A End Semester Thesis Report

On

Distributed SLAM on Quadcopters

By

Sri Mihir Devapi Ungarala

2019AAPS0306H

Under the supervision of

Dr Madhava Krishna, RRC, IIIT Hyderabad

Dr Joyjit Mukherjee, EEE Department, BITS Pilani, Hyderabad

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF BITSF421T

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

HYDERABAD CAMPUS

(May 2023)

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Prof. Madhava Krishna from RRC Lab at IIIT Hyderabad, for allowing me to contribute to projects undertaken by RRC. and Prof. Joyjit Mukherjee from the EEE Department at BITS Pilani, Hyderabad, for supervising my Undergraduate Thesis.

I would also like to thank my teammates in the RRC lab, Bhanu Teja(Team Lead, IIIT Hyderabad), Badri Vishal(Research Intern,IIIT Hyderabad) and Ansh Shah(Research Intern, BITS Pilani, Pilani Campus, currently doing his thesis) who are worked with me in this project. I am grateful to my peers in the EEE Department for all the knowledge and competition.

ABSTRACT

SLAM is a popular algorithm that is used extensively in robotics. It has its applications in mapping, exploration, navigation when GPS signal is not reliable or not available, etc. Whenever there is a huge task or a task that can be done more effectively when there are multiple robots, naturally we want multiple robots to collaborate with each other. Ex: Search-and-rescue, mapping large areas like the whole city. Collaborative SLAM also known as Multi Agent SLAM or Distributed SLAM is an attempt towards this frontier.

The rest of the report is divided into many sections, where important details about SLAM and Multi -Agent SLAM are explained to give a better understanding about the things described in the report. Our setup is also explained followed by results and the work that is currently going on as the project is not yet done followed by conclusion.

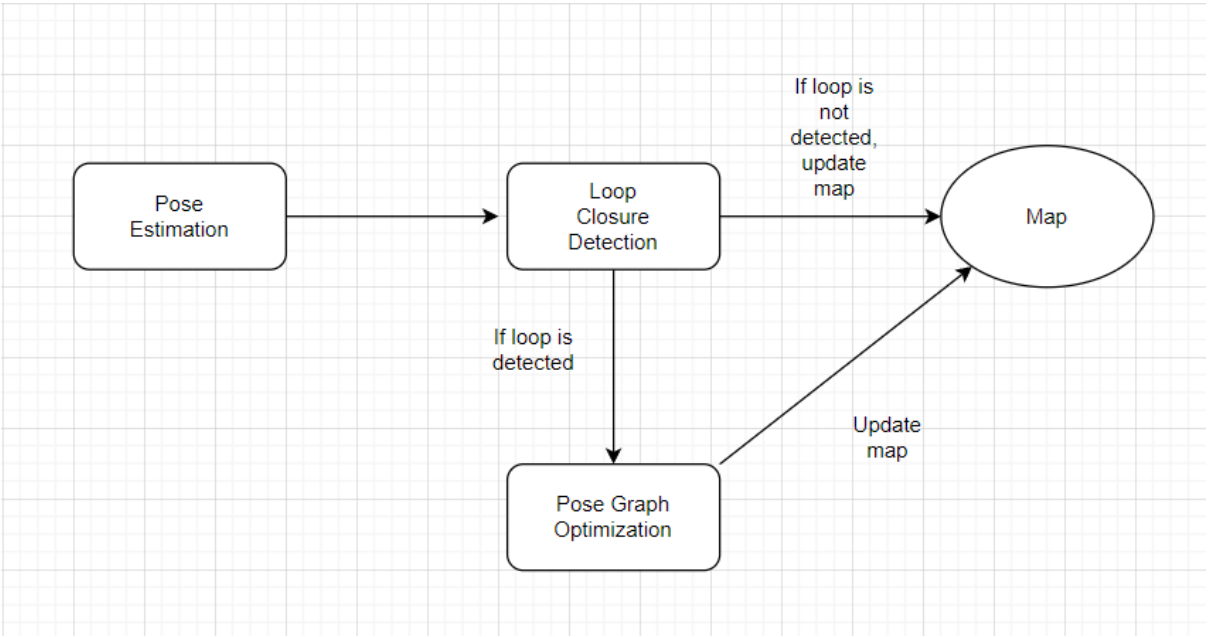
Table of Contents

1. What is SLAM ?

SLAM stands for "**S**imultaneous **L**ocalization **A**nd **M**apping." It is a computational technique used in robotics and computer vision to construct a map of an unknown environment while simultaneously determining the position of the robot or camera within the map. It is used in autonomous cars, mobile robots, drones, etc for various purposes.

SLAM algorithms use sensor data from cameras, lidars, or other range sensors to estimate the location of the robot or camera relative to its surroundings, and use that information to construct a map of the environment. The process is called "simultaneous" because the map and robot location estimates are updated simultaneously as new sensor data becomes available. Below is a very basic pipeline for slam algorithms.

1.1 Pipeline for SLAM



Pose Estimation/Odometry Estimation: Pose is defined as *the position and orientation of the robot* whereas odometry is defined as *change in pose of robot between 2 instances*. They are estimated using the data from the sensor.

Loop closure detection: Loop closure detection is a very important technique in SLAM. Because, no matter how accurate the sensors are, given enough time, drift is accumulated in the measurements. In our case poses. Hence there is a need for correct poses after some time. When the robot revisits a place it has already visited, loop closure is triggered. As revisiting the place provides an additional constraint on all the poses through which accumulated drift can be removed. It is explained in detail in further sections of the report.

Pose graph Optimization: Pose graph is a graph of all the poses of the robot from start with each node being a pose at one point of time and edge being relative pose which is a constraint. When Loop closure provides one more constraint along with this constraints, it forms a non convex optimization problem which, when solved, can correct the poses in pose graph

Map: Using poses from pose graph and data from sensors, a map is created and it is updated whenever there is a change in pose graph.

1.2 Types of SLAM

SLAM algorithms are multi-faceted in nature.

1.2.1 Classification based on Pose Estimation

There are two types of SLAM based on how odometry(change in pose between two instances) is estimated: Feature based and Direct.

- **Feature-based SLAM:** In feature-based SLAM, the algorithm extracts and tracks features in the environment, such as corners or edges, and uses those features to estimate the robot's odometry. Ex: ORB-SLAM[9,10]

- **Direct SLAM:** In direct SLAM, the algorithm, instead of extracting and tracking features, uses all pixels in the images and concepts of photogrammetry to estimate the robot's odometry. Ex: LSD-SLAM [11]

- **Semi direct SLAM:** Semi direct slam is a hybrid of feature based and direct slam. Ex: SVO [13]

1.2.2 Classification based on Sensors

There are different types of sensor setups that can be used for SLAM, each with its advantages and disadvantages. Some of the most common sensor setups for SLAM include:

Visual Sensors: Visual SLAM uses some form of camera for estimating odometry and creates map.

- Monocular Camera: Monocular Visual SLAM uses a single camera as the only sensor for estimating the robot's odometry and constructing a map of the environment. This setup is low-cost and lightweight, making it ideal for small robotic platforms. However, it has limitations in terms of accuracy and robustness, particularly in low-textured environments.

- Stereo Camera: Stereo Visual SLAM uses two cameras mounted on the robot to estimate the robot's odometry and construct a map of the environment. This setup can provide more accurate and robust estimates than monocular SLAM, particularly in low-textured environments. However, it requires more computational power and can be more expensive than monocular SLAM.

- RGBD Camera: RGB-D Visual SLAM uses a camera and a depth sensor, such as a structured light or time-of-flight sensor, to estimate the robot's odometry and construct a 3D map of the environment. This setup can provide more accurate and robust estimates than monocular SLAM, particularly in indoor environments with structured scenes. However, it can be more expensive and less robust than stereo SLAM.

LiDAR Sensors: Lidar SLAM uses a laser scanner to estimate the robot's odometry and constructs a map of the environment. This setup can provide accurate and robust estimates in both indoor and outdoor environments. However, it can be more expensive and heavier than camera-based SLAM setups.

Sensor Fusion: Every sensor has its advantages and disadvantages. Some work better in some conditions while fail in others. Hence multiple sensors are used to mitigate the shortcomings of one another. The most common choice for the second sensor is IMU. For example, Visual Inertial SLAM which uses fused data from camera and IMU sensor. The IMU returns an accurate pose estimate for small time intervals and obtains data at high frequency, but suffers from large drift due to integrating the inertial sensor measurements. Whereas the camera returns an accurate pose estimate over a larger time interval, but obtains data at low frequency and, in case of monocular camera, it suffers from a scale ambiguity for the pose obtained.The fusion of data from multiple sensors can be done in different ways, like using Kalman filters or Particle filters(Monte Carlo filters) or modifications of those. Or techniques like IMU preintegration.

2. What is Multi Agent SLAM?

In SLAM, we only deploy a single agent for exploring unknown territories ,mapping areas, or other performing required tasks. But if the area that is to be mapped or territory that is to be explored is large, it is very inefficient to use a single agent to perform the required tasks.

In Multi agent SLAM, instead of using a single agent, multiple agents(such as drones or mobile robots, etc) are used, that work together, to explore and map environments and perform required tasks.

2.1 Approaches to Multi Agent SLAM

There are two main approaches to multi-agent SLAM: centralized and decentralized

- **Centralized multi agent SLAM:** All agents send their measurements to a central station that creates a global map of the environment. The central station can be a ground station(server) or mobile station(another agent). Ex: COVINS-G[2]

- **Decentralized multi-agent SLAM:** Each agent creates its own local map of the environment and communicates with neighboring agents to share information and merge local maps into a global map. Ex: DDF-SLAM[1]

When compared to SLAM algorithms, there are 3 important aspects that we need to consider in a Multi agent SLAM algorithm.

1. Computational power and Memory
2. Scalability to large environments / Communication
3. Outlier Rejection Process in loop detection

Generally it is a trade-off between computational power and memory and, Scalability to large environments when choosing an approach.

