COMP0130 Coursework 3: Evaluation of ORB-SLAM system

1 Introduction

This coursework evaluates the performance of the monocular ORB-SLAM system under four different conditions: under default configuration settings, with a reduced number of ORB features used by the system, without outlier rejection, and without loop closure. System performance is evaluated in two real-world environments, using the KITTI07 and KITTI10 datasets.

2 Structure of Monocular ORB-SLAM

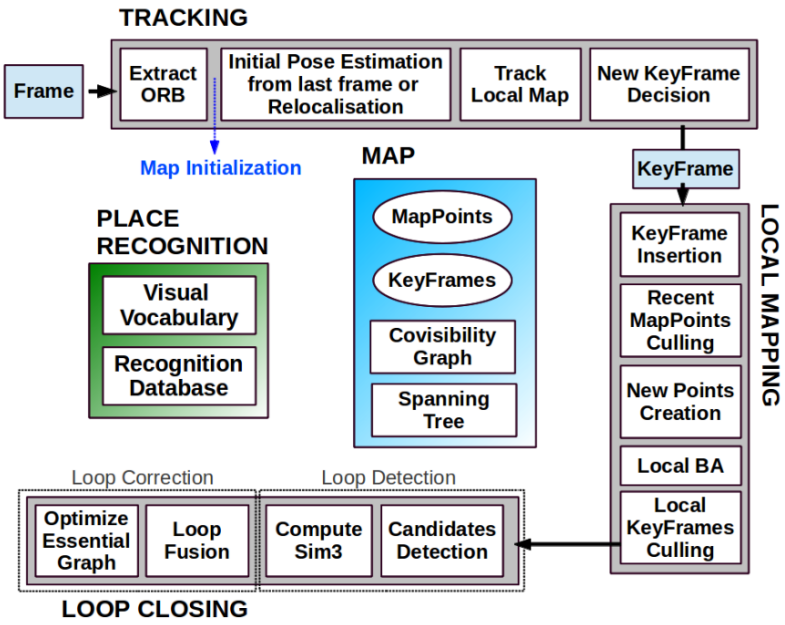


Figure 1: Overview of ORB SLAM [1]

This section explains the methodology of the the tracking and mapping threads in ORB-SLAM, as depicted in Figure 1. The discussion also handles the integral role of Oriented FAST and Rotated BRIEF (ORB) features within these processes.[1]

An ORB feature, introduced in [2], is a type of binary descriptor known for its ability to handle changes in rotation and size requiring limited computational resources. In the ORB-SLAM system, each ORB feature refers to the 2D keypoint location $\mathbf{x}_i$, and associated binary descriptor $\mathbf{D}_i$.[1] The same ORB features are utilised in the tracking, mapping, and loop closure threads.

Each scene feature, i.e. map point p_i, stores: its 3D position $\mathbf{X}_{w,i}$ in the world coordinate system; the viewing direction $\mathbf{n}_i$, a representative ORB descriptor $\mathbf{D}_i$, and the maximum $d_{\max}$ and minimum $d_{\min}$ distances at which the point can be observed according to the scale invariance limits of the ORB features.[1]

The ORB-SLAM system designates a subset of frames as keyframes K_i for which three types of information are stored: the camera pose at that keyframe $\mathbf{T}_{iw}$, a rigid body transformation that transforms points from the world to the camera coordinate system; the camera intrinsics, including focal length and principle point; and all of the ORB features extracted in the frame, which may or may not be associated to a map point.[1]

Covisibility information, i.e. observation of the same map points between keyframes, is represented as an undirected weighted graph where each node is a keyframe.[1] An edge weighted by θ connects two keyframe nodes if they share observations of at least 15 of the same map points. For the sake of computational efficiency, a sparser 'Essential Graph' is maintained that retains all keyframe nodes, but fewer edges. This includes the edges of a spanning tree which encompasses a version of the covisibility graph with the minimal number of connected edges, as well as the edges that represent strong covisibility relationships ($\theta_{\min} = 100$), and loop closure edges.

2.1 Tracking thread

The tracking thread, as implemented in `Tracking.cc`, localises the camera with each frame and decides when to insert a new keyframe.[1]

Feature extraction: The feature extraction component of ORB-SLAM adopts the ORB approach in [2]. This method combines robustness to noise, rotation invariance, and computational efficiency, making it suitable for real-time applications. The process can be detailed as follows:

1. **Detection of corners using FAST:** ORB identifies keypoints in the image using the FAST corner detector.[3] FAST works by considering a circle of sixteen pixels $I_{p,x}$ around a candidate pixel I_p, which is shown in Figure 2. The pixel is classified as a corner if a certain number of contiguous pixels in the circle are all brighter or all darker than the intensity of the candidate pixel plus a threshold t.

$$S_{p,x} = \begin{cases} darker & \text{if } I_{p,x} \leq I_p - t \\ similar & \text{if } I_p - t < I_{p,x} \leq I_p + t \\ brighter & \text{if } I_p + t \leq I_p \end{cases}$$

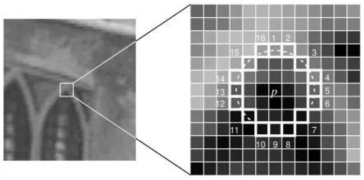


Figure 2: The process of FAST corner detection [4]

2. **Orientation computation:** To achieve rotation invariance, an orientation is assigned to each keypoint, using the intensity centroid method. The moments of a patch around the keypoint are computed, and the orientation is determined by the vector from the center of the corner to the centroid. This vector indicates the direction of the corner.

3. **Generation of the descriptor using BRIEF:** Once keypoints are detected and oriented, the Rotated BRIEF (rBRIEF) descriptor is used to encode the local image gradient patterns around the keypoint. rBRIEF, introduced by [2], is a modification of the original BRIEF descriptor [5], made rotation invariant by steering the sampling pattern according to the orientation of the keypoints.

4. **Efficiency:** ORB features are designed to be computed very quickly, with a typical requirement of less than 33ms per image to extract features. This speed is critical for the real-time capability of ORB-SLAM.[2]

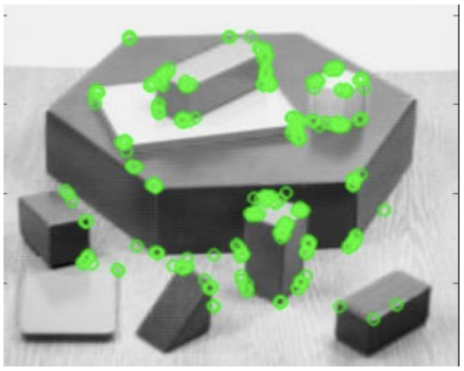


Figure 3: Example of ORB feature extraction [6]

Map initialization: Map initialization entails the computation of the relative camera pose between two frames, which is used to triangulate an initial set of map points. The tracking thread then relies on the initial camera pose to predict the camera's movement in subsequent frames, whereas the mapping thread uses the initial map points to start building the 3D map.

