COMP0130: Robot Vision and Navigation
Coursework 3: Evaluation of ORB-SLAM2

Group 12

May 1, 2023

Question 1

a) Front-end and Back-end

The front-end of ORB-SLAM2 is responsible for processing the incoming images from the camera and extracting features from these images. It uses a feature detection and matching algorithm to find and match ORB (Oriented FAST and Rotated BRIEF) features across frames. Then, it uses these matched keypoints to estimate the camera's pose and the camera's motion.
The back-end of ORB-SLAM2 takes the estimated camera poses and the extracted map features from the front-end and optimizes them to obtain a consistent and accurate map of the environment. It uses loop closure and bundle adjustment techniques to refine the camera poses and the 3D points representing the map features. This optimization step minimizes the reprojection error, which is the difference between the observed image features and their corresponding 3D map points.
They work together in the listed step: The front-end of ORB-SLAM2 receives images from the camera and extracts ORB features from the images, matching them to estimate the camera's pose and motion. It then sends the estimated pose to the back-end. The back-end of ORB-SLAM2 receives the estimated camera poses and the 3D positions of the ORB features from the front-end. And performs and bundle adjustment to optimize the estimated camera poses and the 3D positions of the ORB points, while minimizing the reprojection error.
Then the optimized camera poses and 3D map points are sent back to the front-end, which uses them to detect loop closures and update the estimated camera poses.
The updated camera poses are then sent back to the back-end, and it repeat the bundle adjustment process to refine the map.

b) Keyframes

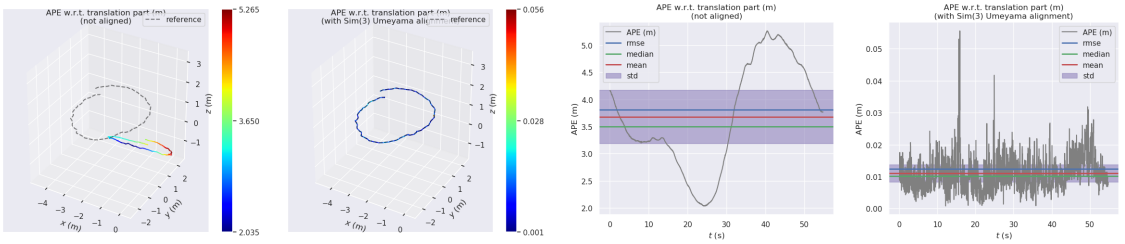
the keyframe-based representation involves the selection of a subset of camera poses and landmarks from the video stream, while discarding intermediate quantities, to construct a sparse map of the environment.
The keyframe-based representation has several advantages over a dense representation:

1. Efficiency: Using a subset of frames as keyframes reduces the computational requirements of the system. This makes the system more efficient and faster.

2. Memory: Storing the entire video stream to construct a map can require a large amount of memory, while a keyframe-based approach reduces memory requirements.

3. Robustness: The use of keyframes provides a robust representation of the environment by allowing the system to recover from tracking failures and maintain consistency in the map.

c) Evaluate Results



(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

Figure 1: Comparison of raw data with and without alignment

The difference is due to if they are Sim(3) aligned. In Figure 1, the left graph shows the absolute pose error without Sim(3) alignment, right is aligned. The graph after alignment has obvious smaller error and it is more related to the algorithm performance. The Sim(3) is a Lie group that combines 3D rigid-body transformations and changes in scale. It is a higher-dimensional group than the more commonly used SE(3) group, which only considers rigid-body transformations without scaling. After sim(3) alignment, the graph can be evaluated intuitively and correctly. The alignment process may fail due to listed case:

1. Insufficient point correspondences can lead to inaccurate transformation estimations, as the algorithm requires a sufficient number of correspondences to accurately compute the transformation.

2. Outliers, or point correspondences that deviate significantly from the majority of the correspondences, can also affect the accuracy of the alignment process.

