ArcIoT: Enabling Intuitive Device Control in the Internet of Things through Augmented Reality

Jie Hua†, Sangsu Lee†, Gruia-Catalin Roman‡, Christine Julien†
†Department of Electrical and Computer Engineering, University of Texas at Austin,
{mich94hj, sethlee, c.julien}@utexas.edu
‡Department of Computer Science, University of New Mexico, gcroman@unm.edu

***Abstract*—Increasing numbers of smart devices have enriched the possibilities of the IoT. These advances also impose significant cognitive overhead on the end-users; to interact with IoT devices, a user must be knowledgeable about the devices, the applications, the network, etc. We present ArcIoT, a system that enables intuitive interaction with IoT devices by leveraging Augmented Reality (AR) and a novel relative indoor localization service. In ArcIoT, the user points a smartphone camera towards a device to control, chooses the device by a tap on the screen, and interacts with the displayed interface. Continuously and transparently, ArcIoT maintains a map of devices in the environment, which allows it to quickly and reliably control them. Compared to existing solutions, ArcIoT does not need predefined markers to recognize a device or require any change to physical infrastructure. ArcIoT is also robust to changes in the locations of the IoT devices in the environment. We successfully deployed ArcIoT in different home environments and evaluated its performance to demonstrate that ArcIoT helps users control devices in everyday environments accurately, responsively and intuitively.**

***Index Terms*—human computer interaction, intelligent environments, augmented reality, indoor localization**

I. Introduction

A lack of intuitive interactions with IoT devices has impeded the widespread adoption of smart spaces. While it is difficult to measure intuitiveness, one can view it as the inverse of how conscious a user is of the underlying technology. To create a smart home today, a user must manually identify and integrate a device, switch between apps to control devices from different manufacturers, and explicitly name devices or choose from long lists of available devices, all of which expose the user to underlying implementation details. Beyond smart homes, humans encounter IoT devices in many spaces, including public ones. Because users have little knowledge of these spaces, they may not even know which apps or naming conventions to use, which makes it even harder to interact.

We propose an **A**ugmented **R**eality **C**ontrol system for the **IoT**, ArcIoT, that allows a user to identify, access and control individual IoT devices in indoor spaces like homes and offices, even when those spaces contain visually identical devices. ArcIoT's approach is depicted in Fig 1. Once ArcIoT has matched the device on the screen to a unique device in the space, the user can tap the image and interact with the device. In ArcIoT, device controllers can be for a generic device type, for a specific device, or made specific to a particular user or situation, based on user preferences or a space's imposed constraints. For instance, a generic light controller is a simple

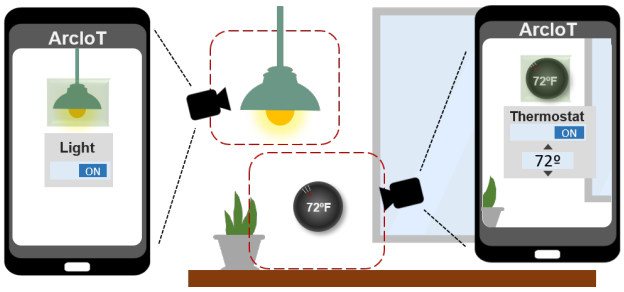


Fig. 1: An ArcIoT user opens an AR "window" to discover nearby IoT devices and their diverse control interfaces.

on/off switch, while a specific controller could allow the user to change the light color. A smart speaker might allow the user to adjust the volume or, depending on the user or time of day, enable changing the audio stream that is playing.

While existing work on computer vision enables identifying and tracking objects from the view of a camera [1], it alone cannot distinguish visually identical devices. ArcIoT's key innovation is the use of *context* to distinguish them. Intuitively, if we know the locations of the devices in the environment and the user's location and heading, we can use geometric techniques to identify a particular device in the camera view. Unfortunately, collecting and maintaining reliable locations in an everyday smart environment is non-trivial. To address this, we construct a relative localization system using beacons sent by the IoT devices combined with data from the smartphone's sensors. To make localization robust to noisy sensor data, ArcIoT incorporates feedback from the user after each interaction. The user may explicitly inform the system that it selected the wrong device or may implicitly provide feedback by accepting the system's selection. ArcIoT uses this feedback to adjust its estimations of the IoT devices' locations.

Our key research contributions are:

- ArcIoT, a mobile AR system enabling intuitive human interaction without requiring onerous setup processes;
- a novel *human-in-the-loop* localization scheme that uses intermittent real-time user feedback to build a map of the relative locations of the devices and the user;
- algorithms to rapidly adapt to changes in the layout of the devices without explicit user intervention; and
- a demonstration that it is feasible to implement ArcIoT on off-the shelf IoT devices in a way that provides responsive user interactions (i.e., with a latency of less than 100ms).

II. Related Work

ArcIoT addresses situations where a user interacts with a nearby visible IoT device. This is in contrast to efforts to completely automate behaviors in smart spaces [2], or in which the user controls a device from another location [3].

