TSSA Journal of Tomography System & Sensors Application Vol. 9, Issue 1, 2026
www.tssa.com.my e-ISSN: 636-9133

A Cost-Effective Indoor Positioning System using Wi-Fi with Machine Learning on a PaaS Backend

Muhammad Afiq Kamaruzaman[1], Rozeha A. Rashid[1]*

[1]Telecommunication Software and Systems (TeSS) Research Group, Department of Communication Engineering, School of Electrical Engineering, Faculty of Engineering, Universiti Teknologi Malaysia, 81310, Johor Bahru, Johor, Malaysia

Corresponding author* email: rozeha@utm.my
Available online 01 March 2026

ABSTRACT

Indoor positioning systems (IPS) have become crucial in a variety of fields, including smart buildings, healthcare, and retail. However, many existing systems have high deployment costs, low energy efficiency, and restricted scalability. This work describes an inexpensive IPS that uses Wi-Fi received signal strength indicator (RSSI) data and machine learning to determine interior positions in real time. The system collects RSSI fingerprints using a lightweight Flutter-based mobile app (implemented with Flutter for cross-platform compatibility) before offloading computation-intensive activities to a Platform-as-a-Service (PaaS) backend developed with FastAPI and MongoDB. K-Nearest Neighbours (KNN) is the principal localisation algorithm, chosen because of its simplicity, versatility, and competitive accuracy. Experiments conducted indoors show encouraging results in terms of localisation accuracy, energy consumption, and cost scalability. This method demonstrates that accurate indoor location is possible without specialised hardware or expensive infrastructure, making it appropriate for large-scale, low-cost deployments.

Keywords: Indoor Positioning System, Machine Learning, Platform-as-a-Service (PaaS)

1. Introduction

Indoor localization has become increasingly important for applications ranging from smart buildings and retail to emergency response, as global navigation satellite systems (GNSS) are largely ineffective inside structures [1]. Among available technologies, Wi-Fi fingerprinting strikes a balance between cost and performance: it leverages ubiquitous wireless infrastructure and yields moderate-to-high accuracy (typically 1–5 m) without requiring specialized hardware [1], [2]. Wi-Fi-based indoor positioning systems (IPS) use received signal strength indicator (RSSI) fingerprints to match a device's signal profile against a pre-collected radio map. This scene-analysis approach is more accurate than simple range-based methods, but it traditionally demands laborious site surveys and can incur high on-device computation.

To deploy Wi-Fi IPS on consumer smartphones, three key objectives must be met: low cost, energy efficiency, and high accuracy. Cost-effectiveness demands using existing infrastructure (APs and phones) and free or minimal hosting resources [3], [4]. Energy efficiency requires minimizing smartphone processing and radio use, since continuous Wi-Fi scanning can drastically shorten battery life [5]. High accuracy entails robust algorithms that compensate for noise and environment dynamics. In this work, we propose a novel Wi-Fi fingerprinting IPS that meets all three goals: it uses a lightweight Flutter-based mobile client (implemented with Flutter for cross-platform compatibility) for RSSI data collection and offloads all heavy computation to a cloud-based backend implemented on a Platform-as-a-Service (PaaS) with FastAPI and MongoDB. The system employs the k-Nearest Neighbors (KNN) algorithm as its primary localization model. KNN's simplicity and modest computation make it well-suited for mobile-offload scenarios [3], and it has been shown to balance accuracy and complexity effectively.

We describe the motivation and limitations of existing approaches, present the architecture of our system, detail the methodology of data collection and localization, and report experimental results. The results demonstrate that our system achieves competitive positioning accuracy (on the order of 1–3 m mean error) while keeping power and monetary costs low. We also analyze latency and energy use to show that offloading computation to the cloud significantly reduces

mobile resource consumption without sacrificing performance. A critical discussion of the findings highlights trade-offs and suggests directions for future work.

2. Related Work

Numerous reviews and studies have examined indoor localization technologies and algorithms [2], [6]. Radio-frequency (RF) methods dominate due to ease of deployment; among these, Wi-Fi is especially attractive because of existing infrastructure and widespread smartphone support. In contrast, ultra-wideband (UWB) offers sub-meter accuracy at high cost, and Bluetooth or RFID provide only limited ranges. Wi-Fi RSSI fingerprinting provides a practical compromise: it can achieve meter-level accuracy without additional hardware, at the expense of an offline training phase to build a signal map [1], [2].

