Benchmarking study on performance between Oriented FAST and rotated BRIEF SLAM (ORB-SLAM) and OpenVSLAM

Ramprasad Rajagopalan
MS in Mechanical Engineering
rrajagopalan1@mines.edu

Bhuvan Tej Kanigiri
MS in Mechanical Engineering
bhuvantejkanigiri@mines.edu

Abstract

Visual SLAM (vSLAM) system is widely used in robotics for localizing the robot and mapping the environment around it in order to make a determined decision for desired trajectories. This project addresses the performance of feature-based vSLAM algorithms ORB-SLAM2 and OpenVSLAM on different generic datasets viz., TUM RGB-D, EuRoC and ETH3D, by comparing them based on Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) for different image sequences and lighting conditions. Loss in tracking, tracking times and efficiency of relocalization is also evaluated. Further, the vSLAM algorithms are evaluated on mobile robot, COZMO, by navigating it in a custom-built maze environment under different lighting conditions.

1. Introduction

Localization and Mapping in an unknown environment contains large image sequences with rolling shutter distortion, pure rotations, scale ambiguity, rapid motions and occlusions that can largely deteriorate the performance of vSLAM algorithms [1]. The requirement to check and evaluate the performance of vSLAM algorithm in all such setting is crucial for real-time sensitive application such as autonomous driving. This can be done by rigorous evaluations on different image sequences from generic datasets. Various methodologies for benchmarking have been proposed in literature for visualizing the accuracy of each algorithm.

Feature based algorithms like ORB-SLAM2 [2] and recently developed OpenVSLAM [3] are widely used methods with state-of-the-art tracking accuracy in real-time implementations. OpenVSLAM pushes the boundaries of vSLAM algorithm by supporting various types of cameras thereby making it highly extendable in AR application and autonomous navigation of robots.

The performance of these algorithms have already been tested on generic datasets for translational RMSE or ATE and tracking times of each thread has been calculated. There were image sequences for both the algorithms wherein the RMSE was large, the tracking was lost, relocalization problem occured or the algorithm couldn't detect loop closure for a variety of reasons. This brings to attention the peculiarity of those specific image sequences and their reasons for making the performance of the algorithm to deteriorate. Albeit various extensions to these algorithm seems to be present in the literature, they are largely focused on individual applications. The training process with all such datasets might be relevant for simulating different lighting, motion sequences and occlusions but fails to recreate a real-time condition.

This project is an effort to bring to light the cause of performance deterioration and hardship in real-time implementation using different approaches. One approach would be to evaluate these algorithms on new set of image sequences from generic dataset for specific conditions. Another approach would be to implement the vSLAM algorithm on a robot and capture the map and trajectory in real-time. Both of these approaches are implemented in this project and performance of ORB-SLAM2 and OpenVSLAM algorithms are evaluated.

2. Previous work

2.1. vSLAM algorithms

Indirect based: This method estimates the pose(6 DOF) & map using feature descriptors & detectors. present in the image. It uses geometric consistency for getting the position of feature points. Methods such as SIFT,SURF are used for getting the correspondences. Algorithms that are considered good for this method are -MonoSLAM [4], PTAM [5], ORB-SLAM . Generally this method is implemented in indoor & outdoor environments. Map density in this method is sparse.

MonoSLAM: This algorithm the camera motion and Map of uncertain environments are estimated using extended Kalman filter(EKF), Here the map initialization is done by a known item in environment.

PTAM: This algorithm is the starting point of using Multi-threads, that does tracking and mapping simultaneously which in-turn reduces the computation cost to a great extent.

Direct method: This method estimates the pose & map using Image Intensities and uses photometric consistency, for error correction. It is basically a featureless approach of SLAM. Algorithms that are considered good for this method are - DTAM [6], LSD-SLAM [7] and SVO [8] - DSO [9]. This method can be used for indoor & outdoor environments. Map density in this method is semi-dense.

DTAM: This algorithm uses stereo and takes the advantage of compare the ram input image sequences with synthetic view of images which are generated from the reconstruction map.

LSD-SLAM This is the state-of-the-art direct method where random values are set as an initial depth value for every pixel and uses synthetic view data for estimating camera motion. Photometric consistency is a metric to say whether a voxel is occupied or not. If it has the same color in the voxel from different cameras,then we can say that the voxel is photo-consistent.

RGBD method: Generally this method is considered as a direct RGB-D method, this method tries to correct pose by taking odometry data, which includes the geometric error term and accomplishes frame to keyframe matching. When there is a new keyframe coming in , it is inserted to the pose graph module.

Here we use the ICP algorithm for getting the depth information, and after reconstructing different depth maps, we color them using algorithms like kinectFusion [10] & SLAM++ [11]. This method has limitations of using it in outdoor environments. Map density in this method is dense.

2.1.1 ORB-SLAM

ORB-SLAM is one of the most widely used feature based vSLAM algorithms that can operate in real-time within large indoor-outdoor environment. The algorithm was initially implemented for monocular cameras [12] and extended to RGB-D and stereo camera [2]. The ORB-SLAM algorithm works similarly to PTAM with robust evolutionary strategy leading to excellent tracking performance, compact local mapping and lifelong implementation.