3. Additionally, the algorithm assumes that the scaling factor between the two sets of points is uniform, and if the scaling factor is non-uniform, the alignment process may fail. Degenerate configurations, such as when all points lie on a plane or in a line, can lead to a situation where the algorithm cannot compute a valid solution.

4. Finally, inconsistent data, or point correspondences containing errors or inconsistencies, can lead to inaccurate estimates of the transformation.

Question 2: Initialize Tracker

a) Monocular Initialization Algorithm

Monocular Initialization Algorithm is mainly realized by the following five steps in [1]. And it is only used when it is certain that the two-view configuration is safe.

1. Find initial correspondences: search for ORB features in the reference frame that match the ORB features extracted at the finest scale in the current frame. If not enough features are found to match, reset the reference frame.

2. Parallel computation of the two models: since ORB SLAM uses two models, one model is evaluated by using the homography matrix and and the other is evaluated by the fundamental matrix. Thus, this algorithm uses the normalized direct linear transformation (DLT) and 8-point algorithms respectively computes the homography matrix and the fundamental matrix in parallel threads and ensures the process homogeneous. During each iteration, the score of each model is calculated and the homography matrix model and the fundamental matrix model with the highest score separately are kept. However, if the models do not have the enough inliers, this algorithm will restart the process again from step 1.

3. Model selection: this algorithm uses $R_H = S_H/(S_H + S_F)$ to evaluate which model should be selected, where S_H is the highest score of homography matrix in step 2 and S_F is the highest score of fundamental matrix in step 2. If $R_H > 0.45$, the homography matrix will be selected. Otherwise, the fundamental matrix will be selected.

4. Motion and structure from motion recovery: if homography matrix is selected in step 2, the algorithm uses DLT to retrieve 8 motion hypotheses. And then directly triangulate the eight solutions and find one solution which is in front of both cameras. And this solution also needs to satisfy most points seen with parallax and low reprojection error. If there is not a clear solution, the process will back to step 1. If fundamental matrix is selected in step 2, convert fundamental matrix in the essential matrix and use the singular value decomposition method to retrieve 4 motion hypotheses. After that, the process is similar to the process of dealing with homography matrix.

5. Bundle adjustment: use full bundle adjustment (BA) to complete the initial process.

This algorithm uses two different models to account for the difference between planar and non-planar scenes. In planar scenes, all points are located on a planar scene, so all points can be described by a homography matrix, which can project a 3D point from the camera coordinate system to another camera coordinate system without changing its position in the 2D image. The problem will not be well constrained if described by a fundamental matrix. In non-planar scenes, all points in 3D space. Thus, it should be described by a fundamental matrix, which can know the transformation relationship between corresponding points in different camera views. If the model is chosen incorrectly, it will lead to inaccurate results. Thus, this algorithm attempt two different models.

Compared with stereo and RGBD cameras that can obtain depth information, monocular cameras can only generate two-dimensional images and there is no depth information. For the initialization algorithm in the case of non-planar, an additional algorithm is required to estimate

the depth information of the image. Therefore, it will be a challenge for monocular camera to use this algorithm.

b) Count Frames

First, I created a global variable “initFrameCounter” to count frames before initialization finished. According to the step 1 in initialization algorithm, the reference frame will be reset when the reference frame and the current frame do not have enough features to match. Thus, in this case one frame will be added. In the implemented code, the following code in ”Tracking.cc” will be executed when insufficient matching points are detected, which means the program will destroy the variable ”mpInitializer” whose type is ”Initializer” when the matching points are insufficient.

```
// Check if there are enough correspondences
if (nmatches < 100) {
  delete mpInitializer;
  mpInitializer = static_cast<Initializer *>(NULL);
  return;
}
```

Thus, “initFrameCounter” need to increment the frame by 1 every time an ”mpInitializer” is regenerated. And we add **code ”initFrameCounter++;” in the ”Initializer::Initializer” function in ”Initializer.cc”**.

