Mapping and Perception for an Autonomous Robot
Project 3

Nadav Marciano 305165698

Part A: EKF-SLAM

a. The given *get_odometry* and *calc_trajectory_from_odometry* functions were used to process the odometry data, resulting in the following ground truth trajectory plot:

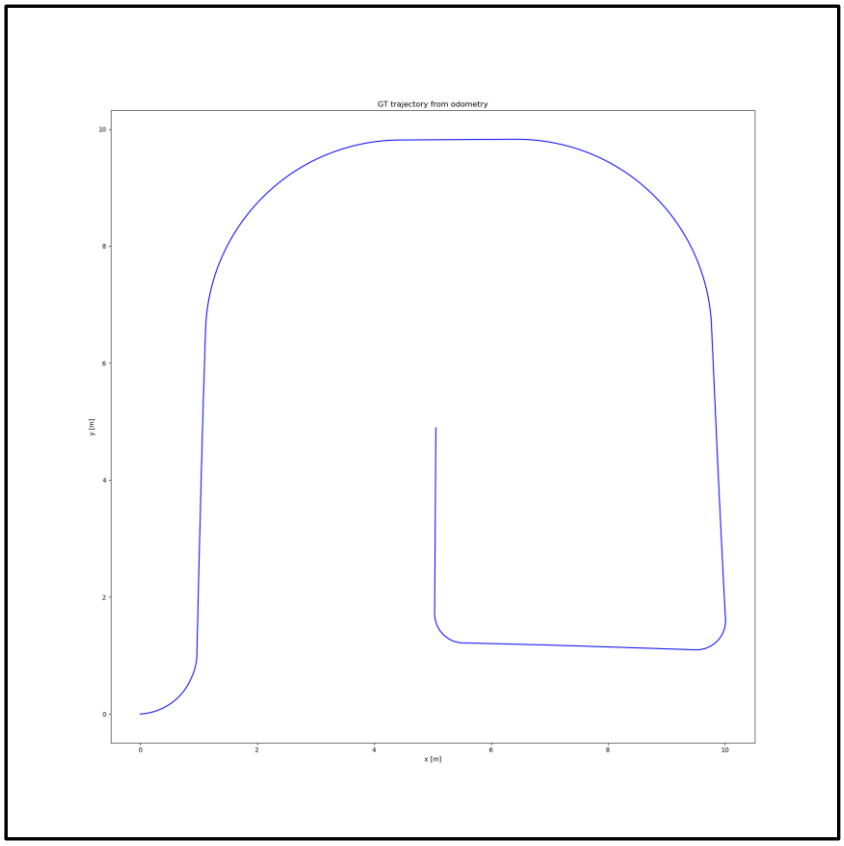


Figure 1 - Ground-truth trajectory from odometry

The route appears almost like a complete circle, except that towards the end it shifts northward.

b. According to the requirement we added Gaussian noise in the movement model:
$\sigma_{rot1} = 0.01$
$\sigma_{rot2} = 0.01$
$\sigma_{trans} = 0.1$

Here is the resulting ground truth trajectory plot with added noise:

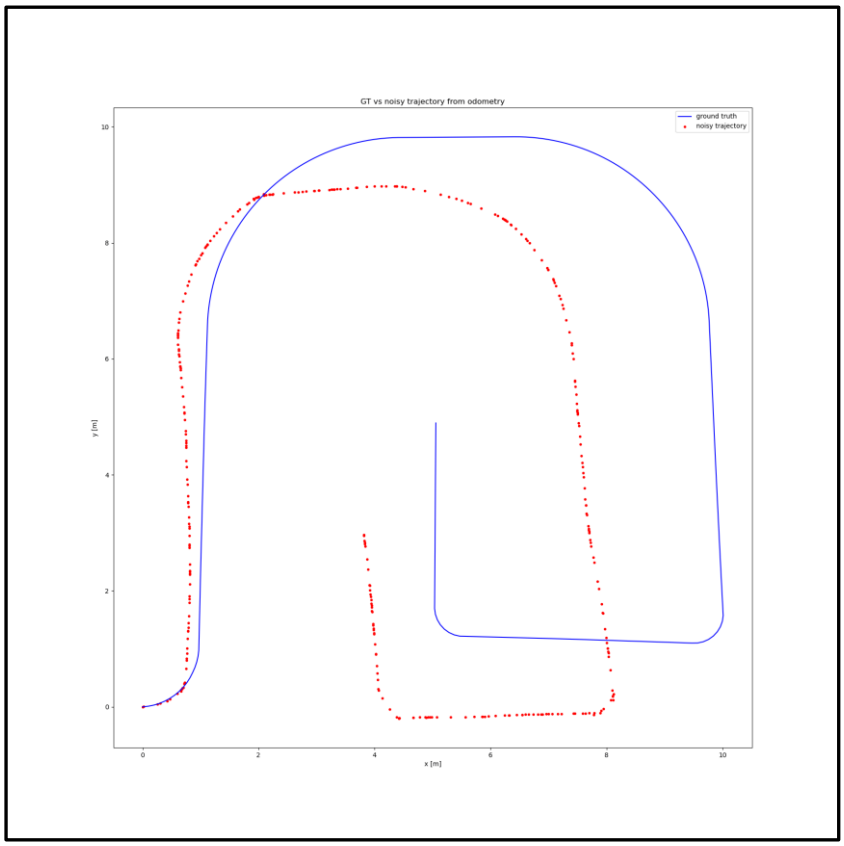


Figure 2 - Ground- truth vs noisy trajectory from odometry

The noise affected the path regarding the position, but the shape remained intact. The plot illustrates the impact of noise on the trajectory. While the actual positions deviate from the ground truth due to the noise, the general pattern of the movement is still recognizable.

c. To begin the EKF-SLAM process, we need to initialize the mean vector μ_0 and the covariance matrix Σ_0. These represent our initial belief about the robot's state and the landmarks in the environment.

Initial Mean Vector μ_0:

- The mean vector μ_0 is a 2N+3×1 vector, where N is the number of landmarks.
- The first 3 components of μ_0 correspond to the initial pose of the robot, which includes its x position, y position, and orientation θ.
- The remaining 2N components correspond to the positions of the N landmarks, with each landmark having an x and y position.

Initial Covariance Matrix Σ_0:

- The covariance matrix Σ_0 is a (2N+3)×(2N+3) matrix.
- The first 3×3 block of Σ_0 represents the uncertainty in the robot's initial pose.
- The remaining 2N×2N block represents the uncertainty in the positions of the landmarks.

The initial mean vector μ_0 was set to zero, indicating no initial knowledge about the robot's pose or the landmarks' positions. The initial covariance matrix Σ_0 was initialized with appropriate values

representing high uncertainty for the robot's pose and the landmarks' positions, reflecting the lack of prior information.

d. Simultaneous Localization and Mapping (SLAM) is a computational technique used in robotics and autonomous systems to construct or update a map of an unknown environment while simultaneously keeping track of the robot's location within it.

The process model in SLAM includes several key steps:

Odometry motion model: $u = (\delta_{rot1}, \delta_{trans}, \delta_{rot2})$ $u \sim N(0, \Sigma)$

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} \delta_{trans} \cos(\theta_{t-1} + \delta_{rot1}) \\ \delta_{trans} \sin(\theta_{t-1} + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{bmatrix}$$

Observations: $z_t = h(\bar{\mu}_t) + \varepsilon_t$ $\varepsilon_t \sim N(0, R_t)$

$$\bar{\mu}_t = [\, x \;\; y \;\; \theta \;\; m_{j,x} \; m_{j,y}]$$

$$h(\bar{\mu}_t) \begin{cases} \hat{z}_r^1 = \sqrt{(m_{1,x} - x)^2 + (m_{1,y} - y)^2} \\ \hat{z}_\emptyset^1 = \tan^{-1}(m_{1,y} - y / m_{1,x} - x) - \bar{\mu}_{t,\theta} \end{cases}$$

Initialize Parameters: initial state μ_0, covariance matrix Σ_0, measurement covariance Q, and noise covariance R.

$$\mu_0 = [x_0 \;\; y_0 \;\; \theta_0 \;\; 0 \;\; . \;\; . \;\; . \;\; 0]$$

$$Q = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\emptyset^2 \end{bmatrix}$$

$$\Sigma_0 = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sigma_y^2 & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sigma_\theta^2 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \infty & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \infty \end{bmatrix}$$

$$\tilde{R}_t = \begin{bmatrix} {\sigma_{rot1}}^2 & 0 & 0 \\ 0 & {\sigma_{trans}}^2 & 0 \\ 0 & 0 & {\sigma_{rot2}}^2 \end{bmatrix}$$

EKF SLAM prediction: we estimate the state and covariance matrix for the next time step based on the current state estimate and the motion model, accounting for the process noise.

$$F_x = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$$

$$\bar{\mu}_t = \bar{\mu}_{t-1} + F_X^T \begin{bmatrix} \delta_{trans} \cos(\theta_{t-1} + \delta_{rot1}) \\ \delta_{trans} \sin(\theta_{t-1} + \delta_{rot1}) \\ \delta_{rot1} + \delta_{rot2} \end{bmatrix}$$

$$G_t = I + F_X^T \begin{bmatrix} 0 & 0 & -\delta_{trans} \sin(\theta_{t-1} + \delta_{rot1}) \\ 0 & 0 & \delta_{trans} \cos(\theta_{t-1} + \delta_{rot1}) \\ 0 & 0 & 0 \end{bmatrix} \qquad \bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + F_X^T R_t^x F_x$$

EKF SLAM correction: the predicted state is updated using the latest observation, reducing state uncertainty by incorporating the measurement information.

