A Fast and Efficient Residual Learning Framework Driven by Approximate Nearest Neighbor Search for Large-Scale Fingerprint-Based Visible Light Positioning

Huy Q. Tran[1]*, Huu Lam Phan[2], Tan Nguyen Van[3]
Robotics and Mechatronics Research Group-Faculty of Engineering and Technology,
Nguyen Tat Thanh University, Ho Chi Minh City, Vietnam[1, 2]
School of Engineering Technology, Thu Dau Mot University, Thu Dau Mot City, Vietnam[3]

***Abstract*—Localization based on received signal strength using the k-Nearest Neighbor method is quite common in indoor localization systems. However, as the fingerprint dataset grows, finding the nearest neighbors becomes time-consuming. In this study, we use Approximate Nearest Neighbor (ApNN) methods to accelerate nearest-neighbor search in RSS-based localization. We further propose a residual learning framework driven by ApNN search, where ApNN provides coarse position estimates and the residual model compensates for the nonlinear relationship between RSS measurements and spatial coordinates. Simulation results show that, compared to the Brute k-Nearest Neighbor method, ApNN algorithms significantly reduce computation time. KD-Tree is the fastest algorithm, with an improvement of approximately 96% compared to kNN and WkNN. Other methods such as HNSW and Ball-Tree also achieved high performance with improvements of around 93–94%, while LSH improved by approximately 84.8%. Regarding positioning error, KD-Tree achieved the best positioning error after applying residual learning, with the highest RMSE reduction of approximately 22.5%. These results demonstrate that the proposed ApNN-based residual learning framework is an effective solution for large-scale received signal strength positioning systems.**

Keywords*—Localization; LED; approximate nearest neighbor; residual learning

I. Introduction

Currently, demand for robot positioning services in homes, schools, hospitals, offices, and factories is increasing, driving the rapid development of indoor positioning technologies. In this context, Visible Light Positioning (VLP) has emerged as a promising solution due to its ability to utilize existing LED lighting infrastructure and combine with Visible Light Communications (VLC). VLP not only achieves high accuracy but also offers lower deployment costs compared to other technologies [1].

Among indoor positioning algorithms, the fingerprint-based method combined with the k-Nearest Neighbors (kNN) algorithm is among the most popular techniques, with relatively good accuracy. The core principle of kNN in fingerprinting-based positioning is to compare the received signal strength (RSS) at the target location with that in a prebuilt database. The system then selects the k reference points with the highest similarity. Finally, it estimates the position by averaging their coordinates [2].

Although widely used, kNN still has several limitations. The main drawbacks of kNN are its slow execution time and high error rate, especially with large datasets [3]. To address these issues, many improvements have been proposed. One of the most common approaches is the development of weighted kNN (WkNN). The main idea is to assign larger weights to closer reference points rather than using a simple average for position estimation. Similarly, a WkNN model was proposed for VLP systems in [4]. This model estimates the receiver's current position based on the locations of k predefined neighboring points stored in a lookup table. The Euclidean distances between the actual position and reference points are weighted to improve accuracy. Simulation results show that the proposed model outperforms trilateration. The accuracy improves 36% and 50% under conditions with and without ambient light noise, respectively. To further improve accuracy, the authors in [5] proposed an Improved WkNN (IWkNN) algorithm for both distance and angle. Specifically, the k nearest reference points are first selected based on distance. Then, the angles between each reference point and the target are calculated. Finally, weights are determined by the inverse of the product of the distance and angle. Simulation results demonstrate that IWkNN outperforms standard WkNN in positioning accuracy.

Another approach was proposed in [6], where the authors observed that the Euclidean distance of RSS is not consistent with the actual spatial distance. They analyzed the relationship between RSS similarity and spatial distance and proposed a WkNN algorithm. This solution is based on Approximate Position Distance (APD). This method involves calculating a Weighted Euclidean Distance (WED) by balancing RSS differences and signal attenuation characteristics. Then, APD is determined based on both the spatial distance and the WED between the reference points.

In addition to weight improvements, many studies focus on determining the optimal value of k or combining kNN with other methods to enhance performance [3]. In the mentioned study, the WOKNN (Weighted Optimum K-Nearest Neighbors) algorithm was proposed for VLC positioning systems,

*Corresponding author.

combining OKNN and WKNN. To reduce computational time, the authors applied Maximum Received Signal Strength Recognition (MRR). OKNN automatically determines the optimal number of neighbors for each location, while WKNN reduces the error caused by averaging Euclidean distances. Experimental results show that the proposed method achieves an average positioning error of only 8 mm in a 1.2 m × 1.2 m space. Additionally, computation time is reduced by 42% - 52% compared to other kNN-based methods.