Then, there are two situations that will cause the algorithm to reinitialize. From the perspective of the algorithm, when the second step is executed, if no model could be found in this step, the entire algorithm will be reinitialized but will not delete ”mpInitializer” mentioned before. And if the scenes is non-planar scenes, this algorithm will be reinitialized due to the solution cannot satisfy in front of both cameras, most points seen with parallax and low reprojection error. Use the following code to determine whether reinitialization is required.

```
// Try to reconstruct from homography or fundamental depending on the ratio
// (0.40-0.45)
bool success = false;
if (RH > 0.40)
success = ReconstructH(vbMatchesInliersH, H, mK, R21, t21, vP3D,
                  vbTriangulated, 1.0, 50);
else // if(pF_HF>0.6)
success = ReconstructF(vbMatchesInliersF, F, mK, R21, t21, vP3D,
                  vbTriangulated, 1.0, 50);
```

Both cases cause the program to re-execute ”Initializer::Initialize” in ”Initializer.cc”. Thus, when the initialization procedure is executed once, the frame will increase by 1. The code **”initFrameCounter++;” should be added in function ”Initializer::Initialize” in ”Initializer.cc”**. Finally, the number of frames required for the successful initialize of the algorithm is written into the “logger” and saved in its “.txt” file.

In the code, valid matching points in the map are also required when initially creating the map. When the number is not enough, it will be reinitialized, as shown in the following code.

```
if (medianDepth < 0 || pKFcur->TrackedMapPoints(1) < 100) {
   cout << "Wrong initialization, reseting..." << endl;
   Reset();
   return;}
```

The calculation of the median of the depth of the scene is incorrect or the number of effective matching points of the initial keyframe is less than 100, which will cause the algorithm to reinitialize. At this time, the number of frames counted by innovation initialization can also be counted by “initFrameCounter“ in function ”Initializer::Initialize” in ”Initializer.cc”. Therefore, we can get the number of frames required for the first successful initialization and populating a map with land-marks for each run.

c) Examine Initializer Behaviors

The second dataset is harder to initialize than the first dataset. From the results, the second dataset will be re-initialized due to insufficient number of valid matching points when creating the map after the monocular camera is initialized. After the first dataset is successfully initialized, it will not be reinitialized for creating a map. Thus, the second dataset used 211 frames to complete the entire initialization process while the first dataset only used 56 frames to complete. Therefore, he second dataset is more challenging than the first dataset for initialization.

Question 3: tracking optimizer

a) Factor graph

The factor graph for the tracking front end is defined in the **"Optimizer.cc"**. An example of Factor graph that used to estimate the tracker pose is shown in the Figure 2. The factor graph consist of vertices and edges, where f_t represents the current frame, and CP_t represents the estimated camera pose. In the front end, the ORB-SLAM2 execute "PoseOptimisation". The

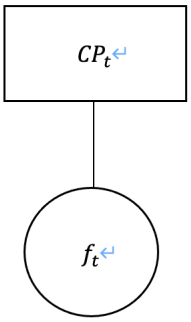


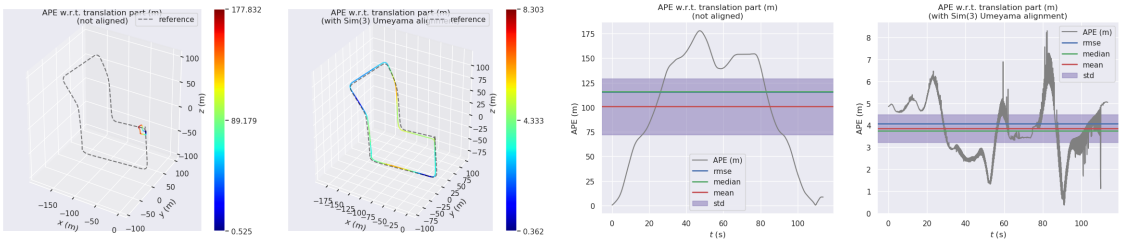
Figure 2: Factor Graph

vertices only donates by keyframes, which uses C++ class "g2o::VertexSE3Expmap".

