A Bisection Reinforcement Learning Approach to 3-D Indoor Localization

Fei Dou, *Member, IEEE*, Jin Lu, Tingyang Xu, Chun-Hsi Huang, and Jinbo Bi

***Abstract*—The demand for indoor localization services in the Internet of Things (IoT) has been increasing dramatically during the last decade. Many indoor localization systems adopt Wi-Fi fingerprinting with received signal strength indicators (RSSIs) as a source of sensors to localize an object because it is cost effective and can give high accuracy. However, the fluctuation of wireless signals resulting from environmental uncertainties leads to considerable variations in RSSIs, which poses a challenge to accurate localization on a single floor, not to mention multifloor or even 3-D localization. Most existing multifloor methods employ a sequential approach where a different algorithm is tailored for each step in the sequence to determine the floor and then the location of an object. In this article, we formulate the indoor localization problem as a Markov decision process rather than a typical classification or regression problem. A deep reinforcement learning method is used to bisect the search space in a hierarchy from the entire building down to a prespecified distance scale to the object position. This approach significantly reduces the time complexity of the searching from $\mathcal{O}(N^3)$ to $\mathcal{O}(\log N)$, where N indicates the localization resolution. The proposed method tackles environmental dynamics with Wi-Fi fingerprinting for 3-D continuous space. The experimental results demonstrate the high accuracy, efficiency, and robustness of the proposed approach.**

***Index Terms*—Deep Q-network (DQN), deep reinforcement learning (DRL), dynamic environment, indoor localization, Internet of Things (IoT), multifloor, received signal strength indicator (RSSI), Wi-Fi fingerprint.**

I. Introduction

THE Internet of Things (IoT) [1] describes a system of connecting different entities or "things" to provide ubiquitous connectivity and enhanced services, which can be achieved by connecting any "thing" with sensors to the Internet [2]. In the past decade, the proliferation of things, such as smartphones and other wireless devices, has promoted services that require accurate and robust indoor localization of certain object(s) in a wide range of living, commerce, production, and public services. Indoor localization aims to identify the location of a target (a device or a user) in an indoor environment and usually in a setting with wireless networks. It has been challenging because the global positioning system (GPS) signal [3], which serves as a standard solution for outdoor localization, cannot penetrate well in the indoor environment. Indoor localization has benefited from modern technologies, such as visible light, acoustic signal, ultrasound, and radio communication technologies, including Wi-Fi, Bluetooth, ZigBee, radio-frequency identification (RFID), and ultrawideband (UWB) [4]–[6]. Various signal metrics are considered, such as received signal strength indicator (RSSI), channel state information (CSI), Angle of Arrival (AoA), Time Difference of Arrival (TDoA), and Time of Flight (ToF).

Among the different techniques, Wi-Fi fingerprinting with RSSIs from different access points (APs) has been an effective source for indoor localization because the collected Wi-Fi signals vary according to indoor locations. It has high accuracy, high feasibility, simplicity, and deployment practicability [3], [7]. Wi-Fi fingerprinting usually involves two phases: 1) an offline phase where RSSIs are collected from different APs at many known locations to build a fingerprint database of the environment and 2) an online phase where the position of a target is estimated by comparing the current captured RSSIs with those in the database.

However, the fluctuation and interference of wireless signal lead to considerable variations on RSSIs, and factors that have been observed affecting the RSSIs include but not limited to fading and shadowing, object presence and movement, open/closed doors [8], and the relative humidity level in a dynamic environment. These environmental uncertainties pose grand challenges to the positioning accuracy of fingerprint-based indoor localization.

Many machine learning algorithms, such as K-nearest neighbors (KNNs) [9], Naive Bayesian [10], support vector machine (SVM) [11], random forest (RF) [12], and neural network (NN) [13], have been employed to find the most probable location from the fingerprints. Basically, existing indoor localization methods can be organized into two categories.

1) *Regression Methods:* The exact location coordinates of a target are predicted according to the Wi-Fi signals captured (e.g., RSSI values). For example, the KNN algorithm obtains the K-nearest matches of the known locations using the root mean-square error (RMSE) based on the offline RSSI measurements stored in the database. The nearest matches are then averaged to

Manuscript received July 16, 2020; revised October 2, 2020 and November 6, 2020; accepted November 12, 2020. Date of publication November 30, 2020; date of current version April 7, 2021. The work of Jinbo Bi was supported in part by the National Science Foundation under Grant IIS-1718738. *(Corresponding author: Jinbo Bi.)*

Fei Dou and Jinbo Bi are with the Department of Computer Science and Engineering, University of Connecticut, Storrs, CT 06269 USA (e-mail: fei.dou@uconn.edu; jinbo.bi@uconn.edu).

Jin Lu is with the Department of Computer and Information Science, University of Michigan–Dearborn, Dearborn, MI 48128 USA (e-mail: jinluz@umich.edu).

Tingyang Xu is with the Tencent AI Lab, Tencent Inc., Shenzhen 518057, China (e-mail: tingyangxu@tencent.com).

Chun-Hsi Huang is with the School of Computing, Southern Illinois University, Carbondale, IL 62901 USA (e-mail: chuang@cs.siu.edu).

Digital Object Identifier 10.1109/JIOT.2020.3041204

predict an estimated location of the target. It suffers from the fact of low regression accuracy.

2) *Classification Methods:* The indoor environment is divided into small grids of equal size, and Wi-Fi signals are then used to classify which predefined grid or zone that the target resides in [13]. Nevertheless, it can be difficult to determine an appropriate size for the grid because it may require prior knowledge, such as the floor plan. If the grid resolution needs to change, localization models often have to be retrained. Furthermore, finer localization requires the entire search space to have a higher resolution grid, and the search complexity is in a cubic order with respect to the resolution in 3-D space.

Most indoor localization studies are based on a 2-D single floor, which cannot meet the demand for multifloor situations, such as large shopping malls, airports, and factories. The state-of-the-art multibuilding and multifloor methods assume a sequential approach to positioning, where the building, floor, and location of a target are estimated in a sequence using different algorithms for each task [14]–[16]. The model for each task is trained with a different subdataset, making the model hard to scale, especially to large and complex indoor environments. There can be two scenarios of 3-D localization: 1) multifloor localization and 2) 3-D localization problems, where continuous distance is also used in the vertical direction rather than discrete floors.

In this article, we propose a bisection method based on reinforcement learning that keeps bisecting the search space into octants (3-D localization) or quadrants (2-D localization) in a hierarchy. Still, it bisects only the octant or quadrant that contains the target as determined by the reinforcement learning model. The initial search space can be a cube in 3-D situations or a square in 2-D localization. The bisecting process continues until it finds an octant/quadrant of small size that substantially intersects with the target so that the localization is within a prespecified distance to the target. The well-studied single-floor and multifloor indoor localization can be solved by our approach as special cases. The contributions of this article are as follows.

1) The proposed approach takes the environmental dynamics into account. It models the indoor localization problem as a Markov decision process (MDP), where an agent runs deep reinforcement learning (DRL) that interacts with the environment dynamically and selects actions to narrow down the bounding region that contains the target.
2) A bisecting method is used at all coordinate directions in 3-D (or 2-D) search space to exponentially reduce the search region. Thus, the runtime complexity is dramatically reduced from $\mathcal{O}(N^3)$ by the grid methods to $\mathcal{O}(\log N)$, where N is the number of grids in each coordinate and determines the localization resolution.
3) A top-down searching approach is developed and has three main advantages. First, it does not require any prior knowledge of the floor plan or building plan in the indoor environment. Second, it is no longer needed to partition the indoor environment into grids so to avoid the drawbacks of grid-based methods. Third, because of the hierarchical structure, the approach is capable of providing on-demand resolution of localization (just by further bisecting) without retraining any model.
4) The proposed approach provides a unified framework for single-floor, multifloor, and 3-D indoor localization where horizontal position (coordinates on one floor) and the vertical position (or floor information) could be inferred simultaneously. It is readily adaptive to multibuilding (4-D) indoor localization.
5) Because DRL has the capability of learning in a real-time manner, the new approach does not need to retrain and memorize all the past data samples. Consequently, our localization model can achieve sufficient accuracy quickly, enabling real-time positioning.

II. Related Work

Single-Floor Localization: A variety of machine learning approaches have been proposed for indoor localization with Wi-Fi RSSIs in IoT. Yang *et al.* [9] proposed a KNN-based method by investigating the sensor data from smartphone and user motions to construct the radio map of a floor plan. Tran and Pham [11] adopted the model-based classification approach based on SVMs. However, these methods used hand-crafted features, so they may not utilize the sensing data fully to learn features. A four-layer deep NN (DNN) was used to extract features from the raw sensing data and estimate locations in [13] by dividing the indoor environment into hundreds of square grids and classifying the target into a grid. Nguyen [17] performed a literature review and compared the performance of the most popular machine learning approaches based on Wi-Fi fingerprinting, e.g., weighted KNN, Naive Bayes, and NNs. It suggested that if only Wi-Fi RSSIs were used, complex algorithms might not outperform simple ones. Despite the simplicity of the weighted KNN method, it excelled in most fingerprinting techniques, which is why KNN is the most widely used benchmarking algorithm for indoor localization based on Wi-Fi fingerprinting.

The most relevant work to our method is the one that uses DRL but only for single-floor localization in [18]. It proposed a semi-supervised DRL framework as a learning mechanism in support of smart IoT services and experimented in an indoor localization system. An unsupervised navigation and localization method was proposed in [19] to use a DRL algorithm. However, these methods require the floor plan to be represented by small grids beforehand. As discussed before, any change in the location accuracy requirement renders the entire space to be repartitioned to a finer grid, and then models need to be retrained. These methods can be inefficient because the agent has to search grid-by-grid to find the target on the floor, which might require hundreds of steps before reaching the target, depending on the initial position. We proposed a top-down searching method using a DRL agent with two fully connected layers to provide on-demand resolution in [20] without the need to divide the floor into grids. This method can localize a target with high accuracy, typically within ten steps. This present article is a comprehensive extension of the method

in [20] to a unified framework that can perform 2-D and 3-D indoor localization.

Multifloor Localization: For multibuilding and multifloor indoor localization, Nowicki and Wietrzykowski proposed a DNN architecture with stacked autoencoder (SAE) and feedforward classifier, but it is only able to predict the building and floor rather than coordinates on the floor [21]. Another method first determined the target's floor and used the KNN method to find its location on that floor [14]. Thus, the task was separated into multiple subtasks. Similarly, Song *et al.* decomposed the problem into three subtasks: 1) building classification; 2) floor classification; and 3) position regression, and proposed a convolution NN (CNN) with SAE to extract features [15]. A major disadvantage of these methods is that the models used at the building, floor, and location levels need to be trained separately with multiple subdatasets. The positioning accuracy has been relatively low for the NN-based methods. Kim *et al.* [16] followed the work in [21] and proposed a method to simultaneously perform building/floor/location prediction using a DNN for multilabel classification. Although the output dimension is reduced from $n_b \times n_f \times n_l$ to $n_b + n_f + n_l$ where n_b, n_f, and n_l denote the numbers of buildings, floors, and locations, respectively, the dimension is still large (hundreds) and the network architecture is complex for the task. Furthermore, it does not have the ability to predict locations beyond the existing samples from the offline database.

Reinforcement Learning [22] is a type of machine learning approach for optimal control and decision making, where an agent learns an optimal policy of actions over a set of system states by interacting with the system environment. It has a wide range of applications, such as robotics [23], games [24]–[26], image classification and object detection [27], [28], etc. The best-known successes of reinforcement learning are Atari 2600 computer gaming system [24], [25] and AlphaGo meeting the challenges of Computer Go [26], to name a few.

Mnih *et al.* [25] introduced a deep Q-network (DQN) and kick started the revolution in scaling reinforcement learning to complex sequential decision-making problems. It presented the first DRL model to successfully learn control policies directly at a human level from high-dimensional sensory input, which contained raw image pixels. This method stabilized the training for estimating a value function using experience replay, and the value function was implemented via a CNN. It also designed a reinforcement learning approach that directly used the image pixels and the game score as inputs. AlphaGo [26] had made historical events by beating several human world champions in the Go game and became a milestone in artificial intelligence. This hybrid learning system was built with reinforcement learning techniques, deep CNN, and Monte Carlo tree search (MCTS).

DQN has been an important milestone, and many extensions have been proposed. Van Hasselt *et al.* [29] proposed a double DQN (DDQN) to tackle the over-estimate problem in Q-learning, which was addressed by decoupling the selection of an action from its evaluation: evaluating the greedy policy according to an online network, whereas using a target network to estimate its value. Built on top of DDQN, Schaul *et al.* [30] proposed *prioritized experience replay* so as to replay important experience transitions more frequently to learn more efficiently. Wang *et al.* [31] proposed the Dueling DQN, which could converge faster than Q-learning to estimate the action-value function by featuring two streams of computation: 1) a state value function and 2) an associated advantage function. In [32], Deep Q-Learning from Demonstrations (DQfDs) was proposed to accelerate the learning process even from relatively small amounts of demonstration data and was able to automatically assess the necessary ratio of demonstration data while learning based on a prioritized replay mechanism. The method of *noisy network* [33] introduced a noisy linear layer that combines a deterministic and noisy stream to overcome the limitations of exploring with the $\epsilon-$greedy policy, where many actions must be executed to collect the first reward. Anschel *et al.* [34] proposed *Averaged-DQN* to reduce variability and instability by averaging the Q-values estimated in the previous rounds. Hessel *et al.* [35] examined six extensions to the DQN algorithm and proposed the *Rainbow* method to combine the improvements from those DQN extensions.

