# RFID Localization for Interactive Systems

by

Ian J. Clester

S.B., Electrical Engineering and Computer Science, Music
Massachusetts Institute of Technology, 2019

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020



Author .....
Department of Electrical Engineering and Computer Science
August 14, 2020

Certified by .....
Fadel Adib
Associate Professor
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

**RFID Localization for Interactive Systems**

by

Ian J. Clester

Submitted to the Department of Electrical Engineering and Computer Science
on August 14, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

## Abstract

Position-tracking is a useful primitive in a wide variety of domains, but positioning at the intersection of robust, accurate, and cheap has remained elusive. Radio-based techniques offer the potential to determine position reliably without relying on line-of-sight visibility or integration methods. Recent work has demonstrated the feasibility of passive UHF RFID tags as cheap tracking targets but has yet to prove its worth for interactive applications. In this thesis, I explore the possibility of RFID localization for interactive systems that require high throughput and low latency. I contribute an SDR-based localization system and its characterization, a library that abstracts away protocol-level details, and two interactive applications that demonstrate the system's capabilities.

Thesis Supervisor: Fadel Adib
Title: Associate Professor

# Acknowledgments

First, I thank my advisor Fadel Adib for taking me on as his student and providing practical insight into the strange world of academia—and, of course, helpful advice in writing this document.

I wish to thank my colleagues Mergen Nachin and Haris Brkic for their camaraderie over the past year. I am glad I had the opportunity to work and play with them. Thanks also to Jason Leng for his friendship and assistance early on.

Finally, I'd like to thank my folks back home for their constant love and support, and my girlfriend who—in addition to my usual nonsense—put up with my research taking over half of the apartment for several months. You're the best.

This research was supported in part by the MIT Media Lab and the National Science Foundation.

# Contents

**1 Introduction** **15**

1.1 Related Work . . . . . 16

1.2 Contributions . . . . . 17

**2 System Overview** **19**

**3 Operation** **21**

3.1 Backscatter Model . . . . . 22

3.2 Communication . . . . . 24

3.2.1 RFID Coding Schemes . . . . . 24

3.2.2 Packet Parameter Estimation . . . . . 25

3.2.3 Maximum Likelihood FM0 Decoder . . . . . 28

3.3 Localization . . . . . 34

3.3.1 Wideband Channel Estimation . . . . . 34

3.3.2 Distance Estimation . . . . . 36

3.3.3 Location Estimation . . . . . 41

3.4 Tracking . . . . . 42

3.4.1 Framerate Optimizations . . . . . 42

3.4.2 Multi-tag . . . . . 45

**4 Implementation & Evaluation** **47**

4.1 Hardware . . . . . 47

4.2 Software . . . . . 47

4.2.1 Protocol . . . . . 49
4.2.2 Throughput . . . . . 49
4.2.3 Library . . . . . 50
4.3 Evaluation Environment . . . . . 51

**5 Results** **53**
5.1 Range . . . . . 53
5.1.1 Powering up . . . . . 53
5.1.2 Decoding . . . . . 54
5.2 Throughput . . . . . 56
5.3 Accuracy . . . . . 57

**6 Applications** **59**
6.1 Sequencer/Synthesizer . . . . . 59
6.2 Multimapper . . . . . 60

**7 Conclusion** **65**

# List of Figures


2-1 System Architecture . . . . . 19

3-1 The Principle of Backscatter . . . . . 22
3-2 Backscatter in the Complex Plane . . . . . 23
3-3 FM0 symbols and bit sequences . . . . . 25
3-4 FM0 Preamble . . . . . 26
3-5 Packet Parameter Estimation . . . . . 27
3-6 FM0 State Machine . . . . . 30
3-7 Naive FM0 Decoder: Four cases . . . . . 31
3-8 An FM0-encoded bit and its neighboring half-bits, with noise. . . . . 32
3-9 BER vs. SNR in theory and simulation . . . . . 33
3-10 Wideband channel estimate . . . . . 35
3-11 IFFTs of channel estimates . . . . . 36
3-12 Phases before and after filtering . . . . . 38
3-13 IFFT magnitude and distance clusters . . . . . 41
3-14 Multilateration . . . . . 43
3-15 One full round of communication: Query followed by QueryAdjusts . 44
3-16 The repeated transmission for TurboTrack vs. the repeated QueryAdjust 44
3-17 Reading a tag: RN16, ACK, EPC . . . . . 45

4-1 Home setup . . . . . 48

5-1 Success in identifying a tag versus distance (simulated by attenuation). 54
5-2 SNR vs. BER, in theory and in practice. . . . . 55

