COMP0130 Robotic Vision and Navigation Group X Coursework 3*

Yuansheng Zhang †, Joseph Rowell ‡& Vanessa Igodifo §

22/04/2022

Introduction

This report reviews the tracking thread of monocular ORB-SLAM 2, detailed in section 1. The Refactored ORB-SLAM 2 algorithm [1] is evaluated on the KITTI [2] and TUM [3] RGB-D SLAM Datasets and Benchmarks, in monocular mode. The algorithm is evaluated under different conditions in section 2.

1 Tracking Thread of Monocular ORB-SLAM 2

The SLAM system used is that of ORB-SLAM 2, an open source SLAM system that was utilised for Monocular SLAM. The general flowchart for the operation of ORB-SLAM 2 is shown in Figure 1 below.

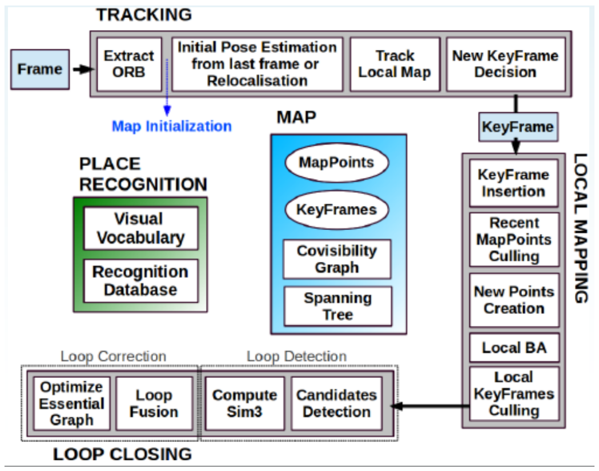


Figure 1: ORB-SLAM 2 Flowchart, composed of three main parallel threads: Tracking 1, Local Mapping and Loop Closing [4].

The tracking is in charge of localizing the camera with every frame and deciding when to insert a new keyframe [5]. For every frame of the input image sequence (where frames are collected synchronously), tracking is performed. This is the process by which the camera position is localised given the current and previous frames as well as estimated feature correspondences. We essentially build up a map of the camera's position through time by iterating over a series of four steps, outlined in Figure 1.

*This work was submitted in partial fulfillment of the UCL COMP0130 Robotic Vision and Navigation Module

†yuansheng.zhang.18@ucl.ac.uk

‡joseph.rowell.21@ucl.ac.uk

§vanessa.igodifo.21@ucl.ac.uk

1.1 Extracting ORB Features

This is the method by which we extract the salient features of the frame input to ORB, each feature with a specific descriptor - a representation of the local image appearance around the point. We assume that there is enough texture in the image to allow apparent motion and distinct features to be extracted [6]. Feature detection is the process by which salient points, typically located in corners or regions of the image with distinctive textural components, are detected. The goal is to be able to accurately match these detected features between consecutive frames, as this will clarify to us that these points represent the projection of the same position in 3D space in both images. We will then have prior information to reason about the camera pose from this knowledge. The disparity between the point's location in these images is enough to come to conclusions about the location of the camera at that time in question.

Feature detectors are used to find these salient points. They locate areas of interesting local texture that can be repeatedly found in other images to establish pairwise correspondences between them in the feature mapping process. The ORB-SLAM 2 (Oriented FAST and Rotation BRIEF Simultaneous Localisation and Mapping) algorithm combines the properties of two feature detectors, FAST (Features from Accelerated Segment Test), and BRIEF [7] (Binary Robust Independent Elementary Features) to detect distinct features in images.

FAST [8] is a corner detection method that is far more computationally efficient than other techniques. It sequentially compares the intensities of each pixel in an image to a surrounding circular neighbourhood of pixels to determine 'interest points' in an image, where an interest point is an area of expressive texture, often where we find edges and object boundaries. This is done using the ID3 algorithm (based on decision tree classifiers) where entropy minimisation is exploited to find the label of the pixel in question as an interest point or not, in the fewest number of operations, hence the name FAST. The process of non-maximal suppression also allows the algorithm to avoid detecting adjacent feature points that will add minimal information to the system. The main evident downfall of FAST is its inability to detect corners that are perfectly aligned with the Cartesian axes, and smoothing filters are often used to blur images in a way that allows for the detection of corners. Although, the FAST algorithm does not give orientation of the corners identified. The steps of the FAST algorithm are as follows [8]:

- Iterate over all pixels p in the image frame, to identify salient points of interest. Let the intensity be Ip.
- Consider a ring of pixels around the pixel p, e.g. Bresenham circle of radius 3 gives 16 pixels.
- Pixel p is identified as a corner if there exists a a set of continuous pixels n in the ring, which exceed the intensity $I_p + t$, or lesser $I_p - t$. States of each pixel are determined as follows in Equation 1:

$$S_{p\to x} = \begin{cases} d(\text{darker}), & \text{if } I_{p\to x} \leq I_p - t \\ s(\text{similar}), & \text{if } I_p - t < I_{p\to x} < I_p + t \\ b(\text{brighter}), & \text{if } I_p + t \leq I_{p\to x} \end{cases} \tag{1}$$

- If $\frac{3}{4}$ pixel values are not above or below $I_p \pm t$, then the pixel in question is not a corner/ point of interest.

BRIEF [9] takes the neighbourhood of pixels around a keypoint and converts this image patch into a binary feature descriptor whose elements contain only the values 1 and 0. A descriptor can be thought of as a quantitative representation of the local image appearance around that a keypoint [6]. A Gaussian kernel is then used to smooth this patch in order to reduce noise, as pixel-level processing occurs as part of this procedure, so this algorithm is very susceptible to variation in pixel intensity [9]. Hence, to increase the robustness and usefulness of the descriptors in the matching process, we perform smoothing. The method for generating the corresponding binary feature vectors and consequent BRIEF descriptor is outlined below:

$$f_{n_d}(\mathbf{p}) := \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y_i})$$
$$\text{where } \tau(\mathbf{p}; \mathbf{x}_i, \mathbf{y_i}) = \begin{cases} 1 & : \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 & : \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases} \tag{2}$$

The process of constructing and matching BRIEF descriptors between images is much faster than for other methods, and recognition rates are also significantly higher so long as it is not a requirement to maintain in-plane rotational invariance [9].

ORB improves on both of these techniques to add an orientation component to the FAST feature detector algorithm as well as improved efficiency, variance and correlation analysis, and decorrelation of BRIEF features in the case where we seek to maintain rotational invariance in the keypoints. Using this, we can extract scale invariant keypoints with an orientation component, as well as build unique, local feature descriptors that maintain rotational invariance. It can be extrapolated that descriptors close in vector space represent the same feature point [10].

1.2 Initial Pose Estimation

From the pairwise correspondences that arise from the feature detection and matching processes, we can form an initial estimate of the camera pose. This can be described as a 2D-to-2D feature-based motion estimation problem, where a constant velocity motion model is incorporated to predict the camera pose [5]. At this point, the image details beyond the keypoints are essentially disregarded, and we simply look to these features for our solution. A global optimisation to optimise the 3D structure and camera poses using a given number of previous frames. The best results would arise in the case that all prior frames are taken into account in this optimisation process, however, this would require a large amount of computational power and memory.