Another approach integrates kNN with Bayesian theory to improve accuracy [7]. A Bayesian K-Nearest Neighbor (BKNN) algorithm was proposed for RFID positioning systems. This method uses a Gaussian filter to remove abnormal RSS values, followed by BKNN for position estimation. Experimental results show that BKNN achieves an average error of about 15 cm, which is lower than that of Gaussian-based methods, LANDMARC, and improved kNN algorithms. Similarly, in [8], kNN was combined with Bayesian theory for VLP systems, achieving an average positioning error of 0.27 m. Additionally, Hoang et al. proposed the SRL-KNN (Soft Range Limited K-Nearest Neighbors) algorithm to improve indoor positioning accuracy. Real-world experiments show that SRL-KNN achieves an average positioning error of 0.66 m, with 80% of errors below 0.89 m. This approach achieves a 45% improvement over traditional kNN under the same conditions [9].

A modern trend in fingerprint-based positioning is the integration of kNN with advanced machine learning techniques, particularly deep learning. In [10], the authors proposed a hybrid method combining a Deep Neural Network (DNN) with an improved kNN algorithm for Wi-Fi RSSI-based fingerprinting. The improved kNN assigns weights to neighbors based on the number of matched access points, addressing the limitations of traditional kNN. The DNN is first used to classify RSSI fingerprint data. Then, candidate locations within each class are further refined using the improved kNN. Experiments conducted in a 13 m × 9 m space show that this hybrid approach is less dependent on access point density and more robust to indoor signal noise. It also outperforms algorithms such as random forest, SVM, and decision tree.

Nearest neighbor search is a fundamental operation in many domains such as databases, machine learning, and computer vision [11]. Although numerous Approximate Nearest Neighbor (ApNN) algorithms have been proposed, until recently, there has been a lack of comprehensive, systematic evaluation of their performance. These authors carried out a large-scale experimental evaluation. They tested 19 algorithms from different fields on 20 different datasets with different metrics and query workloads. The ApNN benchmarks tool provides a standard way to measure performance and quality. A key finding is that many different approaches can be used for the ApNN problem [12]. Among these methods, graph-based approaches have received significant attention [13]. In that study, the authors introduced EFANNA, which is considered a very fast ApNN algorithm. It combines tree structures and graph methods to improve efficiency. Another widely used hierarchical graph structure is HNSW (Hierarchical Navigable Small World) [14]. HNSW has been applied to local feature-based localization. The authors proposed using HNSW as an approximate nearest-neighbor search method to reduce processing time without sacrificing localization accuracy. The results showed no significant increase in localization error, while processing time was substantially improved. In addition, the inverted file (IVF) index approach has been widely adopted. In which partitions data into clusters and searches only within the closest clusters are applied. A related proposal is an adaptive ApNN algorithm [15] that evaluates query complexity to determine the sufficient number of clusters to search dynamically. This approach achieves up to 35% faster search speed while maintaining a high recall of 0.99 on datasets containing up to one billion vectors.

For binary data, the study in [16] proposed mapping binary descriptors to a low-dimensional real-valued space before applying the KD-tree and Hamming ranking. This method achieved an improvement in search accuracy from 67.7% to 83.7% when the search speed was 200 times faster than a linear scan. In large-scale localization, using randomized trees trained with supervised learning to index binary features improved localization speed by an order of magnitude compared to state-of-the-art methods [17].

From the studies reviewed above, it is evident that the application of ApNN in localization applications has not been thoroughly investigated and evaluated. In the field of indoor visible light positioning (VLP) using LED lighting, the use of ApNN methods as an approach to address the slow processing speed of kNN remains largely unexplored. Therefore, in this research, we focus on addressing the following issues. First, we construct an LED model within a large space of approximately 10 × 10 m², where a complete system comprising 81 LED fixtures is installed. We then collect the received signal strength (RSS) or illumination intensity data from each LED. Second, we apply various ApNN methods, including Ball-Tree, KD-Tree, Locality-Sensitive Hashing (LSH), and Hierarchical Navigable Small World (HNSW), to estimate the coordinates and evaluate the execution time of each method. Third, we propose a residual learning framework driven by approximate nearest neighbor (ApNN) search, where ApNN provides coarse positioning and the residual model effectively compensates for the nonlinear relationship between RSS measurements and spatial coordinates. Finally, we provide detailed performance analysis in terms of runtime, latency, and positioning error for different deployment scenarios. The results obtained from each ApNN approach will be compared against those of the traditional kNN method.

