Snap-To-It: A User-Inspired Platform for Opportunistic Device Interactions

Adrian A. de Freitas[1], **Michael Nebeling**[1], **Xiang 'Anthony' Chen**[1], **Junrui Yang**[2], **Akshaye Shreenithi Kirupa Karthikeyan Ranithangam**[3], **Anind K. Dey**[1]
[1] Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA, USA
[2] Peking University, Beijing, China [3] Coimbatore Institute of Technology, Tamil Nadu, India
(adefreit, mnebelin, xiangche, anind)@cs.cmu.edu, (jackieyang51, akshaya.kar)@gmail.com

ABSTRACT
The ability to quickly interact with any nearby appliance from a mobile device would allow people to perform a wide range of one-time tasks (*e.g.,* printing a document in an unfamiliar office location). However, users currently lack this capability, and must instead manually configure their devices for each appliance they want to use. To address this problem, we created Snap-To-It, a system that allows users to opportunistically interact with any appliance simply by taking a picture of it. Snap-To-It shares the image of the appliance a user wants to interact with over a local area network. Appliances then analyze this image (along with the user's location and device orientation) to see if they are being "selected," and deliver the corresponding control interface to the user's mobile device. Snap-To-It's design was informed by two technology probes that explored how users would like to select and interact with appliances using their mobile phone. These studies highlighted the need to be able to select hardware *and* software *via* a camera, and identified several novel use cases not supported by existing systems (*e.g.,* interacting with disconnected objects, transferring settings between appliances). In this paper, we show how Snap-To-It's design is informed by our probes and how developers can utilize our system. We then show that Snap-To-It can identify appliances with over 95.3% accuracy, and demonstrate through a two-month deployment that our approach is robust to gradual changes to the environment.

ACM Classification Keywords
H.5.2 Information Interfaces and Presentation (e.g. HCI): User Interfaces: Input Devices and Strategies, Interaction Styles

Author Keywords
Internet of things, mobile interaction

INTRODUCTION
The rapid proliferation of "smart" appliances (*e.g.,* printers, speakers) has created new opportunities for users to interact and engage with technology in meaningful ways. Taking advantage of this connectivity, however, is a well-known problem [7, 19, 23, 25, 31]. For example, if users want to use a networked printer, they have to know its IP address and install the correct drivers. Similarly, if a user wants to play music on a Bluetooth speaker, they must set the appliance to be discoverable, and then find and pair with it. While this level of effort is acceptable for appliances used frequently, it makes one-time or spontaneous use impractical. This discourages users from interacting with new or unfamiliar appliances, and can even force them to seek alternative solutions (*e.g.,* asking a friend or stranger to print a document on their behalf).

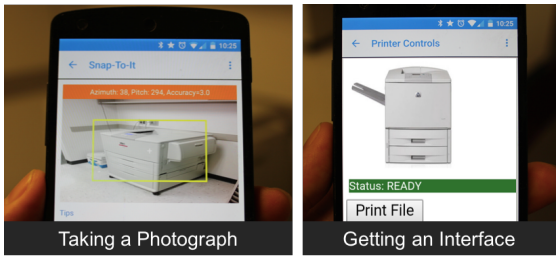


Figure 1. A user takes a photo of the appliance they want to control (left). Our system shares this image (along with the user's location and device orientation) with nearby appliances. The user's device connects to the appliance that best matches the image, and receives a custom UI (right).

Our goal is to empower users with the ability to quickly and easily connect and interact with ubiquitously distributed appliances in *any* environment. To achieve this, we have developed Snap-To-It, a software system that transforms a mobile phone into a universal interaction tool. Inspired by how people already use their smartphones, Snap-To-It lets users select a nearby appliance by "snapping" a photo of it. The Snap-To-It app then transmits this image (along with the user's location and device orientation) across a local area network so that appliances and/or software proxies can analyze it. If a photo matches an appliance, the app connects to it and renders a custom interface (Figure 1). Otherwise, the user receives a list of the most likely candidates and is asked to select from them.

Prior research has explored several solutions for users to control appliances from their mobile device [8, 27, 34, 30, 35]; Snap-to-It expands on this work in three important ways.

First, our work broadens our understanding of the types of opportunistic interactions users would like to perform using a mobile device. Prior to developing Snap-To-It, we conducted

CHI'16, May 07–12, 2016, San Jose, CA, USA
©2016 ACM. ISBN 978-1-4503-3362-7/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2858036.2858177

two probes to learn 1) how users would like to select an appliance in an unfamiliar environment, and 2) how they would like to interact with that appliance using their mobile phone. These probes revealed that participants preferred selecting appliances by taking a photograph (as opposed to scanning QR codes or selecting them from a list), and identified six categories of general use cases, respectively. Snap-To-It's design is directly informed by these findings, providing a research platform that not only shows what types of interactions are *possible* using a mobile device, but that are actually *desired* by end users.

Secondly, Snap-To-It provides a truly opportunistic way of interacting with a wide range of appliances. Rather than require the user to download a list of nearby appliances [19] or connect to a well-known server [10], our system uses multicasting to identify an appliance based on its photo. This makes it particularly useful when users are visiting a location for the first time, and need to use an appliance once or spontaneously. Additionally, while prior work has shown that mobile devices can be used to select and interact with hardware [29, 33] and software [11], they have focused on one *or* the other. In contrast, Snap-To-It compares user-taken photographs against stock images and real-time screenshots. This lets it support both types of appliances using a single interaction technique. Finally, since Snap-To-It works with any photograph, our system can even be used to make non-computational objects (*e.g.,* signs, maps) selectable. This increases the range of appliances, applications, and information that users can interact with without requiring each one to have an embedded computer.

Third, Snap-To-It provides a developer-friendly way to allow appliances to be selectable *via* camera. Our system's middleware automatically processes incoming photos, establishes connections with user devices, and delivers arbitrary HTML/JavaScript-based user interfaces. This allows developers to incorporate Snap-To-It's functionality into appliances without having to implement their own photo recognition. Additionally, our middleware can act as a software wrapper for existing appliances and applications. This greatly expands the range of appliances that can utilize our system without forcing users or site administrators to purchase additional hardware.

In the following section, we describe our probes, and show how user responses have influenced Snap-To-It's design. Afterwards, we describe Snap-To-It's architecture, and present four prototypes that highlight its capabilities. Through three studies, we show that Snap-To-It is sufficiently accurate to be deployed in real-world environments. Finally, we address issues such as usability, responsiveness, and hardware requirements, and close with a discussion of related and future work.

TECHNOLOGY PROBES AND DERIVED REQUIREMENTS

With Snap-To-It, we aim to let users quickly select and interact with any object in their physical environment. Yet while prior work has shown that this two-step interaction model supports many use cases, they focus on the challenges associated with connecting to devices and receiving and/or rendering custom interfaces [19, 27]. Consequently, we lack general knowledge of 1) how users would *like* to select appliances using their mobile devices, and 2) the specific types of interactions they would like to perform, both of which are crucial for building a usable and effective system.

Figure 2. Appliances used in our first probe.

To fill these gaps, we conducted two technology probes that examine the process of selecting and using appliances in detail. In this section, we describe both probes and show how their findings have informed Snap-To-It's design.