The method involves looking at pairs of images and using pairwise correspondences to initialise the camera pose, then by using bundle adjustment we can jointly refine the camera poses and 3D structure. The 3D coordinates of keypoints project onto the 2D coordinates of an image from a camera using the projection matrix that corresponds to this camera, and this will project into the camera at its next location using its projection matrix (which is related to its pose). This continues for all camera locations in the sequence. With good estimates of the camera poses as well as the 3D keypoints, we can use the projection equation to show a minimal image re-projection error — this is the distance between where the point is projected onto and where we measure the feature to actually be. Ultimately, the re-projected estimates of all keypoints via each camera's projection matrix should be the same as the measurements we make when we extract the ORB features [6]. The rotation and translation of the camera in the world coordinate system as well the coordinates of the features will be treated as unknowns in this non-linear least squares problem. Therefore, a good initial estimate of the camera pose is required to ensure the optimisation does not end up converging at a local minima, but instead, the global minimum.

The first step is receiving a new image. Given some point correspondences, the current, and previous frames as well as some matched features, we compute the relative motion of the camera in the time when these two images were captured to estimate its rigid transformation (consisting of a translation and rotation), where we consider 6 degrees of freedom in total.

We'll now discuss perspective projection. Generally, when modelling this, we treat the camera as a simplified pinhole camera model, where all the rays pass through the centre of the lens. The centre of projection lies at the centre of the lens, and real world points are projected onto the image plane after passing through here. We can define the focal length, f, as the distance between the centre of projection and image plane, and the object distance, Z, as the distance between the centre of projection and the object. This setup is shown below in Figure 2. The relationship between the image and object heights can be determined by considering triangle similarities, as shown in Equation 3.

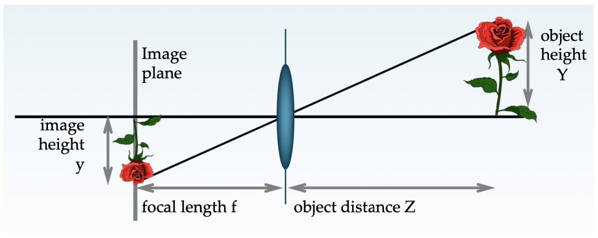


Figure 2: Pinhole Camera Image height [6]

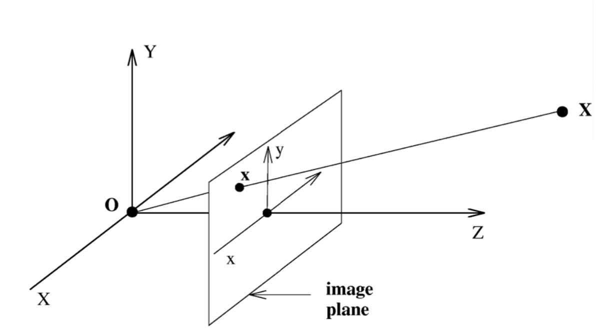


Figure 3: Projection Matrix [6]

$$
\begin{aligned}
\text{Similar triangles: } \frac{y}{f} &= \frac{Y}{Z} \\
\text{Projection: } y &= \frac{f}{Z} Y
\end{aligned}
\tag{3}
$$

Equation 3 can be rearranged to give an equation for calculating the image height in terms of the object height in the real world, the focal length, and the object distance in a pinhole camera setup, shown in Figure 2.

Now, we consider all 3D coordinates points. We take the **X** vector to be the 3D coordinates of the point in the world, **O** to be the centre of projection, and **x** is the projection of the object in the image plane. However, we can use the camera as the origin of the coordinate system and refer to the elements of this vector as the camera coordinates and the focal length.

We firstly define perspective projection in the camera coordinate system, which, to recall is aligned with the central projection, under the assumption we know the camera calibration. It's possible to express the projection of the object in such a way as described by Equation 4, where lambda is defined in Equation 5. This configuration is detailed in Figure 3.

$$
\begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = \lambda \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} \tag{4}
$$

$$
\lambda = f/Z_c \tag{5}
$$

In Cartesian coordinates, it is not possible to write this transformation as a linear mapping, however, when we convert to homogeneous coordinates, this can be done. We use a 3x4 canonical projection matrix to represent how 3D points are mapped to a 2D image.

The only requirements are that we know the location of the centre of projection and its alignment with respect

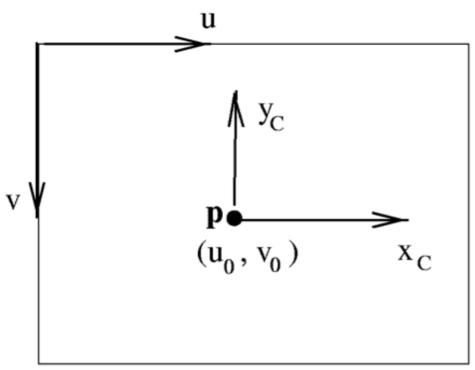


Figure 4: Origin Shift Diagram [6]

to the image plane, as well as the focal length. To change coordinate systems such that we express the image coordinates in the image coordinate system instead of the camera coordinate system, we introduce two new variables u and v and define the origin to be the top left hand corner of the image. The translation of the origin (see Figure 4) is shown in Equations 6 and 7. We must also multiply the coordinates by a factor, k, that allows us to map between units of pixels and length in millimetres, for example.

$$k_u x_c = u - u_0 \tag{6}$$

$$k_v y_c = v_0 - v \tag{7}$$

We can then use the camera calibration matrix, K (see Equation 8) to therefore map between the camera and image coordinate systems, shown in Equation 9.

$$K = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

Where $\alpha_u = fk_u$, $\alpha_v = -fk_v$.

$$x_i = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & -fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} = K \begin{bmatrix} x_c \\ y_c \\ f \end{bmatrix} \tag{9}$$

Where u_0 and v_0 are the coordinates that define the principal point. We use 2 scale factors for the x and y coordinate as some camera sensors do not have square pixels, so we need to account for different scales along both axes. There are 4 parameters to consider. K can be defined in terms of α, which is a factor that gives us the scale for our mapping between pixels and millimetres. If the sensor does not have square pixels, the aspect ratio will not be 1 as the scale in both the x and y axes won't be equal, but this typically isn't the case for most cameras, which will have an aspect ratio of 1 [6].

Next, we consider the fact that we have no knowledge of the camera's pose. We will now discuss the extrinsic camera parameters and express our calculations in the world coordinate frame. We require a transformation that allows this (see Equation 10).

$$X_c = RX_w + t \tag{10}$$

This transformation is rigid, so we must consider both a translation, t and a rotation, R, as shown in Figure 5, where Equation 11 holds.

