Indoor Positioning System for Ubiquitous Computing Environments

Pedro Albuquerque Santos[1,2(✉)], Rui Porfírio[1(✉)], Rui Neves Madeira[1,2(✉)], and Nuno Correia[1(✉)]

[1] NOVA LINCS, NOVA University of Lisbon, Lisbon, Portugal
{pe.santos,r.porfirio}@campus.fct.unl.pt, nmc@fct.unl.pt
[2] Sustain.RD, ESTSetúbal, Polytechnique Institute of Setúbal, Setúbal, Portugal
rui.madeira@estsetubal.ips.pt

Abstract. We developed an Indoor Positioning System (IPS) as part of the effort of creating Ubicomp applications with user interfaces distributed across different co-located devices. It relies on a Client that runs on the devices that we intend to locate and a Server that determines their positions. It currently supports three positioning methods: fingerprinting, trilateration and proximity. Bluetooth Low Energy and Wi-Fi are used as the underlying technologies for the positioning methods. We tested multiple machine learning algorithms during the development of the system to choose the ones providing satisfactory results. A Mean Absolute Error around or below 1 m and 95th percentile errors in the 2 m range were considered acceptable according to the type of target applications. We were also able to integrate the system into our framework and built a cross-device application that took advantage of it.

Keywords: Indoor positioning system · Ubiquitous computing · Proxemics · Machine learning · Wi-Fi · Bluetooth low energy

1 Introduction

We are living in a ubiquitous world as we face widespread ubiquitous computing (UbiComp) due to the billions of devices shaping the very fabric of an active world [18]. Nowadays, computing devices can be found everywhere around people, who are increasingly using a large number of them in their daily lives [13]. Therefore, we took on the challenge and opportunity of leveraging the presence of these co-located devices to build applications that can distribute their user interface (UI) across them (cross-device applications).

We soon identified that we would need a way to continuously pinpoint the location of the multiple devices that are part of a pervasive environment in order to automatically determine which devices are in range of use of each other to automatically distribute UI elements. The Global Positioning System (GPS)

H. Yin et al. (Eds.): IDEAL 2021, LNCS 13113, pp. 611–622, 2021.
https://doi.org/10.1007/978-3-030-91608-4_59

and commercial indoor location-based system work well outdoors and in large indoor spaces (e.g., airports and shopping malls). However, there is no readily available solution for indoor positioning that could be used throughout the rooms in a house or in offices to determine the location of the devices present in the surroundings of the potential users of cross-device applications.

Given the lack of options, we set out to create an indoor positioning system (IPS) that could be easily used by the framework that we implemented for supporting cross-device applications development. We were also concerned with ease of access, cost effectiveness and adaptability to the environment. Therefore, we set out to use commonly available technologies as starting point for our system, e.g., Bluetooth and Wi-Fi. We decided to support multiple positioning techniques, so we created a decision system to select the most appropriate one. Moreover, we avoided having to directly deal with the intricacies of radio wave propagation by employing machine learning (ML) models that were trained according to their behavior. We tested multiple ML algorithms during the development of the system, including: k-Nearest Neighbors, Support Vector Machines, Random Forest, Multilayer Perceptron, Linear Regression, and the BFGS optimization algorithm. The resulting IPS was developed in a way that it can be reused by any other UbiComp applications and systems that may require its services, thus constituting a useful contribution for the UbiComp community.

In the remaining of the paper, we introduce previous research related to our work (Sect. 2). We present the architecture of the IPS and explain how it works (Sect. 3). Section 4 presents some performance metrics for each of the supported positioning techniques, including a discussion regarding the machine learning algorithms and other parameters that enabled us to choose the best default values for our system. We finish by presenting conclusions, based on the experience we had using IPS with application prototypes, and future work.