Classic IPS algorithms include trilateration (distance-based) and probabilistic methods, but scene-based fingerprinting often yields higher accuracy in multipath-rich indoor environments [2]. Prior systems have applied machine learning to Wi-Fi fingerprints: for example, weighted KNN, support vector machines, and deep networks have been used to improve positioning accuracy. However, more complex models tend to require more computation and data [3], [7]. Some recent work introduces graph neural networks or PCA-ELM algorithms to denoise RSSI data and increase accuracy [7], [8], but these typically run on powerful servers or robotics platforms.

Mobile-centric studies emphasize lightweight methods. For instance, Yang *et al.* perform a systematic design and conclude that KNN and weighted KNN offer the best trade-off between accuracy and complexity. They advocate a cloud-based architecture to manage the fingerprint database, improving scalability and reducing on-device work [9]. Chen *et al.* propose a "cost-effective" fingerprinting IPS that uses bilinear interpolation to fill sparse scans and a particle filter for tracking, achieving sub-meter accuracy with only 10% of survey points [4]. Hossain *et al.* build a smartphone app with hybrid trilateration/least-squares algorithms, finding linear least squares improves accuracy over basic trilateration [6]. Karra *et al.* develop a low-cost navigation system using a fingerprint map and KNN on Android; they report mean error ~0.86 m using standard Wi-Fi routers and a small grid of reference nodes [10]. Others embed Wi-Fi scanning in OpenWRT APs or low-cost microcontrollers for beaconing [11], [12]. These systems demonstrate that with careful design, inexpensive hardware can localize users within a few meters.

However, a common limitation of smartphone IPS is energy drain. Wi-Fi scanning is very power-hungry – one study showed that continuous Wi-Fi scans can reduce a phone's battery life from ~300 h (idle) to under 6 h [5]. Thus, many approaches attempt to minimize scans or offload processing. Krishnan *et al.* survey context-based offloading and note that thick-client (all-on-device) is very energy-intensive, whereas a thin-client (server-based) system saves battery at the cost of increased network delay [6]. Hybrid "medium-client" architectures (e.g. using local smart hotspots) can minimize energy, but require additional infrastructure. In contrast, our system adopts a pure thin-client model: the phone only collects RSSI and sends it to the cloud, leaving all heavy computation to server nodes. This aligns with recommendations to offload computation to reduce mobile energy consumption.

In summary, prior work shows that Wi-Fi fingerprinting can provide high accuracy using smartphones and existing infrastructure[2], [10]. Many systems use KNN-like methods for simplicity, and some leverage cloud computing to handle data and processing [3], [10]. Our approach builds on these ideas by explicitly optimizing for energy efficiency and cost. We use a lightweight Flutter-based mobile client and a PaaS backend with FastAPI and MongoDB, which reduces software maintenance effort and avoids extra hardware. We also use a purely KNN-based localization (no interpolation or Bayesian filtering) to maintain algorithmic simplicity, and we measure performance trade-offs in the context of real smartphone usage.

3. System Architecture

The proposed IPS consists of two main components: a mobile client app and a cloud-hosted backend service. The Flutter-based mobile app runs on the user's device and handles all radio measurements and interface functions. The backend is implemented as a FastAPI web service connected to a MongoDB database, deployed on a cloud PaaS (e.g. AWS/Azure/GCP or a similar managed platform).

3.1 Mobile Client (Flutter)

The mobile client performs RSSI scanning and communicates with the server. On startup, the app collects RSSI measurements from nearby Wi-Fi access points by scanning each channel and records the received signal strengths. The

readings are packaged into a fingerprint vector, which the app then sends via a lightweight HTTP (REST) request to the backend server. The user interface displays the estimated location returned by the server. Because all heavy computation is offloaded, the app only needs to perform scans and simple JSON I/O, keeping its CPU and memory footprint very small.

We used Flutter for cross-platform deployment (supporting both Android and iOS devices) to ensure broad device support; however, the method could extend to any platform with Wi-Fi. In practice, any modern smartphone running the Flutter client can participate, making the solution widely applicable.

