Covariance Estimation for LOAM Algorithms Using Hausdorff Distance

Hacettepe University CMP755 Project Report

1st Onurcan YILMAZ
Department of Electrical and Electronics Engineering
Hacettepe University
N21238186

***Abstract*—In this project, I aim to have a variance estimation for the position output of the LOAM algorithm. I use the Hausdorff distance of the last lidar scan with respect to the point cloud map and publish it with the position calculated using the last scan. Later, I fed this output to an Extended Kalman Filter (EKF) and compared its result with algorithms output and another EKF output generated using algorithms output with constant variance.**

***Index Terms*—LiDAR, Mapping, LOAM, Hausdorff Distance, Extended Kalman Filter, ROS**

I. Introduction

The LiDAR Odometry and Mapping (LOAM) algorithm was first introduced by Ji Zhang and Sanjiv Singh in 2014 [1]. Since then, several algorithms have been published above their work like; LeGO- LOAM [2], A-LOAM [3], F-LOAM [4], SC-A-LOAM [5] etc.

The algorithm extract features from LiDAR scans i.e., surfaces and edges. Then, it uses Levenberg–Marquardt (LM) Optimization to match features of two consecutive LiDAR scans to estimate position and orientation change between them which is initial odometry data. When a new position is estimated, the last point cloud is published with the initial position estimate as the origin. Unfortunately, due to the nature of the algorithm, it cannot give a covariance matrix output. Because of that, these algorithms cannot be directly used with probabilistic algorithms like Kalman Filter, Monte Carlo Localization, or SLAM.

Hausdorff distance measures how far away two sets are from each other [6]. It is a metric used to measure the performance of an Iterated Closest Point algorithm [7]. It measures the minimum distance for each point in each cloud and returns the biggest value. If two identical point clouds are perfectly aligned, Hausdorff distance becomes 0.

In this project, I use the LeGO-LOAM-BOR [8] algorithm which is an optimized and more readable fork of the original LeGO-LOAM algorithm. To achieve this, I changed it to publish surface and edge point cloud maps. Later wrote a ROS node to transform the last point cloud origin to the last position estimation and measured the Hausdorff distance of the cloud. Then, I try to make a better estimation using the EKF of robot_localization [9] ROS package.

II. Background

A. Iterated Closest Point (ICP)

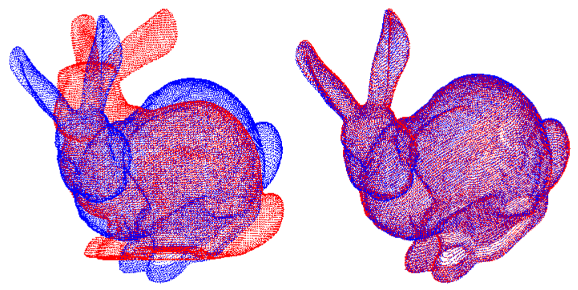

Fig. 1. Example of a shape before and after rigid alignment with ICP. [16]

Iterated Closest Point (ICP) is an algorithm to match two-point clouds. It finds the closest point for each point and calculates a better transformation on each iteration. Since it checks for all points, it is slow and unsuitable for mobile systems with limited computational capacity.

B. Hausdorf Distance

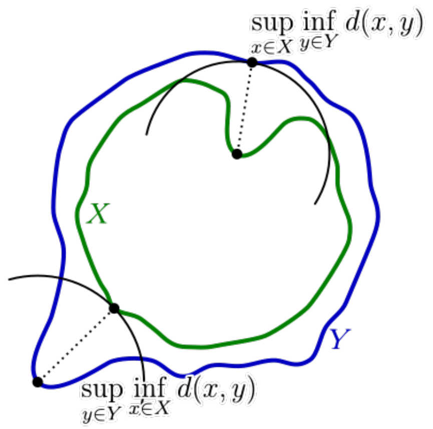
$\sup_{x\in X}\inf_{y\in Y} d(x,y)$
X
Y
$\sup_{y\in Y}\inf_{x\in X} d(x,y)$

Fig. 2. Components of the calculation of the Hausdorff distance between the green curve X and the blue curve Y. [7]

Hausdorff distance is a metric to measure how separated two point clouds are. It is simply the maximum possible distance

