

Article

LiDAR-Based Deep Learning-Enabled Geometric Fingerprinting for Indoor Robot Localization

Harsha Keladi Ganapathi ¹ and Shayok Mukhopadhyay ^{2,*}¹ Master of Science, Data Science Program, University of New Haven, West Haven, CT 06516, USA; hkela1@unh.newhaven.edu² Electrical & Computer Engineering and Computer Science Department, University of New Haven, West Haven, CT 06516, USA

* Correspondence: smukhopadhyay@newhaven.edu

Abstract

Localization is a fundamental requirement for autonomous mobile robot navigation. Several localization techniques exist, but they often require extensive installation of beacons, careful parameter tuning, high computational requirements, or an immense amount of training data. Environmental (e.g., indoor)/resource constraints, sensor degradation, and sudden pose discontinuities can make such methods unreliable. This creates a critical gap: the lack of a simple, lightweight localization method that can operate as a primary localization method or in parallel with other classical systems and provide reliable pose estimates during primary localization system failures. Thus, this paper proposes a lightweight, deep learning (DL)-based, two-dimensional LiDAR localization method. The approach combines LiDAR scan range data with eleven proposed handcrafted geometric features to train a Convolutional Multi-Layer Perceptron (ConvMLP) regression model for predicting the two-dimensional location of a robot, which is further smoothed by an augmented recursive Extended Kalman filter (EKF). The overall system is validated in three real-world environments. The results are compared against various existing machine learning (ML) models and other well-known localization techniques. The experimental results demonstrate a 280 Hz pose-update rate, achieving a 13 cm Root Mean Square Error (RMSE) using the ConvMLP model alone, which further reduces to 5 cm when fused with the recursive EKF.

Keywords: deep learning; LiDAR localization; geometric fingerprinting; kidnapped robot recovery; sensor fusion; fallback localization system

Academic Editors: Kanstantsin Miatliuk and Henrik Gordon Petersen

Received: 24 June 2026

Revised: 23 July 2026

Accepted: 27 July 2026

Published: 3 August 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Accurate and reliable pose estimation is a fundamental requirement for autonomous mobile robot navigation. This is even more important in indoor environments, where ubiquitous beacon-based positioning is unavailable, unlike the outdoors via GPS [1]. While installing landmarks in such an environment can help overcome some issues with beacon-based systems, this may not be possible in all types of indoor environments. Even if installation is possible, it may be extremely time-consuming and/or resource-intensive [2]. Given the availability of positioning data via beacons, landmarks, or inertial sensors onboard robots, classical localization via state estimators, probabilistic filters and SLAM-based systems has demonstrated strong performance in structured settings. However,

they often rely on significant onboard computational ability, careful parameter tuning, and continuous availability of high-quality sensor data [3]. These requirements can limit their usability with low-cost robotics platforms/setups common in educational or early-stage prototyping settings [4].

In recent years, machine learning (ML)-based, regression-oriented localization methods have gained attention due to their ability to directly estimate robot pose from sensor observations [5]. While promising, many existing deep learning-based approaches depend on high-dimensional perception pipelines like large-scale vision-based segmentation, graphical processing unit (GPU)-based hardware acceleration, or availability of extremely large-scale training datasets [6]. As a result, their deployment on resource-constrained robots remains challenging, and the role of such ML-based systems is often envisioned as replacing, rather than complementing, traditional localization systems. In real-life robotic deployments, intermittent localization failures and degradation of localization accuracy are not uncommon. Situations such as sensor occlusion, environmental changes [7], kidnapped robot events (due to manual intervention [8]/complete short-duration sensor dropout causing a robot to experience a discontinuity in pose), or computational overload [2] can cause classical localization methods to degrade or temporarily fail. In such cases, the availability of a computationally lightweight but reasonably accurate localization method can be critical for maintaining basic robot operation and preventing catastrophic failure. Despite this need, relatively little attention has been given to simple, deployable learning-based localization methods, which are either explicitly designed to act as complementary/fallback estimators rather than full replacements or designed for low-cost, low-resource robotic systems.

This paper presents a lightweight 2D LiDAR localization method designed for real-time execution on CPU-only robotics platforms. The proposed approach uses SLAM-derived pose estimates as a reference during training, enabling the model to learn a mapping between raw LiDAR data, handcrafted features based on such data, and planar robot position. It is to be noted that SLAM-derived pose estimates are used as a reference for training and ground truth for comparison because this work does not use any other beacon-based source of ground truth. However, this does not mean that the proposed methodology cannot be used without SLAM. In systems where beacon-based absolute pose measurements are available, the ConvMLP model proposed in this paper can be trained using LiDAR data along with such available direct pose measurements. During inference, the model produces fast, deterministic pose estimates that can be used independently or integrated with classical localization pipelines as a fallback signal.

The proposed method is evaluated in three real-world environments using a differentially driven wheeled mobile robot (WMR), a Turtlebot3 Burger (ROBOTIS Inc., CA, USA) [9]. The performance of the proposed method is assessed against wheel odometry, LiDAR-based localization via a ROS2 Cartographer pose estimator, and pose estimation by Adaptive Monte Carlo Localization (AMCL). Localization accuracy, computation time, memory requirements, pose estimation quality over different robot motion trajectories, and challenging conditions, such as kidnapped robot events, are used to rigorously test the proposed algorithm against the other localization techniques.

The contributions of this work are summarized in the following points.

- (1) A lightweight deep learning-based 2D LiDAR localization model designed for CPU-only deployment.
- (2) A ConvMLP-based pose regression architecture that predicts a robot's position directly from a single (1D) LiDAR scan without requiring an initial pose estimate or explicit maps, integrated with a custom augmented recursive Extended Kalman Filter (EKF).

- (3) A real-world LiDAR dataset and systematic benchmarking study across three indoor maps using a physical Turtlebot3 robot, demonstrating the practicality, repeatability, and comparative performance of the proposed method against classical localization techniques and multiple other ML/DL baselines.

The remainder of this paper is organized as follows: Section 2 reviews related work. Section 3 presents the proposed methodology, experimental navigational map details, and implementation of the learning-based algorithms. Section 4 describes the experimentation setup, and Section 5 reports the results. Section 6 concludes the paper.

2. Related Work

Localization methods for mobile robots have evolved from infrastructure-dependent solutions to probabilistic, SLAM-based, and learning-based approaches. Early localization systems mainly relied on external infrastructure including Global Navigation Satellite Systems (GNSSs), wheel odometry, and inertial sensors [10]. GNSS-based localization provides global positioning, but it is unreliable in indoor environments [11]. Beacon-based systems are common but expensive if accuracy is needed [12]. Wheel odometry suffers from cumulative drift due to wheel slip, uneven terrain, and sensor noise, making pure dead-reckoning approaches unsuitable without external correction [13].

Probabilistic localization methods estimate robot pose by maintaining a belief distribution over the state space. Adaptive Monte Carlo (AMCL) is a widely adopted particle filter-based approach that localizes a robot within a known map using LiDAR observations and motion models [14]. While robust, AMCL requires careful selection of particle counts and sensor models, and it can be computationally expensive on low-power robots/localization setups [15]. SLAM-based localization systems, such as Cartographer [16], GMapping [17], and other scan-matching frameworks, estimate robot pose by aligning incoming LiDAR scans with a global map. These systems achieve high accuracy but rely on continuous scan matching and require significant computational resources [18]. Learning-based localization approaches estimate robot pose directly from sensor data using machine learning or deep learning models [19]. These methods can be broadly divided into pose regression and place recognition-based localization [20].

Pose regression methods use deep neural networks to map sensor data or learned representations directly to robot pose. Recent architectures explored deep architectures operating on point clouds [21] or learned embeddings [22]. While these methods can achieve high accuracy, they often require deep convolutional or transformer-based networks, GPU acceleration, and a large training dataset, thus limiting their deployability on resource-constrained robots/platforms. Place recognition-based pose regression methods retrieve the most similar location from a database of previously observed scenes [23]. These approaches are typically used to provide an initial pose estimate that is refined by subsequent optimization. Although efficient, place recognition methods alone do not provide continuous localization and are often combined with classical estimators. Vision-based localization methods using RGB or RGB-D cameras have also been extensively studied. These approaches benefit from rich visual information but are sensitive to lighting changes, availability of texture, and camera calibration [24]. Additionally, vision-based pipelines often have higher computational costs and experience robustness issues in situations that compromise vision sensor performance.

To address the computational overhead common with raw 1D scans, research on extracting geometric features, such as lines, corners, and key points from 2D-LiDAR data [25], has received significant focus. More recently, Wang Y.T. et al. in [26] proposed a single LiDAR feature fusion algorithm that extracts both line segments and corner points. By integrating these features into a Weighted Parallel Iterative Closest Point (WP-ICP) framework, they achieved robust localization with significantly lower computational cost

compared to raw matching methods. Furthermore, Steinke et al. [27] introduced the concept of geometric fingerprinting for autonomous vehicles. Their work utilized vertical corner features extracted from LiDAR data to create a lightweight feature map, achieving centimeter-level accuracy that outperformed high-cost GNSS solutions in urban environments. These works collectively show that handcrafted geometric features can offer superior efficiency for resource-constrained systems compared to raw deep learning approaches. However, while these methods typically feed features into geometric solvers (like ICP), the work in this paper differentiates itself by feeding these geometric descriptors into a regression neural network.

Recent research combines the pattern recognition capabilities of deep learning models with the stability of classical state estimators like Extended Kalman Filters (EKF). Such integration is necessary due to the inherent performance trade-offs between a deep network's global inference capability and local time-domain consistency enforced by state estimators like EKFs. Moreover, deep neural networks provide frame-by-frame regression-based localization, often exhibiting high-frequency temporal jitter due to sensor noise and environment aliasing [28]. To address this, researchers have proposed a system in which neural networks are used to estimate the global pose, which is then fused with odometry and passed through an EKF to effectively filter outliers and regression noise. This estimation framework ensures that the final pose output maintains both the global accuracy of the learned model and the local smoothness required for stable motion control.

Existing localization approaches exhibit a trade-off between accuracy, robustness, and computational cost [29]. Solely infrastructure-dependent methods may lack long-term reliability and are expensive to implement. Probabilistic and SLAM-based methods are computationally demanding, and many learning-based methods prioritize performance at the expense of deployability, training, computation, and data storage cost. There is a lack of simple, learning-based localization methods that can be operated using low-cost 2D LiDAR sensors, which run in real time on CPU-only platforms, require minimal training infrastructure, and are explicitly designed to complement existing localization systems as fallback pose estimators if the need arises. This work addresses the gap by proposing a lightweight learning-based LiDAR localization approach that emphasizes simplicity, computational efficiency, and practical deployment, rather than replacing established localization pipelines. It uses handcrafted geometric features along with a deep learning model trained on LiDAR scan data and a recursive EKF.

To further clarify this gap, Table 1 provides a structured comparison of representative localization approaches across practical deployment dimensions, including sensing modality, computational requirements, hardware dependency, localization strategy, and real-time feasibility. As shown in Table 1, infrastructure-based methods depend on external hardware; probabilistic and SLAM-based systems require continuous scan matching and significant onboard computation; descriptor-based LiDAR approaches rely on pre-built maps or large databases; and many deep learning models require GPU acceleration or high-dimensional perception pipelines. In contrast, the proposed ConvMLP and handcrafted geometric feature pipeline is designed for CPU-only execution, uses only 2D LiDAR scans, requires no map during inference, and provides deterministic pose regression suitable for lightweight or fallback localization on low-cost robots/localization systems.

Table 1. Systematic comparison of robot localization approaches.

Method Category	Sensing Modality	Computation	Key Hardware Dependency	Localization Strategy	Deployment Constraints	Cited Papers
GNSS/Outdoor or Beacons	External signals	Low	Requires infrastructure	Absolute positioning	Not usable indoors	[11,12]
Odometry/IMU	Wheel and inertial	Very low	Requires on the robot	Dead-reckoning	Drift accumulates	[13]
AMCL	LiDAR and map	Medium–High	Requires map	Probabilistic filtering	Sensitive to particle count	[14,15]
SLAM (Cartographer, GMapping)	LiDAR	High	CPU/GPU	Scan matching and mapping	Heavy computation	[16–18]
ICP/NDT	LiDAR	High	Requires map	Scan-to-map alignment	Sensitive to initialization	[25–27]
Scan Context	LiDAR	Medium	Requires database	Place recognition	Not continuous localization	[23]
DL Pose Regression	LiDAR/Vision	Medium–High	Often GPU	Direct regression	Needs large datasets	[19–21]
Proposed ConvMLP with 11 Features	2D LiDAR	Low (CPU-only)	Label pose	Direct regression and EKF smoothing	Lightweight, map-free inference	-

3. Methodology

The proposed localization approach uses a deep learning (DL)-based pipeline to estimate a robot's position in an environment using 2D LiDAR data. The system operates in three distinct phases, as shown in Figure 1. In the first phase, the robot traverses the map and collects the LiDAR scan data, and the corresponding reference robot pose estimate from the ROS2 Cartographer SLAM module. As the robot traverses the map, the robot pose estimate is sampled. At every physical location of the robot corresponding to a sampled pose estimate, LiDAR range data (from 0° to 360°), which provide the distances of all objects around that position of the robot, are collected. Then, the data are cleaned, and handcrafted features are generated from the LiDAR range data. In the second phase, the data preprocessing includes data augmentation and normalization. Many ML- and DL-based algorithms, such as linear regression, ridge regression, K-nearest neighbors (KNNs), random forest, multi-layer perceptron (MLP), convolutional network, and ConvMLP, are used to train models using the preprocessed data. In the third phase, the best model, which is the ConvMLP DL model, is used in inference mode to predict the robot's location in the environment. Inference involves acquiring LiDAR scan range data for all obstacles ($0^\circ - 360^\circ$) around the robot at a particular position and then feeding this scanned data to the trained ConvMLP model, which then predicts the location of the robot based on the LiDAR scan input. Figure 1 shows the above phases as a schematic, and the subsections below first talk about the robotic hardware used and the experimental environments. Following this, each phase is explained in detail.

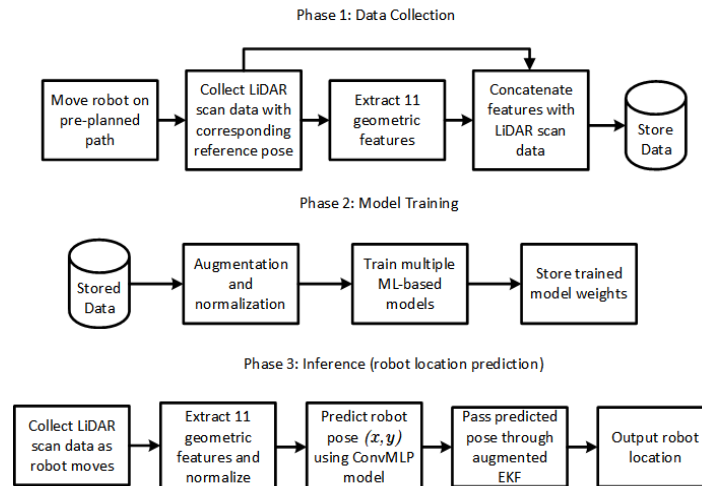


Figure 1. Workflow of the proposed DL-LiDAR localization method.