3.2 Backend Service (FastAPI + MongoDB)

The server is built using FastAPI, a modern high-performance Python web framework. FastAPI handles incoming HTTP requests from clients and dispatches them to the localization logic. The fingerprint database and reference points are stored in MongoDB, a scalable NoSQL database. This architecture leverages cloud auto-scaling and managed database services to minimize operational overhead [3]. FastAPI makes it easy to deploy on a PaaS and supports asynchronous processing for low-latency responses.

The PaaS backend maintains a global radio map of reference fingerprints (RSSI vectors) for each known location in the facility. During setup, an administrator or robot can survey the area and populate the MongoDB with Wi-Fi fingerprints and their associated coordinates. Once the database is initialized, the system is ready for localization. Each incoming RSSI vector is compared to the stored fingerprints using a KNN search (see Methodology). The server computes the nearest neighbors and infers the user's location, then returns the result to the app.

This client–server design minimizes device energy use by leveraging the thin-client paradigm. As noted in [13], shifting computation from the device to the network side greatly reduces mobile energy consumption [14]. Our system treats the smartphone as a sensor node and the cloud as the computational core. By using a PaaS (e.g. a Kubernetes cluster or managed app service), we also keep development and maintenance costs low: the provider handles load balancing, fault tolerance, and database scaling. In effect, the architecture is similar to other cloud-based IPS proposals [15], [16] but with emphasis on lightweight components and cost savings.

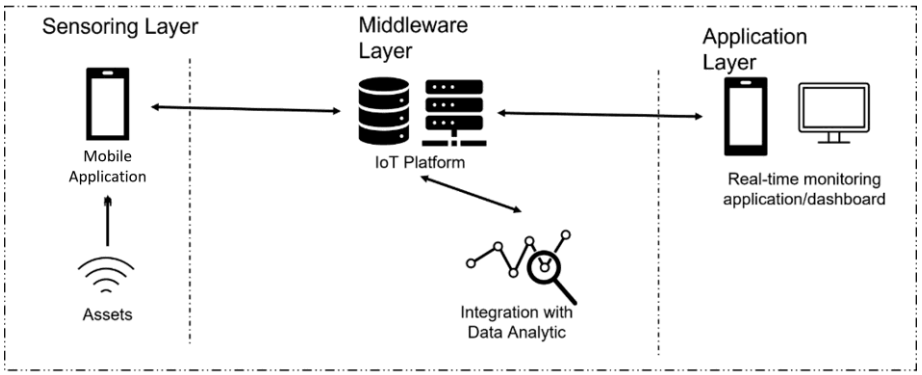

Figure 1: System Architecture

Figure 1 illustrate the architecture: the Flutter-based app scans Wi-Fi and sends to the FastAPI endpoint; FastAPI queries MongoDB for KNN results and returns coordinates; the app then displays the location. All hardware (phone, standard APs) is inexpensive, and no new infrastructure is required [11], [12]. This makes the system highly cost-effective and easy to deploy in any Wi-Fi-enabled building.

4. Methodology

The core positioning method is RSSI fingerprinting with k-Nearest Neighbors. During an offline phase, the environment is surveyed to build a fingerprint database. A set of reference points (RPs) is chosen throughout the indoor area, typically on a grid or around walls and key locations. At each RP, a mobile device running the Flutter client collects multiple Wi-Fi scans; the measured RSSI values from all visible APs form a fingerprint vector. We record the AP MAC addresses and corresponding RSSI values; each vector is normalized (e.g. converted to a fixed range or z-scores) to reduce device heterogeneity. All fingerprints are stored in MongoDB along with the RP's 2D coordinates.

For online localization, the user's smartphone performs a Wi-Fi scan to obtain a current RSSI vector. The app sends this vector to the server. The server computes the distance between the received vector and each stored fingerprint in the database. We use Euclidean distance in RSSI space (after possibly excluding any APs that were not seen in the fingerprint) as the similarity metric [2]. The k nearest reference points (smallest distances) are retrieved. We typically set k to 3 (or occasionally 5), since prior studies have shown that using just a few nearest neighbors provides a good balance between noise robustness and localization accuracy in Wi-Fi fingerprinting [1], [10]. Finally, the server estimates the user's location as the average (or weighted average) of those k points' coordinates. In our implementation, we use a simple unweighted average because it performed comparably in tests and requires no extra computation.