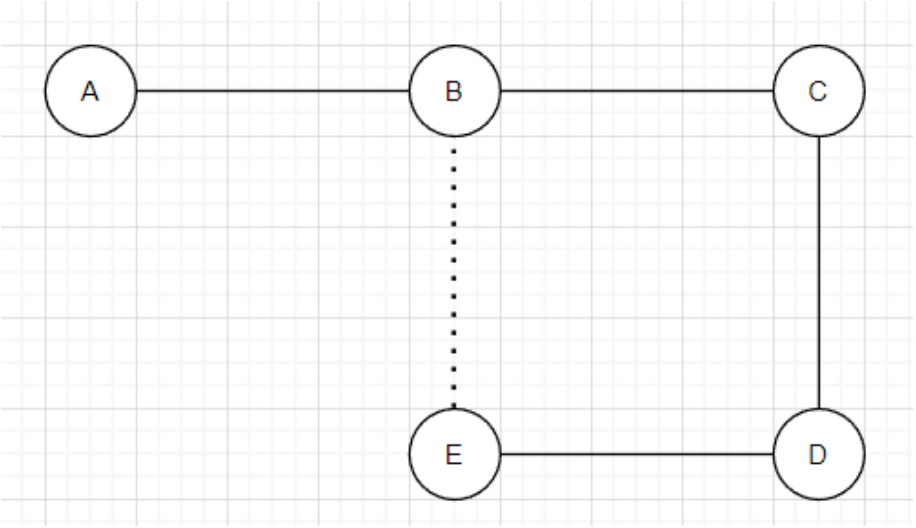
Centralized algorithms trade scalability to large environments for computational power and memory as agents can send data to the central station (mobile or ground) which can process data and get map or move data to a server that is equipped with more resources to process and get high quality map. But agents cannot move beyond a certain distance as they should always be in communication with the central station.

Whereas decentralized algorithms trade computational power and memory for scalability to large environments as there is no need for agents to be able to communicate with central station and exchange information when one is close to another. But agents have to rely on the resources that are available to them.

Outlier rejection process will be explained later in the report(see Section 3).

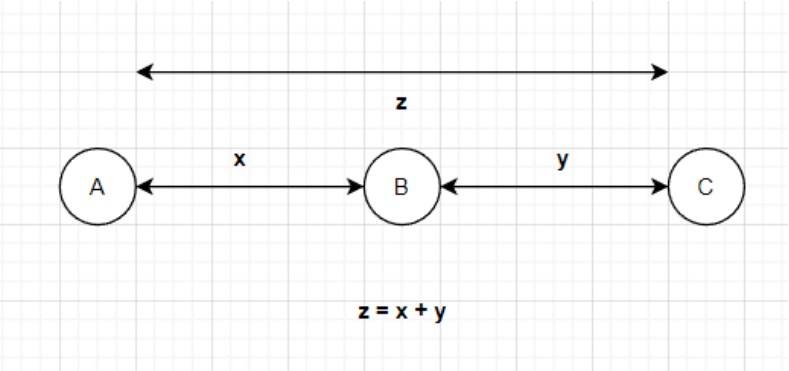
3. Loop Closure Detection and Outlier Rejection Process

3.1 General Idea about Loop Closure Detection



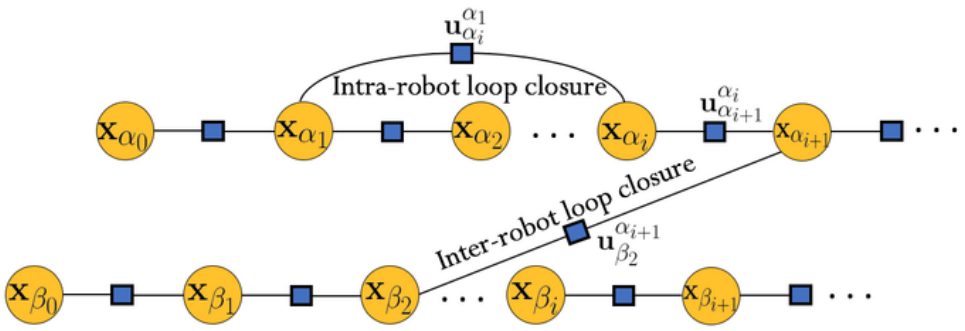
Loop closure detection is an important technique that provides an additional constraint on the poses through which poses are optimized. A loop closure is detected when there is a similarity between sensor data that is obtained and previous sensor data(Similarity in point clouds for LiDAR and similarity in images for Camera).

In above figure A,B,C,D,E are poses of the robot and solid edges are relative poses between them. And let there be a loop closure between B and E, and their relative pose is represented by a dotted line. This relative pose between B and E, i.e loop closure acts as constraint on all poses. Like z = x+y in the figure shown below.



For cameras there are different techniques to evaluate similarity between images. Like using hand crafted keypoint and descriptor techniques (Ex: SIFT, ORB, BRIEF) or using deep learning architectures to get keypoints and descriptors (Ex: SuperPoint) or using deep learning architectures to get a global image descriptor(Ex: NetVLAD) to detect loop based on similarities.

3.2 Outlier Rejection Process in Multi Agent SLAM



(*Source:*https://www.researchgate.net/publication/349195725/figure/fig2/AS:989866506596352@1613013974457/Illustration-of-multi-robot-pose-graph-SLAM-for-a-team-of-two-robots-The-yellow-nodes.ppm)

Unlike Single Agent SLAM, Multi Agent SLAM has 2 types of loop closures.

1. Intra Loop Closures
2. Inter Loop Closures

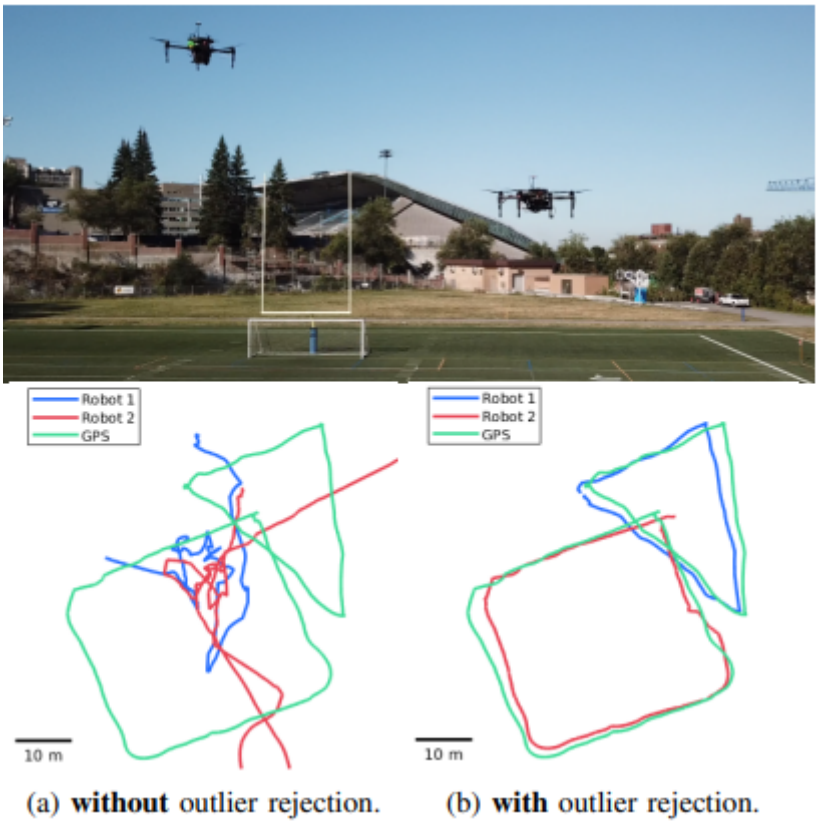
Intra Loop Closures are the loop closures that happen within the agent. These are the type of loop closures that happen in Single Agent SLAM. These are explained above in Section 2.1.1

Inter Loop Closures are the loop closures that happen in between agents. See above figure. If intra loop closures remove the error thereby improving the accuracy of the map constructed by agent, inter loop closures combine the maps that are constructed by multiple agents accurately. Hence, inter loop closures are very important in Multi Agent SLAM for accurate map-fusion. Hence, map fusion in Multi Agent SLAM heavily depends on inter loop closures, making it sensitive to false inter loop closures or outliers.

Need for outlier rejection process

Suppose we are about to map a park using 2 agents. One is sent in the north direction and another in the south direction. After some time, both of them encounter a bench. Since it is a park, all benches are identical.

Now when we are about to fuse maps, if we use the same mechanism that we use intra loop closure detection in inter loop closure detection i.e, similarity in images and similarity in point clouds ,then it is prone to false inter loop closure detections (In this case, benches are considered same, hence loop is detected and inter loop closure is performed. Though the bench that agent1 saw is similar to that of agent2, they are not the same as agent1 went in the north direction and agent 2 went in the south direction implying that this loop detection is an outlier) which can decrease accuracy by large margin. (Check image below). Hence, we need outlier rejection mechanisms to avoid these scenarios.



(a) **without** outlier rejection. (b) **with** outlier rejection.

(*Source: DOOR SLAM paper[3]*)

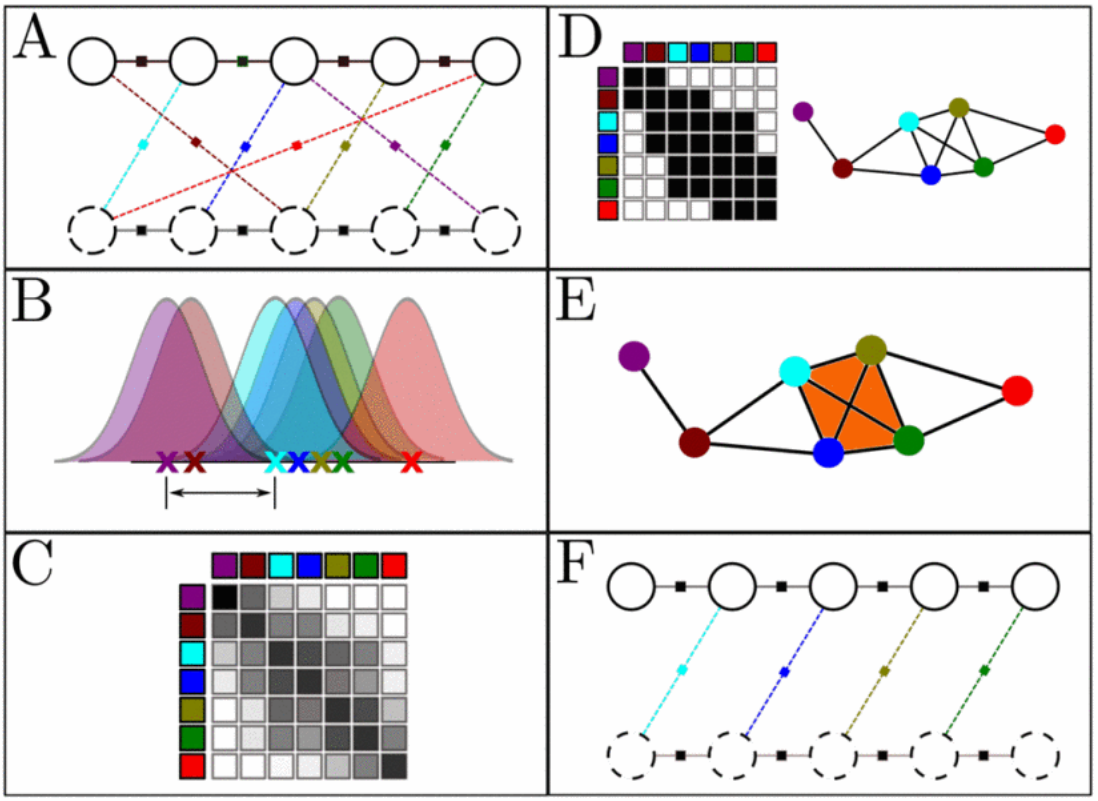
3.3 Different Outlier Rejection Processes proposed across Literature