3.1. Hardware and Environment Setup

Experiments are conducted using a Turtlebot3 Burger mobile robot equipped with a 2D LiDAR sensor (shown in Figure 2a), with a maximum sensing range of 3.5 m and 360° coverage. The LiDAR scans provide range measurements along rays with 1° angular resolution, as shown with the robot at location $(x = 0.75, y = 0.63)$ in Figure 2b. Another such LiDAR scan dataset is visible in Figure 2c, with the robot at the location $(x = 0.44, y = 0.56)$. The dimensions of the Turtlebot3 Burger robot are 138 mm (length (l)) $\times$ 178 mm (width (w)) $\times$ 192 mm (height (h)), and the dimensions of the LiDAR unit are 90 (l) $\times$ 70 (w) $\times$ 42 (h) mm. For this work, three distinct real-world indoor environments with planar walls and a limited number of obstacles are used, as shown in Figures 3–5. As shown in Figure 3, the constructed environment for Map1 is a small square room with three obstacles placed in the environment. Figure 4 shows the constructed Map2, which is an L-shaped, corridor-like environment with three obstacles. Figure 5 shows the real-life environment used as Map3. This is a robotics lab environment with multiple obstacles, and the yellow arrow depicts the door of the environment. Please note that Map2 itself is used as an obstacle in this environment (i.e., the robot cannot move in the interior of Map2 while it is navigating in Map3). Table 2 summarizes some of the physical specifications and collected dataset characteristics for each experimental map. The actual dimensions of the maps themselves can be seen in Figures 6a,b and 7, respectively, for Maps 1, 2, and 3. The dimensions of the obstacles in these maps are also shown to scale in Figures 6a,b and 7, respectively.

Table 2. Summary of environment details.

Map ID	Map Type	Dimensions (m) L $\times$ B = Area	Raw Samples Collected	Augmented Samples
Real Map1 (Figure 3)	Square-shaped area	1.75 m $\times$ 1.75 m = ~ 3 m ²	47,359	189,436
Real Map2 (Figure 4)	L-shaped corridor	3 m $\times$ 3 m = ~ 9 m ²	41,477	165,908
Real Map3 (Figure 5)	Real indoor lab environment	7 m $\times$ 7 m = ~ 49 m ²	29,481	117,924

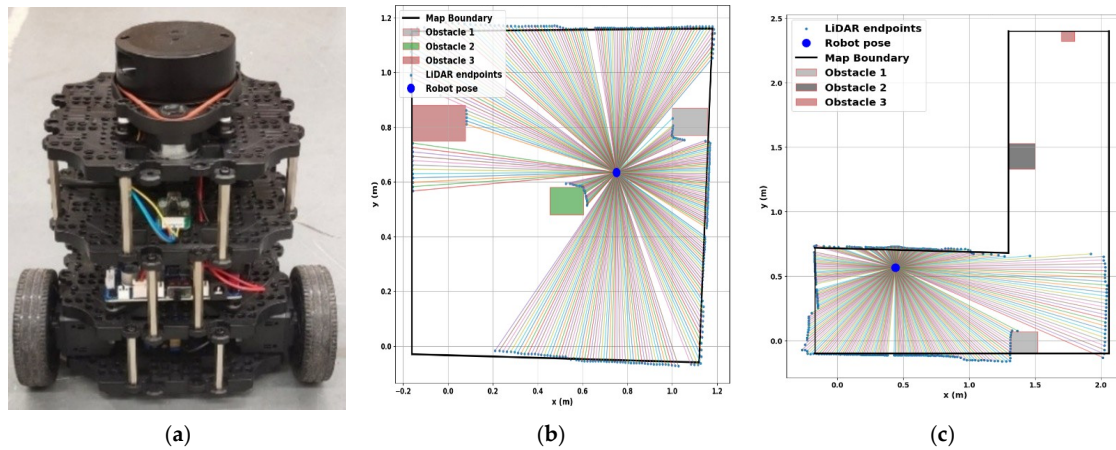


Figure 2. (a) Turtlebot3 Burger; (b) Map1 and an example laser scan; (c) Map2 and an example laser scan.

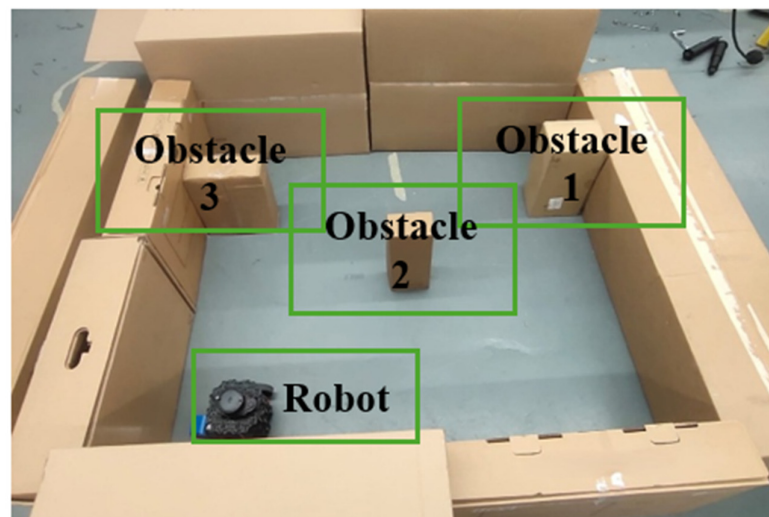


Figure 3. Map1, a square room with three obstacles.

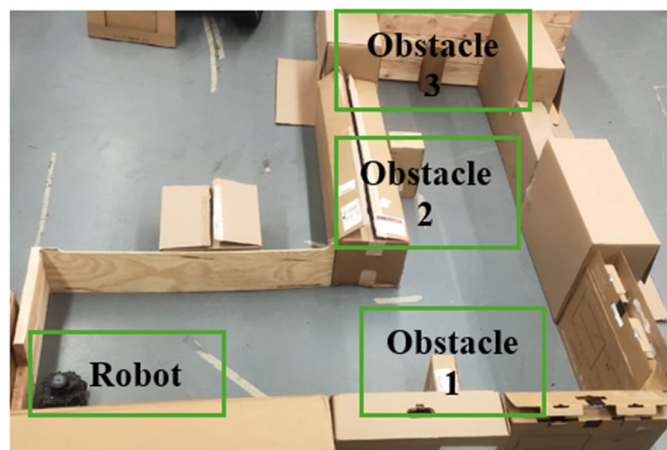


Figure 4. Map2, an L-shaped corridor with three obstacles.

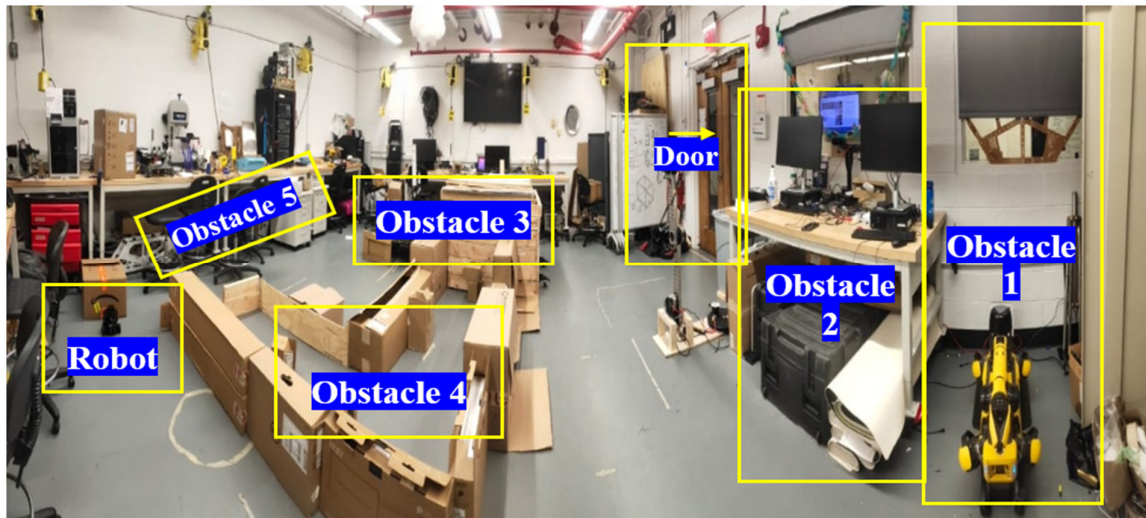


Figure 5. Map3, a large square real laboratory environment with multiple obstacles.

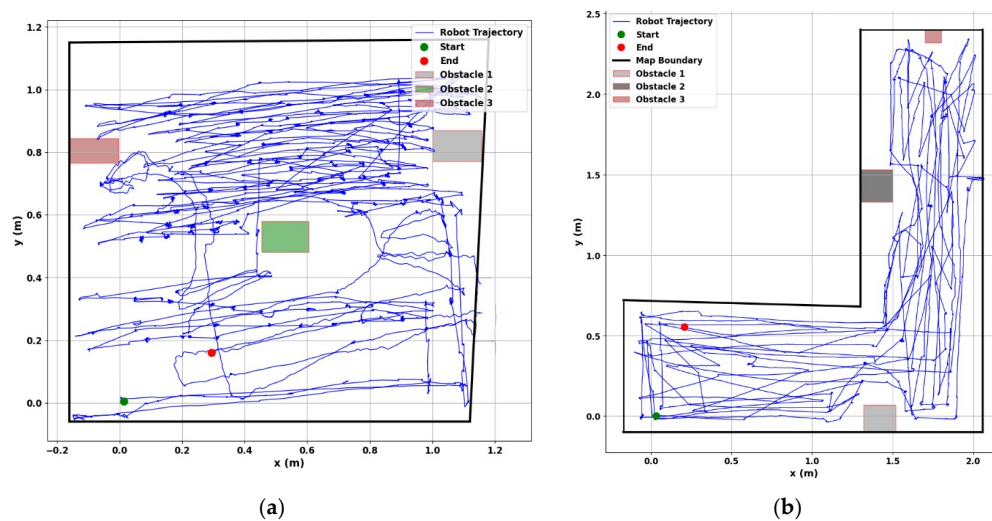


Figure 6. (a) Robot's trajectory during data collection in Map1; (b) robot's trajectory during data collection in Map2.

3.2. Data Collection

The required data is collected by manually teleoperating the robot through the environment in a lawnmower-like pattern while synchronously recording raw LiDAR scans and reference pose information (x, y). Note that the pattern of the path in which the robot is moved for data collection (e.g., lawnmower or other pattern) is not relevant to the developed DL-based localization approach. This is because the trained model associates position information with the LiDAR scan information at a given location, irrespective of how the locations are visited and in what order. For this work, Cartographer SLAM is utilized during the data collection process to generate the reference positions for each LiDAR scan. These reference positions (only the (x, y) coordinates) are the target variable (labels) for prediction. The orientation θ is not predicted, as the current focus is only on predicting (x, y) coordinates in this work. Raw LiDAR scans are the distances of the obstacles/map boundary from the robot position for every angle ($0-360$), in 1° increments. At selected spatial locations, the robot is rotated (θ varied from $(0$ to 360)) in 10° increments, resulting in approximately 36 distinctly oriented LiDAR scans sampled per

robot location. This θ rotation is done automatically at arbitrarily selected robot locations to capture orientation-dependent LiDAR observations. This multi-yaw sampling strategy attempts to reduce perceptual aliasing by ensuring that the model learns to recognize environmental features from all viewing angles wherever rotations were performed. Column four in Table 2 provides a detailed breakdown of the collected data quantities for each map. Figure 6a, b illustrates the robot's trajectory during the data collection process in Maps 1 and 2, respectively, which highlights the path coverage and sampling density. Additionally, small circular or spiral clusters visible in the plotted trajectories correspond to locations where rotations were performed, indicating points where multiple yaw-dependent scans were recorded. Then, the collected data is cleaned before further processing. Data cleaning handles the following cases: (i) if the LiDAR scan does not pick up objects because they are beyond the sensor's measurement range, such readings usually appear in the dataset as "N/A", these are manually set to the maximum LiDAR sensor range, i.e., 3.5 m, (ii) if there are cases such that a set of LiDAR scan range values at a particular physical location are not accompanied with the corresponding reference (x, y) coordinates, then such a set of LiDAR scan range values is removed from the dataset, and (iii) if there are multiple identical sets of LiDAR scan range values at any given reference (x, y) coordinate, then only one set of LiDAR scan range values is kept for that particular reference (x, y) coordinate.

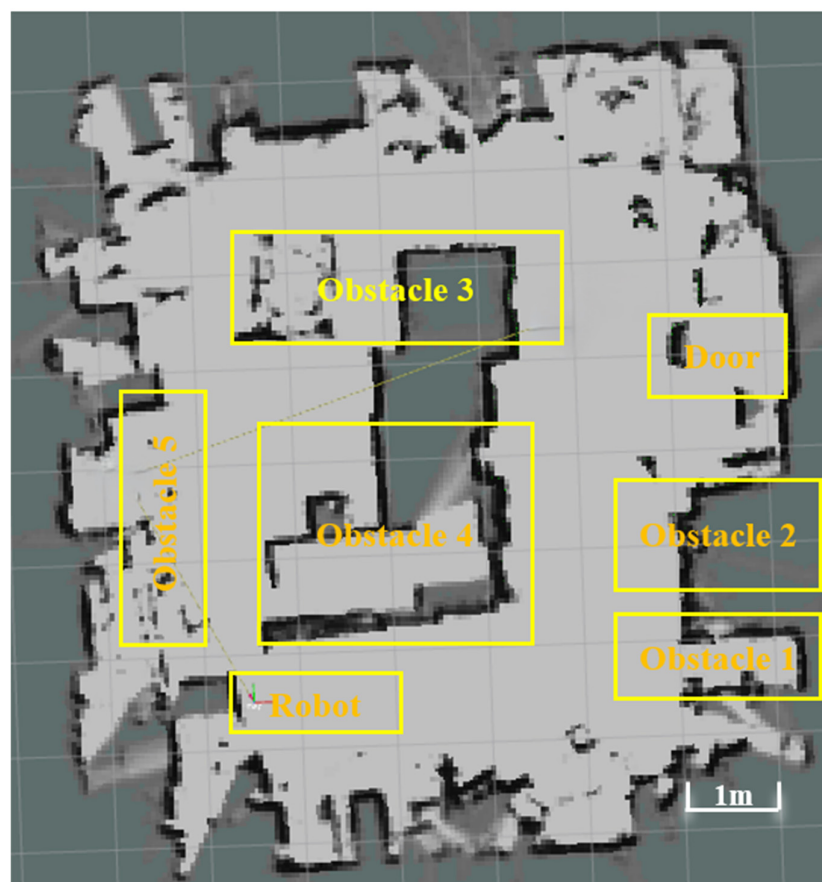


Figure 7. RViz visualization of Map3: a large square laboratory with multiple obstacles.

3.3. Feature Representation

Instead of relying solely on the collected raw LiDAR scans, the proposed method utilizes handcrafted features derived from the 2D LiDAR scan. These include statistical measures, geometric descriptors, and frequency-domain characteristics computed over

360 bins, 1 bin per 1° increment. Let the set $S = \{r_0, r_1, r_2 \dots, r_{359}\}$ represent the LiDAR range measurement at a particular (x, y) position. Here, $r_i \in \mathbb{R}, i \in \{0, 1, 2 \dots 359\}$ represents the range measurements at every 1° increment. In this work, the following 11 features are extracted from S : $d_{min}, d_{max}, r_{spread}, \sigma, \delta, \kappa, \rho, J_s, H, J_l$, and $\mathcal{F}$. These features are observed to help effectively encode spatial information. This specifically allows efficient training by providing ML algorithms more opportunities to detect patterns in the LiDAR scan data compared to just detecting patterns in raw data. The features calculated at each (x, y) position are described further.

The features **minimum distance** (d_{min}) and **maximum distance** (d_{max}) represent $\min_{i \in \{0, 1, 2 \dots 359\}} r_i$ and $\max_{i \in \{0, 1, 2 \dots 359\}} r_i$, respectively. At an (x, y) position, let $\tilde{x}_i = r_i \cos(\theta)$ and $\tilde{y}_i = r_i \sin(\theta)$, where $i \in \{0, 1, 2 \dots 359\}$ and $\theta \in \{0^\circ, 1^\circ, 2^\circ \dots 359^\circ\}$. Let $x_{span} = (\max(\tilde{x}_i) - \min(\tilde{x}_i))$ and $y_{span} = (\max(\tilde{y}_i) - \min(\tilde{y}_i))$. The third feature **spread** (r_{spread}) is now calculated as follows

$$r_{spread} = \frac{\max(x_{span}, y_{span})}{\min(x_{span}, y_{span}) + \epsilon}, \epsilon \in \mathbb{R}, \epsilon > 0 \quad (1)$$

Here, r_{spread} in (1) tries to measure the ratio of the maximum and minimum spreads of distances seen by the LiDAR at a particular robot (x, y) position. Also, ϵ is a small positive number used to avoid division by zero in (1). In this work, $\epsilon = 1 \times 10^{-6}$.