Probe 1: Selecting an Appliance from the Environment

Our first probe was targeted towards learning how users would like to select an appliance using their mobile device. To investigate this, we asked 10 participants (6 male, 4 female; 22-42 years old; mix of novices and experts with mobile phones and QR codes) to try out a variety of techniques and provide feedback. Prior to the study, we gave participants a list of 7 techniques that could be implemented on existing smartphones: taking a photograph, scanning a QR code (on and off the device), near-field communication, "bumping" the phone against the appliance, text search, and selecting the appliance from a list. We then had participants rank order each technique for a variety of appliances (Figure 2), and evaluated the three techniques that were both rated the highest and compatible with the widest range of appliances: taking a picture, scanning a QR code, and selecting from a list.

The probe was conducted in our institution's common area. We selected a mix of hardware/software (Figure 2) that visitors might want to interact with using their phones. Prior to the study, we placed QR codes on each appliance and/or application using best practices for the size and positioning established in [5], and created a list of every appliance within 30 feet of the room (*i.e.,* Bluetooth range). We then gave participants an app that lets them take a photo, scan a QR code (using a common QR code reader app [4]), and select an appliance from an alphabetized list. Participants spent 30 minutes trying each technique in random order from both touching distance and 5-10 feet away, and were compensated with $15 USD.

Feedback on the list-based technique was overwhelmingly negative. Despite being conceptually simple, participants had difficulty identifying a specific appliance from a list of names. While this problem might be overcome by choosing names other than the ones assigned by our network administrators (*e.g.,* "Zircon", "Pewter"), prior research suggests that no naming convention works well with every user [12]. Even though the list only had 11 items, participants reported that they were "scrolling and scrolling and scrolling" (P1). This led them to unanimously rate the list technique as the worst.

Feedback on the QR code technique was also negative. While participants could see the codes on each appliance, eight of them had trouble getting the phone to successfully scan one or more of them. The digital projector (Figure 2b) was especially problematic since it was located above a conference table; this

forced users to scan its code from an angle. While placing multiple codes on each appliance (*e.g.,* one on each side) might help, participants also noted that they disliked having QR codes appear on a screen, especially when multiple codes are needed to differentiate between different applications (Figure 2c). Consequently, while participants felt that QR codes were better than lists, only one rated it as the best technique.

In contrast to the list and QR code methods, feedback on the camera technique was almost universally positive. Participants called the technique "fun" (P3, P4, P7), and noted that it was easy to understand: "I can hand it to my 4-year old, and he'll understand" (P1). Others noted that the time to take a photograph felt shorter than the time needed to scan the QR code, as they did not have to continually hold their phones up to the code for it to be recognized. In fairness, several participants did state that the difference between QR codes and the camera was negligible when the appliance being selected is physically close. However, for software applications and objects farther away, participants agreed that the photo-based selection method offered a better overall experience.

Collectively, these findings illustrate the value of photo-based selection. While our participants were eventually able to interact with every appliance using all three techniques, 9 out of 10 stated that they *preferred* selecting appliances by taking a photo. This motivated us to develop a system that could make camera-based selection a reality.

Probe 2: Types of Interactions

For our second probe, we wanted to know how users would like to interact with appliances once they have been selected. To investigate this, we asked 28 participants (14 male, 14 female; 19-60 years old; mix of novice and experts) to create a "wish list" of interactions that they would like to perform. For this probe, we developed an Android app that let users take photos of appliances and annotate how they would use them. We then asked them to use the app for a week as they went about their normal routine, and compensated them with $20.

Through this process, we collected a total of 195 photographs. Each photo was categorized according to 1) the interaction being performed, and 2) the technologies needed to support them. This information was then directly used to identify general use cases and derive critical technical requirements. Although the number of photos submitted by participants varied, our analysis found that each identified roughly the same number of general use cases (avg=2.1, SD=1.1), making sure that the findings were not overly influenced by any one participant.

Use Case Categories

Participants had a wide range of ideas as to how they wanted to interact with appliances using their mobile phone. In all, we identified six general use case categories (Figure 3).

Nineteen participants (68%) wanted to use their mobile phones to quickly interact with new or unfamiliar appliances (Figure 3, Column 1). The majority took photographs of office appliances such as printers, projectors, and multimedia controls. Others took photos of specialized equipment such as laser cutters and 3D printers. In each case, users wanted to use the appliance without installing software/drivers.

Another popular use case was to control appliances from afar (Column 2). Twenty participants (71%) took pictures of household/office appliances, vehicles and industrial equipment, and stated that they wanted to be able to control these devices from anywhere. In some cases, it was purely for added convenience (*e.g.,* "[I want to] turn off the lights when I'm in bed"). Others viewed their phone as a more hygienic way to interact with public objects (*e.g.,* lights, toilet handles) without having to physically touch them. Finally, several participants noted that being able to activate controls from a distance would be particularly helpful to disabled individuals. We found this to be an unexpected, but interesting use case that we wanted to support.

While we anticipated that participants would want to use their smartphones as a remote control, participants also identified several other ways to interact with appliances. Eight participants (29%) wanted to use their mobile device as a way to upload and download content (Column 3). Several users took pictures of public displays and televisions, and stated that they wanted to be able to send or receive content (*e.g., push* a PowerPoint presentation, *pull* a closed-caption feed). Interestingly, the idea of pushing/pulling content was not limited to computer devices. Several participants took photos of *disconnected objects* such as campus maps and public museum displays, and stated that they wanted to extract the information represented by the object using their phone (*e.g.,* converting a picture of a map into a digital version). In these cases, participants viewed these objects as "physical hyperlinks", and were interested in being able to use their mobile phone as a way to access an object's digital representation.

Seven participants (25%) wanted to use their phone to perform secure transactions (Column 4). Several of them took pictures of locked doors and computers, and noted how it would save them time and effort if they could authenticate to these devices while approaching them. Others took photographs of payment systems (*e.g.,* vending machines), and noted that it would be convenient if they could make purchases electronically.

A fifth use case identified by four participants (14%) was to learn more about a particular appliance (Column 5). One participant photographed a light switch, and wrote: "I am not so sure what these buttons control." Others took pictures of fire alarms, defibrillators, and household appliances, and asked for 1) what the appliance does, and 2) how to use it.

Finally, two participants (7%) wanted to use camera-based selection to easily transfer settings and preferences between appliances (Column 6). One participant took a photo of a treadmill, and said that he wanted to easily carry over his exercise preferences (*e.g.,* speed, incline) to another machine during his next workout session. The other took a photograph of a public television, and stated that he wanted to quickly find and tune to his favorite channels without having to search for them. In both cases, users wanted their phone to remember their past interactions. This would allow them to seamlessly transfer these preferences to new appliances.

Collectively, these responses highlight the need for a system like Snap-To-It. While only 18% of our participants stated that they use more than three devices daily, our probe reveals that

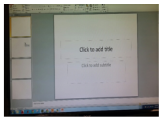
Use Case Category	Quick Control	Long Distance Operation	Transfer Content	Secure Transactions	Increase Intelligibility	Share Preferences
Motivation	*Connect and use a nearby device.*	*Operate a device from a remote location.*	*Allows users to post and receive data.*	*Be able to securely send information to a device.*	*Improves users' knowledge of how a device operates.*	*Allow settings/preferences to be shared across devices.*
Sample Participant Responses and Appliances	 "I want to print"	 "I would like to . . . heat my food before I reach home."	 "Extend my phone screen to a bigger display."	 "Access doors to let me in"	 "I am not so sure what these button controls"	 "Preset my treadmill"
	 "Changing slides in a powerpoint presentation"	 "Hold elevator when I'm in my apartment"	 "If I took a photo of a map, then a 3D map appears, it would be useful."	 "Enter choice thru my phone N pay through my phone"	 "How does this work?"	 "Set my favorite TV channel."