3.3.1 Pairwise Consistency Maximization (PCM)

PCM is a method designed for the multi-robot case that checks for consistent inter-robot loop closures. It checks whether the loop closures detected by each robot are consistent with each other. In Kimera[4] paper, an incremental extension of PCM is used.

In their implementation, it will be verified if each loop closure is consistent with the odometry by checking if the poses along the cycle formed by the odometry and the loop closure compose to the identity. You flag loops as outliers if the error accumulated along the cycle is not consistent with the measurement noise, using a Chi-squared test.

If the loop detected passes odometry check, it will be added to the adjacency matrix incrementally for pairwise consistency check with other loop closures. (Whereas in PCM from [5] adjacency matrix is always built from scratch when a new loop is added) And uses a fast maximum clique algorithm to compute the largest set of consistent loop closures.



(*Source: Pairwise Consistent Measurement Set Maximization for Robust Multi-Robot Map Merging Paper*)

3.3.2 Graduated Non-Convexity (GNC)

Unlike other outlier rejection algorithms, GNC is not an outlier rejection algorithm but a robust optimization algorithm based on classical M-estimation i.e, it is robust/insensitive to outliers. The algorithm employs a sequence of surrogate cost functions that gradually converge to the original non-convex robust cost function. This is achieved by reformulates the problem using the Black-Rangarajan Duality.

Each surrogate problem takes the form of a weighted least-squares problem, and the algorithm alternates between updating the variables and weights and

updating the control parameter μ. The update for each weight has a closed-form expression that depends on the current robust surrogate function ρ_{μ}. The control parameter μ is gradually increased/decreased until it approaches a value close to μ1, which ensures convergence to the original robust cost function. (See below figure)

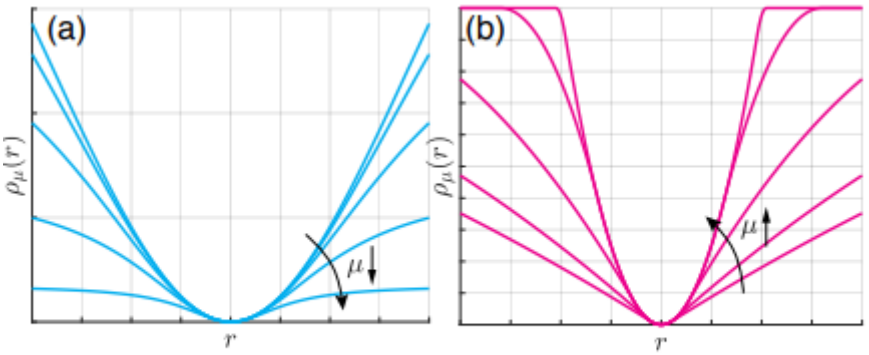


Fig. 1. Graduated Non-Convexity (GNC) with control parameter μ for (a) Geman McClure (GM) and (b) Truncated Least Squares (TLS) costs.

(*Source: Graduated Non-Convexity for Robust Spatial Perception: From Non-Minimal Solvers to Global Outlier Rejection[7]*)

The above method is described in [7], modified for it to work in fully distributed fashion is Kimera Multi[6] by using Riemannian block-coordinate descent (RBCD) solver as the workhorse during iterative optimization.

4. Our Setup

4.1 Hardware Setup

We are using a Carbon Fiber mid sized-drone with a Pixhawk as a controller and Intel NUC Core i7 for computation. Intel RealSense T265 and Stereolabs Zed 2 for RGB-D and Point Cloud Creation.

4.2 Software Setup

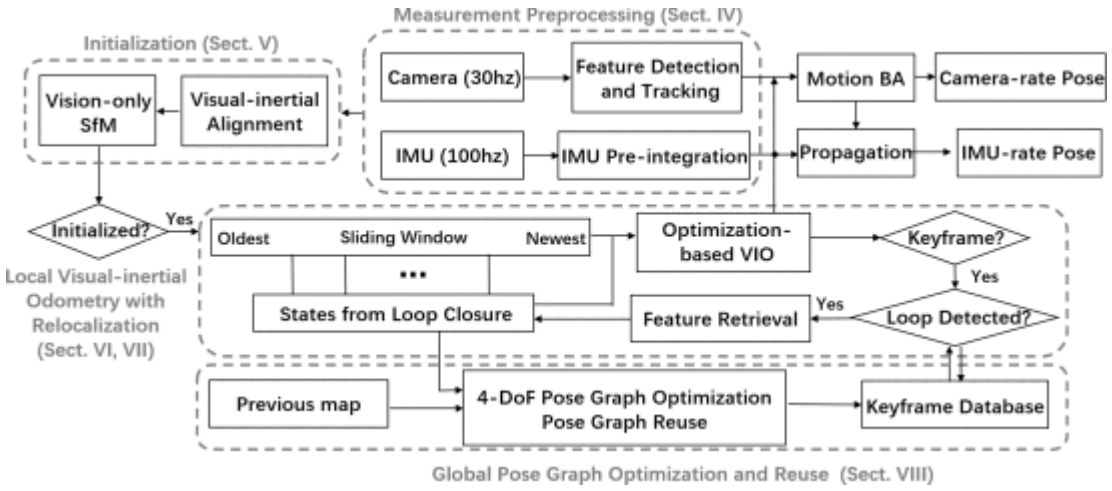
This section covers different algorithms that we are using and reasons for them being used.

4.2.1 Vins MONO

For our use case we have chosen VINS-MONO[8] because it has better documentation, and at the same time is one of the algorithms that performs best for our setup on Intel NUC.

VINS-Mono has several advantages over other SLAM algorithms. It provides accurate and robust estimates of the camera pose. It also has low computational requirements, which makes it suitable for real-time applications.

It uses both Visual and Inertial measurements to estimate camera poses. Though there are straight forward ways of fusing visual and inertial measurements using EKF based approaches and its variants, VINS mono uses IMU preintegration technique which is based on graph optimization formulation. This technique is being used increasingly in recent literature for fusing visual and inertial measurements.



For loop closure detection, Vins-MONO uses DBOW2 with BRIEF descriptors as BRIEF descriptors are faster to compute. Additionally instead of optimizing on 6-DOF (position, orientation), Vins-MONO optimizes only on 4-DOF(x,y,z and yaw angle). This comes from the fact that we can determine the horizontal plane by the gravity vectors, that means we observe the absolute roll and pitch angles all the time. Therefore, the roll and pitch are absolute states in the world frame, while the x, y, z, and yaw are relative estimates with respect to the reference frame. There by taking less time for pose graph optimization. And Vins-MONO formulates pose graph optimization problem using Ceres solver by Google which is a library used for solving optimization problems

4.2.2 NetVLAD

In section 4.2.1, it said that Vins-MONO uses DBOW2 and BRIEF descriptors for loop closure detection. Though BRIEF descriptors are fast to compute,

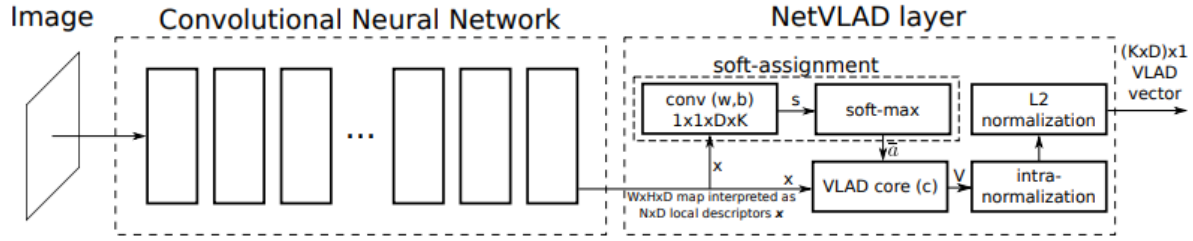
they are not robust enough. And they fail in large lumination changes or in low-light conditions. Hence we are shifting from DBOW2 + BRIEF descriptors to NetVLAD.

NetVLAD[14] is a CNN architecture for weakly supervised place recognition that is trainable in an end-to-end manner. The main component of this architecture, NetVLAD, is a new generalized VLAD layer, inspired by the "Vector of Locally Aggregated Descriptors" image representation commonly used in image retrieval.

Given N D-dimensional local image descriptors as input $\{x_i\}$, and K cluster centers ("visual words") $\{c_k\}$as VLAD parameters, the output VLAD image representation V is K×D-dimensional. For convenience we will write V as a K ×D matrix, but this matrix is converted into a vector and, after normalization, used as the image representation. The (j, k) element of V is computed as follows:

$$V(j,k) = \sum_{i=1}^{N} a_k(x_i)(x_i(j) - c_k(j))$$

where $x_i(j)$ and $c_k(j)$ are the j-th dimensions of the i-th descriptor and k-th cluster center, respectively. $a_k(x_i)$ denotes the membership of the descriptor x_i to k-th visual word, i.e. it is 1 if cluster c_k is the closest cluster to descriptor x_i and 0 otherwise. The idea of NetVLAD is to mimic VLAD in a CNN framework and design a trainable generalized VLAD layer with feature vector from CNN architecture as input.