In the remainder of this study, we present the LED model as well as the configuration of the study area. Next, the ApNN-based positioning methods are described. Subsequently, we present the experimental results and perform a comparison with other methods. Finally, conclusions and directions for future research are provided.

II. Applied Localization Model

In this study, we developed an experimental model within an indoor environment with dimensions of 10 × 10 m². The lighting system consists of 81 LED sources arranged in a uniform grid, with an inter-LED spacing of 1 m. The LEDs located near the walls are positioned 1 m away from the boundaries. As depicted in Fig. 1, this configuration ensures a balanced distribution of optical power across different positions in the room.

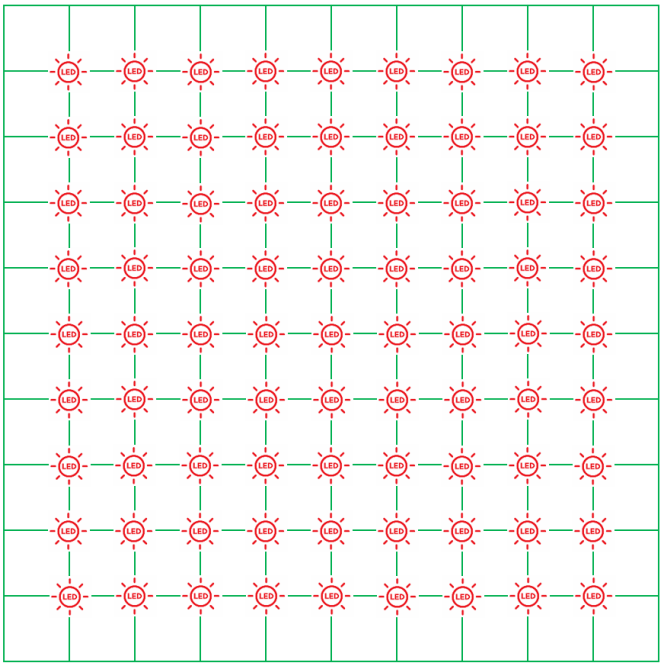

Fig. 1. Room and LED configuration.

The main parameters of the LED sources are defined as follows: the semi-angle at half power is 70°, corresponding to a Lambertian order:

$$m = -\frac{ln(2)}{\ln(\cos(70^0))} \quad (1)$$

Each LED has an optical power of 10 mW, resulting in a total transmitted optical power proportional to the number of LEDs. The photodetector area is 1×10^{-4} m², the optical filter gain is 1, and the refractive index of the concentrator is 1.5. The receiver field of view (FOV) is 60°, leading to a concentrator gain defined as:

$$G = \frac{n^2}{sin(FOV)} \quad (2)$$

where:

- n is the refractive index of the concentrator lens material.
- FOV is the field of view of the receiver.

Fig. 2 illustrates the block diagram of an optical wireless communication system using LEDs and a photodetector. The system consists of three main components: the transmitter, the channel, and the receiver.

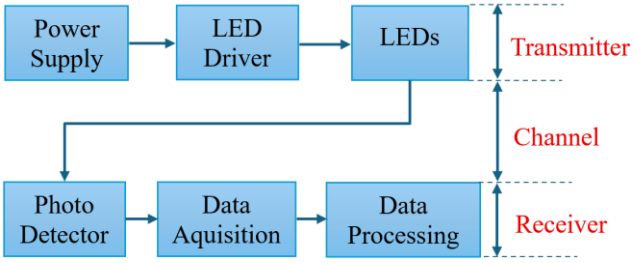

Fig. 2. VLP system diagram.

- At the transmitter side: a power supply provides energy to the LED driver circuit. This driver modulates the electrical signal to control the emission frequency of each LED. The operating frequencies range from 1 kHz to 81 kHz, with a spacing of 1 kHz between adjacent LEDs.
- Channel: the light emitted from the LEDs propagates through the indoor environment, including both line-of-sight (LOS) and reflected (non-line-of-sight, NLOS) components, carrying information from the transmitter to the receiver.
- At the receiver side: the optical signal is captured by a photodetector and converted into an electrical signal. This signal is then passed through a data acquisition unit for digitization and finally processed by the data processing block to recover the transmitted information.