1. **Feature matching**: Features from current frame F_c are matched with those in an initial reference frame F_r.[1] Such feature correspondences are identified by comparing the ORB

feature descriptors $\mathbf{D}_c$ and $\mathbf{D}_r$, and represent a match in the 2D keypoint location $\mathbf{x}_c \leftrightarrow \mathbf{x}_r$. If there are not enough matches, the reference frame is reset.

2. **Computation of geometric models**: A homography $\mathbf{H}_{cr}$ and fundamental matrix $\mathbf{F}_{cr}$ are computed in parallel using the equations:[1]

$$\mathbf{x}_c = \mathbf{H}_{cr}\mathbf{x}_r, \qquad \mathbf{x}_c^{\mathsf{T}}\mathbf{F}_{cr}\mathbf{x}_r = 0$$

After these computations, normalized Direct Linear Transform (DLT) and 8-point algorithms are applied within a RANSAC scheme.[1] This involves iterative processing where, in each iteration, a score is calculated for each model based on the geometric errors between the current reference frame features. The score S_M for each model M is computed as follows:[7]

$$S_M = \sum_i \left(\rho_M\left(d_{cr}^2\left(\mathbf{x}_c^i, \mathbf{x}_r^i\right)\right) + \rho_M\left(d_{rc}^2\left(\mathbf{x}_c^i, \mathbf{x}_r^i, M\right)\right)\right) \tag{1}$$

Note that d_{cr}^2 and d_{rc}^2 are the symmetric transfer errors from one frame to the other,[1] i.e. a measurement of the discrepancy between the location of features in one image and corresponding predicted locations of features in the other. The scoring system uses a penalty function that helps differentiate between inliers and outliers. The penalty function ρ_M is defined as below, with $\Gamma = T_H$:

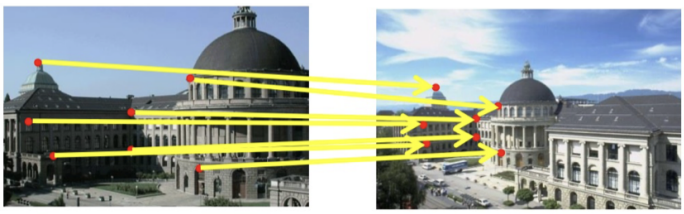
$$\rho_M(d^2) = \begin{cases} \Gamma - d^2 & \text{if } d^2 < T_M \\ 0 & \text{if } d^2 \geq T_M \end{cases} \tag{2}$$

3. **Model selection**: After computing the score, the more suitable model for this scene is selected according to this formula:[1]

$$R_H = \frac{S^H}{S^H + S^F} \tag{3}$$

If $R_H > 0.45$ the homography matrix is chosen, and otherwise the fundamental matrix.

4. **Motion and structure from motion recovery:** Once the model is selected, the associated motion hypotheses describing the camera pose are retrieved, and used to model the 3D structure of the scene as depicted in Figure 4.[1]



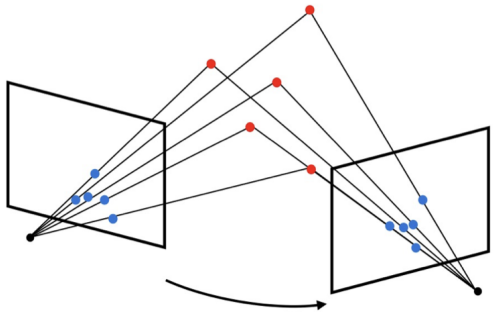


Figure 4: Example of motion estimation from two camera views

If the homography is selected, 8 motion hypotheses for the camera pose are retrieved. If the fundamental matrix is selected, it is converted to an essential matrix using the calibration matrix $\mathbf{K}$, which encapsulates the camera's focal length, principal point and skew. The resulting essential matrix is the algebraic representation of epipolar geometry in this system.[8]

$$\mathbf{E}_{rc} = \mathbf{K}^T \mathbf{F}_{rc} \mathbf{K} \tag{4}$$

Regardless of the method applied, the resulting motion hypotheses are triangulated to find the 3D world points that best align.[1] If there is no clear winner solution, with most points seen with parallax, in front of both cameras and with low reprojection error, then the map does not initialize and the initialization process restarts.

5. **Bundle adjustment**: Finally, a full bundle adjustment (BA) process is done to refine the initial reconstruction.[1] BA is the process through which map point locations $\mathbf{X}_{w,j} \in \mathbb{R}$ and camera poses $\mathbf{T}_{iw} \in \mathrm{SE}(3)$ can be optimized to minimise the reprojection error with respect to the matched keypoints $\mathbf{x}_{i,j} \in \mathbb{R}$. The error term per observation of a map point j in a keyframe i can be defined as:

$$\mathbf{e}_{i,j} = \mathbf{x}_{i,j} - \pi_i(\mathbf{T}_{iw}, \mathbf{X}_{w,j}) \tag{5}$$

Here, $e_{i,j}$ represents the difference between the observed location $\mathbf{x}_{i,j}$ and the projected location $\pi_i(\mathbf{T}_{iw}, \mathbf{X}_{w,j})$, where π_i is a projection function. The projection function π_i, calculated with the focal lengths $(f_{i,u}, f_{i,v})$ and principal points $(c_{i,u}, c_{i,v})$ of camera i, is given by:

$$\pi_i(\mathbf{T}_{iw}, \mathbf{X}_{w,j}) = \begin{bmatrix} f_{i,u} \frac{x_{i,j}}{z_{i,j}} + c_{i,u} \\ f_{i,v} \frac{y_{i,j}}{z_{i,j}} + c_{i,v} \end{bmatrix} \tag{6}$$

In this expression, $x_{i,j}$ and $y_{i,j}$ are the x and y coordinates of the 3D point in the camera's coordinate system, and $z_{i,j}$ represents its depth. The projection function π_i maps the 3D world coordinates $\mathbf{X}_{w,j}$ to 2D image coordinates $\mathbf{x}_{i,j}$.
The reprojection error is then used in the minimization of the cost function, using the Levenberg-Marquardt algorithm implemented in g2o.

$$C = \Sigma_{i,j} \rho_h(\mathbf{e}_{i,j}^T \Omega_{i,j}^{-1} \mathbf{e}_{i,j}) \tag{7}$$

Here, ρ_h is the Huber robust cost function and $\Omega_{i,j}$ is the covariance matrix associated to the scale at which the keypoint was detected. As a result, we obtain the optimal camera poses and 3D world coordinates that best explain the observed 2D image coordinates, given the camera parameters and the observed scales.