(*Source: NetVLAD: CNN architecture for weakly supervised place recognition paper*)

NetVLAD is being increasingly used for loop closure detection and it is robust to luminous changes and works in low-light conditions.

4.2.3 GTSAM

GTSAM[12] (short for "Georgia Tech Smoothing and Mapping") is an open-source library for solving non-linear optimization problems, specifically for robotics and computer vision. It provides a collection of algorithms for robot localization, mapping, and SLAM (Simultaneous Localization and Mapping), as well as a framework for implementing custom optimization-based algorithms.

And GTSAM has the GNC optimizer implementation (not Distributed GNC). Hence we are shifting from Ceres solver to GTSAM for pose graph optimization.

4.2.4 Multi Agent SLAM approach

We are going to adopt the centralized approach of Multi agent SLAM for our work. Reasons will be explained in the next section.

5. Work and Results

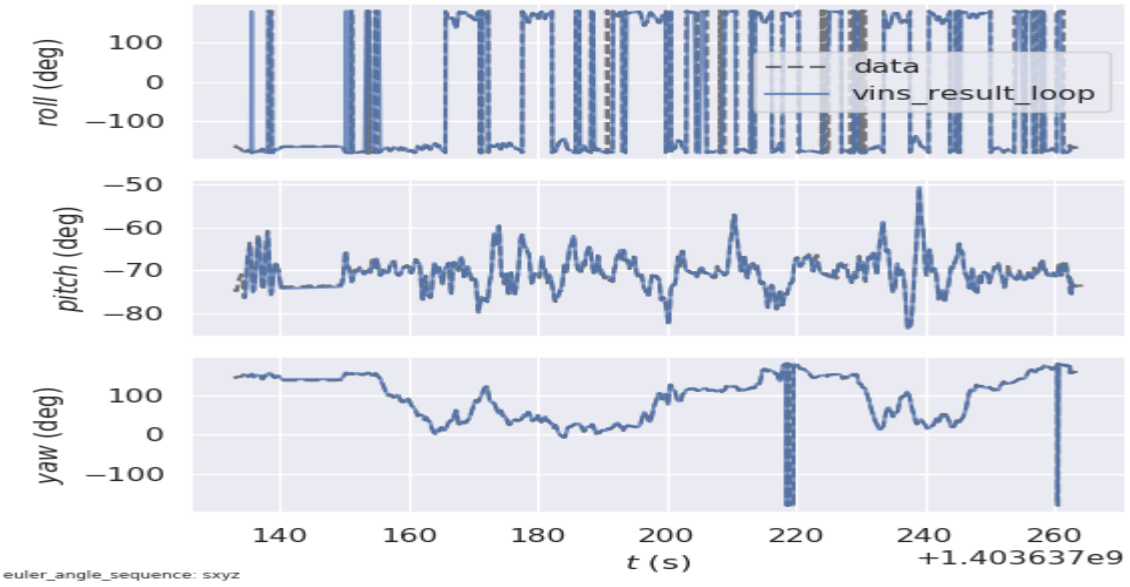
For all the results that are presented in this section, MH_03 of the EuROC dataset is used(both rosbag and ASL Dataset format). And for evaluation of results, EVO library is used, which is a python library. Evo library provides evo_traj for analyzing, plotting or exporting one or more trajectories and, evo_ape and evo_rpe for absolute pose error related plots and relative pose error related plots respectively.

Note: If value has “~” symbol, that means it has averaged over 10 instances

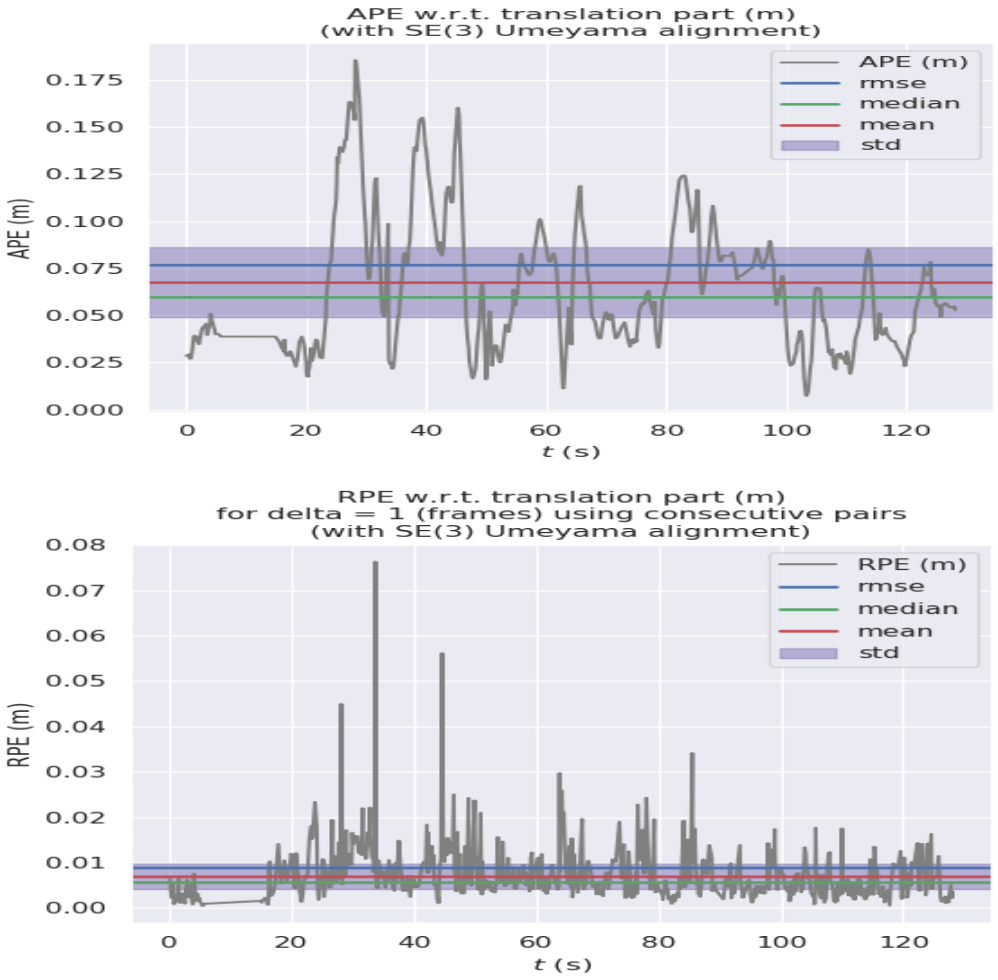
5.1 Vins-MONO

First we would like to check performance of Vins-MONO without any modifications

Time taken for optimization	~ 47.09 ms
Time taken for image to be converted to descriptor (BOW vector using BRIEF)	~ 9.72ms
Number of keyframes detected	860

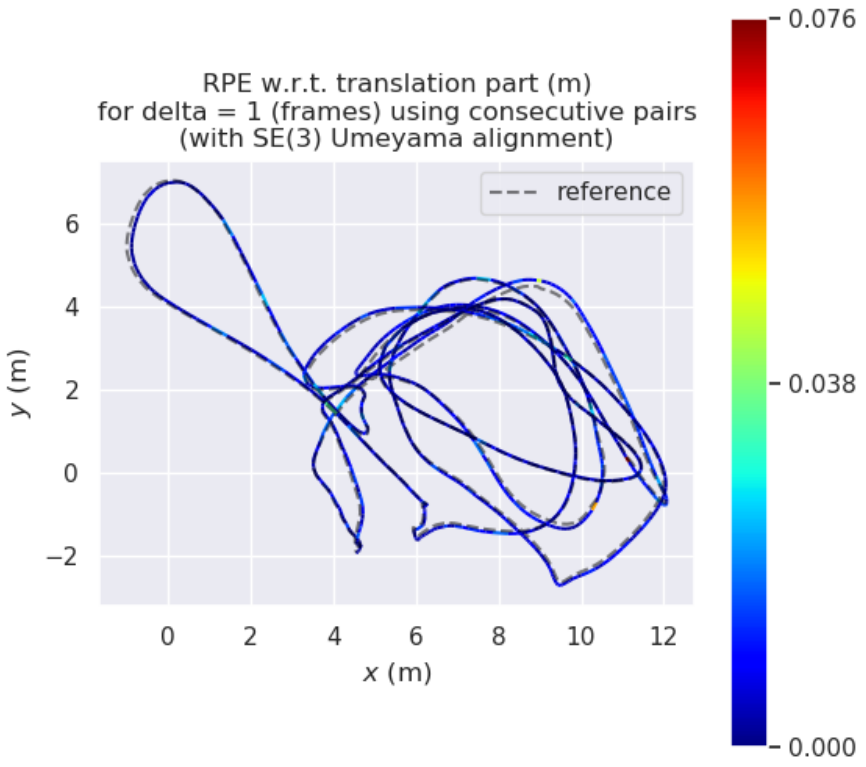


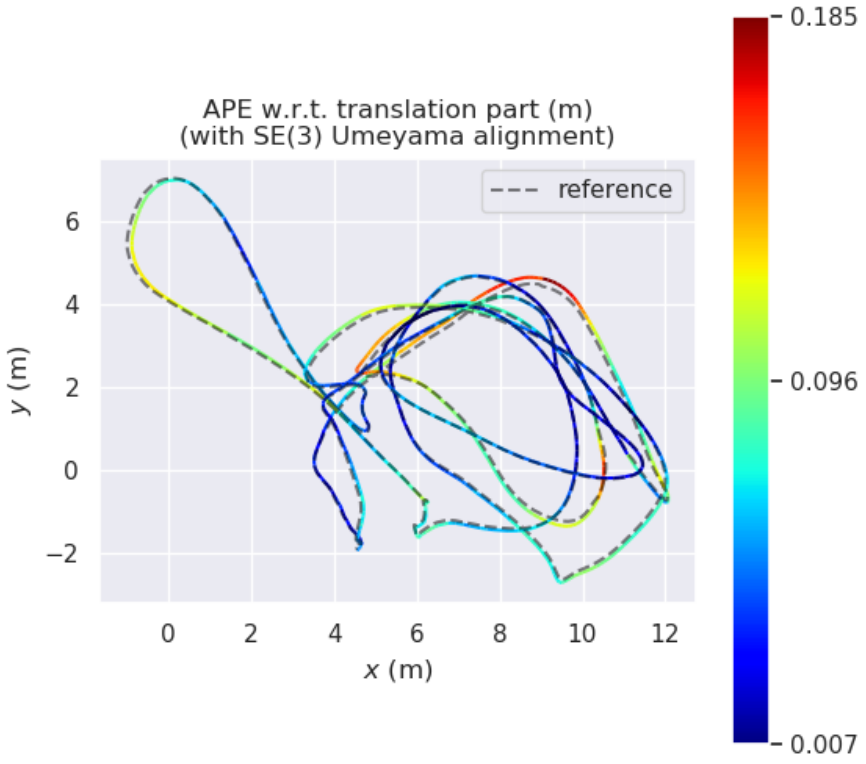
Absolute Pose Error and Relative Pose Error:



Metrics	**APE**	**RPE**
Maximum	0.185399	0.076288
Mean	0.067471	0.007031
Median	0.059738	0.005653
Minimum	0.007298	0.000428
RMSE	0.076914	0.008973
Standard Deviation	0.036923	0.005576

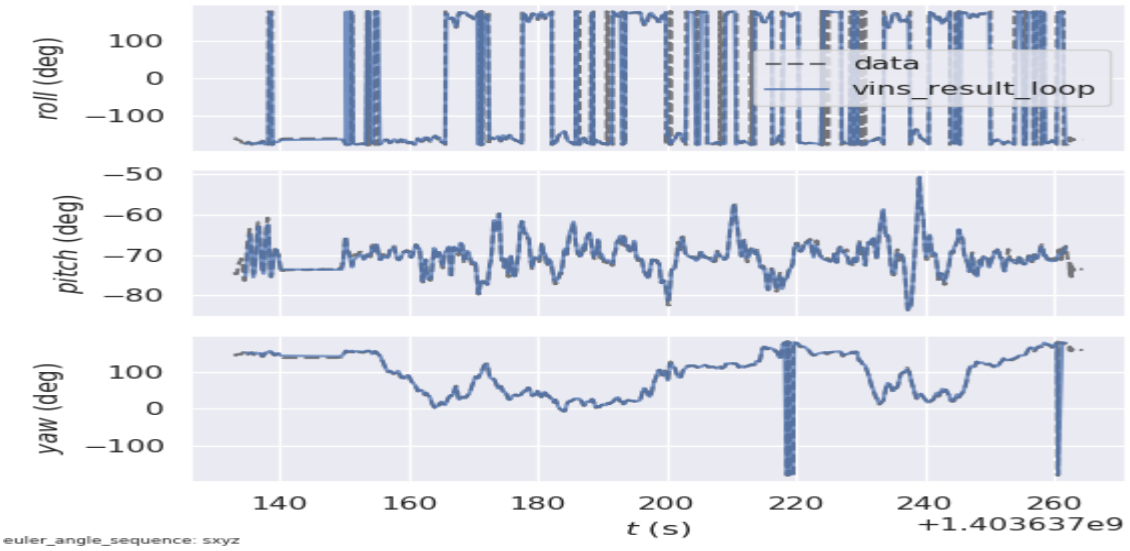
Graphs:

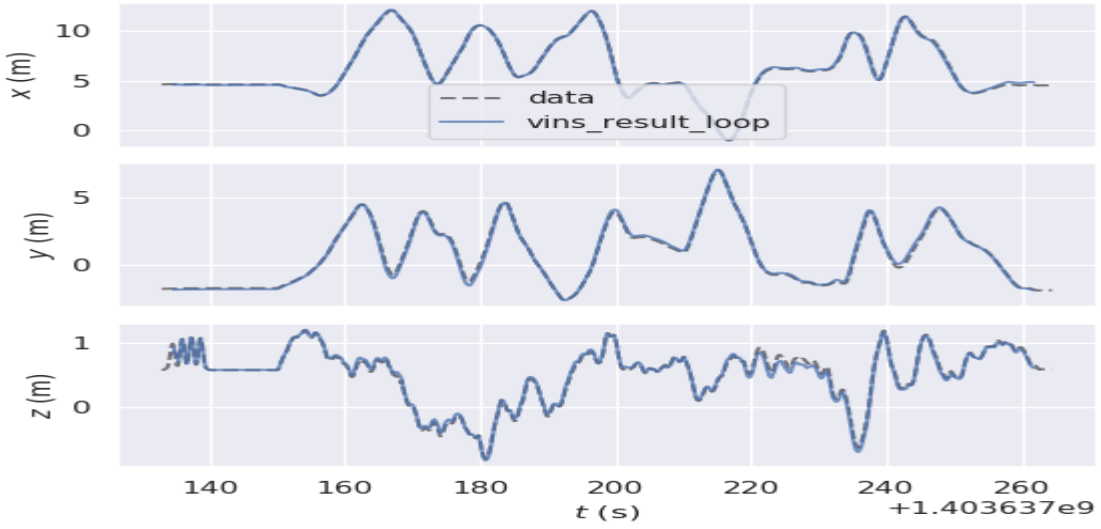

RPE w.r.t. translation part (m)
for delta = 1 (frames) using consecutive pairs
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.076
0.038
0.000


APE w.r.t. translation part (m)
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.185
0.096
0.007

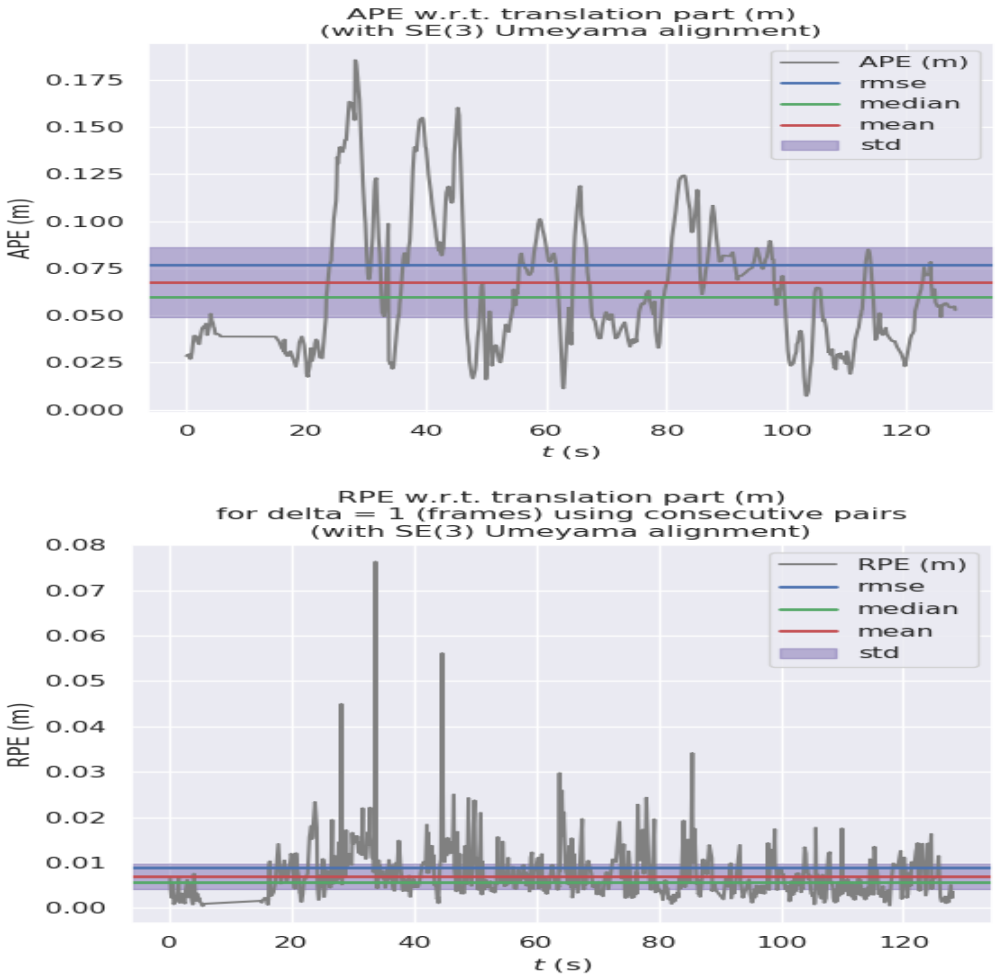
5.2 Vins-MONO + GTSAM

Time taken for optimization	~ 42.91 ms
Time taken for image to be converted to descriptor (BOW vector using BRIEF)	~ 9.54ms
Number of keyframes detected	931