The next features, i.e., σ, δ , and κ , are the **standard deviation** (σ) of all the items in S , the **skewness** (δ) of S , and the **kurtosis** (κ) of S . These statistical quantities are calculated using well-known standard Python libraries. The next feature, **roughness** (ρ), is calculated as follows.

$$\rho = \frac{1}{N-1} \sum_{i=0}^{N-1} (|r_{i+1} - r_i|), N = 360 \quad (2)$$

This can be thought of a measure of the local texture or clutter of the environment obtained by calculating the mean absolute difference of range measurements of adjacent beams in a LiDAR scan.

The feature **small jump rate** (J_s) detects small changes in the distance of the objects measured in an environment. This feature counts the number of changes in range measurements between adjacent beams in a LiDAR scan that exceed a selected sensitivity threshold η . In this work, η is selected to be 0.1 m. This value was chosen to be effectively larger than the LiDAR sensor's intrinsic noise floor (~ 0.02 m) to prevent false positives on planar walls while remaining small enough to reliably detect structural features such as table legs, door frames, corners, and other small obstacles. J_s is calculated as follows. Let,

$$f(x) = \begin{cases} 1, & \text{if } x > \eta, \\ 0, & \text{otherwise} \end{cases} \quad \eta = 0.1 \quad (3)$$

$$J_s = \frac{1}{N-1} \sum_{i=0}^{N-1} f(|r_{i+1} - r_i|), N = 360 \quad (4)$$

The entropy (H) feature measures the information content in a LiDAR scan. For this, we compute the Shannon entropy [30] over the histogram of range measurement data $\{r_0, r_1, r_2 \dots, r_{359}\} \in S$. The formula is presented in (5).

$$H = - \sum_{k=1}^B (p_k \log_2(p_k)), \text{ where} \quad (5)$$

$$p_k = \frac{c_k}{\sum_{j=1}^B c_j}, k = 1, 2, \dots, B \quad (6)$$

The histogram of the raw LiDAR range data $\{r_0, r_1, r_2 \dots, r_{359}\}$ is divided into a histogram containing $B = 20$ bins, with a bin width of $\sim 17.5 \text{ cm}$ ($\frac{3.5 (\text{LiDAR max range})}{20} = 0.175 \text{ m}$). The term p_k represents the probability of range values falling into the k^{th}

bin. To calculate p_k , the ratio of the number of samples in the k^{th} bin to the total number of range samples (r_i) is used. In our case, the total number of samples is $\sum_{j=1}^B c_j = 360$. A low value of H indicates low information content in a scan S , e.g., a uniform environment. But a high entropy value H signifies high information content, e.g., a spatially complex area with diverse range measurements. Entropy does not directly provide the context related to coordinate references; instead, it serves as another criterion for disambiguating LiDAR scan data so that an ML model may effectively associate such information with robot location. It is hypothesized that this can help distinguish (x, y) coordinate pairs where only very subtle differences in the environmental geometry around such a point may be picked up by a LiDAR scan S .

The large jump rate (J_l) works similarly to the small jump rate (J_s), except that no normalization is performed, and in this case, the sensitivity threshold η is chosen to be **1 m**. This feature helps to identify if there are sudden and sharp changes between consecutive range measurements. The exact definition is as below.

$$f(x) = \begin{cases} 1, & \text{if } x > \eta, \\ 0, & \text{otherwise} \end{cases} \quad \eta = 1 \quad (7)$$

$$J_l = \sum_{i=0}^{N-1} f(|r_{i+1} - r_i|), N = 360 \quad (8)$$

The last feature, Fourier transform ($\mathcal{F}$), represents information related to the discrete Fourier transform (DFT) of range measurements $\{r_0, r_1, r_2, \dots, r_{359}\}$ in a LiDAR scan S . The DFT is obtained via the standard packages in Python 3.13.9. Let d represent the DFT of S . Further, let the top 10 components of d with the largest magnitude be d_{10} . The spread of the values in d_{10} is calculated by using the standard deviation operator, and this value is used as the feature $\mathcal{F}$. All the above-mentioned eleven handcrafted features are represented visually in Appendix A for all three maps.

While the raw **360°** LiDAR scan range data provides the granular depth resolution required for coordinate regression, relying solely on raw distance measurements often causes ML models to memorize specific range configurations rather than learning generalizable patterns about spatial geometries. To mitigate this, the above-mentioned 11 explicit feature descriptors are appended to the raw LiDAR data (hereafter called *hybrid feature representation*) for the purpose of creating a dataset to train ML models. This is done so that a high-dimensional noisy input LiDAR scan signal is also imbued with some structured features that may have some semantic relationship to describe a particular (x, y) location with respect to its LiDAR scan. It is observed from the literature that some of the existing geometric or statistical feature-based localization methods often rely on edges, corners, or point-cloud-level geometric primitives. However, such features are either computationally expensive or insufficiently discriminative when two LiDAR scans appear similar but correspond to different physical locations. The selection of eleven descriptors in this paper is guided by a qualitative observation aimed at resolving ambiguities that arise when different physical locations produce similar raw LiDAR scans. It is observed from the results reported in this paper that the above appending of 11 features to raw LiDAR data helps to form a reasonably unique fingerprint for each (x, y) location.

3.4. Data Preprocessing and Augmentation

Following the description of the features in Section 3.3, this section presents details related to any preprocessing required prior to using LiDAR scan data to train an ML model.

- (1) **Synthetic Data Augmentation:** To improve the trained model's generalization capabilities and to prevent overfitting, data augmentation is applied. Because LiDAR sensors deployed in real-world scenarios exhibit intrinsic measurement jitters and dropped packets, we attempt to simulate these effects during data augmentation. For every original LiDAR scan range sample, 3 augmented copies are generated, thus quadrupling the overall dataset size. Each augmented copy simulates one of the following: (i) sensor noise, (ii) random dropout, or (iii) label jittering. For the first item, zero-mean Gaussian noise with standard deviation **0.02 m** is added to the LiDAR scan range readings; this simulates sensor noise and inaccuracy/intrinsic measurement jitters. For the second item, simulating the effects of dropped data packets and sudden temporary LiDAR scan beam failures, 2% of the original LiDAR scan range readings are randomly selected, and the range value for them is set to **3.5 m** (i.e., representing maximum sensor range). Finally, for the third item, reference pose coordinates (x, y) are slightly corrupted by adding uniformly distributed random noise, with values in the interval $([-0.002, 0.002])$ m. This simulates minor uncertainties common in SLAM-generated reference pose (x, y) . Column five in Table 2 provides a detailed breakdown of augmented samples for each map.
- (2) **Feature Normalization:** To improve numerical stability and accelerate the gradient descent-based training convergence of ML models, the input features in the datasets are standardized (by calculating the Z-score) using StandardScaler from the scikit-learn library. This transforms the feature vectors so they have zero mean and unit variance.

3.5. Model Architecture and Training

To evaluate the performance of the proposed hybrid feature representation approach, both classical regression models and DL models are used. The following classical models were implemented using the scikit-learn framework: linear regression, ridge regression, K-nearest neighbors (KNNs, $k = 5$) and random forest (250 trees). In addition, three distinct deep learning architectures, ranging from simple fully connected networks to hybrid convolutional models, were employed in PyTorch. All models accept the standardized input vectors $x \in R^{371}$ (comprising 360 raw 2D LiDAR range measurements and the 11 handcrafted feature descriptors described in Section 3.3) and estimate the 2D pose, $\hat{\xi} = (\hat{x}, \hat{y})$, where $\xi = (x, y)$ represents the actual pose. The three DL models used are described below.

- (1) **Multi-Layer Perceptron (MLP):** A fully connected baseline deep network was implemented to model global nonlinear relationships in the feature space. The architecture consists of three hidden layers with 512, 256, and 128 neurons, respectively, each followed by ReLU activation functions. Dropout with probability ($p = 0.3$) is applied after intermediate layers to mitigate overfitting. Then, the final layer outputs a 2D pose estimate $\hat{\xi}$.
- (2) **1D Convolutional Network (Conv1D):** To exploit the sequential structure of the LiDAR range scan, Conv1D is applied to the full 371-dimensional standardized input, which is treated as a 1D signal. The network consists of three convolutional blocks with kernel size $k = 5$ and progressively increasing channel depth (1, 16, 32, 64). Max-pooling layers with stride 2 are applied after the second and third

convolutional blocks to reduce the spatial resolution. The resulting feature map is flattened and passed through a shallow fully connected regression head to predict the pose $\hat{\xi}$. This design allows the model to detect local geometric features such as edges and wall transitions.

- (3) Hybrid ConvMLP: A ConvMLP [31] hybrid architecture is implemented to balance local geometric feature extraction with global regression capacity. The model consists of a three-convolutional-layer front-end with kernel sizes (7, 5, 5), each followed by ReLU activation with stride 1 and padding (3, 2, 2), respectively. Two MaxPool1d layers with window size = 2 are applied to down-sample the input vector size. Dropout with probability ($p = 0.3$) is applied after each pooling stage to mitigate overfitting. The resulting feature map is flattened and passed through an MLP head with two hidden layers (512 and 256 neurons). Then, the final layer outputs a 2D pose estimate $\hat{\xi}$.

All DL models were trained using the Mean Squared Error (MSE) loss function $L = \frac{1}{N} \sum ||\xi - \hat{\xi}||^2$, where $\hat{\xi}$ is the predicted output pose and ξ is the true (reference) pose. The Adam optimizer was used with the above MSE loss function, with a learning rate of $\mathcal{L} = 1e^{-3}$ and a decay weight of $\omega = 1e^{-5}$ for regularization. Training was performed on a CPU for 70 epochs with a batch size of 64. The dataset was split into training and test sets in an 80% / 20% ratio without shuffling, ensuring that the test set represented robot poses that are sequentially separated from the training data. The intention of this kind of split is to evaluate the model's ability to generalize to regions of the map that were not observed during training. From the training portion (80%), 15% of the samples were randomly selected to form a validation set. During model training, samples within the training set were randomly shuffled to prevent the model from exploiting the sequential nature of the trajectory data. Since the dataset consists of continuous time-based robot motion, adjacent samples may exhibit strong correlations that could lead the model to memorize trajectory patterns rather than learn the underlying spatial relationship between hybrid features and robot pose. The performance of each model was evaluated using a Root Mean Square Error (RMSE) metric, which measures the Euclidean distance (in meters) between predicted and reference robot positions. Based on the above-mentioned evaluation metrics, the best-performing model was selected and used during inference to predict the live pose of the robot.

3.6. Deploying Trained Models for Real-Time Pose Inference

As the robot travels around a map for which LiDAR scan range data was collected for training, the live inference pipeline operates sequentially on incoming LiDAR scan range data. The system receives a raw LiDAR scan (S_t) at time t . The 360 raw range distances are extracted, and the 11 handcrafted features mentioned in Section 3.3 are computed. For inference, i.e., estimating robot pose using a trained model with live LiDAR scan data, the features mentioned in Section 3.3 are generated from the range measurements at a robot position during navigation. Then, the (360 + 11) features are normalized as mentioned in the 2nd item in Section 3.4. The normalized vectors are passed through the trained network to estimate the instantaneous pose $\hat{\xi}_t = (\hat{x}_t, \hat{y}_t)$ at time t . This ML-based prediction is then immediately passed to the following augmented Recursive Extended Kalman Filter (REKF) to get a better-quality estimate of the robot's current pose.

3.7. Recursive Extended Kalman Filter

To refine the noisy pose predictions from the ML models, a custom recursive EKF is used to denoise the estimated pose. Unlike a standard EKF that only fuses raw odometry

with other measurements to estimate robot pose, the proposed recursive EKF utilizes a recursive measurement augmentation strategy to better suppress temporal jitter.

3.7.1. The Standard EKF

The baseline EKF [32] maintains the state vector $\tilde{x}_k = [x, y, \theta]^T$. This model uses odometry and ML-based pose predictions as direct and independent observations of the robot's position. The discretized kinematic model for a WMR is written as follows [33,34]:

$$\begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} x_{k-1} + Tv_k \cos(\theta_k) + \mu_{1k} \\ y_{k-1} + Tv_k \sin(\theta_k) + \mu_{2k} \\ \theta_{k-1} + T\omega_k + \mu_{3k} \end{bmatrix}, \text{ where } (T = t_{k+1} - t_k) \quad (9)$$

$$z_k = \begin{bmatrix} z_{1k} \\ z_{2k} \\ z_{3k} \\ z_{4k} \\ z_{5k} \end{bmatrix} = \begin{bmatrix} x_k^{odom} + \ell_{1k} \\ y_k^{odom} + \ell_{2k} \\ x_k^{ML} + \ell_{3k} \\ y_k^{ML} + \ell_{4k} \\ \theta_k^{odom} + \ell_{5k} \end{bmatrix}, \text{ where } z_k \in \mathbb{R}^5 \quad (10)$$

The process noises μ_{ik} and measurement noises ℓ_{ik} are assumed to be zero mean white, identically independently distributed with covariances represented via the Q_k and R_k matrices, respectively. When using the EKF, the following matrices are calculated, as

shown below. Let $f_k(\tilde{x}_k, \mu_k) = \begin{bmatrix} x_{k-1} + Tv_k \cos(\theta_k) + \mu_{1k} \\ y_{k-1} + Tv_k \sin(\theta_k) + \mu_{2k} \\ \theta_{k-1} + T\omega_k + \mu_{3k} \end{bmatrix}$; then, $F_k = \frac{\partial f_k(\tilde{x}_k, \mu_k)}{\partial \tilde{x}_k} \Big|_{\hat{\tilde{x}}_k^+}$,

where $\hat{\tilde{x}}_k^+$ is the a posteriori estimate from the EKF. Also, $L_k = \frac{\partial f_k(\tilde{x}_k, \mu_k)}{\partial \mu_k} = \text{diag}[1, 1, 1]$, and with $\text{var}()$ representing the variance of the respective random variable representing noise; the state noise and output noise variance matrices are $Q_k = \text{diag}[\text{var}(\mu_1), \text{var}(\mu_2), \text{var}(\mu_3)]$, $R_k = \text{diag}[\text{var}(\ell_1), \text{var}(\ell_2), \text{var}(\ell_3), \text{var}(\ell_4), \text{var}(\ell_5)]$

respectively. Further, let $h_k(\tilde{x}_k, \ell_k) = \begin{bmatrix} x_k^{odom} + \ell_{1k} \\ y_k^{odom} + \ell_{2k} \\ x_k^{ML} + \ell_{3k} \\ y_k^{ML} + \ell_{4k} \\ \theta_k^{odom} + \ell_{5k} \end{bmatrix}$; then, $H_k = \frac{\partial h_k(\tilde{x}_k, \ell_k)}{\partial \tilde{x}_k} \Big|_{\hat{\tilde{x}}_k^-}$,

where $\hat{\tilde{x}}_k^-$ represents the a priori state estimate from the EKF. And $M_k = \frac{\partial h_k(\tilde{x}_k, \ell_k)}{\partial \ell_k} \Big|_{\hat{\tilde{x}}_k^-} =$

$\text{diag}[1, 1, 1, 1, 1]$. The matrices F_k and H_k are $F_k = \begin{bmatrix} 1 & 0 & -Tv_k \sin(\theta_k) \\ 0 & 1 & Tv_k \cos(\theta_k) \\ 0 & 0 & 1 \end{bmatrix}$ and $H_k =$

$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$, respectively. Now, the standard EKF equations can be used as follows:

$$P_k^- = F_{k-1} P_{k-1}^+ F_{k-1}^T + L_{k-1} Q_{k-1} L_{k-1}^T \quad (11)$$

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + M_k R_k M_k^T)^{-1} \quad (12)$$

$$\hat{\tilde{x}}_k^- = f_{k-1}(\hat{\tilde{x}}_{k-1}^+, K_{k-1}, 0) \quad (13)$$

$$\hat{\tilde{x}}_k^+ = \hat{\tilde{x}}_k^- + K_k (Z_k - h_k(\hat{\tilde{x}}_k^-, 0)) \quad (14)$$

$$P_k^+ = [(P_k^-)^{-1} + H_k^T R_k^{-1} H_k]^{-1} \quad (15)$$

The initial guess $\hat{\tilde{x}}_0^+$ is assigned $[x, y, \theta]^T$ at the initial time from the odometry-based readings, and P_0^+ is initialized as **diag** [1, 1, 1]. It is also worth noting from [32]

that the assumptions of the statistical properties of noise are easily relaxable, and any specific distribution is not strictly necessary for the results related to state estimation using KFs to hold.