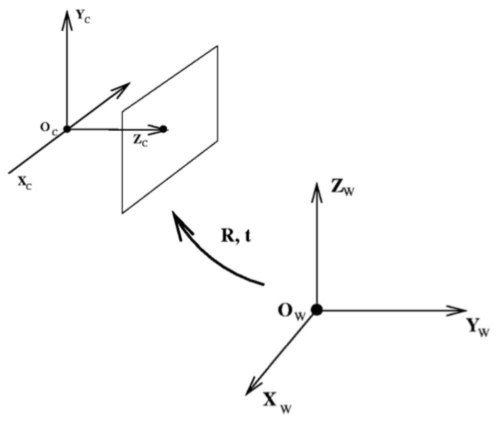


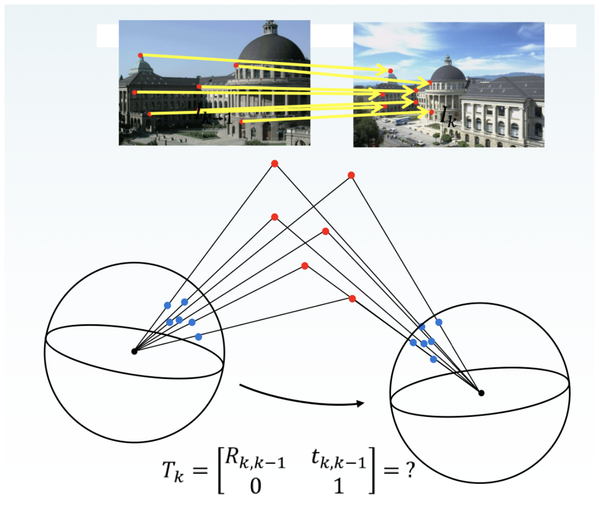
Figure 5: Rotation and Translation [6]

$$\begin{bmatrix} 1 \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} 1 \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{11}$$

We pre-multiply these matrices to first apply the rigid transformation to the 3D world points to express them in the camera coordinate frame, then apply the canonical projection matrix to get the image point in the camera coordinate frame, then finally premultiply by the camera calibration matrix to get the pixel coordinates.

$$x = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{12}$$

The problem we are currently considering is detailed below in Figure 6. We have features in both images and we want to estimate how much the camera has moved in between capturing these two images. The corresponding feature points are the projection of the same 3D point in space since we assume that only the camera has moved and the majority of the scene is static.



T_k
$T_k = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix} = ?$

Figure 6: Feature Matching [6]

Considering epipolar geometry, we can think about the projection of a 3D world point on both the previous and current images. A translation vector connects the two centres of projection and we also observe vectors that connect the centres of projection to the world point to form a plane (see Figure 7), hence, these vectors are coplanar. Now, we express one of the points in the coordinate frame of the other camera location, by applying the rotation and translation. The essential matrix (Equation 13) allows us to express this coplanarity constraint as defined by the epipolar constraint, and we can decompose this to find a rotation matrix and translation vector. Using linear methods and some point correspondences, the elements of the essential matrix (i.e. how much the camera moved in between the two frames in consideration), can be obtained.

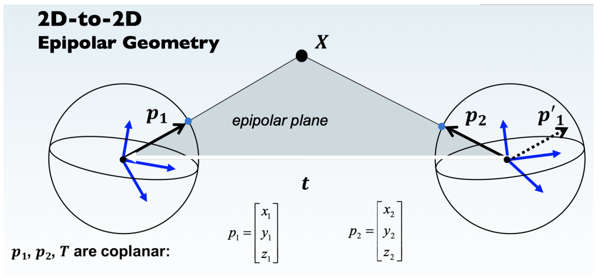

2D-to-2D
Epipolar Geometry
X
p1
epipolar plane
p2
p'1
t
$p_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$
$p_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$
p1, p2, T are coplanar:

Figure 7: Epipolar Geometry diagram [6]

$$E = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} = \begin{bmatrix} e_{11} \\ ... \\ e_{33} \end{bmatrix} \tag{13}$$

To calculate the essential matrix, we can use the 8-point algorithm [14] and use Singular Value Decomposition (SVD) to solve the epipolar constraint equation.

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} f_1 & f_2 & f_3 \\ f_4 & f_5 & f_6 \\ f_7 & f_8 & f_9 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0 \tag{14}$$

We have assumed knowledge of the camera calibration here, but if we haven't calibrated the camera, we'd need to apply K to both sets of image points and establish the fundamental matrix (see Equation 15), which allows us to express the coordinates of both image points in pixel units.

To find the entries of the fundamental matrix, we consider again the epipolar constraint Equation 15, which is satisfied for every pair of corresponding points. As we are estimating the 9 elements of this matrix and we have an equation in terms of homogenous coordinates, we only need 8 corresponding points to solve for all the values in this matrix. A system of linear equations can be constructed and solved by combining the measurement matrix and the fundamental matrix elements.

$$\begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} F \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = 0 \tag{15}$$

Computing correspondences is not an automatically robust process — there is a chance that the algorithm does not produce perfect correspondences. Similarities between feature points can introduce outliers. Robust methods such as RANSAC can be used to eliminate these outliers that arise in the feature matching process. RANSAC allows us to estimate the relative motion that's suggested by observing a small subset of correspondences and see how many of the remaining points agree with this suggested motion.

1. Select the number of points needed to determine the model's parameters at random

2. Estimate the parameters of the model using these points

3. See how many data points agree with this estimated model according to your given margin of error by calculating the squared distance between the point to the line

4. Repeat the sampling process if the ratio of inliers to outliers does not exceed a predefined threshold, else re-estimate the model parameters using all of the points that have been identified as inliers.

Parallel to the computation of the fundamental matrix under the assumption of a non-planar scene is that of a homography, where we assume a planar scene (see Equation 16).

$$\begin{aligned} \mathbf{x}_c^T \mathbf{F}_{cr} \mathbf{x}_r &= 0 \\ \text{where } \mathbf{x}_c &= \mathbf{H}_{cr} \mathbf{x}_r \end{aligned} \tag{16}$$

A score is computed for each model to see which one to use. The components of which are highlighted in Equations 17 and 18.

$$S_M = \sum_i (\rho_M(d_{cr}^2(\mathbf{x}_c^i, \mathbf{x}_r^i, M)) + \rho_M(d_{rc}^2(\mathbf{x}_c^i, \mathbf{x}_r^i, M))) \tag{17}$$

$$\rho_M(d^2) = \begin{cases} \Gamma - d^2 & \text{if } d^2 < T_M \\ 0 & \text{if } d^2 \geq T_M \end{cases} \tag{18}$$

Finally, depending on the planarity of the scene, a homography (for planar scenes) or a well constrained transition matrix (for a non-planar scene) is chosen to model camera pose.

$$R_H = \frac{S_H}{S_H + S_F} \tag{19}$$

If $R_H \leq 0.45$ then it was found that a homography should be chosen over the use of a fundamental matrix.