5-3 Framerate per tag vs. number of tags . . . . . 56
5-4 Accuracy experiment with stationary positions in 2D. . . . . 57
5-5 CDF of errors from the true locations, in centimeters. . . . . 58

6-1 Sequencer/Synthesizer . . . . . 60
6-2 Multimapper for drawing . . . . . 61
6-3 Multimapper for pong . . . . . 61

# List of Tables

4.1 Library API . . . . . 50

# List of Listings

3.1 Packet Parameter Estimation . . . . . 29
6.1 Multimapper Configuration for Drawing . . . . . 63

# Chapter 1

# Introduction

Knowing where things are is fundamental to interpreting and reacting to the world. Humans perform localization operations constantly and subconsciously, relying on light, sound, and touch to determine the location of objects in their environment, and employing proprioception (also known as kinesthesia) to determine their own position and pose.

Attempts to grant the same skills to machines have largely focused on bringing machines the same senses humans use. Computer vision approaches remain popular; submarines use sound for underwater localization; the Roomba relies entirely on touch, via bumping into things; and quadcopters and autonomous vehicles rely on a kind of proprioception afforded by accelerometers.

It is natural that we have pursued these lines of development; they occur to us first and are the easiest for us to conceptualize, as many of us have ample experience with those sensory mediums ourselves. Their limitations similarly "make sense" to us: if a webcam or a Kinect cannot see someone's pose behind a wall, well, of course it can't—after all, we can't either. This line of thinking has especially influenced the fields of Human-Computer Interaction and Robotics, on the basis that it may be easier to interact with (and reason about) machines that are more like us.

But the limitation is artificial. We have technology that can sense what we cannot, senses without human analogy. Foremost among these is radio-based sensing, which has been only occasionally featured in HCI and instrument design (as compared to

more popular methods like vision; the Theremin [3] represents an early exception) but has played an essential role in defense since WWII, in the form of radar [4]. Compared to more common sensing mechanisms in interactive applications, such as vision and acceleration, radio offers the ability to sense through many obstructions while avoiding estimation drift.

Radio-based sensing has its own challenges. It's easy enough to isolate objects that are moving at a different speed from the rest of the environment (hence radar guns), but separating non-moving objects, or identifying individual objects generally, is difficult and expensive. Radio-frequency identification (RFID) technology [5] addresses these problems using devices ("tags") that can selectively respond to radio signals and thus differentiate themselves from the environment and from each other. Conveniently, these tags are often cheap and battery-free, and they are already in wide circulation today for inventory management.

In this thesis, I explore the feasibility of using RFID localization in interactive systems and the benefits and drawbacks of RFID sensing as compared to more conventional methods.

## 1.1 Related Work

There are a few categories of work that have approached this point from different directions. The most closely related efforts, from a technical perspective, are those that deal with high-resolution RFID localization. These include RF-IDraw [22] (which tracks relative motion accurately, but not absolute position), RFind [16] (which computes absolute locations at a low framerate), and TurboTrack [15] (which computes absolute locations quickly but relies on object motion to resolve positional ambiguity). My work builds directly on this legacy, drawing on techniques from RFind and TurboTrack to estimate location quickly and robustly.

Other projects have used RFID tags to faciliate Human-Computer Interaction without localization. These include WISP [20] (which transmits sensor readings as tag IDs), WiSh [11] (which uses phase information to determine the shape of an

object), PaperID [13] (which detects when humans touch RFID tags), and RFIBlocks [14] (which computes the configuration of tagged building blocks). These efforts are in the same vein of synthesis as my project, but they provide more limited measures than tag position. General-purpose localization could enable these applications and more, without requiring modification of existing RFID tags.

Finally, there are a few projects I'd like to mention that allow a human to manipulate objects in the environment to control the state of a computing system. Makey Makey [19] enables a user to map physical touch events (like "squeeze a banana") to more conventional input events (like "press 'K' "). Reactable [12] uses vision to observe the position of marked cubes, which control the state of a musical synthesizer. The Reality Editor [10] uses vision to identify electronic devices in the environment and allows the user to see and manipulate their internal state.

I aim to bridge these categories with my system, which enables high-resolution RFID localization as a source of input for tangible, interactive systems.

## 1.2 Contributions

In this thesis, I present:

1. BackTrack, a complete system for localizing multiple RFID tags quickly and accurately enough to viably support interactive applications, built on off-the-shelf software radios.

2. Characterization and evaluation of key components of the system, including decoding, localization, and tracking.

3. An application programming interface (API) and library that abstract away the details of how the system works, and two demonstration applications using that library.