2 Related Work

The research on indoor positioning systems has been extensive throughout the years resulting in efficient solutions based on a comprehensive set of technologies and techniques [7]. Radio frequency covers a substantial amount of studies that apply technologies like *Bluetooth*, *Wi-Fi*, and *Ultra-Wideband* to estimate the indoor position of a device. Due to its potential to achieve ubiquitous, low-power consumption, and low-cost solutions, *Wi-Fi* and *Bluetooth* established a solid reputation and popularity in the realm of indoor positioning systems [7,16].

Considering *Wi-Fi* approaches, the *RADAR* system was a pioneering work on *Wi-Fi* fingerprinting application for indoor positioning, reaching an accuracy of 2.94 m [4]. It is also worth mentioning Xia et al. work, which provides an in-depth overview of the application of *Wi-Fi*-based fingerprinting to indoor positioning settings [2]. The authors provide an analysis of the benefits (low-cost and high precision) and disadvantages (heavy labor cost to maintain the radio map) of fingerprinting and its influence factors, such as signal attenuation from people's presence in the environment.

Regarding Bluetooth-based research, Faragher and Harle research the impact of Bluetooth Low Energy (*BLE*) devices in advertising mode on fingerprint-based indoor positioning schemes and also propose a comparison between *Wi-Fi* and *BLE* fingerprinting [10]. Feldmann et al. introduce a *Bluetooth*-based system that applied trilateration and the least square estimation for location finding [11].

The design of UbiComp systems with an intrinsic proxemic-based context-awareness has been the core of a significant amount of research throughout the years [1,5]. In particular, the concept of proxemics interactions (i.e., interactions in which the devices have a fine-grained knowledge of nearby people and other devices) has further enhanced the use of proxemics in Human-Computer Interaction (*HCI*) [12].

Furthermore, the research on indoor positioning solutions has been quite fond of the adoption of ML techniques. The interest arises from the benefits ML offers in fields as pattern recognition, which can easily be transposed to indoor positioning environments, such as the online matching process in fingerprinting approaches [2]. A common approach is using a deterministic positioning algorithm like k-Nearest Neighbors (*k-NN*) regression algorithm. *k-NN* compares a device's *RSSI* with previously scanned *RSSI* in distinct reference points called fingerprints. It applies the matching procedure by finding the k most similar fingerprints to the real-time signal strength captured, using a distance metric as Euclidean distance. The k fingerprints will hold the most likely positions of a user. A final step usually applied to increase accuracy is to average the k positions and consider the mean value as the user's final location.

Among the studies that apply ML in indoor positioning solutions, the research of Blasio et al. leverages a modified version of *kNN* (Weighted K-Nearest Neighbor) to employ *Wi-Fi* and *Bluetooth* fingerprinting for harsh environments by applying the Euclidean distance and the weighted *k-NN* as a matching algorithm between fingerprints and real-time scans [6]. To aid in increasing the pervasiveness of an indoor positioning solution, there have been several studies that propose a fuzzy logic support system that exploits the uncertainty of a user's location. Orujov et al. present a fuzzy logic scheme, applying the Mamdani method, to aid in the decision support system aiming to select the most fitting positioning technique, depending upon three crisp inputs: the size of the room, *RSSI* from *BLE* beacons, and the number of available beacons [17].

3 Indoor Positioning System

We now present how the Indoor Positioning System (*IPS*) that we developed to be used as part of our efforts to build more engaging ubiquitous computing applications was developed. It enables applications to react proxemic dimensions of the relationships between the computing devices present in the environment, namely distance, orientation and identity [12].

3.1 Architecture

A general overview of the architecture of the system is provided in Fig. 1.