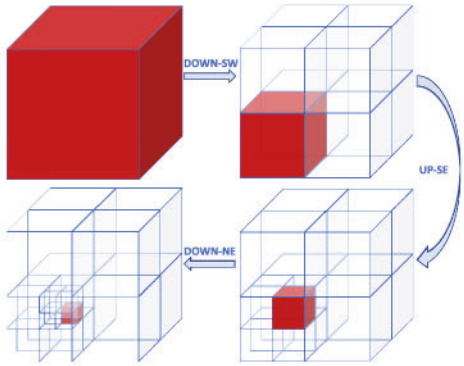


Fig. 1. MDP for indoor localization.

III. Indoor Localization as Markov Decision Process

In an MDP [22], an agent keeps interacting with a system environment and decides an action from the prescribed action space in sequence to achieve a specific goal. In this section, we formulate the indoor localization problem as a dynamic decision-making process rather than a regression problem predicting the target's coordinates or a classification problem where classes represent the coarse region grids.

The geometry of the search space, such as floor plan and/or the floor numbers, and the RSSI signals in the entire search space are defined as the environment, within which the agent shifts and shrinks a bounding region via a series of actions. The agent moves to the next state after taking a specific action at the current state. After the targeted object enters the environment and communicates any RSSI signals, the agent makes progress in iterations to eventually bound it within a cubic region of prespecified small size (a given precision for the localization). In each iteration, the agent determines how to move and reshape the search cube to find the target in as few steps as possible. Fig. 1 gives a simple demonstration of our approach where a building is enclosed in an initial cube,

and the agent selects an action each time to bisect the search space (the red-color octant) further into smaller octants until the octant size reaches a threshold.

We parameterize the MDP with three components: 1) the *state space* $\mathcal{S}$; 2) the *action space* $\mathcal{A}$; and 3) the *reward function* r in terms of an action at a state, which are explained in the following sections. Note that an MDP contains the fourth component: the transition rule or transition probabilities that specify how likely the current state transits to a state in the state space. We omit the transition rule because in our setting, the transition from a state to the next state is deterministic on the action that is taken. We illustrate the three MDP components in the 3-D continuous search space where the searching space at three directions (x, y, z) is all continuous. The *single-floor localization* and *multifloor localization* are two special cases of our MDP, where the z dimension is either fixed or takes discrete values (floor numbers), respectively. In the description of each component for the 3-D environment, we also include discussions for the two special cases.

A. State Space

1) Definitions: The localization state space $\mathcal{S}_{3D}$ of our MDP comprises three elements in the representation $\{\mathcal{S}_{3D} : (\mathbf{RSSI}, \mathbf{cb}, \mathbf{h})\}$.

1) A vector **RSSI** contains all RSSI values.
2) A vector $\mathbf{cb} = [\mathbf{c}, \text{rad}]$, where $\mathbf{c}$ represents the center coordinates (x, y, z) of the current search cube and the radius rad denotes half the length of the cube side.
3) A vector $\mathbf{h}$ records the history of the actions taken in each searching round.

The history vector $\mathbf{h}$ captures all the actions that the agent has performed during each searching round, and is encoded as a one-hot vector. If there are d possible actions to take, each action in $\mathbf{h}$ is represented by a binary vector of length d, where the entry corresponding to the taken action is set to 1, and others are all 0. We use a one-hot encoding instead of integer encoding because integer values have a natural ordered relationship between each other, and this relationship may be misunderstood and harnessed by the algorithm, possibly resulting in poor performance. The history vector of length nd encodes n past actions where n depends on the largest number of steps to localize the target in the indoor environment. The $\mathbf{h}$ vector is used to stabilize the search trajectories. As discussed in the next section, there are $d = 8$ or $d = 4$ possible actions in a 3-D or a 2-D search space, respectively.

2) Remark: The state defined above in a 3-D continuous space can include the single-floor and multifloor situations.

Single-Floor Localization: The state takes the same form of three components (**RSSI**, **cb**, $\mathbf{h}$) except the z dimension of the center **cb** is removed and the bounding region becomes a window: $\mathbf{w} = [(x, y), \text{rad}]$, where (x, y) specifies the center of a square window defined in Section III-B2 and rad is the half of the window side. The history vector $\mathbf{h}$ encodes n prior actions, each of which is represented using a 4-entry vector, leading to a total length of $4n$.

Multifloor Localization: The state also takes the same three components, but the z dimension of the cube is discrete, and the bounding region becomes a 3-D unit where the vertical level only takes floor numbers: $\mathbf{u} = [(x, y, \mathbf{f}), \text{rad}]$ where $\mathbf{f}$ records the selected floor set in each searching round, and (x, y) and rad are the same as those in the single-floor localization. The floor set $\mathbf{f}$ is encoded as a vector with a length equal to the total number of possible floors, and during the search, certain floor numbers will be turned into 0 until only one floor remains. The history vector in multifloor localization encodes n past actions, leading to a total length of $8n$.

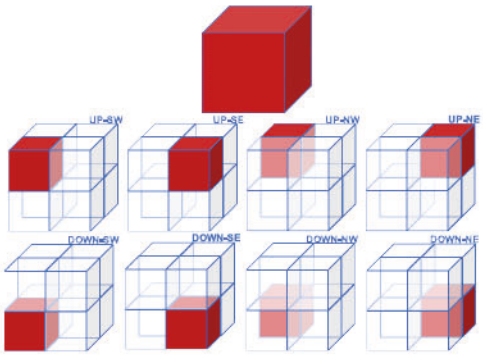


Fig. 2. Eight actions for our MDP in 3-D localization.

B. Action Space

1) Definitions: Because our approach starts the search from the entire building as the search cube, every time when we shrink the cube by bisecting it along all the x, y, and z directions, we obtain eight octants, which are shown in Fig. 2. The proposed action space composes of eight possible actions to shrink the cube, denoted as $\{\mathcal{A}_{3D}$: "UP-SW" (South West), "UP-SE" (South East), "UP-NW" (North West), "UP-NE" (North East), "DOWN-SW" (South West), "DOWN-SE" (South East), "DOWN-NW" (North West), "DOWN-NE" (North East)$\}$.

Each action transits the system state to a new one. Specifically, the center vector $\mathbf{c}_t$ at time point t is updated to $\mathbf{c}_{t+1}$ as follows.

1) *"UP-SW":* $\mathbf{c}_{t+1} = (x_t - \frac{\text{rad}_t}{2}, y_t - \frac{\text{rad}_t}{2}, z_t + \frac{\text{rad}_t}{2})$.
2) *"UP-SE":* $\mathbf{c}_{t+1} = (x_t + \frac{\text{rad}_t}{2}, y_t - \frac{\text{rad}_t}{2}, z_t + \frac{\text{rad}_t}{2})$.
3) *"UP-NW":* $\mathbf{c}_{t+1} = (x_t - \frac{\text{rad}_t}{2}, y_t + \frac{\text{rad}_t}{2}, z_t + \frac{\text{rad}_t}{2})$.
4) *"UP-NE":* $\mathbf{c}_{t+1} = (x_t + \frac{\text{rad}_t}{2}, y_t + \frac{\text{rad}_t}{2}, z_t + \frac{\text{rad}_t}{2})$.
5) *"DOWN-SW":* $\mathbf{c}_{t+1} = (x_t - \frac{\text{rad}_t}{2}, y_t - \frac{\text{rad}_t}{2}, z_t - \frac{\text{rad}_t}{2})$.
6) *"DOWN-SE":* $\mathbf{c}_{t+1} = (x_t + \frac{\text{rad}_t}{2}, y_t - \frac{\text{rad}_t}{2}, z_t - \frac{\text{rad}_t}{2})$.
7) *"DOWN-NW":* $\mathbf{c}_{t+1} = (x_t - \frac{\text{rad}_t}{2}, y_t + \frac{\text{rad}_t}{2}, z_t - \frac{\text{rad}_t}{2})$.
8) *"DOWN-NE":* $\mathbf{c}_{t+1} = (x_t + \frac{\text{rad}_t}{2}, y_t + \frac{\text{rad}_t}{2}, z_t - \frac{\text{rad}_t}{2})$.

The radius rad is updated by $\text{rad}_{t+1} = \alpha \times \text{rad}_t$, where $\alpha = (1/2)$ because we bisect the cube. Note that a more general strategy would be to shrink the cube in a certain ratio other

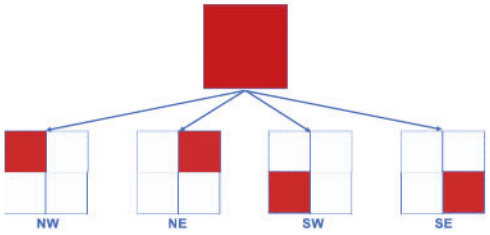


Fig. 3. Four actions for our MDP in single-floor localization.

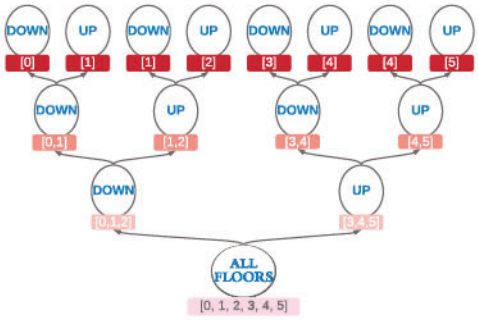


Fig. 4. Actions in the vertical level in multifloor localization.

than only (1/2). For instance, we may allow the search octants to overlap slightly, so the shrinkage ratio is $(1/2)+\epsilon$ where ϵ is a small machine number. Our approach does not require prior information about the search space as long as a container cube can be specified. Because the radius of the search cube keeps reducing during the searching, the localization resolution (or precision) can be determined in a real-time fashion unlike the grid search approach where changing resolution means to vary the grids and retrain the model.

2) Remark: The actions defined above in a 3-D continuous space can be easily modified to cover both single-floor and multifloor environments.

Single-Floor Localization: The action becomes *Window Selection* at the horizontal level, yielding four possible actions: {"SW," "SE," "NW," "NE"} at each step, as shown in Fig. 3. The transition rule on $\mathbf{c}_t = (x_t, y_t)$ is the same as that defined in Section III-B1 (with the z dimension removed).

Multifloor Localization: At each step, the agent performs the same eight actions in $\mathcal{A}_{3D}$ as specified exactly in Section III-B1, except the z dimension is different. In this regard, "UP" means taking the upper half of the floors in the current floor-set, and "DOWN" means taking the lower half of the floors. Fig. 4 shows an example with six floors and the possible actions in the vertical level: "UP" and "DOWN." Note that when the number of floors in the current floor set is an odd number, both UP and DOWN actions should include the middle floor in the next state.

C. Reward Function

1) Definitions: The reward function r is designed to reflect if an action can improve the system from the current state to a better state. The improvement in a 3-D environment is measured using the Intersection of Cube (IoC) between the target cube and the predicted cube where IoC $\in [0, 1]$, as shown in Fig. 5(a). The reward for an action reveals if the

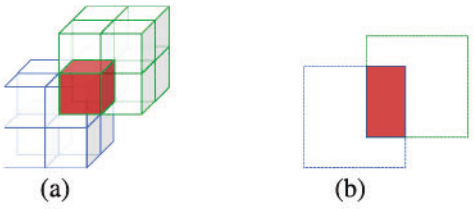


Fig. 5. Measurement of improvement. (a) IoC. (b) IoW.

IoC has been improved by the action. Let $\mathbf{cb}^g$ be the ground-truth cube of the target. Then, the IoC between $\mathbf{cb}$ and $\mathbf{cb}^g$ equals the percentage of the volume in the search cube $\mathbf{cb}$ that overlaps with the ground-truth cube $\mathbf{cb}^g$, i.e.,

$$\mathrm{IoC}\left(\mathbf{cb}, \mathbf{cb}^g\right) = \mathrm{volume}\left(\mathbf{cb} \cap \mathbf{cb}^g\right)/\mathrm{volume}(\mathbf{cb}) \tag{1}$$

where volume() calculates the volume of a cube.

In our top-down searching scheme, the predicted cubic region scales down to the target. At step t, the agent gains a positive reward if the IoC of the next state s_{t+1} is larger than that of the current state s_t, meaning that the agent chooses a "correct" action to get closer to the target. Note that a correct action keeps the target inside the predicted cube while the cube size rad decreases, so the IoC becomes larger. A large positive reward will be assigned to the agent when the IoC of the new state exceeds a threshold δ, and the search is terminated. When the agent chooses a "wrong" action, leading the predicted cube away from the target, it receives a large negative penalty and terminates the search.