Pose estimation: Pose estimation allows the system to understand on an ongoing basis where the camera, and its new observations, are in relation to the world and previously mapped points. Given that tracking was successful in the previous frame, a constant velocity motion model which assumes vehicle move with same speed is used to predict the camera pose in the current frame.[1] This prediction is used to guide a search for the matches between the ORB features associated with map points observed in the last frame, and ORB features in the current frame, F_c. If too few matches are found (< 100),[9] a wider search of the map points around their position in the last frame is conducted.

If the search is successful, the camera pose $\mathbf{T}_{iw}$ at this keyframe is then optimized with respect to the found correspondences. This is done using motion-only bundle adjustment.[1] In motion-only BA, the method is similar to that outlined above when describbing map initialization, but all points are considered as fixed and only the camera pose is optimized.

Global relocalization: If the search was unsuccessful and tracking is lost, the system undertakes global relocalization of the camera in the map.[1] This process relies on the ability to recognise the place based on appearance. This is enabled via a 'bag of words' model [10] which represents distinctive features extracted from images of a place using visual words. The vocabulary database can be queried to identify keyframes that have a similar bag of words representation to the current frame and are therefore likely to depict a similar scene.

Once the keyframe candidates from the vocabulary database have been identified, a subset of the correspondences between the ORB features associated with map points in the current frame and those in each candidate keyframe are computed.[1] RANSAC is performed on these correspondences, alternated with estimating the camera pose relative to the correspondences, using the PnP algorithm as implemented in `PnPsolver.cc`.

If a camera pose with enough correspondence inliers is found, the pose undergoes an initial optimization. This is performed using a local Bundle Adjustment, which minimizes the reprojection error over a subset of keyframes and map points relevant to the current view. The guided search of map points for matches is applied as described above. The camera pose is optimized again, and if supported with enough inlier correspondences, the tracking procedure continues.[1]

Local map tracking: Once tracking has been established, the system projects a local map into the frame and searches for more map point correspondences.[1] The covisibility graph is used to retrieve a smaller, local visibility map which is more computationally efficient. The local map contains the set of keyframes K_1 that share map points with the current frame, and a set K_2 with neighbours to the keyframes K_1 in the covisibility graph. There is also a reference keyframe $K_{ref} \in K$ which shares most map points with the current frame.

Each map point seen in K_1 and K_2 is searched in the current frame, following the steps below to decide whether it is included or discarded.

1. The map point projection $\mathbf{x}$ is computed in current frame and discarded if it is out of the image bounds.

2. The angle between the current viewing ray $\mathbf{v}$ and the map point mean viewing direction $\mathbf{n}$ are computed and ignored if $\mathbf{v} \cdot \mathbf{n} < \cos(60)$.

3. The distance $\mathbf{d}$ from the map point to the camera centre is computed, and ignored if it is out of the scale invariance region of the map point $d \notin [d_{\min}, d_{\max}]$.

4. The scale in the frame is computed by the ratio $\frac{d}{d_{\min}}$. This is done to normalize the scale of the map point in the current frame, which can help with matching and association.

5. The representative descriptor $\mathbf{D}_i$ of the map point is compared with the still unmatched ORB features in the frame, at the predicted scale, near $\mathbf{x}$. The map point is associated with the best match.

The camera pose is then optimized using BA from Equation 5 with the map points found in the frame.[1]

Addition of new keyframes: In ORB-SLAM, keyframes are inserted as fast as possible to make tracking more robust to challenging camera movements, such as rotations.[1] These are then culled during the local mapping process.

Keyframes are inserted if the system is stable, tracks a good understanding of scene structure, and the scene has changed significantly since the last keyframe insertion.[1] This helps balance accuracy, efficiency and computational requirements. Accordingly, keyframes are inserted when: at least 20 frames have passed since the last global relocalization; local mapping is idle or at least 20 frames have passed from the last keyframe insertion; the current frame tracks at least 50 map points, and the current frame tracks less than 90% of the points in K_{ref}.

2.2 Mapping thread

The mapping thread constructs and refines a 3D map of the environment.

Keyframe insertion: A new node, K_i, is added to the covisibility graph.[1] Edges are updated based on shared map points with existing keyframes. K_i is linked to the keyframe sharing the most points in the spanning tree in the Essential Graph, and its bag of words representation is computed to aid in triangulating new points.

Recent map point culling: Map points must pass a test in the initial three keyframes:[1]

1. The point should be found in more than 25% of frames where visibility is expected.

2. If more than one keyframe has elapsed since creation, the point must be observed from at least three keyframes.

If points have passed this test, they are removed only if observed in fewer than three keyframes at

a time, e.g. due to culling of keyframes or outlier rejection in local bundle adjustment.

New map point creation: New map points are triangulated from ORB features in connected keyframes K_c in the covisibility graph.[1] A match for each unmatched ORB in K_i is sought in each of the other connected keyframes, discarding those matches that do not meet the epipolar constraint. The criteria for acceptance include positive depth, parallax, reprojection error, and scale consistency in both cameras.

Local bundle adjustment: Local BA is similar to BA, but only a subset of the camera poses and map points are optimized. Specifically, local BA optimizes the currently processed keyframe K_i and its connected keyframes K_c in the covisibility graph, along with all respective visible map points.[1] Keyframes viewing those map points but not connected to K_i remain fixed. Outlier observations are discarded.

Local keyframe culling: Redundant keyframes are detected and removed to maintain a compact reconstruction and manage bundle adjustment complexity.[1] A keyframe in K_c is discarded if 90% of its map points are also observed from at least three other keyframes in the same or finer scale, ensuring measurement accuracy.

3 Evaluation results on KITTI 07 and KITTI 10 sequences

The KITTI 07 and KITTI 10 sequences are part of the KITTI visual odometry dataset, an outdoor dataset captured by a vehicle driving around Karlsruhe, Germany.[11] It is a challenging dataset for monocular vision given the vehicle's fast rotations, relatively high car speed with sequences recorded at 10fps, and areas with foliage.[1] Given the monocular ORB-SLAM task, the greyscale version of the data was used.