The "BaseUnaryEdge" type edges for estimating monocular camera pose uses C++ class "g2o::EdgeSE3ProjectXYZOnlyPose", while the edges for estimating stereo camera pose uses C++ class "g2o::EdgeStereoSE3ProjectXYZOnlyPose".

b) Number of features extracted per frame

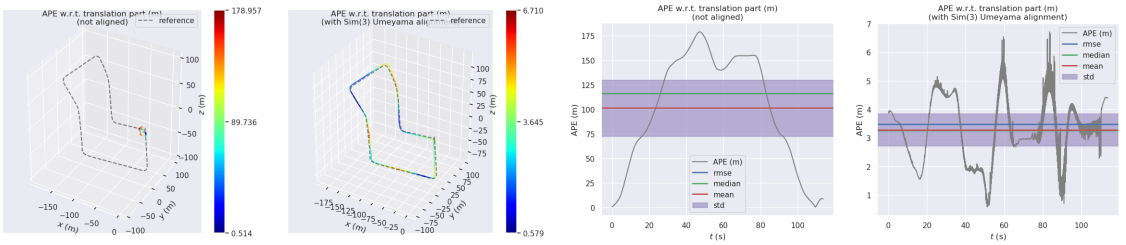
We study the performance on **KITTI/07** with the number of features extratced per frame (NFEPF) of 1300, 1500, 1800, and 2000 (default). When the NFEPF is 1300, the number



(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

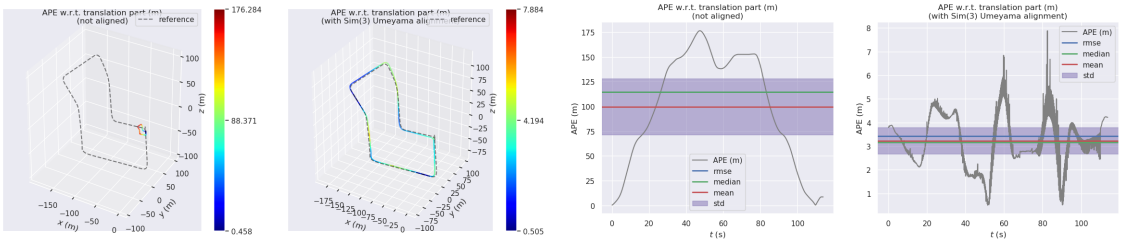
Figure 3: 1300 features extracted per frame on **KITTI/07**

of map points is 113 when the map is created. When the NFEPF is 1500, the number of map points is 130 when the map is created. When the NFEPF is 1800, the number of map points is



(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

Figure 4: 1500 features extracted per frame on **KITTI07**



(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

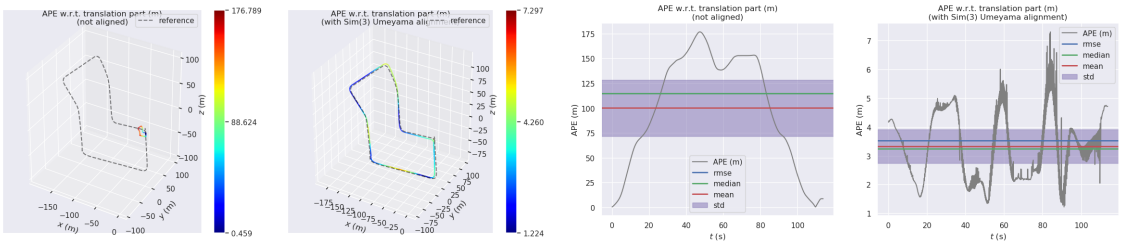
Figure 5: 1800 features extracted per frame on **KITTI07**

144 when the map is created. When the NFEPF is 2000, the number of map points is 166 when the map is created.