The two parameters—the target cube size $\mathrm{rad}_{\mathrm{gt}}$ in $\mathbf{cb}_g$ and the threshold δ—play a tradeoff between the localization precision and searching cost. A larger $\mathrm{rad}_{\mathrm{gt}}$ leads to a lower localization precision because the agent may reach a rough region at the final step in fewer steps. Note that the setting of $\mathrm{rad}_{\mathrm{gt}}$ indicates the finest localization resolution the model could achieve. A larger δ leads to a higher localization accuracy, but the agent may take a longer time to learn a correct policy, or sometimes RSSI data may not have enough granularity for the agent to learn correctly.

When the agent takes the action a_t, the system transits from s_t to s_{t+1}, and the reward function $r_{3D,a_t}^{(s_t,s_{t+1})}$ is defined as

$$r_{3D,a_t}^{(s_t,s_{t+1})} = \begin{cases} +\eta & \text{if } \mathrm{IoC}(\mathbf{cb}^{s_{t+1}}, \mathbf{cb}^g) \in [\delta, 1] \\ +\tau & \text{if } \mathrm{IoC}(\mathbf{cb}^{s_{t+1}}, \mathbf{cb}^g) \in (\mathrm{IoC}(\mathbf{cb}^{s_t}, \mathbf{cb}^g), \delta) \\ -\eta & \text{otherwise} \end{cases} \tag{2}$$

where $\eta > 0$ is much larger than $\tau > 0$. The values of η, τ, and δ are determined as described in Section IV-A.

2) Remark: The reward function defined for a 3-D continuous space can be easily modified to fit with the single-floor and multifloor situations.

Single-Floor Localization: Similar to the IoC, the Intersection of Window (IoW) is defined to measure the improvement in single-floor localization as illustrated in Fig. 5(b). Then, the IoW between the current search window $\mathbf{w}$ and the ground-truth window $\mathbf{w}^g$ is defined as

$$\mathrm{IoW}\left(\mathbf{w}, \mathbf{w}^g\right) = \mathrm{area}\left(\mathbf{w} \cap \mathbf{w}^g\right)/\mathrm{area}(\mathbf{w}) \tag{3}$$

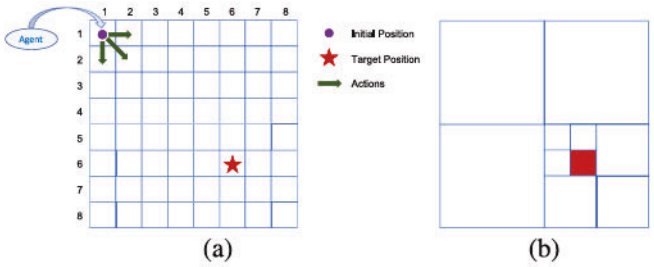


Fig. 6. Time complexity for the search in a 2-D environment. (a) Search strategy in [18]. (b) Our search strategy.

where area() calculates the area of a window. Then, after agent executes the action a_t, the system transits from s_t to s_{t+1}, the reward function $r_{SF,a_t}^{(s_t,s_{t+1})}$ is defined as

$$r_{SF,a_t}^{(s_t,s_{t+1})} = \begin{cases} +\eta & \text{if } \mathrm{IoW}(\mathbf{w}^{s_{t+1}}, \mathbf{w}^g) \in [\delta, 1] \\ +\tau & \text{if } \mathrm{IoW}(\mathbf{w}^{s_{t+1}}, \mathbf{w}^g) \in (\mathrm{IoW}(\mathbf{w}^{s_t}, \mathbf{w}^g), \delta) \\ -\eta & \text{otherwise.} \end{cases} \tag{4}$$

Multifloor Localization: In a multifloor environment, the selected floor set needs to shrink into the exact floor where the target resides simultaneously when the search window is scaled down horizontally to the target location on that floor. Let $\mathbf{f}^{s_t}$ be the currently selected floor set, and $\mathbf{f}^g$ is the floor of the target. At step t, if either $\mathbf{f}^g$ is not included in $\mathbf{f}^{s_t}$ or the current $\mathbf{w}^{s_t}$ does not contain $\mathbf{w}^g$, the agent receives a large penalty $-\eta$ and the search is terminated. When $\mathbf{w}^{s_t}$ overlaps $\mathbf{w}^g$ substantially passing the threshold δ, and $\mathbf{f}^{s_t}$ pinpoints the correct floor, the agent receives a large reward η and terminates the search. In general, when an action a_t makes the state transit from s_t to s_{t+1}, the reward function $r_{MF,a_t}^{(s_t,s_{t+1})}$ is defined as

$$r_{MF,a_t}^{(s_t,s_{t+1})} = \begin{cases} +\eta & \text{if } \mathbf{f}^g = \mathbf{f}^{s_t} \text{ and } \mathrm{IoW}(\mathbf{w}^{s_{t+1}}, \mathbf{w}^g) \in [\delta, 1] \\ +\tau & \text{if } \mathbf{f}^g \in \mathbf{f}^{s_t} \text{ and} \\ & \mathrm{IoW}(\mathbf{w}^{s_{t+1}}, \mathbf{w}^g) \in (\mathrm{IoW}(\mathbf{w}^{s_t}, \mathbf{w}^g), \delta) \\ -\eta & \text{otherwise.} \end{cases} \tag{5}$$

D. Complexity Analysis

We argue that the proposed approach significantly reduces the search complexity from those methods based on grid search. The time complexity is measured by the number of steps a search algorithm has to take in order to find the target under a given localization resolution.

An early method in [18] adopts DRL to indoor localization, shown in Fig. 6(a), where the environment is divided into a grid of equal-sized cells. The action is to move from a cell to a neighboring cell in a direction of north, east, west, south, or in diagonal directions such as north west. Thus, the agent has to move grid by grid to localize the target in the environment. Following this logic, if the environment extends to a 3-D space, illustrated in the upper right part in Fig. 7, the worst case time complexity becomes $\mathcal{O}(N^3)$, where N is the number of cells in each direction of the grid and is determined by the localization precision (or resolution).

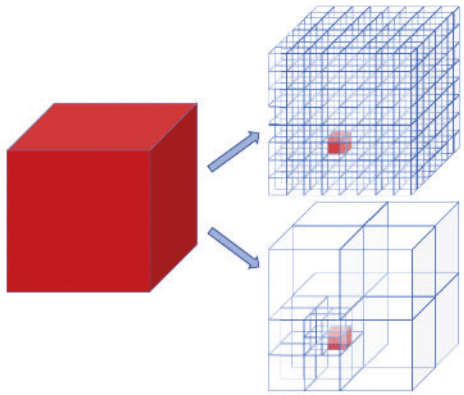


Fig. 7. Time complexity for the search in a 3-D environment.

In our approach, shown in the lower right part in Fig. 7 [or Fig. 6(b) for 2-D localization], the bisecting method applied to all three directions in a 3-D space can exponentially reduce the search space, leading to a time complexity of $\mathcal{O}(\log N)$. This is a significant drop from $\mathcal{O}(N^3)$ in the grid search.

Specifically, the time complexity of our approach in *single-floor localization* is $\mathcal{O}(\log N)$, shown in Fig. 6(b), and that in *multifloor localization* is $\max(\mathcal{O}(\log N), \mathcal{O}(\log M))$, where M denotes the number of total floors in a building and N indicates $N \times N$ grids of the floor.

IV. Deep Reinforcement Learning for Localization

Hessel *et al.* [35] compared the performance of the DQN algorithm and its six extensions, and proposed the *Rainbow* method to combine the improvements of those DQN extensions. In our model, we use the most basic and simple DQN algorithm to show the potential of the proposed MDP model.

The goal of the agent in the MDP is to learn a policy π of selecting actions a_t (from an action space $\mathcal{A}$) to interact with an environment (from one state s_t to another state s_{t+1} in a state space $\mathcal{S}$) so that the expected reward r is maximized. In reinforcement learning, the standard assumption is that future rewards are discounted by a factor $\gamma \in [0, 1]$ for each step, which trades off the importance between the immediate and later rewards. Define the future discounted *return* at time t as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where T is the step at which the search terminates. The true value function $Q(s, a)$ is defined as the expected return function in terms of taking the action a at the state s

$$Q_\pi(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{6}$$

where $\pi = P(a|s)$ is a distributions over actions given s.

The optimal return value is $Q^*(s, a) = \max_\pi Q_\pi(s, a)$ and an optimal policy can be easily derived from the optimal Q function by selecting the action that achieves the highest Q value in a state. $Q^*(s, a)$ obeys the Bellman [22], which is based on the following intuition: if the optimal value $Q^*(s_{t+1}, a_{t+1})$ of a state s_{t+1} at the next time point was known for all possible actions a_{t+1}, then the optimal strategy is to select the action a_t that maximizes the expected

value of $r_t + \gamma Q(s_{t+1}, a_{t+1})$

$$Q^*(s_t, a_t) = \mathbb{E}_{s_{t+1}}\left[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})|s_t, a_t\right]. \quad (7)$$

A reinforcement learning algorithm estimates an action-value function using the Bellman equation as an iterative update, $Q_{i+1}(s_t, a_t) = \mathbb{E}[r_t + \gamma \max_{a_{t+1}} Q_i(s_{t+1}, a_{t+1})|s_t, a_t]$, where i denotes the ith iteration. Such value iteration algorithms converge to the optimal action-value function: $Q_i \to Q^*$ as $i \to \infty$. However, this basic approach can be impractical, because the action-value function is estimated discretely for each observed state and cannot generalize to not-yet-observed states. It is common to represent the action-value function by a model, such as an NN parameterized by θ : $Q(s, a; \theta) \approx Q^*(s, a)$.

It is then derived into the DQN [25] method, which trains a multilayered NN by adjusting the parameter θ at each iteration i to reduce the mean-squared error in the Bellman equation. More precisely, in the DQN, there are two separate deep networks: 1) an online network parameterized with θ and 2) a target network parameterized with θ^-, respectively, aiming at further improving the stability of the network. The target network is the same as the online network except that its parameters are synchronized every C steps from the online network, so that then $\theta_i^- = \theta_{i-1}$, but are fixed at all other steps. In this way, the optimal target values, $r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})$, are substituted with approximate target values $y_{i,t} = r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta_i^-)$. The Q-learning update in iteration i minimizes the following loss function:

$$L_{i,t}(\theta_i) = \mathbb{E}_{s_t, a_t \sim \rho(.)}\left[\left(y_{i,t} - Q(s_t, a_t; \theta_i)\right)^2\right] \quad (8)$$

where

$$y_{i,t} = \mathbb{E}_{s_{t+1}}\left[r_t + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta_i^-\right)|s_t, a_t\right] \quad (9)$$

is the target for iteration i and $\rho(s, a)$ is a probability distribution over states s and actions a, which are referred to as the *behavior distribution*. At each stage of optimization, the parameters from the previous iteration θ_i^- are held fixed when optimizing the ith loss function $L_i(\theta_i)$. Differentiating the loss function with respect to the weights yields the following gradient:

$$\nabla_\theta L_{i,t}(\theta_i) = \mathbb{E}_{s_t, a_t, s_{t+1}}\left[\left(r_t + \gamma \max_{a_{t+1}} Q\left(s_{t+1}, a_{t+1}; \theta_i^-\right) - Q(s_t, a_t; \theta_i)\right)\nabla_{\theta_i} Q(s_t, a_t; \theta_i)\right]. \quad (10)$$

A. Realization of the Q-function

Fig. 8 shows the structure of the DNN that we use to realize the action-value (Q) function in our model. Two NNs of the same architecture as shown in Fig. 8 are employed in the deep Q-learning framework, which is specified in Algorithm 1. The network in Fig. 8 consists of three fully connected hidden layers of 2000, 1448, and 548 nodes, respectively. The activation function used in this network is the rectified linear unit

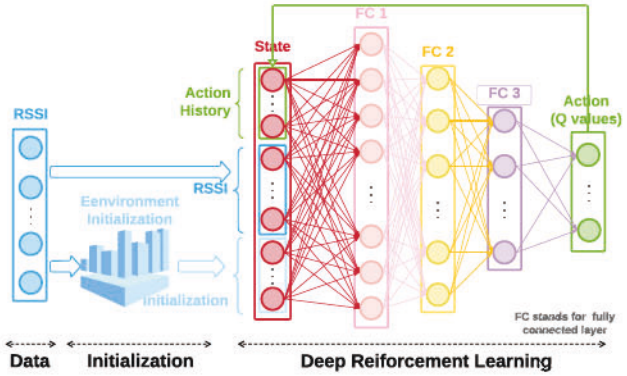


Fig. 8. DNN used in our algorithm.