Figure 3. General use cases for Snap-To-It, as derived from user responses in our second technology probe.

1) all of them *wanted* to interact with more devices than they currently do, and 2) that this desire extends beyond simple remote controls. This emphasizes the relevance of our work, and the need for a more versatile way of interacting with appliances than is currently available.

Functional Requirements

Our probes show that users seek a simple but versatile way of interacting with a wide range of objects/appliances. From their responses, we have identified five functional requirements:

R1: Support Photo-Based Selection. Our first probe shows that users prefer selecting appliances *via* photos. Consequently, while other methods (*e.g.,* QR codes) may be necessary at times, camera-based selection should be used when possible.

R2: Interact with Software and Hardware. Current systems typically focus on interacting with hardware [19, 2] *or* software [11]. Our second probe, however, shows that users frequently interact with both types of appliances (Figure 3). For this reason, tools like Snap-To-It need to support both hardware and software interactions through a single interface.

R3: Interact with Appliances from Afar. Our second probe shows that users want to interact with appliances that are out of sight (Figure 3, Column 2). This points to the need to remember previously used appliances (*i.e.,* a "favorites list") so that remote operation is possible.

R4: Render Complex Interfaces. Many of the use cases identified in Figure 3 call for complex user interfaces (*e.g.,* interactive maps, real-time closed-captioning feeds). These interfaces are more dynamic than the button/slider interfaces supported by existing remote control systems, and illustrate the need to be able to render any arbitrary UI "on the fly."

R5: Support Expandability. The final insight gained from our probes is that users use new technologies in unexpected ways. While we know that there are several obvious use cases that our system *needs* to support (*e.g.,* sending commands to remote appliances), our participants have also identified a number of use cases that are important to them, such as the ability to perform remote transactions and save/transfer settings. These findings highlight the real world challenges of creating a universal interaction tool, and that systems such as Snap-To-It need to be flexible enough to accommodate these use cases as they are discovered.

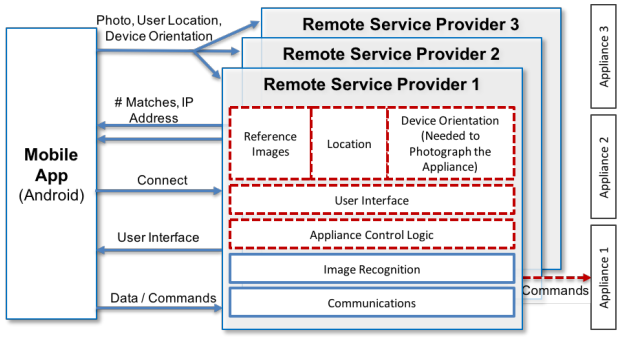


Figure 4. High level system architecture. Solid boxed components are provided by Snap-To-It, dotted boxes by developers/administrators.

In the following section, we show how Snap-To-It's design is directly influenced by our probes. Our system satisfies all five requirements described above, and provides a middleware that makes it easy to add our system's functionality to new or existing appliances. This allows it to support all six use case categories while still leaving room for future expansion.

SYSTEM DESIGN

Snap-To-It consists of two components (Figure 4): 1) a mobile app, and 2) a series of remote service providers (RSP), each of which controls one or more appliances and resides either on the appliance itself or a designated proxy (*i.e.,* a server). When users take a photo, the app sends a multicast datagram containing a URL to the photo, as well as the phone's location and device orientation. RSPs on the same subnet can then use this information to determine if the user is looking at them, and respond with their IP address and port. In our system, RSPs analyze photos, deliver interfaces, and transmit remote commands to the appliance(s) they control. The app, in turn, acts as a thin client, and can interact with any appliance without having to know about it or its capabilities in advance.

This section describes Snap-To-It's important technical details. First, we show how our system recognizes appliances. Afterwards, we show how it renders interfaces, shares preferences, authenticates users, and supports extensibility.

Selecting an Appliance via the Mobile App

Snap-To-It offers two ways to select an appliance. The primary way is by using the mobile device's camera (**R1**). Each time

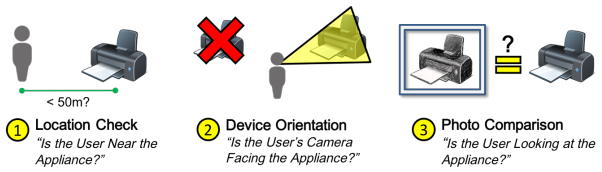


Figure 5. Snap-To-It's appliance recognition process.

a user takes a photo of an appliance, our app automatically uploads a 640×480 JPEG version of it to a cloud server (which simply hosts the photo for RSPs to access), and records the phone's location and orientation (azimuth, pitch, roll). It then transmits a *request message* containing the photo URL, location, and device orientation, and waits for a response. If an RSP reports that it matches a photo with a high degree of confidence, our app automatically connects to it. If no high confidence match is found (or the appliance is obstructed), the app provides users with the five highest matching appliances based on their reported context, and lets them choose which one they want to use. Our app always lets users return to the list of top matches for the most recently taken photo. This lets them gracefully recover in the event they (or the system) select the wrong appliance.

In support of **R3**, our mobile app also allows users to select appliances over long distances. Our system maintains a history of previously connected appliances, and lets users explicitly add appliances to a favorites list. The user can then use either list to reconnect to an appliance without having to be nearby.

Recognizing an Appliance

Snap-To-It matches user requests to specific appliances in three stages (Figure 5). In the first stage, each RSP extracts the latitude/longitude from the request message and calculates the distance between itself and the user. The RSP only then proceeds to the next stage if the user is within a predefined distance (*e.g.,* 50 meters). Since indoor location tracking is imprecise, our system only checks to see if the user is in the general vicinity of the appliance (*i.e.,* the same building). This lets RSPs disregard requests from users that are clearly out of visual range, and lets our system work in networks where a subnet covers a large area (*e.g.,* a college campus).

In the second stage, RSPs check to see if the user's camera is pointing towards the appliance they represent. Each RSP knows what direction a user needs to be facing in order to take a picture of it (specified *a priori* or obtained by using its onboard compass). RSPs can then check to see if the user's device is pointing towards it. Similar to before, this test cannot definitively tell if a user is looking at it. Instead, it only prevents appliances from comparing photos that are obviously taken from the wrong direction. This improves accuracy when appliances look similar, but face different directions.

The third stage examines the user's photograph. Here, each of the remaining RSPs under consideration downloads the user photo and extracts its salient features using the Scale-Invariant Feature Transform (SIFT) algorithm [22]. These features are then compared to reference photos (*i.e.,* photos of the appliance that are either provided in advance, or taken programmatically), and the results (*i.e.,* the maximum number of matches) are sent back to the mobile app. We use SIFT because the features it identifies are resilient against changes in the orientation of the camera and distance from appliances. This lets it effectively compare two images, even when they are taken from slightly different angles and distances.