The algorithm has three thread running in parallel viz., tracking, local mapping and loop closing apart from place recognition and map module. The **tracking thread** performs the localization of the camera for every frame and decides whether the frame is a keyframe. It performs motion-only bundle adjustment (BA) for optimizing the pose. The thread also performs relocalization using bag-of-words representation, DBoW2 algorithm, inside the place recognition module. The ORB features are extracted by picking FAST corners at 8 scale-levels and a preassigned scale factor for a given number of features. The algorithm has an automatic map initialization techniques that picks Homography or Fundamental matrix based on survival of the fittest approach in obtaining the transformation. Using the initial camera pose estimates and feature matches, a local map is tracked with a set of keyframes sharing the map point and neighbouring keyframes in the covisiblity graph. The camera pose is optimized using the tracked local map and robust approach for creating a new keyframe is performed.

The **local mapping thread** takes the new keyframe and inserts it to the covisibility graph and spanning tree. Then it performs robust culling procedure to remove redundant map points as well as create new map points using triangulation. Local BA is called to optimize the keyframe camera pose and map points for the given keyframe. This procedure also takes into account the keyframes that sees current map point but doesn't include them in optimization. The thread takes care of discarding redundant keyframes for lifelong operation and compact map reconstruction.

The **loop closing thread** takes the current keyframe, looks for loop candidates by comparing bag of words vector and keeps the score for all the loop candidates. The candidates whose score is less than certain minimum value are discarded and similarity transformation is performed to find more map correspondences. Loop fusion occurs by fusing the duplicated loop points and inserting edges in the covisibility graph. After all the keyframes updates their edges in the attaching loop closing edge, global BA or essential pose graph, a sparse sub-graph of covisibility graph, optimization is done to correct the accumulated scale drift.

The thread computation times are accelerated by the use of ORB features and essential graph representation. As the same features are used in each thread, the algorithms performance and realibility increases. Following section illustrates another robust algorithm similar to ORB-SLAM.

2.1.2 OpenVSLAM

Scene Modeling The two well-known technquies are Structure from Motion(Sfm), which is general used if we already have a pre-built image frame sequences. This is used for non-real time applications. The other method is Visual SLAM.

Most of the state-of-the art Visual slam algorithms are performs exceptional and are considered as de facto standards of VSLAM algorithms. However using it to our application creates alot of errors in-terms of version restriction in scripts. However using docker would work, but again fitting this our application would be tedious process.

OpenVSLAM is one that algorithm which removes this barriers of using VSLAM to there application. The algo-

rithm takes different kinds of camera models such as fish-eye, equirectangular , RGBD. typical stereo and monocular. On top of that user can have flexibility to add customized optional camera models by specifying the characteristics in config file.

This algorithm has a option to store and load a map database, further the localization would be function of this prebuilt map.

Finally this algorithm is a cross-platform viewer running on web browsers, where the user has a flexibility to open in any browser remotely.

OpenVSLAM basically indirect method and uses FAST algorithm as key point. and Binary Vector as Descriptors method. The complete implementation is done in C++ language, Eigen for matrix calculation, opencv for I/O operation of images and final general graph optimization (g2o) for map optimization. The whole pipeline is divided into three major segments tracking module, mapping module and global optimization module.

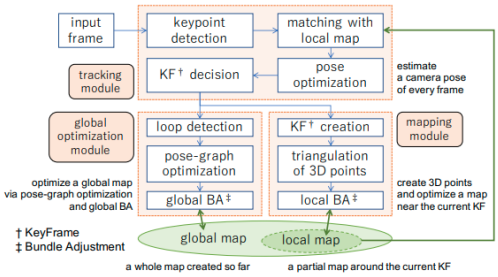

Figure 1. Main modules of OpenVSLAM: tracking, mapping, and global optimization modules [3]

Tracking module: In tracking module, estimates a sensor(camera) poses each frame that is inputted to openvslam via key point matching and pose optimization. The goal is to get correct pose (6DOF) at all the times, for we have to reduce the re-projection error: this process would come under bundle adjustment , which we use it heavily in all the modules.

Mapping module In Mapping Module, It uses the method of triangulation of 3D points using the inserted new keyframes, this process continues to occur hence the map would be extended over time, this module also does some sort of map optimization ,called local bundle adjustment.

Global optimization module. In this module we mainly have three segments happening, firstly Loop detection which always checks whether did I see this scene before my journey , if the answer is yes : then there is a loop closure happening at that point , the algorithm does global optimization of pose and global bundle adjustment. If No it as usually creates a new key frames and again checks for loop detection.

2.2. Benchmarking techniques

"Visual SLAM (vSLAM) systems suffers openly from pure rotation, initialization of the map, rolling shutter distortion and scale ambiguity [1]. Benchmarking is evidently the requirement for comparing the performance of different vSLAM systems. Various methodologies with benchmarking using generic datasets have been implemented and evaluated for vSLAM systems. Each of the dataset uses a specific description of the image, formatting and evaluation tools for benchmarking. In order to understand the effect of the dataset on tracking accuracy and performance of vSLAM systems for one's own environment, custom generated datasets are required. The following enlists and discusses some of the specifics and methodologies used by state-of-art generic datasets for benchmarking vSLAM systems.