(ReLU). The output layer possesses eight nodes, each corresponding to the Q values of one of the eight actions defined in the action space. The input layer may contain varying numbers of nodes according to the specific applications. The input of the network comprises the system state defined in the state space, including nodes for RSSI values of the target, nodes for the cube center, radius, and action history. Hence, the NN takes in the current state and outputs a vector containing the Q values of every action. Based on these Q values, we use an exploration scheme that chooses the action having the highest Q value with high probability $1 - \epsilon$ and chooses a random action with a probability of $\epsilon > 0$.

B. Initialization

The initial **RSSI** is a vector containing all the RSSI values from different APs in the environment, which will be fixed for one data sample in each searching round. Assume all data samples are bounded by $x_{\min}$, $x_{\max}$, $y_{\min}$, $y_{\max}$, $z_{\min}$, and $z_{\max}$, which we refer to as the initial cubic region. For the initial cube $\mathbf{cb}_0 = [\mathbf{c}_0, \text{rad}_0]$, we set $\mathbf{c}_0$ as the center of the bounded cubic region, and set rad_0 to half of the largest side of the region. Precisely, we initialize the parameters as follows:

$$\begin{cases} \mathbf{c}_0 = (x_0, y_0, z_0) = \left(\frac{x_{\min}+x_{\max}}{2}, \frac{y_{\min}+y_{\max}}{2}, \frac{z_{\min}+z_{\max}}{2}\right) \\ \text{rad}_0 = \frac{\max(x_{\max}-x_{\min}, y_{\max}-y_{\min}, z_{\max}-z_{\min})}{2} + \text{rad}_{\text{gt}} \end{cases} \quad (11)$$

where rad_{gt} denotes the estimated radius (or half of the side) of the target cube, which represents localization resolution. Note that the actual size of the target is not needed to run our algorithm. Here, the estimated rad_{gt} reflects the detection precision required by a user. The initial history vector $\mathbf{h}_0$ of appropriate size ($4n$ and $8n$, respectively, in 2-D and 3-D cases) is a vector of zeros.

The initialization of the window information $\mathbf{c}_0 = (x_0, y_0)$ and rad_0 for single-floor localization is the same as defined above, but just ignoring the z dimension. For multifloor cases, except for the initialization of the window, the initial floor-set $\mathbf{f}_0$ contains all the floor numbers in the building, which are sorted from the lowest to the highest floor.

Algorithm 1: DRL for Hierarchical Indoor Localization

Data: A dataset containing RSSI values, labeled location $\mathcal{D}$:{$\mathbf{RSSI}_l, LOC_l$:$(x_l, y_l, z_l), or(x_l, y_l), or(x_l, y_l, f_l)$)}

Input: *environment parameters:* rad_{gt}, α, δ; *agent parameters:* γ, ϵ, $\mathcal{M}$

Initialize action-value function Q network with random parameters θ;

Initialize target action-value network $\hat{Q}$ network with parameters $\theta^- = \theta$;

for $i \leftarrow 0, \ldots, N$ **do**

  for *each data sample* $d_i = \{\mathbf{RSSI}_i, LOC_i\}$ *in* $\mathcal{D}$ **do**

    Initialize state $s_0 = \begin{cases} (\mathbf{RSSI}^i, \mathbf{cb}_0^i, \mathbf{h}_0^i) = (\mathbf{RSSI}^i, [(x_0^i, y_0^i, z_0^i), rad_0], \mathbf{h}_0^i) & \text{in 3D} \\ (\mathbf{RSSI}^i, \mathbf{w}_0^i, \mathbf{h}_0^i) = (\mathbf{RSSI}^i, [(x_0^i, y_0^i), rad_0], \mathbf{h}_0^i) & \text{in Single-Floor;} \\ (\mathbf{RSSI}^i, \mathbf{u}_0^i, \mathbf{h}_0^i) = (\mathbf{RSSI}^i, [(x_0^i, y_0^i, \mathbf{f}_0^i), rad_0], \mathbf{h}_0^i) & \text{in Multi-Floor} \end{cases}$

    for $t \leftarrow 0, \ldots$ **do**

      Select a random action a_t with probability ϵ, otherwise select $a_t = max_a Q^*(s_t, a_t; \theta)$;

      Execute action a_t as to get a reward r_t, new radius rad_{t+1}^i, and new location $\begin{cases} (x_{t+1}^i, y_{t+1}^i, z_{t+1}^i) & \text{in 3D} \\ (x_{t+1}^i, y_{t+1}^i) & \text{in Single-Floor;} \\ (x_{t+1}^i, y_{t+1}^i, \mathbf{f}_{t+1}^i) & \text{in Multi-Floor} \end{cases}$

      Update history vector $\mathbf{h}_{t+1}^i$ with the chosen action a_t;

      Transit from current state s_t to its next state $s_{t+1} = \begin{cases} (\mathbf{RSSI}^i, \mathbf{cb}_{t+1}^i, \mathbf{h}_{t+1}^i) & \text{in 3D} \\ (\mathbf{RSSI}^i, \mathbf{w}_{t+1}^i, \mathbf{h}_{t+1}^i) & \text{in Single-Floor;} \\ (\mathbf{RSSI}^i, \mathbf{u}_{t+1}^i, \mathbf{h}_{t+1}^i) & \text{in Multi-Floor} \end{cases}$

      Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory $\mathcal{M}$;

      Sample random mini batch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{M}$;

      Set $y_j = r_j + \gamma max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}; \theta^-)$;

      Calculate gradient descent according to Equation (10) and update θ by Adam [36] and Dropout [37];

      Every C steps reset $\hat{Q} = Q$, i.e., set $\theta^- = \theta$.

C. Our Learning Algorithm

Algorithm 1 depicts the model training steps. Each data sample in the training set corresponds to a target and contains all RSSIs that this target receives from the environment and its location coordinates. There is a prespecified parameter N, which is the maximal number of possible iterations. In each iteration, all data samples are used to update the NN (i.e., the Q-function). For each data sample, a reinforcement learning trial is performed consisting of multiple steps until the search is terminated either with $+\eta$ or $-\eta$. In each step, the agent selects an action a_t via exploration or exploitation, executes it to obtain a reward, and the system transits to the next state. Then, the history vector is updated, and the transition record is stored in the replay memory from which minibatches are sampled to train the NNs. To be self-contained, several existing techniques involved in our approach are described as follows.

Discounted Factor: In (9), there is a balance between the most immediate reward and the future return. The future return is discounted by a factor γ. In our experiments, we use $\gamma = 0.1$, which means we lean more toward the immediate reward in the balance.

Exploration–Exploitation: The ϵ-greedy policy [22] is used during the model training, which gradually shifts from exploration to exploitation according to the value of ϵ. For exploration, the agent selects random actions and observes the state changes, while for exploitation, the agent greedily selects the action that maximizes the current value function, and then learns from its own successes and mistakes. In our settings, the ϵ-greedy policy starts with $\epsilon = 1$, which means a random choice of action, and decreases to $\epsilon = 0$ with $\epsilon_{i+1} = 0.995 \times \epsilon_i$ at each iteration.

Experience Replay: It is proposed in [24] and [38], where the agent's experiences at each time step, the transition $m_t = (s_t, a_t, r_t, s_{t+1})$ is stored in an experience replay memory $\mathcal{M} = m_1, m_2, \ldots, m_N$. During each training stage, we perform the Q-learning updates using minibatches randomly sampled from the stored experiences, $m \sim \mathcal{M}$. In our settings, we use an experience replay of 100 experiences and a batch size of 50.

History Vector: As discussed in Section III-A, we capture all the actions for each data sample during each iteration in the search for the target. The total number of steps for the agent to find the target in each searching round depends on the initial cube size and the radius of the target cube. However, it can be difficult to use a history vector of arbitrary length as inputs to an NN. In our settings, we fix the length of the history to record at most ten recent actions for each target during each iteration, thus, $\mathbf{h} \in \mathbb{R}^{40}$ for single-floor localization and $\mathbf{h} \in \mathbb{R}^{80}$ for a 3-D or multifloor case. If the agent stops at a specific step $t < 10$, then the remaining entries of $\mathbf{h}$ will be filled with 0 s.

Environmental Parameters: In (2)–(5), the terminating reward η takes the value of 3, so the agent receives a reward of $+3$ when it successfully localizes the target, or a penalty of -3 when it moves away from the target, i.e., IoC or IoW is reduced. A correct action in the intermediate steps is rewarded by $\tau = 1$ when the action makes IoC or IoW larger. The threshold value δ is set to be 0.5.

TABLE I
DATA SET STATISTICS

	Single-Floor Datasets		Multi-Floor Datasets	
	IPIN	UJI_B1F1	UJI_B1	UTS
Number of WAPs	168	168	208	589
Number of RPs	57	60	238	1452
Number of Training Samples	1303	1187	4156	7286
Number of Test Samples	326	297	1040	1822
Number of Floors	1	1	4	16
Floor/Building Plan ($m \times m$)	30 × 5	150 × 170	150 × 174	104 × 32
Area Coverage (m^2)	150	25,500	104,400	44,000

V. EXPERIMENTS

A. Data Description

The proposed model is evaluated on the IPIN2016 Tutorial data set, UTSIndoorLoc data set [15], and UJIIndoorLoc data set [39]. The IPIN2016 Tutorial data set was collected in a small area of 30 m ×5 m (150 m^2) covering a corridor of the School of Engineering of the University of Alcala (Spain) on a single floor. The database comprised 927 training/reference records and 702 test ones, where 168 wireless APs (WAPs) were detected. The UTSIndoorLoc data set was collected in the FEIT Building at the University of Technology Sydney (UTS), which covered 16 floors, including three basement levels. The total area was approximately 44 000 m^2, and 9107 training samples and 387 test ones were provided with 589 different WAPs in total. The UJIIndoorLoc data set was collected in the real world, including three buildings with four or five floors by more than 20 users using 25 different models of mobile devices within several months. The data set included a surface area of 108 703 m^2 in Universitat Jaume I (UJI) and 19 937 training/reference records and 1111 test records. The number of different WAPs appearing in the database was 520.

We extract two sets of data from the UJI data set for experiments, respectively, in the single-floor and multifloor environments: data on Building 1 Floor 1 (B1F1) and data in Building 1 (B1) containing four floors. For every data set, we randomly split it into an offline-training set and an online-test set with a ratio of 80% versus 20%. Table I shows the statistics of our experimental data sets.

B. Online Test Evaluation

The output of our algorithm is a series of bounding regions, as shown in Figs. 6(b) or 7. For each test sample, the algorithm outputs a final predicted bounding region, described by its center coordinates and radius. The localization error is calculated as the Euclidean distance (in m) between the position recorded in the test sample and the center coordinates of the predicted region. The height of a single floor is assumed to be 4 m in the calculation of the prediction errors in the multifloor environment. We compute a cumulative distribution function (CDF) of the localization errors that occurred on all of the test samples, as shown on the figures in this section under different experimental settings.

We experiment with two choices of rad_{gt} : $\text{rad}_{\text{gt}} = 0.5$ m, and $\text{rad}_{\text{gt}} = 0.2$ m. Two settings are considered to evaluate the easy adaptation of the proposed algorithm to change in the resolution requirement. In our experiments shown in later sections, we examine the prediction error that our model achieves for a given percentage of targets under the two choices of rad_{gt}. For instance, in Table II (left-top corner), 50% of targets are localized with an error distance of 0.496 m. In such a table, we also examine (on the right half of the columns) the percentage of targets that is successfully localized within a given distance. For instance, in Table II (middle top area), 18.17% of targets are detected within 0.2 m distance error. A good policy often means that the agent makes the center of the predicted region fall into the target/ground-truth region recorded in $\mathbf{cb}_{\text{gt}}$.

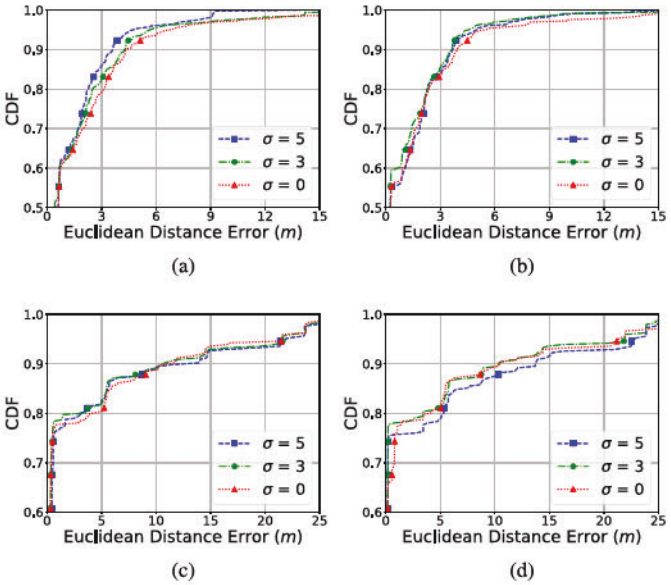


Fig. 9. CDF of Euclidean distance error (m) in single-floor localization under different dynamics with Gaussian noise ($\sigma = 0, 3, 5$). (a) $\text{rad}_{\text{gt}} = 0.5$ m @IPIN. (b) $\text{rad}_{\text{gt}} = 0.2$ m @IPIN. (c) $\text{rad}_{\text{gt}} = 0.5$ m @UJI_B1F1. (d) $\text{rad}_{\text{gt}} = 0.2$ m @UJI_B1F1.