For the above process to work, each RSP needs to know 1) what it looks like, 2) where its appliance is located, and 3) what direction the user needs to face to photograph it. For appliances with a GPS, compass, and display, this information can be automatically obtained by directing the RSP to periodically poll its sensors and take screenshots of its display, respectively. Many appliances (*e.g.,* printers), however, lack this capability. To support them, developers and/or administrators can collect this information *via* our mobile app, and provide the photos (and their associated metadata) to the RSP. While cumbersome, we have empirically found that this approach yields good results with as few as 3 reference photos (one taken from the front, and two from opposite 45 degree angles). This makes it easy for developers to create RSPs for a wide range of hardware, software, and non-computational objects (satisfying **R2**).

Additionally, our process requires that every reference photo contain *some* information about an appliance's surroundings. To achieve this, our app provides users with a targeting reticule (*i.e.,* a box), and instructs them to keep the appliance in the center of the image (Figure 1, left). The resulting photos contain enough background scenery to give the RSP a sense of where it is located in relation to the environment, as well as what appliances are near it. This lets our system differentiate between similar looking appliances, even when they are physically near each other (an examination of Snap-To-It's accuracy is provided in our validation).

Updating/Maintaining Reference Photos

Snap-To-It's accuracy largely depends on the quality of its reference photos. When these photos closely match the appliance's actual appearance, our system is able to identify appliances with a high degree of confidence. However, reference photos become stale over time as the environment changes. Consequently, there needs to be a way for RSPs to update their reference images with minimal assistance.

Snap-To-It overcomes this problem by utilizing user photographs. When users connect to an RSP, the photo they took is compared to the RSP's reference images. If the photo was taken at a similar angle to an existing photo (*e.g.,* within 10 degrees azimuth/pitch/roll) and has a low number of SIFT matches, the RSP replaces the older reference image. Otherwise, the system adds the photo to its current library (up to a specified limit). By letting RSPs update their reference images when the SIFT accuracy drops below a threshold, our system is able to learn how an environment is changing over time. This keeps them up to date, while simultaneously allowing them to recognize appliances from a wider range of angles.

Although this provides RSPs with new photos, it also opens the possibility of RSPs receiving blurry or inaccurate images by mistake (*e.g.,* an image of another appliance). To overcome this, RSPs ask users to judge another user's photo before

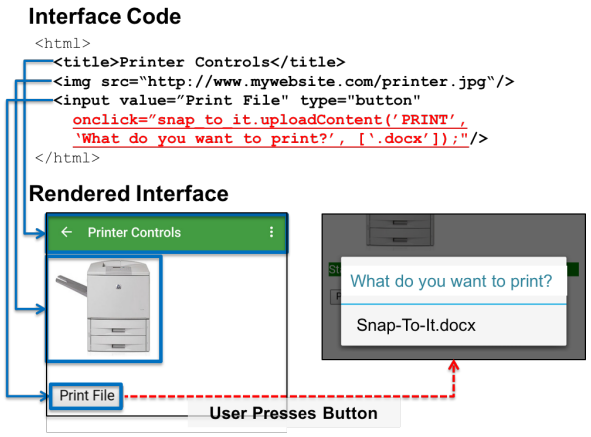


Figure 6. Sample controls for a printer. When users press the "Print File" button, it invokes our JavaScript API (red/underlined) to select and upload a file (bottom right).

adding it to its library. By leveraging user responses, RSPs can "weed out" low quality photos from its library. Moreover, since judging is only needed when a new photo is added, the impact on the end user experience is kept to a minimum.

Defining User Interfaces

From our second technology probe, we know that there are a handful of common operations that users need to perform in order to effectively control an appliance, such as the ability to transmit remote commands. As a result, Snap-To-It provides a JavaScript API that supports these core functions. When developers specify their interface, they can insert calls to our API at key events (*e.g.,* when a button is pressed). These calls then direct the app to perform a desired action (Figure 6).

This approach lets Snap-To-It perform operations that are normally inaccessible (or difficult to perform) from a mobile web browser. Our system allows developers to customize these operations (*e.g.,* specify what command is transmitted), and "push" new UIs without requiring the user to refresh their display. This allows them to create responsive, arbitrarily complex UIs (supporting **R4**), and makes our system extensible to a wide range of appliances and use cases.

Sharing Preferences

RSPs can also store preferences (*e.g.,* favorite TV channels, exercise settings) on the user's device, similar to the way that websites store cookies on a client device. When an RSP needs to save a preference, it calls our API's *setPreference* method, and specifies the name of the preference and its value. These values are then stored in the mobile app, and are provided to the RSP in future sessions.

Moreover, preferences can be shared *across appliances* (supporting **R5**). Each time an RSP replies to a request, it specifies the preference(s) it needs. The mobile app then delivers these preferences (if authorized by the user) to the RSP when it connects. Since RSPs can ask for any preference, our system lets appliances configure themselves (*e.g.,* transfer exercise settings) based on the user's interactions with similar and/or complementary appliances. In doing so, we reduce the need for manual configuration in many situations.

```
public class Printer_RSP extends RemoteServiceProvider {
 public Printer_RSP () {
  addPhoto("Photo1.jpeg",
    LOCATION, AZIMUTH, PITCH, YAW);
 }

 public String getInterface(User u) {
  return "<html> . . . </html>";
 }

 public void onRemoteCommand(User u, Command c) {
  if (c.getCommand().equals("PRINT")){
   // Print File Using URL Stored in c
  }
 }
}
```

STEP 1: Add Reference Photo(s)

STEP 2: Specify the UI

STEP 3: Process Commands

Figure 7. Pseudocode for a remote service provider.

Identifying Users

There are many situations where an RSP needs to customize the services it offers on a per-user basis. For example, an RSP for a printer on a college campus might want to provide students and faculty members with an interface to print in color, while limiting visitors to black and white.

To support these situations, Snap-To-It allows appliances to distinguish between individual users. Each time a mobile app connects to the RSP, it transmits a set of identifying credentials to the RSP. The RSP can then use these values to determine what level of service to provide. For now, our credentials contain the device's Android ID and a hash of the user's email address, as this information is sufficient to identify a particular user (provided the email address or ID is known beforehand). In the future, however, a more robust authentication method (*i.e.,* digital signatures) could be used in its place. This would provide greater security, while still allowing our system to offer relevant and *appropriate* services (supporting **R5** as well).

Creating a New RSP

Snap-To-It's middleware makes it easy for both first and third party developers to create and deploy their own RSPs. To demonstrate this, Figure 7 provides a sample implementation of a printer RSP. This code is simplified for brevity, but highlights the three steps needed to create an RSP from scratch:

1. Add Reference Photos. In the first step, we tell the RSP what its appliance looks like. For this example, we only provide a single image (along with its location, azimuth, pitch, and roll). However, more photos will obviously improve its ability to recognize itself in a user-submitted image (at the cost of increased computation time).

2. Specify the UI. The second step is to specify the user interface that the mobile app will display. Here, we return the HTML code that is described in Figure 6. Alternatively, we can provide the user with an HTTP/S link.

3. Process Commands. The final step is to process incoming commands from the mobile app. From Figure 6, we see that the app will automatically transmit a command (*e.g.,* "`PRINT`") to the RSP each time the user uploads a file. To detect this command, we check for this string in the `onRemoteCommand` method. We then use the URL contained within the command to download and print the file.

We implemented Snap-To-It in Java. It uses the OpenImaj library [17] to perform image comparisons, and UDP multicast and TCP sockets to transmit requests and send commands, respectively. Our system currently works in any environment with a local area network, but is targeted towards work areas such as offices. In the future, however, we hope to use *ad hoc* networking in order to support a wider range of locations.

