

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

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

Leave Authors Anonymous
for Submission
City, Country
e-mail address

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, currently there is a lack of support for this capability, forcing the users to always have to 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 broadcasts the image of the appliance they want to interact with over a local area network. Appliances then use this image (along with the user's location and compass heading) 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 remote control 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 95.5% accuracy, and demonstrate through a two-month deployment that our approach is robust to changes in the environment over time.

ACM Classification Keywords

H5.2 Information Interfaces and Presentation (*e.g.* HCI): User Interfaces: Input Devices and Strategies, Interaction Styles

Author Keywords

Internet of things, mobile interaction

INTRODUCTION

Over the past decade, the rapid proliferation of "smart" appliances (*e.g.*, thermostats, printers, speakers) has created new opportunities for users to interact and engage with technology

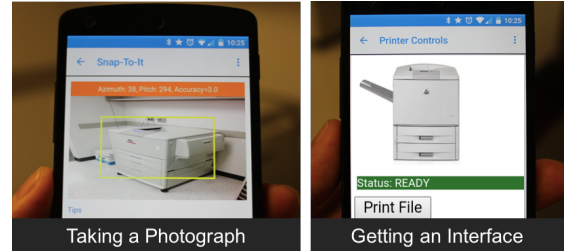


Figure 1. A user takes a photo of the appliance he wants to control (left). Our system shares this image (along with the user's location and camera orientation) with nearby appliances. The user's device connects to the appliance that best matches the image, and receives a custom UI (right).

in meaningful ways. Taking advantage of this increased connectivity, however, is a well-known challenge [7, 18, 21, 23, 29]. If, for example, users want to use a network-connected printer, they have to know its IP address and verify that they have the correct drivers installed. Likewise, if users are at a party and want 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 we use frequently, it makes one-time or spontaneous use impractical. This discourages users from interacting with new or unfamiliar appliances, and can even cause them to go out of their way to seek alternative solutions (*e.g.*, asking a friend or stranger to print a document on their behalf).

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 allows users to select a nearby appliance by "snapping" a photo of it. Our app then broadcasts this image (along with the user's location and compass heading) 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, 25, 32, 28, 33]; Snap-to-It expands on their work in three important ways.

First and foremost, our work broadens our understanding of the types of interactions users would like to opportunistically perform using a mobile device. Prior to developing Snap-To-It, we conducted two technology probes (with 10 and 28

participants, respectively) 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. Through these probes, we found that participants preferred selecting appliances by taking a photograph (as opposed to scanning QR codes or selecting them from a list). Additionally, our probes identified six categories of general use cases for how users would want to interact with their environments, including some that have yet to be explored by other remote control systems, such as 1) the ability to interact with disconnected appliances, and 2) migrate settings between appliances. Snap-To-It's design is directly informed by these findings, providing us with a research platform that not only shows what types of interactions are *possible* using a mobile device, but what are actually *desired* by end users.

Secondly, Snap-To-It provides a truly opportunistic way of interacting with a wide range of appliances. Unlike other approaches, which require the user to download a list of nearby appliances [18] or connect to a well-known server [11], our system uses network broadcasting to identify an appliance based on its photo. This makes our system particularly useful when users are visiting a location for the first time, but need to use an appliance once or spontaneously. Additionally, while prior work has shown how mobile devices can be used to interact with hardware [27, 31] and software [12], they have focused on one or the other. In contrast, Snap-To-It can compare user-taken photographs against stock images or real-time screenshots. This allows our system to 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* a camera. Our system's middleware automatically processes incoming photos, establishes connections with user devices, and delivers arbitrary HTML/JavaScript-based interfaces. This allows developers to incorporate Snap-To-It's functionality into their 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 next section, we describe our technology probes, and show how user responses have directly influenced Snap-To-It's design. Afterwards, we describe Snap-To-It's architecture, and present four prototypes that highlight its capabilities. Through two studies, we show that Snap-To-It is accurate enough 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 create a system that allows users to quickly select and interact with any object in their physical environment. Yet while prior work has shown that such a



Figure 2. Appliances used in our first probe.

feature is *technically* possible, these works mainly focused on the challenges associated with connecting to devices and receiving/rendering custom interfaces [18, 25]. 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 system that will actually be useful.