The experiments described in Sections V-B1, V-B2, and V-B3 are performed to analyze the robustness of the proposed approach under different levels of environmental dynamics, evaluate the ability of on-demand resolution, and compare with other algorithms, respectively.

1) Robustness Analysis in Dynamic Environment: To elevate the level of dynamics in the environment, we inject noise into the input of the DRL network in every decision-making step. The quantitative effects of the dynamic environmental factors were analyzed in [8]. The measurement results demonstrated that the average vibrations on RSSI were approximately 8, 9, and 0.8 dB, respectively, for people, doors, and humidity. In our model, we generate the centered Gaussian noise $\mathcal{N}(0, \sigma^2)$ with the standard deviation $\sigma = 3$ and 5 to simulate approximately 3- and 5-dB variations of RSSIs caused by environmental uncertainty, and compare with the case without environmental uncertainty (i.e., 0 dB).

Single-Floor Localization: The results of the robustness analysis for single-floor localization are included in Fig. 9 and Tables II and III. For example, on the IPIN data set with $\text{rad}_{\text{gt}} = 0.5$ m shown in the upper right part of Table II, the model is capable of localizing 50.31%, 52.45%, or 48.16% of the targets within 0.5 m under different noise settings.

TABLE II
SINGLE-FLOOR (IPIN) LOCALIZATION ROBUSTNESS ANALYSIS UNDER DIFFERENT DYNAMICS WITH GAUSSIAN NOISE ($\sigma = 0, 3, 5$)

rad_{gt}	Noise	Percentile			Euclidean Distance Error (m)					
		50%	75%	90%	<0.2	<0.3	<0.5	<0.6	<1	<6
$rad_{gt} = 0.5m$	$\sigma = 0dB$	0.49677657	2.46988725	4.41344528	18.10%	38.65%	50.31%	50.31%	62.70%	93.56%
	$\sigma = 3dB$	0.40398364	2.14819502	4.29441885	**23.62%**	**44.17%**	**52.45%**	**52.45%**	**82.52%**	95.40%
	$\sigma = 5dB$	0.61460643	1.94809433	3.61571381	16.87%	36.20%	48.16%	48.47%	63.19%	**96.01%**
$rad_{gt} = 0.2m$	$\sigma = 0dB$	0.17674976	0.26386152	2.35289928	31.60%	55.21%	56.44%	56.44%	60.12%	95.40%
	$\sigma = 3dB$	**0.1594505**	**0.24478037**	2.28084253	**34.97%**	**59.51%**	**59.82%**	**59.82%**	**64.11%**	**96.93%**
	$\sigma = 5dB$	0.18386117	0.26386152	2.10968318	30.67%	55.21%	55.52%	55.52%	59.20%	96.01%

TABLE III
SINGLE-FLOOR (UJI_B1F1) LOCALIZATION ROBUSTNESS ANALYSIS UNDER DIFFERENT DYNAMICS WITH GAUSSIAN NOISE ($\sigma = 0, 3, 5$)

rad_{gt}	Noise	Percentile			Euclidean Distance Error (m)					
		75%	80%	90%	<0.2	<0.3	<0.5	<0.6	<1	<6
$rad_{gt} = 0.5m$	$\sigma = 0dB$	0.50971194	4.21280862	11.61790607	20.87%	55.55%	73.73%	76.76%	77.77%	85.52%
	$\sigma = 3dB$	**0.50971194**	**3.54798449**	**10.84133242**	21.21%	**56.22%**	**74.41%**	**78.11%**	**78.45%**	**87.20%**
	$\sigma = 5dB$	0.62458716	3.06216012	13.924585	27.94%	44.44%	68.68%	73.73%	76.43%	**86.53%**
$rad_{gt} = 0.2m$	$\sigma = 0dB$	0.814134	4.1205886	10.35558446	65.65%	65.99%	66.32%	68.68%	75.75%	86.53%
	$\sigma = 3dB$	**0.19734504**	**3.43704572**	**10.35558446**	**77.10%**	**77.77%**	**77.77%**	**77.77%**	**78.11%**	**86.53%**
	$\sigma = 5dB$	0.22569945	5.31054606	13.70288678	74.74%	75.42%	75.42%	75.42%	75.75%	83.83%

TABLE IV
MULTIFLOOR (UJI_B1) LOCALIZATION ROBUSTNESS ANALYSIS UNDER DIFFERENT DYNAMICS WITH GAUSSIAN NOISE ($\sigma = 0, 3, 5$)

rad_{gt}	Noise	Percentile			Euclidean Distance Error (m)					
		25%	50%	75%	<0.2	<0.3	<0.5	<0.6	<1	<6
$rad_{gt} = 0.5m$	$\sigma = 0dB$	0.24728036	0.34689871	0.7460704	17.79%	36.73%	66.73%	72.69%	75.58%	86.44%
	$\sigma = 3dB$	0.22736675	0.34229886	0.50655113	19.32%	39.81%	73.94%	78.17%	81.92%	89.71%
	$\sigma = 5dB$	0.2337569	0.34229886	0.50358052	19.23%	39.32%	**74.90%**	**80.67%**	**82.02%**	**90.09%**
$rad_{gt} = 0.2m$	$\sigma = 0dB$	0.10310226	0.15394343	0.22058083	72.50%	77.40%	77.69%	77.69%	78.07%	85.96%
	$\sigma = 3dB$	**0.10149658**	**0.14641456**	**0.18801542**	**76.44%**	**80.86%**	**81.15%**	**81.34%**	**82.11%**	**89.32%**
	$\sigma = 5dB$	0.1046026	0.15220485	0.20989673	73.75%	78.36%	78.56%	78.84%	79.52%	89.23%

TABLE V
MULTIFLOOR (UTS) LOCALIZATION ROBUSTNESS ANALYSIS UNDER DIFFERENT DYNAMICS WITH GAUSSIAN NOISE ($\sigma = 0, 3, 5$)

rad_{gt}	Noise	Percentile			Euclidean Distance Error (m)					
		80%	90%	95%	<0.2	<0.3	<0.5	<0.6	<1	<6
$rad_{gt} = 0.5m$	$\sigma = 0dB$	0.46869795	0.53696699	3.65700084	13.99%	35.72%	87.15%	90.66%	91.05%	96.65%
	$\sigma = 3dB$	0.46898214	0.54987317	6.03449167	13.72%	35.56%	86.77%	90.39%	90.39%	94.95%
	$\sigma = 5dB$	**0.46401539**	**0.52895119**	**3.42375144**	**14.27%**	**36.22%**	**87.59%**	**91.60%**	**91.87%**	**96.70%**
$rad_{gt} = 0.2m$	$\sigma = 0dB$	0.22237837	0.38948878	3.41343457	71.56%	89.62%	90.17%	90.55%	90.72%	97.20%
	$\sigma = 3dB$	**0.21908864**	**0.26263524**	4.11139834	**72.67%**	**90.99%**	**91.11%**	**91.11%**	**91.38%**	95.77%
	$\sigma = 5dB$	0.22292926	0.95692812	4.46285949	71.40%	89.24%	89.41%	89.52%	90.01%	96.10%

The results in the column for the 50% percentile confirm the performance where the targets are detected within 0.496, 0.403, and 0.614 m, respectively. Similarly, on the UJI_B1F1 data set in Table III, for instance, when $\text{rad}_{\text{gt}} = 0.2$ m, 75% of the targets are detected within around 0.2 m. The overall results show that our model can localize a significant portion of targets (50%–75%) within a very small distance, e.g., around 0.2–0.3 m. Furthermore, Fig. 9 and details in Tables II and III reveal that our model often performs better in a dynamic environment with a small Gaussian noise. For example, it performs the best when the noise is set to 3 dB with $\text{rad}_{\text{gt}} = 0.2$ m on the IPIN and UJI_B1F1 data sets.

Multifloor Localization: The results of the robustness analysis for multifloor localization are shown in Tables IV and V. The same conclusion as in single-floor localization can also be drawn in multifloor situations. For the UJI_B1 data set with $\text{rad}_{\text{gt}} = 0.5$ m and $\text{rad}_{\text{gt}} = 0.2$ m, respectively, 75% of targets are detected within 0.5 and 0.2 m. On the UTS data set, 80% of targets are localized within around 0.5 and 0.2 m in the two rad_{gt} settings under various environmental dynamics. Together with Fig. 10, we also observe that our method performs better in a dynamic environment with a Gaussian noise of $\sigma = 3$ or 5.

Figs. 9 and 10 show that the proposed method obtains higher localization accuracy with small environmental uncertainty ($\sigma = 3$ and 5). We believe that this robustness to environmental dynamics may be partially explained by the adaptivity of deep learning methodology to a dynamic environment. Moreover, sample perturbation is often treated as a data augmentation strategy in image processing and natural language processing [40], and commonly consists of injecting a small Gaussian noise to the samples. For instance, Moreno-Barea *et al.* [41] verified that adding noise to images could help convolutional networks learn robust image features. Effectiveness of noise injection in normalizing deep sequence

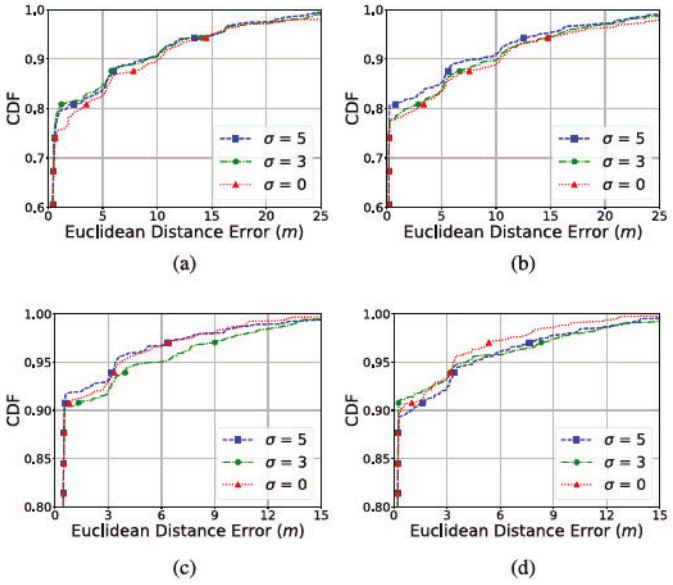


Fig. 10. CDF of Euclidean distance error (m) in multifloor localization under different dynamics with Gaussian noise ($\sigma = 0, 3, 5$). (a) $\text{rad}_{\text{gt}} = 0.5$ m @UJI_B1. (b) $\text{rad}_{\text{gt}} = 0.2$ m @UJI_B1. (c) $\text{rad}_{\text{gt}} = 0.5$ m @UTS. (d) $\text{rad}_{\text{gt}} = 0.2$ m @UTS.

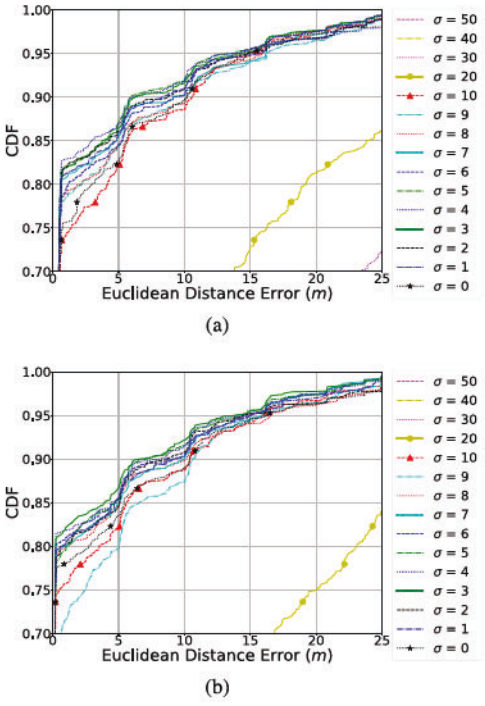


Fig. 11. CDF of Euclidean distance error (m) on UJI_B1 under a large variation of Gaussian noise ($\sigma = 0-50$). The most deviated curves correspond to the noise levels of 30 (in the top plot) and 20 dB (in the bottom plot). The curves for the noise levels of 40 and 50 dB are off the chart. (a) $\text{rad}_{\text{gt}} = 0.5$ m. (b) $\text{rad}_{\text{gt}} = 0.2$ m.

models is also illustrated in [42] and [43]. We believe that the environmental noise may bring out an effect of sample perturbation in the indoor localization problem.