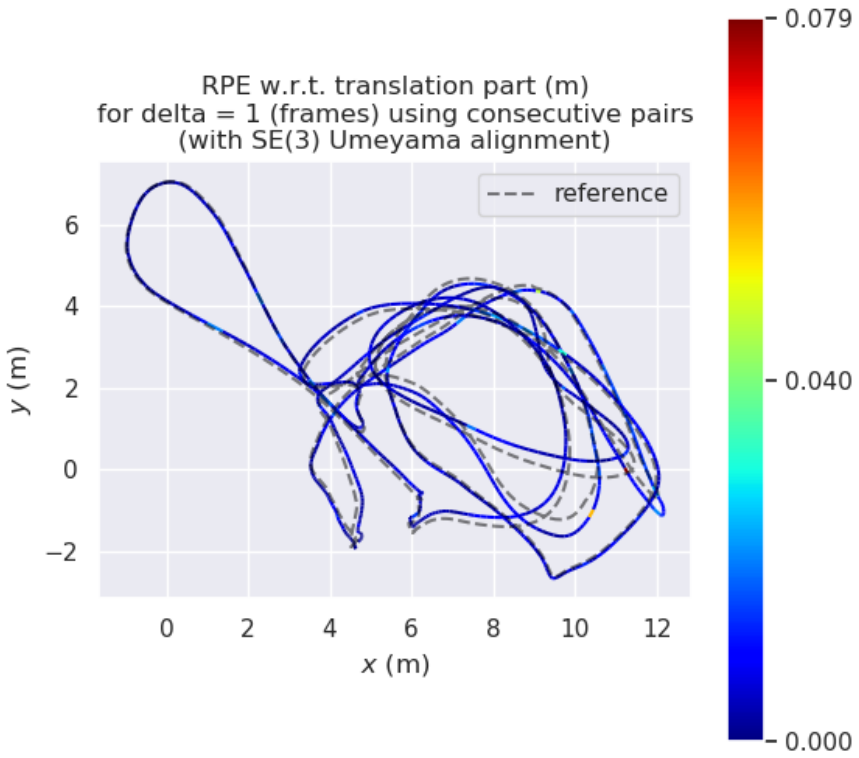


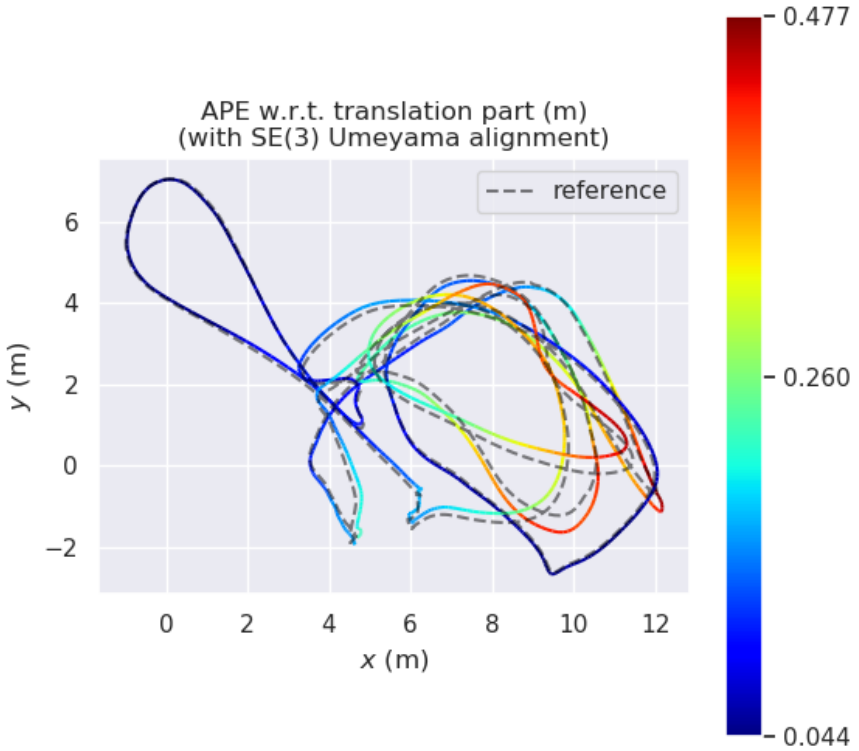
Absolute Pose Error and Relative Pose Error:



Metrics	**APE**	**RPE**
Maximum	0.476563	0.079022
Mean	0.203035	0.005996
Median	0.172866	0.004995
Minimum	0.043675	0.000145
RMSE	0.233571	0.007885
Standard Deviation	0.115464	0.005121

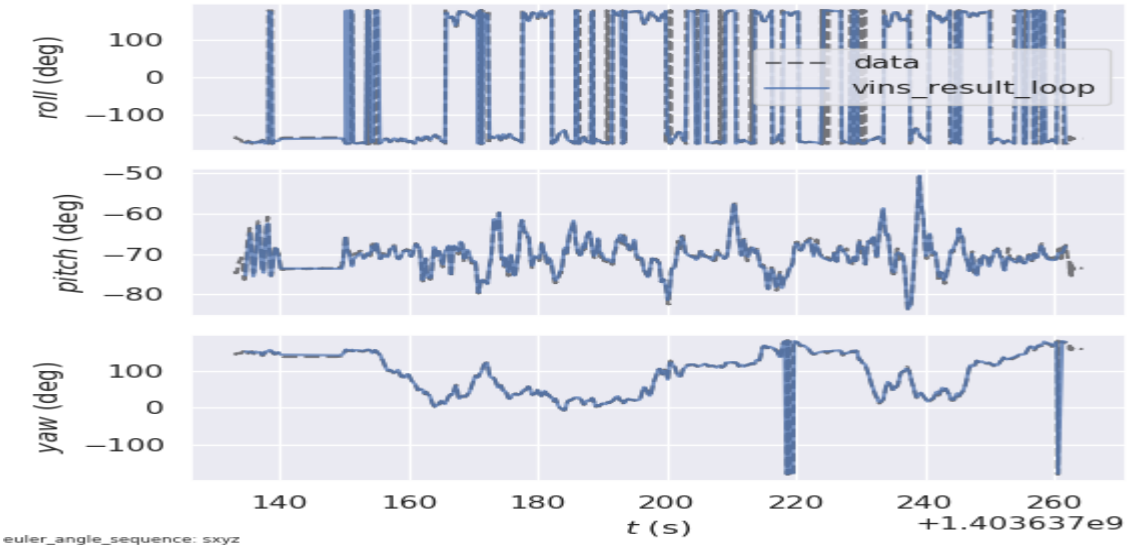
Graphs:

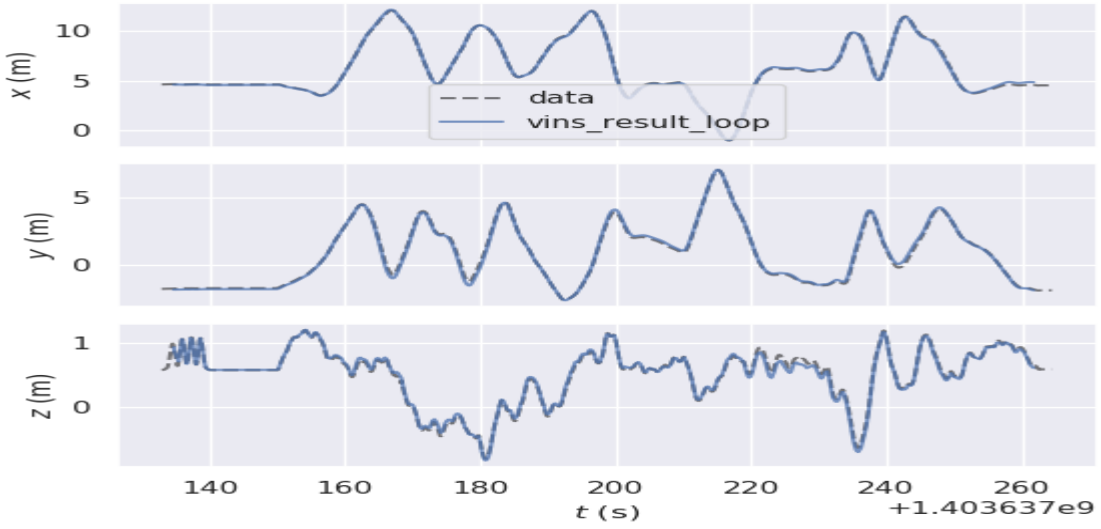

RPE w.r.t. translation part (m)
for delta = 1 (frames) using consecutive pairs
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.079
0.040
0.000


APE w.r.t. translation part (m)
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.477
0.260
0.044

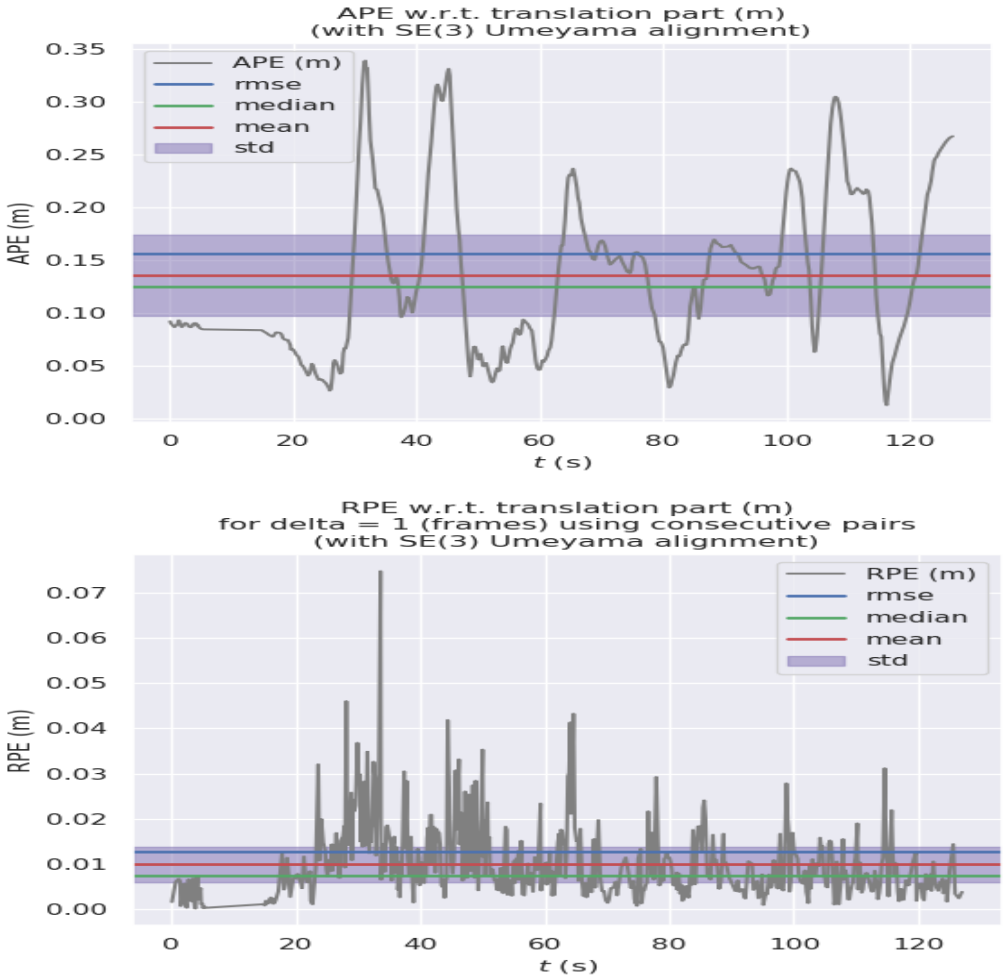
5.3 Vins-MONO + GTSAM and NetVLAD in series

Time taken for optimization	~ 40.48 ms
Time taken for image to be converted to descriptor (NetVLAD descriptor)	~ 125 ms (for input image size of 128 by 128)
Number of keyframes detected	574