3.7.2. The Proposed Augmented Recursive EKF

To address the high-frequency temporal jitter (noise) in the predicted output pose obtained from the ML models, the augmented recursive EKF architecture below is proposed. While most of the items defined above for the standard EKF remain the same, the measurement vector $\mathbf{z}_k$ is augmented as follows.

$$\mathbf{z}_k = \begin{bmatrix} z_{1k} \\ z_{2k} \\ z_{3k} \\ z_{4k} \\ z_{5k} \\ z_{6k} \\ z_{7k} \\ z_{8k} \\ z_{9k} \end{bmatrix} = \begin{bmatrix} x_k^{odom} + \ell_{1k} \\ y_k^{odom} + \ell_{2k} \\ x_k^{ML} + \ell_{3k} \\ y_k^{ML} + \ell_{4k} \\ 0.5(x_k^{odom} + \hat{x}_k^{1+}) + \ell_{5k} \\ 0.5(y_k^{odom} + \hat{y}_k^{2+}) + \ell_{6k} \\ \bar{z}_{xk} + \ell_{7k} \\ \bar{z}_{yk} + \ell_{8k} \\ \theta_k^{odom} + \ell_{9k} \end{bmatrix}, \text{ where } \mathbf{z}_k \in \mathbb{R}^9. \quad (16)$$

In the above equation, $\hat{x}_k^{1+}$ and $\hat{y}_k^{2+}$ denote the aposteriori estimates of the robot's x, y position coordinates as obtained by the standard EKF process. Also, $\bar{z}_{xk} = \frac{1}{N_w} \sum_{i=k-N_w}^k z_{5i}$, $k \geq N_w$, and $\bar{z}_{yk} = \frac{1}{N_w} \sum_{i=k-N_w}^k z_{6i}$, $k \geq N_w$. Please note that for $1 < k \leq N_w$, $\bar{z}_{xk} = x_k^{odom}$ and $\bar{z}_{yk} = y_k^{odom}$. The point of augmenting the $\mathbf{z}_k$ vector shown above is as follows. The first four components are the same as the standard EKF. The next two items (5, 6) construct readings for the x and y position coordinates by averaging the respective odometry readings and the respective standard EKF state estimates. This serves as an immediate smoothing operation of any jitter in the state estimates, which also considers readings from odometry. Further, components 7 and 8 smooth the readings in the 5th and 6th components by using a moving average of window size N_w . This incorporates a further smoothed version of the readings in components 5 and 6 into the output vector. Element 9 of $\mathbf{z}_k$ is the yaw angle obtained from odometry. The aim is that incorporating such repeated data about the x, y coordinates from several sources, and using several different smoothing techniques with different lags, should allow this recursive EKF to produce smoother estimates of the x, y position compared to both ML-based inferencing, and ML-based inferencing results passed through a standard EKF.

$$\mathbf{H}_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (17)$$

In this work, for the window size $N_w = 10$ and corresponding to $\mathbf{z}_k$, the matrix $\mathbf{H}_k$ used for the recursive EKF is as shown in equation (17).

4. Experimental Evaluation

The methodology described in the above section is tested using the setup and experiments as described further. Additionally, a complete demonstration of the proposed method, including system setup, teleoperation, data collection, preprocessing,

model training, and real-time inferencing, is provided as Supplementary Materials Video S1.

4.1. System Setup and Implementation

The experimental evaluation is conducted using a Turtlebot3 Burger mobile robot. The robot itself and the remote workstation (Intel core i7, 16 GB RAM), which connects to the robot, use ROS2 (Humble). The robot is remotely operated by manually issuing motion commands during both the data collection and testing phases. To ensure a direct performance comparison between different pose estimation approaches, all the processing for pose estimation in this work (ML-based, EKF-based, ROS2 Cartographer-based, or AMCL-based) is done on the same workstation. Please note that once the ML-based model is ready, it can be deployed for ML-based pose estimation directly on the onboard Raspberry-Pi of the Turtlebot3 Burger mobile robot because the inferencing operation is not computationally resource-intensive; such direct on-robot deployment of the entire localization approach is left for future efforts, with the current deployment clarified below.

For the purposes of this work, as mentioned above, the inference engine is implemented as a standalone ROS2 node on the workstation. This inferencing node subscribes to the LiDAR sensor topic `/scan` with a “Best-Effort” Quality of Service (QoS) profile. During inference, this node processes incoming laser scans through the feature extraction pipeline, executes the forward pass of the neural network, and publishes the predicted pose estimates to a custom topic (`/localization_eval`) for analysis. To validate performance, the system synchronously listens to the (`/tf`) transform tree provided by the Cartographer SLAM backend, allowing for immediate comparison between the ML prediction and the SLAM-generated reference pose. Also, the ML-predicted pose outputs are passed through the above-mentioned standard and augmented recursive EKFs to generate different smoothed versions of the ML-predicted pose output. A similar topic configuration is utilized during the data collection phase to generate the reference pose labels for training. It is important to note that the Cartographer SLAM poses used during the data collection serve as reference labels for supervised training and contain both high-frequency noise and small drift. However, the augmented recursive EKF applied during inference counters such effects. Any systematic bias present in the SLAM trajectory simply sets a lower bound on achievable accuracy. This is because the model is trained using SLAM-based reference poses; thus, it learns to approximate the SLAM trajectory and cannot outperform the accuracy of the labels it is trained on. This does not cause any generalization issues for the proposed methodology, as the model primarily learns a mapping between LiDAR data geometry and planar pose. Further, if there are any indoor beacon-based systems available, then data from such systems can be used during the training phase to provide reference pose labels instead of using the SLAM-based reference labels. This has the potential to improve accuracy further.

4.2. Testing Protocols and Environments

To ensure a comprehensive evaluation, the system was tested across the three distinct environments (Maps 1, 2 and 3, as shown in Figures 3, 4 and 6, respectively), as detailed in Section 3.1. These environments represent scenarios ranging from feature-sparse open spaces, as in Map 1, narrow corridors with junctions, as in Map 2, and clutter-dense real-life laboratory settings with irregular obstacles, as in Map 3.

4.2.1. Benchmarking

For rigorous testing, the proposed ML-based localization approach is evaluated against the following four localization techniques: wheel odometry, which is purely a

dead-reckoning-based method; Adaptive Monte Carlo Localization (AMCL), which represents the performance of a classical particle filter; a standard Extended Kalman Filter (EKF); and the proposed recursive EKF.

4.2.2. Experimental Protocol

Before the experimental protocols are executed, it is important to pick the best ML-based pose estimation technique from those mentioned in Section 3.5. Prior to such a comparison of the proposed ML-based techniques, the effectiveness of the proposed features was examined. This was done by performing a feature importance analysis using the random forest technique with LiDAR-based pose estimation and an ablation study using the hybrid features. The associated results are available in Figures 8–11.

Then, the testing protocol included moving the robot in three distinct maps, in different paths. These are shown in Figures 12–14. The first path shown in Figure 12 looks approximately like the number 6. Figure 13 shows an L-shaped path in an L-shaped map, and Figure 14 shows a closed path in the third map, which is the actual lab setting. All three types of paths mentioned were tested in each map. Additionally, to account for system latency and sensor noise and to observe average performance across multiple similar trajectories, the robot was driven on Path1 for five separate trials with Map1. The metrics for all the above-mentioned tests are presented further in the paper. Furthermore, while the robot navigated during any one of the above-mentioned experiments, all the sensor data was saved to an ROS2 bag. When comparing with other approaches like AMCL, the ROS2 bag data was replayed, and the pose estimates were constructed using AMCL and the proposed ML-based techniques. The ROS2 bag data includes the robot's raw sensor streams, i.e., LiDAR, odometry, transform (TF) data, timestamps, and cartographer SLAM-based reference pose estimates. Such data were also recorded during manual teleoperation and replayed offline for comparison.

Beyond testing standard navigation, the robustness of the proposed methodology was evaluated using a “kidnapped robot” test. In this scenario, the robot was instantaneously displaced physically to an unknown location without updating the robot's internal states. The proposed ML-based pose estimation system was evaluated based on its ability to successfully re-localize the robot solely based on the incoming LiDAR data so that the updated pose estimates of the robot quickly converge to the expected reference pose estimates.

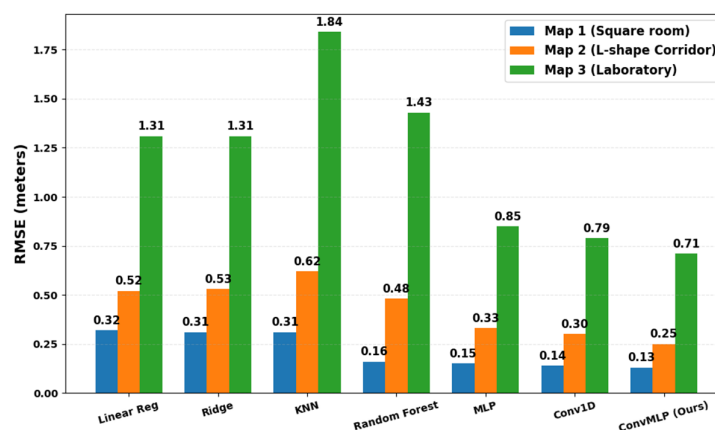


Figure 8. Localization accuracy comparisons between traditional and deep learning models.

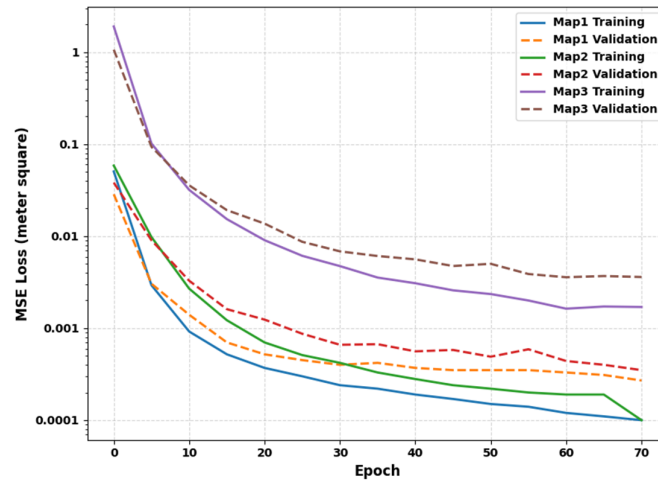


Figure 9. Training and validation loss curves of ConvMLP models for three maps.

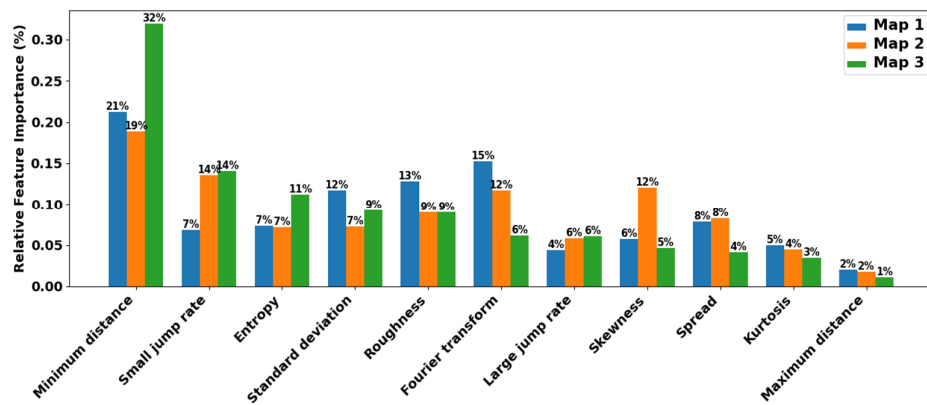


Figure 10. Handcrafted feature importance analysis for pose prediction across three maps; raw LiDAR scan range data might suffer from aliasing.

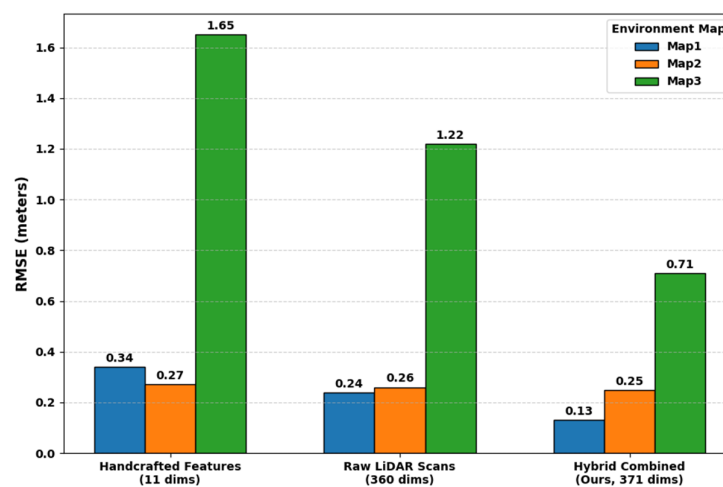


Figure 11. Impact of different input feature modalities on pose prediction.

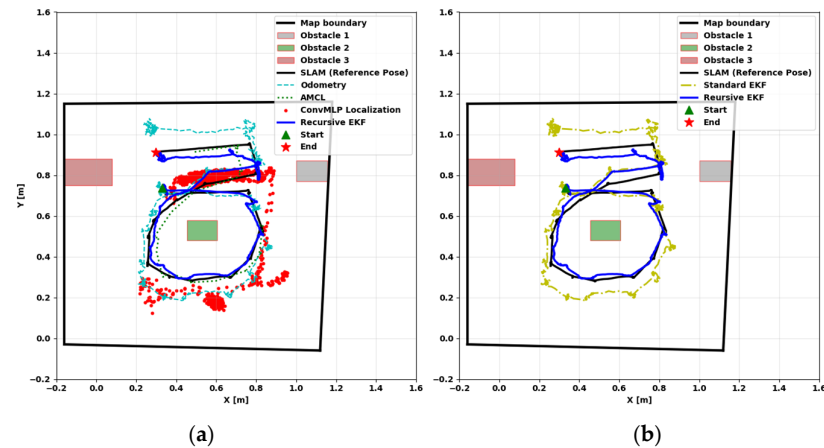


Figure 12. Map1 (Path1): (a) localization comparison using SLAM, odometry, AMCL, the proposed ConvMLP method, and recursive EKF; (b) localization comparison using SLAM, standard EKF, and augmented recursive EKF.

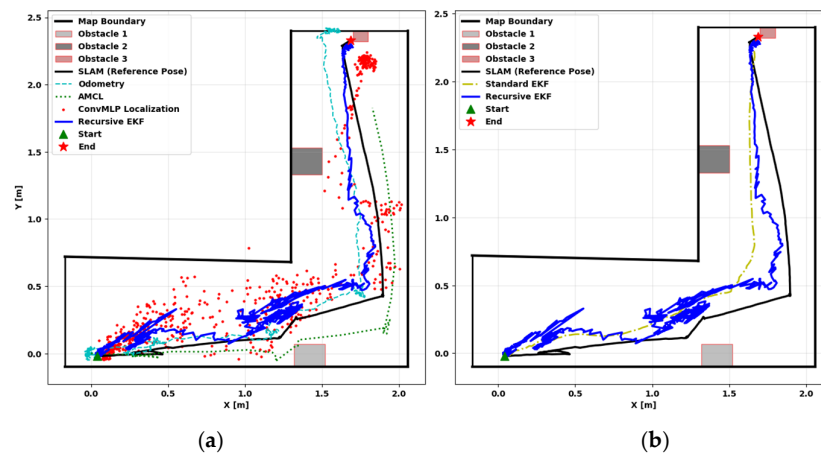


Figure 13. Map2 (Path2): (a) localization comparison using SLAM, odometry, AMCL, the proposed ConvMLP method, and recursive EKF; (b) localization comparison using SLAM, standard EKF, and augmented recursive EKF.