We further use the largest UJI_B1 multifloor data set to examine to what degree the proposed algorithm can adapt to the Gaussian noise. Our experimental results are shown in Fig. 11, where the injected noise is randomly drawn from a

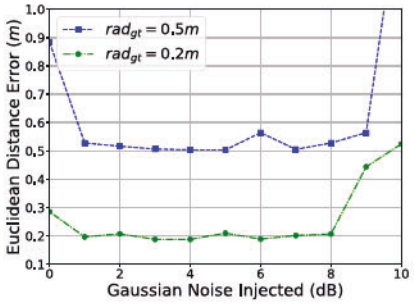


Fig. 12. Average Euclidean distance error (m) at 75% percentile versus noise level on UJI_B1.

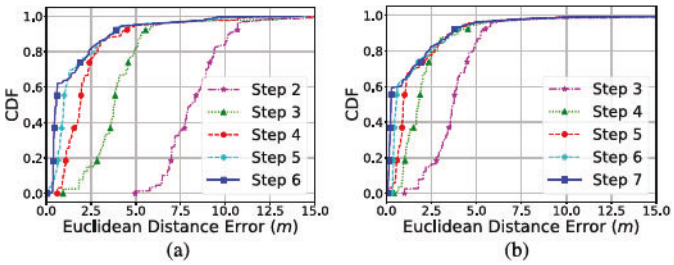
Gaussian distribution with a bandwidth σ, respectively, equal to 50, 40, 30, 20 dB, and then 10 and 9 dB, and all the way to 1 dB. We also include the performance curve when no noise is added (i.e., $\sigma = 0$ dB). It can be seen that the performance shows a large degradation when $\sigma \geq 10$ dB. The CDFs for the noise levels of 40 and 50 dB are invisible in the figure because the signal-to-noise ratio is no longer sensible for the agent to learn good detection steps. For smaller noise, we observe that when $\sigma \leq 8$ dB, the performance is actually improved over the case of $\sigma = 0$ dB. Particularly, the models with 3–5 dB noise have pretty consistently outperformed the others. To further investigate, we plot the localization distance error at a detection rate of 75% targets in Fig. 12, which again confirms that $\sigma = 3$–5 dB is a safe noise zone for robust localization.

2) On-Demand Resolution Analysis: We analyze the predictions in the last five steps of a searching round for two different localization resolutions at 0.5 m when $\text{rad}_{\text{gt}} = 0.5$ and 0.2 m when $\text{rad}_{\text{gt}} = 0.2$ m (equivalent to drawing small grids on a floor area with a side of 1 and 0.4 m, respectively). The last five steps when $\text{rad}_{\text{gt}} = 0.5$ m correspond to localization resolution of 0.5, 1, 2, 4, and 8 m, from the last step backward. If using a grid search method, it means that the floor area needs to be segmented into grid cells of side 1, 2, 4, 8, and 16 m separately. When $\text{rad}_{\text{gt}} = 0.2$ m, the model reaches a relatively finer resolution, where the last five steps correspond to a resolution of 0.2, 0.4, 0.8, 1.6, and 3.2 m, respectively.

Single-Floor Localization: As shown in Fig. 13, the trained models take six or seven steps to zoom into the target region on the IPIN data set to research the localization resolution of 0.5 or 0.2 m, respectively, when $\text{rad}_{\text{gt}} = 0.5$ m or $\text{rad}_{\text{gt}} = 0.2$ m, and eight or nine steps on the UJI_B1F1 data set. The CDFs under different settings in Fig. 13 also demonstrate that the localization resolution gets better as the steps move forward. Our approach can construct a model in one-shot training that can provide different localization resolutions.

For instance, in Fig. 13(a) and Table VI, on the IPIN data set, our experiments to evaluate the localization resolution at the last five steps show that the model can detect 50% of targets within 0.5, 1, 2, 4, and 8 m, which verifies the models' capability of on-demand localization.

Moreover, our model can achieve good performance much quickly than other compared models, such as KNN-based



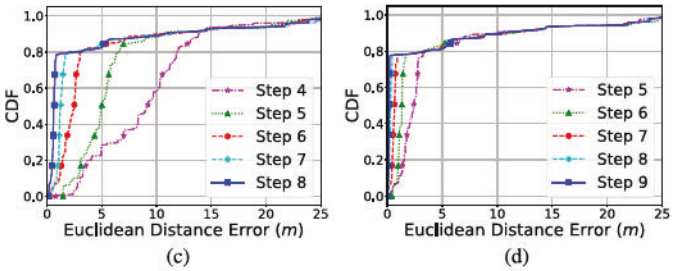
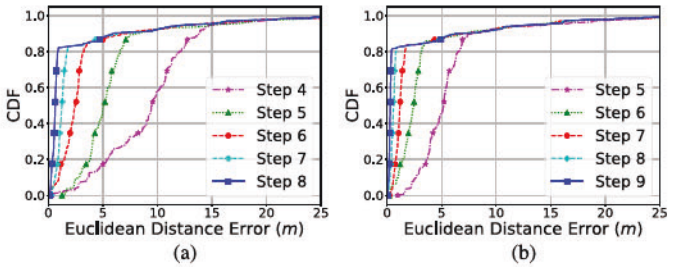


Fig. 13. CDF of Euclidean distance error (m) in single-floor localization on last five steps under $\text{rad}_{\text{gt}} = 0.5$ m and $\text{rad}_{\text{gt}} = 0.2$ m with $\sigma = 3$ or $\sigma = 5$. (a) $\text{rad}_{\text{gt}} = 0.5$ m, $\sigma = 5$ @IPIN. (b) $\text{rad}_{\text{gt}} = 0.2$ m, $\sigma = 3$ @IPIN. (c) $\text{rad}_{\text{gt}} = 0.5$ m, $\sigma = 3$ @UJI_B1F1. (d) $\text{rad}_{\text{gt}} = 0.2$ m, $\sigma = 3$ @UJI_B1F1.



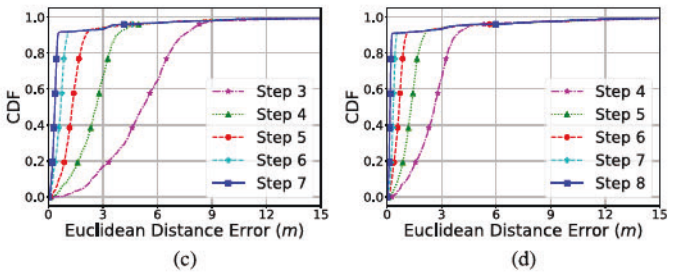


Fig. 14. CDF of Euclidean distance error (m) in multifloor localization on different steps under $\text{rad}_{\text{gt}} = 0.5$ m and $\text{rad}_{\text{gt}} = 0.2$ m with $\sigma = 3$ or $\sigma = 5$. (a) $\text{rad}_{\text{gt}} = 0.5$ m, $\sigma = 5$ @UJI_B1. (b) $\text{rad}_{\text{gt}} = 0.2$ m, $\sigma = 3$ @UJI_B1. (c) $\text{rad}_{\text{gt}} = 0.5$ m, $\sigma = 5$ @UTS. (d) $\text{rad}_{\text{gt}} = 0.2$ m, $\sigma = 3$ @UTS.

TABLE VI
SINGLE-FLOOR (IPIN) LOCALIZATION ON-DEMAND RESOLUTION ANALYSIS AT DIFFERENT STEPS WITHIN 50% PERCENTILE

rad_{gt}	Step 7	Step 6	Step 5	Step 4	Step 3	Step 2
$0.5m$		0.5709745	0.9796782	1.9078235	3.7955055	8.0753095
$0.2m$	0.2447803	0.5488228	0.9243887	1.7952863	3.7085324	

TABLE VII
SINGLE-FLOOR (UJI_B1F1) LOCALIZATION ON-DEMAND RESOLUTION ANALYSIS AT DIFFERENT STEPS WITHIN 75% PERCENTILE

rad_{gt}	Step 9	Step 8	Step 7	Step 6	Step 5	Step 4
$0.5m$		0.5097119	0.7573803	1.5893224	2.9220892	6.2025979
$0.2m$	0.1973450	0.4280092	0.897713	1.6195523	2.7930272	

TABLE VIII
MULTIFLOOR (UJI_B1) LOCALIZATION ON-DEMAND RESOLUTION ANALYSIS AT DIFFERENT STEPS WITHIN 75% PERCENTILE

rad_{gt}	Step 9	Step 8	Step 7	Step 6	Step 5	Step 4
$0.5m$		0.5035805	0.7946793	1.6014869	3.0229795	6.2581297
$0.2m$	0.1880154	0.3751067	0.7675248	1.58895	3.0577386	

TABLE IX
MULTIFLOOR (UTS) LOCALIZATION ON-DEMAND RESOLUTION ANALYSIS AT DIFFERENT STEPS WITHIN 80% PERCENTILE

rad_{gt}	Step 8	Step 7	Step 6	Step 5	Step 4	Step 3
$0.5m$		0.4633451	0.8791922	1.7448887	3.3489760	6.6680579
$0.2m$	0.2190886	0.4413257	0.880845	1.7211385	3.3249144	

and RF methods. As illustrated in Fig. 13(d) and Table VII, the experiment is conducted on the UJI_B1F1 data set when $\text{rad}_{\text{gt}} = 0.2$ m. In the final step, the model detects 75% of targets within a distance of 0.197 m, while at step 5, 75% of targets are already localized within 2.793 m. By cross-referring with Table XI, we see that the KNN-based model, which is the most effective method among all other methods, could only achieve a distance error of 4.882 m at 75% percentile. Our method reduces the localization error by 96% and 43% at the last step and step 5, respectively, from the KNN-based method.

Multifloor Localization: The same observation is obtained in multifloor localization as shown in Fig. 14 and Tables VIII and IX. For instance, on the UJI_B1 data set, 75% of targets are detected within a distance of 0.188 m at the last step and 3.058 m at step 5 when $\text{rad}_{\text{gt}} = 0.2$ m. Our method improves the localization resolution by 96% and 28% at the last step and step 5, respectively, compared with the KNN-based method (detection distance of 4.265 m as shown in Table XII).

We provide a further discussion. The total number of search steps in our method is affected by several factors, including the initial search space size rad_0 calculated in (11) that reflects the size of the building/floor of the indoor environment, the size of the target region rad_{gt}, and the threshold δ discussed in Section III-C1. Hence, the number of needed steps n_{step} can be estimated from rad_0 and rad_{gt}, due to the bisecting approach

$$n_{\text{step}} \approx \lceil \log \text{rad}_0 - \log \text{rag}_{\text{gt}} \rceil. \tag{12}$$

On the UJI_B1F1 data set, as illustrated in Figs. 13(c) and (d) and 14(a) and (b), and Tables VII and VIII, eight steps are taken when $\text{rad}_{\text{gt}} = 0.5$ m and nine steps when $\text{rad}_{\text{gt}} = 0.2$ m for both the single-floor and multifloor tasks. This is because the longest sides of the corresponding floor area and the building plan are very close: 170 m for the UJI_B1F1 and 174 m for the UJI_B1 (by referencing Table I), which brings similar rad_0 for the single-floor and multifloor tasks.

TABLE X
SINGLE-FLOOR (IPIN) LOCALIZATION COMPARISON OF EUCLIDEAN DISTANCE ERROR (M) UNDER DIFFERENT PERCENTILES AND PERCENTILES UNDER DIFFERENT EUCLIDEAN DISTANCE ERROR (M)

	Percentile				Euclidean Distance Error (m)					
	30%	40%	50%	75%	<0.2	<0.3	<0.5	<0.6	<1	<3
DRL(0.2m)	0.18386117	0.21940582	**0.24478037**	1.97171207	**34.97%**	**59.51%**	59.82%	59.82%	64.11%	85.27%
DRL(0.5m)	0.41519974	0.46419895	**0.57097453**	2.07068381	0.61%	0.61%	**47.54%**	**57.05%**	63.80%	86.19%
KNN	0.91664006	1.20000833	1.4093644	2.27844596	3.37%	4.91%	13.80%	15.95%	32.21%	87.73%
RandomF	0.76040055	1.00064042	1.3374128	2.12160435	4.60%	7.97%	14.72%	17.79%	39.88%	87.73%
SVM	1.01229955	1.53231672	1.9195283	3.10010672	8.28%	11.04%	13.80%	14.72%	29.45%	73.31%
Lasso	1.64481243	1.97755286	2.36423883	3.86548731	0%	0.92%	2.76%	3.99%	11.96%	62.88%
Ridge	1.60355575	2.03531899	2.3333411	3.85912371	0.31%	0.61%	2.76%	4.60%	11.96%	59.81%

TABLE XI
SINGLE-FLOOR (UJI_B1F1) LOCALIZATION COMPARISON OF EUCLIDEAN DISTANCE ERROR (M) UNDER DIFFERENT PERCENTILES AND PERCENTILES UNDER DIFFERENT EUCLIDEAN DISTANCE ERROR (M)

	Percentile				Euclidean Distance Error (m)					
	50%	60%	75%	80%	<0.2	<0.3	<0.5	<0.6	<1	<6
DRL(0.2m)	0.16686455	0.17959264	**0.19734504**	3.43704572	**77.10%**	**77.78%**	77.78%	77.78%	78.11%	86.53%
DRL(0.5m)	0.29794088	0.32200393	**0.50971194**	3.54798449	21.21%	56.22%	**74.41%**	**78.11%**	78.45%	87.20%
KNN	1.04017735	2.08299649	4.88217288	6.1027161	44.44%	45.45%	45.45%	46.46%	48.15%	78.79%
RandomF	2.10966344	3.0131779	5.20168333	6.54556976	28.62%	28.62%	28.96%	30.64%	35.01%	74.42%
SVM	8.43512502	12.0376808	15.32321058	16.97653589	5.39%	5.39%	5.72%	5.72%	7.07%	35.35%
Lasso	9.6792809	12.21386393	16.15703809	17.78602958	0%	0%	0.33%	0.33%	3.37%	27.61%
Ridge	10.13856399	12.27806632	16.80432547	18.30826735	0%	0%	0.33%	0.33%	3.03%	29.29%

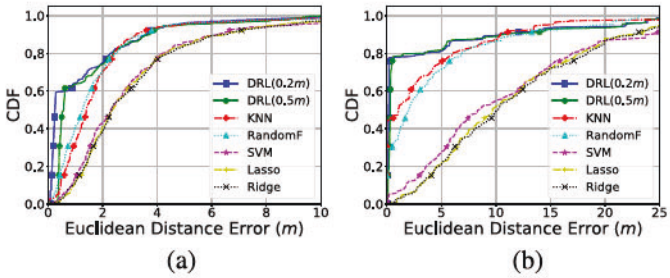


Fig. 15. Single-floor localization performance comparison with different algorithms. (a) @IPIN. (b) @UIJ_B1F1.

