RapID: A Framework for Fabricating Low-Latency Interactive Objects with RFID Tags

Andrew Spielberg[*†] aespielberg@csail.mit.edu
Alanson Sample[*] alanson.sample@disneyresearch.com
Scott E. Hudson[*‡] scott.hudson@cs.cmu.edu
Jennifer Mankoff[*‡] jmankoff@cs.cmu.edu
James McCann[*] jmccann@disneyresearch.com

[*]Disney Research Pittsburgh Pittsburgh, PA
[†]Massachusetts Institute of Technology Cambridge, MA
[‡]Carnegie Mellon University Pittsburgh, PA

Figure 1. A wireless, batteryless game of Tic-tac-toe is designed using components in the RapID standard library, fabricated using laser cutting along with stick-on RFID tags and foil, and tracked with low latency using the RapID runtime library.

ABSTRACT

RFID tags can be used to add inexpensive, wireless, batteryless sensing to objects. However, quickly and accurately estimating the state of an RFID tag is difficult. In this work, we show how to achieve low-latency manipulation and movement sensing with off-the-shelf RFID tags and readers. Our approach couples a probabilistic filtering layer with a monte-carlo-sampling-based interaction layer, preserving uncertainty in tag reads until they can be resolved in the context of interactions. This allows designers' code to reason about inputs at a high level. We demonstrate the effectiveness of our approach with a number of interactive objects, along with a library of components that can be combined to make new designs.

Author Keywords

RFID; probabilistic modeling; computational fabrication

INTRODUCTION

Currently, graphical user interfaces can be constructed very quickly by composing them from a component library and writing a small amount of "glue" code to carry out user actions. Unfortunately, the same cannot yet be said for interfaces constructed with physical objects although there have been significant advances in this direction (*e.g.*, [6, 9, 15]). One challenge in the design of interactive physical objects is integrating sensing, processing, and communications components and their associated circuitry. For example, physical objects must usually be specially designed to make room for circuit boards, wiring, and power sources. Passive Radio Frequency Identification (*RFID*) tags offer a number of potential advantages which might overcome many of these difficulties. Passive RFID tags use very tiny electronic circuits which can be easily affixed to, or embedded in, many objects. Each RFID tag can be programmed with a 12-byte identifier known as an Electronic Product Code (*EPC*), which can be wirelessly queried by devices known as readers. Importantly, they also operate without a local power source, as they draw power from the remote reading antenna.

Recent advances in the analysis of the underlying RFID signals used for identifying tags have demonstrated that manipulations of tags, such as human touch, cover by conductive or dielectric materials, and tag motion, can be reliably detected using standard RFID tags and conventional readers positioned up to several meters away [16]. This opens up the possibility of using these tags as input devices. To produce high accuracy inputs from these low level signals takes time, waiting for multiple tag reads or other evidence that the system is correctly understanding what manipulations have caused the signal changes before reporting inputs. This can produce delays on the order of two seconds for most types of manipulations, limitinga the kinds of interactions they can be used for.

In this work, we introduce a new probabilistic framework for processing RFID tag signals which allows the application to

CHI'16, May 07 - 12, 2016, San Jose, CA, USA

ACM 978-1-4503-3362-7/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2858036.2858243

respond to input much more quickly. By accurately modeling the probabilities of partially understood input (using methods related to [31]), it is possible to make less conservative, but more informed decisions, and to deliver them faster – with typical delays under 200 msec. This means that input can generally be reported with a latency at, or under, typical minimum human perception-action cycle times [3] – more in line with most interactive systems than the two second delay that characterized previous work [16]. Further, the probabilistic formulation used to process the RFID tag signals can smoothly incorporate state information from an application, and probabilistic estimates can optionally be exposed to the application so that it can reason about them in an informed fashion.

To facilitate use of the techniques described here, we have also built an extension to the SketchUp 3D drawing package [18] which allows a library of interactive components built using our techniques to be easily integrated with physical objects The library we currently provide is implemented entirely with small, inexpensive UHF RFID tags (which in quantity can cost as little as USD$0.10 per tag [17]). Fabrication also requires cut to size copper foil to provide covers that can hide the RFID tag signal, which is important for several of the tag manipulations we support. Our system includes the code necessary to use each library component in our probabilistic framework at runtime.

After reviewing background on RFID sensing and key concepts needed to interact with them under uncertainty, we provide a system overview of RapID, our framework for RFID-based interaction. Following that, we describe the main contribution of this paper, our solution for speeding up interaction by interpreting interaction with RFID tags as probabilistic inputs and managing probabilistic states. We then present a developer API for creating more advanced applications using these probabilistic inputs, along with a CAD extension for quickly designing and fabricating interactive objects. Finally, we validate our method through a series of example applications implemented with our framework including a tic-tac-toe game, a pong game and an audio controller, and conclude by analyzing the strengths and limitations of our approach.

Background and Related Work

There is a rich history of developing systems which enable users to rapidly prototype interactive objects (*e.g.*, [6, 21, 34, 29]). Since the beginning, such work has emphasized simplicity of programming, often in a fashion mimetic to traditional UI programming environments (*e.g.*,[6]). More recently, work has added a facility to embed electronics in ordinary objects (*e.g.*, [21, 34, 26]). However these interactive circuits are limited in scope by the size of their setting, requiring users to make a tradeoff between small circuit footprints and larger component devices. An onboard power source must also be supplied. Further, while these frameworks automate much of the low-level design work, fabricating circuits is still a manual process which does not exploit digital manufacturing tools or the quick peel-and-stick advantages of RFID tags.

Other competing technologies for fabricating custom interactive components within physcial objects include acoustic sensing of gestures and other physical manipulation (*e.g.*, [14, 28] and capacitive touch sensing (*e.g.*,[10, 29, 7]). However, each of these technologies needs to create or use intricate and somewhat bulky components and/or wiring inside objects, all of which must typically be designed around when creating the object. It is also possible to use computer vision with depth or conventional cameras [36]. However, this approach suffers from occlusion problems and recognition of fine manipulations is difficult. One approach to overcoming this limitation is to place the camera inside the object [27]. This overcomes occlusion and resolution issues, but requires hollow objects with mostly unoccluded interiors, and still requires bulky and possibly costly electronics and power for each object.

RFID technologies fill an important gap in this body of work by enabling real-time interaction with objects that are easily fabricatable, require no on-board power, and require nothing more than stick-on tags and copper covers to add functionality. RFID systems consist of small, robust and inexpensive *tags* and more complex *readers* which can interrogate tags within range. RFID readers are designed to be part of the infrastructure of a space and to communicate with a host computer. Current RFID readers can cost around $130 for embedded modules [32], and around $1,000 for enterprise grade readers [13]. The readers used in this work make use of *far-field* RF effects at distances as far as 9 meters from the reader antenna to the tag(s). This can be contrasted with the *near-field* effects of readers now being built into some cell phones, which are limited to a few centimeters in range.