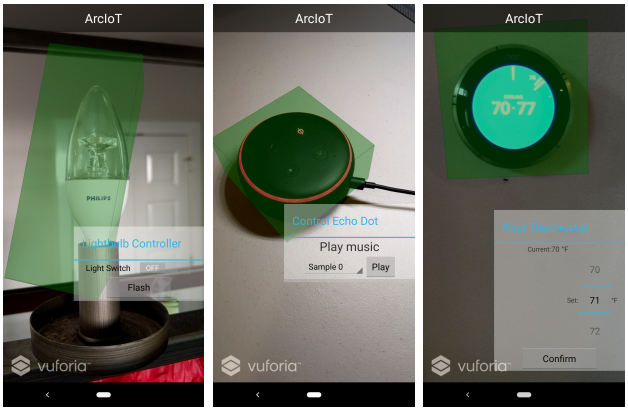
Existing commercial systems (e.g., from Amazon, Apple, or Google) bring diverse devices under the control of a single third-party, while middleware research [4] supports diverse devices under a single programming interface. These approaches all unify the interface for controlling the devices, but they do not address the problem of making interactions more intuitive. That is, they still require identifying devices by name and navigating many menus or voice controls.

A common way to identify and track objects in a camera view is using 3-D object recognition, often relying on deep learning trained on massive datasets [5], [6]. However, obtaining suitable training datasets with associated ground truth is difficult. In contrast, we enable fast identification based on key features of a modeled object [7]. When combined, computer vision and AR make it possible to recognize real-world objects and augment them with digital control overlays. However, an essential problem remains unresolved: it is common to encounter devices whose visual representations are impossible to distinguish. For instance, a single smart home may have tens of identical lights. It is not sufficient to correctly label a light as *a* light; to control it, one must identify *the* specific light that is in the camera frame.

Similar needs pervade other applications, e.g., browsing in the web of things [8]. Recent breakthroughs can identify everyday objects [9], but only visually different objects can be identified uniquely. Early AR systems for smart spaces either rely on preset devices [10] or markers. To uniquely identify devices, explicit visual markers [11]–[13] have been used, but they can be distracting and they make the user aware of the technology used for identification.

Another way to differentiate visually identical devices is to use additional context, and one obvious piece of context is location. Generating precise absolute location indoors at the granularity needed to differentiate IoT devices is difficult and often relies on customized hardware [14]. Other approaches attempt to fingerprint the wireless environment to establish location [15], [16]; these approaches require intensive training and are not resilient to changes in the environment. Some efforts have sought to supplement radio maps with additional information, for instance from a smartphone's inertial motion sensors [17]. Similar efforts seek to turn smartphones into universal remotes, relying on inertial sensors and radio signals to create a map of the space [18], though these approaches have significant deployment limitations.

Motivated by these work, and coupled with the observation that relative location between the user and the devices is sufficient, we model our problem as one of Simultaneous Localization and Mapping (SLAM) [19], [20]. SLAM usually relies on a *control signal* from a robot's motor and *observations* from a robot's "eyes" to track obstacles in the environment. In our re-



(a) (b) (c)

Fig. 2: Screenshots of ArcIoT after ArcIoT has identified the device and displayed its controls. (a) The user can turn a light on and off; (b) send a media file to a speaker; or (c) adjust a thermostat set point.

envisioning, the signal from the smartphone's motion sensors serves as the control signal and Bluetooth Low Energy (BLE) beacons from IoT devices are the observations. The benefit of this abstraction is that we can directly leverage existing algorithms [21]–[23] to improve accuracy and stability.

III. The ArcIoT System

ArcIoT supports a user interacting with nearby IoT devices. To interact with a device, the user frames it in the camera view. ArcIoT recognizes a device as a member of a *product family*, using an internal database of scanned objects. For instance, ArcIoT may recognize a light bulb, a smart speaker, or a thermostat, as shown in Fig. 2. IoT devices announce their presence via wireless *beacons* detected by the smartphone. These beacons serve two purposes: they aid in uniquely identifying a device and they carry device-specific information that ArcIoT uses to tailor interactions with the device.

Continuously and in the background, ArcIoT uses the content of device beacons, the received signal strength indicator (RSSI) values of those beacons, and the smartphone's onboard sensors (e.g., internal motion unit (IMU)) to maintain an internal coordinate system that maps all nearby IoT devices relative to the smartphone. When ArcIoT recognizes a device in the camera view, it uses this map to determine which device the user most likely wants to interact with. For simplicity, ArcIoT uses a 2-D coordinate system that is bootstrapped using an *origin*, which is defined as the location where the user first starts interacting with ArcIoT.

ArcIoT consists of two modules: the augmented reality (AR) module and the localization service (Fig. 3). The AR module is the interaction point with the user. It recognizes devices on the smartphone's screen and displays the corresponding user interfaces when the user taps to interact. The localization service tracks the locations of the user and devices.