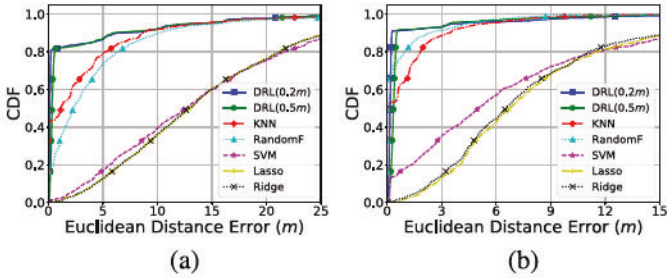


Fig. 16. Multifloor localization performance comparison with different algorithms. (a) @UJI_B1. (b) @UTS.

This suggests that no extra runtime cost will be introduced when making the online query in complex multifloor indoor environments to achieve the same resolution as in the single-floor case if the longest side remains the same length. The number of execution steps will not change even though the offline-survey sample size in the multifloor case is usually far larger than that in the single-floor case. Nevertheless, the time complexity of the brute-force neighbor search in the KNN-based algorithm is proportional to the size of the training data set for each test sample [44], which is not scalable with massive data in complex indoor environments.

3) Comparison With the State of the Art: We compare the proposed approach against five widely-used methods: the KNNs, RF, SVM, Lasso Regression, and Ridge Regression.

Single-Floor Localization: Our approach outperforms all other comparison algorithms in single-floor environments, both with $\text{rad}_{\text{gt}} = 0.5$ m and $\text{rad}_{\text{gt}} = 0.2$ m. For example, the results on the IPIN data set are shown in Fig. 15(a) and more detail is illustrated in Table X. Among the classic machine learning algorithms, the RF method excels the others, but our approach outperforms these algorithms significantly. Specifically, when $\text{rad}_{\text{gt}} = 0.2$ m, 50% of targets are detected with a distance that is 83%, 82%, 87%, 90%, and 89% closer than KNN, RF, SVM, Lasso, and Ridge, respectively. Similar observation is obtained on UJI_B1F1 data set as shown in Fig. 15(b) and Table XI.

Multifloor Localization: Two multifloor environments, the UJI_B1 with 4 floors and the UTS with 16 floors, are used in our experiments, and results are shown in Fig. 16 and Tables XII and XIII. Again, the same comparison results seen in single-floor localization are observed. Specifically, our approach on the UJI_B1 data set at the 75% detection rate with $\text{rad}_{\text{gt}} = 0.2$ m and 0.5 m outperforms the best one of the comparison methods by 95% and 88%, respectively. On the UTS data set, the detection distances at the 80% percentile are at least 77% and 51% better than that of other methods.

Note that there is an alternative training procedure for multi-floor environments: training single-floor models floor by floor. This approach is time consuming and wastes time for the floors on which the object does not reside. Furthermore, it may lose coherent floor-to-floor information. In contrast, our approach for multifloor indoor environments improves the model generalization ability.

TABLE XII
MULTIFLOOR (UJI_B1) LOCALIZATION COMPARISON OF EUCLIDEAN DISTANCE ERROR (M) UNDER DIFFERENT PERCENTILES AND PERCENTILES UNDER DIFFERENT EUCLIDEAN DISTANCE ERROR (M)

	Percentile				Euclidean Distance Error (m)					
	50%	60%	75%	80%	<0.2	<0.3	<0.5	<0.6	<1	<6
DRL(0.2m)	0.14641456	0.16322652	**0.18801542**	0.23267903	**76.44%**	**80.86%**	81.15%	81.34%	82.11%	89.32%
DRL(0.5m)	0.34229886	0.38622938	**0.50358052**	0.57766541	19.23%	39.32%	**74.90%**	**80.67%**	82.02%	90.09%
KNN	1.17250237	2.12247767	4.26518625	5.28064634	43.36%	43.46%	43.75%	44.04%	47.02%	82.88%
RandomF	2.24823372	3.10797416	5.03139127	6.21808848	20.19%	21.25%	23.36%	24.90%	31.06%	79.13%
SVM	16.54373206	19.35557075	24.00114485	26.4065361	0.77%	1.25%	1.44%	1.54%	2.02%	22.02%
Lasso	12.95332697	14.95938104	19.59771724	21.1767986	0%	0%	0.19%	0.38%	1.06%	17.50%
Ridge	12.88732559	15.01004171	19.59308972	21.28604442	0.09%	0.09%	0.19%	0.29%	0.96%	17.59%

TABLE XIII
MULTIFLOOR (UTS) LOCALIZATION COMPARISON OF EUCLIDEAN DISTANCE ERROR (M) UNDER DIFFERENT PERCENTILES AND UNDER DIFFERENT EUCLIDEAN DISTANCE ERROR (M)

	Percentile				Euclidean Distance Error (m)					
	60%	75%	80%	90%	<0.2	<0.3	<0.5	<0.6	<1	<6
DRL(0.2m)	0.180194796	0.20463872	**0.21908864**	0.26263524	**71.68%**	**90.39%**	90.45%	90.50%	91.16%	96.05%
DRL(0.5m)	0.387418993	0.44114424	**0.46401539**	0.52895119	14.27%	36.22%	**87.59%**	91.60%	91.88%	96.71%
KNN	0.628033777	1.48670229	1.82680632	3.41736164	52.85%	53.02%	54.17%	58.34%	64.92%	88.52%
RandomF	0	0.50315592	0.95507277	2.54147542	67.39%	70.96%	74.92%	75.85%	80.62%	91.44%
SVM	6.32712355	9.88741834	11.65529585	17.18357329	12.73%	13.67%	14.21%	15.31%	18.06%	35.34%
Lasso	7.87929378	10.24626683	11.53491626	15.87920593	0.16%	0.22%	0.38%	0.44%	2.03%	12.89%
Ridge	7.67207733	10.27303419	11.07382976	15.70234405	0.33%	0.60%	1.32%	1.64%	2.69%	14.54%

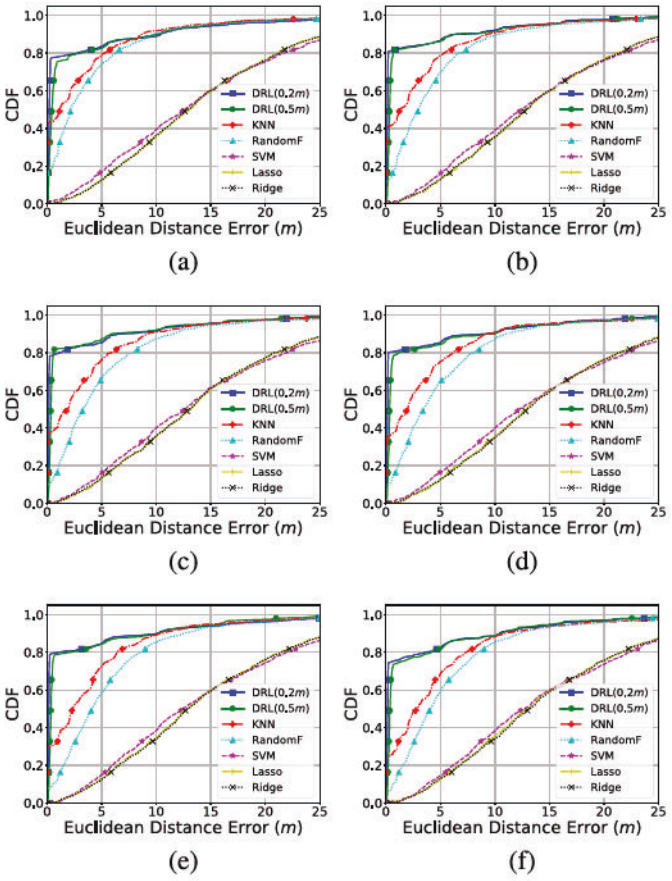


Fig. 17. UJI_B1 localization performance comparison with other algorithms under different dynamics with Gaussian Noise ($\sigma = 0, 3, 5, 6, 8, 10$).

We also evaluate how robustly the comparison methods react to the noise injection *during training* as shown in Fig. 17. Our models with a noise level under 10 dB perform either better than or closely to the noiseless case, while the performance of the baseline methods declines obviously as the injected noise level increases, especially for KNN and RF. Our method shows robustness in training performance likely also because the deep RL model is adaptive to dynamics which the other baseline localization models do not naturally possess.

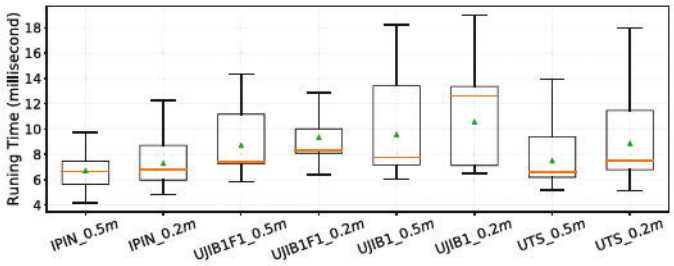


Fig. 18. Computational time of the testing phase.

4) Analysis of Computational Time: We compare the runtime [in milliseconds (ms)] between our bisection method and the early RL-based grid search method on a single GTX 1080Titan card in this section. The boxplots in Fig. 18 show the runtime statistics of our approach, including the median (the orange line), the mean (the green triangle), and the quantile information for each experimental setting when the trained models are deployed to search targets. Table XIV gives the detailed average runtime of each trial. In general, our model takes approximately 6–10 ms to localize a target in the environments ranging from the small single-floor plan, such as in IPIN to the large multifloor plan, such as in UJI_B1. Note that for each data set, the computational time under the setting of $\text{rad}_{\text{gt}} = 0.2$ m is on average greater than that of $\text{rad}_{\text{gt}} = 0.5$ m as illustrated in Table VI–IX in Section V-B2 because the agent takes one more step to reach 0.2 m from 0.5 m.

TABLE XIV
AVERAGE COMPUTATIONAL TIME NEEDED FOR LOCALIZING A TARGET

	IPIN_0.5m	IPIN_0.2m	UJI_B1F1_0.5m	UJI_B1F1_0.2m	UJI_B1_0.5m	UJI_B1_0.2m	UTS_0.5m	UTS_0.2m
(ms)	6.7	7.3	8.7	9.3	9.5	10.5	7.5	8.8

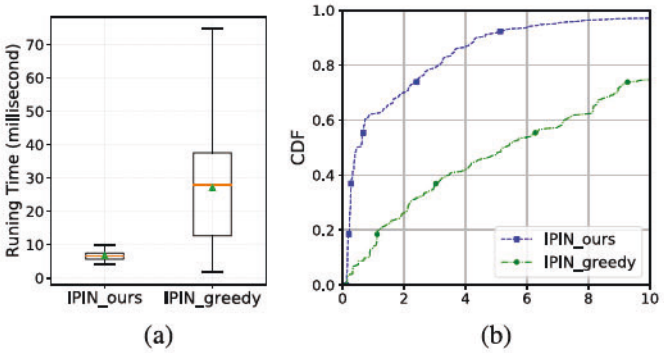


Fig. 19. Comparison of our algorithm with the greedy grid-by-grid search algorithm on the IPIN data set. (a) Computational time. (b) CDF of Euclidean distance error (m).