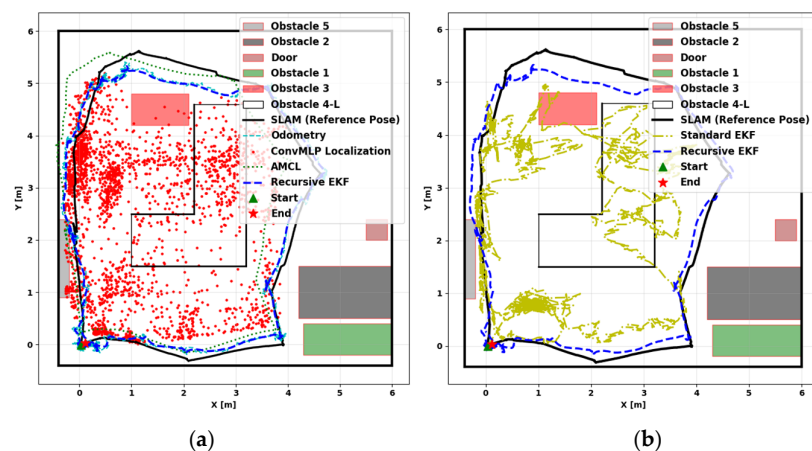


Figure 14. Map3 (Path3): (a) localization comparison using SLAM, odometry, AMCL, proposed ConvMLP method, and augmented recursive EKF; (b) localization comparison using SLAM, standard EKF, and recursive EKF.

5. Experimental Evaluation and Results

The following subsections first present results related to comparing the ML models used for robot localization based on LiDAR range data. The motivation behind comparing the performance of such ML models is to select the best model so that only the best model is deployed on the real-life hardware for robot localization. Then, the importance of each of the 11 features mentioned in Section 3.3 is analyzed. Further, the effect of only using LiDAR data, only using the 11 features, or performing ML-based localization when LiDAR data and the 11 features mentioned in this work are used (i.e., hybrid feature representation) is compared.

5.1. Performance Comparison of ML Models

To identify the optimal ML model architecture, the localization performance of the following ML models is evaluated. Four traditional regression models (linear, ridge, KNN and random forest) and three DL-based models (MLP, Conv1D and ConvMLP) are compared. To compare these models, actual LiDAR data and the reference pose are collected in the three maps, as described in Section 3. The LiDAR and reference pose data collected for each of the three maps, as shown in Figures 3, 4 and 6; is split for training, validation and testing, as described in Section 3.5. The results of the testing, i.e., predicting robot (x, y) coordinates when the trained model is given an unknown test set of LiDAR scan range data, are summarized in Figure 8. The linear, ridge, and KNN regression models clearly underperformed in Maps 2 and 3, exhibiting average RMSE values > 0.45 m. Similarly, for Map1, these three models had average RMSE values > 0.25 m. On the other hand, the MLP and Conv1D models showed average RMSE values > 0.25 m for Maps 2 and 3 and average RMSE values around 0.15 m and 0.14 m, respectively, for Map1. Random forest showed an average RMSE around 1.43 m for Map3, around 0.48 m for Map2, and around 0.16 m for Map1. The overall best performance is shown by ConvMLP with average RMSE values around 0.71 m for Map3, 0.25 m for Map2 and around 0.13 m for Map1. The performance gaps between MLP, Conv1D and ConvMLP are very small. However, ConvMLP performed better than the rest of the models overall and had the best performance in Map3, which is the real-life lab environment.

Although ConvMLP outperformed the other models, the performance of the random forest model is close to ConvMLP in Map1. So, in Section 5.2, we analyze the importance of the 11 features using random forest. Note that all further sections after this section report results using the best-performing model, i.e., ConvMLP, as identified above.

To further analyze the training behavior of the selected ConvMLP model, the training and validation loss curves for the three maps are shown in Figure 9. The models were trained using the MSE loss function, as described in Section 3.5, for 70 epochs. At the beginning of training (epoch 0), the models exhibit relatively high loss values, with the training MSE (RMSE) starting at approximately **0.05053 m² (0.22 m)**, **0.05853 m² (0.24 m)**, and **1.90288 m² (1.37 m)** for Maps 1, 2, and 3, respectively, while the validation MSE (RMSE) begins at approximately **0.02857 m² (0.16 m)**, **0.03823 m² (0.19 m)**, and **1.06500 m² (1.03 m)** for Maps 1, 2, and 3, respectively. As the model training progresses, in all three maps, both the training and validation losses decrease rapidly, indicating that the model quickly learns the relationship between hybrid features and robot pose. Then, after approximately 40–50 epochs, the loss values gradually stabilize, indicating convergence of the model parameters. By the end of training (epoch 70), the MSE (RMSE) losses reduce to approximately **0.00010 m² (0.01 m)**, **0.00010 m² (0.01 m)**, and **0.00170 m² (0.04 m)** for training and approximately **0.00027 m² (0.01 m)**, **0.00035 m² (0.01 m)**, and **0.00360 m² (0.06 m)** for validation in Maps 1, 2, and 3, respectively. The small gap between the training and validation losses indicates that the model does not overfit the training data and maintains

consistent performance on unseen samples from the validation set. Map 3 exhibits higher loss values compared to Map1 and Map2 due to the increased structural complexity of the real laboratory environment. Overall, the consistent convergence behavior across all three maps demonstrates stable training of the ConvMLP model.

It should be noted that the validation set consists of randomly sampled data from the training dataset, representing spatial regions that are close to those observed during training. In contrast, the test sets described in Section 3.5 contain trajectory segments that were not included in the training data. Therefore, the reported test RMSE values of the ConvMLP model for all three maps, as shown in Figure 8, reflect the model's ability to localize the robot's pose in previously unseen spatial regions of the map, which represents a more realistic deployment scenario.

5.2. Feature Importance Analysis

To observe the effect of each of the 11 proposed handcrafted feature descriptors mentioned in Section 3.3, random forest's built-in feature importance analysis is used by measuring mean decrease impurity (MDI). As illustrated in Figure 10, the analysis reveals the distinct tiers of feature relevance. The most critical feature descriptor is the minimum distance (d_{min}), which takes the leading role in the model's decision-making ability. This highlights that proximity to the nearest obstacle plays an important role in this work. In contrast, the spread (r_{spread}), kurtosis (κ), and maximum distance (d_{max}) features have low contribution in all the three maps. All the other features have some contribution for the case of every map, but the feature contributions vary with respect to map type. The reason for using so many features in addition to the LiDAR scan range data is that such features may disambiguate geometrically similar areas, such as identical hallways or corridors, by reducing false positives and improving generalization to unseen poses.

5.3. Feature Ablation Study

To empirically validate the architectural design of the ConvMLP model, an ablation study was conducted on all three maps by comparing the model performance across three input feature modalities: (i) 11 proposed handcrafted feature descriptors (as in Section 3.3) only, (ii) raw LiDAR scan range data only, and (iii) hybrid features (11 handcrafted feature descriptors along with LiDAR scan range data). As illustrated in Figure 11, when the model inputs are only the 11 handcrafted features, the ConvMLP model performed at its worst (RMSE = 0.34 m for Map1, RMSE = 0.27 m for Map2, and RMSE = 1.65 m for Map3). When only raw LiDAR scan range data is used, the ConvMLP model achieved significantly better accuracy (RMSE = 0.24 m for Map1, RMSE = 0.26 m for Map2, and RMSE = 1.22 m for Map3). The hybrid approach achieved the best performance (RMSE = 0.13 m for Map1, RMSE = 0.25 m for Map2, and RMSE = 0.71 m for Map3). This shows that the raw scans provide the necessary high-resolution data, while the 11 handcrafted features act as a high-level spatial anchor.

5.4. Computational Efficiency and Performance Analysis

If a localization system is to be used as a fallback safety system in case the main localization system becomes inoperative, it helps if such a localization system is computationally lightweight so it can run on relatively resource-constrained computational platforms/embedded hardware or on small WMRs. To check this, the runtime computational performance of the proposed ConvMLP-based localization method is compared against the following methods: (i) AMCL, (ii) ROS2 Cartographer SLAM, and (iii) odometry. The results of the computational performance of these algorithms are summarized in Table 3. As mentioned in Section 4.1, the same workstation on which the proposed ConvMLP model is tested is also used to compare the

computational performance of all the algorithms mentioned. The proposed ConvMLP-based method demonstrates superior performance by achieving a mean total latency of 3.5 ms per scan, corresponding to an update rate of ~280 Hz. Detailed profiling of total latency (3.5 ms) is divided into three categories: the geometric feature extraction during inference time consumes 0.86 ms, scaling the features takes 0.08 ms, and the model inference, i.e., the forward pass of the inputs through the model and predicting the (x, y) coordinates of the robot, requires 2.56 ms. This shows the proposed system provides near-instantaneous pose regression.

As visible in Table 3, the proposed method is many times faster than the existing AMCL method, which has an update rate of ~0.45 Hz. The ROS2-based Cartographer SLAM has an update rate of ~41 Hz for localization, whereas the odometry-based localization has a ~20 Hz update rate. Note that the reported runtime performance was measured on an Intel i7 CPU workstation, as mentioned earlier. This comparison is meaningful because all baseline methods were evaluated under identical CPU-only conditions, allowing a fair relative assessment of computational efficiency. Although absolute latency might be higher on low-cost embedded platforms, the proposed method has a lightweight model, requiring no GPU, and is one step away from full deployment of the proposed pipeline onboard embedded robotic hardware, as planned in future work. Further, as shown in Table 3, the performance enhancement of the proposed method has a trade-off in terms of memory utilization. However, given the size of current-day onboard memory modules on low-cost embedded platforms, such a trade-off is a worthy investment to get a faster update rate for localization, without requiring any beacon-based infrastructure alterations for an indoor localization system. Note that the ~220 MB memory requirement of the proposed method includes all the PyTorch and Python runtime libraries. But the actual trained ConvMLP model file size is extremely compact (13 MB). This confirms the feasibility of deploying the model as a lightweight fallback localization method on resource-constrained platforms by replacing the runtime libraries with more memory-efficient C++ equivalents [35]. Additionally, as shown in Table 3, the CPU utilization of the proposed ConvMLP method is comparable to ROS2 Cartographer SLAM. There are, however, DL-based methods that use images, LiDAR data from several datasets, and trained large models [36]. While such methods may produce lower localization RMSE; corresponding metrics related to training dataset size, training time, computation time, memory requirements, or localization update rate are usually not available [37–41].

Table 3. Computation performance of localization methods.

Methods	Update Rate (Hz)	CPU Utilization (%)	Memory (MB)
Proposed ConvMLP, LiDAR-based Localization	~280	~3–5%	~220
AMCL	~0.45	~7–10%	~140
Cartographer SLAM	~41	~3%	~50
Odometry	~20	<1%	~10

5.5. Trajectory Analysis and Accuracy

To validate the localization capability of the proposed ConvMLP localization method, extensive trajectory tracking experiments on the three distinct indoor environments shown in Section 3.1 are performed. The experiments compare localization results using the proposed ConvMLP method, AMCL, odometry, standard EKF, and the proposed recursive EKF. Note that AMCL and odometry are considered as standard baselines, and ROS2 Cartographer SLAM is used as a reference pose for all comparisons.

The input to the standard EKF and the proposed augmented recursive EKF are (x, y, θ) from odometry and (x, y) estimates from the ConvMLP method, as described in Section 3.7. The overall comparison results are summarized in Tables 4–6. Prior to this, the results are presented following the experimental protocols described in Section 4.2.2. Trajectory tracking performance is first examined for representative paths (Map1:Path1, Map2:Path2, Map3:Path3) and displayed by plotting pose estimates from the proposed ConvMLP method, odometry, AMCL, the proposed augmented recursive EKF and reference pose, all plotted together for comparison, as shown in Figures 12a, 13a and 14a, respectively. Similarly, Figures 12b, 13b and 14b display the estimated pose trajectories of the standard EKF and proposed augmented recursive EKF for all three cases (Map1: Path1, Map2:Path2, Map3:Path3). These results, using all the above-mentioned techniques, are also presented in Table 4. The baseline set of techniques chosen to evaluate this work was selected because they are readily reproducible, can be executed under identical CPU-only conditions, and are similar to direct pose regression approaches. In contrast, descriptor-based LiDAR methods, such as ICP/NDT, Scan Context or any learning-based methods, either require map matching or GPU-accelerated feature extraction, which itself makes them hard to replicate. For these reasons, the chosen baseline techniques (odometry, AMCL, Cartographer SLAM, EKF variants) represent practical, widely used fallback localization modules suitable for resource-constrained robotic/small desktop localization system deployment.

Additionally, for Map1, five ConvMLP-based localization test runs are carried out by moving the robot along Path1. This is done to verify the repeatability of localization performance in a given map, for the same path, with the results shown in Table 5. To rigorously test ConvMLP-based localization performance across all three paths in all three maps, the robot is run along each of the three paths in each map, and the localization results are cataloged in Table 6. In Tables 4–6, the following metrics are used to compare the localization performance: (i) Root Mean Square Error (RMSE), (ii) Mean Average Error (MAE), (iii) Maximum Error (this represents the maximum localization error per run), and iv) standard deviation of error. Further note that localization error here means the distance between the reference position at a point and its ConvMLP-based estimate.

Finally, tests are also performed using a kidnapped robot scenario in Map2. Such a scenario represents a case where a robot is forcibly displaced from its trajectory and placed in an entirely different location.

Table 4. Performance of different localization methods across maps (Map1: Path1, Map2: Path2, Map3: Path3).

Localization Method	MAP, PATH	RMSE (m)	MAE (m)	Max Error (m)	Standard Deviation (m)
Odometry	Map1, Path1	0.279	0.224	0.583	0.167
	Map2, Path2	0.521	0.410	0.982	0.254
	Map3, Path3	0.785	0.682	1.274	0.369
AMCL	Map1, Path1	0.442	0.692	1.178	0.268
	Map2, Path2	0.612	0.554	1.350	0.381
	Map3, Path3	0.833	0.630	1.514	0.439

Proposed ConvMLP, LiDAR-based Localization	Map1, Path1	0.131	0.104	0.341	0.062
	Map2, Path2	0.253	0.186	0.451	0.146
	Map3, Path3	0.717	0.511	1.32	0.277
Standard EKF	Map1, Path1	0.141	0.123	0.459	0.070
	Map2, Path2	0.198	0.165	0.620	0.112
	Map3, Path3	0.268	0.220	0.979	0.153
Augmented Recursive EKF	Map1, Path1	0.050	0.040	0.150	0.025
	Map2, Path2	0.082	0.065	0.240	0.045
	Map3, Path3	0.060	0.047	0.216	0.029

Table 5. Map1 (Path1): 5 runs on the same path, performance with ConvMLP only (no EKF).

Metrics	Proposed ConvMLP, LiDAR-Based Localization					Average	
	No. of Runs	N1	N2	N3	N4		N5
RMSE (m)		0.131	0.129	0.135	0.128	0.132	0.131
MAE (m)		0.104	0.101	0.108	0.099	0.106	0.104
Max Error (m)		0.341	0.335	0.352	0.320	0.345	0.341
Standard Deviation (m)		0.062	0.060	0.065	0.059	0.063	0.062

Table 6. Performance of the DL-based LiDAR localization method across 3 paths in 3 maps, with ConvMLP only (no EKF).