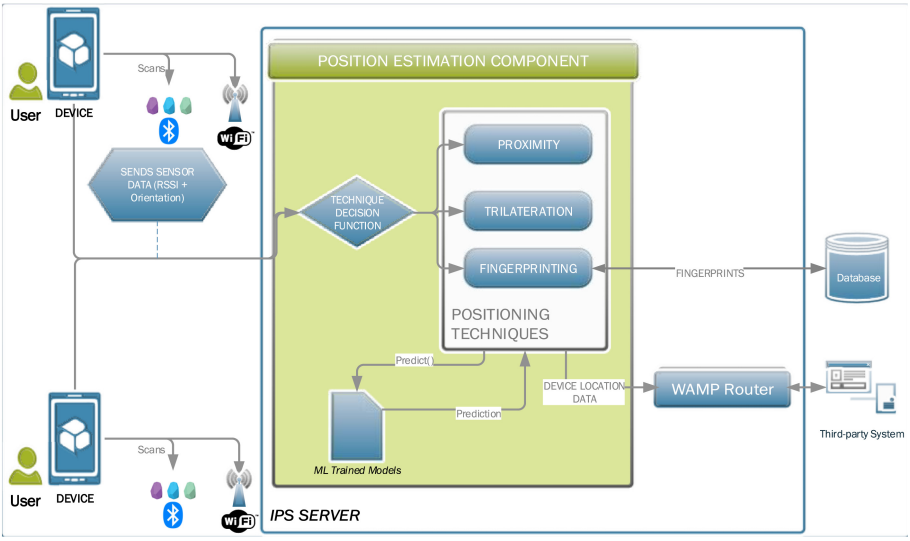

Fig. 1. Architecture of the *IPS* system

The indoor positioning system was implemented by following a *client-server model*. The *Client* runs on each device the system intends to locate. It scans for Bluetooth Low Energy (*BLE*) beacons, *Wi-Fi* access points, and collects the orientation of the device. It then continuously sends that information to the *Server*. The *Client* component has two implementations: one supporting mobile devices running *Android 6.0* or higher; and another for *Linux PCs*.

The *Server* was implemented using the *Python* programming language because it gives us access to useful data analysis and ML libraries. The *Server* is responsible for determining the position of the devices from that information using the supported positioning techniques. It is also capable of communicating with the *Clients* and third-party systems that are interested in the position of the devices. The communication between *Client* and *Server* relies on a *REST API*. However, the communication between the *Server* and third-party systems requires the ability to continuously push positioning events containing the current position of client devices. Therefore, we used the Web Application Messaging Protocol (*WAMP*). It provides the implementation of a *publish-subscribe* pattern over *WebSocket* connections [20].

3.2 Scanning Procedure

The *Client* applications hold the responsibility of continuously scanning for *BLE* beacons, *Wi-Fi* access points, and for collecting the orientation of the devices.

The data saved in each sample includes the Received Signal Strength Indicator (*RSSI*) values for the *BLE* beacons and *Wi-Fi* access points, and the *azimuth* angle in *radians*, i.e., rotation about the *-z-axis* [3]. An Universally Unique Identifier (*UUID*) is also included in order to identify the devices in a particular environment.

Each *Wi-Fi* scan takes a couple of seconds and reports all the access points it finds in a single batch [9]. Conversely, the *BLE* scan continuously reports any beacon as soon it is found. Moreover, previous research highlights a more severe fast fading multipath interference problem with *BLE* in indoor positioning settings than with *Wi-Fi* mainly due to a shorter channel length [10]. From an usability perspective, according to Kim et al. the response time of a proximity detection positioning system should range between 1 to 3 s [14]. Therefore, to alleviate such problem and accommodate the different scanning behaviors, the scanning rate is set to 3 s and in the case of the *BLE* beacons a moving average is applied to help smoothing out any errors.

3.3 Position Estimation

We chose to use *BLE* and *Wi-Fi* because they are widely supported technologies in modern computing devices. Moreover, *Wi-Fi* access points are widely deployed, *BLE* beacons are relatively inexpensive, and many devices can be configured to emit their own beacon signals. There was also an extensive body of research in indoor positioning systems using *Bluetooth* and *Wi-Fi* technologies with satisfactory results for our use case from which we could draw inspiration.