3.1 Evaluation metrics and methods

The end-to-end performance of the entire SLAM system can be evaluated by comparing the estimated trajectory to the ground truth, when available, using absolute trajectory error (ATE). The ATE aligns the respective trajectories and evaluates the absolute distances between the two.[12]

The alignment of the coordinate frames of the estimated camera and ground-truth poses can be done for monocular systems using 7-degrees of freedom (DoF) alignment of rotation, translation and scale.[8] In the `evo` library, the alignment is calculated using the closed-form solution in [13], with an option to align scale.[14] This is applied in the `evo_ape` bash script using `--as`.[8] Note that the KITTI ground truth data must be converted to TUM format for use here, which was done using the pre-provided function `kitti_to_tum.py`.[8, 14]

The `evo` library provides a straightforward way to extract ATE as a form of the absolute pose error (APE) metric.[15] The APE is based on the absolute relative pose E_i between the true

reference pose $\mathbf{T}_{iw,\text{ref}}$ and current estimated pose $\mathbf{T}_{iw,\text{est}} \in \text{SE}(3)$ at timestep i:

$$E_i = \mathbf{T}_{iw,\text{est}} \ominus \mathbf{T}_{iw,\text{ref}} = \mathbf{T}_{iw,\text{est}}^{-1} \mathbf{T}_{iw,\text{ref}} \in \text{SE}(3)$$

Here, $\ominus$ represents the inverse compositional operator which takes two poses and gives their relative pose.[14, 16]

The APE in the `evo` library can be interpreted in several variations, including the translation (i.e. distance), the rotation angle, or the rotation part (i.e. orientation) errors.[14] Accordingly, we isolate the translation error component using `--pose_relation trans_part` in the `evo_ape` bash script, which calculates the ATE per timestep:[14, 17]

$$\text{ATE}_i = ||\text{trans}(E_i)||$$

The `evo` library also allows for visualization of the ATE values over time, as well as the associated metrics covering the cumulative results of the run (Root Mean Squared Error (RMSE), mean, median, and standard deviation).[15] The RMSE is the primary metric used for assessment of ATE in [12], and is calculated as:

$$\text{RMSE} := \sqrt{\frac{1}{n} \sum_{i=1}^{n} \text{ATE}_i^2}$$

An alternative view can be provided via a heatmap of ATE values over time, overlaid onto a map of the trajectory projected with the ground truth.[15] We use the heatmap as the primary visualization for our analysis, but include the cumulative result charts for additional information. Please note that while we have not amended the automatic titling of the charts exported via `evo`, where 'APE' is included in chart titles it corresponds to the ATE component as detailed here.

3.2 Randomness in experiments

In each simulation with ORB-SLAM, it is possible to generate slightly different results due to the following factors:

1. **Threshold sensitivity:** Various thresholds within the ORB SLAM process are sensitive to minor variations, which can alter trajectory estimations and map optimization outcomes.

2. **Optimization heuristics:** The heuristics used for feature matching and map optimization can converge to slightly different solutions, influenced by initial conditions and specific paths of convergence.

3. **Loop closure detection:** Variations in feature detection and matching across simulations can lead to differences in loop closure detection, influencing global map consistency.

4. **Initial conditions and state estimation:** Slight variations in the initialization phase can propagate through the SLAM process, resulting in different outcomes.

Accordingly, we have illustrated the results of multiple runs in some cases below.

3.3 Default system configuration

In our experimental setup, the system was first operated using its standard configuration options which include 2000 features per frame. As documented in [1], this setting is best for analysis of the KITTI dataset, characterized by its relatively high resolution of 1241×376 pixels.

This and other configuration parameters are detailed in the `KITTI04-12.yaml` file, accessible within the `Install/etc/orbslam2/Monocular` directory.[9] It specifies the `ORBextractor.nFeatures` parameter, which dictates the number of ORB features extracted per image. It also includes the lens calibration parameters tailored for the KITTI dataset sequence, as detailed in [18].

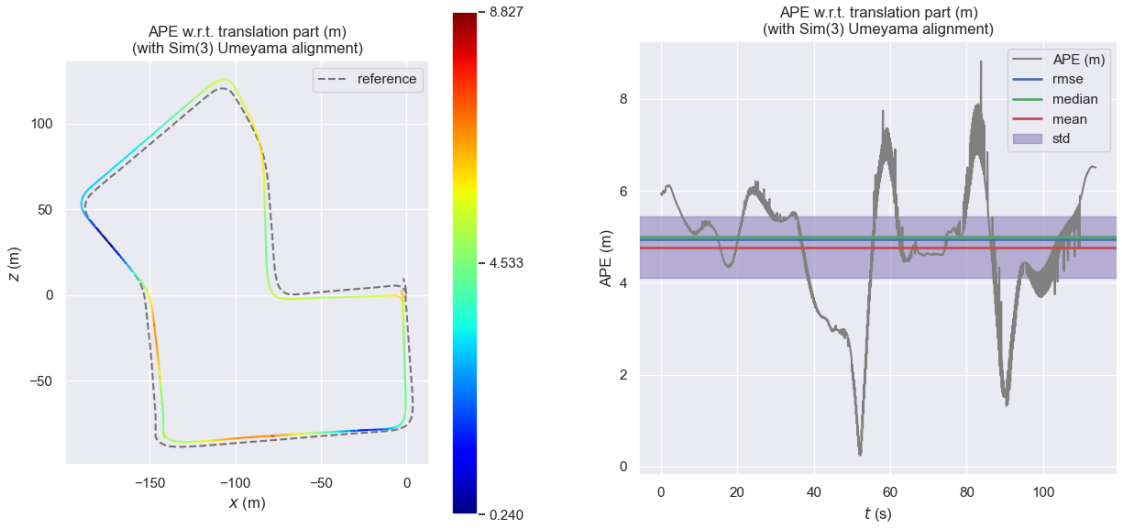


Figure 5: Map overlay and plot of ATE on KITTI 07 with system default settings

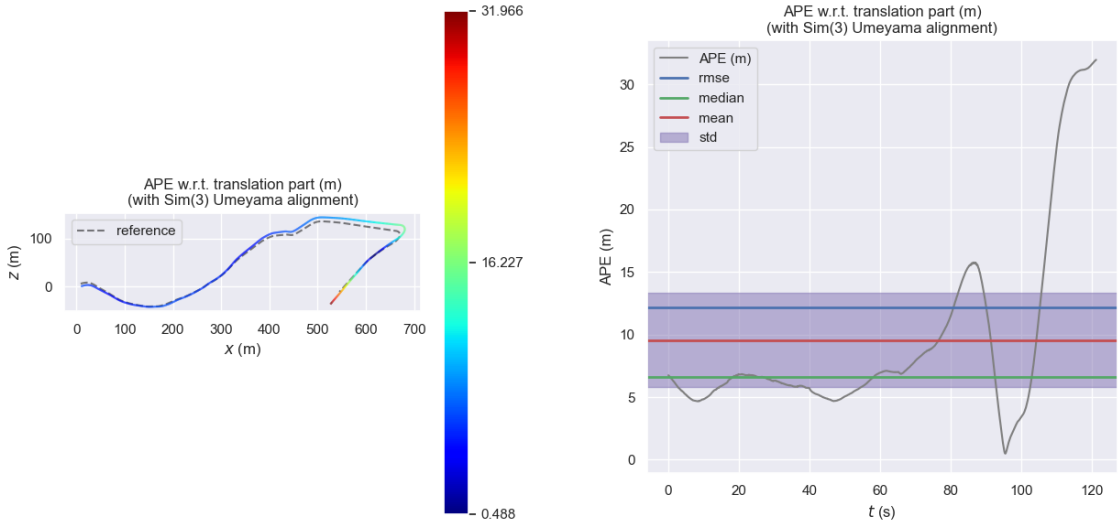


Figure 6: Map overlay and plot of ATE on KITTI 10 with system default settings

3.4 Reduce the number of ORB features

We explored the impact of ORB feature quantity on system performance. We systematically varied the number of ORB features utilized by the system, conducting tests across three settings: 1000, 1100, and 1600 features.