1.3 Track Local Map

Tracking the local map is localising the position of the camera given the current frame and previous frame, and estimated feature correspondences is required to happen in real time e.g. 30Hz for a 30fps camera. Once an estimation of the camera pose and an initial set of feature matches is obtained, the map can be projected on to the frame, and more map point correspondences can be searched for. Projecting to only a local map bounds the complexity and thus computational labour. There are 3 types of keyframes that are generated [5]; $\mathcal{K}_1$ shares mapping points with the current frame, $\mathcal{K}_2$ has neighbours to the keyframes $\mathcal{K}_1$ in the covisibility graph, and the reference keyframe $\mathcal{K}_{ref} \in \mathcal{K}_1$ which shares most map points with the current frame. Each map point observes in $\mathcal{K}_1, \mathcal{K}_2$ is searched for in the current frame in the steps as follows [5]:

1. The map point projection x in the current frame is computed, and discarded if it lays out of the image bounds.

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

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

4. Scale in the frame is computed via ratio $\frac{d}{d_{min}}$

5. Compare the descriptor of the map point to the unmatched ORB features in the frame.

1.4 New Keyframe Decision

The final step of the tracking thread of ORB-SLAM 2 is to decide if the current frame is a new keyframe. The new keyframe Decision step is performed by inserting keyframes often to make tracking more robust to rotations. The keyframes used for the subsequent mapping thread of ORB-SLAM 2 are determined by the criterion [11][5] below:

1. More than 20 frames have passed since the last global relocalisation

2. The local mapping is idle

3. Current frame tracks at least 50 points, more likely 200.

4. The current frame tracks more than 90% of the reference keyframe.

The first condition ensures a good relocalisation, the third condition ensures good tracking. If a keyframe is inserted when the local mapping is busy (as stated in condition 2), a signal is sent to stop the local Bundle Adjustment (BA) so that it can process as soon as possible the new keyframe.

1.4.1 Motion Only Bundle Adjustment

Motion only bundle adjustment is a non-linear method for refining structure and motion [6] and is used to minimise the error in placing each feature in its correct position (minimizing re-projection error), this optimises camera pose. Bundle adjustment (BA) is known to provide accurate estimates of camera localisations as well as a sparse geometrical reconstruction [12]. The re-projection error corresponds to the image distance between a projected point and a measured point, and is used to quantify how closely the estimate of a 3D point recreates the true projection [13]. The projection error cost function through bundle adjustment is shown in Equation 20. The keyframe decision is vital as computationally, only a small number of past frames can be processed [14].

$$E(\mathbf{P}, \mathbf{X}) = \sum_{i=1}^{m} \sum_{j=1}^{n} D(\mathbf{x_{ij}}, \mathbf{P_i X_j})^2 \tag{20}$$

These estimates from local bundle adjustment allow for real time Structure from Motion (SfM) The final estimate of the 3D world points can then be used in the mapping thread of ORB-SLAM 2 algorithm.

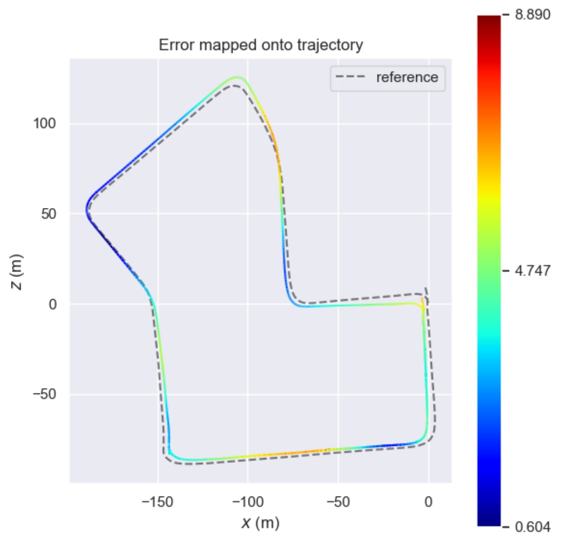
2 Evaluations

2.1 Base Case Implementation

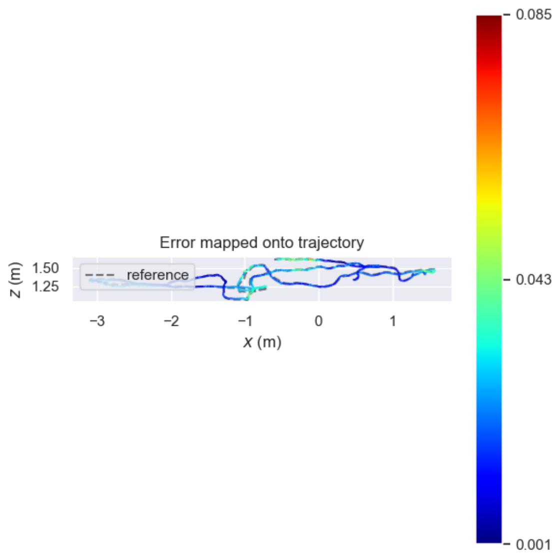
In this evaluation, we implemented ORB-SLAM2 on KITTI 07 sequence and TUM sequence separately with default setting applied. KITTI 07 sequence includes grey scale video frames taken from a car moving on the street for one complete loop. It covers an area of size of approximately $200m \times 200m$. In comparison, TUM data was shot by a handheld RGB-D camera moving slowly around an indoor office space of around $5m \times 1.5m$. The configuration file of `KITTI00-02.yaml` is used for KITTI 07 sequence while `TUM1.yaml` for the TUM sequence.

To implement ORB-SLAM 2, we firstly compiled the program with `./Build.sh` script. Then, executable files `mono_kitti` and `mono_tum` under the directory of `./install/bin` are called respectively indexing respective data packages, result files and and configurations files. This process saves the estimated trajectory in CSV text tile with TUM format where each pose is represented by frame number, 3-axis transitions and quaternions. To be consistent, KITTI 07 ground truth file was also converted to TUM format with the provided `kitti_to_tum.py` script.

The EVO tool was employed to plot the trajectory and absolute pose errors (APE) with regards to transition components, a.k.a absolute trajectory errors (ATE). Since all data were collected with monocular cameras, a 7-DOF transformation is required covering the scale in addition to the pose. Hence, when carrying out the operation, a parameter for aligning and auto scaling, '-as', was added to the command. On my machine, it was also required to select MacOsX as the Matplotlib backend to create the plot successfully.