VALIDATION

We validate Snap-To-It in two parts. First, we present four examples that were created using our system. We then evaluate Snap-To-It's accuracy, both under controlled conditions and after a two-month deployment.

Example Applications

In this section, we describe four applications (Figure 8a-d) that were developed using Snap-To-It. Each application is inspired by our general use cases, and demonstrates the range of interactions (hardware and software) our middleware supports.

Application #1: Game Controller

Our first application uses Snap-To-It to quickly control an appliance. For this application, we deployed a laptop running both a commercial game and an RSP. When a user takes a photo of the screen, Snap-To-It sends a multicast datagram containing a link to the image, as well as the user's location and device orientation. Our RSP then verifies that the user's phone was pointed towards the laptop and looking at the screen, and sends back a response containing the number of visual feature matches and its IP address. When the user connects to the RSP, the mobile app receives an HTML interface for a gamepad and renders it on the screen. As the user presses buttons on the UI, the interface directs the mobile app to send commands to the RSP (*e.g.,* "`LEFT`"). These commands are then translated into keyboard presses that control the game.

This application demonstrates our system's ability to select appliances, render interfaces, and transmit commands without requiring either the mobile app or the RSP to know of each other *a priori*. Additionally, this application also shows how preferences can be shared through our system. In addition to the default controller shown in Figure 8a, the application also lets users choose between multiple controller layouts. This preference is then stored on the user's device, and can recreate their control settings when playing on a different computer.

Note that we did not use a special API to control the game. Instead, our RSP merely translates the user's controller commands into keyboard presses to give the user the impression that they are controlling the game directly. The ability to wrap Snap-To-It around existing applications is inspired by prior work in software overloading [14], and we believe that this is a general technique that can be used to instrument a wide range of existing hardware and software appliances.

Application #2: Digital Projector

Our second application shows how Snap-To-It can be used to upload/download content. We created an RSP that is linked to a conference room's multimedia control system. When users take a photo of the room's digital projector, the RSP provides them with an interface that lets them upload a presentation. The RSP then downloads and projects the presentation, and provides the user with slide deck controls (Figure 8b).

This application also shows how Snap-To-It can support multiple users at once. Our RSP differentiates between the user that uploaded the presentation and users that connect afterwards (*i.e.,* audience members). The RSP can then provide audience members with a separate interface to 1) see the currently visible slide, and 2) download a copy of the presentation. Although simple in concept, this application shows how Snap-To-It can support multiple user types through a single RSP. This allows our system to support a wide range of collaborative and cooperative activities—a capability not explicitly supported in existing remote control systems.

Application #3: Paint App

Our third application uses Snap-To-It to enhance a traditional software UI. By taking a photo of a paint application, users are provided with a series of drawing tools on their mobile display. The user can spread these controls across multiple devices (Figure 8c) so that they can access these controls without taking up room on the main display.

This application is heavily inspired by prior research in cross-device interactions [16]. However, our system builds on this work by allowing devices to share interfaces without having to install the same software or pair in advance. This lets users utilize any nearby device as an extended control surface, while simultaneously making these types of interactions easy to develop and deploy.

Application #4: Campus Map

Our last application uses Snap-To-It to interact with a non-computational object (*i.e.,* a campus map). When users photograph the map (Figure 8d), the RSP provides them with a digital version. They can then search for points of interest without having to remain near the physical object.

Note that the sign was not instrumented in any way. Instead, we simply gave the RSP a photo of the sign, and linked it to our institution's online map. In doing so, we show how Snap-To-It could one day be used to help users extract deep knowledge directly from their environment.

Experimental Evaluation

To evaluate Snap-To-It, we performed three small studies. The first compares Snap-To-It to SIFT, and shows how our system's use of multiple contexts helps it identify appliances with higher accuracy. The second compares Snap-To-It to QR codes, and shows how our system supports a wider range of angles and ranges. The third was conducted after a two-month live deployment, and shows how our system can continue to accurately identify appliances after an extended period of time.

Snap-To-It vs. SIFT Photo Recognition Accuracy

For the first study, we took eleven pictures of every printer (13), copy machine (3), and fax machine (2) in our institution's building (18 appliances total). The first five photos served as reference images, and were taken from the front and sides. The following six images served as our test set, and were taken by two experimenters on different days. The test images were taken from multiple angles from separate phones, and were

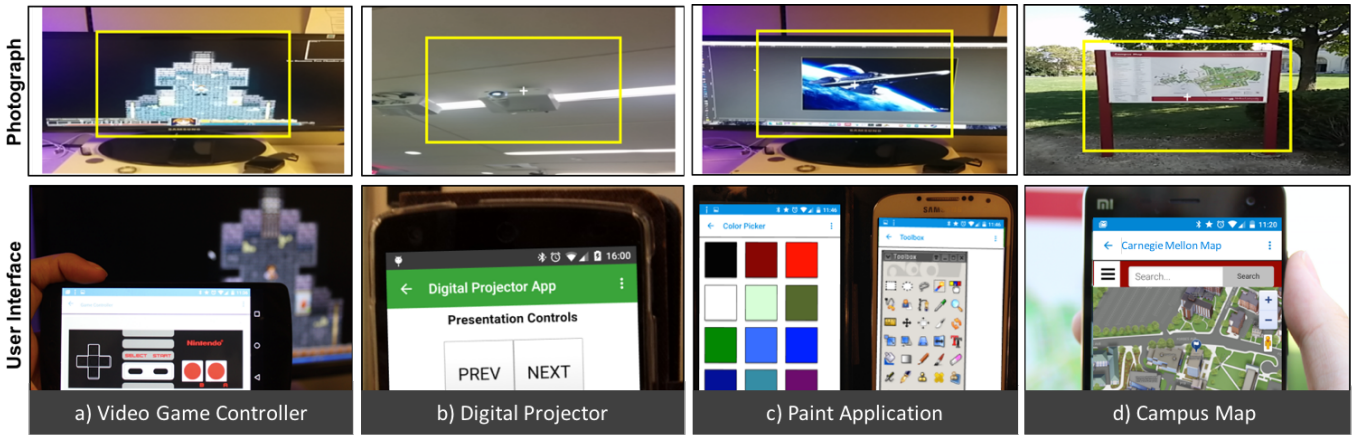


Figure 8. Example Snap-To-It applications. By taking a picture (top), our system can support a wide range of interactions (bottom).

List Position	SNAP TO IT 1 Ref Photo	SNAP TO IT 3 Ref Photos	SNAP TO IT 5 Ref Photos	SIFT ONLY 1 Ref Photo	SIFT ONLY 3 Ref Photo	SIFT ONLY 5 Ref Photo
Top 1	81.5%	87.0%	89.0%	67.6%	79.6%	85.2%
Top 3	87.0%	92.5%	94.6%	75.9%	92.6%	90.7%
Top 5	88.8%	95.3%	95.5%	83.3%	93.5%	92.5%

Table 1. Comparison of Snap-To-It and SIFT's photo recognition accuracy. Percentages show the number of times the correct appliance (out of 18) appeared in a list of the top 1, 3 and 5 matches.

not filtered to remove blurry images. For our evaluation, we created 18 RSPs (1 for each appliance), and provided each with 1, 3, or 5 reference photographs. We then had the RSPs evaluate each photo using both our system and pure SIFT.