It can be observed that when the NFEPF increases from 1300 to 1800, the average absolute pose error decreases by 0.5m, while it remains unchanged when the NFEPF increase to 2000. It indicates that if the NFEPF is larger, and the number of map points in the created map increase, it will lead to more accurate tracking, but this influence will converge when the NFEPF is sufficiently large. The particular challenge is: If the NFEPF is small,

- The position and attitude of the tracking camera are based on feature point matching. If the NFEPF is small, it will lead to tracking loss, which will make the system unable to effectively estimate the position and attitude of the camera.
- The ORB-SLAM2 system needs to locate in the previously established map. If the NFEPF is small, the quality of the map will be degraded, resulting in failure of positioning, which will make the system unable to provide reliable information for subsequent tasks. positioning information.
- it will lead to inaccurate map construction, which may cause the problem of map drift. Map drift refers to the gradual increase of the difference between the map and the actual environment, which will cause the errors of positioning and navigation to accumulate continuously.

Even more, if the NFEPF is extremely small (say 1000), there are insufficient matched features to track the camera pose and position, leading the map unable to be constructed.



(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

Figure 6: 2000 features extracted per frame on **KITTI07**

c) Outlier approaches

- **Reprojection error:** Reprojection error is used in ORB-SLAM2 to detect and reject outliers in the feature matching stage. The reprojection error is the difference between the observed feature location and the estimated location of the corresponding feature in the camera image. If the reprojection error is large, it means that the estimated feature location is not accurate, and the corresponding feature is likely to be an outlier.

  In ORB-SLAM, features with high reprojection errors are rejected, and the remaining features are used for pose estimation. By removing outliers in the feature matching stage, ORB-SLAM improves the accuracy of pose estimation and reduces the number of false-positive matches, leading to a more robust tracking algorithm.

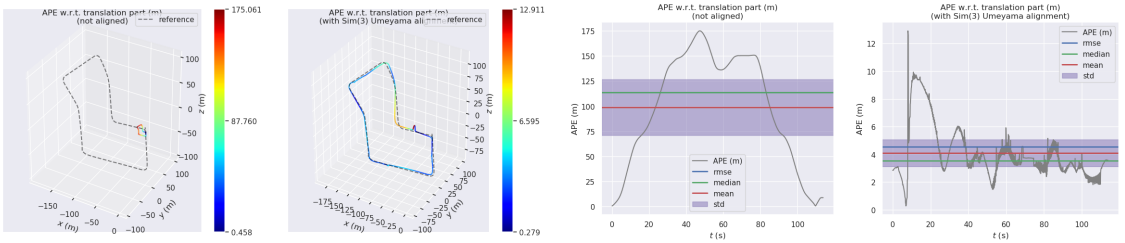
- **RANSAC:** RANSAC is used in ORB-SLAM to reject outliers during the pose estimation stage. RANSAC is an iterative method that fits a model to a set of data, while rejecting outliers that do not conform to the model. In ORB-SLAM, RANSAC is used to estimate the camera pose from the matched features while rejecting outliers that do not correspond to the estimated model.

  During each iteration of RANSAC in ORB-SLAM, a subset of matched features is randomly selected, and the camera pose is estimated using these features. The reprojection error is then computed for all the matched features using the estimated camera pose, and the features with large reprojection errors are considered outliers and rejected. The process is repeated until the camera pose with the least number of outliers is found.