The *Server* is responsible for estimating the position of the client devices through a set of positioning techniques: *fingerprinting*, *proximity*, and *trilateration*. It uses *fuzzy logic* to choose the most appropriate positioning technique for the current environment. The Mamdani system takes four variable features of the environment into consideration [15]: number of scanned *BLE* beacons; percentage of scanned *Wi-Fi* access points and number of scanned *BLE* beacons that have a substantial impact on position prediction (high *feature importance*); and number of scanned *BLE* beacons with known coordinates.

The *fingerprinting* technique consists of an *offline phase* responsible for data acquisition, and an *online phase* in which position predictions are made. The *offline phase* is performed by the mobile *Client* application which collects fingerprint samples scanned in a particular environment and sends them to the *Server*. After being signaled that an area has been completely scanned, the *Server* retrieves the samples from the database and converts them to a Comma-separated Values *CSV* file representing a radio map. The *Server* can then loads the radio map and train a ML model. During the *online phase*, the samples scanned in real-time by a *Client* application are sent to the server. If *fingerprinting* is selected as the positioning technique to be used by the decision system, the server will have to select the best matching radio map based on the detected access points and beacons. It will then used the corresponding ML model that was previously trained to produce a prediction of the absolute position of the device based on the received samples.

The **proximity** technique focuses on the notion of Proximity-based Services, which is the notion of relative location or context-based position (e.g., user U is near beacon B). Therefore, it only predicts the distance between a device and a *BLE* beacon, i.e., it only returns information about the relative position of a device. It is also comprised of an *offline phase* and an *online phase* since we use a ML approach instead of calculating distances from *RSSI* values based signal on path loss. An *offline phase* is required to collect samples to train the algorithm. The mobile *Client* application is responsible for data collection, sending the scanned samples at specific distances from a reference beacon to the *Server*. Although the system could rely on *Wi-Fi* access points, it is best designed to work with *BLE* beacon signals since their deployment in the environment is more flexible and best suited for real-life scenarios. This radio map collected during the *offline phase* is then used to train the ML model. During the *online phase*, if the decision system chose *proximity* as the positioning technique to use, the *BLE* beacon samples collected by a client device will be used by the trained model to make a distance prediction.

Trilateration does not require a dedicated *offline phase*. However, it relies on the *proximity* method to compute the distance between a client device and a set of beacons present in the surrounding environment. It also requires that the *Server* is configured with the coordinates of at least three beacons or access points. Similarly to the *proximity* technique, *trilateration* is designed to work with both radio technologies, but *BLE* beacons are recommended. When *trilateration* is chosen as the most appropriate positioning technique by the decision system, the *Server* will retrieve the locations of the beacons captured by the *Client*, compute the distance between the client device and the beacons using the *proximity* technique and it will estimate the absolute position of the device using least-squares optimization to minimize the sum of squared errors. This optimization process is required because it is very unlikely that the circles defined by the calculated distances intersect each other on a single point which leads to a set of equations without an analytical solution.

For all of the positioning techniques, once the prediction for the position of a client device is made, the *Server* publishes the prediction to the *onLocationUpdate* topic on the *WAMP* router. This pattern allows any interested third-party service to subscribe to this topic to continuously receive updates about the positions of client devices.

4 Evaluation

Each indoor positioning technique has multiple variables that can be adjusted. They also have their own advantages and disadvantages when it comes to accuracy and precision. Therefore, we designed a series of experiments to determine the default configuration for each technique and also to determine which technique performs better in certain situations in order to configure the decision system based on *fuzzy logic*. In this section, we present a summary of the results of these experiments and the decisions that were taken.

4.1 Fingerprinting

The dataset used to predict the absolute position of a device in a cartesian coordinate system using *fingerprinting* was a radio map scanned in $4\,\text{m} \times 4\,\text{m}$ room with 25 reference points and 30 fingerprints scanned per reference point (750 samples).