Our work uses passive UHF tags, which are ultra-thin tags powered solely by the radio frequency energy emitted by the reader, making them battery-free. Tags operate with low power because instead of generating a signal, they communicate with a reader by modulated backscatter, which reflects the reader's carrier signal back to the reader along with an encoded tag id. Manipulations of tags can influence the details of this signal and past work has demonstrated that it is feasible to detect these signal changes and interpret them in terms of user input.

Prior work has used RFID tags for input in a number of ways. Philipose *et al.* instrumented people with near-field detectors, and objects with RFID tags to detect activities [25]. Buettner *et al.* use a custom platform for detecting movement of tagged objects in large rooms instrumented with RFID readers to disambiguate between twelve daily household activities [2]. Recent work has demonstrated the feasibility of using passive UHF RFID tags for detecting motion [5, 24, 16], tag coverage [16], and touch and swipe gestures [16]. However, robust detection has required multiple readers and antennas [5], multiple tags per object [5], or accumulated signal data for up to 2 seconds [16], limiting the viability of the approach for interaction. In this paper we demonstrate how to overcome many of these limitations.

SYSTEM OVERVIEW AND MOTIVATING EXAMPLE

RapID allows for RFID-based interaction at low latencies by exploiting a probabilistic model of tag state. These probabilities are then managed in an intelligent way by a downstream application. This process is diagrammed in Figure 2.

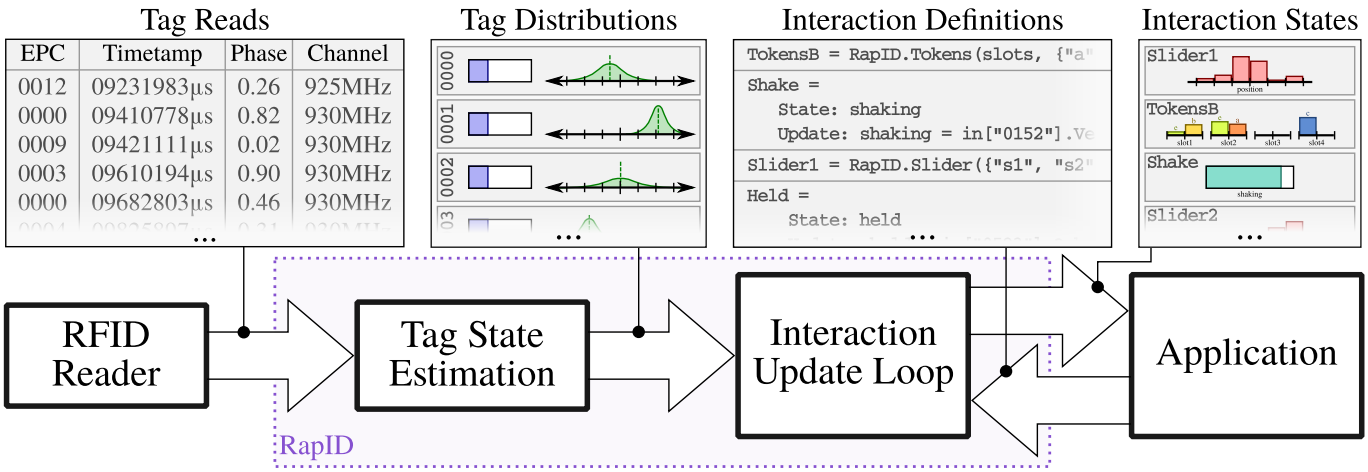

Figure 2. RapID propagates uncertainty about RFID tag states to applications by maintaining probability distributions over tag velocity and cover state, and sampling from these to update the state distributions of application-defined interactions.

Our solution is comprised of:

1. A *Tag State Estimation* method, which allows a system to incrementally update probability distributions over the state of each RFID tag, given a stream of reads from an RFID reader.

2. An *Interaction Update* method, which translates these low-level probabilistic inputs into a probability distribution over the states of states of higher-level interactions. These interactions can reason about multiple tags at once, and only need to be written using straightforward, non-probabilistic code.

3. A runtime API, which makes it easy for a developer to launch and monitor interactions (both pre-defined and developer-created).

4. An extension for SketchUp [18], including a standard library of pre-defined starter interactors and associated code, for designing physical objects for interaction; including code to export descriptions of designed objects for use by the API.

Together, these components allow a user to design physical objects, and with little to no additional code, enable user input to an application based on how the physical objects are manipulated.

As a motivating example, consider a scenario in which a developer desires to create an interactive game of Tic-Tac-Toe. The application must have the ability to track the game state, which includes detecting moves and when the game has ended. The application must also be able to provide feedback to players by both visually displaying the game state and providing audio feedback at key moments during the game, such as the game start, the start of each player's move, and the game's end.

Creating Tic-Tac-Toe with our framework is straightforward (see Figure 1 and video figure). The developer designs the board in SketchUp. Using our extensions, she creates a 3x3 array of game pieces and slots. The developer then assigns groups (X, O, board) and IDs (board positions) to the pieces, and uses our extension to export a table associating these names with the EPC numbers she will program into the RFID tags during construction. She exports a .dxf file for laser-cutting, and fabricates the board and game pieces – attaching foil and RFID tags where indicated.

In her game code, the developer writes code to initialize our API (passing it the exported name/EPC table), instantiate a token/slot interaction from our standard library, associate it with the appropriate tags, and translate the events it produces into game actions such as visual and auditory feedback, all of which can be written compactly.

PROBABILISTIC FORMULATION

One of the challenges in interpreting RFID interaction data at high speeds is the high level of uncertainty caused by the noisy signal. Uncertain input is an issue in other domains that involve recognition such as touch [1] and speech [23].

Several approaches exist for dealing with uncertainty. The most common but least pliable approach is to remove uncertainty before generating any application input. This approach is what led to the two second delay in Li *et al.*'s work, as they waited to inform the application until input presence and type was known with high confidence [16]. A second approach is to model the interface as a type of signal feedback system, engaging the user in a tight loop that visualizes uncertainty across the user interface [35]. However, this method requires a radically different approach to interface construction. A final approach is to maintain distribution over possible inputs that could have happened [11, 19, 20, 30, 31]. By modeling this as a monte-carlo sampling of a distribution over possible application states [31], it is possible for the developer to reason deterministically about the application, while the underlying infrastructure keeps track of any uncertainty.

This final approach is well suited to handling the type of uncertainty that arises when interpreting RFID signals as input. Schwarz *et al.* [30, 31] assume that each event is a complete distribution of possible inputs. RFID reads, however, only

provide data on a single tag at a time, providing an incomplete view of the true interaction state of the population. Attempting to change the state of the interface from each read event can cause the *order* of tag reads to have a serious impact on behavior. Thus, some way of accumulating information over time is needed. Therefore, we have taken a two-level approach, where we probabilistically model the state of each tag based on signal-level information, and use these models to generate higher level inputs that describe how we believe the tag is being manipulated. These in turn are used to (probabilistically) update interactor state.