Assumptions. To identify an IoT device in the smartphone's camera view, ArcIoT uses information about the physical appearance of the product. We assume this is provided by the manufacturer in the form of a 3-D model. To enable ArcIoT to

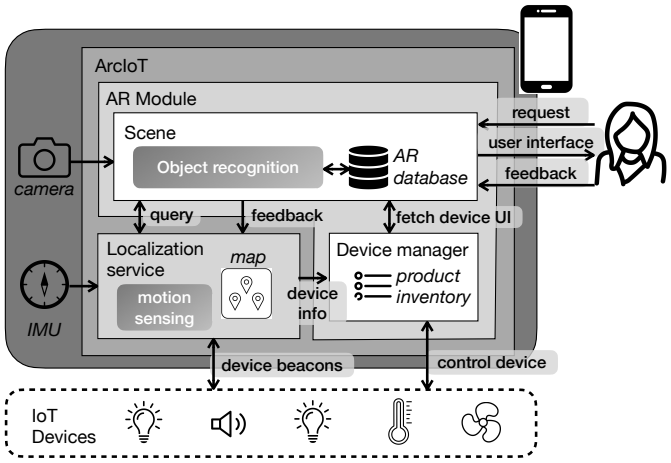


Fig. 3: ArcIoT system architecture. The user interacts with the AR module, which relies on the localization service to find devices.

discover IoT devices in the surroundings, we assume that each device periodically transmits device discovery beacons. In our implementation, we rely on Bluetooth beacons. We assume that the smartphone can scan for and interpret these beacons and is equipped with an inertial motion unit (IMU).

In ArcIoT, we assume that the location and heading of the smartphone are the location and heading of the user. The IoT devices may be moved between interactions, but we assume each device is stationary *during* a single interaction. Finally, for interactions in public spaces, it will be necessary to enforce authentication and authorization. These needs are orthogonal to the contributions of this paper; we assume the user is already authenticated with all available devices. This could be accomplished, for example, by having the user's smartphone include a validation key with every request.

Augmented Reality Module: Scene. We use the Vuforia SDK [7] to build an AR object recognition pipeline. We scan the IoT products used in ArcIoT and store the extracted feature points in the *AR database* in Fig. 3. When an object on screen matches a stored target, the AR module generates a report that contains the product name and the pose of the object relative to the smartphone's camera. The product name is used to retrieve a user interface from the device manager. The pose represents the transformation from the object's coordinate system to the internal coordinate system created by ArcIoT's localization service. A transformation vector $v_t = (x, y, z)$ indicates that the tracked device is displaced below the camera by the value x, to the right of the camera by the value y, and is estimated to be $-z$ distance in front of the camera. The localization service matches the current location and heading of the user to the locations of the known devices to find a device with the highest likelihood of being the desired device.

As shown in Fig. 2, the AR scene draws a bounding box around the tracked device to highlight it to the user. When the user taps on the object, ArcIoT overlays a control interface that is provided by the device manager. If ArcIoT identifies the incorrect object, the user can provide feedback through the AR module. This information is used to update the relative position information for the IoT device. In our current implementation, this feedback is collected either explicitly or not at all; in the future, we could also incorporate implicit feedback, noting positive reinforcement if the user simply accepts the provided interface and proceeds and negative reinforcement if the user immediately retries the same query.

Augmented Reality Module: Device Manager. ArcIoT's device manager stores information about the devices and manages interactions with them. The product inventory stores a unique identifier and a user interface template for each IoT device. Each product has a base functionality shared among all products of the same type. These base capabilities can be augmented for a specific device using data carried in the device's beacon. Using this information, the Device Manager renders the device user interface, and hands it to the Scene to be displayed. As an example, a product of type "smart light" may be configured with an on/off switch, but a specific device may add a dimming capability.

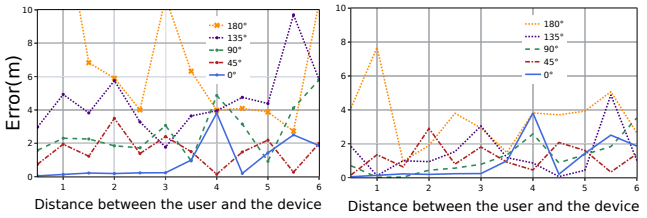
Because ArcIoT encapsulates concerns associated with device capabilities and behaviors within the Device Manager, adding or updating a product does not require knowledge of other parts of ArcIoT. To add a new product, an application developer simply needs to add a product class to the product inventory and implement the mechanism to render the device's UI. The developer does not need to be concerned with how the device is discovered, identified, or accessed. This also makes ArcIoT a convenient platform for manufacturers to push new features to users without installing or updating a separate app.

Interface to the Localization Service. ArcIoT's localization service tracks the relative locations of the user and the IoT devices. ArcIoT's localization service continuously collects beacons, processes their RSSI values, and maintains estimates of the device locations on a relative 2-D coordinate system. The localization service's map indicates, by device identifier, where each device is estimated to be located, relative to the user's position. To respond to a query, the localization service uses the estimated locations of known devices, the location and heading of the user (u_x, u_y, δ), and the translation vector $v_t = (x, y, z)$ provided by the AR Module's Scene component. We compute the location of the observed device, (o_x, o_y) as: $(u_x - z \cdot \cos(\delta) + y \cdot \sin(\delta), u_x - z \cdot \sin(\delta) - y \cdot \cos(\delta))$.

The next step is to determine which of the known devices is most likely the one at (o_x, o_y). A simple solution is to choose the device with the minimal distance to the observed location. However, this approach does not account for the heading of the user and could incorrectly choose a device behind the user even though that device is not likely to appear on the screen. We penalize the devices estimated to be behind the user with a back-turn factor. The corrected distance is: $(2 - \cos(\theta/2)) \times$ (uncorrected distance), where the *heading angle*, θ, captures the angle between the heading of the user and the vector from the user's location to the estimated location of each device.