To fill these gaps, we conducted two technology probes. The first probe investigates how users would like to select an appliance using their mobile device. The second looks at how users would like to interact with that appliance once it is selected. In this section, we describe both probes. We then show how our findings have informed Snap-To-It's final 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 conducted a study with 10 participants (6 male, 4 female; 22-42 years old; mix of novices and experts with mobile phones) in which we had them try out a variety of techniques firsthand. Prior to the study, we gave participants a list of 7 possible techniques that could be implemented on existing smartphones, which included: taking a photograph of the appliance, scanning a QR code, "bumping" the phone against the appliance, 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 were compatible with the widest range of appliances: taking a picture, scanning a QR code, and selecting from a list.

Our probe was conducted in one of our institution's common areas. We selected a combination of appliances (Figure 2) that visitors might need to interact with, making sure they varied in type (*i.e.*, hardware and software) and physical distance from the user (*i.e.*, near and far). Prior to the study, we placed QR codes on each appliance using the size and positioning guidelines established in [5], and created a list of every appliance within 30 feet of the room (*i.e.*, Bluetooth range). We then provided participants with an app that let them take a photo (using the camera app), scan a QR code (using a QR code reader app [4]), and select an appliance from an alphabetized list. Participants were paid \$15 to try each technique (in random order), and the entire study took 30 minutes.

Feedback about the list-based technique was overwhelmingly negative. Although participants had initially stated that they would not mind selecting appliances from a list, they found that doing so was difficult in practice because they did not know what each appliance was named; while this problem might be overcome by choosing user-friendly names rather than the ones assigned by our network administrators (*e.g.*,

"Zircon", "Pewter"), prior research suggests that there is no single naming convention that works well with every user [13]. Although our list only had 11 items, participants frequently reported that they felt like they were "scrolling and scrolling and scrolling" (P1). This led them to unanimously rate the list as the worst of the three techniques.

Feedback from the QR code technique was similarly negative. Despite already knowing how to scan a QR code, eight had difficulty selecting our printer and/or digital projector because their phone's camera had trouble focusing on the code. While participants did not mind seeing the code on most devices, they noted that having a persistent code on a screen was distracting. As a result, while participants found the QR code to be better than the list, only one rated it as the best overall solution.

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 and explain: "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 noted that the difference between the QR code and camera method was negligible in cases when the appliance being selected is physically close. However, for software applications and objects farther away, participants agreed that the photo 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 participants stated that they would *prefer* to select an appliance by taking a photograph. Despite the relatively small number of participants in our probe, the uniformity of their responses, combined with their mix of expertise with mobile devices, led us to believe that these findings are generalizable.

Probe 2: Types of Interactions

For our second probe, we were interested in seeing 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 to 60 years old; mix of novice and expert mobile phone users) to provide us with a "wish list" of the interactions that they would like to perform. For this probe, we provided participants with an Android app that allowed them to take photos of appliances and annotate how they would use them. We then paid them \$20 to take photos for a week as they went about their normal routine.

Through this process, we collected a total of 195 photographs. Each photo was categorized according to 1) the type of interaction being performed, and 2) the technologies needed to support these interactions. This information was then directly used to identify general use cases and derive critical technical requirements, respectively.

Use Case Categories

Participants had a wide range of ideas as to how they wanted to interact with other 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). Many subjects 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. The reasons for doing so widely varied. 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 suggestion to be particularly insightful, as it identified 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 for Snap-To-It, 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 Snap-To-It 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


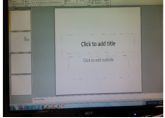


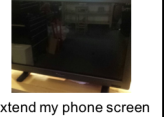







Use Case Category	Quick Control	Long Distance Operation	Upload/Download Content	Secure Transactions	Increase Intelligibility	Share Preferences
Motivation	Connect and use a nearby device.	Operate a device from a remote location.	Allows users to post and receive data.	Be able to securely send information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

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

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 transfer preferences to new appliances without having to input them manually.