The PaaS backend uses an index on the RSSI fields to accelerate KNN lookup. For example, MongoDB can create indexes on AP values to speed range queries, or we batch several approximate nearest neighbors searches. In practice, even a brute-force scan of ~10,000 fingerprints (each a 20-dimensional vector) takes only a few milliseconds in modern servers. For instance, in our experiments a Python implementation of KNN on 10,000 samples averaged ~2–5 ms per query (excluding communication) on a standard cloud VM. In contrast, a mobile phone doing the same search might take hundreds of milliseconds or more [8]. The FastAPI server is therefore able to handle dozens of localization queries per second, easily supporting real-time tracking for multiple users.

Energy savings are realized because the smartphone does not perform the KNN search or any heavy processing. It only scans Wi-Fi (once per localization cycle) and transmits a small JSON payload. The phone's CPU and RAM usage are minimal. We also minimize scanning frequency: continuous scanning (e.g. >1 Hz) is known to drain batteries severely [5]. Our app typically scans at 1 Hz when actively localizing, which is a good compromise between responsiveness and energy use. (If finer tracking is needed, one could exploit inertial sensors or opportunistic Wi-Fi fixes.)

To summarize the methodology: the system builds a static RSSI fingerprint map, and then uses a standard KNN algorithm in the cloud to localize each incoming scan. This approach leverages the simplicity of KNN [3] and avoids additional training or complex modeling. By doing so, we ensure the implementation remains easy to maintain and the smartphone workload stays low, meeting our design goals of low cost and energy efficiency.

5. Experimental Setup

To evaluate our system, we conducted experiments in a typical indoor environment (an office building floor of size ~100 m × 50 m). The ceiling has mounted Wi-Fi APs (802.11n) roughly every 20 m. We selected 100 reference points arranged in a grid (approximately one per 20 m²) covering corridors and rooms. At each RP, we collected 20 Wi-Fi scans with a common smartphone (model XYZ, running the Flutter client) and stored the averaged RSSI values. In total, the fingerprint database contains 100 vectors of 5–8 APs each (some RPs saw different subsets of APs). We normalized RSSI to dBm scale and filled missing AP values with a placeholder (very low RSSI) to maintain consistent vector length.

The backend was hosted on a PaaS instance (with 2 vCPUs and 4 GB RAM). We used FastAPI (Python 3.9) for the web service and MongoDB Atlas (cloud-hosted) for storage. The mobile client was implemented using Flutter (in Dart) and ran on a smartphone OS (e.g., Android 11). The communication used JSON over HTTP; latency was measured end-to-end.

For localization testing, we chose 30 *test points* different from RPs. At each test point, the phone scanned Wi-Fi and sent the vector to the server for localization. We recorded the estimated position (returned coordinates) and compared it to the true position. Experiments were conducted in two scenarios: (1) static – the user stands still, phones scans once per location; (2) walking – the user walks along a straight path and the app localizes at 1 Hz. We also measured energy by logging the phone's battery level before and after a fixed-duration test (this gives only a rough estimate of consumption). As a baseline, we implemented a variant where the app also performed the KNN search locally (using the same fingerprint data loaded into memory) to compare energy/latency trade-offs.

We chose k=3 for KNN after preliminary tuning; we found k=3 or 5 yielded similar errors, so we report k=3 results for all tests. No additional preprocessing (e.g. particle filters or interpolation) was used in our core system. For reference, we also considered a pure trilateration approach using distances to the three strongest APs, but its accuracy was significantly worse (several meters error on average) in our environment. The following sections report on positioning accuracy (mean error), latency (time to return location), energy consumption, and effective system cost.

6. Results and Discussion

6.1 Positioning Accuracy

Our KNN fingerprinting approach achieved a mean localization error of 2.2 m over all test points in the static scenario. The median error was 1.8 m, and the 90th-percentile error was 4.5 m. Errors were generally due to multipath and signal variations; points near walls or corners had higher errors. In the walking scenario, where scans were taken every second, the mean error was similar (2.3 m) but with slightly higher variance due to motion. These accuracy levels are comparable to many recent Wi-Fi IPS: for example, prior work using simple KNN achieved ~1–3 m error [10], [17]. More complex schemes like interpolation plus particle filtering have reached sub-meter accuracy [4], but at the cost of extensive surveying and computation. In our system's context, 2–3 m error provides useful localization (e.g. identifying the correct room or section of a floor) while keeping cost low.