The modifications necessary for this experiment were implemented in the `KITTI04-12.yaml` configuration file, situated in the `Install/etc/orbslam2/Monocular` directory.

3.4.1 Number of ORB features: 1000

The SLAM system failed to initialize when the number of ORB features was set to 1000, in both sequences. This failure indicates that the quantity of features at this threshold was inadequate for initializing a reliable initial map of the environment according to the methodology described in Section 2.1. Feature-based SLAM systems require a certain minimum number of features to successfully triangulate and orient themselves within the space.

3.4.2 Number of ORB features: 1100

As illustrated in this section, increasing the ORB feature count to 1100 resulted in diminished performance in estimating the camera trajectory in both the KITTI 07 and KITTI 10 datasets.

For the KITTI 07 sequence, this impacted initialization as well as the ability to maintain tracking. As mentioned in Section 3.2, variability in system runs allows for slightly different results. Two cases below, run with the same dataset and system settings, allow for detailed analysis of this experiment.

Figure 7 shows the case whereby the system maintained tracking for the entire run. However, similar to the experiment in Section 3.4.1, the limited number of features made map initialization difficult. Tracking started after a substantial number of frames have been processed (taking around 10 seconds), precluding loop closure and resulting in an unconnected map. In addition, the overall ATE level is significantly higher, with an RMSE of approximately 15 compared to 5 in Section 3.4.1. This observation underscores that a feature count of 1100 ORB features is insufficient for reliable mapping of the KITTI 07 sequence.

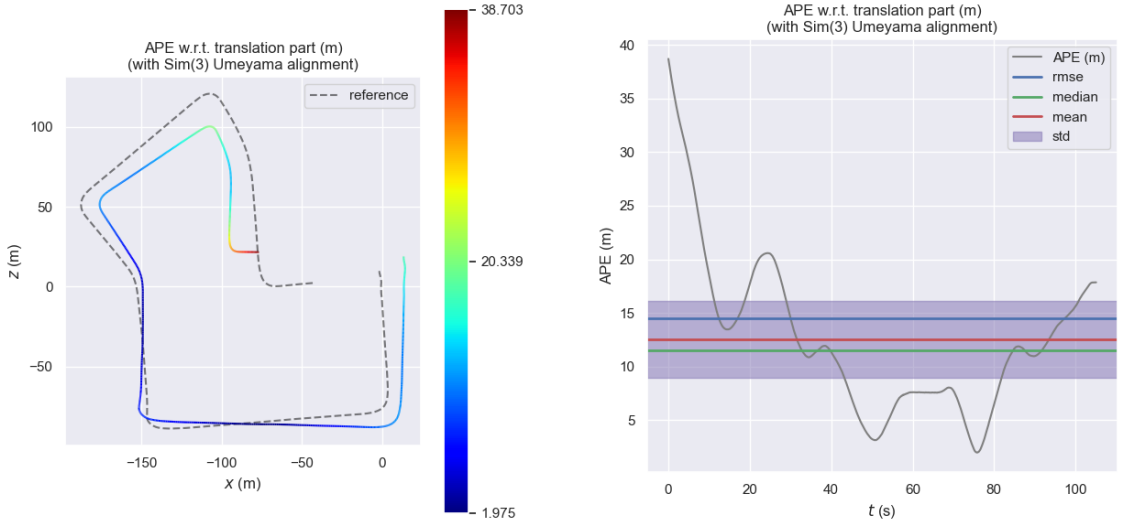


Figure 7: Map overlay and plot of ATE on KITTI 07 with 1100 features, case 1

The second case in Figure 8 showed that while the feature count was high enough to allow the system to initialize, performance was also limited due to the intrinsic characteristics of the detected ORB features. In scenarios where the visual landscape is either texture-sparse or has repetitive patterns, ORB feature descriptors may lack sufficient distinctiveness for reliable inter-frame matching. Therefore, in the presence of low-quality features, even an ostensibly sufficient feature count may fail to avert tracking discontinuities. This is evident in Figure 8, where the system lost track of its location in the KITTI 07 sequence and failed to accomplish global relocalization as discussed in Section 2.1. The overall ATE was also higher than in Section 3.4.1, here 10 as opposed to 5.

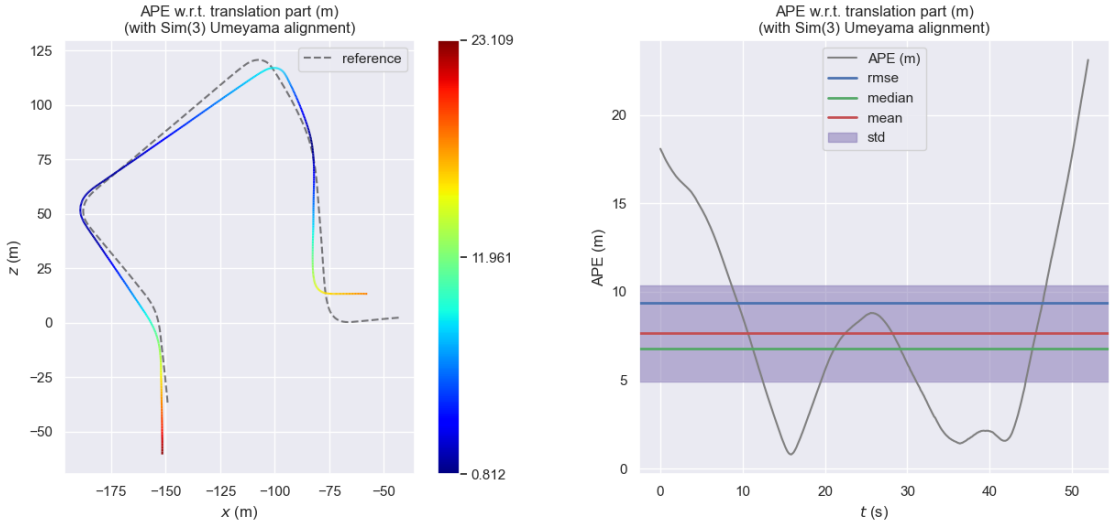


Figure 8: Map overlay and plot of ATE on KITTI 07 with 1100 features, case 2

These dynamics also helped to explain system performance on KITTI 10, in which the images show a fairly repetitive forest sequence. As evident in Figure 9, the SLAM system was only able to maintain operation for less than 10 seconds, after which global relocalization failed. It also took substantial time to initialize. The results are so limited that it is impossible to recognize the trajectory in comparison with Figure 6, and the overall RMSE is extremely low reflective of the limited data on which to calculate this metric.