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 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 [18, 2] *or* software [12]. 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, whether it is to provide users with an interactive map, or a real-time feed of closed-captioning. 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. From our probes, we know that there are several obvious use cases that our system *needs* to support (*e.g.*, sending commands to remote appliances). At the same time, however, 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 systems such as Snap-To-It need to be flexible enough to accommodate these use cases as they are discovered.

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. Additionally, Snap-To-It also provides an extensible middleware that makes it easy to add our system's functionality to new or existing appliances. This allows our system 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 broadcasts a multicast datagram containing a URL to the photo, as well as the phone's location and compass heading. 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 custom 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 first (and primary) way is by using the mobile device's camera (**R1**). Each time a user takes a photo of an appliance, our

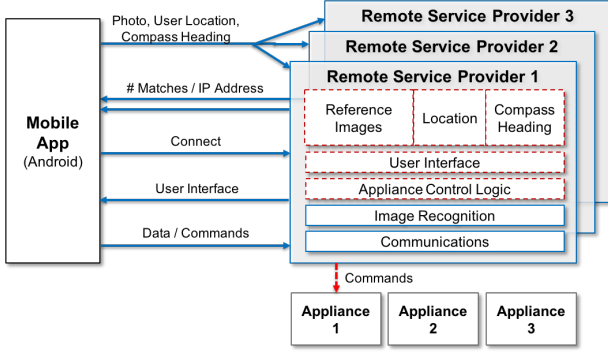


Figure 4. High level system architecture. Solid boxed components are provided by Snap-To-It, while dotted boxes are provided by developers and/or administrators.

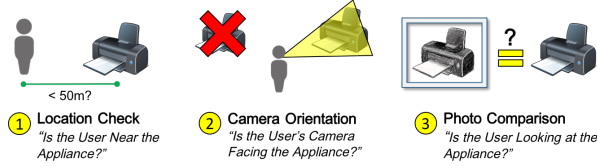


Figure 5. Snap-To-It's appliance recognition process.

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 camera heading (azimuth, pitch, roll). It then broadcasts a *request message* containing the photo URL, location, and heading, 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 (more details on the recognition process are provided in the following section). Otherwise, the app provides users with a list of the five highest matching appliances to select from.

Alternatively, our mobile app also allows users to select appliances over long distances (in support of **R3**). When users connect to an appliance, they are given the option of adding it to their favorites list. This causes the app to store its connection details, and allows the user to reconnect to that appliance without having to retake its photo or be nearby.

Recognizing an Appliance

Snap-To-It matches user requests to specific appliances in three stages (Figure 5). In the first stage, the RSP extracts the latitude/longitude from the request message and calculates the distance between itself and the user. The RSP then only 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 allows our system to 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 camera is facing the same way. 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 [20]. 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. This allows the algorithm to provide us with good results, even when the images are taken from slightly different angles.

In order for the above process to work, every RSP must know where it is located, what direction it is facing, and what the appliance it represents looks like. For appliances with a GPS, compass, and display, this information can be obtained automatically by directing the RSP to periodically poll its sensors and take screenshots of its display, respectively. Many appliances (e.g., printers), however, lack the ability to obtain this information on their own. To support them, developers and/or administrators will collect this information *via* our mobile app. They can then provide these photos (and their associated meta-data) to the RSP. While this approach is more cumbersome, we have empirically found that it yields good results with as few as three reference photos (one taken from the front, and two from opposite 45 degree angles). This makes it easy for developers to associate an RSP with any type of hardware, software, or non-computational object (satisfying **R2**).

Additionally, our process requires every reference photo contain some information about an appliance's surroundings so that it can differentiate between appliances that are of the same make and model. To support 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. This lets our system differentiate between appliances that look similar, but have different surroundings. Our study has found that this strategy allows Snap-To-It to identify the correct appliance more than 95% of the time (an examination of Snap-To-It's accuracy is included in our validation).

Updating/Maintaining Reference Photos

Snap-To-It's accuracy is closely tied to 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 accuracy. However, reference photos become out of date as the environment changes. This results in decreased accuracy, and forces developers and/or administrators to continually resupply each RSP with fresh pictures.