IV. Human-in-loop Localization Framework

In theory, any indoor localization system can be integrated into ArcIoT. However, in the spirit of decreasing user overhead



(a) Ordinary RSSI estimate (b) RSSI estimate after correction

Fig. 4: Error of the ranging based on RSSI for different angles between the user and the transmitting IoT device before (a) and after (b) accounting for human body shadowing.

and increasing intuitiveness, we implement the localization framework without using any information about the floor plan or requiring any specialized infrastructure to aid localization. Thus, ArcIoT can be quickly and easily deployed in any new space, relying exclusively on the user's smartphone and the IoT devices themselves. Anytime an ArcIoT smartphone enters a new space, it can map that new space, using the beacons received from the IoT devices in the space. Our approach builds on the FastSLAM algorithm [20].

A. Background: FastSLAM and Motion Detection

Simultaneous Localization and Mapping (SLAM) is an approach for measuring movement and updating a map in an unknown environment. In SLAM, a robot's location is commonly sampled from its motor control signal. In ArcIoT, we instead rely on data from the smartphone's on-board inertial motion unit (IMU), which contains a geomagnetic field sensor, an accelerometer, and a gyroscope. We derive the heading of the user from the geomagnetic field sensor and use the accelerometer and gyroscope to detect the user's steps. When we detect a step, we update the user's location as one iteration in FastSLAM. The location of the user is a vector of euclidean coordinates $\boldsymbol{s_t} = [u_{t,x}, u_{t,y}]^T$. The motion model is thus defined as:

$$\begin{aligned} \delta' \sim \mathcal{N}(\delta, \sigma_\delta), r' \sim \mathcal{N}(r, \sigma_r) \\ \boldsymbol{s_t^m} = \boldsymbol{s_{t-1}^m} + [r' \cos(\delta'), r' \sin(\delta')]^T \end{aligned} \tag{1}$$

where δ and σ_δ are the user's measured heading and its variance; r and σ_r are the mean and variance of the step length.

B. RSSI of Discovery Beacons

We use the RSSI of a beacon to estimate the distance from the user to the device, using the *log distance path loss model*:

$$\text{RSSI} = -10n \log(\frac{d}{d_0}) + \text{RSSI}_0 \tag{2}$$

where n is the path loss exponent, d is the distance between the transmitter and receiver, and RSSI_0 is the mean RSSI measured at the reference distance d_0 (usually one meter). We pre-process the signal with a low-pass filter to reduce random noise and unpredictable multipath effects.

In ArcIoT, the human body is an additional source of propagation loss. Fig. 4a shows the problem graphically. Each line represents a different angle between the user and the device, ranging from 0° (the device is directly in front of the user) to 180° (the device is exactly behind the user). As the angle increases, distance measurements become increasingly unreliable. To resolve this issue, we correct the distance d from Equation 2 using the following equation:

$$d_c^\theta = d - \gamma_{\text{body}} \times (1 - \cos(\theta/2)) \tag{3}$$

where θ is the estimated heading angle of the user relative to the IoT device. When the line-of-sight path is blocked by the body (i.e., $\theta > \frac{3\pi}{2}$), we discard the measurement entirely. Fig. 4b demonstrates the result after this correction, overall, the mean squared error between the measured distance and the ground truth reduces from 5.4 in Fig. 4a to 2.2 in Fig. 4b.

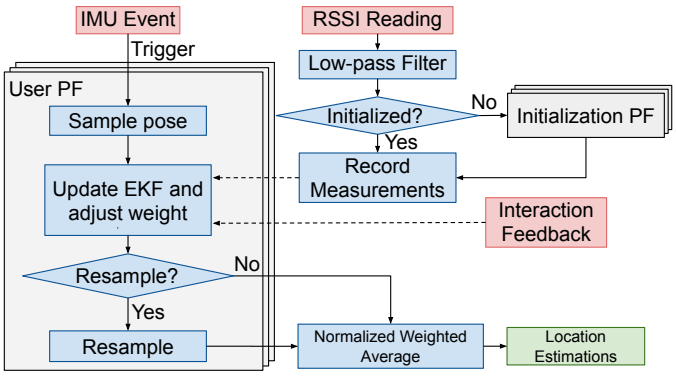


Fig. 5: The ArcIoT localization framework. Red boxes indicate inputs to the system; the green box is the output.

C. Localization framework in ArcIoT

Fig. 5 shows the algorithmic flow of ArcIoT's localization service. RSSI values feed into the low-pass filter, which is adjusted for human body shadowing. The IMU signal feeds the particle filter (PF) that underlies the FastSLAM algorithm.