The received optical power at a given point from the i^{th} LED is [3]:

$$P_{los,i} = P_{LED} H_i T_s G \quad (3)$$

The LOS channel gain is given by:

$$H_i = \frac{(m+1)A_P}{2\pi d_i^2} cos^m(\alpha_i)\cos(\beta_i) \quad (4)$$

where:

- P_{LED}: transmitted optical power of a single LED
- A_P: photodetector area
- d_i: distance between the i^{th} LED and the receiver
- α_i: irradiance angle
- β_i: incidence angle
- m: Lambertian order
- T_s: optical filter gain (≈ 1)
- G: concentrator gain

The total received power is the sum of contributions from all LEDs:

$$P_{los\_total} = \sum_{i=1}^{81} P_{los,i} \quad (5)$$

III. Residual Learning Framework Driven by Approximate Nearest Neighbor Search

ApNN search aims to efficiently identify data points that are close to a query point while trading off a small amount of accuracy for significantly reduced computational cost, especially in high-dimensional spaces. In this study, four representative ApNN methods are considered, including KD-tree, Ball-tree, Locality-Sensitive Hashing (LSH), and Hierarchical Navigable Small World (HNSW), each employing different strategies for efficient similarity search.

A. Ball - Tree

The Ball-tree is a hierarchical spatial data structure. It was introduced by Stephen M. Omohundro in 1989. The structure organizes data points into a binary tree. Each node represents a

hypersphere, also called a ball. These balls can contain other smaller balls inside them. The Ball-tree is commonly used for nearest neighbor search. It helps reduce the number of distance calculations. This makes the search process more efficient, especially in higher-dimensional data.

Each node in a Ball-tree is defined by a hypersphere with a center $c \in R^d$ and radius r, where the radius is the maximum distance from c to all points in the node [18]:

$$r = max_{x \in S} \|x - c\| \quad (6)$$

Here, s is the set of points contained in the node, and this formulation ensures that all points lie within the hypersphere centered at c. Based on this definition, the node can be interpreted as a hypersphere that includes all points satisfying the following condition [18]:

$$B(c, r) = \{x \in R^d | \|x - c\| \leq r\} \quad (7)$$

B. KD - Tree

In a KD-tree, the data space is recursively partitioned using axis-aligned hyperplanes. At each node, a splitting dimension k and a threshold value t are selected to divide the dataset into two subsets [19]:

$$S_{left} = \{x \in R^d | x_k \leq t\} \quad (8)$$

$$S_{right} = \{x \in R^d | x_k > t\} \quad (9)$$

Here, x_k denotes the value of a point x along the k^{th} dimension. This partitioning process is applied recursively, forming a binary tree where each node represents a hyper-rectangular region of the space.

C. Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) maps data points into the same bucket with high probability if they are close in the feature space [20]. A hash function $h(\cdot)$ satisfies:

$$P[h(x) = h(y)] = \begin{cases} high, if \ \|x - y\| \leq R \\ low, \ if \ \|x - y\| > cR \end{cases} \quad (10)$$

where, R is a distance threshold and $c > 1$ is the approximation factor. This ensures that nearby points are more likely to collide in the same hash bucket.

D. Hierarchical Navigable Small World

Hierarchical Navigable Small World (HNSW) performs approximate nearest neighbor search by navigating a multi-layer graph structure, aiming to minimize the distance between a query point q and data points [21]:

$$x^* = argmin_{x \in S} \|q - x\| \quad (11)$$

The search process follows a greedy strategy, starting from upper layers and progressively refining the result at lower layers.

Table I presents the main specifications of the five nearest neighbor search methods. For kNN, the number of nearest points is set to 5 for all algorithms to ensure a fair comparison, as this value also provided the best accuracy during the error evaluation process. This value helps to determine the number of neighbors that are used for position prediction. The Euclidean metric is used to calculate spatial distances. The KD-Tree and Ball-Tree methods both use a leaf size of 30, which controls the granularity of tree partitioning. For LSH, the number of projection dimensions is reduced to 8. In HNSW, three parameters are crucial: the maximum number of connections per node is 16, the dynamic list size during construction is 200, and the search breadth during querying is 50.