In the remainder of this section, we first describe the low-level tag state distribution estimators in more detail, and then discuss how RapID translates probabilistic input state into probabilistic interaction state.

Tag State Estimation

For each tag in the environment, RapID maintains a probabilistic estimate of *a)* whether it is covered/visible and *b)* its velocity.

As tag reads arrive, RapID updates these distributions using Bayes filters. Chapter 2 of [33] provides an introductory background to Bayes filters. A Bayes filter is a procedure that, given a system evolving over time and a sequence of i observations $\mathbf{z}_{1:i}$, estimates the current distribution of possible states $\mathbf{x}_i$. The variable $\mathbf{x}_i$ can be a discrete or continuous. In our setting, $\mathbf{x}_i$ will either correspond to whether the tag is in a discrete covered or visible state, or it will correspond to a continuous value representing the tag's velocity. The conditional distribution of the current tag state, given the sequence of observations, $bel(\mathbf{x_i}) \equiv p(\mathbf{x}_i|\mathbf{z}_{1:i})$, is known as the current *belief*. The belief at step $i-1$, is used to estimate the belief at step i recursively:

$$\overline{bel}(\mathbf{x_i}) = \int_{\mathbf{x}_{i-1}} p(\mathbf{x}_i|\mathbf{x}_{i-1}) bel(\mathbf{x_{i-1}}) \tag{1}$$

$$bel(\mathbf{x_i}) \propto p(\mathbf{z}_i|\mathbf{x}_i)\overline{bel}(\mathbf{x_i}) \tag{2}$$

These equations first account for the evolution of the system's state between observations using a *transition model* prior, $p(\mathbf{x}_i|\mathbf{x}_{i-1})$; the updated state estimate $\overline{bel}(\mathbf{x}_i)$ is then re-weighted using an *observation model* $p(\mathbf{z}|\mathbf{x})$. We use $\propto$ in Equation (2) to indicate that the calculated belief is proportional to the true belief and thus normalization may be required after this step. The Bayes filter is thus a recursive application of Bayes' rule, *modulo* normalization. For readers who are familiar with particle filtering, we note that a particle filter is a special class of Bayes filter where the belief distributions are approximated through a collection of samples; in our Bayes filter, the belief distributions are calculated and maintained explicitly. Since we maintain a visibility filter and a velocity estimation filter for each tag, for a collection of m tags, we maintain $2m$ filters.

In the following sections, we describe the transition and update models for our filters for visibility and velocity estimation. First, however, we take a moment to describe the RFID protocol, which provides our system with its observations.

RFID Protocol

The Gen2 UHF RFID protocol [8] allows a single reader to inventory a collection of tags. These tags have no local power source, and communicate by harvesting energy from the reader's transmissions and using it to selectively change the RF reflectivity of their antennae. The reader measures these reflected emissions to receive information from the tags.

However, if multiple tags modulate their reflectivity at the same time, the reader cannot properly decode their transmissions. Thus, tag inventory proceeds in a series of rounds, where, in each round, the reader repeatedly allocates a number of response slots, and asks all tags that have not yet replied to reply in one of these slots. When a tag replies in a slot, the reader queries it for its EPC, and sends this number, along with ancillary data – in our case, a timestamp, the phase of the reflected carrier wave, and the channel frequency – to the host computer. Once no more tags reply, the reader starts a new round.

In practice, this means that the time between reads of RFID tags is randomly distributed, and depends on the total number of uncovered tags.

In turn, we now describe the how we formulate our tag cover and velocity filters as applications of the general Bayes filter as described above.

Tag Cover State

When a user action causes a tag to be covered (*e.g.,* by a foil-covered X or O in Tic-Tac-Toe), it is effectively invisible to the reader. Conductive (*e.g.* foil) or dielectric (*e.g.* hand) material very close to the tag will have this effect. Thus, the *lack* of a read can indicate a user action. The longer the reader fails to receive reads from a tag over time, the more probable it is that that tag is covered.

The state space for tag cover state is binary: $\mathbf{x}_i \in \{\text{visible}, \text{covered}\}$ – so the corresponding Bayes filter operates on a distribution represented by a single value $v_i \equiv \mathbb{P}(\mathbf{x_i} = \text{visible}|\mathbf{z_{1:i}})$. The observation space is continuous: $\mathbf{z}_i \in \mathbb{R}^+$, and is equal to the time that has elapsed since the previous read.

The transition function is straightforward – given the amount of time t since the last observation, the unobserved visibility distribution $\bar{v}_i = \overline{bel}(\mathbf{x_i})$ is computed from the previous visibility v_{i-1} by decaying toward a uniform distribution:

$$\bar{v}_i \leftarrow \frac{1}{2} + \frac{1}{2^{t/h}}\left(v_{i-1} - \frac{1}{2}\right) \tag{3}$$

where h sets the half-life of the decay. RapID's default value of h is 1 second. Developers may override this value if they know a better model for their application.

Unfortunately, the RFID protocol does not provide information about a "missed read." While the start-of-round signal might provide some evidence, it isn't exposed by common reader interfaces. Instead, we build our observation model on the elapsed time between subsequent reads. To estimate

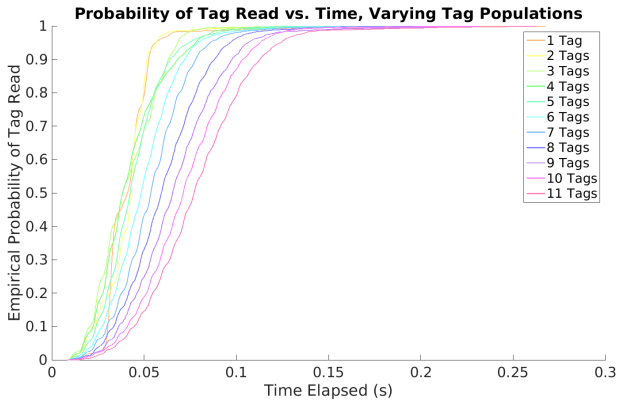

Figure 3. The probability a tag will have been read after a given amount of time, over various tag population sizes. On average, adding another tag adds approximately 5.13 ms of latency, with a tag in a population of 10 being seen after 140 ms with 95% probability.

the probability densities $p(\mathbf{z_i}|\text{visible})$ and $p(\mathbf{z_i}|\text{covered})$ from data, we conducted two sets of observations.

In the first, we recorded the time between reads for an uncovered tag with a varying total numbers of tags. The cumulative densities of these read times are shown in Figure 3. As expected, tag reads are randomly distributed, with larger population size correlated with more time between reads.

For the second set of observations we measured responses of covered tags. We used the same setup but covered one of the tags partially with aluminum foil. The read latency distributions of this covered tag – as one might expect – had large variance and mean (on the order of 2 s).