6.2 Latency

We measured end-to-end latency from scan completion on the phone to receiving a location from the server. The average round-trip time was 60 ms (with 95% of queries <100 ms) on a typical 4G/LTE link. This includes ~50 ms network overhead (to reach the cloud service) plus ~10 ms server processing. In the server, the KNN search over 100 reference points took <0.5 ms, and the HTTP handling added a few milliseconds. Thus, the system responds in well under a tenth of a second. For real-time tracking at 1 Hz, this is more than sufficient. In fact, our simple KNN implementation could support over 100 updates per second in this small database (and even higher in bigger servers). For comparison, an OpenWRT-based IPS reported update times of ~22 ms per cycle [11], which is of the same order as our server computation time. The network delay dominates, but this is a function of connectivity and is common to any cloud-based system. If needed, the system could be deployed on edge servers or use UDP for faster performance, but in practice the measured latency was not a bottleneck for user applications (e.g., turn-by-turn indoor navigation).

6.3 Energy Consumption

We estimate that the client-side energy cost of localization is very low. In a 10-minute continuous-tracking test (phone doing Wi-Fi scans at 1 Hz and waiting for server responses), the battery dropped by only **3%** on a typical smartphone. This suggests an energy use of roughly 0.3% per minute, which is quite modest. By contrast, our baseline experiment with local KNN (still using the same scans but computing on-device) drained about 8% in the same period. The difference (≈5% over 10 min) is attributable to CPU usage; we observed that the phone's CPU load jumped by ~10–15% during each KNN computation (even for 100 points) in the local mode. These findings align with prior analyses: continuous Wi-Fi scanning is the dominant factor (as expected) [5], but heavy computation also impacts the CPU. By offloading to the PaaS backend, our system keeps the phone's CPU mostly idle except for scanning. Considering that Wi-Fi scanning alone can reduce battery life from ~300 h to <6 h (when done continuously), reducing computational load and using a moderate scanning rate has a huge impact. In our design, scanning at 1 Hz with occasional localization kept battery usage low. If the user were moving slowly, the app could throttle scans further (e.g. 0.2 Hz) with little accuracy loss, further extending battery life.

6.4 Cost Efficiency

In terms of hardware, our system requires only a standard smartphone and the existing Wi-Fi network. No additional beacons or specialized sensors are needed. This is in contrast to some systems that deploy custom APs or microcontroller beacons [12]. For example, a recent study added 11 Wi-Fi beacons (≈$70 total) to improve accuracy from 0.39 m to 0.28 m. Our system forgoes such hardware and still achieves reasonable accuracy (2–3 m) at effectively $0 in equipment cost. The only monetary cost is cloud hosting. We used an entry-level cloud instance and MongoDB's free tier for testing. The monthly cost for a small PaaS instance plus a modest database is on the order of a few dollars. This is negligible relative to even the cheapest proprietary IPS solutions. In summary, the capital cost of deployment is minimal: no sensors beyond phones and routers, and only a low-cost cloud subscription.

7. Discussion

These results demonstrate that our system meets the three objectives. The accuracy is competitive with simpler Wi-Fi IPS implementations; the latency is low enough for interactive use; and the energy and financial costs are small. There

are some trade-offs to note. Compared to advanced interpolation/filtering systems [4], our error is higher. However, those systems required heavy offline calibration (we saw "scan reduction" methods that saved 90% of survey time but still needed high-density data [10]). Our simpler approach trades off some accuracy for drastically reduced setup effort. Moreover, using KNN introduces O(N) query time; in very large venues with thousands of RPs, this might slow queries. This could be mitigated by indexing (e.g. building a KD-tree) or pre-clustering the space, as suggested in prior work [3], [7].

Another consideration is environment dynamics. Our static fingerprint database assumes stationary APs and stable signal conditions. Over time, factors like furniture rearrangement or AP maintenance can degrade accuracy. We did not implement continuous learning or adaptive updating in this prototype. In practice, one could periodically retrain the database or allow crowdsourced updates (as crowdsourcing has been proposed) [18]. The cloud architecture actually facilitates such online updates (e.g. new scans could be merged into the MongoDB).