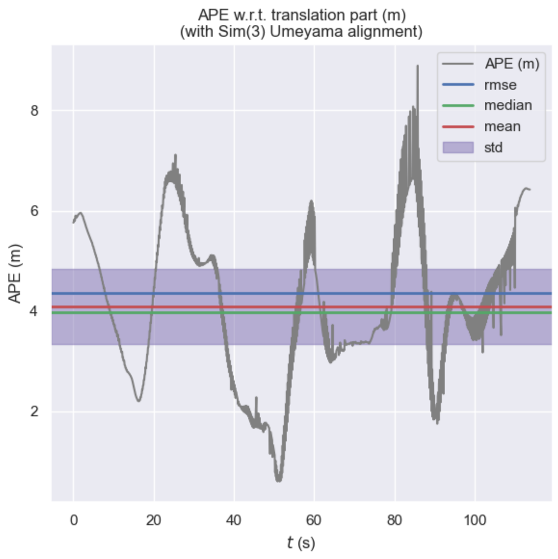


(a) KITTI7 Sequence

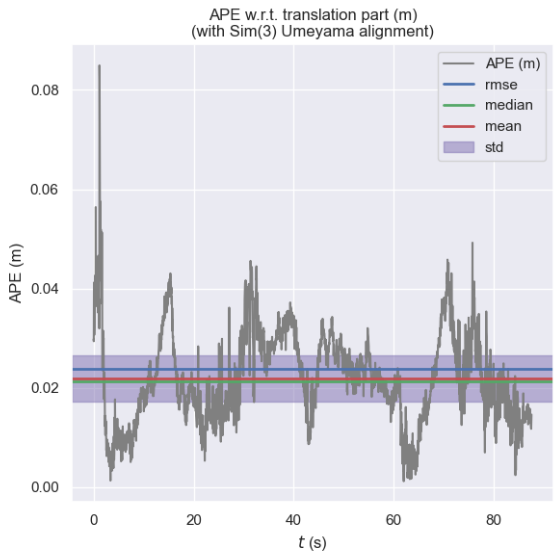


(b) TUM Sequence

Figure 8: Trajectory Graphs with Mapped Errors



(a) KITTI7 Sequence



(b) TUM Sequence

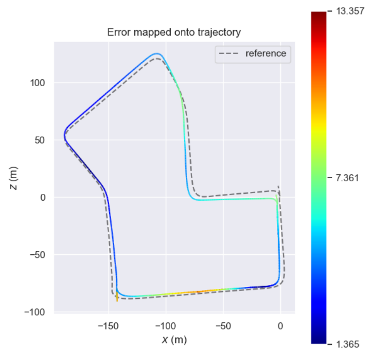
Figure 9: Absolute Trajectory Error

At first glance, both sequences performed ORB-SLAM well. On a closer look, however, the average error of KITTI07, is about 200 times that of TUM sequences being 4m compared to 0.02m. This behaviour is closely related to the two data sequences. KITTI07 was obtained on open street which is an uncontrolled environment. Moving vehicles, pedestrians, leaves and changing brightness can all affect feature extraction. In comparison, TUM was obtained with static observing objects. Furthermore, KITTI07 was captured in grey-scale with 10FPS from a moving car, while TUM in RGB-D with 30FPS. The camera for TUM sequence also moved in a deliberately prolonged pace. ORB-SLAM extract features from consecutive image frames. The more confined and controlled environment, slow moving speed and high FPS TUM sequence offered minimises the difference between each image and thus enhanced the accuracy. TUM sequence also contains depth information which makes it a 3D-to-2D problem. Compared to 2D-to-2D, like with the KITTI07 data, which required a minimum of 5-point correspondence to solve, 3D-to-2D only needs 3.

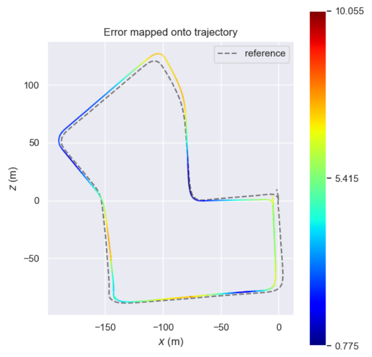
The errors in both sequences can be reduced by fine-tuning the camera calibration and distortion parameters in the yaml configuration files. However, this should not affect the observation of the behavioural changes for evaluations in the later sections.

2.2 Reduced ORB Features

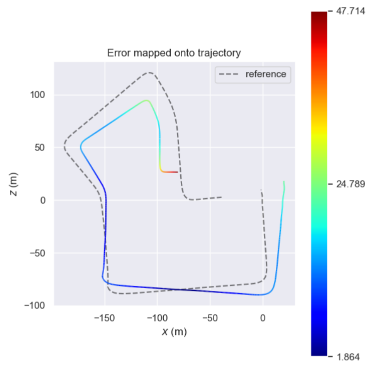
For this evaluation, we reduced the number of features detected in each frame by modifying the variable `nFeatures` in the `ORBextractor` class from the configuration files (`./Install/etc/orbslam2/Monocular/KITTI00-02.yaml` and `TUM1.yaml`). We picked 3 different levels of feature numbers with 10%, 25% and 40% reduction respectively from the base case (2000 features for KITTI sequence and 1000 for TUM). From experiments, it can be found that further feature reduction can cause failure in the camera tracking thread for KITTI 07 sequence.



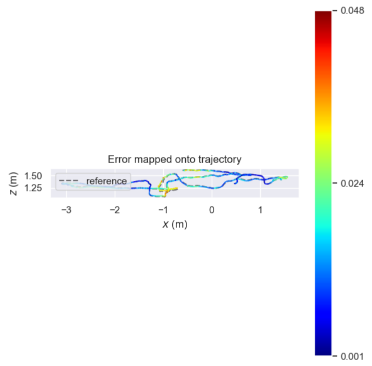
(a) KITTI07 1800 Features



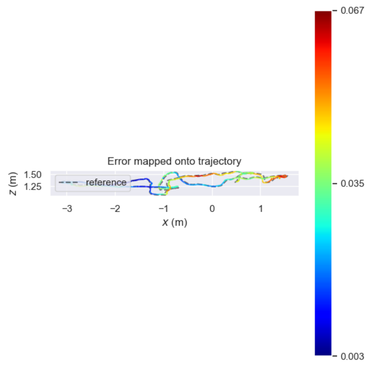
(b) KITTI07 1500 Features



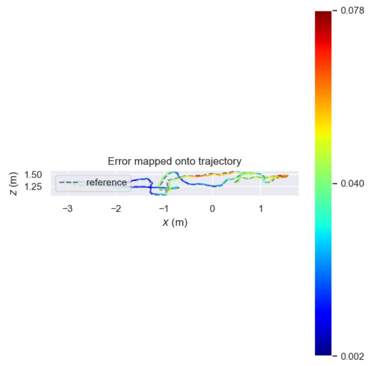
(c) KITTI07 1200 Features



(d) TUM 900 Features

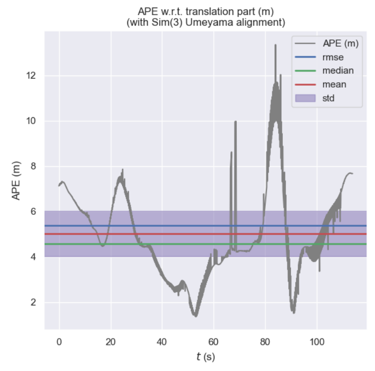


(e) TUM 750 Features

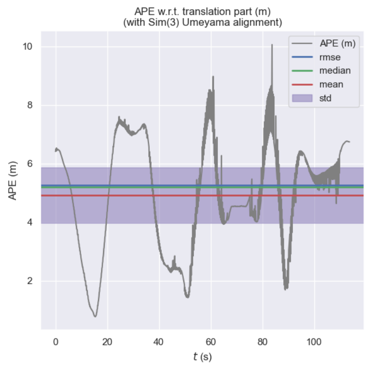


(f) TUM 1400 Features

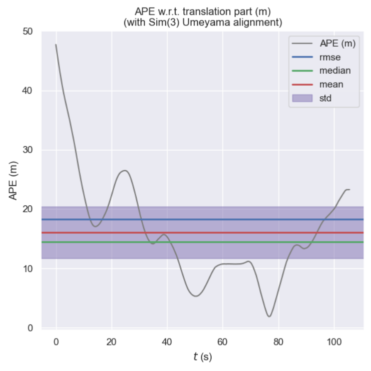
Figure 10: Trajectory Graphs with Reduced ORB Features