$$Q_t = \begin{bmatrix} \sigma_r^2 & 0 \\ 0 & \sigma_\emptyset^2 \end{bmatrix}$$

$for\ all\ observed\ features\ Z_t^i = \left(r_t^i, \emptyset_t^i\right)^T do$:

$j = C_t^i$

$if\ landmark\ j\ never\ seen\ before$:

$$\begin{bmatrix} \bar{\mu}_{j,x} \\ \bar{\mu}_{j,y} \end{bmatrix} = \begin{bmatrix} \bar{\mu}_{t,x} \\ \bar{\mu}_{t,y} \end{bmatrix} + \begin{bmatrix} r_t^i \cos(\emptyset_t^i + \bar{\mu}_{t,\theta}) \\ r_t^i \sin(\emptyset_t^i + \bar{\mu}_{t,\theta}) \end{bmatrix} \qquad end\ if;$$

$$\delta = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} = \begin{bmatrix} \bar{\mu}_{j,x} - \bar{\mu}_{t,x} \\ \bar{\mu}_{j,y} - \bar{\mu}_{t,y} \end{bmatrix}$$

$$q = \delta^T \delta$$

$$\hat{Z}_t^t = \begin{bmatrix} \sqrt{q} \\ atan2(\delta_y, \delta_x) - \bar{\mu}_{t,\theta} \end{bmatrix}$$

$$F_{x,j} = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 \end{bmatrix}$$

$$H_t^i = \frac{1}{q} \begin{bmatrix} -\sqrt{q}\delta_x & -\sqrt{q}\delta_y & 0 & \sqrt{q}\delta_x & \sqrt{q}\delta_y \\ \delta_y & -\delta_x & -q & -\delta_y & \delta_x \end{bmatrix}$$

$$K_t^i = \bar{\Sigma}_t H_t^{i^T} (H_t^i \bar{\Sigma}_t H_t^{i^T} + Q_t)^{-1}$$

$$\bar{\mu}_t = \bar{\mu}_t + K_t^i \left(Z_t^i - \hat{Z}_t^t\right)$$

$$\bar{\Sigma}_t = \left(I - K_t^i H_t^i\right) \bar{\Sigma}_t \qquad end\ for;$$

$$\mu_t = \bar{\mu}_t$$

$$\Sigma_t = \bar{\Sigma}_t$$

e. Result analysis ground-truth and estimated results:

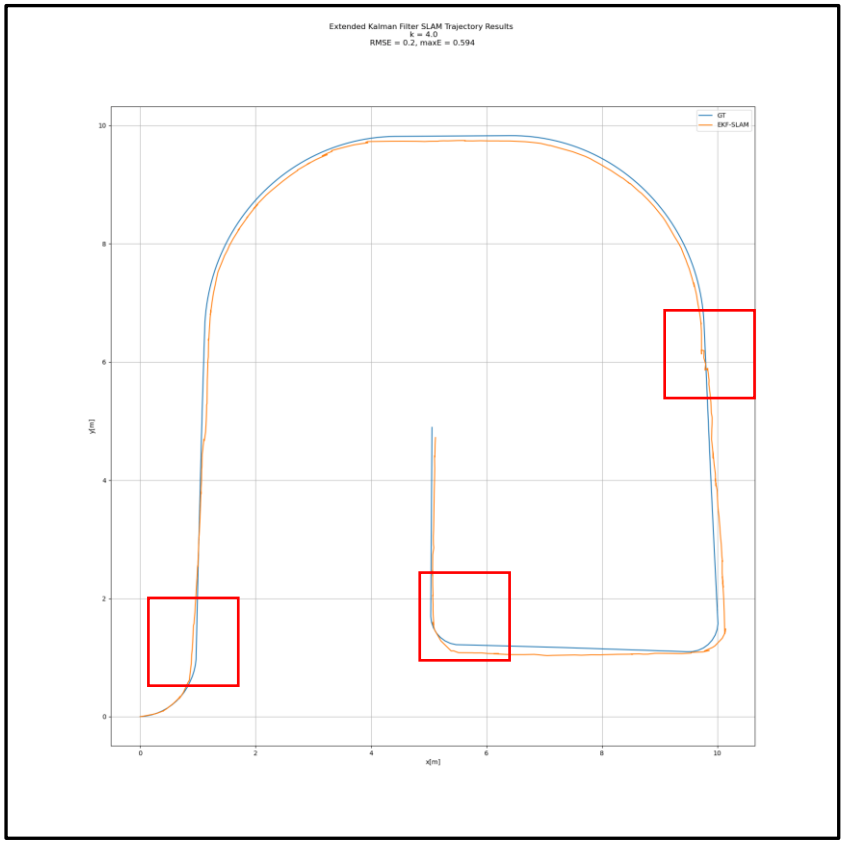
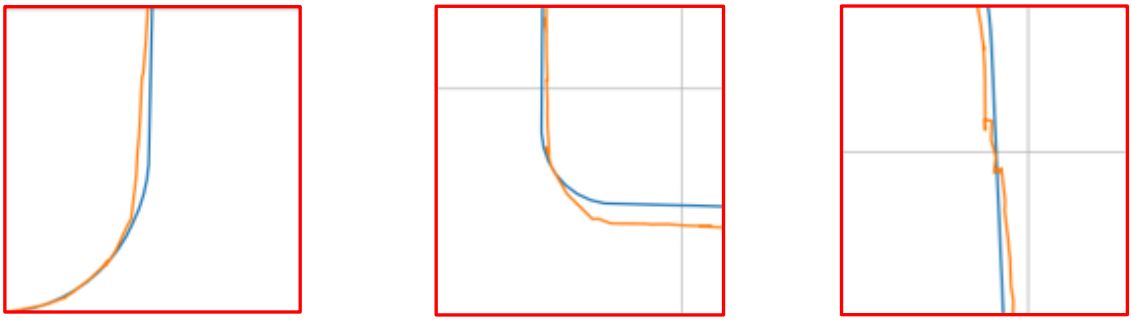


Figure 3 - Extended Kalman SLAM filter results vs GT



The estimated path of the robot demonstrates a strong alignment with the ground-truth trajectory, showing that the SLAM system effectively follows the true path of the robot. However, minor deviations are observed, which may be attributed to several factors such as sensor noise, environmental conditions, or limitations in the SLAM algorithm. While the positional estimates (x, y) are closely aligned with the ground-truth values, the angular estimates (yaw) exhibit noticeable discrepancies. This indicates that the SLAM system faces challenges in accurately estimating the robot's orientation.

Explanation of Changes in the Size of the Ellipse:

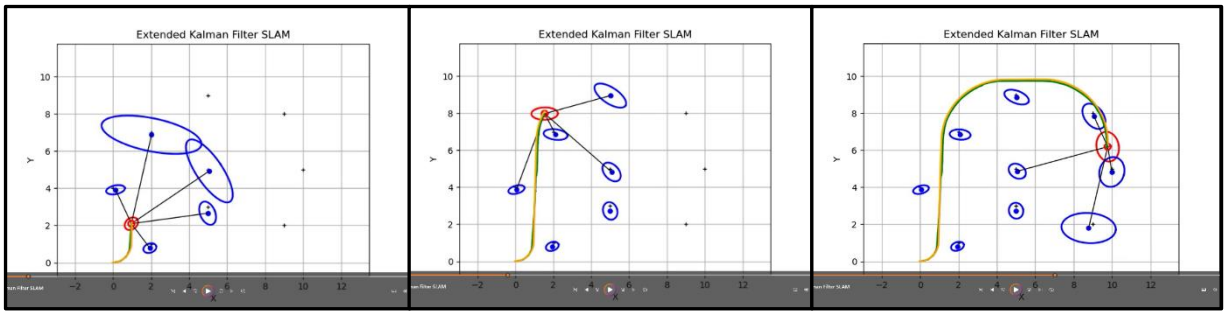


Figure 4 - Visualizing uncertainty reduction in landmark estimations: snapshots from the SLAM animation process

The animation shows vividly how the uncertainty values of the landmarks change, as represented by the diagonal values of the Σ matrix. Initially, when a landmark is first observed, the calculated position carries a high degree of uncertainty. However, with repeated observations of the same landmark, our estimations improve, causing the ellipse to shrink. This indicates that the SLAM algorithm is becoming more confident in the robot's position and the landmarks. when approaching or observing landmarks, the uncertainty might temporarily increase due to measurement errors or noisy observations. this can cause the ellipse to expand, reflecting increased uncertainty.

The minimum values of maxE and RMSE achieved:
maxE: 0.594 [m] , **RMSE:** 0.2[m]

The minimum values of maxE and RMSE achieved in our analysis indicate that the SLAM algorithm performs with high accuracy and reliability. The low values suggest that the algorithm can closely follow the ground truth, with minimal deviation even in the worst-case scenario. This level of precision is essential for applications requiring precise navigation and mapping, such as autonomous driving and robotic exploration.