Estimating device locations. When ArcIoT first discovers a device, we initialize a local model of the device's position. We adopt a separate particle filter (PF) [21], [24] for each IoT device. This initialization-PF uses an iterative process to refine the estimate of the device's location. The initialization PF consists of N particles, each of which represents a different estimate of the device's location. Using the initial RSSI value, we initialize the particles to a circle around the user with the heading angle, θ_i, of particle i varying from 0° to 360°. A particle i's distance d_i and angle θ_i (relative to the user) are: $d_i \sim \mathcal{N}(d_c^{\theta_i}, \sigma_z)$ and $\theta_i = \frac{2\pi i}{N}$ where $d_c^{\theta_i}$ is from Equation 3, and σ_z is the variance of the RSSI measurement. Initially, we assign a weight of one to each particle, since all locations are equally likely. As we acquire additional RSSI samples, each particle's weight is adjusted based on how well the particle's estimate matches each new RSSI value using: $w_{i,t} = w_{i,t-1} f(d_c^{\theta_i} | d_{i,u}, \sigma_z)$, where $d_{i,u}$ is the distance from particle i's estimate of the device's location to the user's current location and f is the pdf of the normal distribution. To avoid *particle degeneracy*, or the situation in which most particles have weights close to zero, we adopt the Sequential Importance Resampling algorithm (SIR) [25]. After each iteration, we compute the mean and variance of the N particle estimates. Once the variance is smaller than a

threshold, the device's location is considered initialized. We convert the relative distance d_i and angle θ_i to coordinates based on the current estimation of the user's location, and incorporate the measurements into the Extended Kalman Filter (EKF) for the device. We label the estimated location of device b as $\mu^m_{b,t}$ and the covariance as $\Sigma^m_{b,t}$.

Estimating the user's location. We estimate the user's location using the prior location, input from the IMU, and information about the distance to known devices. We use another particle filter in which each particle represents a possible location of the user. At time t, the location estimate $\boldsymbol{s^m_t}$ in particle m is sampled using Equation 1. We use this new estimate to update the weights of the particles and to update the EKFs estimating the device locations. We only update the EKFs and the weights if a new location estimate is generated (i.e., the user moved) or a sufficient number of RSSI measurements have been observed. We use a standard EKF update. For device b at time t, the EKF is updated as:

$$h(\boldsymbol{s^m_t}, \mu^m_{n_t,t-1}) = \left\|\boldsymbol{s^m_t} - \mu^m_{n_t,t-1}\right\| \tag{4}$$

$$\boldsymbol{H} = \frac{\partial h}{\partial \mu^m_{n_t,t-1}} = \frac{\boldsymbol{s^m_t} - \mu^m_{n_t,t-1}}{\left\|\boldsymbol{s^m_t} - \mu^m_{n_t,t-1}\right\|} \tag{5}$$

$$\boldsymbol{S} = \boldsymbol{H}\Sigma^m_{n_t,t-1}\boldsymbol{H}^T + Q_t \tag{6}$$

$$\boldsymbol{K} = \Sigma^m_{n_t,t-1}\boldsymbol{H}^T\boldsymbol{S}^{-1} \tag{7}$$

where h is the expected distance from the previous model, $\boldsymbol{S}$ is the innovation covariance computed with the partial derivative of h over the estimation μ. The Kalman gain $\boldsymbol{K}$ represents the information in this update. In the standard update for $\boldsymbol{S}$ (Equation 6), Q_t is a constant minimal covariance. In ArcIoT, based on the characteristics of the RSSI measurements (e.g., a small RSSI value has less variance), we adjust this value based on the current measurement ($d_c^{\theta_b}$):

$$Q_t = Q_{\text{min}} + \frac{d_c^{\theta_b}}{Q_d} \tag{8}$$

where Q_{min} is the standard minimal covariance and Q_d is a transfer parameter characterizing the wireless technology used for beacons. We define a condition A:

$$A = (d_c^{\theta_b} < \frac{\sigma_{\text{init}}}{2}) \wedge (|d_c^{\theta_b} - h(\boldsymbol{s^m_t}, \mu^m_{n_t,t-1})| < \sigma_{\text{init}})$$

that holds if the ranging measurement $d_c^{\theta_b}$ is smaller than half of the initialization threshold σ_{init} while the estimation h is far from the measurement. In this instance, instead of updating the current estimation, we reinitialize the estimation at the user's current location. Thus the updates for μ and Σ are:

$$\mu^m_{n_t,t} = \begin{cases} \mu^m_{n_t,t-1} + \boldsymbol{K}(d_c^{\theta_b} - h) & \text{if } A \text{ is FALSE} \\ \boldsymbol{s^m_t} & \text{otherwise} \end{cases} \tag{9}$$

$$\Sigma^m_{n_t,t} = \begin{cases} \Sigma^m_{n_t,t-1} - \boldsymbol{K}\boldsymbol{S}\boldsymbol{K}^T & \text{if } A \text{ is FALSE} \\ \begin{pmatrix} \sigma_{\text{init}}/2 & 0 \\ 0 & \sigma_{\text{init}}/2 \end{pmatrix} & \text{otherwise} \end{cases} \tag{10}$$

At each step, we also use $\boldsymbol{S}$ to update the weight of the particle: $w^m_t = w^m_{t-1} f(d_c^{\theta_b} | h, \boldsymbol{S})$. Similar to the initialization PF, we run the SIR algorithm to resample the particles when the number of particles with significant weights is low.