Sturm et al. [13] presents a novel benchmarking criterion for evaluating the performance of RGB-D vSLAM systems. The TUM RGB-D dataset captures 39 sets of image sequence using Microsoft Kinect at 30Hz with 640 x 480 image resolution. The image sequences are obtained for indoor environments like office setting and industrial hall. The sequences has slow motions, rotations and longer trajectories with and without loop closures for debugging and comparison of different approaches in a cluttered indoor environment. The image sequences were recorded for handheld motion as well as for robot ground locomotion mounted with the sensor. The ground truth trajectory estimates with 6DoF are obtained by high resolution motion capture cameras located inside the indoor environment. The evaluation tools for calculating accumulated scale drift and global pose error is proposed to be invariant of the vSLAM system. The time synchronization between the data from motion capture systems estimating 6DoF pose of the camera and color data with depth maps from Kinect are not done as the time stamp delay is around 20ms. This dataset doesn't consider rolling shutter distortion and variation in scene illumination due to automatic setting of the exposure time by commodity camera. The work proposes a novel metric for evaluating the performance of vSLAM systems based on relative pose error (RPE) and absolute trajectory error (ATE). The RPE correlates with accumulated drift and ATE correlates with global consistency of the estimated trajectory. The proposed high-quality dataset provides meticulous validation of vSLAM systems with given metrics.

The methodology illustrated above applies only for indoor setting with variation to exposure time and suffers from time synchronization of the sensor information. The work done by Geiger et al. [14], KITTI, establishes 389 stereo and optical flow image pair with odometry sequence for trajectory length of 39.2 km in a cluttered environment with more than 200k annotated 3D objects. The datasets are recorded using 4 stereo cameras with 1392 x 512 image

resolution and a Velodyne 3D laser scanner with 64 laser beam and 100m range mounted on an autonomous car. The image sequences are captured by driving around a mid-size city, suburban areas and freeways. The ground truth localization is recorded with the help of GPS, GLONASS, IMU and RTK correction signals. Unlike the previous approach, the cameras, laser scanners and GPS systems are calibrated and synchronized for providing exact ground truth defined by the time stamps. The metric proposed for evaluating the vSLAM approaches are individual error function of rotation and translation in terms of velocity and trajectory length. The error is expressed in terms of estimated and true poses of the stereo cameras. The stereo matching performance in the natural scene setting is better than inner-city settings due to the absence of disparity occlusions and largely textured environment. The dataset seems to provide maximum information than other state-of-the-art benchmarking technique.

It is evident from the above-mentioned methodologies that in a real-world dataset it is challenging to obtain the required environment/scene condition, camera setting and parameter, absence of occlusions, error in alignment and estimation of calibration parameters and arduous work in retrieving ground truth estimate. This led to the research on building synthetic datasets with photorealistic image information using computer graphics (CG). Earlier works of Martull et al. [15] focused on developing the dataset for stereo matching and camera tracking evaluation using realistic CG rendering techniques. The dataset utilized stereo pair images from University of Tsukuba to recreate a scene a 3D using Autodesk Maya software. Textured 3D models were created by designers with shaders and lighting to approximately mimic real-world behavior. The dataset has 1800 full-color stereo image pairs rendered under different illumination condition and 256 levels of ground truth disparity maps. A video sequence is also provided along with "non-occluded area masks and near depth discontinuity masks" apart from virtual camera's 3D rotational and translational information for each frame of the sequence. Although this work fulfilled the requirements with scene lighting and camera setting information, it lacked annotation, accurate depth information for each pixel, motion blur effects and other commodity camera critical for evaluating and validating vSLAM systems. Recent work of Wang et al. [16] provided synthetic dataset accompanied with exhaustive information required for testing, debugging and evaluating state-of-the-art vSLAM systems. The Tongji Computer Graphic (TCG) synthetic dataset comprises of 16k 1280 x 960 resolution photorealistic images sequence of an underground garage through motion of a virtual car for length of 1km. Fig.2 provides the pipeline for TCG dataset. This dataset possesses "6DoF camera pose for each frame, undistorted wide-angle and fish-eye images, bounding boxes for significant object and their relative transformation, depth for each pixel, semantic segmentation, edge map, intrinsic parameters, stereo image pairs and images with motion-blur and depth of effects". The plethora of accompanying information can be utilized for isolating the algorithms based on feature selection techniques, effects of communication between different modules in vSLAM systems. The study provides with numerous metrics for rating the vSLAM systems viz., reconstruction error (RE) and standard deviation (RE_STD) associated with 3D position of features in a keyframe, cruciality of features (CF) measuring information on static object features, significance of features (SF) in evaluating a vSLAM sampled feature in describing shapes of objects in environment and outlier removal capability (ORC) of vSLAM in removing the outlying features of moving objects. The ground truth information for depth is simulated with noise using real Kinect mechanism method. The garage lighting conditions are controlled along with high beam illumination of the vehicle. The dataset evaluated some state-of-the-art vSLAM systems like DSO and ORB-SLAM2 based on environmental conditions affecting camera tracking and mapping.