We employ the grid environment used in the early RL approach [18], where the agent learns to move from its initial location grid-by-grid to localize a target on a single floor. The search method in [18] divides the environment into 1 m $\times$1 m grids. We adopt the same NN structure in the greedy method of [18] and our method. We improve the search efficiency by allowing the agent to move toward the target greedily, so it does not get the worst case complexity for a fair comparison. Fig. 19(a) shows the runtime of the testing phase after both models are trained in the same number of epochs. As the figure shows, the proposed method's computational time is significantly less than the grid search method. The average detection time of our approach is 6.7 ms, while the grid search method is 27.1 ms. This observation confirms the time efficiency of the proposed bisection method. Our algorithm shows a more stable runtime ranging from 4 to 10 ms because the number of searching steps is fixed to six before reaching the predefined precision. However, the grid search method may require as long as 74 ms due to the large variation in the steps needed to move from the beginning position to the target location.

Furthermore, we evaluate the accuracy of the greedy grid search method and compare it with that of the bisection algorithm in Fig. 19(b). After the agents of the two methods are trained with the same number of epochs, the bisecting method outperforms significantly in terms of the test accuracy, which might be partially due to the difficulty in training the grid search model. Theoretically, the greedy search with grids can reach $\mathcal{O}(N)$ time complexity in 2-D search space, but it is still exponentially longer than ours in $\mathcal{O}(\log N)$ time.

We experiment with the grid search on the smaller IPIN data set because the method can be very challenging and time consuming to train and test in other more complex settings. In contrast, there is only a slight increase in computational time of the bisection method in large and complex multifloor environments, such as in the UJI and UTS data sets as shown in Fig. 18 and Table XIV.

VI. CONCLUSION

We have developed an effective hierarchical model that unifies both single-floor and multifloor indoor localization based on DRL, by formatting the problem as an MDP other than the traditional classification or regression problem. It detects the location of a target by consecutively bisecting the search space to a small cube or window that significantly overlaps with the target. This approach runs faster than existing indoor localization methods, even including the recent ones that also employ DRL. The fast speed of this approach renders it the potential for real-time tracking of an object. Furthermore, it does not require any prior knowledge about the search space, such as floor plans or any beforehand partitioning of the search space. The DRL framework allows the model to automatically adapt to environmental dynamics causing the variation of RSSIs, and the impact of the environmental dynamics on the model learning is worthy of further exploration in the future. Although the proposed approach has been implemented based on WiFi RSSI values, it can be applied to any other suitable signals to localize objects.

There are several directions to extend this work. Although the basic Q-learning method with networks of two to three layers in our current implementation already shows superior empirical performance, applying more advanced deep learning techniques may further improve the model (e.g., speed up or stabilize training, or improve accuracy). We have tested the proposed approach on three data sets in both single-floor and multifloor settings. We hope that truly 3-D indoor data sets (continuous in the vertical direction) will be available in the near future, which may be the best scenario to demonstrate the advantages of this approach.

ACKNOWLEDGMENT

The authors' code is available at https://github.com/FayeDou/Reinforcement-Learning-Indoor-Localization.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.
[2] F. Zafari, A. Gkelias, and K. K. Leung, "A survey of indoor localization systems and technologies," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 3, pp. 2568–2599, 3rd Quart., 2019.
[3] S. He and S.-H. G. Chan, "Wi-Fi fingerprint-based indoor positioning: Recent advances and comparisons," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 466–490, 1st Quart., 2016.
[4] J. Luo, L. Fan, and H. Li, "Indoor positioning systems based on visible light communication: State of the art," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2871–2893, 4th Quart., 2017.
[5] 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, no. 8, p. 1806, 2017.
[6] M. Ridolfi *et al.*, "Experimental evaluation of UWB indoor positioning for sport postures," *Sensors*, vol. 18, no. 1, p. 168, 2018.

[7] S. Yiu, M. Dashti, H. Claussen, and F. Perez-Cruz, "Wireless RSSI fingerprinting localization," *Signal Process.*, vol. 131, pp. 235–244, Feb. 2017.

[8] Y.-C. Chen, J.-R. Chiang, H.-H. Chu, P. Huang, and A. W. Tsui, "Sensor-assisted Wi-Fi indoor location system for adapting to environmental dynamics," in *Proc. 8th ACM Int. Symp. Model. Anal. Simulat. Wireless Mobile Syst.*, 2005, pp. 118–125.

[9] Z. Yang, C. Wu, and Y. Liu, "Locating in fingerprint space: Wireless indoor localization with little human intervention," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 269–280.

[10] P. Xiang, P. Ji, and D. Zhang, "Enhance RSS-based indoor localization accuracy by leveraging environmental physical features," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–8, Jan. 2018.

[11] D. A. Tran and C. Pham, "Fast and accurate indoor localization based on spatially hierarchical classification," in *Proc. IEEE 11th Int. Conf. Mobile Ad Hoc Sens. Syst. (MASS)*, Philadelphia, PA, USA, 2014, pp. 118–126.

[12] Y. Wang, C. Xiu, X. Zhang, and D. Yang, "WiFi indoor localization with CSI fingerprinting-based random forest," *Sensors*, vol. 18, no. 9, p. 2869, 2018.

[13] W. Zhang, K. Liu, W. Zhang, Y. Zhang, and J. Gu, "Deep neural networks for wireless localization in indoor and outdoor environments," *Neurocomputing*, vol. 194, pp. 279–287, Jun. 2016.

[14] J. Luo, Z. Zhang, C. Wang, C. Liu, and D. Xiao, "Indoor multifloor localization method based on WiFi fingerprints and LDA," *IEEE Trans. Ind. Informat.*, vol. 15, no. 9, pp. 5225–5234, Sep. 2019.

[15] X. Song *et al.*, "A novel convolutional neural network based indoor localization framework with WiFi fingerprinting," *IEEE Access*, vol. 7, pp. 110698–110709, 2019.

[16] K. S. Kim, S. Lee, and K. Huang, "A scalable deep neural network architecture for multi-building and multi-floor indoor localization based on Wi-Fi fingerprinting," *Big Data Anal.*, vol. 3, no. 1, p. 4, 2018.

[17] K. A. Nguyen, "A performance guaranteed indoor positioning system using conformal prediction and the WiFi signal strength," *J. Inf. Telecommun.*, vol. 1, no. 1, pp. 41–65, 2017.

[18] M. Mohammadi, A. Al-Fuqaha, M. Guizani, and J.-S. Oh, "Semisupervised deep reinforcement learning in support of IoT and smart city services," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 624–635, Apr. 2018.

[19] Y. Li, X. Hu, Y. Zhuang, Z. Gao, P. Zhang, and N. El-Sheimy, "Deep reinforcement learning (DRL): Another perspective for unsupervised wireless localization," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6279–6287, Jul. 2020.

[20] F. Dou, J. Lu, Z. Wang, X. Xiao, J. Bi, and C.-H. Huang, "Top-down indoor localization with Wi-Fi fingerprints using deep Q-network," in *Proc. IEEE 15th Int. Conf. Mobile Ad Hoc Sens. Syst. (MASS)*, Chengdu, China, 2018, pp. 166–174.

[21] M. Nowicki and J. Wietrzykowski, "Low-effort place recognition with WiFi fingerprints using deep learning," in *Proc. Int. Conf. Autom.*, 2017, pp. 575–584.

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, vol. 1. Cambridge, MA, USA: MIT Press, 1998.

[23] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Singapore, 2017, pp. 2786–2793.

[24] V. Mnih *et al.*, "Playing atari with deep reinforcement learning," in *Proc. Deep Learn. Neural Inf. Process. Syst. Workshop*, 2013.

[25] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[27] M. Bellver, X. Giró-i Nieto, F. Marqués, and J. Torres, "Hierarchical object detection with deep reinforcement learning," in *Proc. Deep Reinforcement Learn. Workshop (NIPS)*, 2016.

[28] J. C. Caicedo and S. Lazebnik, "Active object localization with deep reinforcement learning," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, Santiago, Chile, 2015, pp. 2488–2496.

[29] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif.Intell.*, vol. 2. Phoenix, AZ, USA, 2016, pp. 2094–2100.

[30] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2015.

[31] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

[32] T. Hester *et al.*, "Deep Q-learning from demonstrations," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3223–3230.

[33] M. Fortunato *et al.*, "Noisy networks for exploration," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[34] O. Anschel, N. Baram, and N. Shimkin, "Averaged-DQN: Variance reduction and stabilization for deep reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn. Vol. 70*, 2017, pp. 176–185.

[35] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3215–3222.

[36] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: arXiv:1412.6980.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[38] L.-J. Lin, "Reinforcement learning for robots using neural networks," Dept. School Comput. Sci., Carnegie-Mellon Univ., Pittsburgh PA, USA, Rep. CMU-CS-93-103, 1993.

[39] J. Torres-Sospedra *et al.*, "UJIIndoorLoc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems," in *Proc. Int. Conf. Indoor Position. Indoor Navig. (IPIN)*, Busan, South Korea, 2014, pp. 261–270.

[40] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *J. Big Data*, vol. 6, no. 1, p. 60, 2019.

[41] F. J. Moreno-Barea, F. Strazzera, J. M. Jerez, D. Urda, and L. Franco, "Forward noise adjustment scheme for data augmentation," in *Proc. IEEE Symp. Series Comput. Intell. (SSCI)*, Bangalore, India, 2018, pp. 728–734.

[42] H.-Y. Kim, Y.-H. Roh, and Y.-G. Kim, "Data augmentation by data noising for open-vocabulary slots in spoken language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguist. Student Res. Workshop*, 2019, pp. 97–102.

[43] Y. Cheng, Z. Tu, F. Meng, J. Zhai, and Y. Liu, "Towards robust neural machine translation," 2018. [Online]. Available: arXiv:1805.06130.

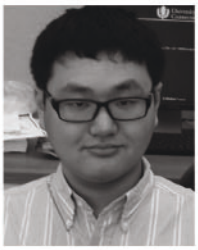
[44] Z. Deng, X. Zhu, D. Cheng, M. Zong, and S. Zhang, "Efficient kNN classification algorithm for big data," *Neurocomputing*, vol. 195, pp. 143–148, Jun. 2016.

Fei Dou (Member, IEEE) received the M.S. degree in electrical engineering from Tianjin University, Tianjin, China, in 2014. She is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of Connecticut, Storrs, CT, USA.

She is currently with the Laboratory of Machine Learning and Health Informatics, University of Connecticut. Her research interests include indoor localization, deep learning, object detection, and deep reinforcement learning.

Jin Lu received the Ph.D. degree in computer science and engineering from the University of Connecticut, Storrs, CT, USA, in 2019.

He is an Assistant Professor with the Department of Computer and Information Science, University of Michigan, Dearborn, MI, USA, where he is a Faculty Member of Precision Health and also affiliated with the Michigan Institute for Data Science. His major research interests include machine learning, data mining, optimization, matrix analysis, deep learning, biomedical informatics, and health informatics.



Tingyang Xu received the Ph.D. degree from the University of Connecticut, Storrs, CT, USA, in 2017.

In 2017, he joined Tencent AI Lab, Tencent Inc., Shenzhen, China, where he is a Senior Researcher with Machine Learning Center. He has published in data mining and machine learning top conferences, e.g., KDD, WWW, NeurIPS, ICLR, CVPR, and ICML. Since then, he has been working on deep graph learning and applying the deep graph learning model to various applications, such as indoor localization, molecular generation, and rumor detection.

Dr. Xu has served as a Reviewer for ICML, NeurIPS, KDD, WWW, AAAI, and scientific journals such as IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING.

Chun-Hsi Huang received the B.S. degree in computer science and engineering from the National Chiao-Tung University, Hsinchu, Taiwan, in 1989, the M.S. degree in computer science and engineering from the University of Southern California at Los Angeles, Los Angeles, CA, USA, in 1994, and the Ph.D. degree in computer science and engineering from the State University of New York at Buffalo, Buffalo, NY, USA, in 2001.

He is currently a Professor and the Director of the School of Computing, Southern Illinois University, Carbondale, IL, USA. His research interests are in the areas of extreme-scale computing and data analytics, computational biology and life-science informatics, as well as combinatorial algorithms and experimental algorithmics.

Jinbo Bi received the B.S. degree in mathematics and the M.S. degree in automation from Beijing Institute of Technology, Beijing, China, in 1996 and 1999, respectively, and the Ph.D. degree in mathematics from Rensselaer Polytechnic Institute, Troy, NY, USA, in 2003.

She is currently a Frederick H. Leonhardt Chair Professor of Computer Science and Engineering with the University of Connecticut, Storrs, CT, USA. Prior to her current appointment, she worked with Siemens Health, Malvern, PA, USA, on computer aided diagnosis research from 2003 to 2009, and Massachusetts General Hospital, Boston, MA, USA, on clinical decision support systems from 2009 to 2010. Her research interests include machine learning, artificial intelligence, computer vision, bioinformatics, biomedical informatics, and drug discovery.