TABLE I. MAIN TECHNICAL SPECIFICATIONS

Method	Parameters	Values
kNN Brute	Number of nearest neighbors	5
	Euclidean distance metric	Minkowski (p=2)
KD-Tree	Number of nearest neighbors	5
	Maximum points per leaf node	30
	Euclidean distance metric	Euclidean
Ball_Tree	Number of nearest neighbors	5
	Maximum points per leaf node	30
	Euclidean distance metric	Euclidean
LSH	Target projection dimensions	8
	Random number generator state	None
	Number of nearest neighbors	5
	Linear search in projected space	Brute
HNSW	Euclidean distance space	l2
	Input data dimensionality	16
	Maximum elements in index	340
	Dynamic list size during construction	200
	Maximum connections per node	16
	Dynamic list size during query	50
	Number of nearest neighbors	5

After obtaining the indices of the 5 nearest neighbors in the training set, all five methods apply the same averaging formula to compute the predicted coordinates:

$$\hat{Y}_{ApNN}^{(i)} = \frac{1}{k} \sum_{j \in N_K(X_{ApNN)}^{(i)})} Y_{train}^{(j)} \quad (12)$$

where:

- k: the number of nearest neighbors
- $N_K(X_{ApNN)}^{(i)})$: the set of indices of the k nearest neighbors to $X_{ApNN)}^{(i)}$ in the training set
- k: the index of a training point belonging to N_K
- $Y_{train}^{(j)}$: the normalized coordinates (XR, YR) of the j^{th} training point

Algorithm 1: ApNN-RL Algorithm

Input:

$X \in R^{2500 \ x \ 81}$: RSS fingerprint dataset

$Y \in R^{2500 \ x \ 2}$: Ground-truth positions on a 50×50 grid

K: Number of nearest neighbors

ApNN(·):Approximate nearest neighbor search method
Output:
$\hat{Y}$: Estimated positions
Evaluation metrics (RMSE, total time, latency)

INITIALIZATION

Normalize X and Y
Split dataset

OFFLINE PHASE

| For (each training sample) do
    | Find K nearest neighbors using ApNN
    | Estimate coarse position $\hat{Y}_{ApNN}^{(i)}$
| End
Compute residual $r^{(i)}$ for each sample
Construct training input
Initialize residual network $g(\cdot)$
While (epoch < MaxEpoch) do
  | For (each training sample) do
    | Predict residual $r_{pred}^{(i)}$ using $g(\cdot)$
    | Compute loss between $r^{(i)}$ and $r_{pred}^{(i)}$
    | Update network parameters θ
  | End
End

ONLINE PHASE

| For (each test sample) do
    | Retrieve K using ApNN
    | Estimate coarse position
    | Construct input (RSS + coarse position)
    | If (model is trained) then
        | Predict residual error
        | Refine position $\hat{y}^{(i)}$
    | End
| End

EVALUATION

Compute:
    RMSE between predicted and true positions
    ApNN search time
    Average time per sample
Return final estimated positions and evaluation metrics

However, because ApNN is approximate and the relationship between RSS measurements and physical locations is nonlinear, this initial estimate may still have noticeable errors. To fix this, we use a Residual Learning approach to improve the prediction by learning the difference between the estimated position and the true coordinates. The detailed workflow of the proposed ApNN-based residual learning (ApNN-RL) method is illustrated in Algorithm 1. The proposed ApNN-based Residual Learning (ApNN-RL) model combines the fast ApNN method with a learning-based refinement strategy to achieve both fast computation cost and improved positioning accuracy. In the offline phase, an ApNN method is employed to efficiently retrieve the k nearest points for each sample. Then, coarse positions are estimated through neighbor averaging. Based on this initial estimate, the residual error between the predicted and true positions is computed. We then apply a residual neural network to learn this error using both the original RSS data and the coarse estimate as input. In the online phase, the same ApNN search is applied to quickly generate a coarse position for each query, which is then refined by the trained residual model to produce the final estimate.

TABLE II. SPECIFICATIONS OF RESIDUAL LEARNING NETWORK

Parameter		Value
Input dimension		81
Output dimension		2
Architecture		Fully-connected NN
Layer 1	Linear	Input_dimension (128)
	Activation	ReLU
	Dropout	0.2
Layer 2	Linear	128 - 64
	Activation	ReLU
Layer 3	Linear	64 - 2
Training	Loss function	MSELoss
	Optimizer	Adam
	Learning rate	0.001
	Epochs	2000

The residual is defined as:

$$r^{(i)} = y^{(i)} - \hat{y}_{ApNN}^{(i)} \tag{13}$$

A neural network $g(\cdot)$ is then trained to learn this residual mapping using both the original input and the coarse estimate:

$$g\left(x^{i}, \hat{y}_{ApNN}^{(i)}\right) \approx r^{(i)} \tag{14}$$

The final refined position is obtained by combining the ApNN estimate with the predicted residual:

$$\hat{y}^{(i)} = \hat{y}_{ApNN}^{(i)} + g\left(x^{i}, \hat{y}_{ApNN}^{(i)}\right) \tag{15}$$

Then, Root Mean Square Error (RMSE) is used to evaluate the prediction accuracy of each method by measuring the difference between the actual and predicted positions after inverse transformation, with the formula defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\|y^{(i)} - \hat{y}^{(i)}\|^2} \tag{16}$$

The total query time is defined as:

$$T_{total} = \sum_{i=1}^{n} t_i \tag{17}$$

where, n is the number of test samples and t_i is the query time for the i^{th} sample.