The analysis of the estimation errors for the X, Y, and Theta axes reveals the following results:

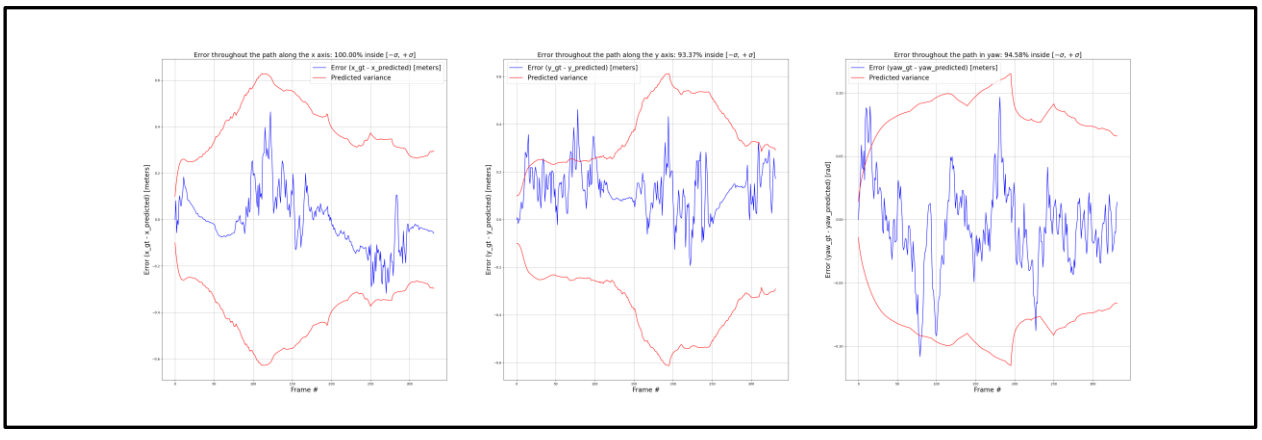


Figure 5 – State vector error - GT – EKF_SLAM

These results indicate that the estimation errors significantly exceed the expected Gaussian distribution of around 68%. I attempted to add an additional noise matrix. Sometimes, this approach can allow the

algorithm to converge better. However, the results did not improve. The maximum error remained greater than 1, whereas our goal is to minimize the maximum error to below 1 and the average error to below 0.3.
To further refine the algorithm, the parameter K was investigated over a range of values from 1 to 40. It was observed that up to K=10, the maximum error remained below 1, beyond which the results were not relevant. Subsequently, I tested K values ranging from 1 to 10 in increments of 0.1 to determine the optimal K. A comprehensive table displaying these findings is included in the accompanying Excel document. Based on the analysis, K=4 was selected as it provided the lowest RMSE value.

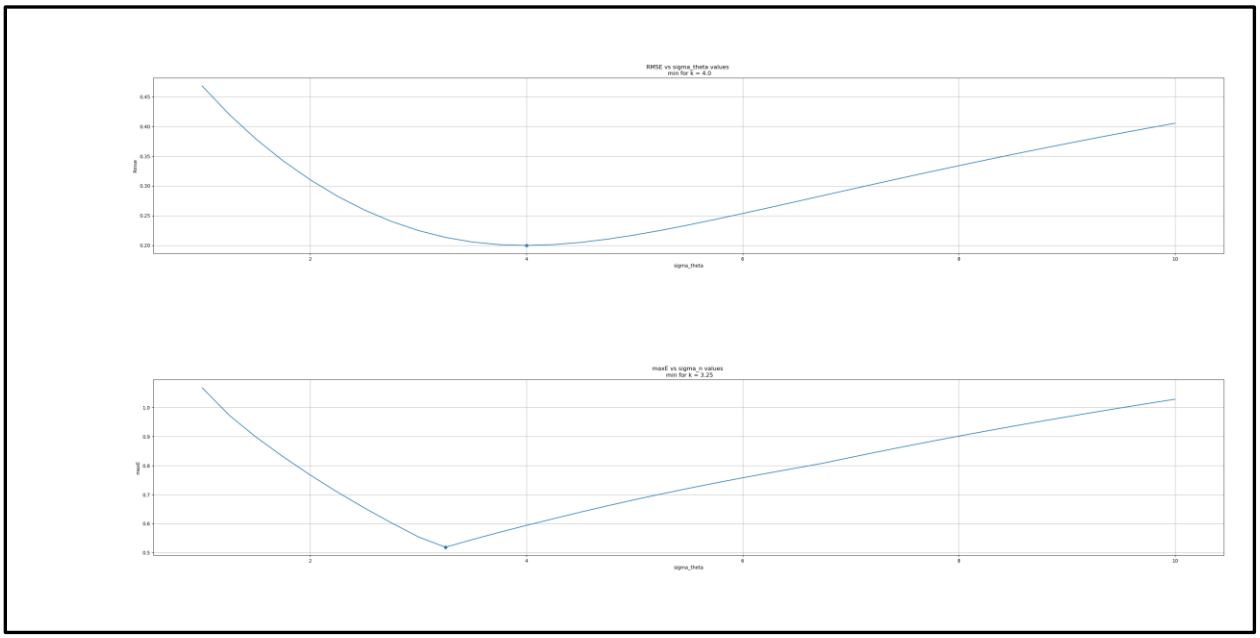


Figure 6 – Extended Kalman filter SLAM calibration

To further analyze the estimation accuracy, we selected two landmarks and performed a detailed analysis. Below are the plots showing the ground truth (GT) versus estimated values for each landmark, along with the corresponding sigma values (± sigma) represented on the graphs.

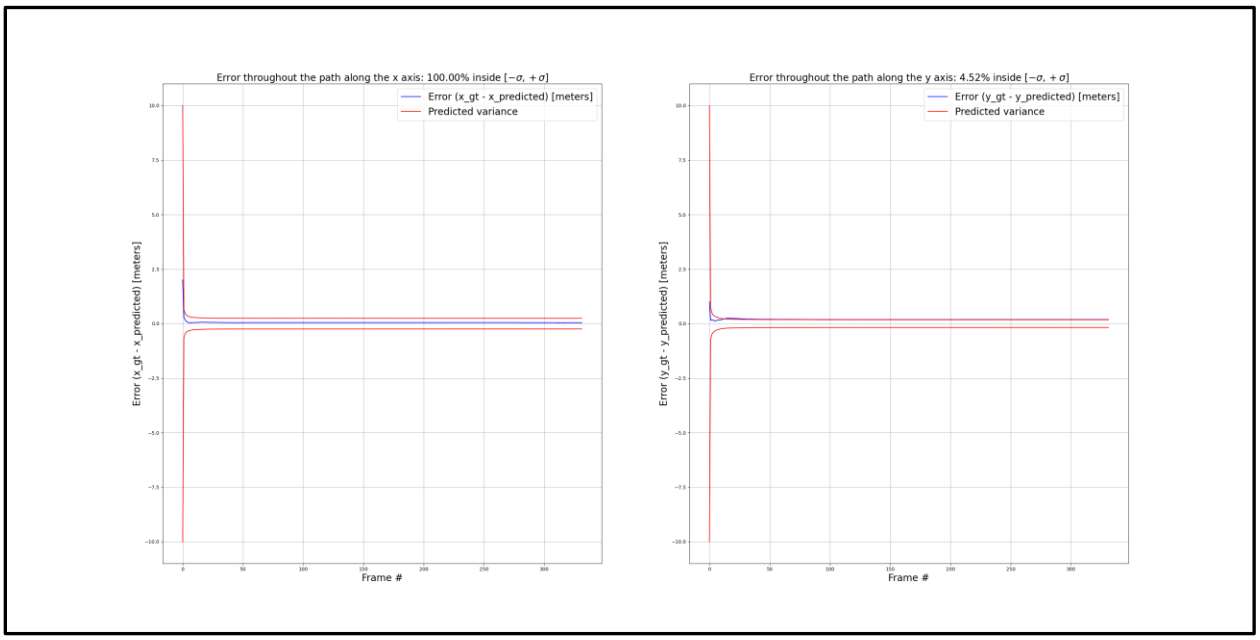


Figure 7 – landmark1 Error- GT – EKF SLAM

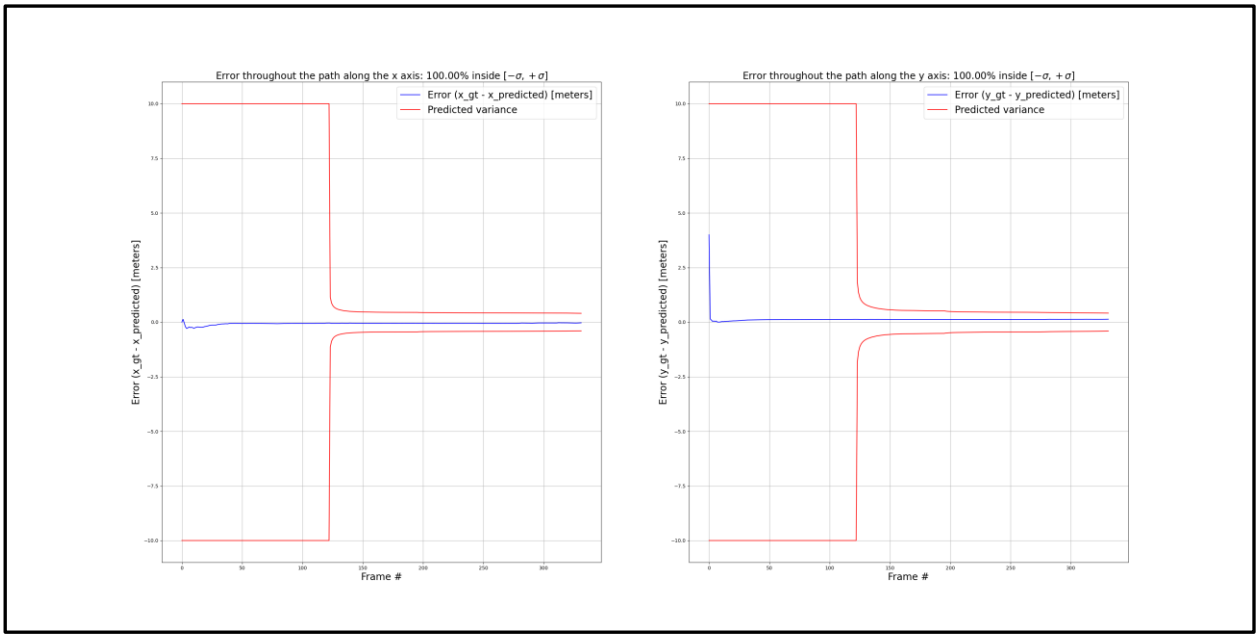


Figure 8 – landmark2 Error- GT – EKF SLAM

The analysis of the graphs for Landmark 1 and Landmark 2 reveals a significant initial estimation error, represented by a large sigma value. Over time, as the robot continues to encounter and observe these landmarks, the estimation error decreases, and the sigma values converge towards the actual positions of the landmarks. This behavior is akin to a funnel effect, where the uncertainty narrows down, and the confidence in the estimated positions increases.