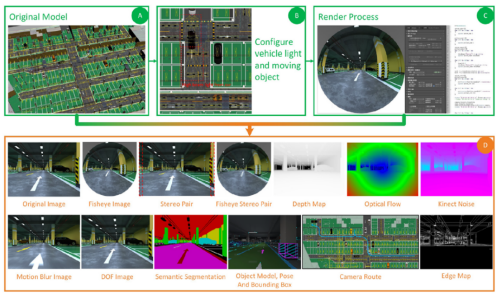

Figure 2. Pipeline of TCG synthetic dataset [16]

While synthetic dataset seems to be promising and effective benchmarking technique for vSLAM systems, it cannot be done by recreating every unknown environment and might not be robust in simulating every condition. It will be worthwhile to explore option of obtaining real-world dataset with easy or in the absence of ground truth estimates. Engel et al. [17] provided novel mathematical implementation of photometrically calibrated benchmark for vSLAM systems using monocular camera. The dataset comprised of fifty 100 minutes long real-world sequences calibrated photometrically for exposure times and non-parametric vignette recorded in indoor and outdoor conditions. The evaluation of vSLAM systems are studied for tracking accuracy without requiring ground truth information and analyzed the "effect of image resolution, camera field of view, and camera motion direction". The dataset comprises of exposure time for each frame and loop closure is achieved by having loopy camera motion at start and end of each sequence in front of textured objects with easy to

obtain features. The images of the sequences are reduced in resolution from raw 1028 x 1024 px to 640 x 480 px after photometric calibration and vignette. The evaluation of vSLAM systems like ORB-SLAM and DSO were done for root mean squared alignment error at start and end of each sequence. Other work like Rebert et al. [18] addresses methodology for obtaining ground truth data based on placing fiducial markers in an environment and estimating camera pose."

The exhaustive review on benchmarking techniques addresses some of the advantages and limitation for each generic datasets. The review also brings to light the open challenges faced by some vSLAM algorithms.

3. Technical approach

3.1. Dataset

3.1.1 Generic

The performance of ORB-SLAM2 and OpenVSLAM algorithms are done on three generic datasets viz., TUM RGB-D, EuRoC and ETH3D dataset. The ORB-SLAM2 fails to detects loops in sequences with few frames and performs poorly in highway sequences due to few closer points and high speed. The algorithm fails for image sequence having severe motion blur. Further, the algorithm performs poorly for miscalibrated image sequences.

In order to test the robustness of each algorithm, specific image sequences from each dataset is chosen. The `freiburg1_desk` and `freiburg2_pioneer_360` image sequences are chosen from TUM RGB-D dataset. The `freiburg1_desk` contains rapid handheld camera movement inside an office setting and `freiburg2_pioneer_360` has shaky dim-lit image sequences of the pioneer robot inside the large industrial hall. The image sequences have 640 x 480 px image resolution. The EuRoC dataset is image sequences of 752 x 482 px image resolution taken from UAV mounted with stereo camera and IMU. The ground truth is given by external motion capture system, laser tracker and 3D structure scan. The `MH01` and `MH04` image sequences of the EuRoC dataset are chosen as it has longer trajectory and rapid UAV movements inside a huge machine hall workspace. The `V101` and `V103` image sequences of the EuRoC dataset are also used for performance evaluation. These sequences are taken inside a highly textured room with rapid UAV movements. The performance of the algorithm in such sequence demonstrate the robustness in highly textured indoor environment.

Apart from the above mentioned datasets, ETH3D SLAM dataset with custom built stereo camera rig depth image sequence with motion capture and IMU ground truth is used. This dataset has many sequences with shaky camera motions, difficult illumination condition, occlusions and camera 'kidnapping' sequences. The image sequences have 739 x 458 px image resolution and are not stereo rectified. The `cables_1` image sequence is used as it contains untextured dim-lit environment. The `large_loop_1` sequence of ETH3D is also used as it contains large loop inside a textureless lab environment with rapid camera movements.

TUM RGB-D dataset is available in `https://vision.in.tum.de/data/datasets/rgbd-dataset/download`, EuRoC dataset in `https://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets` and ETH3D dataset is available in `https://www.eth3d.net/slam_datasets`

3.1.2 Custom

The performance evaluation of the algorithm on custom dataset is helpful in understanding deeper workings of the algorithm. In order to demonstrate performance evaluation, ground-truth information is required. The project utilised the COZMO mobile robot from ANKI for the purpose of dataset generation and ground-truth estimation. The COZMO robot has an inbuilt pre-calibrated monocular camera with 320 x 240 px image resolution. The robot has inbuilt IMU which can give pose estimates in 6 DoF. The ground-truth pose along with RGB images are stored with timestamps in a txt file as per the generic dataset format using the open-source python SDK. The image sequence used for evaluation is from custom built maze in an indoor environment under bright lighting conditions. The maze is built using highly textured books, Rubik's cube and household objects in a circular loop. The robot is controlled using joystick with 20 mm/s forward speed and limited rotation. The robot is made move approximately in a hexagonal trajectory multiple times for 10 minutes. Fig.3 illustrates the custom built maze setup.