d) Disable the reprojection-based outlier rejection system

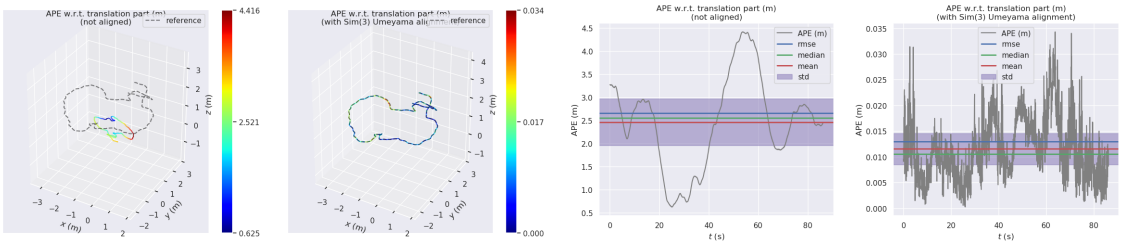
The reprojection-based outlier rejection is performed in the local bundle adjustment algorithm, where it loops through the monocular error edge and applies the Chi-Squared Test (chi2) to compare their reprojection error with a specific threshold. If the error larger than the threshold, the edge would be regarded as outlier and be removed from the graph.
We introduce a boolean variable **"enable_outlier_projection"** and disable the reprojection-based outlier rejection system by set the varibale to **"false"**. In this case, we run ORB-SLAM2 on the datasets **KITTI/07**, **TUM/rgbd_dataset_freiburg3_long_office_household** and **TUM/rgbd_dataset_freiburg3_nostructure_texture_near_withloop**. Here, For convenience,

abbreviate them to **KITTI07**, **TUM1** and **TUM2**. The results are draw in the Figure 7, 8, 9 and 10.



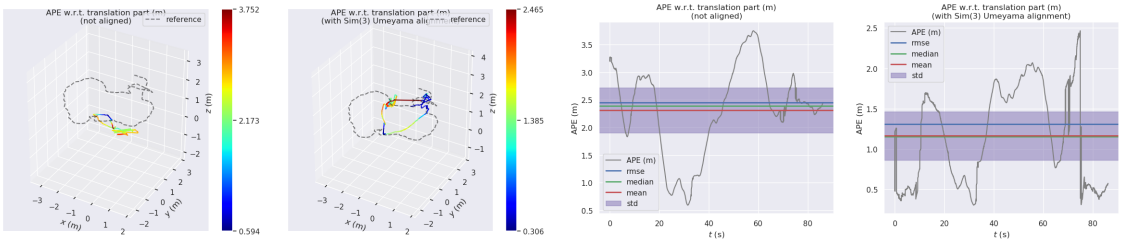
(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

Figure 7: Disable the outliers on **KITTI07**



(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

Figure 8: Enable the outliers on **TUM1**

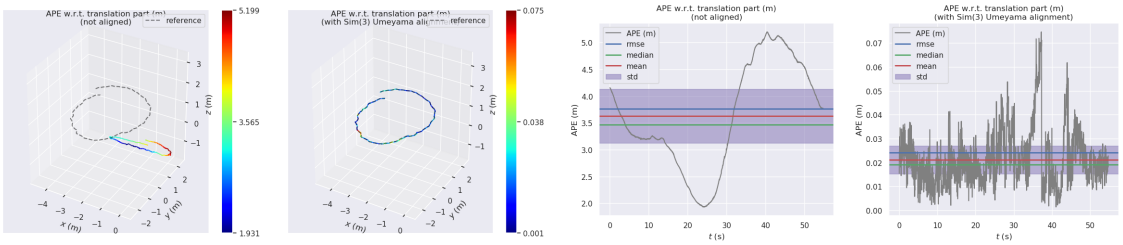


(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

Figure 9: Disable the outliers on **TUM1**

It can be observed that after disabling the reprojection-based outlier rejection system, the absolute pose error increases remarkably, and the estimated camera pose significant deviation from the reference track. The reason of these behaviours occurring might be that disabling the reprojection-based outlier rejection system will

- leads to an increased number of false matches in the feature matching stage. This would result in the inclusion of outliers in the feature set used for pose estimation, leading to inaccurate camera pose estimates.



(a) Absolute Pose Error with no alignment (b) Absolute Pose Error with Sim(3) alignment (c) Absolute Pose Error with no alignment (d) Absolute Pose Error with Sim(3) alignment

Figure 10: Disable the outliers on **TUM2**

- also leads to an increased number of false-positive matches, which could result in tracking failures. This could happen because the algorithm might attempt to track incorrect or spurious features that are not present in the current view.