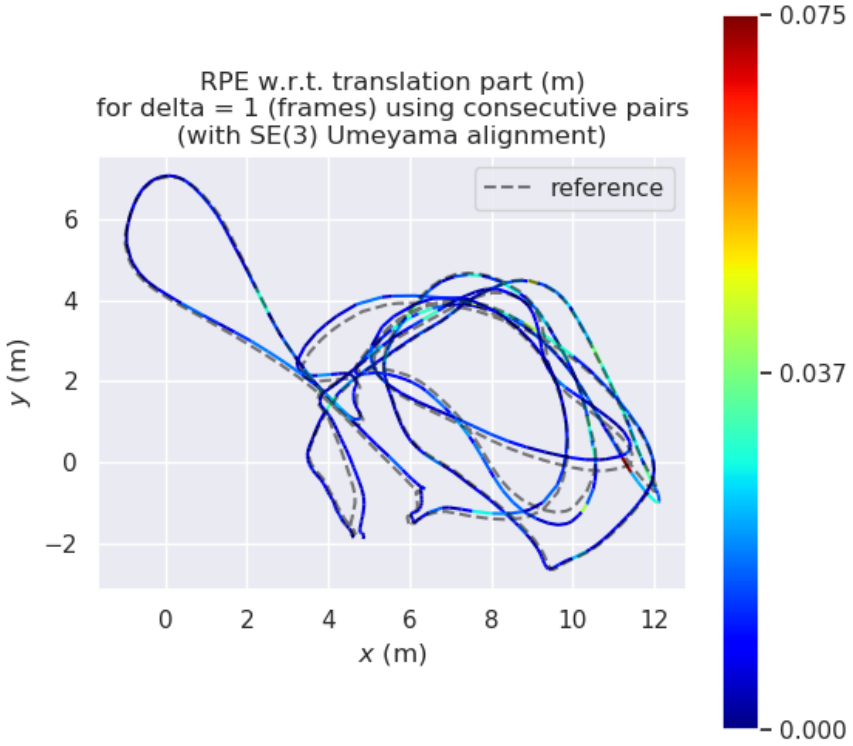


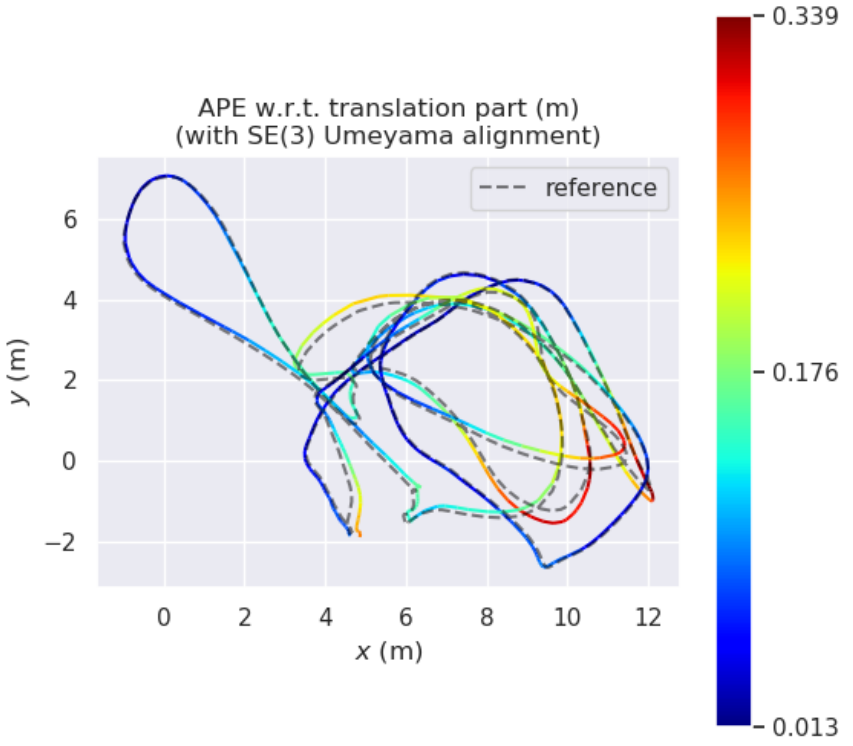
Absolute Pose Error and Relative Pose Error:



Metrics	**APE**	**RPE**
Maximum	0.338593	0.074699
Mean	0.135553	0.009950
Median	0.124867	0.007548
Minimum	0.012708	0.000219
RMSE	0.155942	0.012760
Standard Deviation	0.077094	0.007989

Graphs:


RPE w.r.t. translation part (m)
for delta = 1 (frames) using consecutive pairs
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.075
0.037
0.000


APE w.r.t. translation part (m)
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.339
0.176
0.013

Observations:

- Number of keyframes is decreased approximately to half when compared with Vins-MONO + GTSAM setup. This could be because when adding a keyframe to the list of keyframes, it is first checked if it has a closure or not and then will be added to the keyframe list.

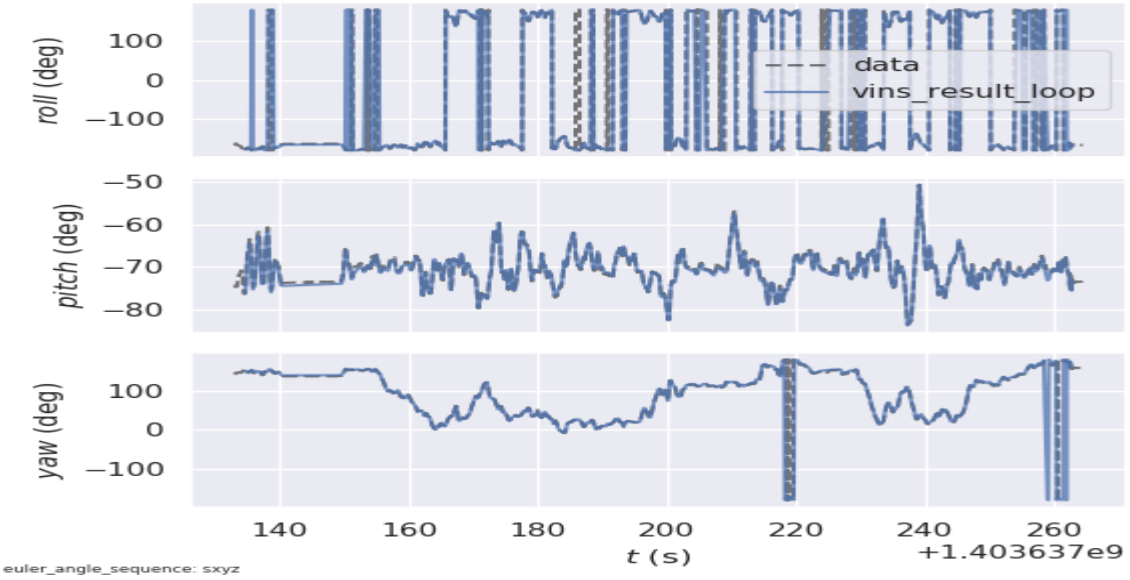
  Since inference time for NetVLAD is large, even when input image size is 128 by 128, and both of these processes are in series, thereby creating a drop in number of keyframes.

  So we thought of creating a separate buffer and run loop closure detection is a separate thread. Now when the keyframe is being added to the keyframe list, it will also be added to the buffer. And in the separate thread, when loop closure detection is done, it checks for loop closure for the keyframe in the buffer. Now both of the processes run in parallel.

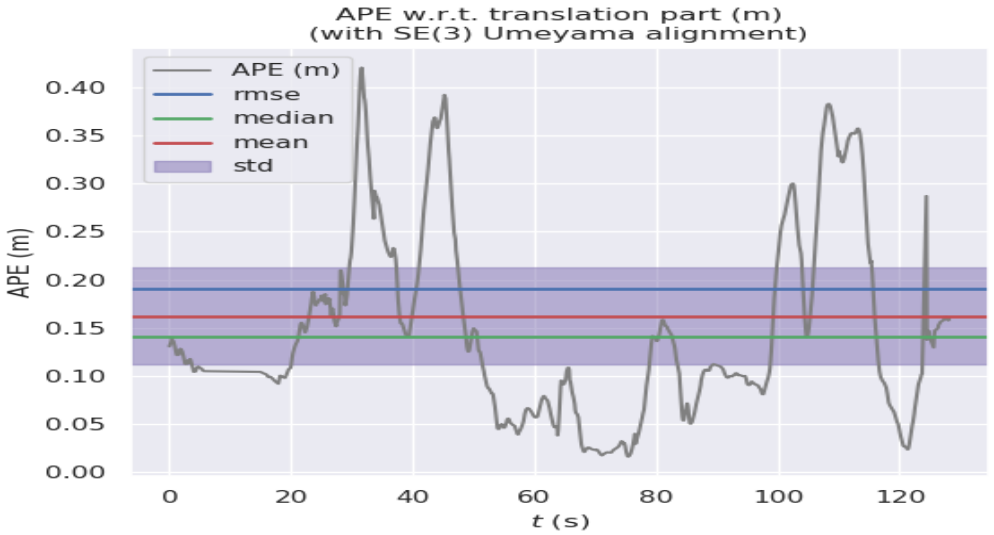
- Inference time(time taken for transforming image to descriptor) for NetVLAD is too large. This is suspected to be due to the large size of the NetVLAD descriptor that we are using i.e 256*128 where 256 is the number of clusters. We are looking for other NetVLAD models which have less descriptor size thereby having less inference time.