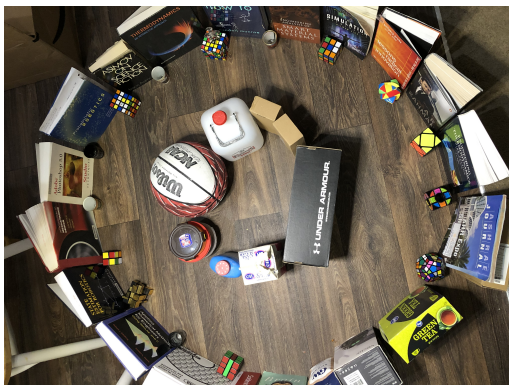

Figure 3. Custom built maze, `circle_cam` sequence

3.2. Code Template

The ORB-SLAM2 and OpenVSLAM algorithms are run on personal laptop with Ubuntu 18.04 LTS OS. The algorithms have templates of code provided in corresponding GitHub repository and elaborate procedure on running generic dataset. The GitHub URL for ORB-SLAM2 is `https://github.com/raulmur/ORB_SLAM2.git` and OpenVSLAM is `https://github.com/xdspacelab/openvslam.git`.

The algorithms are expected to take in image sequences, camera configurations and ORB vocabulary file in order to generate an estimated trajectory with corresponding timestamps.

The SDK of COZMO robot is python based and free-to-use. The tutorials provide a good starting place for implementing a control on the robot. The SDK url is `http://cozmosdk.anki.com/docs/index.html`

3.3. Evaluation metrics

The estimated trajectory from the algorithm has a different co-ordinate system from ground-truth trajectory. If the trajectories are not aligned, it would result in a large error. Further in case monocular camera, the estimated trajectory would be off by a scale value. Therefore, a similarity transformation is performed to scale and align the estimated trajectory with ground-truth trajectory before evaluating the error metrics. Fig.4 visualizes the procedure that needs to be performed before evaluating the error metrics.

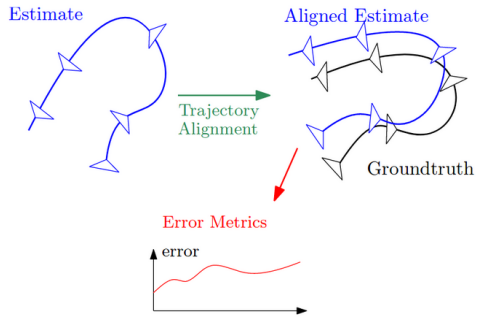

Figure 4. Procedure to evaluate trajectory [19]

The evaluation metric chosen for this project is Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) which are essential in evaluating the global and local consistency of the estimated trajectory [13]. The evaluation metric needs ground truth data, which can be obtained along with dataset, and estimated trajectory, which is generated by running the algorithm. The ATE is given by the following equation,

$$\mathbf{F}_i = \mathbf{Q}_i^{-1}\mathbf{S}\mathbf{P}_i$$

$$\mathbf{ATE} = \mathbf{RMSE}(\mathbf{F}_{1:n}) = \left(\frac{1}{n}\sum_{i=1}^{n}||trans(\mathbf{F}_i)||^2\right)^{0.5}$$

where $\mathbf{Q}_i$ is the ground-truth trajectory, $\mathbf{P}_i$ is the estimated trajectory and $\mathbf{S}$ is the similarity transformation.

The RPE is given by the following equation,

$$\mathbf{E}_i = \left(\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\delta}\right)^{-1}\left(\mathbf{P}_i^{-1}\mathbf{P}_{i+\delta}\right)$$

$$\mathbf{RPE} = \mathbf{RMSE}(\mathbf{E}_{1:n}, \delta) = \left(\frac{1}{m}\sum_{i=1}^{m}||trans(\mathbf{E}_i)||^2\right)^{0.5}$$

where $\mathbf{Q}_i$ is the ground-truth trajectory, $\mathbf{P}_i$ is the estimated trajectory and δ is the time interval.

The project seeks to use the *evo* open-source python package, available at `https://michaelgrupp.github.io/evo/`, for evaluating and plotting the results. These metrics can provide us with useful information about the performance of both the algorithms in terms of image sequence conditions.

3.4. System workflow

Using evo python package, Openvslam and ORB-SLAM2 is getting evaluated using the generic datasets which are not used while performing those algorithm.
Example workflow for performing kittidatasets on these algorithm.
Step-0: Presuming the user have done all the installation for evo from the official website, we proceed to the implementation and analysing results.
Step-1: Initially we plot multiple trajectories, for the example given below, we plot two KITTI pose files with its corresponding ground truth. This can be done using `evo_traj` command. This would show us the overall trajectories evaluated by the algorithm with ground also plotted, from which we can get to know to where did the algorithm deviated from the ground truth.
Step-2: Secondly we need to need to run a metric on the trajectories for calculating absolute trajectories from the SLAM algorithm we are using. This can be done using `evo_ape`. We need to know that `KITTI_00_gt.txt` would as a ground truth file. From this we can calculate the Absolute Percentage Error (APE) with frame count correspondingly, In addition to that we get mean, median , standard deviation, root mean square.
Step-3: This step for processing mutliple results from a metric. For doing we use `evo_res`.
Finally we get all the results combined a single command This line would print all the infos and the required statics ,followed by plots and tables.
Similarity transformation[20] is a method for combining

two different coordinates frames into a single reference frame, this method method utilizes a known vertical direction(from IMU) of a camera to solve the generalized relative pose and scale. This is used mainly in loop closure in visual odometry for avoid drifting and merging multiple Sfm's reconstructions.