Our final observation model approximates these distributions as Gaussians, where the visible observation model depends on tag population size:

$$z_{\text{visible}} \sim \mathcal{N}^+(\mu_v(m), \sigma_v^2(m)) \tag{4}$$

$$z_{\text{covered}} \sim \mathcal{N}^+(\mu_c, \sigma_c^2) \tag{5}$$

$\mu_v(m)$ and $\sigma_v^2(m)$ correspond to the mean and variance respectively for the observed uncovered populations of size m; μ_c and σ_c^2 correspond to the mean and variance respectively for the covered observations. We found that in the uncovered case, both the mean and variance of the Gaussian had a strong linear correlation with the tag population size (R^2 with mean: 0.92, R^2 with standard deviation: 0.9072). Thus, our distribution for visible tags is parameteric in the tag population size, m. In the covered case, the mean and variance were mostly constant with respect to population size. We write $\mathcal{N}^+$ to indicate that RapID discards the negative portion of each Gaussian (negative read latencies don't make sense) and normalizes the remainder.

These Gaussians are well separated (Figure 4), making the observation very informative.

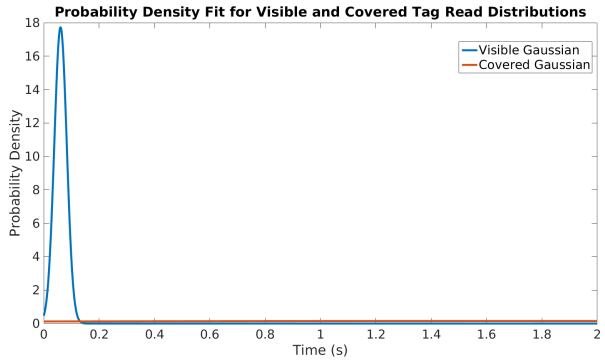

Figure 4. An example of the fit Gaussians to the covered and uncovered distributions for populations of 8 tags. The variance on the covered distribution is so large that it appears nearly flat. The covered state becomes more probable after 133 ms without a read.

These transition and observation models serve to update the visibility belief for each tag whenever a new read is received.

However, the visibility distribution is also required between (potentially sporadic) reads in order to update interaction state. Between reads, RapID constantly maintains the state belief by applying the transtion model and then integrating over the observation model, in order to estimate the probability of having not read the tag for t seconds:

$$\mathbb{P}(\text{no read for } t = T \mid \text{visible}) \propto 1 - \int_0^T \mathcal{N}^+(\mu_v(m), \sigma_v^2(m))dt \tag{6}$$

$$\mathbb{P}(\text{no read for } t = T \mid \text{covered}) \propto 1 - \int_0^T \mathcal{N}^+(\mu_c, \sigma_c^2)dt \tag{7}$$

Finally, we note that the filters are initialized with a default $v_0 = 0.5$ for each tag.

Velocity

User interactions that involve motion of a tag can also easily be sensed in terms of velocity relative to the tag reader. In this case, the state space is continuous – $\mathbf{x}_i \in \mathbb{R}$ – so the corresponding Bayes filter operates on a continuous distribution (modeled as a Gaussian) $\mathbf{x_i}|\mathbf{z_{1:i}} \sim \mathcal{N}(\mu_{\mathscr{V}_i}, \sigma_{\mathscr{V}_i}^2)$. The observation space is continuous: $\mathbf{z}_i \in \mathbb{R}$, and in this case is dependent on the difference in RF phase angles between subsequent tag reads. RapID represents the tag velocity as a normal distribution, and updates the distribution's mean, $\mu_{\mathscr{V}_i}$, and standard deviation, $\sigma_{\mathscr{V}_i}$, at observation i as follows.

RapID models velocity as being influenced by a normally-distributed acceleration over time t. We let $\overline{\mu}_{\mathscr{V}_i}, \overline{\sigma}_{\mathscr{V}_i}^2$ represent the updated mean and variance of the velocity estimate after the transition update. The transition update at observation i is thus:

$$\overline{\mu}_{\mathscr{V}_i} \leftarrow \mu_{\mathscr{V}_{i-1}} \tag{8}$$

$$\overline{\sigma}_{\mathscr{V}_i}^2 \leftarrow \sigma_{\mathscr{V}_{i-1}}^2 + at \tag{9}$$

We set a to 1m/s, which works well in practice, though developers using RapID may override this selection if they have a better model for their application.

In UHF RFID, the phase of the carrier wave returning to the reader is proportional to the path length between the antenna and tag (modulo the wavelength of the signal) [22]. Thus, given two subsequent reads separated by t seconds and a tag velocity of $\mathscr{V}$, one would expect to observe a phase difference of

$$\Delta\phi = \frac{2t\mathscr{V}}{\lambda} + N \tag{10}$$

where wavelength $\lambda = \frac{c}{f}$ can be computed from the frequency of the carrier wave and the speed of light, with noise $N \sim \mathscr{N}(0, 0.01\pi^{-2})$ according to the manufacturer of our reader [12]. From this equation, one can immediately compute the observation model $p(\Delta\phi|\mathscr{V})$. This model works well as long as t is small; this is typically the case when the tag is uncovered, and it is not an unnatural constraint to require uncovered tags for motion-based interactions. Note that both the observation and transition models are Gaussian, and thus the velocity belief can be computed as a closed form Gaussian at each step.

The velocity distribution is updated whenever two consecutive reads of the same tag are performed on the same channel (that is, with a carrier wave of the same frequency). However, RFID readers are required by the FCC to change their carrier frequency every 400ms [4]. In practice, about 80% of reads are same-channel, which still provides plenty of data to keep the model updated.

UHF RFID employs carrier frequencies in the 900-930MHz range, giving it a wavelength of approximately 32 cm. However, the reader we are using can only determine phase ϕ up to a half-wavelength, so any object that moves more than 8cm between reads ($\approx$ 2m/s) runs the risk of having its velocity underestimated; of course, an object moving 2m/s will also quickly leave the working volume of the reader.

Finally, we note that the filters are initialized with a default $\mu_{\mathscr{V}0} = 0, \sigma_{\mathscr{V}0} = 1$ for each tag.

Interaction State Distributions

Since interactions in RapID have uncertain inputs, they may be in uncertain states. Given a developer-supplied deterministic state update rule, RapID uses sampling to approximate the distribution of possible interaction states, similar to [30].

Developers specify an interaction by providing a sampling budget q, a starting state r_0, and an update function $u(r,i,t) \to r'$ which, given a current interaction state r and a sample i drawn from the current *joint* tag state distribution distribution, along with an elapsed time t, produces an updated state sample. Note that this function does not need to reason about probability distributions or weights, as it takes as input only concrete samples, rather than distributions. A typical update function might do things like implement a finite-state controller for an interactor.

For every current interaction, RapID maintains a population of states R, which it updates every timestep in Monte Carlo fashion by applying u to q sampled inputs and states (Algorithm 1).