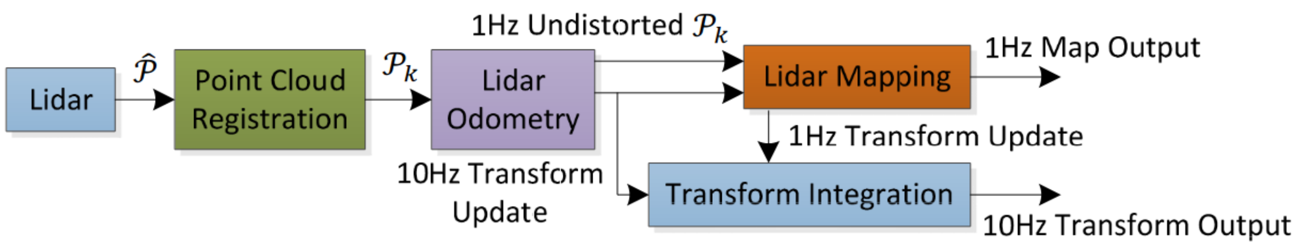

Fig. 3. Block diagram of LOAM algorithm. [2]

from any point of a given point set to any point of the other point set. as a result of this, it is used as a performance metric for the results of ICP algorithms. Its equation is;

$$d_H(\mathcal{X},\mathcal{Y}) = \max\{\sup_{x\in\mathcal{X}} \inf_{y\in\mathcal{Y}} d(x,y), \sup_{y\in\mathcal{Y}} \inf_{x\in\mathcal{X}} d(y,x)\} \quad (1)$$

where $d(a,b)$ can be any distance function like Euclidean or Manhattan.

C. LOAM

LOAM is a position estimation algorithm for LiDARs used on mobile robots. Its difference from ICP is, it extracts the feature point of each scan, which corresponds to surfaces and edges of the environment and uses these feature points to construct a nonlinear optimization problem. Then uses LM Optimization to solve it to estimate position. Since it uses fewer points for estimation it is faster and needs less computational power.

It also produces a map at a lower frequency and uses this map to increase position accuracy of initial position estimation, at the output with;

$$T^{world}_{tobe} = T^{world}_{current} * (T^{world}_{bef\_map})^{-1} * T^{world}_{aft\_map} \quad (2)$$

where $T^{world}_{current}$ is the initial odometry estimation received from the LiDAR Odometry node and $T^{world}_{bef\_map}$ is previous and $T^{world}_{aft\_map}$ last position received position estimation received from LiDAR Mapping node. [10]

D. LeGO-LOAM

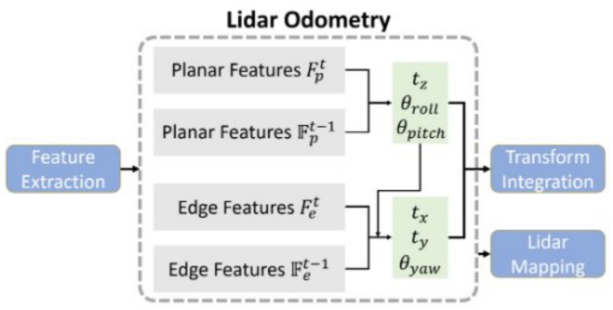
Lidar Odometry
Planar Features F_p^t
Planar Features $\mathbb{F}_p^{t-1}$
Edge Features F_e^t
Edge Features $\mathbb{F}_e^{t-1}$
t_z θ_{roll} θ_{pitch}
t_x t_y θ_{yaw}
Feature Extraction
Transform Integration
Lidar Mapping

Fig. 4. Block diagram of LeGO-LOAM odometry. [2]

LeGO-LOAM is a variant of the LOAM algorithm. It uses two-step LM Optimization. In the first step, It finds the best fit by using only edge features and ground plane to estimate a transformation for only z, $roll$, and $pitch$ axes. Then uses this transformation as a constraint and the rest of the surface features to find an estimate for x, y, and yaw axes. Thus, it decreases computational cost and increases speed.

In the project, I used another version of the original LeGO-LOAM algorithm which is named LeGO-LOAM-BOR [8]. It does not change anything on the algorithm side but only optimizes the code and increases readability.

III. Project Work

A. Changes on Original Code

I only changed the /MapOptimization node of the original code. Here I added two publishers, "/cov_map_corner_cloud" and "/cov_map_surf_cloud", both work at 2 Hz. Instead of using the whole map, I only published scans that have origins in 50 meters radius of the current position.