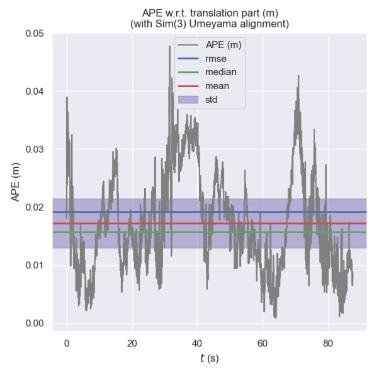
(a) KITTI07 1800 Features



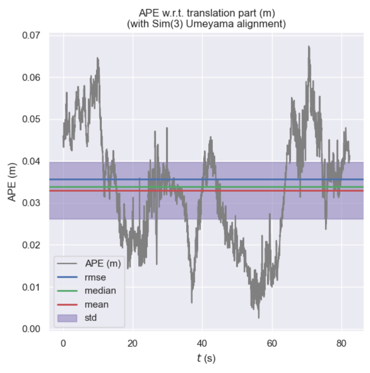
(b) KITTI07 1500 Features



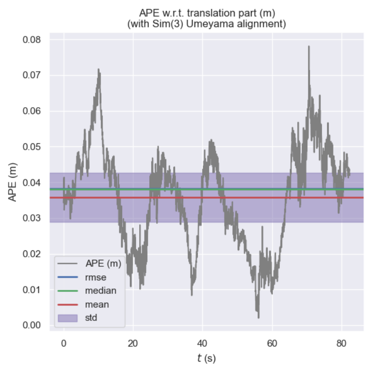
(c) KITTI07 1200 Features



(d) TUM 900 Features



(e) TUM 750 Features



(f) TUM 600 Features

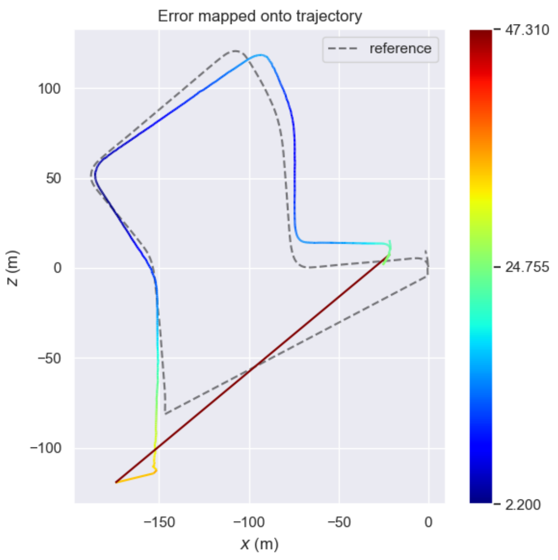
Figure 11: Absolute Trajectory Error with Reduced ORB Features

We can easily observe a general pattern where the error increases as the number of detected features reduces. This is natural as the principle of the algorithm is to match features to perform mapping. As more features ignored, the possibility of a match not found or pairing up the different features from two consecutive frames grows.

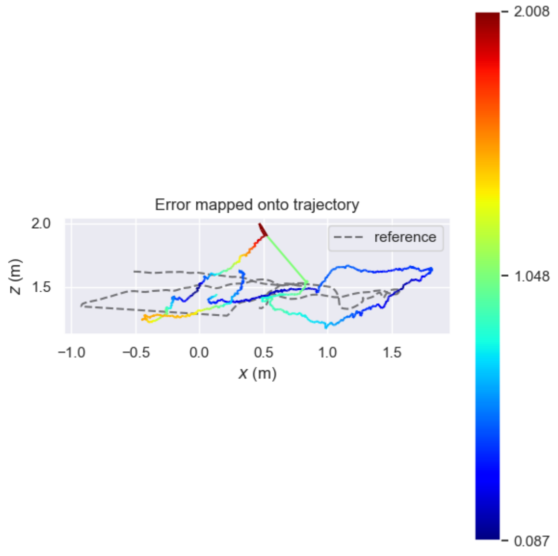
It is noticeable that the overall performance of the TUM sequence is better than KITTI07 as features detected reduced. While the mean ATE almost doubled for TUM between the base case and 1500 feature case, the absolute increase is about $0.02m$. In comparison, the mean ATE for KITTI07 increased by about $1m$. It also failed to complete loop closure with 1200 features. At 40% feature reduction, it appears that KITTI07 took about $20s$ to initialise, i.e. extracting the first batch of features. Thus about the first $40m$ of distance was uncovered and unmapped. As a result, there existed no matching features for loop closure to be implemented upon reaching the end of the trip. This is shown furthermore by the trend of the root mean squared error (RMSE) increasing inversely proportionally to the number of features, in both the KITTI07 and TUM datasets.

2.3 Removal of Outlier Rejection Process

To include the outlier in the optimisation process in the monocular mode, lines 386-387 were altered in `Source/Libraries/ORB SLAM2/src/Optimizer.cc`. The values of `mvbOutlier` were set to false. Counting for outliers, i.e. `nBad` variable, was suspended. The level settings with the `setLevel` function were changed to 0, which allows g2o library to include all outliers during optimisation.