Part B: ICP- Iterative Closet Points

a. Basic ICP Algorithm (Vanilla) Process Description:
The Iterative Closest Point (ICP) algorithm is a fundamental method for aligning two-point clouds. Here is a detailed description of the process:

- **Determine Corresponding Points**: For each point p_i in the source point cloud (frame 1), find the closest point x_i in the target point cloud (frame 2) using a K-D tree for efficient nearest neighbor search.

$$tree = KDTree(b)$$

- **Subtract Center of Mass from Each Subset**: Compute the centroids (center of mass) of both the source p_i and target x_i point sets:

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \qquad\qquad \mu_p = \frac{1}{N_p}\sum_{i=1}^{N_p} p_i$$

Subtract the respective centroids from each point set before calculating the transformation:

$$X' = \{x_i - \mu_x\} = \{x_i'\} \qquad\qquad p' = \{p_i - \mu_p\} = \{p_i'\}$$

- **Compute Rotation R and Translation t via SVD**: Construct a covariance matrix W using the centered point sets:

$$W = \sum_{i=1}^{N_p} x_i' \, p_i'^T$$

Perform Singular Value Decomposition (SVD) on the covariance matrix to obtain matrices U, Σ, and V:

$$W = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} V^T \qquad where \;\; U, V \in R^{3X3} \; are \; unitary$$

$$and \; \sigma_1 \geq \sigma_2 \geq \sigma_3 \; are \; the \; singular \; values \; of \; W$$

Compute the rotation matrix R: $R = VU^T$

Compute the translation vector t by finding the difference between the centroids of the target and source point sets after applying the rotation: $t = \mu_p - R\mu_x$

- **Apply R and t to the Points of the Set to be Registered**: Form the transformation matrix T from the rotation matrix R and translation vector t:

$$T = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}$$

Apply the transformation matrix to the source point cloud to align it closer to the target point cloud: $p_i = Rx_i + t$

- **Compute the Error E(R,t)**: Calculate the mean square error (MSE) between the transformed point cloud and the target point cloud:

$$E(R,t) = \frac{1}{N_x} \sum_{i=1}^{N_x} ||p_i - Rx_i + t||^2$$

- **Iterate Until Convergence**: If the error has decreased and is greater than a predefined threshold, repeat steps above. Continue iterating until the error converges or a maximum number of iterations is reached.

- **Stop and Output Final Alignment:** Once the error converges or the maximum number of iterations is reached, stop the algorithm. Output the aligned point cloud.

b. Before applying the Iterative Closest Point (ICP) algorithm, we show the figures of the point clouds from frames 5 and 9. The vehicle's path along a narrow road without dynamic objects between these two frames. The road is lined with trees and includes a static object, a lamp post.

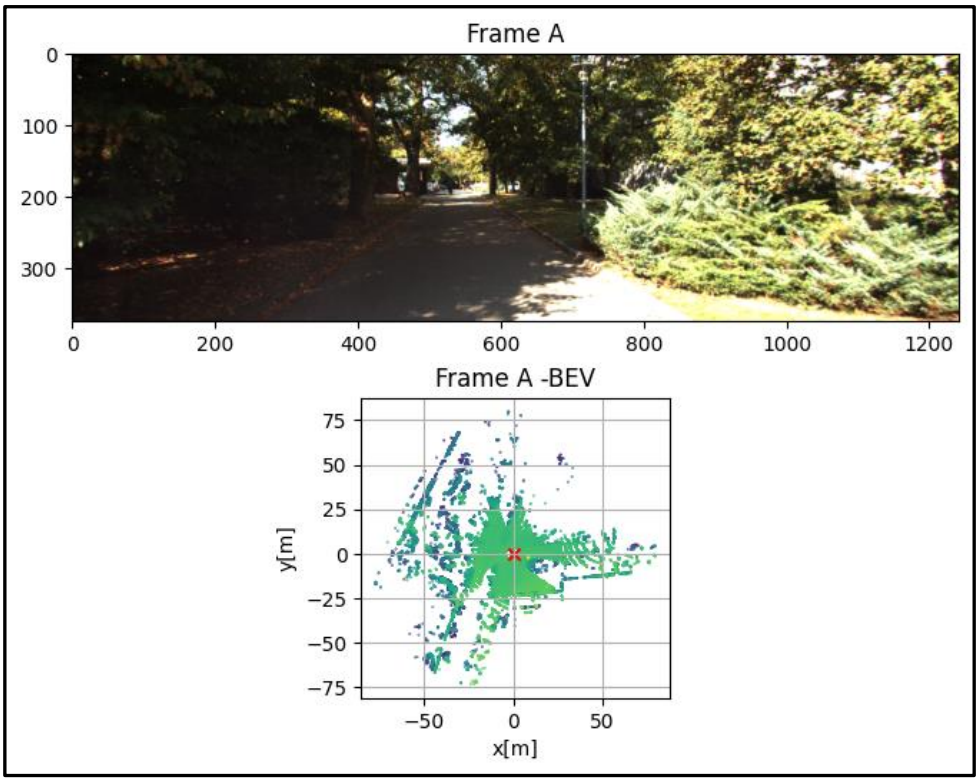


Figure 9 – Frame A number 5

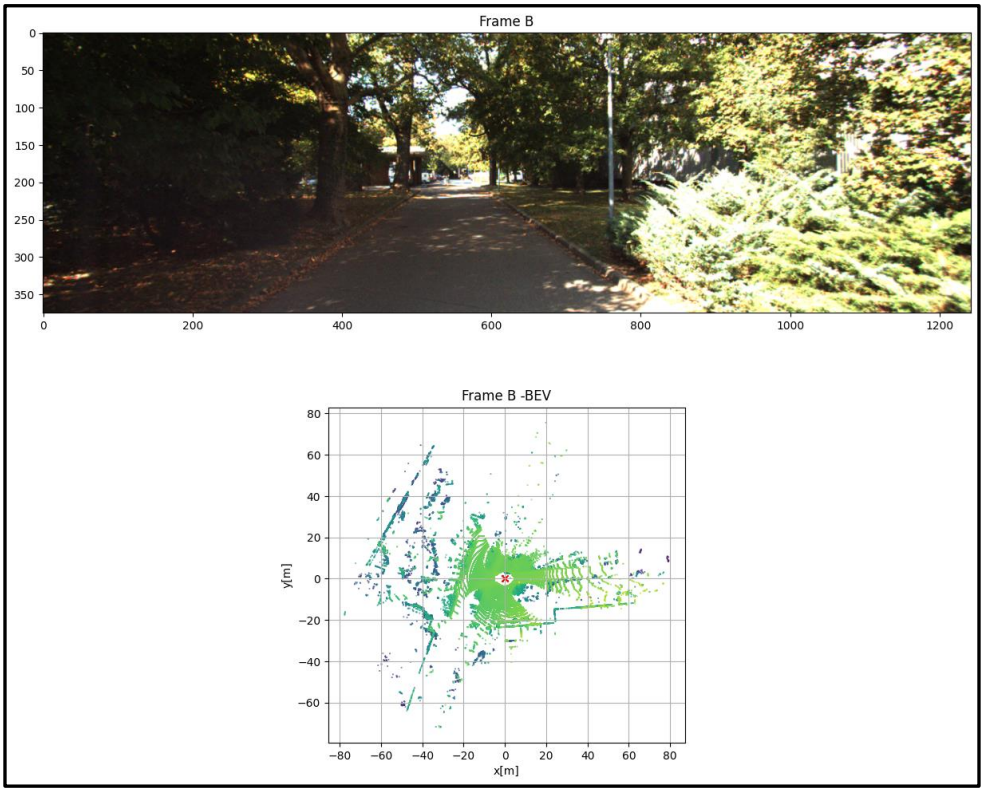


Figure 10 – Frame B number 9

Expected Distance Measurement for full point cloud:

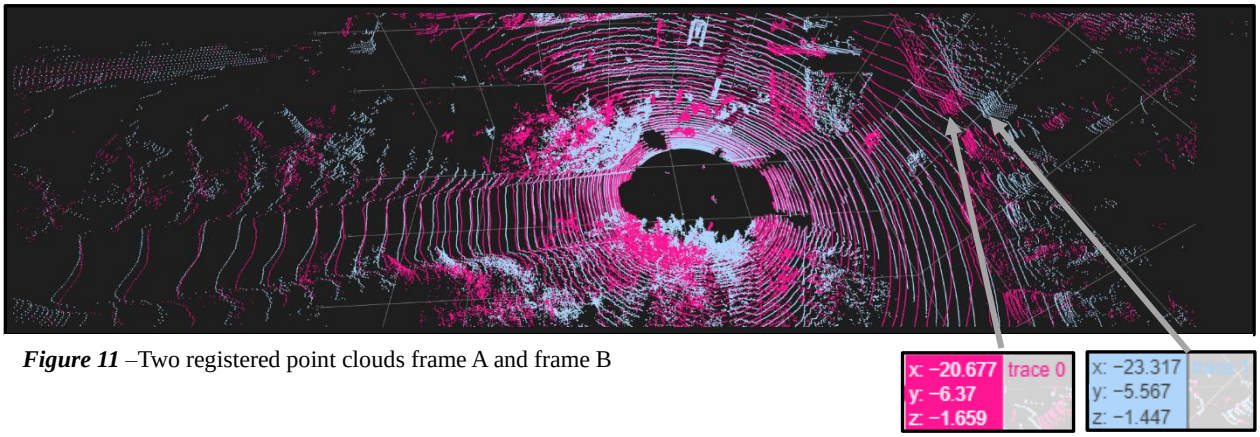


Figure 11 –Two registered point clouds frame A and frame B

Final scan correction results:

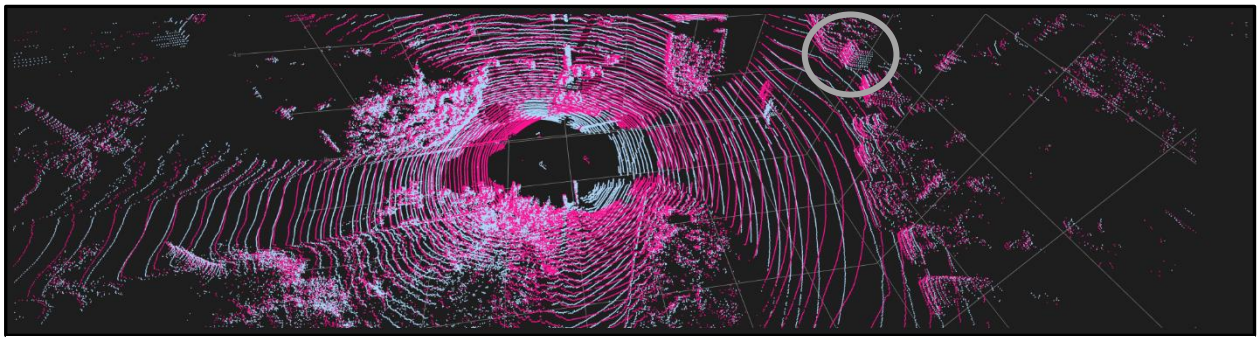


Figure 12 – Final Scan Correction Results using Full Point Cloud with KDTree-Based Association Pairs in ICP

The animation illustrates the final scan correction results after applying the Iterative Closest Point (ICP) algorithm. In the animation, the initial point clouds are shown in their original positions. The gray circle highlights the changes made during the correction process, showing how the point clouds aligned over successive iterations.

Expected Distance Measurement for filtered point cloud:

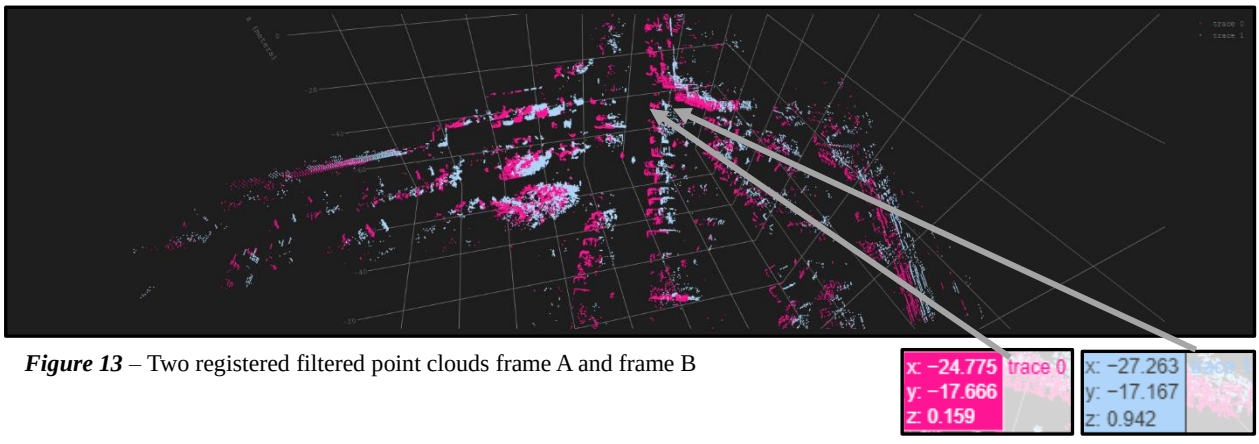


Figure 13 – Two registered filtered point clouds frame A and frame B

Filtered point cloud with association pairs based on KDTree ICP:

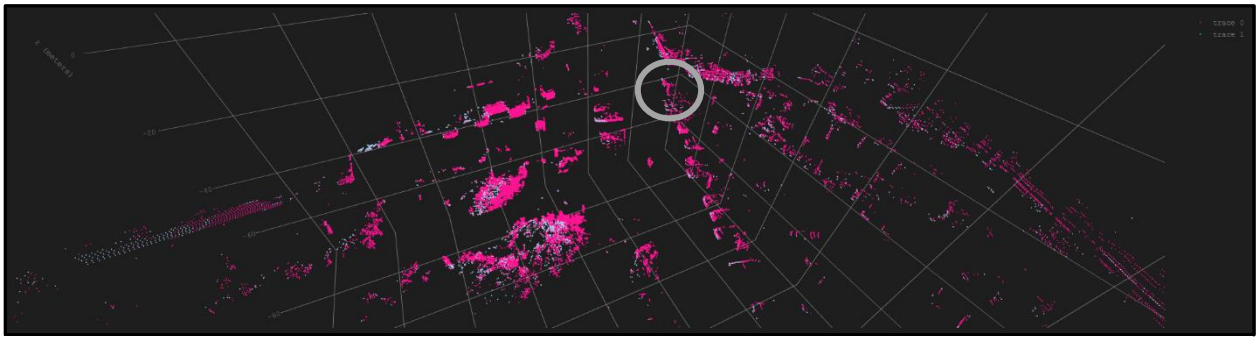


Figure 14 – Filtered point cloud with association pairs based on KDTree ICP

Final scan correction results:

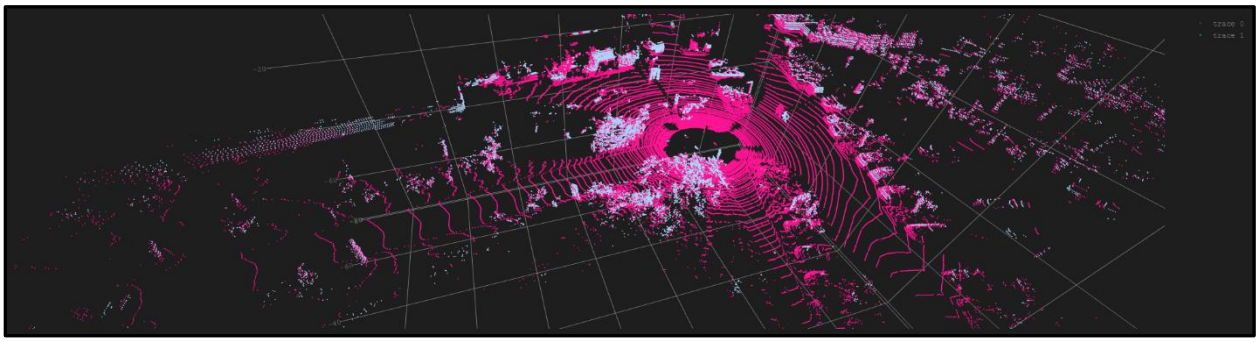


Figure 15 – Final Scan Correction Results using Filtered Point Cloud with KDTree-Based Association Pairs in ICP

Performance comparison of full and filtered point clouds:

The ICP algorithm's performance on the full point cloud started with an initial error of 0.425 and converged to approximately 0.23 after 45 iterations, showing a gradual decline in error. This process took an elapsed time of 33.08 seconds. In contrast, the filtered point cloud began with a higher initial error of 0.9 but also converged to about 0.23, achieving this in only 19 iterations, with an elapsed time of 4.11 seconds. This indicates that while the filtered point cloud had a higher starting error, it required fewer iterations and significantly less time to reach a similar final error. Filtering out redundant points on the road, which were similar and added noise, allowed for a more efficient and faster alignment process, highlighting the importance of preprocessing steps in improving the accuracy and convergence speed of the ICP algorithm. Additionally, the filtered point cloud's rapid error reduction suggests that removing irrelevant or repetitive data points can significantly enhance the algorithm's performance by focusing computational resources on more informative features.