I also added Ouster OS1-64 LiDAR configuration to test it wit MulRan Dataset. [12]

B. Covariance Estimation

The main work of the project was on the node named /covarianceEstimation which subscribes to topics:

- /cov_map_corner_cloud: Point cloud of edge features retrieved from LiDAR Mapping node at 2Hz.
- /cov_map_surf_cloud: Point cloud of surface features retrieved from LiDAR Mapping node at 2Hz.
- /laser_cloud_corner_last: Point cloud of edge features retrieved from Feature Extraction node at 10Hz.
- /laser_cloud_surf_last: Point cloud of surface features retrieved from Feature Extraction node at 10Hz.
- /integrated_to_init: Final odometry result retrieved from Transform Integration node at 10Hz.

/covarianceEstimation node simply waits for /laser_cloud_corner_last, /laser_cloud_surf_last and /integrated_to_init messages and sets corresponding received flags. When all messages are received, it checks their timestamps which are inherited from the sensor message. If all timestamps are the same, it transforms the last point cloud's origin received position. If not, deletes older messages and sets their flags to False. Then it calculates Hausdorff

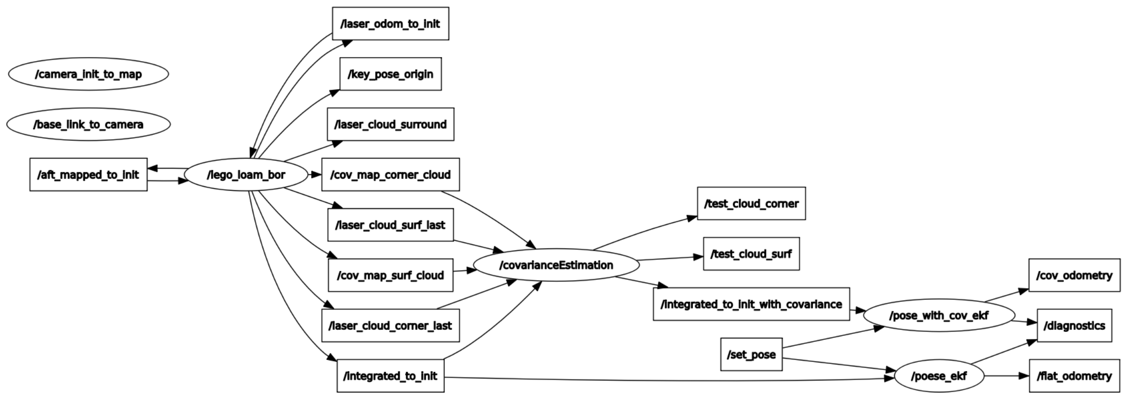

Fig. 5. ROS node and topic graph of project

distance for surface and edge point clouds separately using equations 3 and 4 then returns to the bigger one.

$$d_E(\mathcal{L}_E, \mathcal{M}_E) = \sup_{l \in \mathcal{L}_E} \inf_{m \in \mathcal{M}_E} d(l, m). \tag{3}$$

$$d_S(\mathcal{L}_S, \mathcal{M}_s) = \sup_{l \in \mathcal{L}_S} \inf_{m \in \mathcal{M}_S} d(l, m). \tag{4}$$

Where L_E and L_S are the last scan's edge and surface clouds and M_E and M_S are the last received edge and surface map clouds respectively.

Notice that, the actual Hausdorff distance is taken bidirectionally but due to the problem's nature, I only searched scan points in the map. Also, when LiDAR starts to see a new area first time, this value increases significantly. To overcome this, I added a condition to the calculation. It finds the 5th closest point and if its distance is more than 1 meter, it does not consider it for distance calculation. This condition is also used in /MapOptimization node to eliminate outlier points.

Finally, I set the calculated distance as the diagonal values of the covariance matrix and publish the new message to /integrated_to_init_with_covariance topic.

The code can be found in GitHub [9]

C. Extended Kalman Filter

To check the system performance, I created two EKF nodes using robot_localization [9] ROS package.

The first one subscribes to /integrated_to_init topic and publishes its result to /flat_odometry topic. It uses packages default 10^{-2} value at covariance matrix, and process noise.

The second one subscribes to /integrated_to_init_with_covariance topic and publishes its result to /cov_odometry topic. It uses packages default 10^{-2} value at process noise.