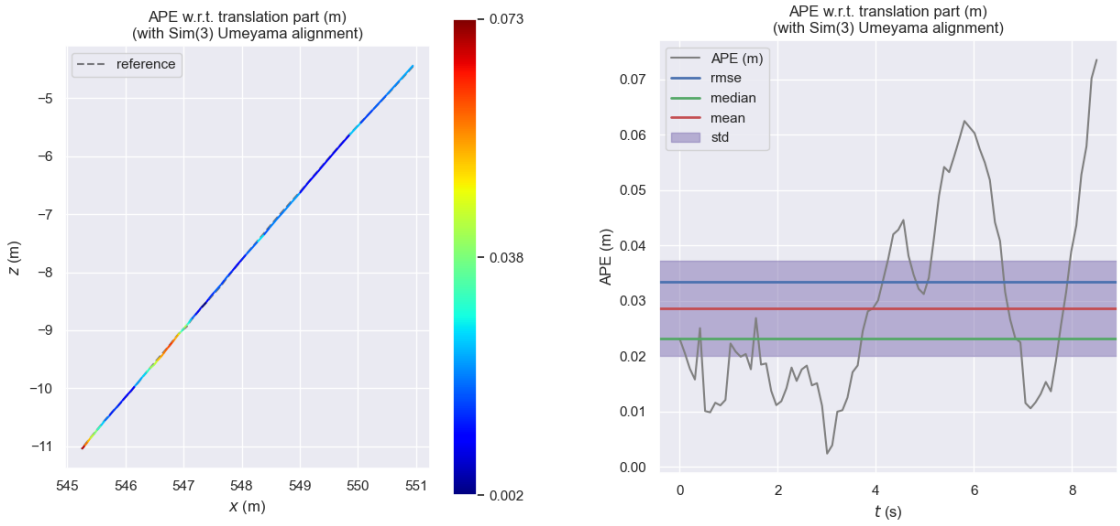


Figure 9: Map overlay and plot of ATE on KITTI 10 with 1100 features

3.4.3 Number of ORB features: 1600

Figure 10 and Figure 11 show that indicate that the trajectory generated with 1600 features, using both datasets, closely resembles that produced by the default configuration. The overall ATE levels also are fairly similar, suggesting a good degree of accuracy. However, the ATE error in the KITTI 07 run appears noisier over time, likely due to the reduced number of ORB features detected and utilized during system operation.

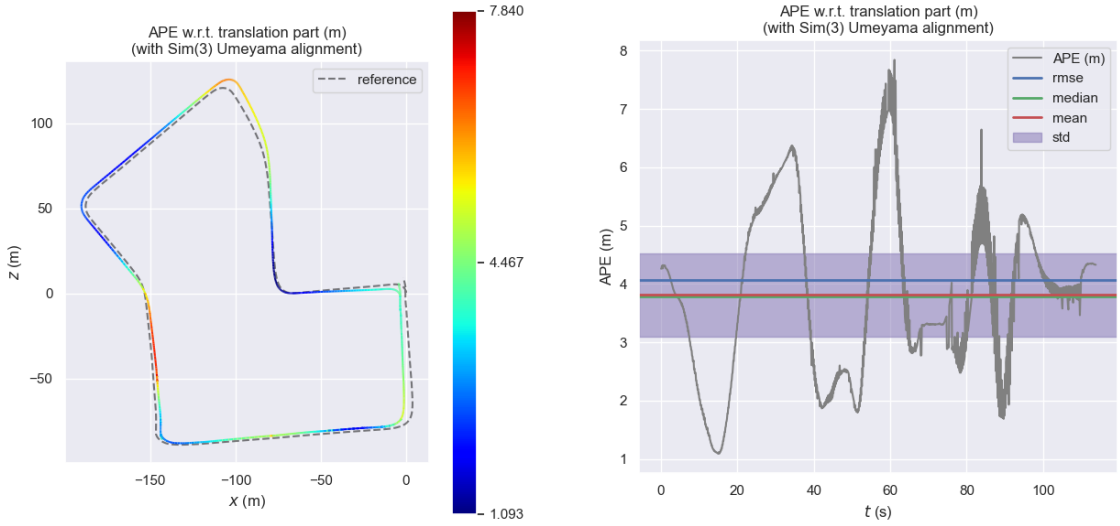


Figure 10: Map overlay and plot of ATE on KITTI 07 with 1600 features

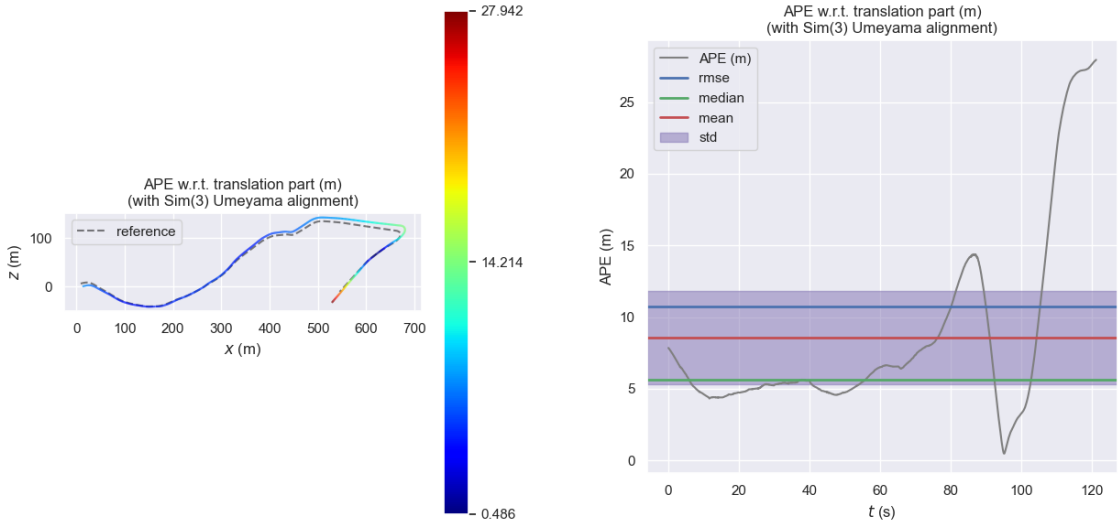


Figure 11: Map overlay and plot of ATE on KITTI 10 with 1600 features

3.5 Turn off the outlier rejection stage

A modification was made in the ORB-SLAM2 system's source code to turn off outlier rejection (RANSAC). The method invocation `setLevel(1)` at line 386 was changed to false (0) in the

`Optimizer.cc` file located in the `Source/Libraries/ORB_SLAM2/src` directory. Originally intended to enable the outlier rejection mechanism which is calculated at several points in the system, this change ensures that all data points, including potential outliers, are included during the optimization process.