4. Experiments and Results

The ATE, RPE, tracking times for different image sequences are evaluated and recorded in the following section.

4.1. vSLAM Evaluation

The ORB-SLAM2 and OpenVSLAM are tested on generic datasets. The obtained results for ATE (in this case APE) of a sequence in EuRoC is shown in Fig.5. It can be seen that the RMSE results around 0.042m which seems to be the least performance value of ORB-SLAM2. This sequence is taken inside huge industrial hall under poor lighting conditions. It can be observed that the estimated trajectory seems to match the ground truth. The sequence contains around 3682 images and takes 0.136731s of mean tracking time.

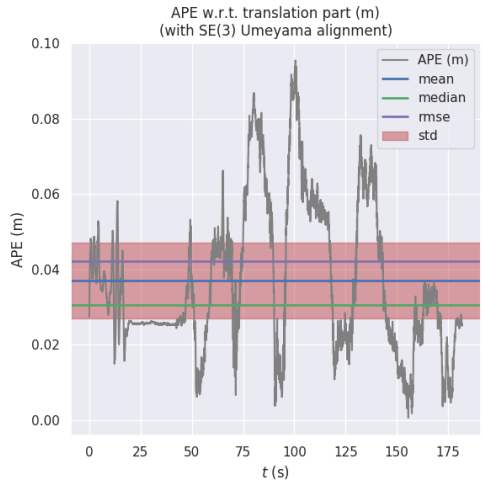

Figure 5. The graph shows the ATE for the `MH_01_easy` sequence of EuRoC dataset

The obtained results for ATE for a sequence in TUM RGB-D dataset is shown in Fig.6. It can be seen that the RMSE results around 0.0186m which seems to the least performance value of ORB-SLAM2. This sequence is taken inside an indoor cluttered desk setting. This sequence contains 573 images and takes 0.0509s of mean tracking time. The loss of tracking in the sequence can be explicitly observed during 6-11s. This could be due to the swift movement of the camera at that particular time. Apart from the above mentioned sequences the algorithm was evaluated for

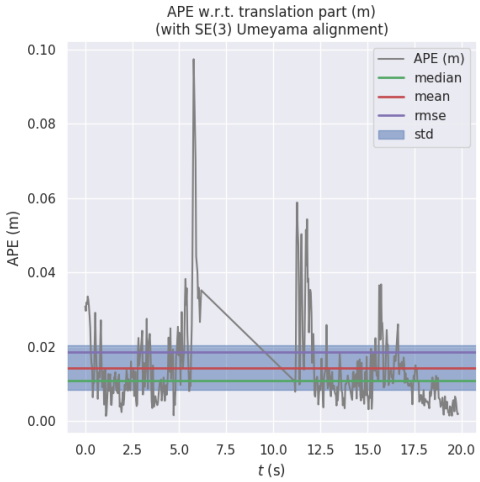

Figure 6. The graph shows the ATE of the fr1/desk sequence of TUM RGB-D dataset

Dataset	Sequence	ATE-RMSE	
		ORB-SLAM	OpenvSLAM
TUM RGB-D	fr1_desk	0.013	0.018
	fr2_p_360	0.0139	0.093
EuRoC	MH01	0.045	0.042
	MH04	0.071	0.221
	V101	0.096	0.095
	V103	0.071	0.068
ETH3D	cables_1	-	0.248
	large_loop_1	-	no loop
COZMO	circle_cam	0.746	-
	square_cam	0.901	0.829

Table 1. ATE - RMSE for monocular camera

different sequences for ATE, RPE, mean tracking time and number of times the tracking got lost. Table.1 shows the ATE of both the algorithms for monocular camera calculated for different sequences. Table.2 illustrates the RPE of both the algorithms for monocular camera calculated for different sequences and Table.3 illustrates the mean tracking time along with tracking lost statistics.

4.2. COZMO sequence evaluation

The `circle_cam` sequence of COZMO is run using ORB-SLAM algorithm at real-time (30fps) and the map obtained after 10 mins sequence is shown in Fig.7. The mapped environment can be compared to the environment represented in Fig.3. The loop-closure and global BA occurred only after completing 3 loops. Fig.8 demonstrates the scaled and aligned estimated trajectory of the `circle_cam` sequence obtained using the *evo* metrics.