The launch and parameter files can be found here [11]

D. Loop Closure

LeGO-LOAM has its own loop closure and optimization module which is based on iSAM2 [13] algorithm. However, it is not working well even the key frame search radius increased significantly. Also there is a implementation of Scan Context [15] on LOAM named SC-LeGO-LOAM [14]. I attempted to include it to my code but unfortunately couldn't succeeded.

IV. Results

I used the LiDAR point cloud of the MulRan Dataset as my input for the results.

A. Current Position

In the first experiment, I used current position estimation as the input of the EKF. In the Figure 6, results of Ground Truth, Mapping node and EKF nodes can be seen.

I also calculated absolute and RMS Euclidean distance errors of EKF outputs with respect to Mapping node output and Ground Truth in Figure 7.

In this scenario, performance results of EKF output usind Hausdorff distance is worse. However, notice that this errors were respect to mapping node output, not the ground truth.

B. Transformation from Last Position

In the second experiment, I used transformation from last position as the input of the EKF. In the Figure 8, results of Ground Truth, Mapping node and EKF nodes can be seen.

I also calculated absolute and RMS Euclidean distance errors of EKF outputs with respect to Mapping node output and Ground Truth in Figure 9.

In this scenario, performance results of EKF output usind Hausdorff distance are better except Z axis. However, notice that this errors were respect to mapping node output, not the ground truth.

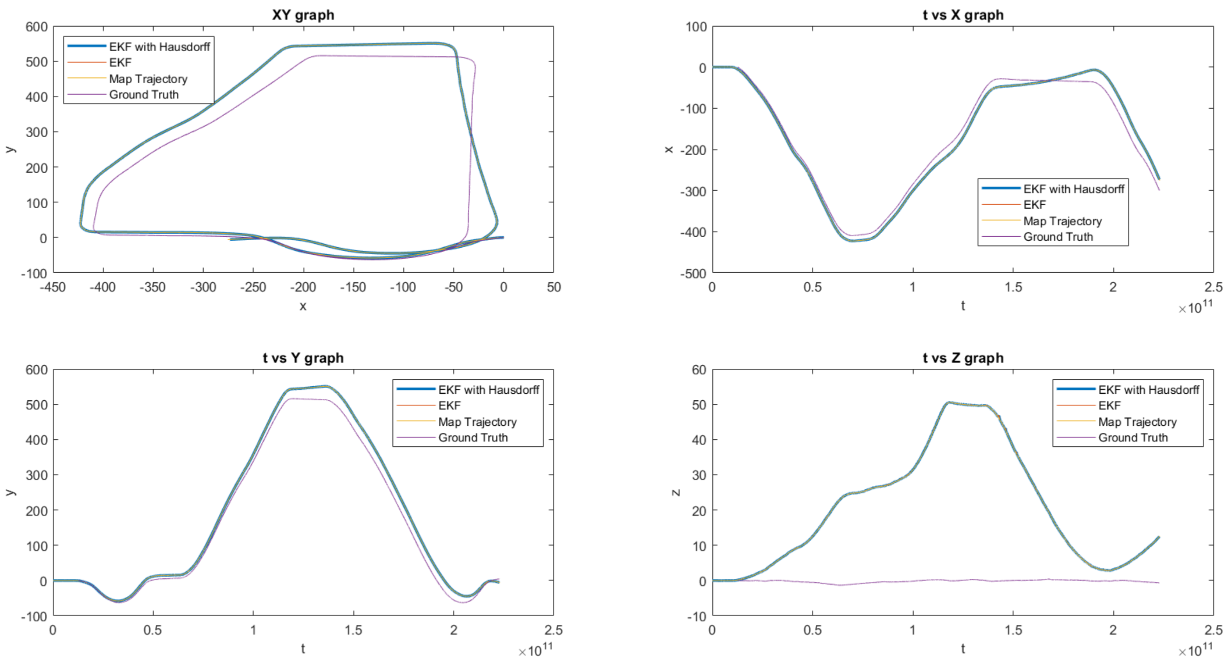

Fig. 6. Results of Mapping node and EKF nodes with position input.

C. *Transformation from Last Position corrected by Map output*

Finally, I used transformation from last position as the input of the EKF also supported it with position received from map. In the Figure 10, results of Ground Truth, Mapping node and EKF nodes can be seen.