Metrics	Proposed ConvMLP, LiDAR-Based Localization								
	Map1			Map2			Map3		
	Path1	Path2	Path3	Path1	Path2	Path3	Path1	Path2	Path3
RMSE (m)	0.131	0.134	0.129	0.277	0.253	0.333	0.907	0.802	0.717
MAE (m)	0.104	0.110	0.094	0.207	0.186	0.248	0.711	0.657	0.511
Max Error (m)	0.341	0.365	0.314	0.520	0.451	0.548	2.325	1.878	1.32
Standard Deviation (m)	0.062	0.075	0.068	0.302	0.125	0.179	0.477	0.503	0.277

5.5.1. Localization Accuracy Across Three Maps, with Different Paths

Figure 12 shows the localization results when the robot traverses a number-6-like path in Map 1. As seen in Figure 12a, the raw wheel odometry (cyan dashed lines) begins to drift after the first turn, deviating from the reference pose (black line). Localization via AMCL (green dotted line) generally tracks the path but exhibits small oscillatory drifts. In contrast, the proposed ConvMLP method-based localization results (red dots) cluster around the true path. Figure 12b highlights the localization performance improvements made possible by integrating different EKF's following the ConvMLP stage. When a standard EKF (yellow dash-dot line) is used to fuse odometry readings with ConvMLP estimates, the resultant localization estimates show a smoother trajectory compared to ConvMLP alone, but it still suffers from errors. In contrast, the proposed augmented recursive EKF (blue solid line) fuses a different

set of augmented readings with the ConvMLP estimates, as mentioned in Section 3.7.2, and the localization estimates closely follow the reference.

Figure 13 shows the results of running the robot in an L-shaped trajectory within Map 2. The colors and line styles used for different localization methods are exactly the same as mentioned for Figure 12. In Figure 13a, the odometry drift is clearly observed over the longer path length. The proposed ConvMLP method remains robust, providing consistent trajectory prediction that resembles the reference path, although jitter in the pose estimates is observed. Figure 13b again confirms the benefits of fusing ConvMLP estimates with odometry via the standard and augmented recursive EKF. The proposed augmented recursive EKF maintains superior tracking accuracy compared to the standard EKF when used in combination with the ConvMLP approach.

The map shown in Figure 14 represents a more complex, cluttered environment with larger dimensions and with comparatively less LiDAR range (room dimensions are $7\text{ m} \times 7\text{ m}$, but the LiDAR range is only 3.5 m). Figure 14a reveals that the ConvMLP predictions show increased variance due to the challenging environment. The proposed augmented recursive EKF successfully fuses odometry data with ConvMLP estimates and filters such noisy localization estimates (Figure 14b). This results in a smooth trajectory that significantly outperforms both the standard EKF and direct ConvMLP-based estimation. This shows that the combination of the proposed ConvMLP method with the proposed recursive EKF has the potential to be used not only as a fallback localization approach but also effectively as a primary localization method in indoor/GPS-denied environments, where odometry readings and feature-rich raw LiDAR scan data are available.

The visual results in Figures 12–14 are supported by the quantitative metrics summarized in Table 4. Note that Table 4 compares all of the localization methods on representative paths for each map. The proposed augmented recursive EKF achieves the lowest RMSE across all maps, with a value of 0.050 m for Map1:Path1, compared to 0.141 m for the standard EKF and 0.442 m for AMCL. Even in the challenging Map3: Path3 scenario, the proposed augmented recursive EKF maintains an RMSE of 0.060 m , demonstrating its ability to robustly fuse noisy ConvMLP-based predictions with odometry data for a precise state estimate. Based on this, the combination of ConvMLP-based localization with the augmented recursive EKF is considered the best result. The results presented in Table 4 are visually summarized in Figure 15. As clearly shown in Figure 15, the proposed ConvMLP method alone already performs better than both odometry and AMCL. However, the performance of the ConvMLP method increases even further when combined with a state estimator, with the proposed augmented recursive EKF clearly outperforming the others. To further probe the quality of the ConvMLP-based method, the following results are presented.

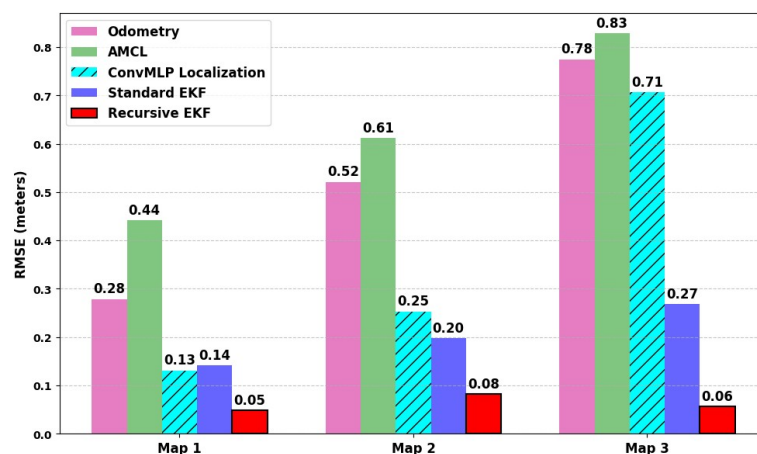


Figure 15. Performance of different localization methods across maps.

5.5.2. Localization Repeatability Assessment

To verify the performance of the proposed ConvMLP method only (without further passing ConvMLP estimates through any filter/state estimator), the robot was made to follow Path1 in Map1 five times. The localization results for these five runs are as detailed in Table 5. As shown in Table 5, the proposed ConvMLP method demonstrated high repeatability, with the Root Mean Square Error (RMSE) fluctuating minimally around a mean of 13 cm in ($\sim 3 \text{ m}^2$) of the Map1 area. The Mean Absolute Error is around 10 cm. The worst-case scenario observed includes an approximately 34 cm maximum localization error. The ~ 6 cm average standard deviation implies that the proposed ConvMLP method's performance is relatively consistent.

To further assess the robustness of the proposed ConvMLP method against varying environmental geometries and trajectory shapes, the method is evaluated across three distinct maps (Map1–3), each tested with three unique path configurations (Path1–3), as discussed in Section 4.2.2. The results are summarized in Table 6. Map1 generally shows better results overall because of its small area, and this is advantageous for the low-range LiDAR onboard the Turtlebot3 Burger. As the sizes of the maps increase relative to the LiDAR range, or as the map path shapes become more complicated, the RMSE, MAE, Max Error, and standard deviation all show corresponding small increases.

5.6. Kidnapped Robot Recovery Analysis

To evaluate the system's utility as a fail-safe mechanism, a "kidnapped robot" scenario is simulated as a stress test for localization algorithms. During robot traversal in Map 2, the robot is manually lifted from coordinates (1.3, 0.8) (marked by the X in Figure 16) and placed at a new location inside Map 2, i.e., coordinates (1.8, 2.3) (marked by a purple triangle mark in Figure 16). Also, the robot traverses in a straight line after drop-off (shown by the dashed purple line). As shown in Figure 16a, AMCL, odometry, and ROS2 Cartographer SLAM fail completely, where wheel odometry (cyan curve) continues to integrate motion from the old (pre-kidnap) location because it is unaware of the displacement. AMCL (green dotted line) shows behavior similar to odometry, but with some corrective action. This is because, as noted by [42,43], classical particle filters struggle during sudden displacements, often requiring a robot to move significantly before the filter can recover. Similarly, Refs. [44,45] highlight that without expensive re-initialization or an external sensor aid, LiDAR-based particle filters behave poorly in kidnap recovery scenarios. In sharp contrast, the ConvMLP-based method proposed (red dots) immediately appears at the correct drop-off zone (1.8, 2.3). This is because the proposed method relies on LiDAR-based fingerprinting for prediction and infers pose solely from such a geometric signature in a map, independent of motion history. Thus, it successfully estimates the new location in a single inference step (~ 3.5 ms inference time). Consequently, the proposed augmented recursive EKF (blue dashed line in Figure 16b) can reject the false odometry data and snap to the correct pose estimate, resuming accurate tracking within seconds. It is observed that AMCL, odometry, and SLAM fail to estimate the robot's position following the drop-off location, even minutes after the robot was dropped off. When these localization algorithms are left continuously running, it is observed that their localization estimates become progressively worse with time.

While the kidnap start location is easily available from the Cartographer SLAM-based reference pose data, the following is worth mentioning regarding finding the post-kidnap drop-off location. In a run prior to the kidnap robot test, the robot is manually

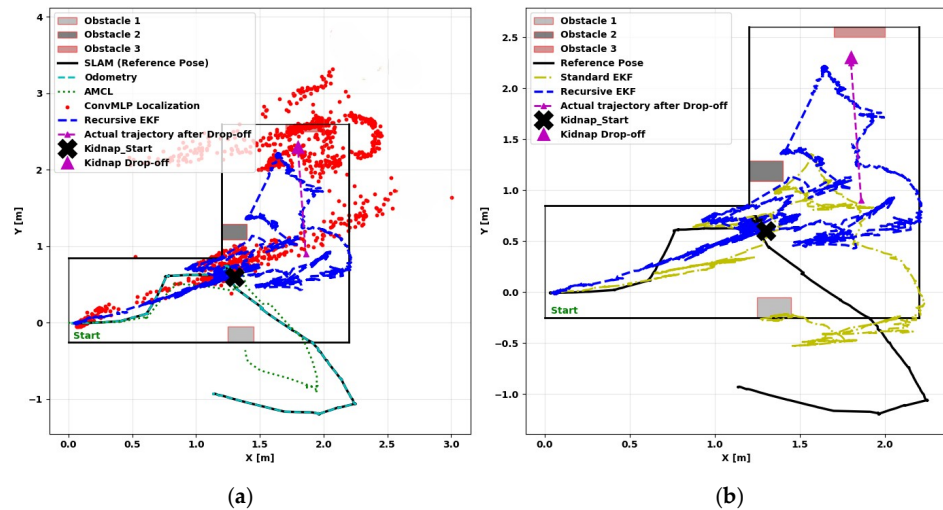


Figure 16. Map2: (a) trajectory comparisons during a kidnapped-robot event; (b) trajectory comparisons between filters.

moved to the drop-off location and then moved along the trajectory after drop-off at a known speed. The corresponding reference positions reported by ROS2 Cartographer SLAM are noted. The physical drop-off location and the path after drop-off are manually marked on the physical Map2 floor for re-use during the kidnap robot test.

6. Conclusions

This research work was motivated by the need for a lightweight and computationally efficient localization method that can operate either as a primary pose estimator on resource-constrained systems or as a reliable fallback mechanism when classical localization fails. To address this gap, this work introduced a hybrid feature representation that combines 1D raw LiDAR scan range data with eleven handcrafted geometric descriptors, enabling a ConvMLP model architecture to learn essential environment-specific spatial signatures for pose prediction. The proposed model achieves a ~280 Hz update rate (~3.5 ms inference latency) with a 13 MB sized, trained ConvMLP model. This makes it suitable for real-time deployment on low-cost platforms. This efficiency is particularly valuable for a fallback localization method, where rapid and stable pose estimates are required to keep the robot operational during failures of the primary localization system without imposing additional computational load on the robot. Additionally, the kidnapped robot experiment directly validates the motivation for fallback localization in this work. While AMCL/SLAM/odometry-based localization methods failed to recover after pose discontinuities, the proposed method produced instant and stable pose estimates, demonstrating its clear advantage in scenarios where the primary localization pipeline collapses.

The method was evaluated across three distinct indoor environments, including a cluttered laboratory, to demonstrate robustness under different operating conditions. The hybrid feature representation proved its effectiveness in disambiguating similar regions by providing unique geometric feature fingerprints across robot poses. The ConvMLP model achieves a 13 cm RMSE; then, the integration of a custom-configured recursive augmented Extended Kalman Filter (EKF) along with the proposed ConvMLP-based method further reduces the RMSE to 5 cm. The advantage of the proposed method over particle filter/SLAM-based methods is that it does not need an explicit map to run localization; it just needs a reference pose, along with a LiDAR scan at such a pose, to learn the relevant features for localization in a given environment.

A limitation of the proposed method is that the trained model assumes a mostly static indoor environment during both mapping and deployment. Significant structural changes and dynamic changes, such as moving people, temporary occlusions, and obstacle displacements, in a map alter the hybrid feature fingerprint. This can increase pose estimation error. However, this can be easily compensated by gathering updated LiDAR data in a quick automated training run in the new/updated environment. Deciding when or how often such training updates are required and addressing dynamic environments remains an important direction for future work. Also, future research may focus on extending the proposed hybrid feature representation, i.e., the developed ConvMLP-LiDAR-augmented recursive EKF chain for geometric fingerprinting in 3D environments. Overall, this work provides a deployable, reproducible, and practical solution for reliable indoor localization on computationally constrained mobile robots. Small laboratory/class-room environments can benefit from such a localization solution, because in such settings a known environmental setup of a certain limited size is expected. Also, the emphasis is on allowing users in such settings to test and develop their robotics capabilities without having to first figure out how to get a robot's position. And generally, in such situations, the cost of installing an elaborate beacon-based system is prohibitive. It is therefore envisioned that the proposed system in this work may help provide localization capabilities in such setups, which may not necessarily be limited to laboratory/class-room settings but any setup facing similar constraints/requirements.

Supplementary Materials: The following supporting information can be downloaded at <https://www.mdpi.com/article/10.3390/app16157718/s1>: Supplementary Video S1—Full demonstration of the ConvMLP, recursive EKF localization pipeline, including teleoperation, data collection, preprocessing, model training and inferencing.

Author Contributions: Conceptualization, S.M. and H.K.G.; methodology, S.M. and H.K.G.; software, H.K.G.; validation, H.K.G. and S.M.; formal analysis, H.K.G. and S.M.; investigation, H.K.G.; resources, S.M.; data curation, H.K.G.; writing: original draft preparation, H.K.G. and S.M.; writing: review and editing, H.K.G. and S.M.; visualization, H.K.G.; supervision, S.M.; project administration, S.M. All authors have read and agreed to the published version of the manuscript.

Funding: No specific funding was received for this work.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is available at <https://github.com/HarshaKGBhat/LiDAR-based-deep-learning-enabled-geometric-fingerprinting-for-indoor-robot-localization->.

Conflicts of Interest: The authors declare no conflicts of interest.

Appendix A

This appendix visually represents the eleven handcrafted geometric features described in Section 3.3. The x-axis represents the scan index. The scan index associates each available three-tuple (x, y, θ) , i.e., robot position and orientation variables, with the corresponding LiDAR scan range values.

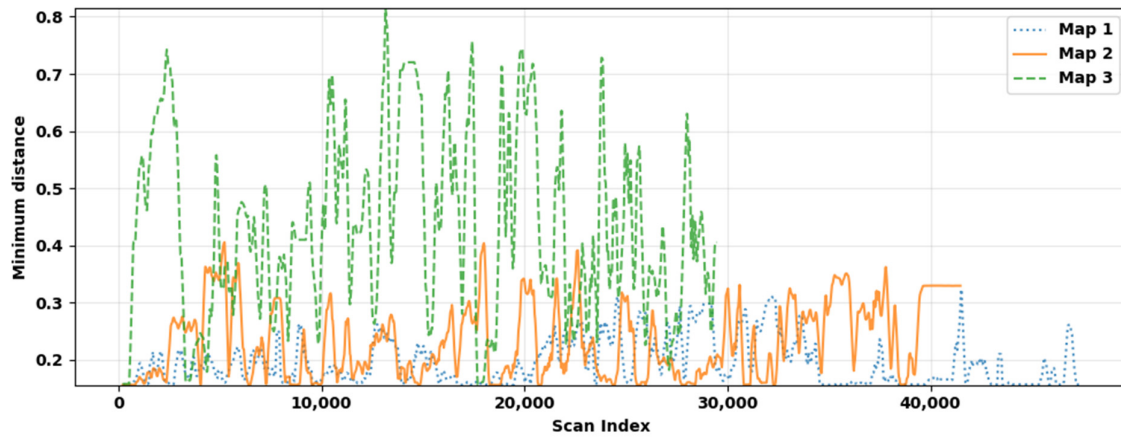


Figure A1. The minimum distance feature_variation across scans on all three maps.