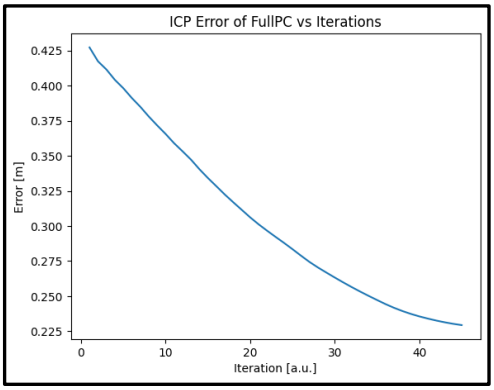


Figure 16– ICP Error of full PC vs Iterations

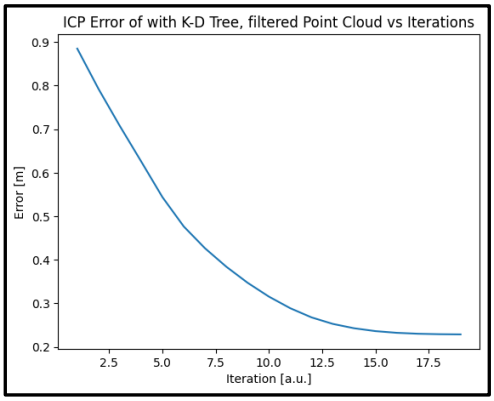


Figure 17– ICP Error of filtered PC vs Iterations

Filtered point cloud with associations pairs based on Nearest Neighbors:

When comparing the ICP results for the filtered point cloud using Nearest Neighbors (NN) versus K-D Tree, several key differences are evident. The ICP with Nearest Neighbors took 7.83 seconds and required 19 iterations to converge, achieving a final error of 0.229 meters. In contrast, the ICP using K-D Tree on the same filtered point cloud was significantly faster, with an elapsed time of just 4.11 seconds for 19 iterations, also reaching a final error of 0.229 meters. Although both methods resulted in the same final error, the K-D Tree approach demonstrated a notable advantage in terms of processing speed. This efficiency can be attributed to the K-D Tree's effective data structure for handling spatial queries, which accelerates the nearest neighbor search compared to the NN method.

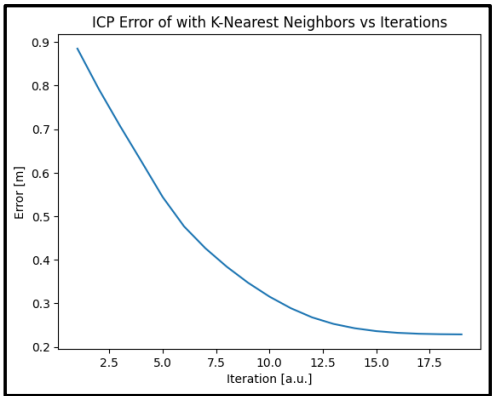


Figure 18– ICP Error of with KNN vs Iterations

Analysis of Final Scan Correction Results:

The scan correction performance varied significantly between the methods and conditions used. For the K-D Tree with the full point cloud, the final error was 0.2294 m with a processing time of 33.08 seconds. The relatively higher error and longer processing time suggest that the presence of redundant and noisy data points from the road negatively impacted the alignment accuracy and efficiency.
In contrast, the K-D Tree with the filtered point cloud achieved a slightly lower final error of 0.2286 m and a significantly faster processing time of 4.11 seconds. This improvement is due to the removal of redundant points, which reduced noise and facilitated a more accurate and efficient alignment. The K-Nearest Neighbors method with the filtered point cloud produced the same final error of 0.2286 m but took longer to converge, with an elapsed time of 7.826 seconds. Although the final accuracy was comparable to the K-D Tree with filtered data, the increased processing time indicates that K-Nearest Neighbors was less efficient in this scenario. Overall, the results demonstrate that filtering the point cloud enhanced both accuracy and processing speed, with the K-D Tree providing the best balance between efficiency and accuracy. The discrepancies in processing times and errors highlight the importance of choosing the appropriate method and preprocessing steps for optimal scan correction.

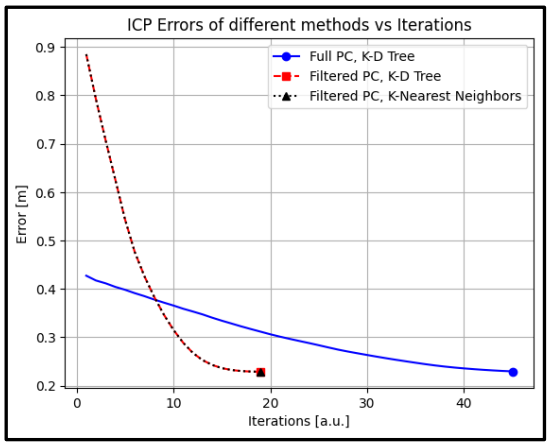


Figure 19– ICP Error of different methods vs Iterations

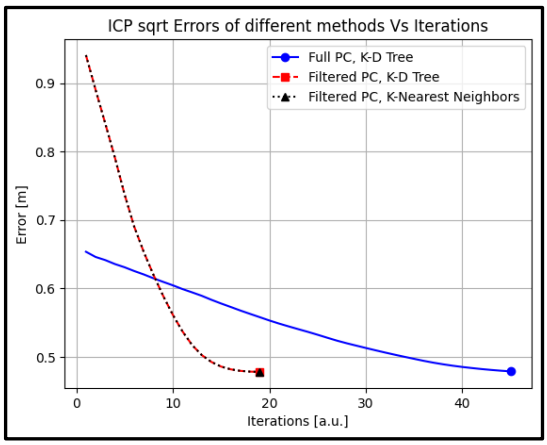


Figure 20– ICP sqrt Error of different methods vs Iterations

c. Before applying the Iterative Closest Point (ICP) algorithm, we present figures of the point clouds from frames 10 and 15. The vehicle's path follows a narrow road with a dynamic object, a car, on the right side. On the left, there is a row of trees, and there are additional static objects, such as parked cars. Additionally, a uniform fence lines the right side along the vehicle's route.

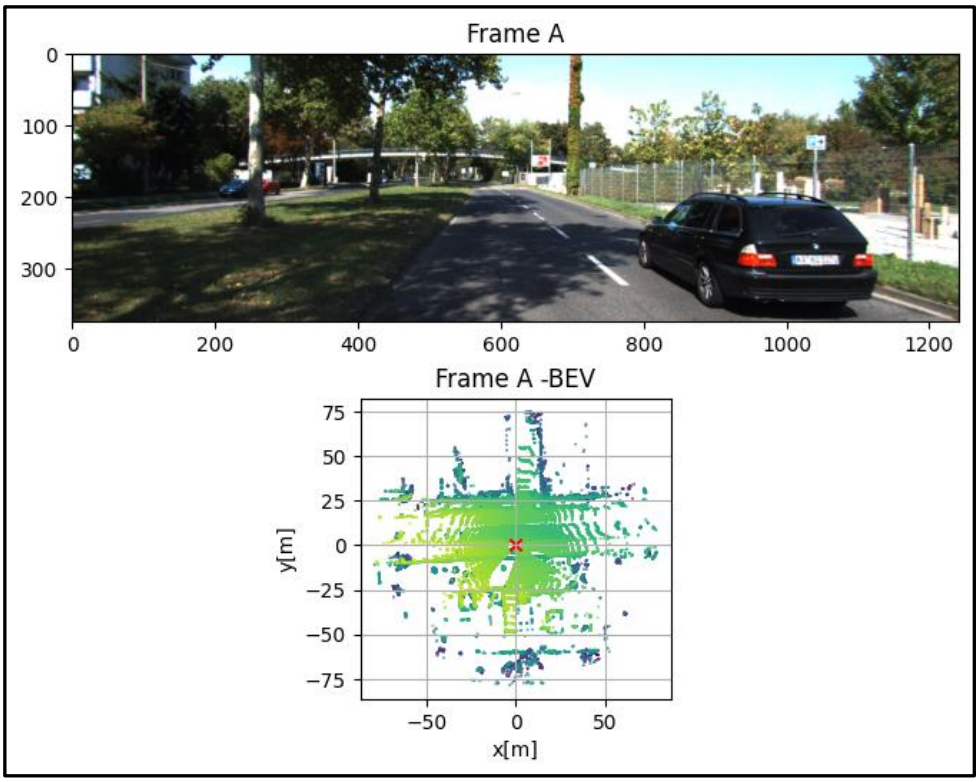


Figure 21 – Frame A number 10

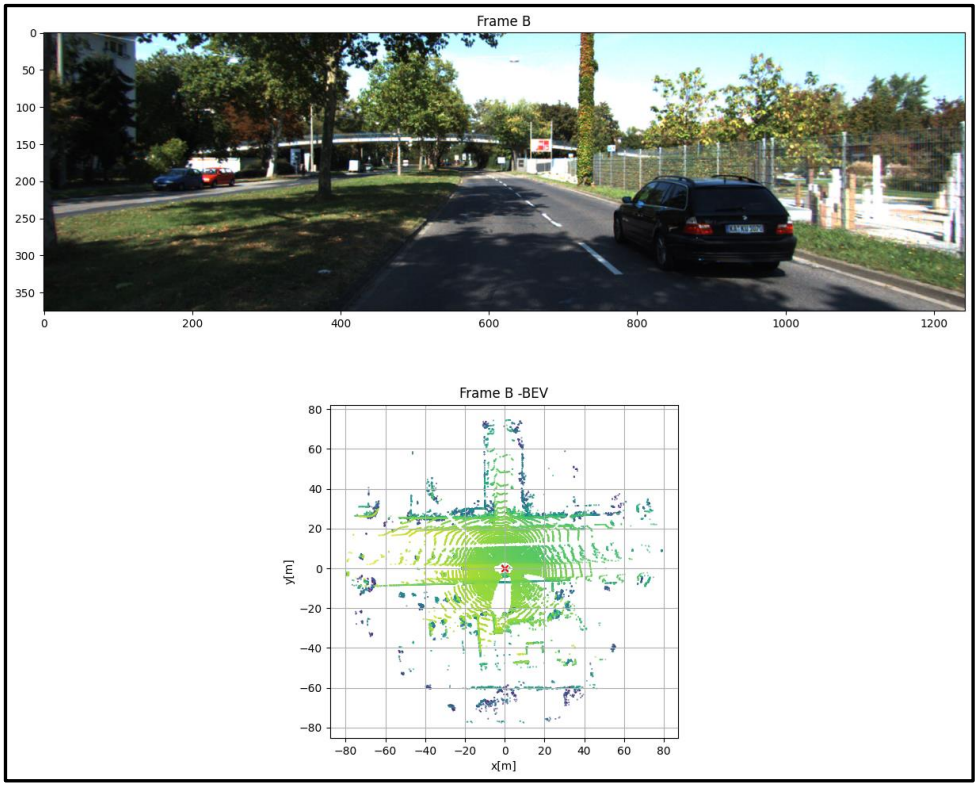


Figure 22 – Frame B number 15

Expected Distance Measurement for full point cloud:

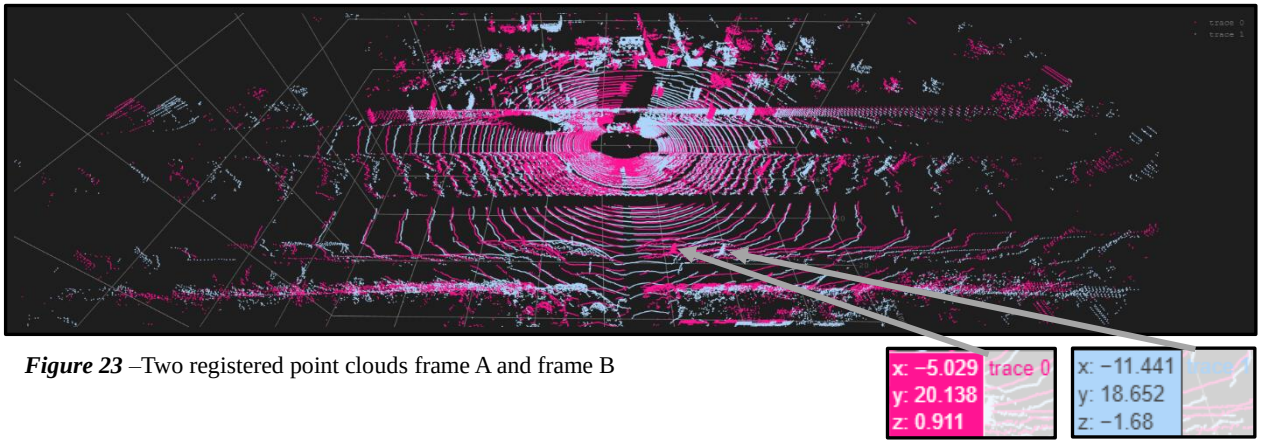


Figure 23 –Two registered point clouds frame A and frame B

Final scan correction results:

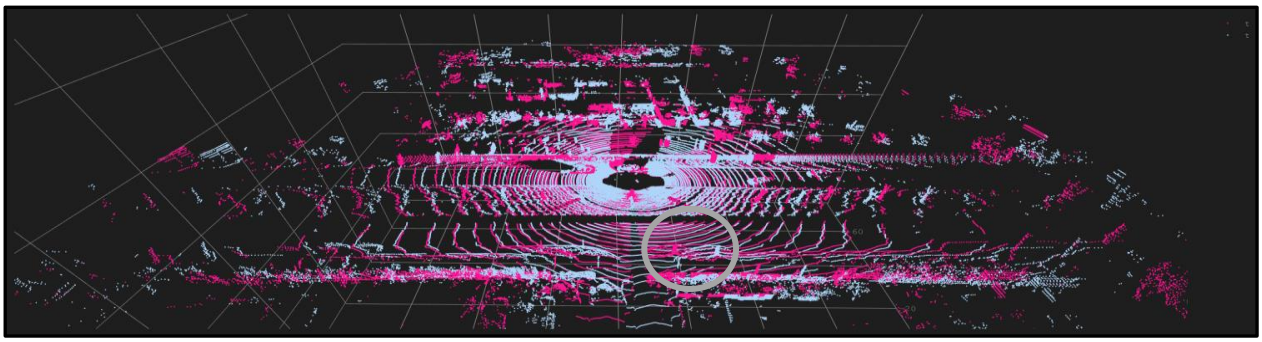


Figure 24 – Final Scan Correction Results using Full Point Cloud with KDTree-Based Association Pairs in ICP

The animation illustrates the final scan correction results after applying the Iterative Closest Point (ICP) algorithm. In the animation, the initial point clouds are shown in their original positions. The gray circle highlights the changes made during the correction process, showing how the point clouds were supposed to align over successive iterations. However, it is evident that the point clouds did not match correctly after the algorithm was applied. This misalignment indicates that the scan correction process was not successful in this case, likely due to insufficient feature correspondence or noise in the data.

Expected Distance Measurement for filtered point cloud:

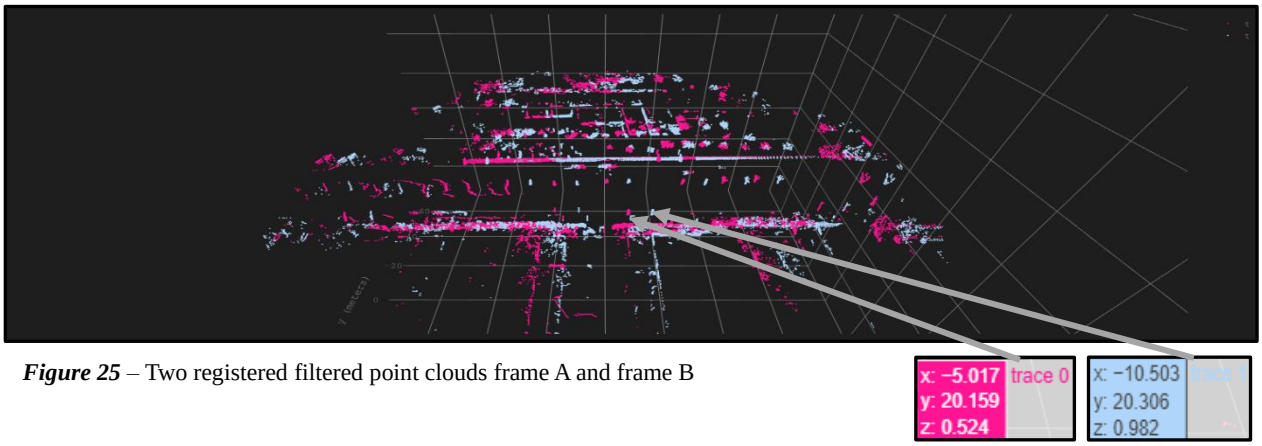


Figure 25 – Two registered filtered point clouds frame A and frame B

Filtered point cloud with association pairs based on KDTree ICP:

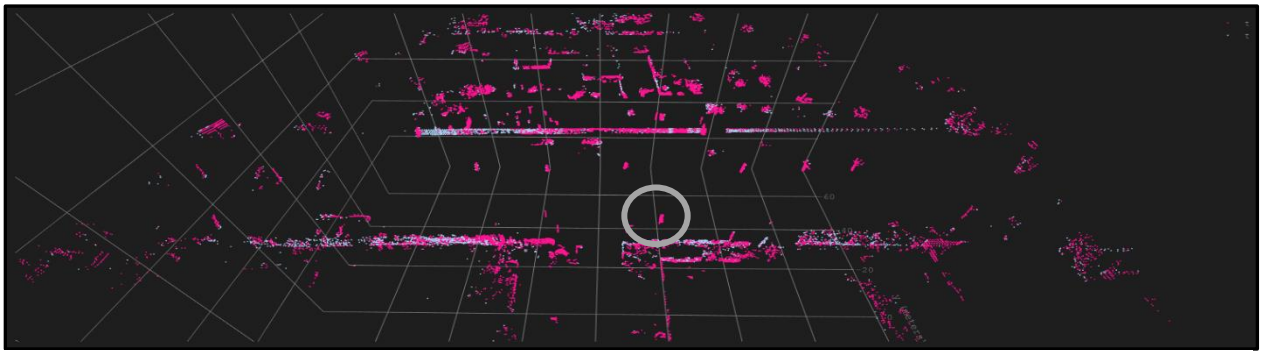


Figure 26 – Filtered point cloud with association pairs based on KDTree ICP

Final scan correction results:

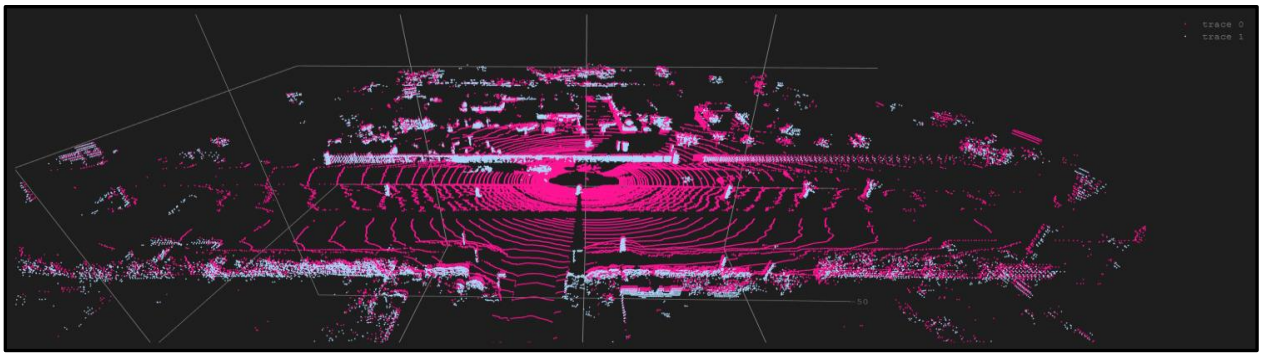


Figure 27 – Final Scan Correction Results using Filtered Point Cloud with KDTree-Based Association Pairs in ICP

Performance comparison of full and filtered point clouds:

The ICP algorithm's performance on the full point cloud failed to converge properly. The initial error started at 0.4745 and remained high, ending with a final error of 0.491 after just 5 iterations. This process took an elapsed time of 3.94 seconds. This lack of convergence is likely due to dynamic objects such as the car, the fence, and the numerous similar features on the road that made it difficult for the algorithm to find proper point correspondences. In contrast, the filtered point cloud, despite having a higher initial error of 1.55, eventually reduced to a final error of 0.876 after 33 iterations, taking an elapsed time of 3.39 seconds. This suggests that the filtered point cloud, although it did not start with a lower error, required more iterations to converge to its final error.