Integrating feedback. ArcIoT's localization framework incorporates real-time feedback from the user. The feedback is a binary input indicating whether ArcIoT identified the correct device. We integrate this information by adjusting the weight of the user particles. At time t, the user tries to interact with the device on screen with the observed location $[o_x, o_y]$ and ArcIoT matches the device on screen to device b. If the user provides feedback for this interaction, for each user particle m, we adjust the weight w^m_t based on the distance between the observed location $\boldsymbol{o}$ and the estimated location μ^m_b.

$$w^m_t = \begin{cases} \frac{w^m_{t-1}}{\left\|\boldsymbol{o} - \mu^m_b\right\|}, \text{ if feedback is positive} \\ w^m_{t-1} \cdot \max\{1, \frac{\left\|\boldsymbol{o} - \mu^m_b\right\|}{\gamma_{\text{neg}}}\}, \text{ if feedback is negative} \end{cases}$$

where γ_{neg} is a constant parameter for negative feedback.

Localization Output. The outputs of the localization framework are the location estimates of the user and the devices. These values are computed as the normalized weighted averages of the estimations of the particles in the respective filters.

V. EVALUATION

We implemented ArcIoT in Java on a Google Pixel 3 running Android 10. We used the Nordic Thingy52 [26], a BLE-enabled IoT sensor device, as a beacon associated with each IoT device. This allowed us to control discovery beacon parameters and take accurate measurements of our localization service. The devices were configured to send beacons every 100 milliseconds. To demonstrate bi-directional interaction, we also directly controlled the Thingy's LED and speaker from ArcIoT control interfaces. In addition, we implemented control interfaces for proprietary devices such as Philips Hue lights.

We performed three experiments. First, we used a scripted set of smart-home activities to evaluate ArcIoT's potential day-to-day performance in a smart home. Second, we drilled more into ArcIoT's localization service with a set of benchmarks in a single controlled environment. Third, we report on the overhead of ArcIoT on a commodity smartphone.

The first experiment used five different deployments in real home environments.[1] The rest were performed in the home of the first author. Before our experiments, we initialized ArcIoT by simply placing an ArcIoT-enabled device in the space and turning it on. When the user is completely stationary, ArcIoT can initialize a new device in less than 5 seconds; when a user is moving around the device, initialization takes less than 21 seconds.

Experiment 1: Interacting with devices. We first evaluated how accurately ArcIoT responds to users' intentions and its resilience to changes in the layout of devices. We used a scripted set of interactions and define accuracy as the ratio of successful interactions to all interactions. A failed interaction is when the user issues a command but a device different from the one on the screen performs the action. Since our main challenge is selecting from among visually identical devices, this experiment used three identical Thingy devices. In every

[1]COVID-19 curtailed our planned user studies. Instead, five members of our research team (excluding the first author) evaluated ArcIoT in their homes.

TABLE I: Accuracy of ArcIoT

Deployment	No feedback	Neg. feedback	All feedback
One room	90.5 %	93.3 %	95.2 %
Two rooms	92.4 %	90.5 %	96.2 %

Event	W/o re-initialization	W/ re-initialization
Displacement	81.0%	92.4%

interaction, the user first locates and points at the device. Then he/she taps on the displayed controller and observes the action. At last the user provide feedback after the action.

In each home, we installed the devices at arbitrary locations at least three meters apart. Our script consisted of 21 interactions: 7 with each of device, randomly interleaved. Each environment had two deployments; in the first, all of the devices were in the same room; in the second, the three devices were spread across two adjoining rooms. We tested three feedback modes: "no feedback", in which the user did not give any feedback; "negative feedback", in which feedback was only incorporated when it was negative; and "all feedback", in which the user gave a positive or negative feedback after every interaction. In both deployments, we executed the script of 21 interactions three times: once for each feedback mode.

The results are in the top of Table I. Accuracy was not notably affected by the configuration of devices across one or two rooms. ArcIoT was able to perform better when the user provided more feedback, but the performance with no feedback was reasonably good. The lower value for the negative feedback case in two rooms is likely due to users placing two devices on opposite sides of the same wall, confusing the localization service. From our observations, persistent failures for a specific device could be intuitively remedied by the user intentionally standing near a device briefly (i.e., less than five seconds) to allow the PF to stabilize around the detected RSSI. Overall, ArcIoT reliably identifies devices, even when they are visually identical. ArcIoT benefits from user feedback, but it can also benefit from users' natural self corrections.

To evaluate how resilient ArcIoT is to changes in device layout, we moved a device at the end of the last experiment above, in the case of two rooms and all feedback. ArcIoT deals with device displacement by re-initializing the device using Equation 9. A user can explicitly trigger re-initialization by staying close to the device for 5 seconds. The results for when the user did *not* explicitly re-initialize the device are shown in the bottom of Table I. With explicit re-initialization, ArcIoT's accuracy immediately recovered to more than 90%. However, even without explicit re-initialization, ArcIoT recovered by the end of the script, indicating that it was able to adjust its internal model of the IoT space.