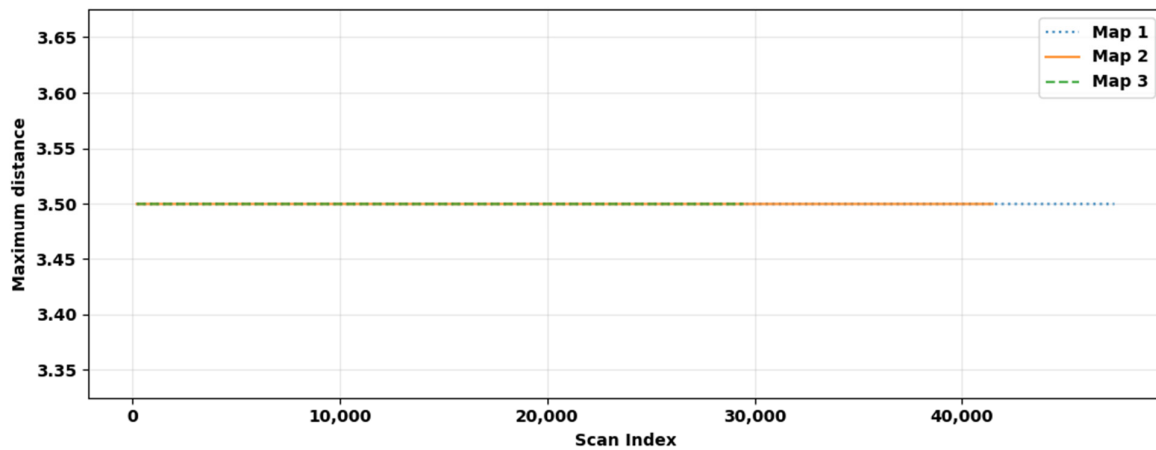


Figure A2. The maximum distance feature variation across scans on all three maps.

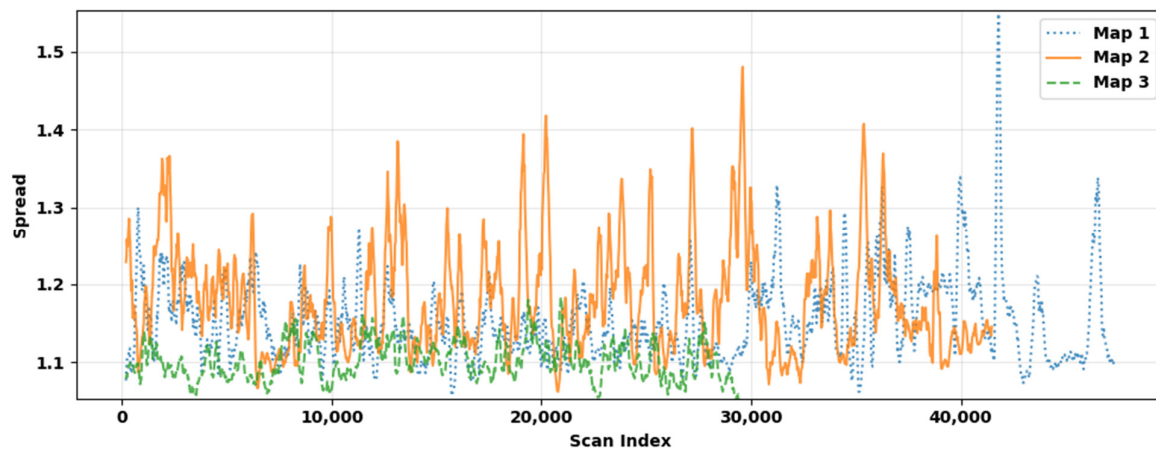


Figure A3. The r_{spread} feature variation across scans on all three maps.

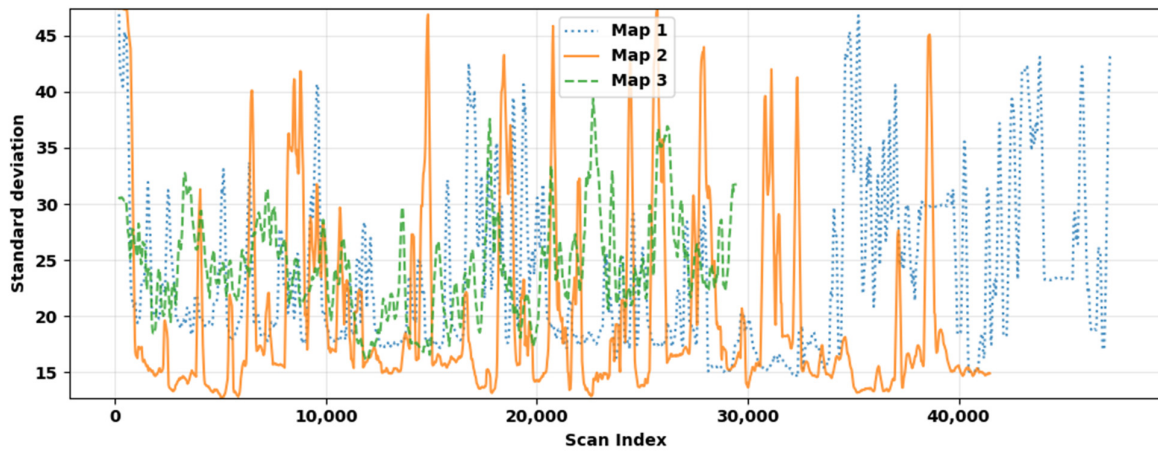


Figure A4. The standard deviation feature variation across scans on all three maps.

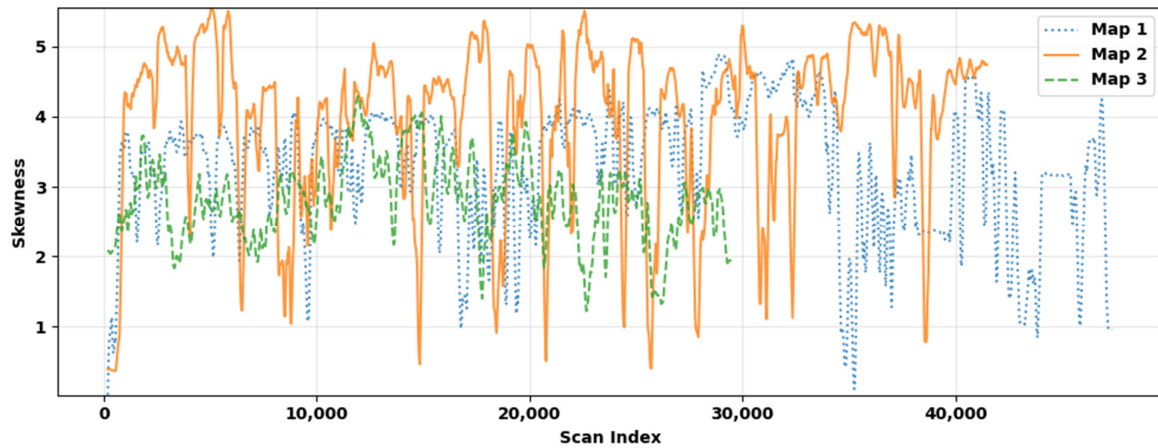


Figure A5. The skewness feature variation across scans on all three maps.

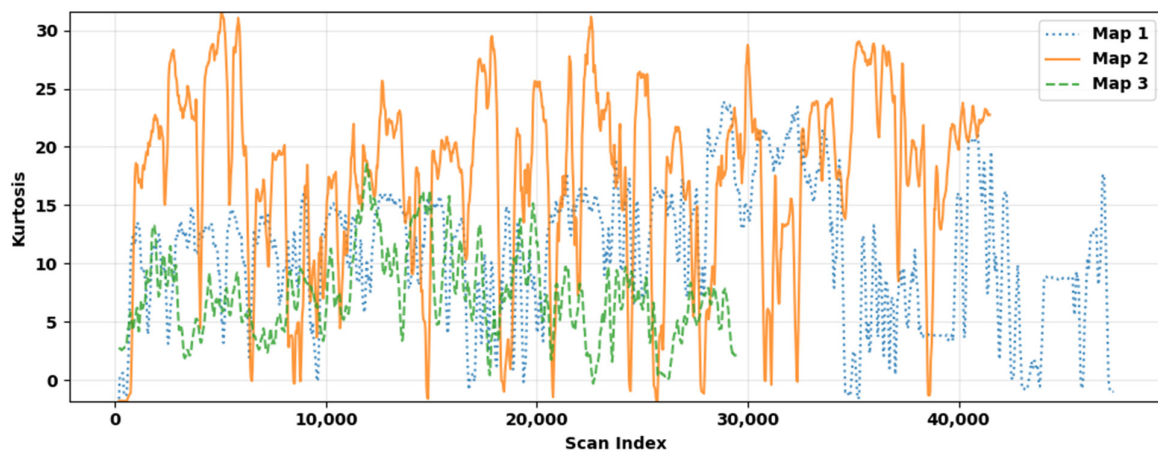


Figure A6. The kurtosis feature variation across scans on all three maps.

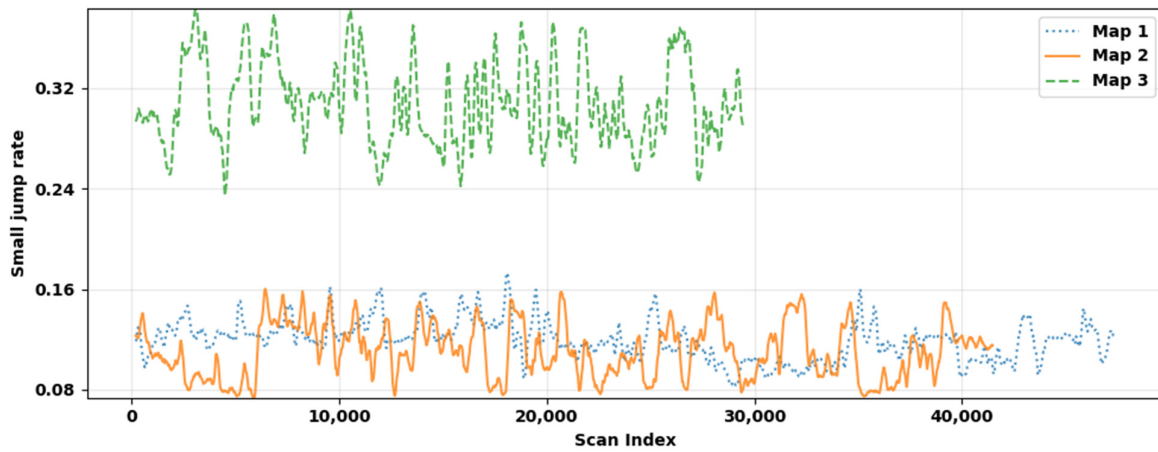


Figure A7. The small jump rate feature variation across scans on all three maps.

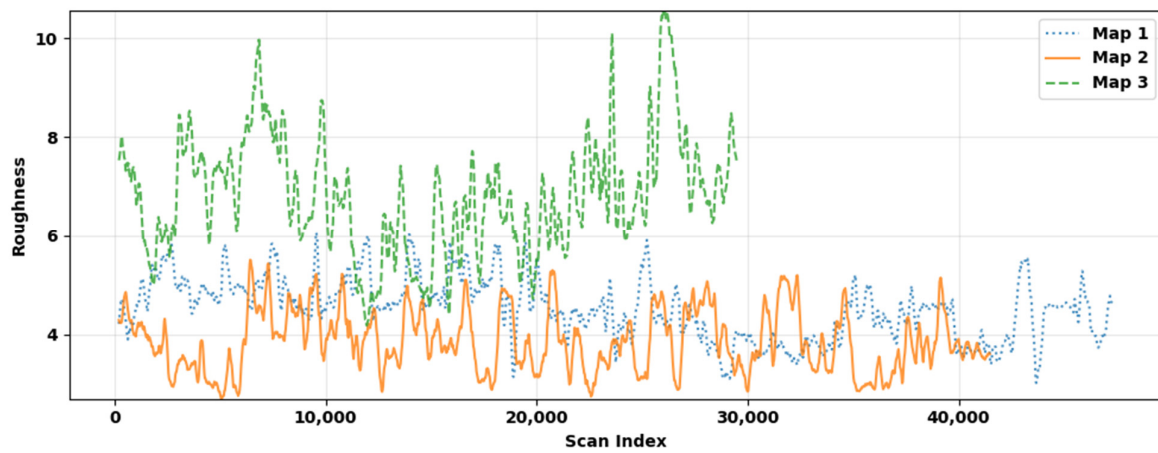


Figure A8. The roughness feature variation across scans on all three maps.

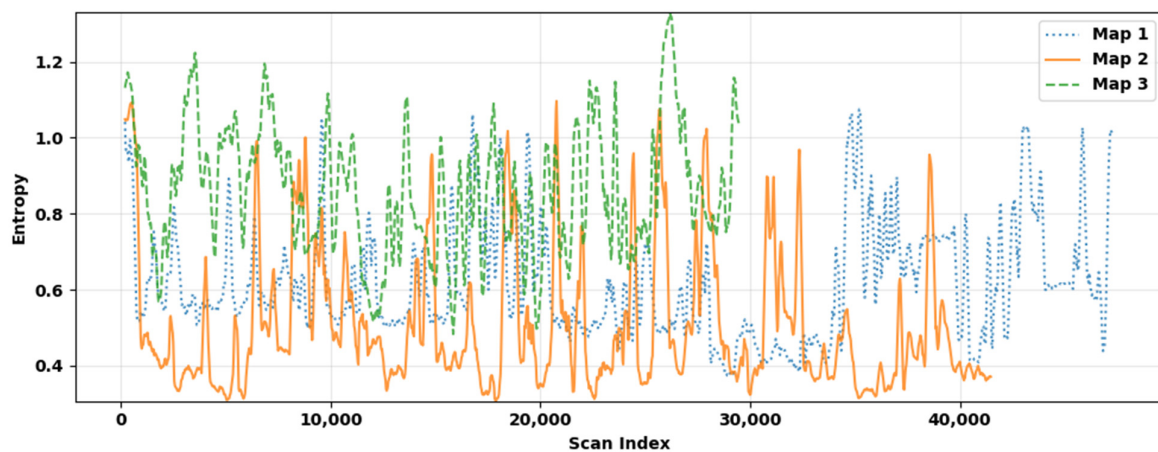


Figure A9. The entropy feature variation across scans on all three maps.

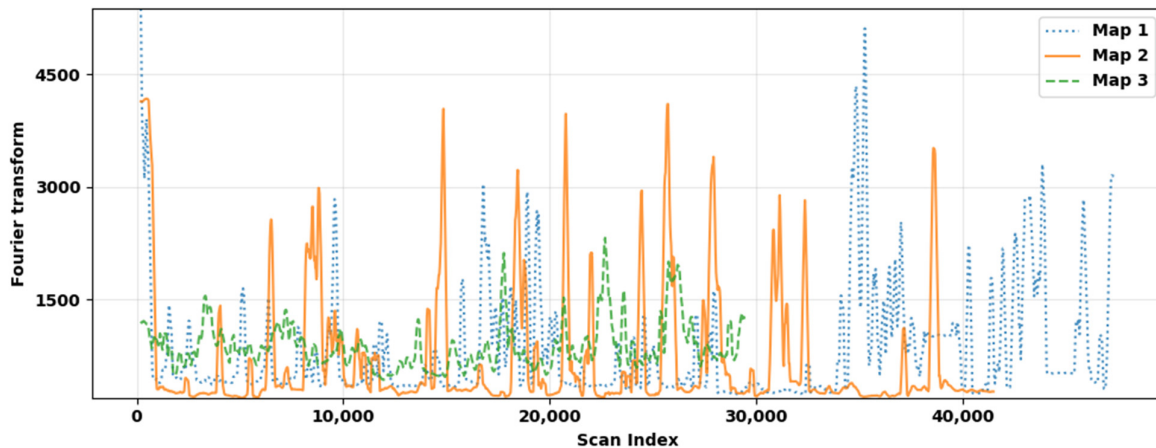


Figure A10. The Fourier transform feature variation across scans on all three maps.