Question 4: Loop Closing

a) Sim3 vs SE3

SE3 is Euclidean transformation, which represents translations and rotations in 3D space. Therefore, it has three translational degrees of freedom and three rotational degrees of freedom. The Euclidean transformation maintains the length and angle of the vector, which is equivalent to moving or rotating a rigid body without changing its own appearance.

Sim3 is similarity transformation, which commonly used in scenes involving scale changes due to it changes the scale of the original object. Thus, it has seven degrees of freedom, it can accomplish scaling compared to SE3.

The reason of ORB-SLAM using Sim3 for monocular SLAM is that this case the algorithm will use Monocular scale shift to find the deep information of scene. Due to stereo and RGB can directly obtain the deep information, these can set the parameter $s = 1$, where s is the scale in Sim3. And if $s = 1$, Sim3 has the same effect as SE3. Thus, ORB-SLAM use Sim3 to close loops for monocular SLAM, but uses SE3 for stereo and RGB.

b) Code of Loop Closer

The code of loop closer is implemented in "LoopClosing.cc". And in this file, it contains "LoopClosing::DetectLoop()", which is the first step of the loop closer called loop candidates detection. And the second step, "LoopClosing::ComputeSim3()" is corresponding to the compute the similarity transformation. Then, "LoopClosing::CorrectLoop()" is the implementation of loop fusion. And finally, "Optimizer::OptimizeEssentialGraph()" is the last main step called essential graph optimization. Geometry will be affected by environmental factors such as light and occlusion on the detection of geometric features. However, visual appearance is more robust. This is because visual appearance usually do not change due to slight viewing angle changes or noise. Thus, visual appearance is used to detect loops.

c) Loop Closure Event

For each run, the candidate frame will be selected by "LoopClosing::DetectLoop()". However, this process cannot decide the closed loop frame. Then, the algorithm will solve for each candidate frame in "LoopClosing::ComputeSim3()". In this function, it uses RANSAC framework to calculate a Sim3 initial value, and then reproject the map points of the candidate key frame to the current frame to get more matches by the Sim3 initial value. Finally, optimize Sim3 until a frame passes the test, then a closed-loop frame is selected. Thus, the loop closure event occurs when this frame is detected. And the corresponding scale change will stored in the variable "mg2oScw". And the test results are shown below.

- First dataset: the scale change is 2.530401.
- Second dataset: the scale change is 1.039466.
- Third dataset: the scale change is 1.048500.

And we modify the code in "LoopClosing::Run()" to show the loop closure event occurs and the scale and the code is shown below.

```
// Check if there are keyframes in the queue
if (CheckNewKeyFrames()) {
  // Detect loop candidates and check covisibility consistency
  if (DetectLoop()) {
    // Compute similarity transformation [sR|t]
    // In the stereo/RGBD case s=1
    if (ComputeSim3()) {
      // q4c. record the scale change which must happen
      // q4c. identify when the loop closure event occurs
      stringstream scaleRecord;
      scaleRecord << "The loop closure event occurs. \r Scale of loop
          closure: " << std::to_string(mg2oScw.scale()) << "\r";
      std::cout << scaleRecord.str() << endl;

      // Perform loop fusion and pose graph optimization
      CorrectLoop();
    }
  }
```

The size of the scale factor will be affected by the distance between two keyframes. Since the scale factor is calculated using the distance the camera travels between the keyframes participating in the loop. Thus, the distance will change the size of the scale factor. The complexity of the scene may also affect the scale factor. In textured or detailed scenes, it may be easier to recover the correct scale, whereas in featureless or uniform environments, scale may be more difficult to estimate accurately.