Algorithm 1 The update loop run for every interaction.

```
function UpdateLoop(q, r_0, u)
    R_0 ← {r_0}
    k ← 0
    while InteractionIsRunning do
        t ← ElapsedTime()
        for i = 1 to q do
            r'_i ← u(SampleStates(R_k), SampleInput(I_k), t)
        end for
        R_{k+1} ← {r'_i}
        k ← k + 1
    end while
end function
```

A nice property of this method is that the solution is scalable: updates among samples can occur in embarrassingly parallel fashion. This means that although R_k is an approximation of the true distribution, q can be increased and the approximation thus improved simply by adding more CPU or GPU cores.

By default, interactions run indefinitely. However, developers may choose to specify a post-update function which is called after every update step. This function examines the current interaction state distribution and can, *e.g.*, dispatch events, update summary statistics, terminate the interaction, or even modify the sample population. One such post-update function that we have found convenient terminates the interaction and dispatches an event whenever an interaction state accounts for more than a (user-specified) proportion p of the total pool of states.

DEVELOPER API

Our Development API, which is implemented in C#, includes three developer-facing classes, **Application**, **Interaction**, and **InteractionState**. Though these classes may be extended for custom behavior, RapID already includes subclasses to support the interactors in its standard library. For example, with the game piece interactor, developers can use an off-the-shelf Interaction to track piece placement, and must only specify what happens when specific game pieces are placed. Figure 5 provides a visualization of how these classes are related in our API.

Application is a singleton class which handles interaction with the RFID reader hardware, including setting a tag mask to avoid reading tags that are currently irrelevant to the Interactions. It also runs tag state estimation and the Interaction update loop.

We have produced a version of RapID whose Application class includes additional functionality, including a general-purpose main loop and functions to visualize the superposition of interaction states; however, the demos shown in the video were all produced using a version of RapID whose Application class sits alongside user code, rather than subsuming it.

The **Interaction** class represents interaction processes which run in an Application. Each Interaction is responsible for

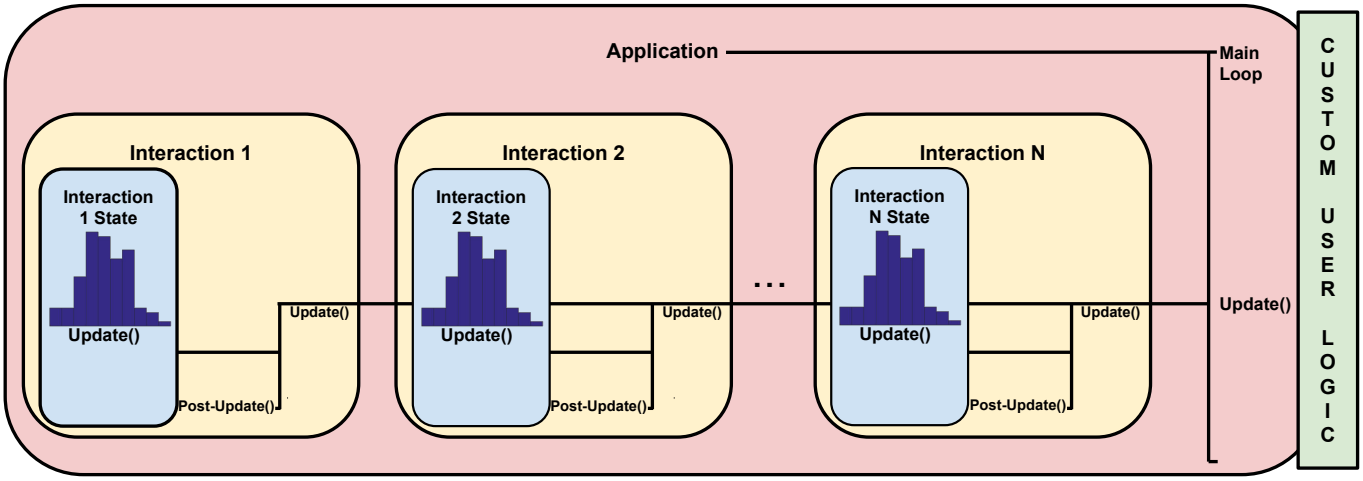

Figure 5. A visualization of our API. A main application contains a collection of Interactions, each with its own internal probabilistic InteractionState representation. The application iteratively updates each Interaction, which in turn update their InteractionStates. Additional user code discrete from inputs can live inside the Application but distinct from our RapID API.

maintaining and updating a distribution over possible InteractionStates and contains an overridable post-update function, as discussed in the previous section.

Finally, **InteractionState** wraps the state and update function for each interaction. Developers creating a new interaction will generally create a subclass of InteractionState and override its Update method.

STANDARD COMPONENT LIBRARY

While RFID tags can be embedded on any object, we provide an interface for designing new objects from scratch. Our interface is designed as a component library and extension for SketchUp [18], a commonly used CAD package aimed at novice designers, which can export designs to .obj and .dxf files for digital manufacturing.

The component library (Figure 6) contains several basic interactive elements, already marked up with special materials to indicate where to place foil and tags. The extension script assigns a unique EPC to every RFID tag placed in the model; creates a text label so that the developer will know what EPC to set on their RFID tags during construction; and saves a table mapping these EPCs to SketchUp object names, for use when referencing tags in the API.

RapID's standard library contains the following objects:

Bare Tags and Covers Tags and foil covers are the building blocks of any interaction, and tags can also be used on their own to sense motion or touch.

Game Pieces and Slots. Game pieces and slots have tags and covers placed so that when a piece is inserted into a slot, both the piece's tag and the slot's tag will be covered. Corresponding Interaction code identifies correlated (dis)appearances of tags to keep track of which pieces have been placed into which slots. Such objects are a useful building-block for defining most board games (our Tic-Tac-Toe application uses pieces for both the "x"'s and the "o"'s; they can also represent pawns in checkers or various chess pieces), or registration of figurines for applications such as Skylanders or Amiibo[1]. Pieces are keyed by trimming a corner to ensure that they are always placed in an orientation in which the respective tags and covers to touch.

Sliders. A slider is a row of tags with a sliding cover, allowing it to take on a range of states depending on which tags are covered. Sliders are especially useful for range-based inputs, and the slider can be fabricated at arbitrary lengths. The number of slider states is linear in its length. The standard Interaction for sliders estimates the slider state by looking for either a single covered tag or two adjacent covered tags. Our standard library provides sliders with and without rails to hold the cover in place (since "cover not present" can be a useful interaction signal).

Rotary Encoder. Our rotary encoder places n tags radially from its center, and uses a foil spinnable cover cut into an n-bit circular gray-code pattern. An encoder with n tags will have 2^n states. The state is estimated by sampling the cover state of each of the tags - each subset of covered tag states corresponds to a unique encoder state.