The average latency per query is given by:

$$T_{Latency} = \frac{T_{total}}{n} \tag{18}$$

The main parameters of the residual learning model are shown in Table II. They include the network structure, training settings, and input–output configuration. The residual learning network is implemented as a fully connected neural network with two hidden layers (128 and 64 neurons) using ReLU activation and dropout regularization. It is trained with the Adam optimizer and MSE loss to learn the residual error between the coarse ANN-based estimate and the ground-truth position, enabling refined localization accuracy.

IV. Results and Discussion

As shown in Fig. 3, all methods improved accuracy after applying the proposed ApNN-RL algorithm. In all cases, the KD-Tree method had the best error reduction at 22.5%. The kNN and WkNN methods also showed significant improvement, with RMSE decreasing 20% and 18.6%, respectively. Similarly, Ball-Tree and HNSW also decreased from 5.99 cm to 4.72 cm and 4.84 cm, respectively. Among these methods, only LSH showed the least improvement in RMSE at 2.3%.

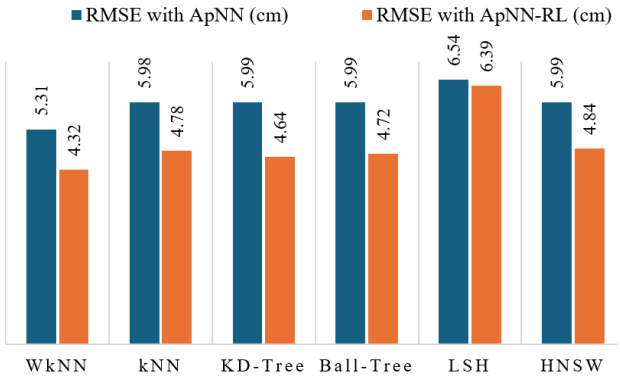

Fig. 3. Positioning error comparison.

In contrast, the LSH method exhibits only a marginal improvement, with only 2.3%. For HNSW, although the improvement is not as great as that of kNN, KD-Tree, or Ball-Tree, a clear reduction is still observed, from 5.99 cm to 4.84 cm. These results indicate that HNSW still benefits from ApNN-RL while maintaining its advantage in computational efficiency.

In Fig. 4, we evaluate the execution speed of each method. The reported Time (s) shows the total query time for all test samples. The ms/sample value shows the average time for each localization sample. The results show clear differences in speed between the methods. The traditional kNN method has the highest query time, reaching 0.1775 s. Its average time per sample is 0.3549 ms. The WkNN algorithm also shows a similar computational performance. In contrast, methods like KD-Tree and Ball-Tree run much faster. KD-Tree reduces the time to 0.0070 s, while Ball-Tree takes 0.0125 s. These results are equivalent to 0.0140–0.0251 ms per sample, which is approximately 14–25 times faster than kNN. In which, the HNSW method achieves a query time of 0.0106 s and an average latency of 0.0211 ms per sample. This corresponds to an improvement of around 17 times compared to kNN. Meanwhile, LSH exhibits an intermediate query time of 0.0270, which is faster than kNN but still slower than KD-Tree, Ball-Tree, and HNSW. Overall, regardless of the ApNN method used, the computational cost of each solution was significantly reduced. This is particularly important in positioning systems that use a large number of fingerprints.

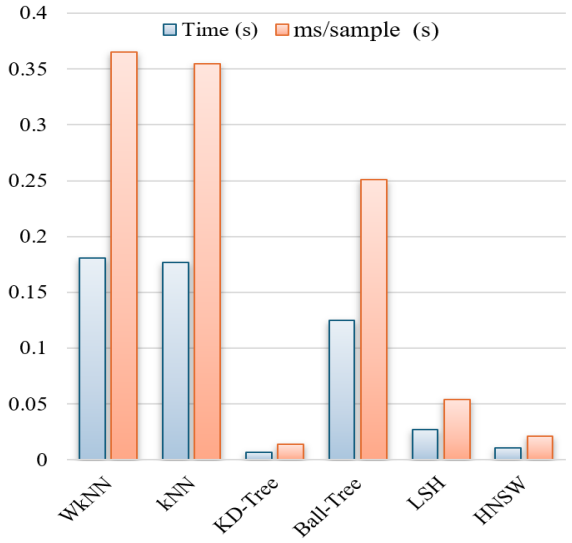

Fig. 4. Runtime performance comparison.