d) Disable Loop Closure

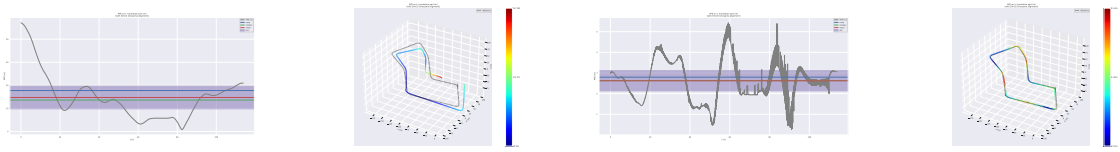
In order to disable the loop closure system, we created a variable named ”loopClosure” and its type is ”bool”. If disable the loop closure system, it can be set to ”false” in the ”LoopClosing.h”. And it is added in ”LoopClosing::Run()” in ”LoopClosing.cc” as shown below.

```
if (CheckNewKeyFrames()) {
     if (loopClosure)
     {
       // Detect loop candidates and check covisibility consistency
       if (DetectLoop()) {
         // Compute similarity transformation [sR|t]
         // In the stereo/RGBD case s=1
         if (ComputeSim3()) {
           // q4c. record the scale change which must happen
           // q4c. identify when the loop closure event occurs
           stringstream scaleRecord;
           scaleRecord << "The loop closure event occurs. \r Scale of loop
               closure: " << std::to_string(mg2oScw.scale()) << "\r";
           std::cout << scaleRecord.str() << endl;

           // Perform loop fusion and pose graph optimization
           CorrectLoop();
         }
       }
     }
   }
```

e) Evaluate Results

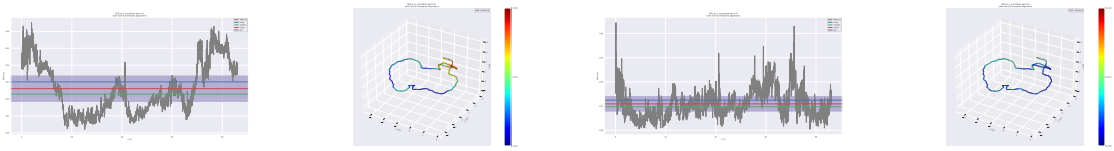
For the first dataset, the results are shown in Figure 11. It shows there is a problem of discontinuity in the trajectory. And it deviates greatly from the actual trajectory after disabling loop closure.



(a) APE with disable loop closure (b) Map with disable loop closure (c) APE with enable loop closure (d) Map with enable loop closure

Figure 11: First Dataset: Disable vs Enable

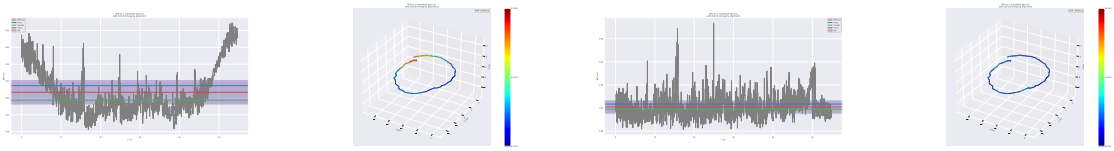
For the second dataset, the results are shown in Figure 12. Obviously, this dataset has less error compare with the first dataset when disable the loop closure. However, compare to the case of enabling the loop closure, its trajectory deviates more.



(a) APE with disable loop closure (b) Map with disable loop closure (c) APE with enable loop closure (d) Map with enable loop closure

Figure 12: Second Dataset: Disable vs Enable

For the third dataset, the results are shown in Figure 13. It is easy to find it is similar to the case of the second dataset. Compare to the enable the loop closure, it has a deviation in the trajectory. But its error is much less than the first dataset.



(a) APE with disable loop closure (b) Map with disable loop closure (c) APE with enable loop closure (d) Map with enable loop closure

Figure 13: Third Dataset: Disable vs Enable

Consider the results in Question 4.c, we can find the effect of disable the loop closure is increasing with the larger the scale factor. This is because the function of the loop closure system is to automatically establish a connection between the two locations when the system detects that the current camera location is very similar to the location that has been visited before. When the loop closure system is disabled, the system cannot automatically detect loop closure and cannot connect the frames. Thus, the oop closure system plays a very important role in reducing the error and improving the accuracy of ORB SLAM.

References

[1] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

END OF COURSEWORK