The online matching phase in *fingerprinting* assesses the similarity between samples scanned in real-time by a device and the fingerprints previously saved in a radio map. Since it is assumed that the fingerprints already have labels associated, supervised learning algorithms are commonly used as part of the matching stage. Therefore, a set of five experiments were developed to test the following ML algorithms: k-Nearest Neighbors (*k-NN*), Random Forest (*RF*), Support Vector Machine (*SVM*), and Multilayer Perceptron (*MLP*).

The experiment encompassed tuning the hyperparameters of each algorithm using *Stratified K-Fold* cross-validation. The Leave One Group Out (*LOGO*) strategy was also used to simulate the worst-case scenario, i.e., the algorithm is trained with samples from all reference points except the one currently being tested. Tables 1 show the comparison between the regression performance results of the multiple ML algorithms we considered for fingerprinting.

Table 1. Performance Evaluation (best results in green, selected algorithms in bold)

Algorithm	**MAE**	**RMSE**	r^2	P_{95}
Fingerprinting				
k-NN	0.292113	0.602620	0.909212	1.489585
SVM	0.548867	0.705709	0.875494	1.496627
RF	**0.260042**	0.485201	0.941145	1.187140
MLP	0.879875	1.028073	0.735767	1.875221
k-NN (LOGO)	1.462987	1.603255	0.357393	2.597401
SVM (LOGO)	1.390924	1.547562	0.401263	2.631749
RF (LOGO)	1.394118	1.508732	0.430932	2.520780
MLP (LOGO)	**1.390476**	1.515245	0.401356	3.060226
Proximity				
k-NN	**0.483617**	0.723037	0.746707	1.150000
SVM	0.471081	0.806262	0.685040	1.353657
LR	0.784444	0.960112	0.553372	1.547085
MLP	0.579432	0.793403	0.694958	1.253388
k-NN (LOGO)	**0.945925**	1.295717	0.372113	1.816667
SVM (LOGO)	1.144035	1.402022	0.047464	1.726999
LR (LOGO)	1.078450	1.325616	0.148456	1.722772
MLP (LOGO)	1.190065	1.456349	0.027785	1.985990
Trilateration				
Localization Package	1.041392	1.140057	-	1.895565
SciPy Minimize	**1.041344**	1.140028	-	1.89558
Brute-force	0.948854	1.131371	-	1.882843

RF is not always the best algorithm, but it achieved good performance across all of the experiments and it was selected as the one to be used by default. Another aspect that had to be considered is that collected fingerprints often have missing data for certain access points and beacons because they may appear in a certain place and during a certain scan, and yet be missing from other scans. Therefore, the missing data must be replaced somehow. We experimented with multiple replacement strategies, including replacing the missing values with the *maximum*, *minimum*, *median* or *mean* value of the dataset. We also experimented with calculating these globally (i.e., across the whole dataset) or for each access point or beacon.

The best results were obtained with the *mean* or *median* strategies when applied to each access point or beacon. However, the *minimum* replacement strategy also provided good results and it has a more sound reasoning behind it. Since lower *RSSI* values should correspond to access points or beacons that are further away, it seems logical that those that are not found during a scanning cycle are represented as being as further away as possible. Therefore, this was the strategy adopted for the remaining experiments and the final solution.

4.2 Proximity

Similarly to *fingerprinting*, the *proximity* technique also applies a ML algorithm to match the samples collected during the *online phase* with the data gathered during the *offline phase*. The following algorithms were compared to select the one which performed better: *k-NN*, *SVM*, *MLP*, and Linear Regression (*LR*). The dataset used to determine which ML algorithm to use was a radio map with 10 reference points placed between 0 and 4.5 m away from the beacon (24834 samples in total). Table 1 depicts the performance results of the various regression algorithms that were tested for the proximity technique.