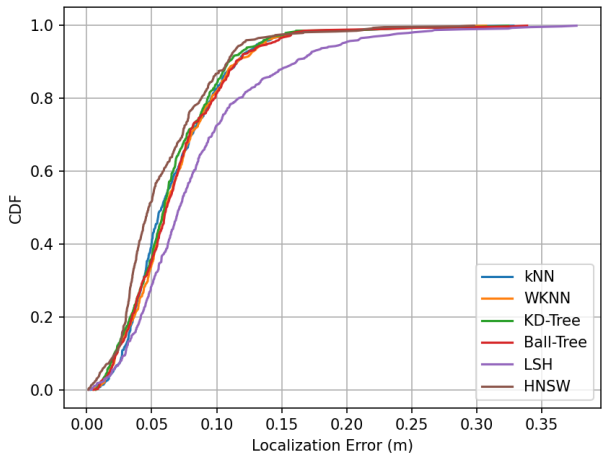

Fig. 5. CDF comparison.

To provide a comprehensive evaluation of localization accuracy, the cumulative distribution function (CDF) is used to illustrate the distribution of positioning errors across all samples. As shown in Fig. 5, all methods show a rapid increase in CDF values at low error ranges. Within the range of 0.05-0.1 m, the curves representing the WkNN, kNN, KD-Tree, Ball-Tree, and HNSW methods almost overlap. This demonstrates that these methods achieve comparable accuracy in terms of error distribution. Conversely, the LSH curve is significantly right-skewed. Clearly, LSH has lower accuracy and a less concentrated error distribution. This is also illustrated in Fig. 3. LSH produced lower accuracy because the Gaussian random projection compresses RSS fingerprints into a lower-dimensional space, causing the loss of important neighborhood similarity information. Therefore, the retrieved neighbors are less representative than those obtained by exact search methods such as WkNN, kNN, KD-Tree, and HNSW.

The heatmap of KD-Tree illustrates the spatial distribution of localization errors across the whole area as shown in Fig. 6. Most regions exhibit low error values, as indicated by the darker color intensity. Higher error values appear at several isolated locations and near certain boundary areas. These variations suggest that the localization performance is not completely uniform across space, with some positions showing larger deviations than others. Overall, the error distribution is dispersed rather than concentrated in a single region, indicating no dominant area with consistently high error.

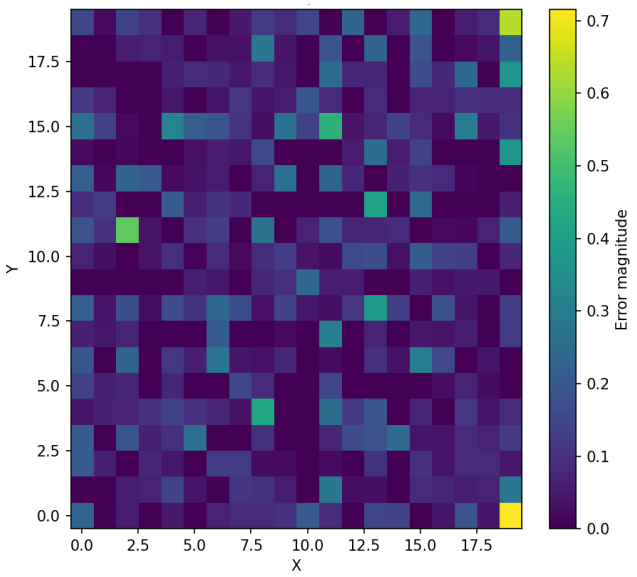

Fig. 6. Heat map of KD-tree.

V. Conclusion

In this study, we combined ApNN methods with residual learning to improve both localization accuracy and execution time. Regarding accuracy, the KD-Tree method combined with residual learning achieved the highest accuracy not only in localization error but also in execution time. Simulation results showed that this method achieved the lowest RMSE of 4.64 cm. In terms of runtime, KD-Tree also yielded low latency of 0.014 ms per sample. After KD-Tree, HNSW provided slightly lower computational cost but slightly lower accuracy. Overall, the integration of residual learning and ApNN improved localization accuracy and especially computation time across most methods.