EXAMPLE APPLICATIONS

In order to test the RapID framework and component library, we created several applications which use tracked RFID tags. We encourage readers to view the accompanying video to see these applications in action.

Our testing environment (Figure 7) was a general-purpose office space. We placed an Alien ALR-9611-CR antenna under small wooden table (chosen because it is RF-transparent). Atop the table we placed a monitor for visual feedback, along with our custom RFID-tracked input devices. We connected the antenna to an Impinj Speedway Revolution R420 UHF RFID reader, which, in turn, was connected to the computer

[1] See http://www.skylanderscharacters.com/figures.asp and http://www.nintendo.com/amiibo respectively.

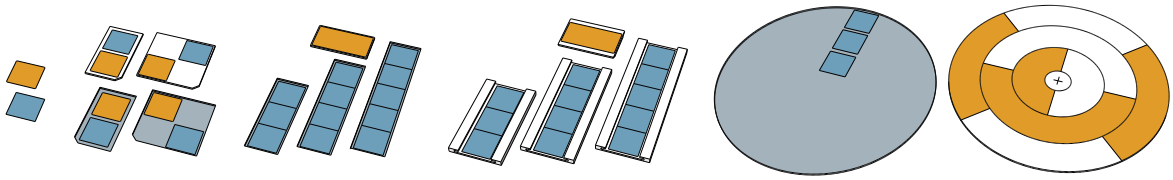

Figure 6. Our standard component library for SketchUp includes – from left to right – bare tags and covers, game pieces and slots, sliders without and with locking rails, and a rotary encoder. Blue material indicates tags, while orange material indicates metallic foil covers.

Figure 7. Our testing area consisted of an RFID antenna placed on the floor, a non-metallic table, and a monitor for visual feedback. The RFID reader and the computer running the application (both not shown) are placed to the side, and connected *via* ethernet.

running the application *via* ethernet. Latency of the ethernet connection, as reported by the 'ping' command, was less than 1 ms.

Our applications use our RapID C# API for input, and the Unity game engine for graphics and sound. All reported numbers are from application tests run on a Late 2013 MacBook Pro Retina, 15-inch, with a 2.6 GHz Intel Core i7 processor and 16 GB of RAM. In all cases RapID-related processing was a negligible part (< 10%) of the overall runtime of the application.

Each example application required fewer than 200 lines of code to be explicitly written by the developer, with the most complex (Pong) taking 188 lines (mostly concerned with complex ball physics), and the least complex (Spaceship) only requiring 73. Sampling budgets were never an isuse.

Figure 8. Our Tic-Tac-Toe application's X and O markers. From left to right: as designed in SketchUp; pending RFID and foil attachment; completed.

Tic-Tac-Toe

Our Tic-tac-toe application, Figure 1 and 8, provides visual and audio feedback about a game of Tic-tac-toe, prompting players to place game pieces into slots on a physical fabricated board, and congratulating them if they win.

Tic-tac-toe was designed by using our SketchUp components for the physical design, fabricated from laser-cut plywood, and programmed using RapID's default "game piece and slot" interaction, which maintains a list of piece/slot correspondences, incrementally updating the state when the majority of its samples agree on the correlated (dis)appearance of a pair of "piece" and "slot" tags.

The finished example has the most tags (18) of any example, but at 530 reads per second, each tag state is still updated at approximately 29Hz.

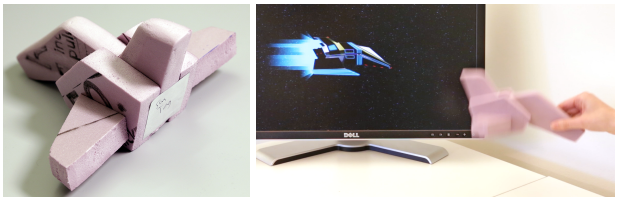

Figure 9. In our spaceship application, a single tag is used to track the motion of a spaceship model. This motion is used to animate an on-screen ship.

Spaceship

Our spaceship application (Figure 9) demonstrates how single-tag motion tracking can add interactivity to an otherwise inert object. When the user shakes the spaceship prop on the table, the spaceship on the screen accelerates.

This application animates the on-screen ship using the value reported by our our default "speed" interaction, which computes the average magnitude of its state's samples of a tag's velocity. This application also serves to test the upper bound on single-tag update speed – 146 reads per second, of which approximately 140 were same-channel reads suitable for velocity updates.

Figure 10. In our pong-like application, two slider components fabricated using different technologies are used to control the paddles in the game.

Pong

Our pong application (Figure 10) uses two slider components from the component library to control a pong-like game.

Such a game requires responsive (low-latency) controls to be playable.

The sliders are basic parts, included in our SketchUp component library. We chose to use a different fabrication process for each slider to show that RapID is material-agnostic. One slider was printed on an Object Connex 260 3D printer using Vero Clear material, while the other was made from laser-cut acrylic, stacked and glued. Both perform well, providing no noticable advantage in play to one side or the other.

Each slider is read using RapID's default "slider" interaction. This interaction's states each estimate the slider position based on which tags they see as covered in their sample of the tag distributions, and a post-update function combines these estimates with a weighted average.

During our tests, RapID processed about 415 reads per second – that is, it was able to update each of the eight tags at approximately 50Hz.

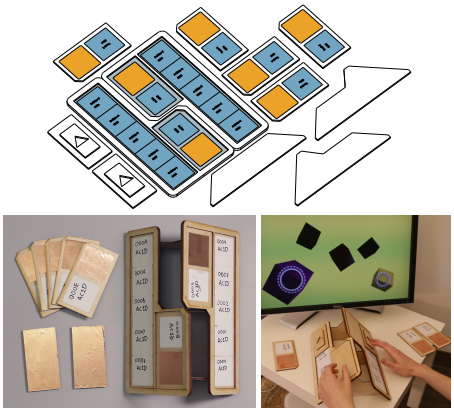

Figure 11. Our audio controller application uses several RapID components and associated interactions to enable an interactive music-mixing experience.

Audio Controller

Another application example is an audio controller (Figure 11). Users of the controller place pieces associated with music loops into two slots to activate them, and move sliders to change the volume of the respective currently active loops. This audio controller shows how a device can be quickly composed using different components and interactions from RapID's library, and shows how multiple interactions (two slider interactions and a piece/slot interaction) can be combined to build a more sophisticated experience.

The completed device contains 17 tags, and – in our tests – produces about 490 reads per second, resulting in an approximately 28 Hz per-tag update rate.

Quiz Game

In our quiz game application (Figure 12), players are asked a multiple choice trivia question; players then select their answer by launching a puff of air from a vortex cannon at one of three flags. The quiz game example demonstrates how RFID tracking can be used to build large interactive experiences (the read range of the tags we are using is several meters), and shows how custom interactions can be useful.