The results indicate that the presence of dynamic objects and repetitive features in the full point cloud hampered the algorithm's ability to align the scans correctly. Filtering out redundant and dynamic points allowed for a more focused and efficient alignment process, but further preprocessing steps might be necessary to improve the accuracy and

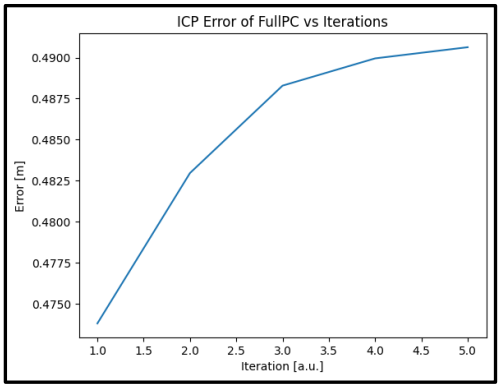


Figure 28 – ICP Error of full PC vs Iterations

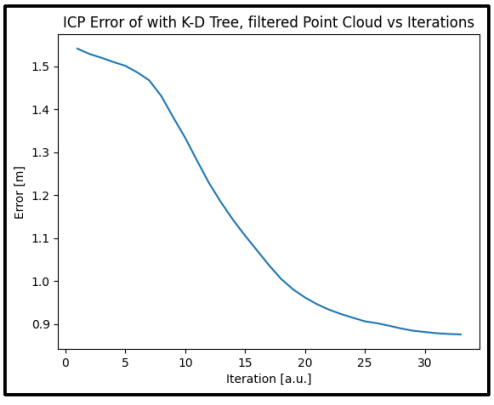


Figure 29 – ICP Error of filtered PC vs Iterations

convergence speed of the ICP algorithm. Removing irrelevant or repetitive data points can significantly enhance the algorithm's performance by focusing computational resources on more informative features.

Filtered point cloud with associations pairs based on Nearest Neighbors:

When comparing the ICP results for the filtered point cloud using Nearest Neighbors (NN) versus K-D Tree, several key differences are evident. The ICP with Nearest Neighbors took 7.94 seconds and required 33 iterations to converge, achieving a final error of 0.876 meters. In contrast, the ICP using K-D Tree on the same filtered point cloud was significantly faster, with an elapsed time of just 3.39 seconds for 33 iterations, also reaching a final error of 0.876 meters. Although both methods resulted in the same final error, the K-D Tree approach demonstrated a notable advantage in terms of processing speed. This efficiency can be attributed to the K-D Tree's effective data structure for handling spatial queries, which accelerates the nearest neighbor search compared to the NN method.

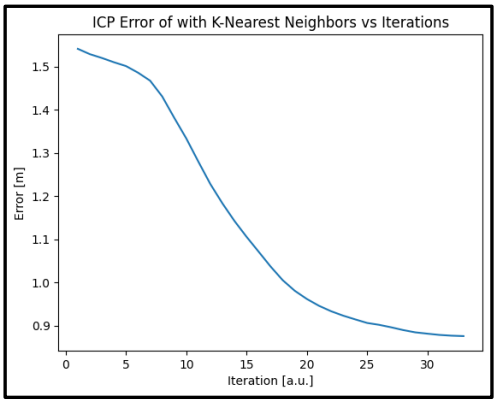


Figure 30 – ICP Error of with KNN vs Iterations

Analysis of Final Scan Correction Results:

In the case of the full point cloud, the ICP algorithm failed to converge, resulting in a final error of 0.491 meters after 5 iterations, with an elapsed time of 3.94 seconds. This indicates that the correction was not performed properly. The failure can be attributed to dynamic objects like the car, the uniform fence, and the highly similar features in the frames, which confused the algorithm and prevented accurate point matching. For the filtered point cloud using the K-D Tree, the algorithm successfully converged, achieving a final error of 0.876 m after 33 iterations, with an elapsed time of 3.39 seconds. Filtering out redundant and dynamic points significantly improved the performance, allowing for more accurate point matching and alignment. Using the filtered point cloud with Nearest Neighbors, the algorithm also performed well, with a final error of 0.876 m after 33 iterations and an elapsed time of 7.94 seconds. This result was comparable to the K-D Tree approach, demonstrating that both methods can effectively handle filtered point clouds to produce accurate scan corrections.

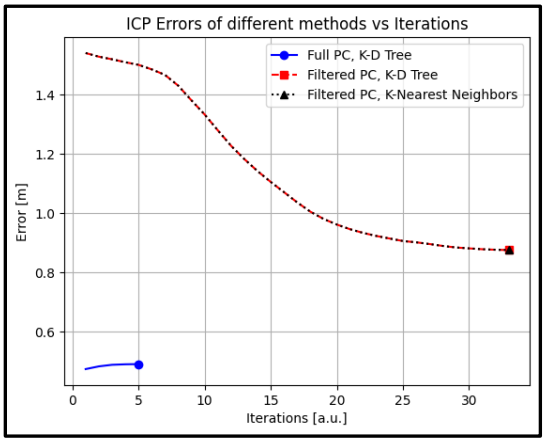


Figure 31 – ICP Error of different methods vs Iterations

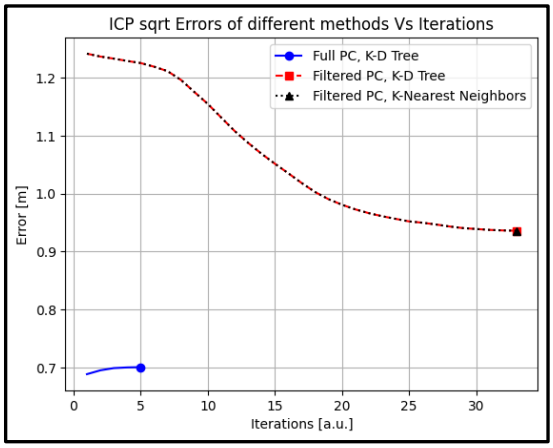


Figure 32– ICP sqrt Error of different methods vs Iterations

Summary:

The analysis of the Iterative Closest Point (ICP) algorithm on both full and filtered point clouds reveals several critical insights.

Firstly, performing preprocessing steps, such as filtering redundant points, is essential for improving the accuracy and convergence speed of the ICP algorithm. The presence of dynamic objects and repetitive features in the full point cloud can hinder the matching process, leading to slower convergence and higher initial errors. By filtering out these redundant points, the algorithm can focus on more informative features, resulting in faster and more efficient alignment.

Secondly, the K-D Tree method proves to be superior to the Nearest Neighbors (KNN) method when implemented in a brute-force manner. The K-D Tree's efficient data structure for handling spatial queries significantly accelerates the nearest neighbor search, offering a notable advantage in processing speed compared to the KNN method. This efficiency makes K-D Tree the preferable choice for implementing the ICP algorithm on filtered point clouds.

In conclusion, to achieve optimal performance in point cloud alignment using the ICP algorithm, it is crucial to preprocess the data by filtering redundant points. Additionally, utilizing the K-D Tree method for nearest neighbor search can greatly enhance the algorithm's efficiency and effectiveness.

The paper ***"KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way"*** presents several contributions to the Iterative Closest Point (ICP) process, aiming to enhance its efficiency and robustness. Two contributions of this paper are:

- **Handling Dynamic Objects with Robust Optimization:**
  The KISS-ICP method also incorporates strategies to handle dynamic objects, enhancing its robustness in environments with moving elements. By integrating motion compensation and robust data association, the algorithm effectively manages dynamic objects, ensuring accurate pose estimation and stable performance.
  "Behley and Stachniss [1] propose the surfel-based method SuMa to achieve LiDAR odometry estimation and mapping. It has also been extended to account for semantics [8] and explicitly handle dynamic objects [7]." (Vizzo et al., 2023, p. 2)

- **Robust and Simplified Data Association and Optimization:**
  The paper introduces a robust method for data association that avoids the use of data-dependent features such as normals, curvature, or other descriptors, which can be unreliable with noisy or sparse LiDAR data. This design choice enhances the generalization of the ICP algorithm across different sensor resolutions and environments. Additionally, the paper employs a robust optimization process to improve pose estimation, which minimizes the sum of point-to-point residuals. This approach ensures more accurate and stable pose estimates, even in challenging conditions.
  "The advantage of this choice is that we do not need to compute data-dependent features such as normals, curvature, or other descriptors, which may depend on the scanner or the environment." (Vizzo et al., 2023, p. 4, E. Step 4)

Appendix

The directory for Part A is named EKF-SLAM.
which contains the code as “EKF_SLAF_ex3”, a subdirectory called Figures that includes all the relevant images and animation, and an Excel file for error analysis and the improvement of the K factor.

For Part B, the directory is named ICP, which contains two subdirectories: one for the code and one for the figures. Each subdirectory includes the code and the corresponding figures folder with the relevant images and animations.