Another experiment compared if collecting more data at each reference point improved results and in general the results showed that it did have an appreciable effect on results. Therefore, we recommend collecting as much data as possible when configuring our system. We were also interested in assessing if the model trained to estimate the distance from one beacon could be used interchangeably for other beacons. Similarly, we were interested in determining if a model trained from a dataset collected in an environment could be used on a different one (e.g., on another room). In both cases, there was a loss in performance but they were small enough that we believe that the trade-off between the additional work of having to collect data for each specific beacon/environment and the performance loss is generally worth it. Therefore, the system makes predictions based on the same radio map, regardless of the target beacon.

4.3 Trilateration

Trilateration leverages the *proximity* technique to predict the distance between a device and reference beacons in the environment, meaning there is a dependency between the performance of *trilateration* and *proximity*. The goal was to study

the best approach to predict the position of a device based on the distance estimates provided by the *proximity* technique.

The analytical approach of determining the intersection of multiple circles from their equations cannot be used because the inaccuracy of the distance estimates often leads to the definition of circles that do not intercept in a single point. Therefore, the *trilateration* experiments focused on solving the problem from an optimization point of view. The optimization process tries to find the point X that provides the best approximation to the device position P. Specifically, X is the point whose distance to the known beacons best fits the distance between a target device's position and the beacons in a particular environment. A common approach is to find the point X that minimizes the sum of squared errors, i.e., a least-squares optimization problem. Consequently, the experiments studied how the following optimization approaches perform:

- **Brute Force Approach**: Finds the point X out of a list of possible P that best fits the estimated distances to each beacon.
- **Optimization Algorithm**: Given the positions of the beacons, use an optimization algorithm that iteratively finds the point X that best fits the estimated distances to each beacon.

It is assumed that the device is at one of the 25 points with saved *BLE RSSI* samples and will analyze the prediction errors at each point according to the Mean Absolute Error (MAE) metric. The brute force approach finds the point X that minimizes the mean squared distance error between X and the set of known beacons, with X being one of a fixed set of available points. In order to analyze its performance, the experiment considered three possible datasets with varying granularity: a $4\,\text{m} \times 4\,\text{m}$ room divided into a grid of 25 points spaced 1 m apart; 100 points spaced 0.5 m apart; or 2500 points spaced 0.1 m apart. This results in a MAE of 1.04747, 1.044542 and 0.948854, respectively. As expected, the increased granularity improves results at the cost of increased computational complexity. Moreover, the brute force approach is a pretty native algorithm and it requires a dataset to be generated beforehand, which includes information about all possible candidate points that form the grid pattern.

Optimization algorithms can be used as an alternative that only requires the positions of the beacons in the environment. Therefore, we experimented with the following libraries that implement least-squares optimization:

- **SciPy Minimize Function**: It was used by applying a bound-constrained optimization algorithm (L-BFGS-B [8,21]) with the sum of the squared error as the target error function to be minimized [19]. The initial prediction of the function is the target device's nearest beacon.
- **Localization package**[1]: It applies *trilateration* by trying to find a point that minimizes the sum of squared distance errors to the position of the devices. The system only requires the distance of the target device to known beacons and their respective positions. The underlying optimization algorithms are also provided by SciPy [19].

[1] https://github.com/kamalshadi/Localization.

Table 1 also outlines the performance results for trilateration, with the best brute force result as a baseline. The two tested implementations performed identically. The *SciPy Minimize Function* was selected since it is part of a well established library that offers more options to further improve the results.

4.4 Discussion

Since *proximity* only provides relative positions, it cannot be directly compared with *fingerprinting* and *trilateration*, which provide the absolute location of a device. Absolute positions offer more information and can also be used to infer the relative positions in respect to other devices and *PoIs*. Moreover, although *fingerprinting* present good overall results, its performance results in the worst-case scenario ($LOGO$) degrade considerably. Therefore, *trilateration* presents itself as the positioning technique with the good performance and more consistent results. It also has the added advantage of requiring considerably less data collection during the *offline phase*.