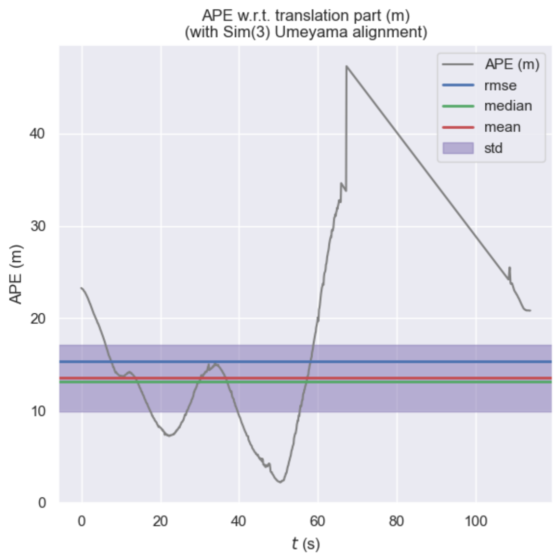


(a) KITTI7 Sequence

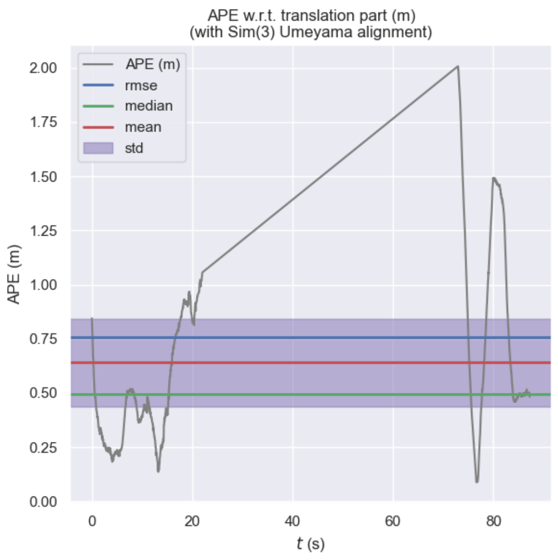


(b) TUM Sequence

Figure 12: Trajectory Graphs without Outlier Rejection Process



(a) KITTI7 Sequence



(b) TUM Sequence

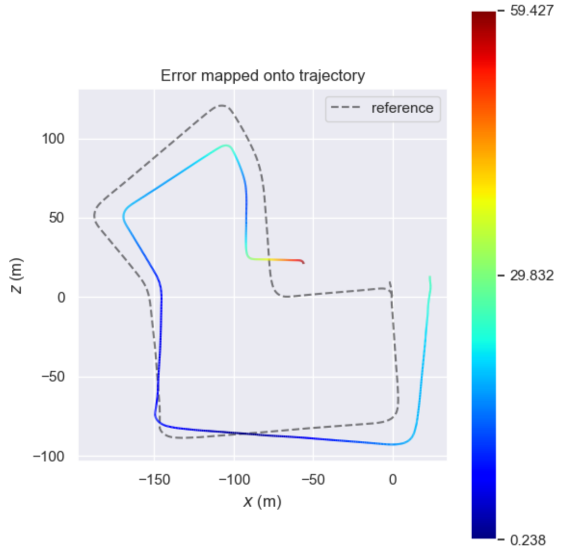
Figure 13: Absolute Trajectory Error without Outlier Rejection Process

In our case, an edge is considered as an outlier when its chi-squared value is larger than the pre-defined value of 5.991. Outliers are usually disregarded to keep the mapping consistent with the tracking. When removed, the errors increase dramatically. For the two sequences, the initial errors increase for around 4× and 9× respectively compared to the base case.

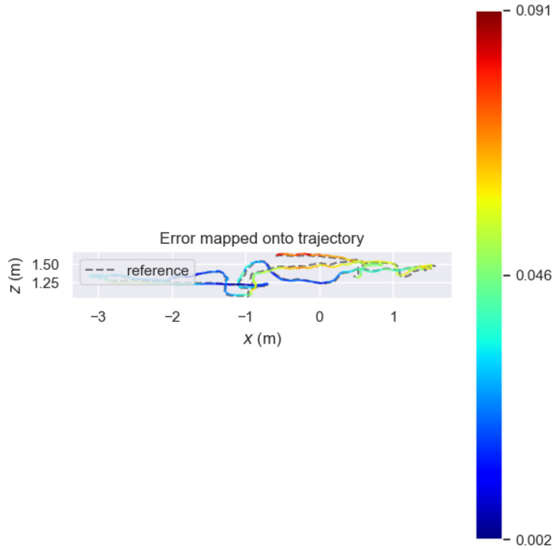
For the first $\frac{1}{3}$ section of the duration, both error trends appear to be decreasing as more features were being extracted before significantly deviating from the ground truth and losing tracking completely. This explains the straight line in both error and trajectory graphs. From the point of losing tracking to the starting point where the same features were detected again in the end, the loop closure algorithm simply connected the 2 points with a straight line. This is also reflected on the ground truth trajectory where available data for untracked timestamps were simply skipped. Consequently, the error plot of the particular period follows a linear trend.

2.4 Deactivating Loop Closing

Under the directory of `Source/Libraries/ORB SLAM2/src/`, line 1388-1390 in `Tracking.cc`, line 80 in `LocalMapping.cc` and line 146-148, 159, 162-165, 338, 346 in System.cc were altered by removing relevant execution commands of functions within `LoopClosing.cc` to deactivate loop losing. Other methods, including removing content in the `run` function of `LoopClosing.cc` and clearing loop closing candidate vector `vpCandidateKFs`, were also tried but without success.

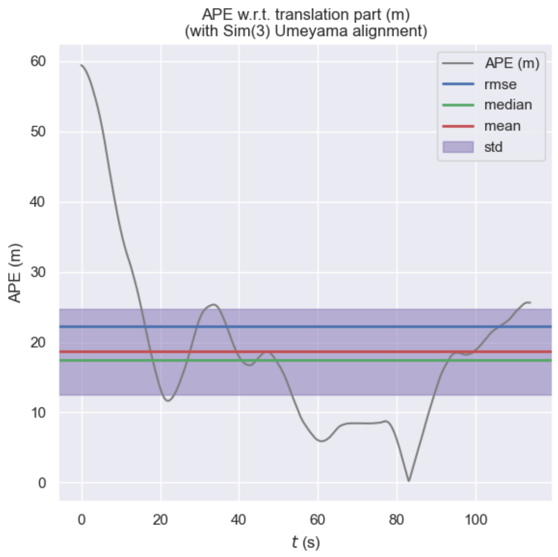


(a) KITTI7 Sequence

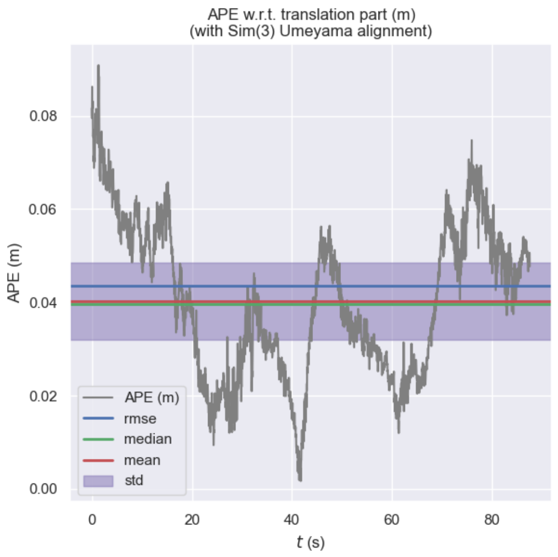


(b) TUM Sequence

Figure 14: Trajectory Graphs without Loop Closing



(a) KITTI7 Sequence



(b) TUM Sequence

Figure 15: Absolute Trajectory Error without Loop Closing

Without loop closing, the accuracy of both sequences was affected. The mean ATE of KITTI07 sequence dramatically increases from roughly 4*m* in the base case to 19m, while mean ATE of TUM sequences increases by only about $0.02m$. The behaviour further proves the discussion in section 2.1 where high FPS, slow camera movement, the capture of depth data and static environment can significantly reduce the error in a visual SLAM system. It also demonstrates that loop closure effectively enhances the SLAM system accuracy when the same feature was extracted more than once, especially in an uncontrolled environment.