Snap-To-It overcomes this problem by utilizing user photographs. Each time a user connects to an appliance, the photo they used to select the appliance is compared to the reference

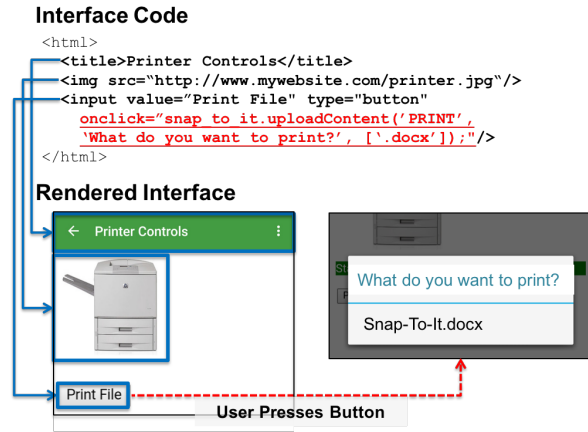


Figure 6. Sample controls for a printer. When users press the "Print File" button, it invokes our JavaScript API (red/underlined) to select and upload a file (bottom right).

images on record. If the photo is taken at a similar angle to an existing photo (e.g., within 10 degrees azimuth/pitch/roll), the RSP replaces it with the new image. Otherwise, the system adds the photo to its current library (up to a specified limit). By allowing RSPs to update their reference images, RSPs can learn how an appliance's environment is changing over time. This keeps them up to date, while simultaneously allowing them to recognize appliances from a wider range of angles.

Yet while this approach provides RSPs with new photos, it also introduces the possibility of RSPs receiving blurry or inaccurate images (i.e., an image of another appliance). To address this, RSPs periodically ask users (when they connect) to rate their reference images for accuracy and clarity. By leveraging the user community's responses, RSPs are able to detect and "weed out" low quality images from their library. This allows the RSP to self-maintain a library of relevant photos without the need for constant administration.

Defining User Interfaces

Based on 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 precanned action (Figure 6).

This approach lets Snap-To-It perform operations that are normally inaccessible (or difficult to perform) from within 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

```
public class Printer_RSP extends RemoteServiceProvider
{
  public Printer_RSP () {
    // Step 1: Add Reference Photo(s)
    addPhoto("Printer1.jpeg", LOCATION, AZIMUTH, PITCH, YAW);
  }

  public String getInterface(User u) {
    // Step 2: Create and Deliver a User Interface
    return "<html> . . . </html>";
  }

  public void onRemoteCommand(User u, Command c) {
    // Step 3: Process Commands Sent by Mobile App
    if (c.getCommand().equals("PRINT")) {
      print(c.getURL());
    }
  }
}
```

Figure 7. Pseudocode for a remote service provider.

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 them utilize settings that were created by other appliances. This lets appliances configure themselves (e.g., transfer exercise settings) based on the user's interactions with similar and/or complementary appliances, and reduces the need for manual configuration.

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. Similarly, an RSP for a laptop or car might want to let its owner log in or unlock the doors using his/her phone, respectively, but prevent others from doing the same.

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 comes with an extensible middleware that 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 if the interface is stored online, or if a secure channel is required, respectively.

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 obtained 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 uses UDP multicast and TCP sockets to broadcast requests and send commands, respectively. For now, RSPs are limited to desktop operating systems, as there is currently no way to programmatically take screenshots in Android without severely disrupting the end user experience. In the future, however, we hope to eliminate this limitation so that we can increase the range of interactions we can support.

VALIDATION

We validate Snap-To-It in two parts. First, we present four examples that were built 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 present four examples that were built using Snap-To-It (Figure 8a-d). Each example is inspired by one or more of the general use cases, and demonstrates the range of applications (hardware and software) our system supports.

Example #1: Game Controller

Our first example shows how we can use 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, our app broadcasts a link to the image (as well as the user's heading and location) over the network. 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, it receives an HTML interface for a gamepad, and renders it on the user's phone. As the user presses buttons on the UI, the interface directs the mobile app to send commands to the RSP (*e.g.*, "LEFT"). The RSP then translates these commands into keyboard presses that control the game.

This example 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 example also shows how we can share preferences through our system. In addition to the default controller shown in Figure 8a, our application also lets users choose between multiple controller layouts. This preference is then stored on the user's device, and can recreate her control settings when playing on a different computer.