The decision system's rules support the performance results by establishing a hierarchy among the available positioning techniques according to their overall performance. The rules outline a hierarchy of the positioning techniques with *trilateration* being the default technique that is used when all conditions are *good enough*. Therefore, the hierarchy is *trilateration* > *fingerprinting* > *proximity*.

Nonetheless, the prioritization of *fingerprinting* above *proximity* is not so much about performance results but about the information provided. *Fingerprinting* provides the absolute position of a device allowing to compute the relative position in relation to another absolute position. However, *proximity* only displays the relative position in relation to a beacon in the environment, a beacon attached to a device, or a beacon signal emitted by a device.

5 Conclusions and Future Work

The current version of the IPS has been able to meet most of our requirements and expectations. The errors that we presented in Sect. 4 were acceptable given our use case. Mean absolute errors (MAE) around or below 1 m and a 95th percentile errors in the 2 m range seem reasonable for the type of applications that we envisioned. Being capable of determining if two devices are in the same area within a room is enough for most cross-device applications and may be also true for many other potential UbiComp applications.

We have integrated successfully IPS with our framework, allowing to create applications that take into consideration the proxemic relationships established between the devices running them. We used one of those applications in a user study to evaluate the concept of cross-device applications. Most users agreed that the application detected quickly the presence of the other devices during the experiment, also meaning the underlying IPS responded positively to the users' expectations. During the user study, we used the *proximity* technique to get estimates of the distances between devices. This decision was taken to

simplify the deployment of our test environment, because building a radio map for *fingerprinting*, or carefully placing *BLE* beacons for *trilateration*, was not required. Besides, we only needed the relative distance and the orientation of the devices for the scenarios that were being tested. Moreover, the estimates returned by *proximity* in previous experiments were more stable and accurate than relative distances calculated from absolute positions provided by the *fingerprinting* or *trilateration* methods.

These issues are probably due to the fact that we are dealing with two position estimates, each one already with some associated error, which gets magnified when the euclidean distance between two points is calculated. Furthermore, there is currently no direct relationship between the position of a device at moment t and at $t + 1$ because the computing of each position is stateless, which means that subsequent position estimates may diverge from the previous one by an unrealistic amount (e.g., distance between positions is larger than it would be possible at walking speed). This effect may be attenuated if it incorporates information about previous estimates into the current one, i.e., by using LSTM (Long short-term memory) neural networks or Hidden Markov Models (HMM).

We also believe that we should still be able to improve the accuracy and response times of the IPS. It is still possible to explore better ways to process data during the *online* and *offline phases*, and to improve the usage of the available ML algorithms through better parameterization or by customizing them to better fit this application domain. Fine grained centimeter level accuracy may be desirable to build more complex and richer interaction scenarios. There is the possibility of extending the system with other technologies such as Wi-Fi Round Trip Time (RTT) and Ultra-wideband (UWB), which are becoming more easily available and have the potential of reaching higher levels of accuracy.

References

1. Mueller, F., et al.: Proxemics play: understanding proxemics for designing digital play experiences. In: Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, DIS, pp. 533–542. Association for Computing Machinery (2014). https://doi.org/10.1145/2598510.2598532
2. Indoor fingerprint positioning based on Wi-Fi: An overview (May 2017). https://doi.org/10.3390/ijgi6050135
3. SensorManager - Android Developers (2021). https://developer.android.com/reference/android/hardware/SensorManager
4. Bahl, P., Padmanabhan, V.N.: RADAR: An in-building RF-based user location and tracking system. Technical Report (2000). https://doi.org/10.1109/infcom.2000.832252
5. Ballendat, T., Marquardt, N., Greenberg, S.: Proxemic interaction: Designing for a proximity and orientation-aware environment. In: ACM International Conference on Interactive Tabletops and Surfaces, ITS 2010, pp. 121–130 (2010). https://doi.org/10.1145/1936652.1936676
6. de Blasio, G., Quesada-Arencibia, A., García, C.R., Molina-Gil, J.M., Caballero-Gil, C.: Study on an indoor positioning system for harsh environments based on Wi-Fi and bluetooth low energy. Sensors (Switzerland) **17**(6), 1299 (2017). https://doi.org/10.3390/s17061299, http://www.mdpi.com/1424-8220/17/6/1299