3 Conclusion

When the KITTI07 and TUM RGB-D datasets are evaluated on the ORB-SLAM2 algorithm, we can observe the effects on the performance of the quality of camera tracking by altering parameters such as ORB features used by the system, removing outlier rejection, and removing loop closure; as performed in section 2. The base case of standard parameters and a normally functioning ORB-SLAM 2 was evaluated in section 2.1, for use as a comparison. Each case was evaluated using ATE and APE plots, to quantify and visualise the performance. It is concluded that the reduction in ORB features used by the system impacts the feature tracking greatly, as the error observed increases as the number of features used decreases, this is shown in the evaluations in Fig. 11. Although the TUM dataset was found to be slightly more robust to the reduction in ORB features when evaluated with ORB-SLAM 2, as the absolute increase in error was but $0.02m$, where as the absolute increase in error was $1m$ in the KITTI07 dataset for a proportional decrease in features.

The inclusion of outliers was also explored in section 2.3, this was performed by removing the outlier rejection process in ORB-SLAM 2 in monocular mode. Outliers are usually identified and removed when its chi-squared value is larger than a predetermined threshold, in our case 5.991. The lack of removal of values with a high chi-squared value means there are edges with a great difference between the observed and expected values. Thus, the uncertainty increases and the errors increased greatly, as observed in Figure 13. The errors increased $4\times$ and $9\times$ in magnitude for the KITTI07 and TUM datasets respectively, compared to the base case performed in section 2.1.

It was also concluded that the absence of loop closure negatively affects the performance of the ORB-SLAM 2 tracking process, as the gradual increase in error is never diminished as it would be in usual loop closure. Deactivating loop closure greatly reduced the accuracy of the system, as shown in Figure 15, in both the KITTI07 and TUM datasets, as the extraction of the same feature multiple times would have allowed for the correction of errors that have accumulated over time and distance travelled.

References

[1] Sjulier, "Sjulier/refactored orbslam2." [Online]. Available: https://github.com/sjulier/Refactored_ORB_SLAM2

[2] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[3] "Informatik ix chair of computer vision &amp; artificial intelligence," Mar 2016. [Online]. Available: https://vision.in.tum.de/data/datasets/rgbd-dataset

[4] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[5] 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.

[6] L. Agapito, "Comp0130 lecture 2," 2022.

[7] OpenCV, "Brief (binary robust independent elementary features)," https://docs.opencv.org/3.1.0/dc/d7d/tutorial_py_brief.html, 2015.

[8] D. Tyagi, "Introduction to fast (features from accelerated segment test)," Apr 2020. [Online]. Available: https://medium.com/data-breach/introduction-to-fast-features-from-accelerated-segment-test-4ed33dde6d65

[9] ——, "Introduction to brief(binary robust independent elementary features)," Apr 2020. [Online]. Available: https://medium.com/data-breach/introduction-to-brief-binary-robust-independent-elementary-features-436f4a31a0e6

[10] "Tracking thread of orb-slam2." [Online]. Available: https://blog.katastros.com/a?ID=01050-81e9c5e2-b469-4be0-8068-802b29cf3350

[11] "University of oslo, lecture 10.3 orb-slam." [Online]. Available: https://www.uio.no/studier/emner/matnat/its/TEK5030/v20/forelesninger/

[12] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *International workshop on vision algorithms*. Springer, 1999, pp. 298–372.

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

[14] H. Strasdat, J. Montiel, and A. J. Davison, "Visual slam: Why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885612000248

Appendices

A 2.1 Commands

```
$ cd MSc_RC_Repo/Refactored_ORB_SLAM2
$ mono_kitti KITTI00-02.yaml data/KITTI_Sequences/07 Results/2.1_kitti07.txt
$ python Evaluation/kitti_to_tum.py data/KITTI_Sequences/07/07.txt data/KITTI_Sequences/07/
    times.txt data KITTI_Sequences/07/07_tum.txt
$ evo_ape tum data/KITTI_Sequences/07/07_tum.txt  Results/2.1_kitti07.txt -as --plot --
    plot_mode xz
```

Listing 1: 2.1 Commands

B 2.2 Commands

```
$ mono_kitti KITTI00-02_1200.yaml data/KITTI_Sequences/07 Results/2.2_kitti07_1200.txt
$ evo_ape tum data/KITTI_Sequences/07/07_tum.txt  Results/2.2_kitti07_1200.txt -as --plot --
    plot_mode xz

$ mono_kitti KITTI00-02_1200.yaml data/KITTI_Sequences/07 Results/2.2_kitti07_1500.txt
$ evo_ape tum data/KITTI_Sequences/07/07_tum.txt  Results/2.2_kitti07_1500.txt -as --plot --
    plot_mode xz

$ mono_kitti KITTI00-02_1800.yaml data/KITTI_Sequences/07 Results/2.2_kitti07_1800.txt
% evo_ape tum data/KITTI_Sequences/07/07_tum.txt  Results/2.2_kitti07_1800.txt -as --plot --
    plot_mode xz

$ mono_tum TUM1_600.yaml data/TUM_Sequence Results/2.2_tum_600.txt
$ evo_ape tum data/Tum_Sequence/groundtruth.txt  Results/2.2_tum_600.txt -as --plot --
    plot_mode xz

$ mono_tum TUM1_750.yaml data/TUM_Sequence Results/2.2_tum_750.txt
$ evo_ape tum data/Tum_Sequence/groundtruth.txt  Results/2.2_tum_750.txt -as --plot --
    plot_mode xz

$ mono_tum TUM1_900.yaml data/TUM_Sequence Results/2.2_tum_900.txt
$ evo_ape tum data/Tum_Sequence/groundtruth.txt  Results/2.2_tum_900.txt -as --plot --
    plot_mode xz
```

Listing 2: 2.2 Commands

C 2.3 Commands

```
$ mono_kitti KITTI00-02.yaml data/KITTI_Sequences/07 Results/2.3_kitti07.txt
$ evo_ape tum data/KITTI_Sequences/07/07_tum.txt  Results/2.3_kitti07.txt -as --plot --
    plot_mode xz
$ mono_tum TUM1.yaml data/TUM_Sequence Results/2.3_tum.txt
$ evo_ape tum data/Tum_Sequence/groundtruth.txt  Results/2.3_tum.txt -as --plot --plot_mode
    xz
```

Listing 3: 2.3 Commands

D 2.4 Commands

```
$ mono_kitti KITTI00-02.yaml data/KITTI_Sequences/07 Results/2.4_kitti07.txt
$ evo_ape tum data/KITTI_Sequences/07/07_tum.txt  Results/2.4_kitti07.txt -as --plot --
    plot_mode xz
$ mono_tum TUM1.yaml data/TUM_Sequence Results/2.4_tum.txt
$ evo_ape tum data/Tum_Sequence/groundtruth.txt  Results/2.4_tum.txt -as --plot --plot_mode
    xz
```

Listing 4: 2.4 Commands