Dataset	Sequence	RPE-RMSE	
		ORB-SLAM	OpenvSLAM
TUM RGB-D	fr1_desk	0.016	0.022
	fr2_p_360	0.018	0.041
EuRoC	MH01	0.580	0.532
	MH04	0.780	0.745
	V101	0.612	0.640
	V103	0.545	0.483
ETH3D	cables_1	-	0.248
	large_loop_1	-	no loop
COZMO	circle_cam	1.823	-
	square_cam	0.091	0.281

Table 2. RPE - RMSE for monocular camera

Sequence	Mean tracking time (ms)		Tracking lost
	ORB-SLAM	OpenvSLAM	
fr1_desk	40.516	34.359	1
fr2_p_360	39.041	25.806	5
MH01	48.185	81.647	-
MH04	42.11	34.582	-
V101	51.365	43.763	-
V103	38.994	31.856	6
cables_1	41.17	52.269	1
large_loop_1	23.942	20.74	2
circle_cam	-	-	5
square_cam	-	-	3

Table 3. Mean tracking times (ms) and number of times tracking was lost for monocular camera

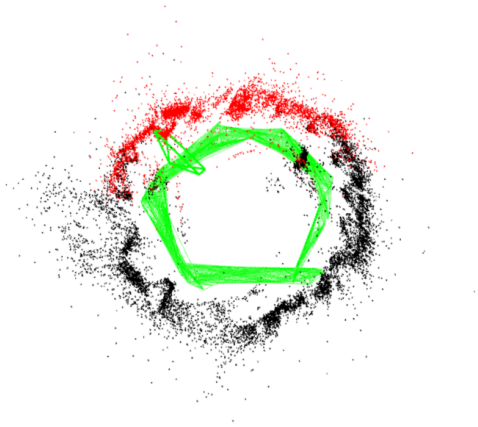

Figure 7. Local map of `circle_cam` sequence - red and black are map points, red - local map points for BA, green - essential graph

5. Discussion

The ATE for TUM RGB-D is the lowest obtained out of all other image sequences. The ORB-SLAM outperforms OpenVSLAM in both sequences of TUM RGB-D dataset but takes more time to track and map the environment. The OpenVSLAM outperforms ORB-SLAM in EuRoC dataset image sequences. Tracking was not lost for three sequences in EuRoC dataset which demonstrates the robustness of both algorithm on UAV platforms. Moreover, higher resolution contributes to better feature extraction but increases tracking time as shown in Table.3. ORB-SLAM failed in all image sequences of the ETH3D dataset but OpenvSLAM, even though performed poorly, was able to map the environment. This illustrates better vocabulary and place recognition module of OpenvSLAM. The custom dataset of COZMO robot resulted in poor performance in both the algorithm due to lower resolution and automatic exposure setting. It can be seen evidently from Fig.8 that estimated trajectory has large offsets and drifts due to large number tracking loss. Although the tracking failed in the custom sequence, loop-closing was identified properly and map fairly represented the environment as shown in Fig.7.

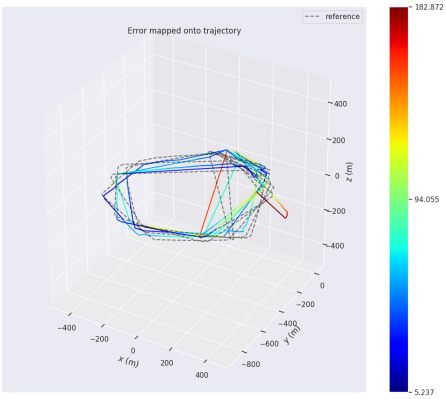

Figure 8. Scaled-aligned estimated trajectory of `circle_cam` image sequence

The project discussed extensively on benchmarking ORB-SLAM and OpenvSLAM on different datasets. Difficult image sequences with large rotation, rapid movements and different lighting condition were chosen from generic dataset to test the robustness of the algorithm. The project also demonstrated the real-time implementation on mobile robot and evaluated the performance of these algorithms on lower resolution images and indoor lighting conditions.