I also calculated absolute and RMS Euclidean distance errors of EKF outputs with respect to Mapping node output and Ground Truth in Figure 11.

In this scenario, performance results of both EKF outputs are similar.

References

[1] J. Zhang and S. Singh, "Loam: Lidar Odometry and mapping in real-time," Robotics: Science and Systems X, 2014.

[2] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[3] HKUST-Aerial-Robotics, "HKUST-Aerial-Robotics/A-LOAM: Advanced implementation of loam," GitHub. [Online]. Available: https://github.com/HKUST-Aerial-Robotics/A-LOAM. [Accessed: 02-Nov- 2022].

[4] H. Wang, C. Wang, C.-L. Chen, and L. Xie, "F-loam: Fast lidar odometry and mapping," 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2021.

[5] Gisbi-Kim, "Gisbi-Kim/SC-A-LOAM: Robust lidar slam with a versatile plug-and-play loop closing and pose-graph optimization.," GitHub. [Online]. Available: https://github.com/gisbi-kim/SC-A-LOAM. [Accessed: 02-Nov-2022].

[6] F. Huang, W. Wen, J. Zhang, and L.-T. Hsu, "Point wise or feature wise? A benchmark comparison of publicly available lidar odometry algorithms in urban canyons," IEEE Intelligent Transportation Systems Magazine, pp. 2–20, 2022.

[7] "Hausdorff distance," Wikipedia, 29-Aug-2022. [Online]. Available: https://en.wikipedia.org/wiki/Hausdorff_distance. [Accessed: 02-Nov-2022].

[8] Facontidavide, "FACONTIDAVIDE/Lego-loam-Bor: Lego-loam-Bor: Optimized lidar odometry and mapping," GitHub. [Online]. Available: https://github.com/facontidavide/LeGO-LOAM-BOR. [Accessed: 09-Jan-2023].

[9] O. YILMAZ, "URTII/Lego-loam-CMP755: Lego-loam-Bor: Optimized lidar odometry and mapping," GitHub. [Online]. Available: https://github.com/Urtii/LeGO-LOAM-CMP755. [Accessed: 09-Jan-2023].

[10] "LeGo-LOAM" GWH Blog, Jul. 31, 2020. https://gutsgwh1997.github.io/2020/08/01/LeGo-LOAM%E4%B8%AD%E7%9A%84transformFusion%E8%8A%82%E7%82%B9-%E7%9B%B8%E5%85%B3%E6%95%B0%E5%AD%A6%E5%85%AC%E5%BC%8 (accessed Jan. 08, 2023).

[11] O. YILMAZ, "robot_localization," GitHub, Dec. 12, 2022. https://github.com/Urtii/robot_localization_urt (accessed Jan. 08, 2023).

[12] G. Kim, Y. S. Park, Y. Cho, J. Jeong and A. Kim, "MulRan: Multimodal Range Dataset for Urban Place Recognition," 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 6246-6253, doi: 10.1109/ICRA40945.2020.9197298.

[13] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard and F. Dellaert, "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," 2011 IEEE International Conference on Robotics and Automation, 2011, pp. 3281-3288, doi: 10.1109/ICRA.2011.5979641.

[14] Irapkaist, "IRAPKAIST/SC-Lego-loam: LIDAR SLAM: SCAN CONTEXT + LEGO-loam," GitHub. [Online]. Available: https://github.com/irapkaist/SC-LeGO-LOAM. [Accessed: 09-Jan-2023].

[15] G. Kim, S. Choi and A. Kim, "Scan Context++: Structural Place Recognition Robust to Rotation and Lateral Variations in Urban Environments," in IEEE Transactions on Robotics, vol. 38, no. 3, pp. 1856-1874, June 2022, doi: 10.1109/TRO.2021.3116424.

[16] "PS 1 - rigid shape registration," PS 1 - Rigid alignment. [Online]. Available: https://www.lix.polytechnique.fr/ maks/Verona_MPAM/TD/TD1/. [Accessed: 09-Jan-2023].