It is important to 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, thereby giving the user the impression that she is controlling the game directly. This ability to wrap Snap-To-It around existing services is an important feature of our system, and we believe that this technique can be used to instrument a wide range of existing hardware and software appliances.

Example #2: Digital Projector

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

This application also shows how Snap-To-It can support multiple users at once. Our RSP can differentiate between the user that started the presentation, and those that connect afterwards (*i.e.*, audience members). The RSP then provides 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 example 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.

Example #3: Paint Application

Our third example 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), thereby allowing them to access these controls without taking up room on the main display.

This example 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 allows users to potentially utilize any nearby device as an extended control surface, while simultaneously making these types of interactions easy to develop and deploy.

Example #4: Campus Map

Our final example uses Snap-To-It to interact with a non-computational object. Here, we created an RSP that is linked to a physical map of our campus (Figure 8d). When users take a picture of the map, our RSP provides them with an interactive map. They can then view the map and search for particular points of interest (*e.g.*, buildings, food court) without having to stand near the sign.

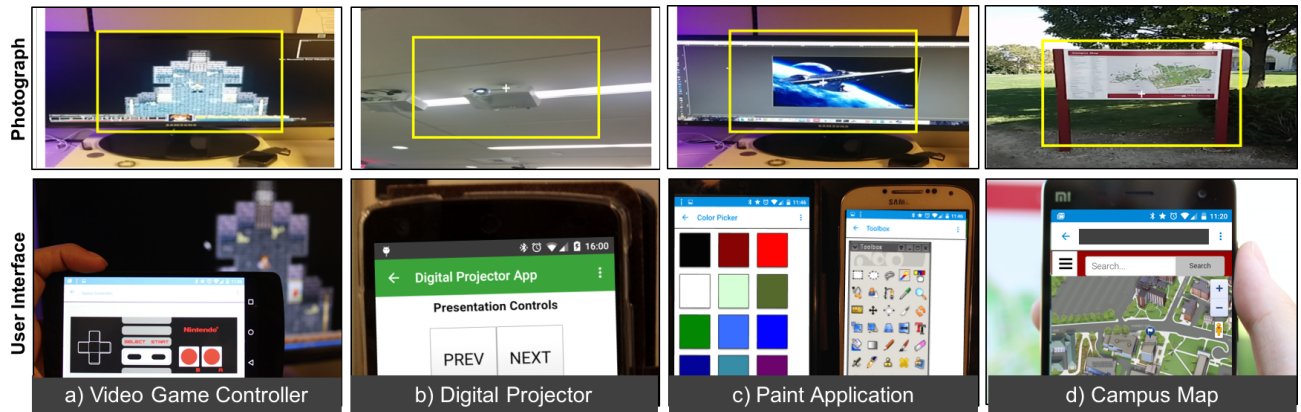


Figure 8. Snap-To-It examples. By taking a picture (top), our system can support a wide range of interactions (bottom).

	SNAP-TO-IT			SIFT ONLY		
	1 Ref Photo	3 Ref Photo	5 Ref Photo	1 Ref Photo	3 Ref Photo	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. Appliance recognition accuracy using Snap-To-It and SIFT for 18 real-world appliances.

For this example, we did not instrument the sign in any way. Instead, we simply associated a photograph of the map with an RSP, and linked it to our institution’s online map. This ability to link arbitrary objects to digital services/data is an important aspect of our system, and shows how Snap-To-It could one day be used help users extract deep knowledge (*e.g.*, help manuals, instructions) from their environment.

Appliance Recognition Accuracy

To evaluate Snap-To-It’s accuracy, we performed two studies using real-world appliances. Our first study was conducted under controlled conditions, and shows how our system is able to find the correct appliance over 95% of the time. Our second study 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.

Controlled Study

For the first study, we took eleven photographs of every public printer (13), copy machine (3), and fax machine (2) in one of our institution’s buildings (18 appliances total). The first five photographs 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 a wide range of angles from separate cameras, and were 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 provide a single reference image

(taken from the front), there is an 82% chance that the correct appliance will appear at the top of the list. When we increase the reference photos to include 2 or 4 side shots, however, the chances of the correct appliance appearing at the top increases to 87% and 89%, respectively; similarly, the chances of the device appearing in the top five exceeds 95%.