6. Appendix

6.1. Dataset acquisition - COZMO SDK

```
#!/usr/bin/env python3

import sys
import time
import os
import math
import keyboard
```

```
import cv2

try:
    import numpy as np
except ImportError:
    sys.exit("Cannot import numpy: Do `pip3 install --user numpy` to install")

try:
    from PIL import Image
except ImportError:
    sys.exit("Cannot import from PIL: Do `pip3 install --user Pillow` to install")

import cozmo
from cozmo.util import degrees, distance_mm, speed_mmps

def cozmo_stream_images(robot: cozmo.robot.Robot):
  drive_dir = 1
  turn_dir = 0
  FORWARD_SPEED = 35
  TURNING_SPEED = 0
  TIME_LENGTH_OF_IMAGE_SEQUENCE = 600
  ''' Enable streaming of images and color images and obtain the camera configuration for yaml file '''
  robot.camera.image_stream_enabled = True
  robot.camera.color_image_enabled = True

  # Get robot camera configurations
  robot_focal_length = robot.camera.config.focal_length
  robot_image_center = robot.camera.config.center

  # Set text file object to write timestamps and the images
  file_rgb = open("rgb.txt", "a")
  file_config = open("config.txt", "w+")
  file_ground = open("groundtruth.txt", "a")

  file_config.write("%s %s" % (robot_focal_length, robot_image_center))

  # Set start time and capture image sequence for 20s
  end_time = time.time() + TIME_LENGTH_OF_IMAGE_SEQUENCE

  turn_time = 20
  last_time = time.time()
  drive_control = 1
  longLength = 1
  c = 'l'

  while time.time() <= end_time:

    if keyboard.is_pressed('w'):
      drive_dir = 1
      turn_dir = 0
      # Drive the robot with desired speed
      l_wheel_speed = (drive_dir * FORWARD_SPEED) + (turn_dir * TURNING_SPEED)
      r_wheel_speed = (drive_dir * FORWARD_SPEED) - (turn_dir * TURNING_SPEED)
      robot.drive_wheels(l_wheel_speed, r_wheel_speed)

    elif keyboard.is_pressed('s'):
      drive_dir = -1
      turn_dir = 0
      # Drive the robot with desired speed
      l_wheel_speed = (drive_dir * FORWARD_SPEED) + (turn_dir * TURNING_SPEED)
      r_wheel_speed = (drive_dir * FORWARD_SPEED) - (turn_dir * TURNING_SPEED)
      robot.drive_wheels(l_wheel_speed, r_wheel_speed)

    elif keyboard.is_pressed('a'):
      drive_dir = 0
      turn_dir = 1
      # Drive the robot with desired speed
      l_wheel_speed = (drive_dir * FORWARD_SPEED) + (turn_dir * TURNING_SPEED)
      r_wheel_speed = (drive_dir * FORWARD_SPEED) + (turn_dir * 35.0)
      robot.drive_wheels(l_wheel_speed, r_wheel_speed)

    elif keyboard.is_pressed('d'):
      drive_dir = 0
      turn_dir = 1
      # Drive the robot with desired speed
      l_wheel_speed = (drive_dir * FORWARD_SPEED) + (turn_dir * 35.0)
      r_wheel_speed = (drive_dir * FORWARD_SPEED) + (turn_dir * TURNING_SPEED)
      robot.drive_wheels(l_wheel_speed, r_wheel_speed)

    # Get the current image frame from camera
    current_image = robot.world.latest_image

    if current_image is not None:
      # Obtain the received time of the image
      current_image = current_image.raw_image
      received_time = time.time()

      # State of the robot or ground truth
      positionx = robot.pose.position.x
      positiony = robot.pose.position.y
      positionz = robot.pose.position.z
      rotation1 = robot.pose.rotation.q0
      rotation2 = robot.pose.rotation.q1
      rotation3 = robot.pose.rotation.q2
      rotation4 = robot.pose.rotation.q3

      # Set directory and file name to store the image
      image_directory = os.path.join("image01", "%s" % received_time)
      image_name = image_directory + ".png"
      current_image.save(image_name)
      file_rgb.write("%s %s\n" % (received_time, image_name))
      file_ground.write("%s %.4f %.4f %.4f %.4f %.4f %.4f %.4f\n" % (received_time, positionx, positiony, positionz, rotation2, rotation3, rotation4, rotation1))


  # Close the file objects
```

```
  file_config.close()
  file_rgb.close()

cozmo.robot.Robot.drive_off_charger_on_connect = False # Cozmo can stay on his charger for this example
cozmo.run_program(cozmo_stream_images)
```

References

[1] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual slam algorithms: a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 16, 2017.

[2] 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.

[3] S. Sumikura, M. Shibuya, and K. Sakurada, "Openvslam: A versatile visual slam framework," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 2292–2295.

[4] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 29, p. 2007, 2007.

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2007, pp. 225–234.

[6] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time."

[7] J. Engel and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *In ECCV*, 2014.

[8] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 15–22.

[9] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," in *arXiv:1607.02565*, July 2016.

[10] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[11] V. Ila, L. Polok, M. Solony, and P. Svoboda, "Highly efficient compact pose SLAM with SLAM++," *CoRR*, vol. abs/1608.03037, 2016. [Online]. Available: http://arxiv.org/abs/1608.03037

[12] 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.

[13] 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*. IEEE, 2012, pp. 573–580.

[14] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[15] S. Martull, M. Peris, and K. Fukui, "Realistic cg stereo image dataset with ground truth disparity maps," in *ICPR workshop TrakMark2012*, vol. 111, no. 430, 2012, pp. 117–118.

[16] S. Wang, J. Yue, Y. Dong, S. He, H. Wang, and S. Ning, "A synthetic dataset for visual slam evaluation," *Robotics and Autonomous Systems*, vol. 124, p. 103336, 2020.

[17] J. Engel, V. Usenko, and D. Cremers, "A photometrically calibrated benchmark for monocular visual odometry," *arXiv preprint arXiv:1607.02555*, 2016.

[18] M. Rebert, D. Monnin, S. Bazeille, and C. Cudel, "A review of the dataset available for visual odometry," in *Fourteenth International Conference on Quality Control by Artificial Vision*, vol. 11172. International Society for Optics and Photonics, 2019, p. 111720W.

[19] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 7244–7251.

[20] C. Sweeney, L. Kneip, T. Höllerer, and M. Turk, "Computing similarity transformations from only image correspondences," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3305–3313.