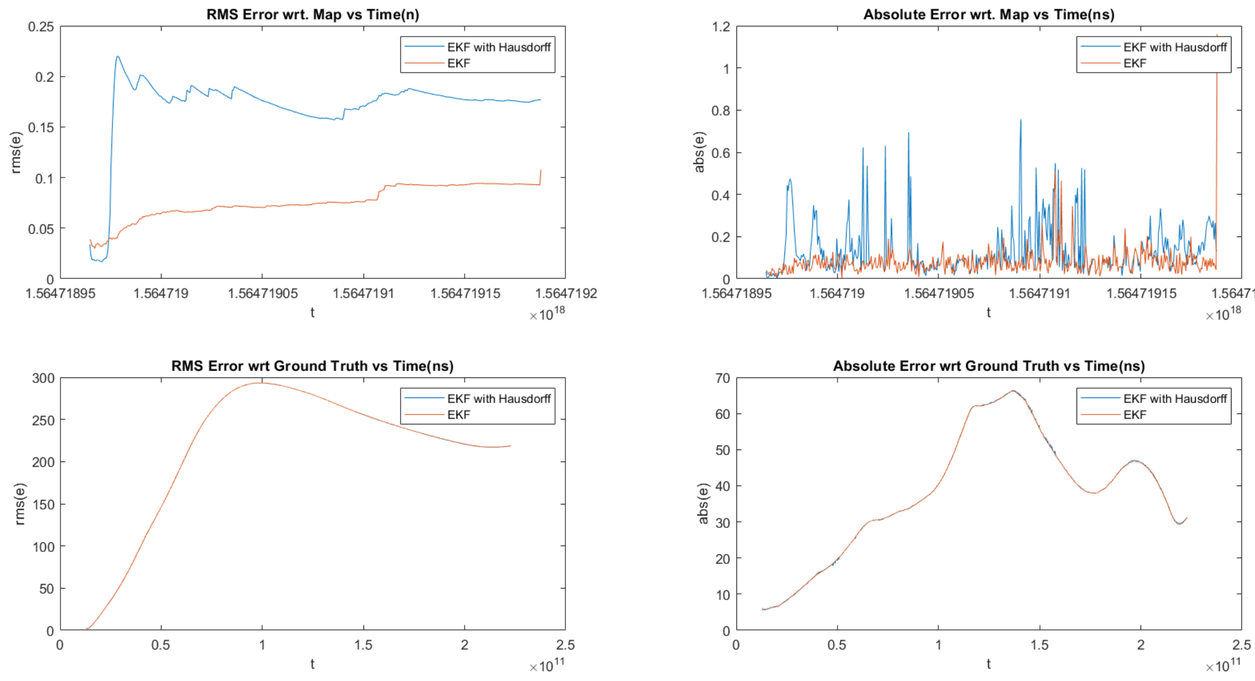

Fig. 7. Absolute Euclidean distance errors with position input wrt. Mapping position vs. POSIX time (ns).