Experiment 2: Localization accuracy benchmarks. Our second experiment evaluated the localization system in a home environment whose layout is depicted in Fig. 6. The devices (stars) were placed against the wall or on the desk. The smartphone was carried by a user starting from the location marked by the red dot and following the path shown with arrows. In each run, the user repeated the path 5 times, taking 120 steps in total. The user did not interact with or stop in front of the devices. We continuously computed three values:

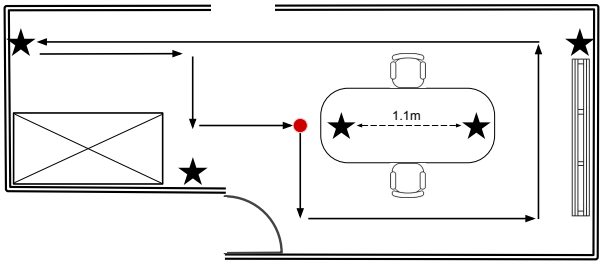


Fig. 6: Sketched floor plan for Experiment 2 (drawn to scale). The stars mark the locations of the devices. The user starts at the red dot.

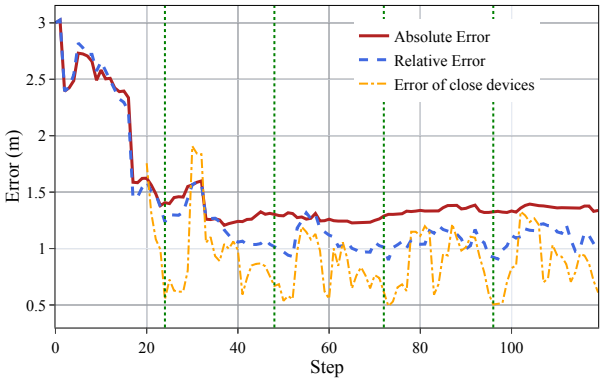


Fig. 7: Localization results in ArcIoT for absolute location, relative location, and the location of the closest two devices. The vertical lines mark the repeat of the path.

(i) the *absolute error*, i.e., the average distance between the estimated locations of the devices and the ground truth; (ii) the *relative error*, i.e., the average distance between the estimated relative location and the ground truth relative location; and (iii) the *error of close devices*, i.e., the average relative error of the two devices closest to the user.

We show the results (averaged over six runs) in Fig. 7. The devices were quickly initialized and the absolute error dropped below 1.5m, which is comparable to state-of-the-art infrastructure-free Bluetooth-based indoor localization systems [27]. The absolute error increased slightly at the end because we do not use predefined calibration landmarks. However, ArcIoT requires only relative locations of devices. In the first 40 steps, the absolute and relative errors are close, but later, the absolute error slightly increases while the relative error remains low. Because small RSSI readings tend to have less noise, we introduce adaptive covariance in Equation 8 and thus ArcIoT should have better estimates for closer landmarks. This is borne out by the third measurement; the error of the two closest devices was below the average of all devices most of the time and was often as low as 0.5m.

Experiment 3: ArcIoT system overhead. Finally, we evaluated the overhead of ArcIoT in terms of latency and energy. We focused on latency to capture the responsiveness of ArcIoT. During a single interaction, there are three types of delays that may be noticeable to the user: (i) *interaction delay*, i.e., the time between a device being visible on the screen and it being interactive; (ii) *interface display delay*, i.e., the time between the user making the intention (tapping on the screen) to interact and the control interface being displayed;

TABLE II: Energy consumption for five minutes of use (in mAh)

Component	ArcIoT (foreground)	ArcIoT (background)	Philips Hue (foreground)
Major	67.0 (Camera)	0.40 (CPU)	13.2 (Display)
Total	***80.38***	***0.47***	***13.81***

and (iii) *communication delay*, i.e., the time between a control command being issued and the device's response.

We used ArcIoT to control a Philips Hue Go light (with an associated Thingy beacon) then issued a command to the light via the Philips Hue bridge. The averaged latency over 5 runs of experiments is as follows: *interaction delay*: 12.8ms, *interface display delay*: 63.0ms, and *communication delay*: 61.0ms. The latency for all components was below 100ms, which means the delay was not noticeable to a human user. The Communication Delay depends on the technology used to control the device (in this case WiFi) which is not part of ArcIoT. Overall, ArcIoT enables interacting with real smart home devices without human-perceptible delays.

We measured ArcIoT's energy consumption using the same setup and used Android dumpsys to measure battery discharge. Table II shows the major energy consumer for ArcIoT in the foreground and background along with the results for the Philips Hue app. Most of the energy consumed by ArcIoT was due to camera and screen use, which only occur when the user interacted with a device. When ArcIoT and its localization service were running in the background, the energy consumption was minimal. ArcIoT's energy footprint is within reason for executing on commodity smartphones. For context, the Pixel 3's battery capacity is 2915mAh.

VI. Conclusion

In this paper, we demonstrated ArcIoT's ability to enable intuitive and responsive device interactions in the IoT. ArcIoT reliably monitors the mobility of the user and maintains a map of devices by leveraging built-in sensors and low-cost BLE beacons. We evaluated ArcIoT in various home environments to show that it performs with high accuracy and is resilient to environmental dynamics. We also showed that ArcIoT's overhead with respect to latency and energy consumption is within reason on commodity smartphones.