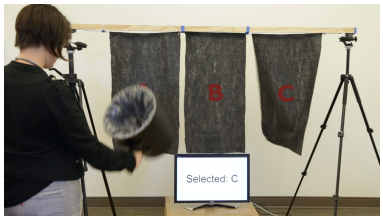

Figure 12. In this quiz game, the player answers each question by shooting a puff of air at one of three curtains using a vortex cannon toy. A custom RapID interaction reads the velocities of tags attached to the bottom of each curtain to decide which (if any) has been hit.

For this example, an interaction was written that compares the average speeds of tags attached to the three flags, terminating with a confirmed selection whenever one tag's speed is significantly higher than the others with high probability.

Discussion

As we have demonstrated, off-the-shelf RFID tags and readers can support reasonably low-latency wireless interactivity. However, our system is not without limitations. Velocity can only be detected for motion toward or away from the antenna, so extra antennas would be needed to extract full 3D motion. Another limitation is that tags cannot be placed directly on dielectric surfaces, meaning that intermediate surfaces must be put in between the tags and any dielectric surfaces.

Since the average time between tag reads scales linearly with the number of tags in view, there is a limit on the number of tags that can be added before the responsiveness for touch events begins to suffer. Similarly, because only a limited number number of tag reads happens within a single channel, and two reads of the same tag must happen within the same channel in order to extract motion features, with enough tags tracking quick motion accurately becomes infeasible. This wasn't an issue in any of our demonstrations (up to 20 tags), but with enough tags we expect it to become one.

The UHF RFID Gen2 protocol includes support for the selection of subsets of tags using tag masking. This should allow for responsive interactions with large populations of tags, as long as only a small subset needed to be read at once, such as only the X's and board tags or only the O's and board tags in Tic-Tac-Toe. While we did test dynamically changing the tag mask, we did not include it in any of our examples because we found changing the mask to be an expensive operation for our reader (with delays up to 150 ms), and this added overhead was not worth the increased tag responsiveness in any of our applications.

While classification with RapID is roughly ten times faster than the prior state of the art, IDSense [16], the methods cannot be compared directly. IDSense relies on a discriminative model for activity recognition whereas RapID employs a generative model. Thus, IDSense waits and collects sufficient information about a tag before giving a definitive answer about its state; while RapID immediately responds but only converges over time to the correct label. In practice, RapID converges to the correct cover/uncover labels with high probability and high accuracy within 200ms, ten times faster than

the flat 2s required for IDSense's method. A more rigorous probabilistic evaluation and benchmarking of RapID's methods may be desired in the future.

Hardware or reader firmware modifications could further lower latency and improve the range of possible interactions. For example, modified reader firmware that was more careful about frequency hopping would allow more tags to be simultaneously tracked for speed; while firmware that could poll different tags at different frequencies would allow for more flexible interface designs. In addition, it might be possible to create custom RFID tags specifically for interaction, e.g., by having them perform some sort of low-power sensing locally, rather than just reporting an ID. It might also be possible to create tags which can be placed on more kinds of material surfaces. Finally, the RFID protocol itself wasn't designed for interactive applications; it could be revised to better support the sorts of sensing we have described. In the future, it would also be valuable to explore processes that can embed the tags into objects during digital fabrication.

CONCLUSION

We demonstrated how a designer can use RFID tags to add inexpensive, wireless, low-footprint, batteryless, low-latency input sensing to objects. The key to our approach was a probabilistic formulation which enabled low-level uncertainty to be handled in a high-level fashion. This, coupled with an API for managing probabilistic interactions and a simple design interface embedded in SketchUp makes RapID both an easy to use and flexible design and development platform. By making it easy to add RFID-based sensing to objects, RapID enables the design of new, custom interactive objects with a very fast development cycle.

Acknowledgments

We thank Moshe Mahler, Sahana Vijai, and Kyna McIntosh for our demos' digital assets. We thank Hanchuan Li for providing starter code for interfacing with the RFID Reader. We thank Lea Albaugh, Eric Brockmeyer, and Alex Alspach for their advice and assistance in the fabrication of widgets.

REFERENCES

1. Xiaojun Bi and Shumin Zhai. 2013. Bayesian touch: a statistical criterion of target selection with finger touch. In *The 26th Annual ACM Symposium on User Interface Software and Technology, UIST'13, St. Andrews, United Kingdom, October 8-11, 2013*. 51–60. DOI: http://dx.doi.org/10.1145/2501988.2502058

2. Michael Buettner, Richa Prasad, Matthai Philipose, and David Wetherall. 2009. Recognizing daily activities with RFID-based sensors. In *UbiComp 2009: Ubiquitous Computing, 11th International Conference, UbiComp 2009, Orlando, Florida, USA, September 30 - October 3, 2009, Proceedings*. 51–60. DOI: http://dx.doi.org/10.1145/1620545.1620553

3. Stuart K Card, Allen Newell, and Thomas P Moran. 1983. *The psychology of human-computer interaction*. L. Erlbaum Associates Inc.

4. Federal Communication Commission (FCC). 2011. Title 47: Telecommunication, Part 15 Radio Frequency Devices. www.fcc.gov. (January, 31 2011). http://www.fcc.gov/

5. Kenneth P. Fishkin, Bing Jiang, Matthai Philipose, and Sumit Roy. 2004. I Sense a Disturbance in the Force: Unobtrusive Detection of Interactions with RFID-tagged Objects. In *UbiComp 2004: Ubiquitous Computing: 6th International Conference, Nottingham, UK, September 7-10, 2004. Proceedings*. 268–282. DOI: http://dx.doi.org/10.1007/978-3-540-30119-6_16

6. Saul Greenberg and Chester Fitchett. 2001. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology, UIST 2001, Disney's BoardWalk Inn Resort, Walt Disney World, Orlando, Florida, USA, November 11-14, 2001*. 209–218. DOI: http://dx.doi.org/10.1145/502348.502388

7. Tobias Alexander Große-Puppendahl, Yannick Berghoefer, Andreas Braun, Raphael Wimmer, and Arjan Kuijper. 2013. OpenCapSense: A rapid prototyping toolkit for pervasive interaction using capacitive sensing. In *2013 IEEE International Conference on Pervasive Computing and Communications, PerCom 2013, San Diego, CA, USA, March 18-22, 2013*. 152–159. DOI: http://dx.doi.org/10.1109/PerCom.2013.6526726

8. GS1. 2004. EPC Radio-Frequency Identity Protocols Generation-2 UHF RFID. (2004). http://www.gs1.org/sites/default/files/docs/epc/Gen2_Protocol_Standard.pdf.