In the future, we can further improve localization quality by optimizing the residual learning network architecture and adaptively tuning the ApNN parameters. Furthermore, extending the method to more complex environments or larger datasets is also a promising approach.

Acknowledgment

We would like to thank Nguyen Tat Thanh University, Ho Chi Minh City, Vietnam, for the support in this study.

References

[1] Z. Zhu, Y. Yang, M. Chen, C. Guo, J. Cheng, and S. Cui, "A survey on indoor visible light positioning systems: Fundamentals, applications, and challenges," IEEE Commun. Surveys Tuts., vol. 27, no. 3, pp. 1656–1686, 2025.

[2] J. Torres-Sospedra et al., "Let's talk about k-NN for indoor positioning: Myths and facts in RF-based fingerprinting," in Proc. IPIN Conf., pp. 1–6, 2023.

[3] H. Q. Tran and C. Ha, "High precision weighted optimum k-nearest neighbors algorithm for indoor visible light positioning applications," IEEE Access, vol. 8, pp. 114597–114607, 2020.

[4] M. T. Van, N. Van Tuan, T. T. Son, H. Le-Minh, and A. Burton, "Weighted k-nearest neighbour model for indoor VLC positioning," IET Commun., vol. 11, pp. 864–871, 2017.

[5] R. Liu, Z. Liang, Z. Wang, and M. Song, "Accuracy enhancement of indoor visible light positioning using improved WKNN algorithm," in Proc. ICICN Conf., pp. 109–114, 2023.

[6] B. Wang et al., "A novel weighted KNN algorithm based on RSS similarity and position distance for Wi-Fi fingerprint positioning," IEEE Access, vol. 8, pp. 30591–30602, 2020.

[7] H. Xu, Y. Ding, P. Li, R. Wang, and Y. Li, "An RFID indoor positioning algorithm based on Bayesian probability and K-nearest neighbor," Sensors, vol. 17, p. 1806, 2017.

[8] G. Chen, S. Jian-Hua, K. Wei, and Z. Chun-Yan, "A visible light indoor positioning algorithm based on fingerprint," in Proc. ICNISC Conf., pp. 71–77, 2018.

[9] M. T. Hoang et al., "A soft range limited K-nearest neighbors algorithm for indoor localization enhancement," IEEE Sensors J., vol. 18, no. 24, pp. 10208–10216, 2018.

[10] P. Dai, Y. Yang, M. Wang, and R. Yan, "Combination of DNN and improved KNN for indoor location fingerprinting," Wireless Commun. Mobile Comput., Art. no. 4283857, 2019.

[11] W. Li et al., "Approximate nearest neighbor search on high dimensional data—Experiments, analyses, and improvement," IEEE Trans. Knowl. Data Eng., vol. 32, no. 8, pp. 1475–1488, 2020.

[12] M. Aumüller, E. Bernhardsson, and A. Faithfull, "ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," Inf. Syst., vol. 87, p. 101374, 2020.

[13] C. Fu and D. Cai, "EFANNA: An extremely fast approximate nearest neighbor search algorithm based on kNN graph," arXiv preprint arXiv:1609.07228, 2016.

[14] R. Kotroczó, D. Varga, J. M. Szalai-Gindl, B. Formanek, and P. Vaderna, "Localization with approximate nearest neighbour search," IET Image Process., vol. 20, no. 1, p. e70242, 2026.

[15] V. Kazakovtsev et al., "Fast adaptive approximate nearest neighbor search with cluster-shaped indices," Big Data Cogn. Comput., vol. 9, p. 254, 2025.

[16] B. Fan et al., "Efficient nearest neighbor search in high dimensional hamming space," Pattern Recognit., vol. 99, p. 107082, 2020.

[17] Y. Feng, L. Fan, and Y. Wu, "Fast localization in large-scale environments using supervised indexing of binary features," IEEE Trans. Image Process., vol. 25, no. 1, pp. 343–358, 2016.

[18] S. M. Omohundro, Five Balltree Construction Algorithms. ICSI Tech. Rep. TR-89-063, 1989.

[19] J. L. Bentley, "Multidimensional binary search trees used for associative searching," Commun. ACM, vol. 18, no. 9, pp. 509–517, 1975.

[20] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in Proc. VLDB Conf., pp. 518–529, 1999.

[21] Y. Malkov and D. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," IEEE Trans. Pattern Anal. Mach. Intell., vol. 42, no. 4, pp. 824–836, 2018.