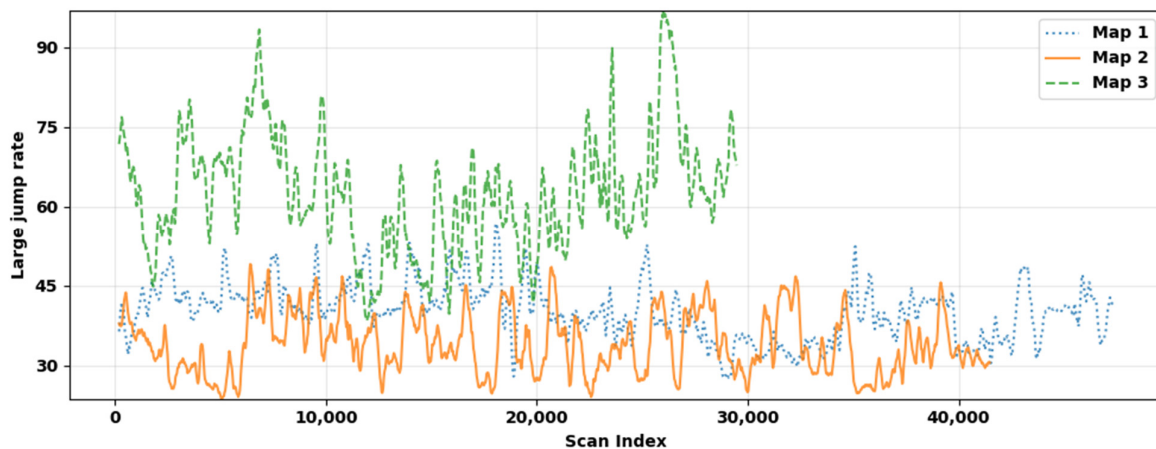


Figure A11. The large jump rate feature variation across scans on all three maps.

References

1. Cao, Z.; Liu, R.; Yuen, C.; Athukorala, A.; Kiat, B.K.; Mathanraj, M. Relative Localization of Mobile Robots with Multiple Ultra-WideBand Ranging Measurements. In *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*; IEEE: New York, NY, USA, 2021; pp. 5857–5863. <https://doi.org/10.1109/IROS51168.2021.9636017>.
2. Kuntho, J.; Karkar, A.G.; Al-Maadeed, S.; Al-Ali, A. Indoor positioning and wayfinding systems: A survey. *Hum.-Centric Comput. Inf. Sci.* **2020**, *10*, 18. <https://doi.org/10.1186/s13673-020-00222-0>.
3. Huang, J.; Junginger, S.; Liu, H.; Thuro, K. Indoor Positioning Systems of Mobile Robots: A Review. *Robotics* **2023**, *12*, 47. <https://doi.org/10.3390/robotics12020047>.
4. Phillips, J.M.; Monroe, C.; Shue, S.; Conrad, J.M. A Versatile, Low-Cost Robotic System for Educational Applications. In *Proceedings of the SoutheastCon 2025, Concord, NC, USA*; IEEE: New York, NY, USA, 2025; pp. 1142–1148. <https://doi.org/10.1109/SoutheastCon56624.2025.10971660>.
5. Altaf Khattak, S.B.; Fawad; Nasralla, M.M.; Esmail, M.A.; Mostafa, H.; Jia, M. WLAN RSS-Based Fingerprinting for Indoor Localization: A Machine Learning Inspired Bag-of-Features Approach. *Sensors* **2022**, *22*, 5236. <https://doi.org/10.3390/s22145236>.
6. Guo, Y.; Li, Y.; Ren, D.; Zhang, X.; Li, J.; Pu, L.; Ma, C.; Zhan, X.; Guo, J.; Wei, M.; et al. LiDAR-Net: A Real-Scanned 3D Point Cloud Dataset for Indoor Scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, WA, USA; IEEE: New York, NY, USA, 2024; pp. 21989–21999. <https://doi.org/10.1109/CVPR52733.2024.02076>.
7. Kim, J.; Chung, W. Robust Localization of Mobile Robots Considering Reliability of LiDAR Measurements. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, Australia; IEEE: New York, NY, USA, 2018; pp. 6491–6496. <https://doi.org/10.1109/ICRA.2018.8460648>.

8. Bukhori, I.; Ismail, Z.H.; Namerikawa, T. Detection strategy for kidnapped robot problem in landmark-based map Monte Carlo Localization. In *Proceedings of the 2015 IEEE International Symposium on Robotics and Intelligent Sensors (IRIS), Langkawi, Malaysia*; IEEE: New York, NY, USA, 2015; pp. 75–80. <https://doi.org/10.1109/IRIS.2015.7451590>.
9. Guizzo, E.; Ackerman, E. The TurtleBot3 Teacher [Resources_Hands On]. *IEEE Spectr.* **2017**, *54*, 19–20. <https://doi.org/10.1109/MSPEC.2017.8000281>.
10. Stefanoni, M.; Kovács, I.; Sarcevic, P.; Odry, Á. A Survey on the Main Techniques Adopted in Indoor and Outdoor Localization. *Electronics* **2025**, *14*, 2069. <https://doi.org/10.3390/electronics14102069>.
11. Quezada-Gaibor, D.; Torres-Sospedra, J.; Nurmi, J.; Koucheryavy, Y.; Huerta, J. Cloud Platforms for Context-Adaptive Positioning and Localisation in GNSS-Denied Scenarios—A Systematic Review. *Sensors* **2022**, *22*, 110. <https://doi.org/10.3390/s22010110>.
12. Gonçalves Ferreira, A.F.G.; Fernandes, D.M.A.; Catarino, A.P.; Monteiro, J.L. Localization and Positioning Systems for Emergency Responders: A Survey. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2836–2870. <https://doi.org/10.1109/COMST.2017.2703620>.
13. Moore, D.C.; Huang, A.S.; Walter, M.; Olson, E.; Fletcher, L.; Leonard, J.; Teller, S. Simultaneous local and global state estimation for robotic navigation. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan*; IEEE: New York, NY, USA, 2009; pp. 3794–3799. <https://doi.org/10.1109/ROBOT.2009.5152763>.
14. Macenski, S.; Martin, F.; White, R.; Clavero, J.G. The Marathon 2: A Navigation System. In *Proceedings of the 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA*; IEEE: New York, NY, USA, 2020; pp. 2718–2725. <https://doi.org/10.1109/IROS45743.2020.9341207>.
15. Lima, O.; Ventura, R. A case study on automatic parameter optimization of a mobile robot localization algorithm. In *Proceedings of the 2017 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Coimbra, Portugal*; IEEE: New York, NY, USA, 2017; pp. 43–48. <https://doi.org/10.1109/ICARSC.2017.7964050>.
16. Hess, W.; Kohler, D.; Rapp, H.; Andor, D. Real-time loop closure in 2D LIDAR SLAM. In *Proceedings of the 2016 IEEE International Conference on Robotics and Automation (ICRA), Stockholm, Sweden*; IEEE: New York, NY, USA, 2016; pp. 1271–1278. <https://doi.org/10.1109/ICRA.2016.7487258>.
17. Grisetti, G.; Stachniss, C.; Burgard, W. Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *Proceedings of the Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain*; IEEE: New York, NY, USA, 2005; pp. 2432–2437. <https://doi.org/10.1109/ROBOT.2005.1570477>.
18. Cao, Y.; Ni, J.; Huang, Y. DPCR-SLAM: A Dual-Point-Cloud-Registration SLAM Based on Line Features for Mapping an Indoor Mobile Robot. *Sensors* **2025**, *25*, 5561. <https://doi.org/10.3390/s25175561>.
19. Li, J.; Zhan, H.; Chen, B.M.; Reid, I.; Lee, G.H. Deep learning for 2D scan matching and loop closure. In *Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada*; IEEE: New York, NY, USA, 2017; pp. 763–768. <https://doi.org/10.1109/IROS.2017.8202236>.
20. Lassance, C.; Latif, Y.; Garg, R.; Gripon, V.; Reid, I. Improved Visual Localization via Graph Filtering. *J. Imaging* **2021**, *7*, 20. <https://doi.org/10.3390/jimaging7020020>.
21. Wang, W.; Wang, B.; Zhao, P.; Chen, C.; Clark, R.; Yang, B.; Markham, A.; Trigoni, N. PointLoc: Deep Pose Regressor for LiDAR Point Cloud Localization. *IEEE Sens. J.* **2022**, *22*, 959–968. <https://doi.org/10.1109/JSEN.2021.3128683>.
22. Kendall, A.; Grimes, M.; Cipolla, R. PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), Santiago, Chile*; IEEE: New York, NY, USA, 2015; pp. 2938–2946. <https://doi.org/10.1109/ICCV.2015.336>.
23. Isaacson, S.; Kung, P.-C.; Ramanagopal, M.; Vasudevan, R.; Skinner, K.A. LONER: LiDAR Only Neural Representations for Real-Time SLAM. *IEEE Robot. Autom. Lett.* **2023**, *8*, 8042–8049. <https://doi.org/10.1109/LRA.2023.3324521>.
24. Bresson, G.; Yu, L.; Joly, C.; Moutarde, F. Urban Localization with Street Views using a Convolutional Neural Network for End-to-End Camera Pose Regression. In *Proceedings of the 2019 IEEE Intelligent Vehicles Symposium (IV), Paris, France*; IEEE: New York, NY, USA, 2019; pp. 1199–1204. <https://doi.org/10.1109/IVS.2019.8813892>.
25. Li, Y.; Olson, E.B. Extracting general-purpose features from LIDAR data. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation, Anchorage, AK, USA*; IEEE: New York, NY, USA, 2010; pp. 1388–1393. <https://doi.org/10.1109/ROBOT.2010.5509690>.
26. Wang, Y.-T.; Peng, C.-C.; Ravankar, A.A.; Ravankar, A. A Single LiDAR-Based Feature Fusion Indoor Localization Algorithm. *Sensors* **2018**, *18*, 1294. <https://doi.org/10.3390/s18041294>.

27. Steinke, N.; Ritter, C.-N.; Goehring, D.; Rojas, R. Robust LiDAR Feature Localization for Autonomous Vehicles Using Geometric Fingerprinting on Open Datasets. *IEEE Robot. Autom. Lett.* **2021**, *6*, 2761–2767. <https://doi.org/10.1109/LRA.2021.3062354>.
28. Li, Z.; Tang, Z.; Kim, K.S.; Li, S.; Smith, J.S. EKF-Based Fusion of Wi-Fi/LiDAR/IMU for Indoor Localization and Navigation. In *Proceedings of the 2025 Thirteenth International Symposium on Computing and Networking Workshops (CANDARW)*, Yamagata, Japan; IEEE: New York, NY, USA, 2025; pp. 22–28. <https://doi.org/10.1109/CANDARW68385.2025.00012>.
29. Lu, Y.; Ma, H.; Smart, E.; Yu, H. Real-Time Performance-Focused Localization Techniques for Autonomous Vehicle: A Review. *IEEE Trans. Intell. Transp. Syst.* **2022**, *23*, 6082–6100. <https://doi.org/10.1109/TITS.2021.3077800>.
30. Toconas, B.G.D.; Alcoba, M.S. Enhanced 3D Mapping for Mobile Robots: Post-Processing of Dense Stereo-VISLAM PointClouds. In *Proceedings of the 2025 XXV Symposium of Image, Signal Processing, and Artificial Vision (STSIVA)*, Armenia, Colombia; IEEE: New York, NY, USA, 2025; pp. 1–6. <https://doi.org/10.1109/STSIVA66383.2025.11156649>.
31. Li, J.; Hassani, A.; Walton, S.; Shi, H. ConvMLP: Hierarchical Convolutional MLPs for Vision. In *Proceedings of the 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Vancouver, BC, Canada; IEEE: New York, NY, USA, 2023; pp. 6307–6316. <https://doi.org/10.1109/CVPRW59228.2023.00671>.
32. Simon, D. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*; Wiley-Interscience: Hoboken, NJ, USA, 2006.
33. Chen, L.; Hu, H.; McDonald-Maier, K. EKF Based Mobile Robot Localization. In *Proceedings of the 2012 Third International Conference on Emerging Security Technologies*, Lisbon, Portugal; IEEE: New York, NY, USA, 2012; pp. 149–154. <https://doi.org/10.1109/EST.2012.19>.
34. Barfoot, T.D. *State Estimation for Robotics*, 2nd ed.; Cambridge University Press: Cambridge, MA, USA, 2024.
35. Cordova-Cardenas, R.; Amor, D.; Gutiérrez, Á. Edge AI in Practice: A Survey and Deployment Framework for Neural Network on Embedded Systems. *Electronics* **2025**, *14*, 4877. <https://doi.org/10.3390/electronics14244877>.
36. Hussain Kalan, S.; Giin Lee, B.; Chung, W.-Y. CNN-LiDAR-SLAM: Multimodal Fusion with Object Detection for Indoor Localization. *IEEE Access* **2025**, *13*, 198864–198877. <https://doi.org/10.1109/ACCESS.2025.3635243>.
37. Zhou, Z.; Guo, C.; Pan, Y.; Li, X.; Jiang, W. A 2-D LiDAR-SLAM Algorithm for Indoor Similar Environment with Deep Visual Loop Closure. *IEEE Sens. J.* **2023**, *23*, 14650–14661. <https://doi.org/10.1109/JSEN.2023.3260104>.
38. Spampinato, G.; Bruna, A.; Guarneri, I.; Giacalone, D. Deep Learning Localization with 2D Range Scanner. In *Proceedings of the 2021 7th International Conference on Automation, Robotics and Applications (ICARA)*, Prague, Czech Republic; IEEE: New York, NY, USA, 2021; pp. 206–210. <https://doi.org/10.1109/ICARA51699.2021.9376424>.
39. Yin, H.; Wang, Y.; Ding, X.; Tang, L.; Huang, S.; Xiong, R. 3D LiDAR-Based Global Localization Using Siamese Neural Network. *IEEE Trans. Intell. Transp. Syst.* **2020**, *21*, 1380–1392. <https://doi.org/10.1109/TITS.2019.2905046>.
40. Ballesta, M.; Payá, L.; Cebollada, S.; Reinoso, O.; Murcia, F. A CNN Regression Approach to Mobile Robot Localization Using Omnidirectional Images. *Appl. Sci.* **2021**, *11*, 7521. <https://doi.org/10.3390/app11167521>.
41. Xu, Z.; Su, Z.; Dai, D. LSTM Network-Assisted Binocular Visual-Inertial Person Localization Method under a Moving Base. *Appl. Sci.* **2023**, *13*, 2705. <https://doi.org/10.3390/app13042705>.
42. Shi, H.; Yang, J.; Shi, J.; Zhu, L.; Wang, G. Vision-Sensor-Assisted Probabilistic Localization Method for Indoor Environment. *Sensors* **2022**, *22*, 7114. <https://doi.org/10.3390/s22197114>.
43. Ismail, Z.H.; Bukhori, I. Efficient Detection of Robot Kidnapping in Range Finder-Based Indoor Localization Using Quasi-Standardized 2D Dynamic Time Warping. *Appl. Sci.* **2021**, *11*, 1580. <https://doi.org/10.3390/app11041580>.
44. Wang, Y.; Zhang, W.; Li, F.; Shi, Y.; Nie, F.; Huang, Q. UAPF: A UWB Aided Particle Filter Localization for Scenarios with Few Features. *Sensors* **2020**, *20*, 6814. <https://doi.org/10.3390/s20236814>.
45. Nguyen, P.T.-T.; Yan, S.-W.; Liao, J.-F.; Kuo, C.-H. Autonomous Mobile Robot Navigation in Sparse LiDAR Feature Environments. *Appl. Sci.* **2021**, *11*, 5963. <https://doi.org/10.3390/app11135963>.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.