As expected, Snap-To-It's ability to recognize an appliance depends on the quality and variety of its reference photos (Table 1). When we only provided a single reference image (taken from the front), there was an 82% chance that the correct appliance appeared at the top of the list and an 89% chance it was in the top five. With three and five reference photos, however, the chance of the correct appliance appearing at the top increased to 87% and 89%, respectively. The chance of the device appearing in the top five then exceeded 95%.

There are two important takeaways from this study. First, it shows that our system does not need a large number of reference images. Table 1 shows that our system's accuracy starts to level off with three reference images. This means that on-site administrators and/or developers only have to provide RSPs with a handful of images to obtain good performance.

Second, these results show how our system outperforms standard image recognition in a real-world setting. Of the 18 printers, six were the same make/model and ten were either located next to each other (and hence appear in each others' photographs) or placed in similar looking office rooms. While these similarities confused SIFT (especially in situations when the photo contains only a small amount of background scenery), Snap-To-It was able to more accurately differentiate between appliances that looked the same but faced different directions. Moreover, while SIFT came closer to Snap-To-It's accuracy with more reference photos, the results also reveal that our system is more likely to have the correct appliance at the *top* of the list. This makes our system more user-friendly in an opportunistic setting, and lets users be more confident that the appliance at the top of the list is the correct one.

Snap-To-It vs. QR-Code-Based Device Selection

Our second study compared device selection using Snap-To-It and QR codes. We placed 3 and 8-inch tall QR codes on the front and sides of a printer used in the previous study. We then tried to scan the code with a popular mobile app [4] from a variety of angles, starting from the front of the appliance and moving around at 20 degree increments from distances of three and six feet. At each location, we tried three times to see if the app could recognize any of the codes visible on the appliance within five seconds (the amount of time it typically takes Snap-To-It to identify the appliance). We then used Snap-To-It three times from the same location to see how it would perform (using three reference photos).

The results (Figure 9) show that QR codes are sensitive to both angle and distance. When the user was directly in front of the QR code, the app had no trouble scanning it. However, when the user looked at the code from an angle (40-60 degrees), the app was often unable to scan it reliably. Even after placing the code on multiple sides, there were some angles where none of the codes could be successfully scanned. Additionally, while a 3-inch code was sufficient when the user was one meter away, an 8-inch code was required for the app to recognize it from two meters. This may be acceptable for larger appliances such as the printer we used, but it is likely too big for smaller or narrower devices (*e.g.,* projectors). Snap-To-It was able to identify the printer from *all* of the locations at both distances. While the system's confidence varied depending on the angle (with the best results when angle and distance were closest to the three supplied reference images), the printer was always the top match from the 18 possible appliances. Our system's ability to collect and process reference photos from additional angles lets it become even more accurate, and suggests that Snap-To-It can be more reliable than using multiple QR codes.

Long Term Feasibility

While our first two studies show that Snap-To-It is accurate, we were also interested in seeing how well our system performs over time. To evaluate this, we deployed 24 RSPs in our environment for common appliances such as printers (18),

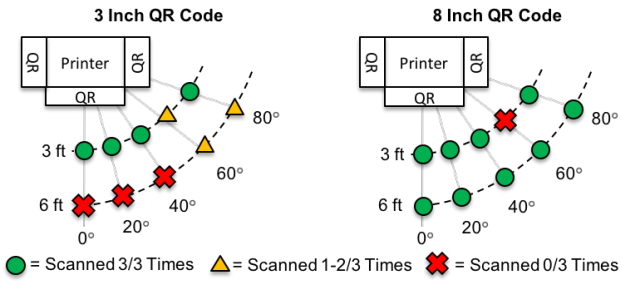


Figure 9. QR code scanning accuracy for a single appliance from multiple angles (codes placed on the front and sides).

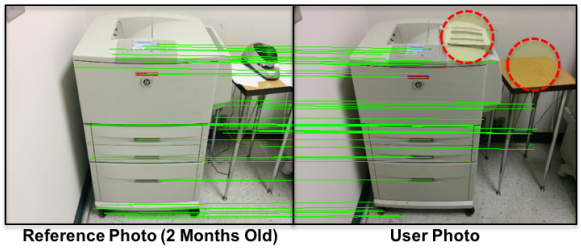


Figure 10. Sample from our long term feasibility study. Despite noticeable differences (circled), our system still identified the correct appliance using two month old reference photos (SIFT matches shown as lines).

computer displays (2), and digital projectors (4). We then published the Snap-To-It app in the Google Play store, and placed advertisements throughout our building letting users know that the app was available. During that time, a total of nine different users interacted with one or more appliances, and provided our RSPs with 25 new photographs (using the updating logic described earlier).

After running Snap-To-It for 59 days, we asked 16 new participants to interact with an appliance using our system. Our participants consisted of first-year Ph.D. students and visitors, as both groups were new to our environment and not affiliated with our work. For this study, we emailed them a document or a PowerPoint presentation, and asked them to use Snap-To-It to either print or project them on an appliance that they have never used before, respectively. We then observed them as they performed the task on their phones.

Even though nine participants interacted with appliances with two-month old reference photographs, our system was still able to identify the correct appliance in all 16 cases (Figure 10). Additionally, in all but one case, the correct appliance was at the top of the list; in the other case, the correct appliance was second. While additional studies are needed to verify that Snap-To-It works over longer stretches of time, these results show that it works despite small but frequent changes to an environment (*e.g.,* papers placed on a printer). This suggests that our system is robust enough to be used outside of the lab.

DISCUSSION

In this section, we highlight three important aspects of our system: usability, responsiveness, and hardware requirements.

Usability

There are two factors that affect Snap-To-It's usability:

Which Appliances are Compatible? One challenge with deploying a system like Snap-To-It is letting users know which appliances are selectable. This is a well-known problem for Ubicomp systems [9]. Although icons/markers can help, tagging every hardware/software appliance goes against the simplicity our system offers. At the same time, however, we know that users will stop using a system if it does not work consistently [13]. Consequently, it is important to provide *some* guidance so that users never feel like they are guessing.

We expect that users' uncertainty with Snap-To-It will decrease as more appliances become compatible. To expedite this process, however, we used multiple strategies. First, we posted a series of advertisements throughout our institution to let users know when they are entering/leaving a Snap-To-It enabled area. In addition, we are also experimenting with letting the user know when they are in Bluetooth range of a Snap-To-It compatible appliance, either by displaying a notification on the user's mobile device, or by allowing appliances to display an icon and/or beep. No approach works for all appliances. Collectively, however, they reduce the chances of users taking a photo when no services are available.

Security. Although Snap-To-It provides basic mechanisms for user authentication, it assumes that the appliances in an environment are trustworthy. This creates two security concerns. If a malicious RSP states that it strongly matches every user photograph, it can prevent users from connecting to other appliances. Additionally, since Snap-To-It uses JavaScript for its interfaces, rogue developers can potentially use it to execute malicious code on a user's device.

While concerning, it is important to remember that establishing trust is an issue in every networked system. One way to overcome this is to use blacklists to block rogue RSPs. A more robust solution, however is to mandate the use of digital certificates. The infrastructure to support this already exists, and would allow our mobile app to ignore advertisements from an RSP that has not been vetted by a trusted authority.