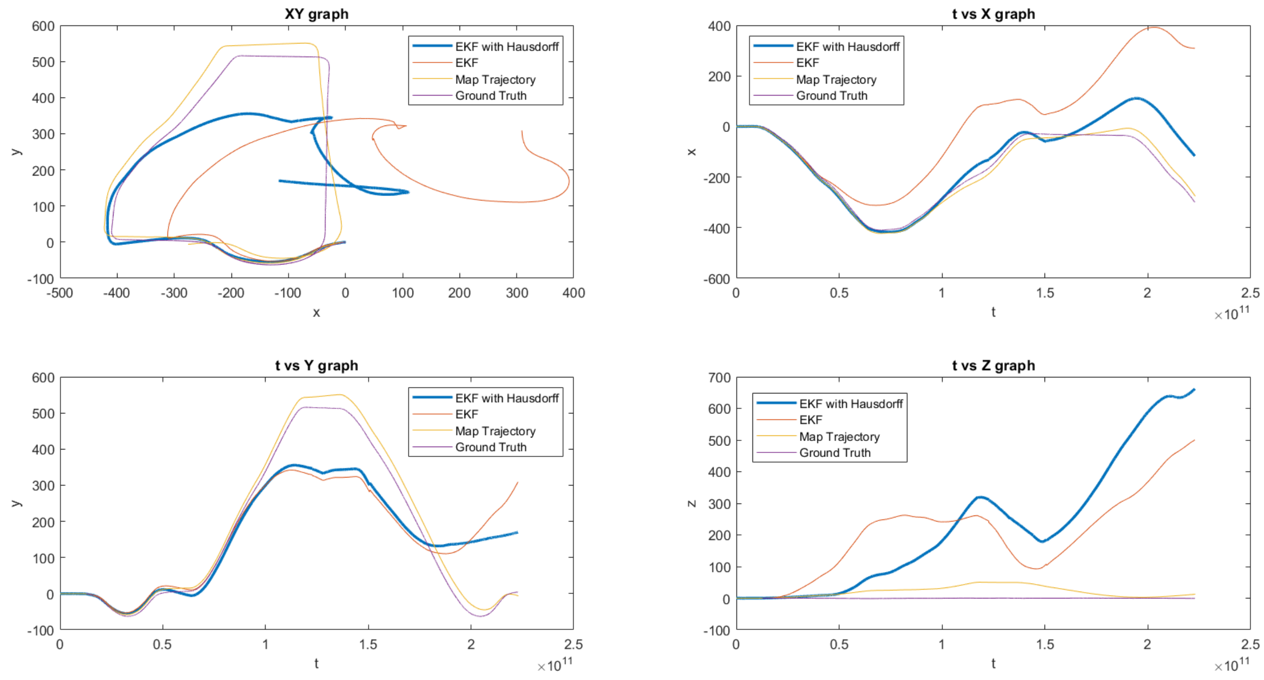

Fig. 8. Results of Mapping node and EKF nodes with transformation input.

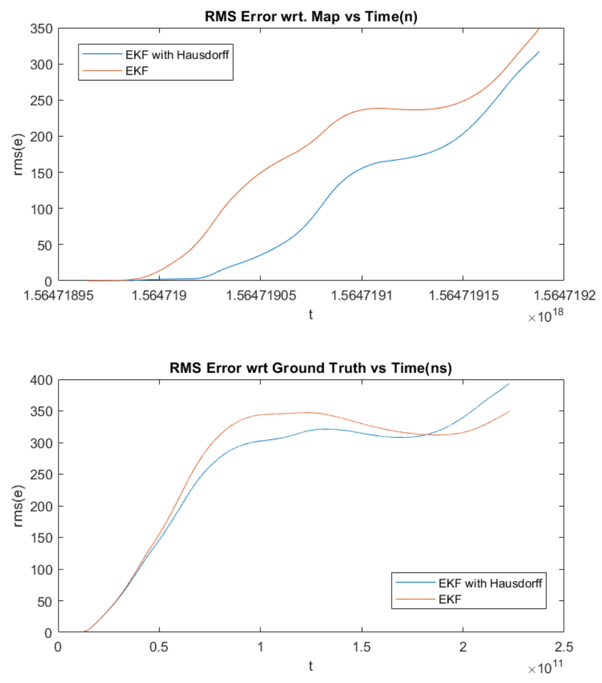

Fig. 9. Absolute Euclidean distance errors with transformation input wrt. Mapping position vs. POSIX time (ns).