While we evaluated ArcIoT in smart home applications, many other applications can also be realized. For example, in a store, ArcIoT can be used to render information next to a product; in an office, ArcIoT can be used to control projectors, displays, speakers, etc. In this context, ArcIoT enacts the user's intentions on digital devices in the surrounding world by leveraging the combined benefits of AR and contextual information provided by pervasive computing devices. Overall, ArcIoT demonstrates the intuitiveness and efficiency of visual-based interactions and opens new possibilities of an IoT world where the user is exposed to minimal technological details.

Acknowledgements

The authors would like to thank the members of the Mobile and Pervasive Computing group at UT Austin for their assistance in the evaluation. This work was funded in part by the National Science Foundation under grants CNS-1813263, CNS-1909221, CNS-1907959. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] P. Achlioptas *et al.*, "Referit3d: Neural listeners for fine-grained 3D object identification in real-world scenes," in *Proc. of ECCV*. Springer, 2020, pp. 422–440.
[2] J. Hua *et al.*, "rIoT: Enabling seamless context-aware automation in the internet of things," in *Proc. of IEEE MASS*, 2019.
[3] I. Ullah *et al.*, "Cloud based IoT network virtualization for supporting dynamic connectivity among connected devices," *Electronics*, vol. 8, no. 7, 2019.
[4] Y. Saputra *et al.*, "Warble: Programming abstractions for personalizing interactions in the internet of things," in *Proc. of MOBILESoft*, 2019.
[5] S. Song and J. Xiao, "Deep sliding shapes for amodal 3D object detection in RGB-D images," in *Proc. of CVPR*, 2016.
[6] J. Yu *et al.*, "A vision-based robotic grasping system using deep learning for 3D object recognition and pose estimation," in *Proc. of ROBIO*, 2013.
[7] A. Simonetti Ibañez and J. Paredes Figueras, "Vuforia v1. 5 SDK: Analysis and evaluation of capabilities," Master's thesis, Universitat Politècnica de Catalunya, 2013.
[8] T. Zachariah and P. Dutta, "Browsing the web of things in mobile augmented reality," in *Proc. of ACM HotMobile*, 2019.
[9] D. Jo and G. J. Kim, "IoT+ AR: pervasive and augmented environments for "Digi-log" shopping experience," *Human-Centric Computing and Information Sciences*, vol. 9, no. 1, pp. 1–17, 2019.
[10] S. Mayer *et al.*, "Device recognition for intuitive interaction with the web of things," in *Proc. of ACM Ubicomp*, 2013.
[11] K. Ruan and H. Jeong, "An augmented reality system using QR code as marker in android smartphone," in *Proc. of IEEE SCET*, 2012.
[12] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *Proc. of IEEE/RSJ IROS*, 2016.
[13] V. Heun, S. Kasahara, and P. Maes, "Smarter objects: using AR technology to program physical objects and their interactions," in *Proc. of CHI*, 2013.
[14] Y. Park, S. Yun, and K.-H. Kim, "When IoT met augmented reality: Visualizing the source of the wireless signal in AR view," in *Proc. of ACM MobiSys*, 2019.
[15] F. Subhan *et al.*, "Indoor positioning in bluetooth networks using fingerprinting and lateration approach," in *Proc. of ICISA*, 2011.
[16] R. Faragher and R. Harle, "Location fingerprinting with bluetooth low energy beacons," *IEEE J. on Selected Areas in Communications*, vol. 33, no. 11, pp. 2418–2428, 2015.
[17] A. R. Jiménez and F. Seco, "Finding objects using UWB or BLE localization technology: A museum-like use case," in *Proc. of IPIN*, 2017.
[18] M.-S. Pan and C.-J. Chen, "Intuitive control on electric devices by smartphones for smart home environments," *IEEE Sensors Journal*, vol. 16, no. 11, pp. 4281–4294, 2016.
[19] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.
[20] M. Montemerlo *et al.*, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proc. of AAAI*, 2002.
[21] W. Bulten, A. C. Van Rossum, and W. F. Haselager, "Human SLAM, indoor localisation of devices and users," in *Proc. of IEEE IoTDI*, 2016.
[22] B. Jang, H. Kim, and J. W. Kim, "IPSCL: An accurate indoor positioning algorithm using sensors and crowdsourced landmarks," *Sensors*, vol. 19, no. 13, p. 2891, 2019.
[23] F. Seco and A. R. Jiménez, "Autocalibration of a wireless positioning network with a FastSLAM algorithm," in *Proc. of IPIN*, 2017.
[24] J. Svečko, M. Malajner, and D. Gleich, "Distance estimation using RSSI and particle filter," *ISA Transactions*, vol. 55, pp. 275–285, 2015.
[25] J. D. Hol, T. B. Schon, and F. Gustafsson, "On resampling algorithms for particle filters," in *Proc. of IEEE NSSPW*, 2006.
[26] Nordic, "Nordic Thingy52 Sensor Tag," https://www.nordicsemi.com/eng/Products/Nordic-Thingy-52.
[27] U. M. Qureshi, Z. Umair, and G. P. Hancke, "Evaluating the implications of varying bluetooth low energy (BLE) transmission power levels on wireless indoor localization accuracy and precision," *Sensors*, vol. 19, no. 15, p. 3282, 2019.