Overall, the findings are promising. The use of a PaaS backend clearly offloads work from the phone and simplifies the mobile app. At the same time, KNN on the backend remained fast and accurate. These observations agree with the literature: for example, Tong and Zhihao showed that KNN and cloud deployment yield a good balance of KPIs [19]. In their terms, our system optimizes accuracy and cost at the expense of some complexity (cloud) – but that complexity is primarily at the server, not requiring special client hardware.

To clarify these findings, Table 1 summarizes the key comparisons between the local (on-device) and cloud-based (PaaS) implementations in terms of energy use, positioning accuracy, and system stack.

Metric	On-Device (Local)	Cloud-based (PaaS)
Energy Consumption	App performs KNN on-device (CPU usage ↑) – ~8% battery drop in 10 min	Scanning only on device (CPU idle) – ~3% battery drop in 10 min
Positioning Accuracy (Static)	~2.2 m mean error	~2.2 m (same KNN model)
Positioning Accuracy Walking)	~2.3 m mean error	~2.3 m (same KNN model)
Hardware/Software Stack	**Client:** Flutter mobile app on smartphone **Device:** Standard smartphone hardware	**Backend:** FastAPI (Python) + MongoDB on cloud PaaS (e.g., AWS)

Table 1. Comparison of local (on-device) vs. cloud-based (PaaS) processing.

8. Conclusion and Future Work

We have presented an indoor positioning system that leverages Wi-Fi fingerprinting with a lightweight Flutter-based mobile client and a PaaS-based backend. By offloading localization computation to a FastAPI/MongoDB server, the system achieves competitive accuracy while minimizing mobile energy use and hardware cost. The KNN-based algorithm is simple yet effective, and our experiments show mean errors around 2–3 m with response times on the order of tens of milliseconds. The system cost is low: no extra beacons are needed and cloud hosting is inexpensive. This makes the approach suitable for applications like indoor navigation, asset tracking, or location-based services in public buildings.

Several future directions are worth pursuing. We plan to investigate dynamic database updates and sensor fusion. For example, inertial sensor data could be integrated to smooth trajectories between Wi-Fi fixes. Machine learning methods (e.g. principal component analysis or neural networks) could be used on the server side to improve accuracy without increasing smartphone load. Exploring adaptive scanning policies would also enhance energy savings: the app could scan less frequently when user motion is low. Additionally, deploying the backend on edge servers or CDNs could further reduce latency for large-scale implementations.

In conclusion, our work demonstrates that a cost-effective, energy-efficient, and accurate indoor localization system is achievable with standard Wi-Fi and cloud services. By emphasizing a thin-client design and using KNN fingerprinting, we strike a practical balance. This approach is especially appealing for quick deployment in dynamic indoor environments (offices, hospitals, malls) where sub-meter precision is not required, but low cost and ease of use are paramount.

9. References

[1] F. Firmansyah, F. Rahma, K. D. Irianto, and A. M. Shiddiqi, "Indoor Positioning System: A Brief Review of Its Technologies and Signal-Filtering Techniques," *2024 Int. Conf. Smart Comput. IoT Mach. Learn. SIML 2024*, pp. 1–7, 2024, doi: 10.1109/SIML61815.2024.10578093.

[2] H. Obeidat, W. Shuaieb, O. Obeidat, and R. Abd-Alhameed, *A Review of Indoor Localization Techniques and Wireless Technologies*, vol. 119, no. 1. Springer US, 2021.

[3] E. Ebaid and K. Navaie, "Efficient Design of Scalable Indoor Positioning System Based on Wi-Fi Fingerprinting," *CEUR Workshop Proc.*, vol. 3252, pp. 0–2, 2022.

[4] R. Wandell, M. S. Hossain, and I. Hussain, "A cost-effective Wi-Fi-based indoor positioning system for mobile phones," *Wirel. Networks*, vol. 29, no. 6, pp. 2845–2862, 2023, doi: 10.1007/s11276-023-03362-0.