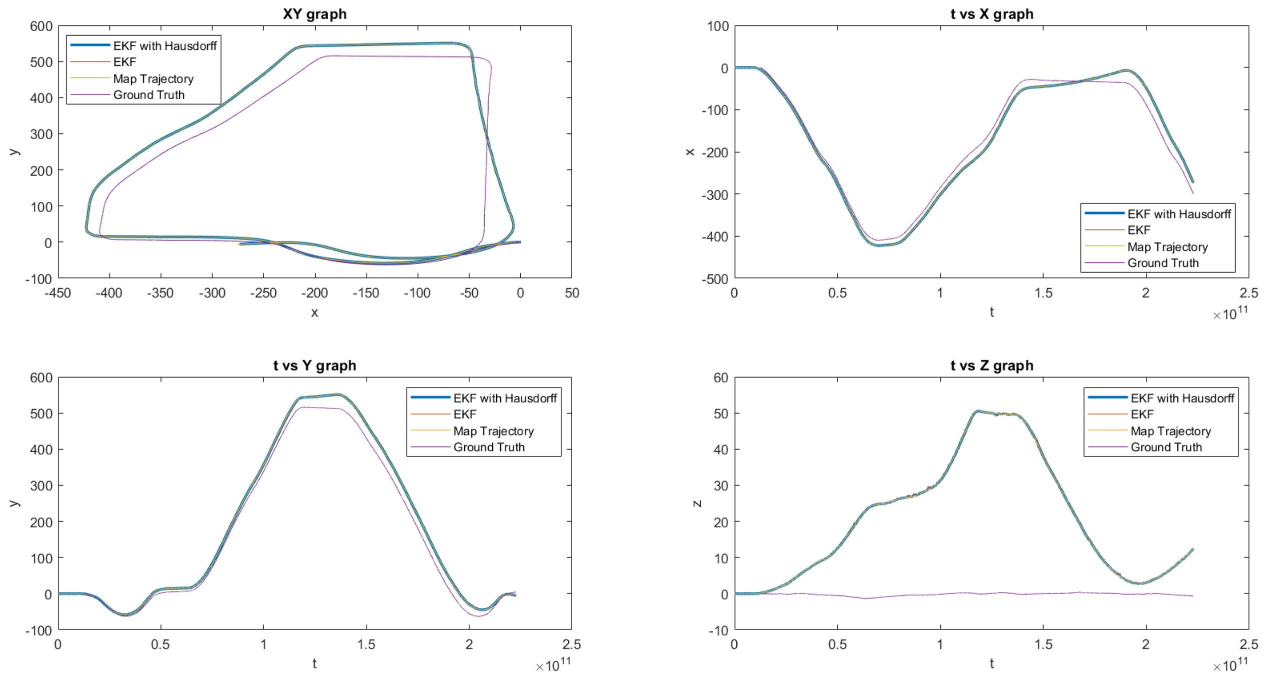

Fig. 10. Results of Mapping node and EKF nodes with transformation input.

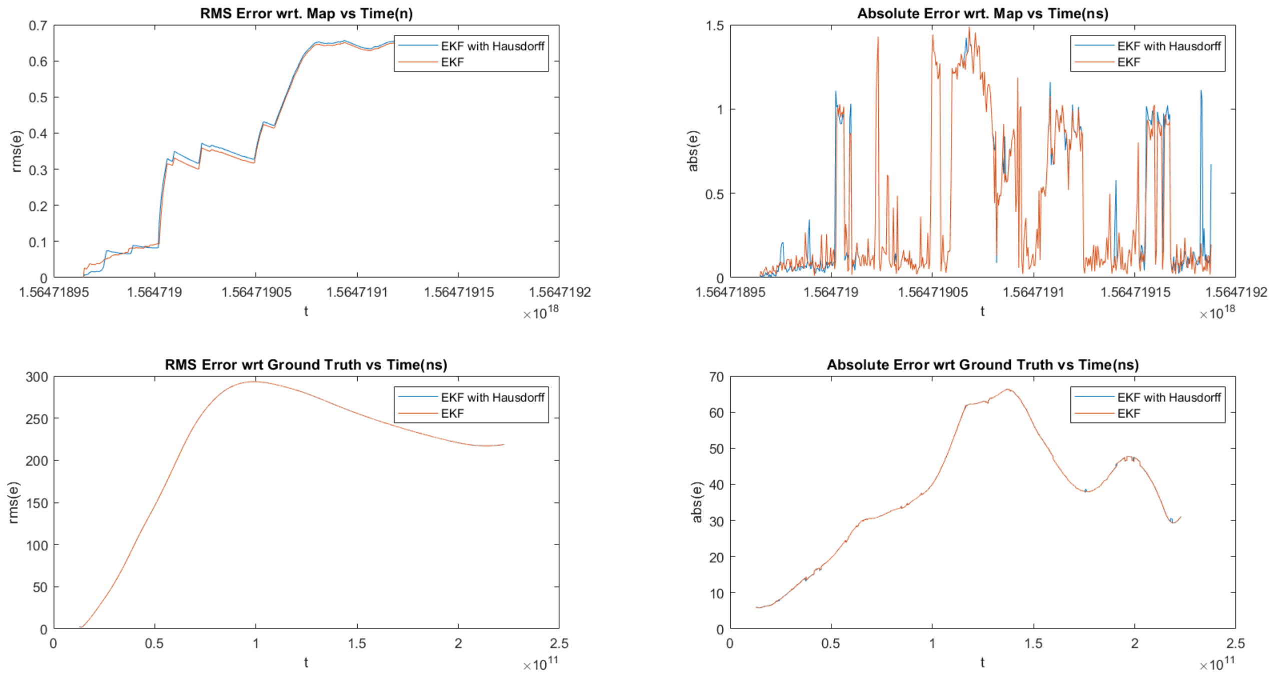

Fig. 11. Absolute Euclidean distance errors with transformation input wrt. Mapping position vs. POSIX time (ns).