Responsiveness

On average, Snap-To-It takes 4 seconds to find an appliance from a photograph. This includes the time needed for our app to upload an image (1360ms) and for the RSP (running on a Macbook) to download it (1043ms), calculate its features (780ms), and compare it against one or more reference images (267ms each). Once the app has connected, an additional delay is required to download the interface. This delay varies depending on internet connectivity and complexity of the UI, but ranged between 1-3 seconds in our tests.

Although 5-7 seconds may seem long, the participants in our final study found it to be fast compared to the time it takes for a user to pair with an appliance or install a driver/app. This time could be reduced even further by including the photograph in the *request message*, as opposed to using URLs. Furthermore, we have also improved the responsiveness of our mobile app so that it displays results as they arrive, rather than forcing users to wait for all RSPs to respond. This, combined with our system's sub-100ms latency for sending/receiving commands, makes Snap-To-It fast enough for many interactive use cases.

Hardware Requirements

Snap-To-It's architecture assumes that each appliance runs its own RSP. However, many appliances currently lack the computational power to analyze photos on their own. This is especially the case when it comes to *calculating* SIFT features, as the process can take upwards of 28 seconds *per image* on low-powered hardware (*e.g.,* a Raspberry Pi).

Fortunately, Snap-To-It can take advantage of existing infrastructure to give users the impression that they are directly interacting with an appliance. All of the RSPs described in this paper were hosted on laptop/desktop computers; servers can provide similar functionality. Additionally, our approach does not require appliances to process images on their own. In our ongoing work, we have modified the mobile app so that it calculates the SIFT features of its photos *via* a web service prior to sending a request. This approach increases the time needed for the app to upload a photo (from 1360ms to 3564ms), making it slightly slower for RSPs that can already calculate SIFT features quickly. However, this strategy only requires low-powered RSPs to *compare* SIFT features (which takes 2.4s per image on a Raspberry Pi), thereby making our goal of running RSPs on appliances technically feasible.

RELATED WORK

The idea of using mobile devices to control appliances has been explored often in previous work. In [19], Hodes *et al.* developed an architecture that rendered arbitrary controls on a handheld device. Their system let users control appliances from a PDA, but required them to connect to a well known server and select devices from a list/map. This made the system cumbersome when the user was new to the environment, or when the list contained dozens of appliances.

Since then, researchers have developed numerous techniques to let users specify which appliance(s) they want to use (or "address" [9]). Prior work has experimented with laser pointers [8, 29, 31], handheld projectors [33], RFID tags [32], and head mounted wearables [12] to allow users to physically point, tap, or stare at the devices they want to select. While accurate, these techniques rely heavily on non-standard hardware (*e.g.,* transmitters/receivers), making them difficult to deploy outside of the lab. Other systems use mounted cameras [10, 15] and magnetometers [36, 35] in order to determine where a user is and what appliance(s) he/she is pointing at. These systems prevent *every* appliance from having to be instrumented, but only work in preinstrumented areas. Finally, researchers have explored using smartphone cameras to select objects, either by affixing fiducial markers to each appliance [26, 18, 20], or by utilizing image recognition [11, 21] or machine learning [24].

While Snap-To-It also relies on visual recognition, our approach differs in three ways. First, our system does not require the user's device to have *a priori* knowledge of an image processing server, as is required by systems such as [26, 18, 20]. Instead, it uses multicasting to share images, which lets it discover and connect to appliances in any network-connected environment. Secondly, our approach does not limit users to appliances that they already own (as was done in [11]), require the user's device to know which devices are selectable beforehand [24], or require the user to keep the appliance in view in order to use it (as is the case with augmented reality systems). This lets us support a wider range of opportunistic interactions through a single system. Finally, while other systems rely entirely on images to identify appliances, Snap-To-It also takes the user's location and device orientation into consideration. This allows our system to more accurately differentiate between similar looking appliances, thereby improving our system's ability to be used in real-world environments.

In addition to selection techniques, researchers have also looked into the *types* of interactions that are possible using a mobile device. While much work still focuses on remote control functionality (*e.g.,* rendering control interfaces [27], supporting multiple interaction modalities [28], sending remote commands [34]), there is now increasing interest in other types of interactions. Pick and Drag and Drop, for example, supports cross-device interactions by allowing users to "grab" a file from one machine and move it to another. The Digifieds system, on the other hand, uses smartphones to modify public displays, allowing users to easily post a quick note or reposition content [6]. Deep Shot lets devices easily transfer program tasks between one another, thus allowing users to start a task on one device and finish it on another [11]. In the Open Project, researchers allowed users to project their phone's screen on unused public displays, thereby allowing them to share content with friends or strangers. Finally, products such as AllJoyn [2], Bonjour/ZeroConf [1], and Universal Plug and Play [3] are not designed for a single interaction, but instead provide a general solution to allow appliances to communicate with each other without prior coordination. However, these systems still require users to know a device's name or capabilities before they can (manually) select them.

Snap-To-It contributes to this body of work by helping us better understand the types of interactions users *want* to have, and how we can support them through a single system. We do not claim that our system is feature complete. It does, however, support a wide range of use cases, and its ability to be deployed in real-world environments provides a solid foundation for future research.

CONCLUSION

This paper has presented Snap-To-It, a system that allows users to select and control nearby appliances using their mobile device's camera. Our work takes an important step towards increasing a user's ability to interact with new and unfamiliar appliances. Yet while Snap-To-It's interaction technique already works with many appliances, it can be improved further. One idea is to use Snap-To-It to interact with multiple appliances at the same time. For example, it would be interesting to allow users to connect to multiple appliances (*e.g.,* all of the lamps in a room) by photographing them all at once. Additionally, while our system already uses background scenery and compass orientation to differentiate between objects that look the same, it would be interesting to incorporate additional types of context (*e.g.,* barometric pressure to estimate elevation) into the recognition pipeline to further increase accuracy.

ACKNOWLEDGMENTS

This work is supported by the Software Engineering Institute, Carnegie Mellon University.