These results are significant for two reasons. First, they show 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 yield good performance.

Second, these results show how our system outperforms standard image recognition in a real-world setting. Of the 18 printers used in this study, six are of the same make/model, and ten are either located in the same area (and hence appear in each others’ photographs), or placed in similar looking office rooms. Yet while these similarities cause SIFT to mistake one appliance for another (especially in situations when the photo contains only a small amount of background scenery), Snap-To-It is able to more accurately differentiate between appliances that looked the same, but face different directions (location was not useful in this study since all of the appliances are in the same building). Moreover, while our results show that SIFT eventually comes close to Snap-To-It’s accuracy with enough reference photos, they also show that our system is more likely to have the correct appliance at the *top* of the list as opposed to “somewhere” in the top five. This makes our system more user-friendly in an opportunistic setting, and allows users to be more confident that the appliance at the top of the list is in fact the one they are trying to interact with.

Long Term Feasibility Evaluation

While our controlled study shows that Snap-To-It is accurate, we were also interested in seeing how well our system performed after some time had passed, to evaluate its robustness to possible changes in the environment. To evaluate this, we first deployed 24 RSPs in our environment for common appliances such as printers (18), 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 9 different users interacted with one or

more appliance, and provided our system with 25 new photographs (using the updating logic described earlier).

After collecting photographs for 59 days, we conducted a second study in which we asked 16 participants to interact with an appliance using our system. To make this evaluation as opportunistic as possible, our participants consisted of first-year Ph.D. students and visitors, as both groups were new to our environment and its appliances. For this study, we either 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.

The results from this study are encouraging. Despite the fact that 9 of our participants interacted with appliances with two month old reference photographs, our system was still able to identify the correct appliance in all 16 cases. Additionally, in all but one case, the correct appliance appeared at the first position in the list; in the one exceptional case, the correct appliance appeared second in the list. While additional studies are required to verify our system works over longer stretches of time (*e.g.*, 6 months, a year), these results show that our system can work well after an appliance is deployed. This makes Snap-To-It feasible for deployment in a wide range of real-world environments.

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 in deploying Snap-To-It has been helping users know which appliances are selectable through our system. This is a well-known problem for Ubicomp systems [9]. Although prior work has shown that using icons/markers can help users see with which appliances they can interact, the idea of tagging every appliance goes against the simplicity we were trying to achieve through our system. At the same time, however, it is known that users will stop using a system if it does not work consistently [14]. Consequently, it is important to provide *some* guidance to users so that they 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 ways of 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 alter their behavior (*e.g.*, displaying an icon, beeping). Each approach has its downsides (*e.g.*, decreased battery life). Collectively, however, these techniques help users understand which appliances are compatible with our system, thereby preventing them from 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 ever connecting to other appliances. Additionally, since our system relies on JavaScript to convert button presses to commands, rogue developers can use Snap-To-It to execute malicious code on a user's device.

While these issues are concerning, it is important to remember that malicious users/appliances are a problem in any networked system. Thus, our system can benefit from existing solutions. The simplest solution is to use blacklists to block rogue RSPs. A more robust solution, however is to mandate the use of digital certificates. This approach is commonly used to verify Internet websites, and would allow our mobile application to ignore advertisements from an appliance that has not been vetted by a trusted authority. More importantly, the infrastructure for this type of server-side authentication already exists, and Snap-To-It could utilize it while remaining transparent to users.

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 extremely fast compared to the time it takes for a user to pair to 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.

Hardware Requirements

When we conceived of Snap-To-It, we assumed that each appliance would run its own RSP. While this long-term goal has not changed, many appliances 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, there are two mitigating factors. First, 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. Second, 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

message. Preliminary tests show that this roughly triples 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 setup 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 in numerous works. In [18], Hodes *et al.* developed an architecture that rendered arbitrary controls on a handheld device. Their system allowed users to control appliances from their 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 proposed numerous techniques to allow users to more easily specify which appliance(s) they want to use (or "address" [9]). To date, researchers have experimented with laser pointers [8, 27, 29], handheld projectors [31], RFID tags [30], and head mounted wearables [13] 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 used mounted cameras (*i.e.*, Kinect) [11, 15] and magnetometers [34, 33] 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 utilized handheld cameras (such as those commonly available on smartphones) to select objects, either by affixing fiducial markers to each object [24, 10], or by utilizing image recognition [12, 19] or machine learning algorithms [22]. These approaches require extensive infrastructure support, but offer a simple and intuitive way of selecting appliances.