This adjustment has a noticeable impact on the system's performance on both datasets. The inclusion of outliers in the dataset can significantly impact the optimization process, which attempts to minimize the error across all data points, including the erroneous outliers. This can lead to inaccurate estimates of both the camera pose and the 3D structure of the scene, as the optimization process is skewed by the attempt to accommodate the outliers.

For KITTI 07, as illustrated in Figure 12 there is a significant spike in ATE around the 70-second mark, which is also reflected in the mis-estimated trajectory along the bottom of the map. However, the system was still able to correct its estimated position via loop closure.

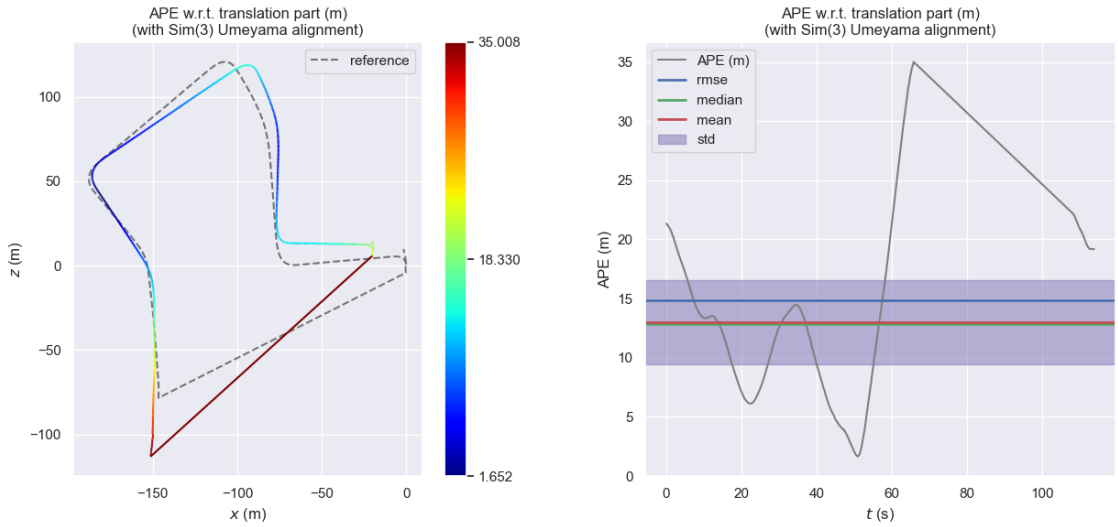


Figure 12: Map overlay and plot of ATE on KITTI 07 without outlier rejection

Similarly to the experiment in Section 3.4.3, the system running on KITTI 10 sequence was only able to maintain operation for a short period over 16 seconds, as shown in Figure 13. The ATE is low given the limited data on which to calculate this metric, and extremely noisy. The system was not able to develop an accurate or recognizable map of this dataset.

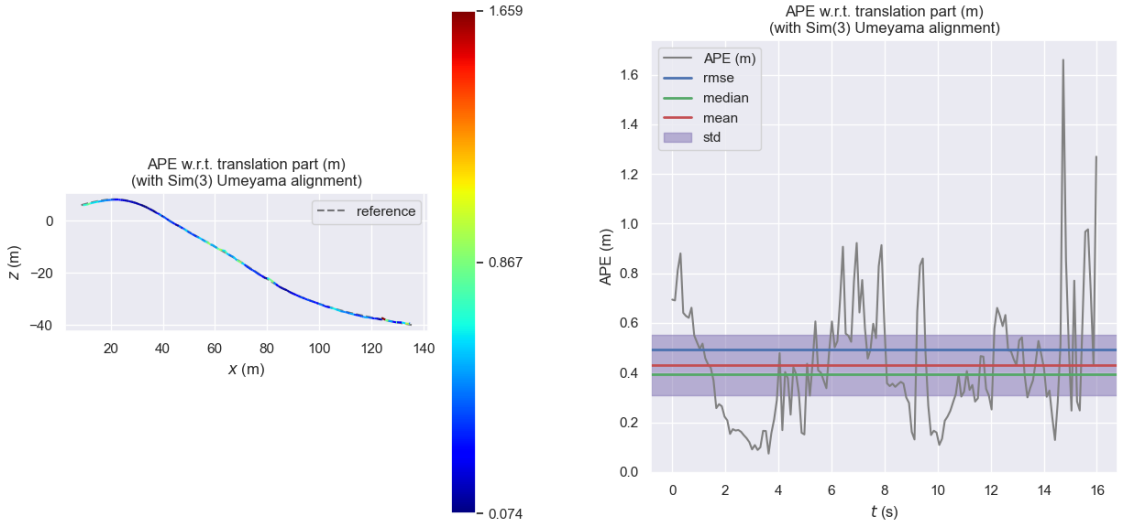


Figure 13: Map overlay and plot of ATE on KITTI 10 without outlier rejection

3.6 Turn off the loop closure

Loop closure enables the system to recognize previously visited locations and corrects the drift accumulated over time. It does so by merging points that exhibit similar features, such as corners or distinct objects, thereby ensuring the consistency and accuracy of the map. To assess the impact of disabling the loop closure mechanism, modifications were made to the ORB-SLAM2 system's source code. Specifically, lines 495 to 507 in the `LoopClosing.cc` file, located in the `Source/Libraries/ORB_SLAM2/src` directory, were commented out, effectively deactivating the loop closure function.

The result of this modification is clearly observable in Figure 14. Without the loop closure functionality, there is a noticeable disconnect between the starting and ending points of the trajectory. Under system default configuration (Figure 5 in Section 3.3), the map's endpoints converge, accurately reflecting the actual navigation path where the start and end locations coincide. In contrast, the absence of loop closure leads to a disjointed representation, highlighting the essential role of this feature in maintaining the integrity and continuity of the map.