7. Brena, R.F., García-Vázquez, J.P., Galván-Tejada, C.E., Muñoz-Rodriguez, D., Vargas-Rosales, C., Fangmeyer, J.: Evolution of Indoor Positioning Technologies: a survey (2017). https://doi.org/10.1155/2017/2630413
8. Byrd, R.H., Lu, P., Nocedal, J., Zhu, C.: A limited memory algorithm for bound constrained optimization. SIAM J. Sci. Comput. **16**(5), 1190–1208 (1995). https://doi.org/10.1137/0916069
9. Faragher, R., Harle, R.: An analysis of the accuracy of bluetooth low energy for indoor positioning applications. In: Proceedings of the 27th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2014), pp. 201–210 (2014). http://www.ion.org/publications/abstract.cfm?jp=p&articleID=12411
10. Faragher, R., Harle, R.: Location fingerprinting with bluetooth low energy beacons. IEEE J. Sel. Areas Commun. **33**(11), 2418–2428 (2015). https://doi.org/10.1109/JSAC.2015.2430281, http://ieeexplore.ieee.org/document/7103024/
11. Feldmann, S., Kyamakya, K., Zapater, A., Lue, Z.: An indoor Bluetooth-based positioning system: concept, implementation and experimental evaluation. In: Proceedings of the International Conference on Wireless Networks, pp. 109–113 (2003)
12. Greenberg, S., Marquardt, N., Ballendat, T., Diaz-Marino, R., Wang, M.: Proxemic interactions: the new ubicomp? interactions **18**(1), 42 (2011). https://doi.org/10.1145/1897239.1897250, http://portal.acm.org/citation.cfm?doid=1897239.1897250
13. Kantar TNS Germany: The Connected Consumer (2019). https://www.google.com/publicdata/explore?ds=dg8d1eetcqsb1_
14. Kim, D.Y., Kim, S.H., Choi, D., Jin, S.H.: Accurate indoor proximity zone detection based on time window and frequency with bluetooth low energy. Procedia Comput. Sci. **56**(1), 88–95 (2015). https://doi.org/10.1016/j.procs.2015.07.199
15. Mamdani, E.H., Assilian, S.: An experiment in linguistic synthesis with a fuzzy logic controller. Int. J. Man. Mach. Stud. **7**(1), 1–13 (1975). https://doi.org/10.1016/S0020-7373(75)80002-2, https://linkinghub.elsevier.com/retrieve/pii/S0020737375800022
16. Mautz, R.: Indoor Positioning Technologies. Ph.D. thesis (2012). https://doi.org/10.3929/ethz-a-007313554, http://e-collection.library.ethz.ch/eserv/eth:5659/eth-5659-01.pdf
17. Orujov, F., Maskeliūnas, R., Damaševičius, R., Wei, W., Li, Y.: Smartphone based intelligent indoor positioning using fuzzy logic. Future Gener. Comput. Syst. **89**, 335–348 (2018). https://doi.org/10.1016/j.future.2018.06.030
18. Rodden, T.: Living in a ubiquitous world. Philos. Trans. Royal Soc. Math. Phys. Eng. Sci. **366**(1881), 3837–3838 (2008). https://doi.org/10.1098/rsta.2008.0146
19. Virtanen, P., et al.: SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Methods **17**(3), 261–272 (2020). https://doi.org/10.1038/s41592-019-0686-2, http://www.nature.com/articles/s41592-019-0686-2
20. WAMP: The Web Application Messaging Protocol - Web Application Messaging Protocol version 2 documentation (2021). https://wamp-proto.org/
21. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-BFGS-B. ACM Trans. Math. Softw. **23**(4), 550–560 (1997). https://doi.org/10.1145/279232.279236