Snap-To-It builds on prior work in that it also uses image recognition to select appliances. However, our approach differs in three ways. First, our system does not rely on a well-known server to process imagery, as is required by many photo-based systems [19]. Instead, our system utilizes network broadcasting to transmit images, thereby allowing it to discover and connect to appliances in any network-connected environment. Secondly, our approach does not limit users to appliances that they own (as was done in [12]), or require that the user's device know which devices are selectable beforehand [22]. This makes our approach better-suited for situations when a user needs to find and connect to an appliance opportunistically. Finally, while photo-based systems rely entirely on images, Snap-To-It also takes the user's location and camera orientation into consideration when matching a user request to an appliance. 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 many works still focus on remote

control functionality (*e.g.*, rendering dynamic interfaces [25], supporting multiple interaction modalities [26], sending remote commands [32]), there is now increasing interest in using mobile devices to interact with appliances in other ways. Pick and Drag and Drop, for example, uses a smartphone as a cross-device interaction tool, allowing users "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 allows devices to easily transfer program tasks between one another, thus allowing users to start a task on one device and finish it on another [12]. 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. This allows these systems to support a wide range of interactive tasks, but requires developers to create their own apps and interfaces.

Snap-To-It contributes to this growing body of work by helping us better understand the types of interactions that 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 can already be deployed in a real environment. Consequently, we believe that our work offers a strong foundation for future research.

CONCLUSION

This paper presents Snap-To-It, a system that allows users to select and control nearby appliances using their mobile device's camera. Through two technology probes, we explored how users would like to select appliances from their environment, and the types of interactions that they would like to perform. We then designed a single system that satisfies these needs, while still leaving room for future expansion. Through four applications, we showed how Snap-To-It can support a wide range of interactions. We then evaluated our system's accuracy, and showed that our system is practical both in the lab and in real-world environments.

Our work takes an important step towards increasing a user's ability to interact with new and unfamiliar appliances. Nevertheless, there are a number of ways we can improve our system even further. One idea is to use Snap-To-It to interact with multiple appliances at the same time. Our system already lets users select an appliance *via* a photo, and in the future, we will 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 current system uses background scenery and compass orientation to differentiate between objects that look the same, there are still rare occasions where our system can mistake one appliance for another. In future work, we will improve our system by examining the *contents* of the image (*e.g.*, the words printed on an appliance or sign) in addition to raw SIFT features. This will increase our system's accuracy even further, and bring us closer to allowing users to interact with every appliance using a single technique.

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 Kubitz, 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. Sebastian Boring, Marko Jurmu, and Andreas Butz. 2009. Scroll, tilt or move it. In *OZCHI '09*. New York, New York, USA, 161–168. DOI: <http://dx.doi.org/10.1145/1738826.1738853>
11. 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>
12. 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>
13. Yu-Hsiang Chen, Ben Zhang, Claire Tuna, Yang Li, Edward A Lee, and Björn 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>
14. 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>
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. 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>
19. 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>
20. 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>
21. 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>
22. Simon Mayer, Markus Schallch, 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>
23. 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>
24. 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>

25. 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>
26. 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>
27. 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
28. Jukka Riekk, 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>
29. Matthias Ringwald. 2002. UbiControl: Providing New and Easy Ways to Interact with Various Consumer Devices. In *UbiComp '02*. 81,82.
30. Ichiro Satoh. 2011. A Management Framework for Context-Aware Multimedia Services.. In *DMS*. 165–170.
31. 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>
32. Chuong Cong Vo. 2013. *A Framework for a Task-Oriented User Interaction with Smart Environments Using Mobile Devices*. Ph.D. Dissertation. La Trobe University.
33. Hanno Wirtz, Jan R  th, Martin Serror, J     gila 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>
34. 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>