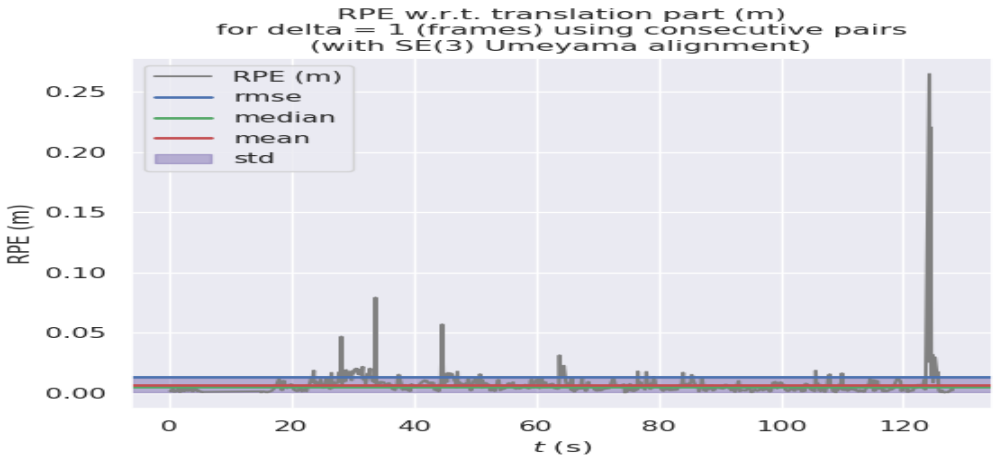
5.4 Vins-MONO + GTSAM and NetVLAD in parallel

Time taken for optimization	~ 38.46 ms
Time taken for image to be converted to descriptor (NetVLAD descriptor)	~ 144 ms (for input image size of 128 by 128)
Number of keyframes detected	966



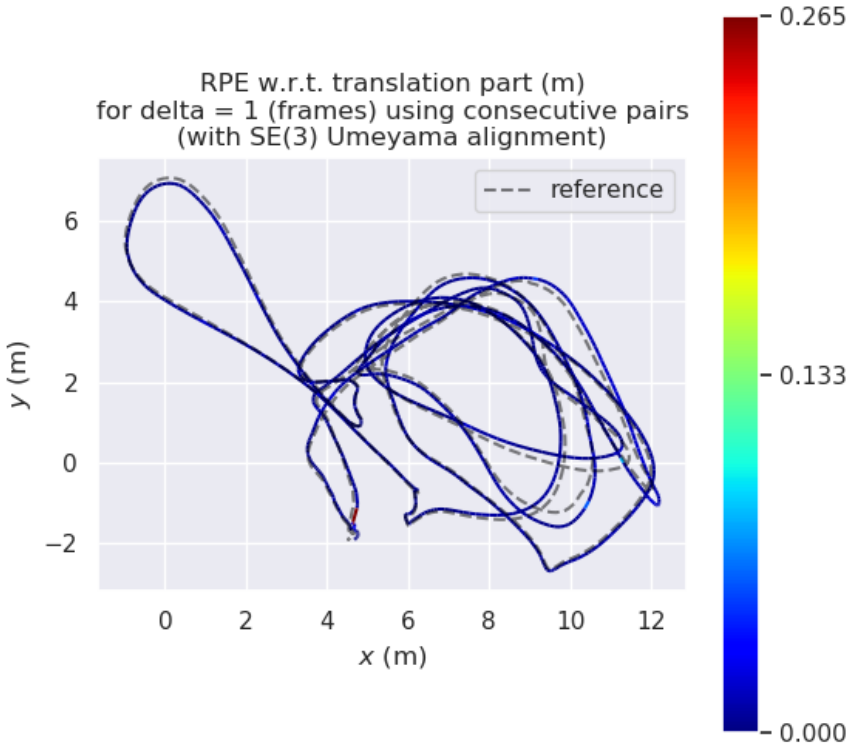
Absolute Pose Error and Relative Pose Error:

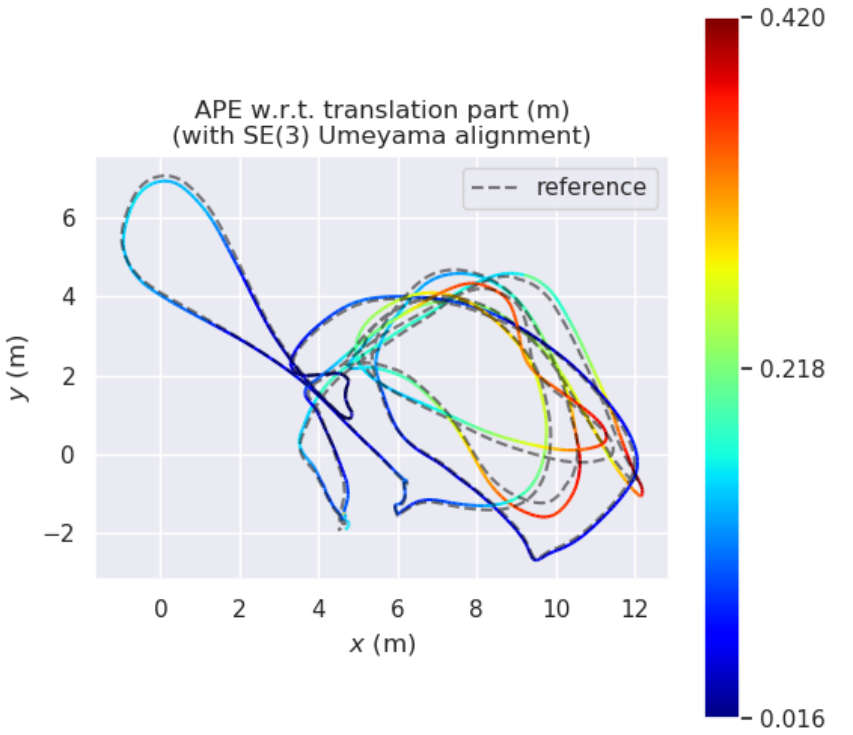




Metrics	**APE**	**RPE**
Maximum	0.420216	0.264638
Mean	0.162274	0.006675
Median	0.141001	0.005010
Minimum	0.016380	0.000447
RMSE	0.190779	0.013818
Standard Deviation	0.100318	0.012099

Graphs:


RPE w.r.t. translation part (m)
for delta = 1 (frames) using consecutive pairs
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.265
0.133
0.000


APE w.r.t. translation part (m)
(with SE(3) Umeyama alignment)
reference
y (m)
x (m)
0.420
0.218
0.016

5.5 Why centralized Multi Agent SLAM?

If you look at observations made in section 5.3, they are:

- Large inference time for NetVLAD
- Drop in keyframes due to above

Along with these observations, a solution for each of them is proposed and the solution for the drop in number of keyframes is implemented in section 5.4. From the comparing results of section 5.4 to that section 5.3, we can say that it is working well. But the problem of large inference time is not solved.

We cannot use the configuration that is used in Section 5.4 i.e, Vins-MONO+GTSAM+NetVLAD parallel because though there is no drop in number of keyframes, there is lag when keyframe is added and when pose related to that keyframe gets optimized. This can affect the global path planner and obstacle avoidance algorithm of the agent.

Hence, it is decided that for intra loop closure detection we will use DBOW2+BRIEF and for inter loop closure detection we will use NetVLAD. Now agent can make the local map faster without lag, but there will be lag in building the global map. But this won't be a problem as individual agents don’t need a global map for navigation safely and are already capable of making their own map, thus successfully removing the problems that can occur in global path planning and obstacle avoidance.

But this kind of solution is very inefficient in a decentralized approach as agents have to wait for a long time before completing the exchange of information and start mapping again. And due to limited resources, map quality can decrease. Hence the centralized approach was taken. We are currently working on setting up a VPN network to facilitate communication with the central station.

6. Conclusion

After a huge literature survey, looking through different theoretical concepts, engineering ideas and numerous debates, we have reached here. So far, the results are looking good. When the last piece of the puzzle is solved i.e, setting up communication between agents, we can go for testing in the real-world with everything combined. Apart from that, we can also try replacing BRIEF descriptors with ORB descriptors which are as fast as BRIEF descriptors and are more robust than BRIEF descriptors.

When this project is done, it has many applications like mapping, exploration, search-and-rescue, etc.

7. References

1. A. Cunningham, M. Paluri and F. Dellaert, "DDF-SAM: Fully distributed SLAM using Constrained Factor Graphs," 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, Taiwan, 2010, pp. 3025-3030, doi: 10.1109/IROS.2010.5652875.(link)

2. Manthan Patel, Marco Karrer, Philipp Bänninger, Margarita Chli, “COVINS-G: A Generic Back-end for Collaborative Visual-Inertial SLAM” arXiv:2301.07147v2 [cs.RO] (link)

3. Pierre-Yves Lajoie, Benjamin Ramtoula, Yun Chang, Luca Carlone, Giovanni Beltrame, “DOOR-SLAM: Distributed, Online, and Outlier Resilient SLAM for Robotic Teams” arXiv:1909.12198v2 [cs.RO] (link)

4. A. Rosinol, M. Abate, Y. Chang and L. Carlone, "Kimera: an Open-Source Library for Real-Time Metric-Semantic Localization and Mapping," 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France, 2020, pp. 1689-1696, doi: 10.1109/ICRA40945.2020.9196885. (link)

5. J. G. Mangelson, D. Dominic, R. M. Eustice and R. Vasudevan, "Pairwise Consistent Measurement Set Maximization for Robust Multi-Robot Map Merging," 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 2018, pp. 2916-2923, doi: 10.1109/ICRA.2018.8460217.(link)

6. Yulun Tian, Yun Chang, Fernando Herrera Arias, Carlos Nieto-Granda, Jonathan P. How, Luca Carlone, “Kimera-Multi: Robust, Distributed, Dense Metric-Semantic SLAM for Multi-Robot Systems” arXiv:2106.14386v2 [cs.RO] (link)

7. Heng Yang, Pasquale Antonante, Vasileios Tzoumas, Luca Carlone, “Graduated Non-Convexity for Robust Spatial Perception: From Non-Minimal Solvers to Global Outlier Rejection” arXiv:1909.08605v4 [cs.CV] (link)

8. T. Qin, P. Li and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," in IEEE Transactions on Robotics, vol. 34, no. 4, pp. 1004-1020, Aug. 2018, doi: 10.1109/TRO.2018.2853729.(link)

9. C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel and J. D. Tardós, "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM," in IEEE Transactions on Robotics, vol. 37, no. 6, pp. 1874-1890, Dec. 2021, doi: 10.1109/TRO.2021.3075644.(link)

10. R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," in IEEE Transactions on Robotics, vol. 33, no. 5, pp. 1255-1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103. (link)

11. J. Engel, J. Stückler and D. Cremers, "Large-scale direct SLAM with stereo cameras," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 1935-1942, doi: 10.1109/IROS.2015.7353631. (link)

12. GTSAM website (link)

13. C. Forster, M. Pizzoli and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China, 2014, pp. 15-22, doi: 10.1109/ICRA.2014.6906584. (link)

14. Relja Arandjelović, Petr Gronat, Akihiko Torii, Tomas Pajdla, Josef Sivic, “NetVLAD: CNN architecture for weakly supervised place recognition” arXiv:1511.07247v3 [cs.CV] (link)