REFERENCES

1. 2014. Bonjour. (2014). https://www.apple.com/support/bonjour/.
2. 2014. A Common Language for the Internet of Everything - AllJoyn. (2014). https://www.alljoyn.org/.
3. 2014. UPnP Forum. (2014). http://www.upnp.org/.
4. 2015. Barcode Scanner. (2015). https://play.google.com/store/apps/details?id=com.google.zxing.client.android&hl=en.
5. 2015. QRcode.com. (2015). http://www.qrcode.com/en/index.html.
6. Florian Alt, Alireza Sahami Shirazi, Thomas Kubitza, and Albrecht Schmidt. 2013. Interaction techniques for creating and exchanging content with public displays. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*. ACM Press, New York, New York, USA, 1709. DOI: http://dx.doi.org/10.1145/2470654.2466226
7. Debasis Bandyopadhyay and Jaydip Sen. 2011. Internet of Things: Applications and Challenges in Technology and Standardization. *Wireless Personal Communications* 58, 1 (April 2011), 49–69. DOI: http://dx.doi.org/10.1007/s11277-011-0288-5
8. Michael Beigl. 1999. Point & click-interaction in smart environments. In *Handheld and Ubiquitous Computing*. Springer, 311–313. DOI: http://dx.doi.org/10.1007/3-540-48157-5_31
9. Victoria Bellotti, Maribeth Back, W Keith Edwards, Rebecca E Grinter, Austin Henderson, and Cristina Lopes. 2002. Making Sense of Sensing Systems: Five Questions for Designers and Researchers. 1 (2002), 415–422. DOI:http://dx.doi.org/10.1145/503376.503450
10. Matthias Budde, Matthias Berning, Christopher Baumgärtner, Florian Kinn, Timo Kopf, Sven Ochs, Frederik Reiche, Till Riedel, and Michael Beigl. 2013. Point & control–interaction in smart environments: you only click twice. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM, 303–306. DOI: http://dx.doi.org/10.1145/2494091.2494184
11. Tsung-Hsiang Chang and Yang Li. Deep shot: a framework for migrating tasks across devices using mobile phone cameras. In *CHI '11*. New York, New York, USA, 2163–2172. DOI: http://dx.doi.org/10.1145/1978942.1979257
12. Yu-Hsiang Chen, Ben Zhang, Claire Tuna, Yang Li, Edward A Lee, and Bjï£¡rn Hartmann. 2013. *A Context Menu for the Real World: Controlling Physical Appliances Through Head-Worn Infrared Targeting*. Technical Report UCB/EECS-2013-200. EECS Department, University of California, Berkeley. DOI: http://dx.doi.org/No.UCB/EECS-2013-182
13. Fred D Davis. 1985. *A technology acceptance model for empirically testing new end-user information systems : theory and results*. Ph.D. Dissertation. http://dspace.mit.edu/handle/1721.1/15192
14. James R. Eagan, Michel Beaudouin-Lafon, and Wendy E. Mackay. 2011. Cracking the Cocoa Nut: User Interface Programming at Runtime. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 225–234. DOI: http://dx.doi.org/10.1145/2047196.2047226
15. David Fleer and Christian Leichsenring. 2012. MISO: a context-sensitive multimodal interface for smart objects based on hand gestures and finger snaps. In *UIST Adjunct Proceedings '12*. New York, New York, USA, 93–94. DOI:http://dx.doi.org/10.1145/2380296.2380338
16. Peter Hamilton and Daniel J. Wigdor. 2014. Conductor. In *CHI '14*. New York, New York, USA, 2773–2782. DOI:http://dx.doi.org/10.1145/2556288.2557170
17. Jonathon S Hare, Sina Samangooei, and David P Dupplaw. 2011. OpenIMAJ and ImageTerrier: Java libraries and tools for scalable multimedia analysis and indexing of images. In *MM '11*. New York, NY, USA, 691–694. DOI: http://dx.doi.org/10.1145/2072298.2072421
18. Valentin Heun, Shunichi Kasahara, and Pattie Maes. 2013. Smarter Objects: Using AR Technology to Program Physical Objects and Their Interactions. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems (CHI EA '13)*. ACM, New York, NY, USA, 961–966. DOI: http://dx.doi.org/10.1145/2468356.2468528
19. Todd D. Hodes, Randy H. Katz, Edouard Servan-Schreiber, and Lawrence Rowe. 1997. Composable ad-hoc mobile services for universal interaction. In *MobiCom '97*. New York, New York, USA, 1–12. DOI:http://dx.doi.org/10.1145/262116.262121
20. Can Liu, Stephane Huot, Jonathan Diehl, Wendy Mackay, and Michel Beaudouin-Lafon. 2012. Evaluating the Benefits of Real-time Feedback in Mobile Augmented Reality with Hand-held Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2973–2976. DOI: http://dx.doi.org/10.1145/2207676.2208706
21. Qiong Liu, Paul McEvoy, Don Kimber, Patrick Chiu, and Hanning Zhou. 2006. On Redirecting Documents with a Mobile Camera. In *Workshop on Multimedia Signal Processing*. 467–470. DOI: http://dx.doi.org/10.1109/MMSP.2006.285352
22. D.G. Lowe. 1999. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, Vol. 2. 1150–1157. DOI: http://dx.doi.org/10.1109/ICCV.1999.790410

23. Friedemann Mattern and Christian Floerkemeier. 2010. From the Internet of Computers to the Internet of Things. *From Active Data Management to Event-Based Systems and More* (Jan. 2010), 242–259. http://dl.acm.org/citation.cfm?id=1985625.1985645

24. Simon Mayer, Markus Schalch, Marian George, and Gábor Sörös. 2013. Device recognition for intuitive interaction with the web of things. In *UbiComp '13 Adjunct Proceedings*. New York, New York, USA, 239–242. DOI: http://dx.doi.org/10.1145/2494091.2494168

25. Daniele Miorandi, Sabrina Sicari, Francesco De Pellegrini, and Imrich Chlamtac. 2012. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks* 10, 7 (Sept. 2012), 1497–1516. DOI: http://dx.doi.org/10.1016/j.adhoc.2012.02.016

26. Matei Negulescu and Yang Li. 2013. Open project: a lightweight framework for remote sharing of mobile applications. In *UIST '13*. New York, New York, USA, 281–290. DOI: http://dx.doi.org/10.1145/2501988.2502030

27. Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. 2002. Generating remote control interfaces for complex appliances. In *UIST '02*. New York, New York, USA, 161–170. DOI: http://dx.doi.org/10.1145/571985.572008

28. Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. 2006. UNIFORM: automatically generating consistent remote control user interfaces. In *CHI '06*. New York, New York, USA, 611–620. DOI: http://dx.doi.org/10.1145/1124772.1124865

29. Shwetak N Patel and Gregory D Abowd. In *UbiComp '03*. Springer, 200–207. DOI: http://dx.doi.org/10.1007/978-3-540-39653-6_16

30. Jukka Riekki, Ivan Sanchez, and Mikko Pyykkönen. 2008. Universal Remote Control for the Smart World. In *Ubiquitous Intelligence and Computing (Lecture Notes in Computer Science)*, Frode Eika Sandnes, Yan Zhang, Chunming Rong, Laurence T. Yang, and Jianhua Ma (Eds.), Vol. 5061. Berlin, Heidelberg, 563–577. DOI: http://dx.doi.org/10.1007/978-3-540-69293-5

31. Matthias Ringwald. 2002. UbiControl: Providing New and Easy Ways to Interact with Various Consumer Devices. In *UbiComp '02*. 81,82.

32. Ichiro Satoh. 2011. A Management Framework for Context-Aware Multimedia Services.. In *DMS*. 165–170.

33. Dominik Schmidt, David Molyneaux, and Xiang Cao. 2012. PICOntrol: using a handheld projector for direct control of physical devices through visible light. In *UIST '12*. ACM Press, New York, New York, USA, 379–388. DOI: http://dx.doi.org/10.1145/2380116.2380166

34. Chuong Cong Vo. 2013. *A Framework for a Task-Oriented User Interaction with Smart Environments Using Mobile Devices*. Ph.D. Dissertation. La Trobe University.

35. Hanno Wirtz, Jan Rüth, Martin Serror, Jó Ágila Bitsch Link, and Klaus Wehrle. 2014. Opportunistic interaction in the challenged internet of things. *Proceedings of the 9th ACM MobiCom workshop on Challenged networks - CHANTS '14* (2014), 7–12. DOI: http://dx.doi.org/10.1145/2645672.2645679

36. Jiahui Wu, Gang Pan, Daqing Zhang, Shijian Li, and Zhaohui Wu. 2010. MagicPhone: pointing & interacting. In *UbiComp '10*. New York, New York, USA, 451–452. DOI: http://dx.doi.org/10.1145/1864431.1864483