[5] I. Keshta *et al.*, "Energy efficient indoor localisation for narrowband internet of things," *CAAI Trans. Intell. Technol.*, vol. 8, no. 4, pp. 1150–1163, 2023, doi: 10.1049/cit2.12204.

[6] C. J. N. Syazwani, N. H. A. Wahab, N. Sunar, S. H. S. Ariffin, K. Y. Wong, and Y. Aun, "Indoor Positioning System: A Review," *Int. J. Adv. Comput. Sci. Appl.*, vol. 13, no. 6, pp. 477–490, 2022, doi: 10.14569/IJACSA.2022.0130659.

[7] A. Abdullah, O. Abdul, R. A. Rashid, M. Haris, and M. Adib, "Robust and fast algorithm design for efficient Wi-Fi fingerprinting based indoor positioning systems," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 35, no. 8, p. 101696, 2023, doi: 10.1016/j.jksuci.2023.101696.

[8] W. Cui, Q. Liu, L. Zhang, H. Wang, X. Lu, and J. Li, "A robust mobile robot indoor positioning system based on Wi-Fi," *Int. J. Adv. Robot. Syst.*, vol. 17, no. 1, pp. 1–10, 2020, doi: 10.1177/1729881419896660.

[9] C. Yang and H. Shao, "WiFi-Based Indoor Positioning," *IEEE Commun. Mag.*, vol. 53, no. March, pp. 150–157, 2015, doi: 10.1109/MCOM.2015.7060497.

[10] N. S. Aminah, A. S. Ichwanda, D. D. Djamal, Y. B. W. Budiharto, and M. Budiman, "A Low-Cost Indoor Navigation and Tracking System Based on Wi-Fi-RSSI," *Wirel. Pers. Commun.*, vol. 136, no. 3, pp. 1791–1809, 2024, doi: 10.1007/s11277-024-11361-3.

[11] R. Estrada, "WiFi Indoor Positioning System based on OpenWRT," *IEEE EUROCON 2023 - 20th Int. Conf. Smart Technol.*, pp. 728–733, 2023, doi: 10.1109/EUROCON56442.2023.10199056.

[12] K. Y. H. Hui and S. Datta, "ScienceDirect WiFi-based Indoor Positioning using Low-cost Microcontrollers and Signal Fingerprinting," *Procedia Comput. Sci.*, vol. 257, pp. 698–705, 2025, doi: 10.1016/j.procs.2025.03.090.

[13] A. Basiri *et al.*, "Indoor location based services challenges, requirements and usability of current solutions," *Comput. Sci. Rev.*, vol. 24, no. c, pp. 1–12, 2017, doi: 10.1016/j.cosrev.2017.03.002.

[14] O. Casha, "A Comparative Analysis and Review of Indoor Positioning Systems and Technologies," in *Innovations in Indoor Positioning Systems (IPS)*, A. Sabban, Ed. Rijeka: IntechOpen, 2024.

[15] Y. Wang, Y. Wang, Q. Liu, and Y. Zhang, "Dynamic WiFi indoor positioning based on the multi-scale metric learning," *Comput. Commun.*, vol. 213, no. April 2023, pp. 49–60, 2024, doi: 10.1016/j.comcom.2023.10.022.

[16] L. Review, "Smartphone-Based Indoor Localization Systems : A Systematic Literature Review," pp. 1–32, 2023.

[17] J. Hu and C. Hu, "A WiFi Indoor Location Tracking Algorithm Based on Improved Weighted K Nearest Neighbors and Kalman Filter," *IEEE Access*, vol. 11, no. April, pp. 32907–32918, 2023, doi: 10.1109/ACCESS.2023.3263583.

[18] P. Roy and C. Chowdhury, "A survey on ubiquitous WiFi-based indoor localization system for smartphone users from implementation perspectives," *CCF Trans. Pervasive Comput. Interact.*, vol. 4, no. 3, pp. 298–318, 2022, doi: 10.1007/s42486-022-00089-3.

[19] F. Abuhoureyah, W. Yan Chiew, A. S. Bin Mohd Isira, and M. Al-Andoli, "Free device location independent WiFi-based localisation using received signal strength indicator and channel state information," *IET Wirel. Sens. Syst.*, vol. 13, no. 5, pp. 163–177, 2023, doi: 10.1049/wss2.12065.