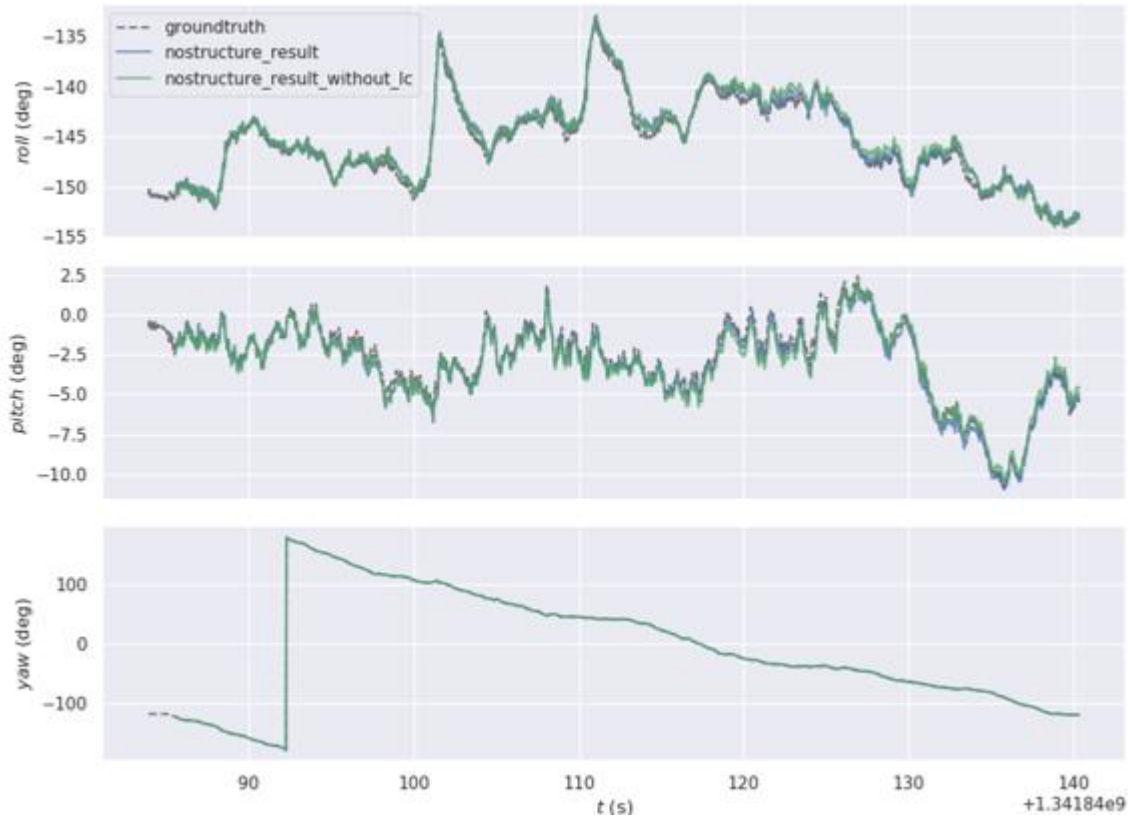

Figure 31: The trajectories of the TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop data results with and without LoopClosing system in rpy view compared with groundtruth

Present the results of disabling the loop closure on each of the datasets above. Describe the behaviours you observe.

We simulated the scenarios with and without loop closure and plotted their results together for comparison. Looking at the individual b and c plots from the three graphs, we can see that the trajectory with loop closure is closer to the true value than the scenario without loop closure. This kind of superiority may not be as apparent in Figures 2 and 3, but in Figure 1, the difference is significant. This can also be observed from the d and e plots of the three graphs.

This occurs because, without loop closure detection, the SLAM system solely relies on feature matching between the previous and current frames to estimate camera motion without correcting for the altered scale factor. In other words the covariance will accumulate by time.

How do these relate to the results you saw in part c.?

In section C, we can find that the dataset "07" has a larger and more fluctuating scale factor, indicating that the distance between the camera and the feature objects is constantly changing, making the observation more challenging. Correspondingly, after we turn off loop closure, the accumulated error of dataset "07" becomes large, which is consistent with what the scale factor implies in section C.

The main reason for this difference is that kitti_07 is set outdoors, unlike the other two datasets which are located in relatively smaller indoor environments. The outdoor setting results in a more complex environment with greater depth changes. This, in turn, leads to an extended computation time and increased computational load. These factors contribute to more noticeable scale factor changes and larger differences from the preset values. Ultimately, when loop closure detection is turned off, this results in more obvious deviations from the ground truth.

For the other two datasets, we can see from Figures 2 and 3 that turning off loop closure does not have a significant impact on their self-localization. This may be because they are located in indoor office environments, where spatial variations are not as large. This observation is consistent with the low scale factors we observed for these datasets in section C.