9. Björn Hartmann, Scott R. Klemmer, Michael S. Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology, Montreux, Switzerland, October 15-18, 2006*. 299–308. DOI: http://dx.doi.org/10.1145/1166253.1166300

10. Scott E. Hudson and Jennifer Mankoff. 2006. Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology, Montreux, Switzerland, October 15-18, 2006*. 289–298. DOI: http://dx.doi.org/10.1145/1166253.1166299

11. Scott E. Hudson and Gary L. Newell. 1992. Probabilistic State Machines: Dialog Management for Inputs with Uncertainty. In *Proceedings of the Fifth ACM Symposium on User Interface Software and Technology, UIST 1992, Monteray, CA, USA, November 15-18, 1992*. 199–208. DOI: http://dx.doi.org/10.1145/142621.142650

12. Impinj. 2013. Low Level User Data Support. Application Note. (2013). https://support.impinj.com/hc/en-us/articles/202755318-Application-Note-Low-Level-User-Data-Support.

13. Impinj, Inc. 2011. *Impinj Speedway Revolution R220* (www.impinj.com ed.). Impinj, Inc., 701 N. 34th Street, Suite 300 Seattle, WA 98103.

14. Gierad Laput, Eric Brockmeyer, Scott E. Hudson, and Chris Harrison. 2015. Acoustruments: Passive, Acoustically-Driven, Interactive Controls for Handheld Devices. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. 2161–2170. DOI: http://dx.doi.org/10.1145/2702123.2702414

15. Johnny C. Lee, Daniel Avrahami, Scott E. Hudson, Jodi Forlizzi, Paul H. Dietz, and Darren Leigh. 2004. The calder toolkit: wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, Cambridge, MA, USA, August 1-4, 2004*. 167–175. DOI: http://dx.doi.org/10.1145/1013115.1013139

16. Hanchuan Li, Can Ye, and Alanson P. Sample. 2015. IDSense: A Human Object Interaction Detection System Based on Passive UHF RFID. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. 2555–2564. DOI: http://dx.doi.org/10.1145/2702123.2702178

17. RFID Journal LLC. 2015. How much does an RFID tag cost today? (sep 2015). http://www.rfidjournal.com/faq/show?85

18. Trimble Navigation Ltd. 2000–2015. Sketchup. http://www.sketchup.com/. (2000–2015).

19. Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. 2000a. Interaction techniques for ambiguity resolution in recognition-based interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, UIST 2000, San Diego, California, USA, November 6-8, 2000*. 11–20. DOI: http://dx.doi.org/10.1145/354401.354407

20. Jennifer Mankoff, Scott E. Hudson, and Gregory D. Abowd. 2000b. Providing integrated toolkit-level support for ambiguity in recognition-based interfaces. In *Proceedings of the CHI 2000 Conference on Human factors in computing systems, The Hague, The Netherlands, April 1-6, 2000.* 368–375. DOI: http://dx.doi.org/10.1145/332040.332459

21. David Mellis, Sam Jacoby, Leah Buechley, Hannah Perner-Wilson, and Jie Qi. 2013. Microcontrollers as material: crafting circuits with paper, conductive ink, electronic components, and an "untoolkit". In *Seventh International Conference on Tangible, Embedded, and Embodied Interaction, TEI'13, Barcelona, Spain, February 10-13, 2013*. 83–90. DOI: http://dx.doi.org/10.1145/2460625.2460638

22. P.V. Nikitin, R. Martinez, S. Ramamurthy, H. Leland, G. Spiess, and K.V.S. Rao. 2010. Phase based spatial identification of UHF RFID tags. In *2010 IEEE International Conference on RFID*. 102–109. DOI: http://dx.doi.org/10.1109/RFID.2010.5467253

23. Sharon L. Oviatt. 1999. Mutual Disambiguation of Recognition Errors in a Multimodel Architecture. In *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15-20, 1999*. 576–583. DOI: http://dx.doi.org/10.1145/302979.303163

24. Siddika Parlak and Ivan Marsic. 2013. Detecting object motion using passive RFID: A trauma resuscitation case study. *Instrumentation and Measurement, IEEE Transactions on* 62, 9 (2013), 2430–2437.

25. Matthai Philipose. 2005. Large-Scale human activity recognition using ultra-dense sensing. *The Bridge, National Academy of Engineering* 35, 4 (2005).

26. Raf Ramakers, Kashyap Todi, and Kris Luyten. 2015. PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. 2457–2466. DOI: http://dx.doi.org/10.1145/2702123.2702487

27. Valkyrie Savage, Colin Chang, and Björn Hartmann. 2013. Sauron: embedded single-camera sensing of printed physical user interfaces. In *The 26th Annual ACM Symposium on User Interface Software and Technology, UIST'13, St. Andrews, United Kingdom, October 8-11, 2013*. 447–456. DOI: http://dx.doi.org/10.1145/2501988.2501992

28. Valkyrie Savage, Andrew Head, Björn Hartmann, Dan B. Goldman, Gautham J. Mysore, and Wilmot Li. 2015. Lamello: Passive Acoustic Sensing for Tangible Input Components. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. 1277–1280. DOI: http://dx.doi.org/10.1145/2702123.2702207

29. Valkyrie Savage, Xiaohan Zhang, and Björn Hartmann. 2012. Midas: fabricating custom capacitive touch sensors to prototype interactive objects. In *The 25th Annual ACM Symposium on User Interface Software and Technology, UIST '12, Cambridge, MA, USA, October 7-10, 2012*. 579–588. DOI: http://dx.doi.org/10.1145/2380116.2380189

30. Julia Schwarz, Jennifer Mankoff, and Scott E. Hudson. 2011. Monte carlo methods for managing interactive state, action and feedback under uncertainty. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, Santa Barbara, CA, USA, October 16-19, 2011*. 235–244. DOI: http://dx.doi.org/10.1145/2047196.2047227

31. Julia Schwarz, Jennifer Mankoff, and Scott E. Hudson. 2015. An Architecture for Generating Interactive Feedback in Probabilistic User Interfaces. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems, CHI 2015, Seoul, Republic of Korea, April 18-23, 2015*. 2545–2554. DOI: http://dx.doi.org/10.1145/2702123.2702228

32. ThingMagic, A Division of Trimble 2015. *M6E-NANO: Embedded UHF RFID Module* (www.thingmagic.com ed.). ThingMagic, A Division of Trimble, One Merrill Street Woburn, MA 01801.

33. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. 2005. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

34. Alexander Wiethoff, Hanna Schneider, Julia Küfner, Michael Rohs, Andreas Butz, and Saul Greenberg. 2013. Paperbox: a toolkit for exploring tangible interaction on interactive surfaces. In *Creativity and Cognition 2013, C&C '13, Sydney, NSW, Australia, June 17-20, 2013*. 64–73. DOI: http://dx.doi.org/10.1145/2466627.2466635

35. John Williamson. 2006. *Continuous uncertain interaction*. Ph.D. Dissertation. University of Glasgow.

36. Robert Xiao, Chris Harrison, and Scott E. Hudson. 2013. WorldKit: rapid and easy creation of ad-hoc interactive applications on everyday surfaces. In *2013 ACM SIGCHI Conference on Human Factors in Computing Systems, CHI '13, Paris, France, April 27 - May 2, 2013*. 879–888. DOI: http://dx.doi.org/10.1145/2470654.2466113