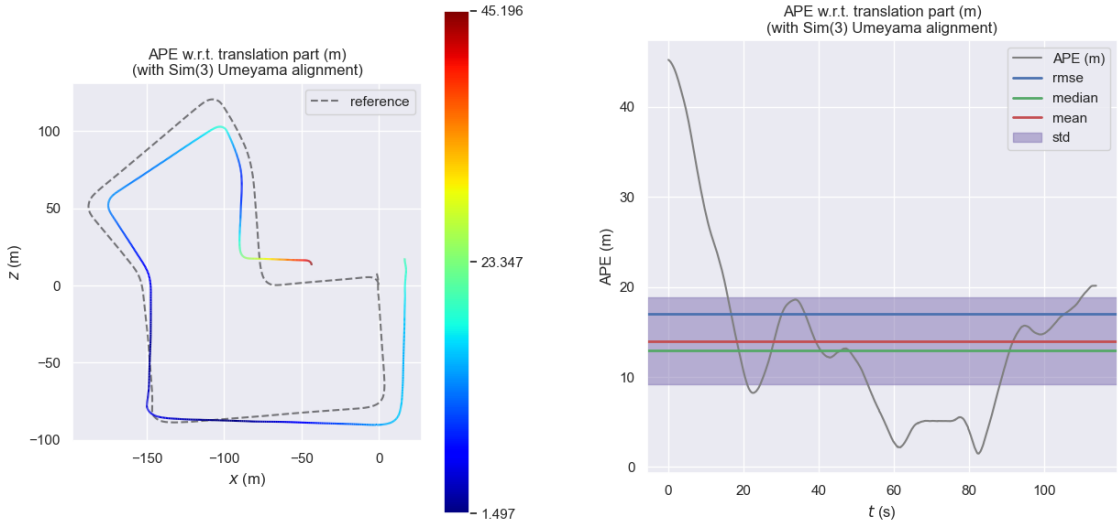


Figure 14: Map overlay and plot of ATE on KITTI 07 without loop closure

Given there is no loop in KITTI 10 sequence (Figure 15), the results between the default and setting without loop closure are not different, since the function of loop closure is activated when the system detects similar features and corrects its position.

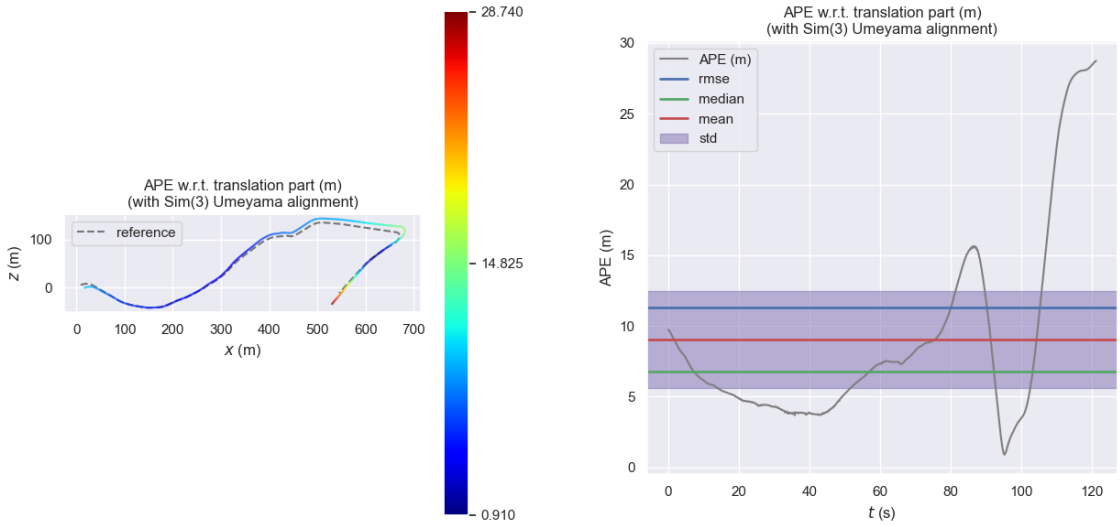


Figure 15: Map overlay and plot of ATE on KITTI 10 without loop closure

4 Conclusion

Figure 16: Example of system operation [9]

In this coursework we evaluated ORB-SLAM, specifically its performance on two distinct sequences from the KITTI dataset. Theoretical analysis was done on the tracking and mapping threads, which ensure real-time responsiveness to and self-localization in the environment, while progressively enriching the map. Modifications to the system included varying the number of ORB features, as well as toggling outlier detection and loop closure functionalities. These alterations allowed us to assess the individual impact of each component within the algorithm, thereby gaining a deeper understanding of monocular visual feature-based SLAM systems.

The experimentation with decreasing the number of ORB features showed that this can have a significant impact on the initialization of the system, as well as the system's ability to maintain tracking and successfully apply global relocalization. Lower numbers of ORB features impacted the system to the point that in the worst cases it was unable to initialize at all, or when it ran it did not provide a recognizable map of the environment. The next experiment, removing the outlier rejection capabilities, was shown to significantly impact the optimization process and either result in extremely erroneous trajectories over time as outliers accumulated, or once again impact the system's ability to maintain tracking and mapping and result in short system operation times. Finally, removing loop closure showed the significant extent of accumulated drift in self localization over time, where the position at the start and end of a single loop were no longer matched. However, this was only evident in datasets with a loop.

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, p. 1147–1163, Oct. 2015. [Online]. Available: http://dx.doi.org/10.1109/TRO.2015.2463671

[2] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[3] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:1388140

[4] D. Tyagi, "Introduction to orb (oriented fast and rotated brief)," January 2019. [Online]. Available: https://medium.com/@deepanshut041/introduction-to-orb-oriented-fast-and-rotated-brief-4220e8ec40cf

[5] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision – ECCV 2010*, K. Daniilidis, P. Maragos, and N. Paragios, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 778–792.

[6] "Oriented fast and rotated brief." [Online]. Available: https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html

[7] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[8] L. Agapito. Robot Vision and Navigation Topic 3 Lab 2. Google Docs. [Online]. Available: https://docs.google.com/document/d/1K9x8G20f2VZFSz8fIfaxA8LH9IFnlZK-9Px2qB7ScIU/edit?usp=sharing&usp=embed_facebook

[9] R. Mur-Artal, J. D. Tardos, J. M. M. Montiel, and D. Galvez-Lopez, "Orb-slam2," January 2017. [Online]. Available: https://github.com/raulmur/ORB_SLAM2/tree/master?

[10] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 2161–2168.

[11] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[12] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 573–580.

[13] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 4, pp. 376–380, 1991.

[14] Evo/notebooks/metrics.py_API_Documentation.ipynb at master · MichaelGrupp/evo. [Online]. Available: https://github.com/MichaelGrupp/evo/blob/master/notebooks/metrics.py_API_Documentation.ipynb

[15] M. Grupp, "evo: Python package for the evaluation of odometry and slam." https://github.com/MichaelGrupp/evo, 2017.

[16] F. Lu and E. E. Milios, "Globally consistent range scan alignment for environment mapping," *Auton. Robots*, vol. 4, no. 4, pp. 333–349, 1997. [Online]. Available: https://doi.org/10.1023/A:1008854305733

[17] APE and ATE · Issue #247 · MichaelGrupp/evo. GitHub. [Online]. Available: https://github.com/MichaelGrupp/evo/issues/247

[18] Running Mono · UCL/COMP0130_22-23_Topic_03 Wiki. [Online]. Available: https://github.com/UCL/COMP0130_22-23_Topic_03/wiki/Running-Mono