Like its predecessors RFind and TurboTrack, BackTrack performs wideband localization of off-the-shelf passive RFIDs to compute a tag's location relative to the antennas (rather than determining its relative motion as in RF-IDraw). Unlike RFind,

it can localize *quickly* and *often*, decreasing time per estimate from about 6.4 seconds down to about 0.03, with a corresponding improvement of framerate by over two orders of magnitude—putting the system within a plausible range for real-time interaction [16]. Unlike TurboTrack, it can use a bandwidth beyond 100 MHz for localization; as it relies on the tuning range rather than the sampling rate, this bandwidth can obtained with cheaper hardware [15]. Due to the increased bandwidth, it requires minimal post-processing of location estimates, and can accurately estimate the location of static tags (in addition to moving tags).

More broadly, whereas previous systems served as proof-of-concept, this system takes the techniques that its predecessors proved and optimizes, clarifies, and combines them in a robust architecture. The end result is a complete platform for tracking tagged objects and building interactive applications.

# Chapter 2

# System Overview

In this paper, I present a system for tracking RFIDs for interactive applications. This system is comprised of three major components as shown in Fig. 2-1. From low-level to high-level, these are: software-defined radio hardware and the code that drives it, a DSP component that estimates wireless channels and tag locations, and a library that allows applications to control the operation of the radio and extract computed position data without handling any radio- or protocol-level details. On top of these, I include two small applications that each allow a user to interact with virtual systems by manipulating physical objects in their environment.

The goals of this architecture are ensuring that the system can run the radios continuously (without underflow) and query the RFID tags as fast as possible to

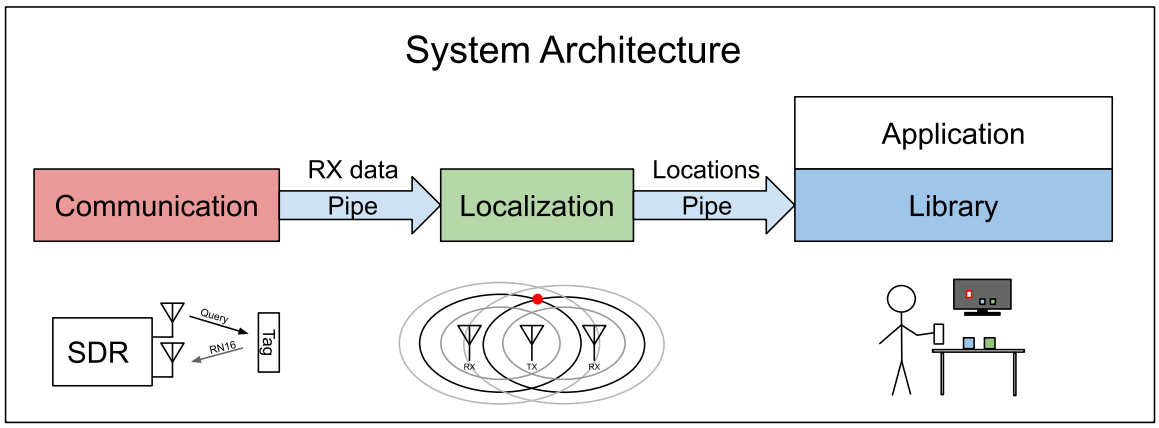


Figure 2-1: System Architecture

maximize throughput, and isolating the end-user application from the rest of the system, so that processing radio data does not block or interrupt user interaction.

Brief descriptions of these three components follow:

1. The lowest-level component is the radio hardware and the code that deals with it. The system is built to run on USRP N210 software radios.[18]. It runs the radios constantly to maximize framerate, and passes on the relevant RX data to the next stage, localization. This component is isolated in its own OS process to minimize the risk of late commands or TX underflow, which can corrupt commands sent to the tag or cause it to lose power. This is the only part of the system that deals with the radio hardware, so it's the only part that needs to change if the system is ported to another SDR platform (such as bladeRF, as discussed in Chapter 7).

2. Next, the localization process gets the raw RX data from the query process via a pipe. It then decodes the RN16 packets to find the reflective and non-reflective segments in each hop, computes channel estimates for each frequency, performs a super-resolution algorithm to compute distance estimates from each antenna pair, and finally performs multilateration to determine the best point of intersection. These distance estimates are then passed on to the application process via the library.

3. The library should provide a simple, high-level API for acquiring location estimates for RFID tags. Hence, it consists of only a handful of functions; these are described in Table 4.1.

On top of these three components, a programmer can build their application. This is the part of the system that faces the user. It can use the library to obtain RFID tag locations, which serve as the user's input to the system, and it may respond by one or more modes (visually, aurally, tangibly via actuators, etc.). The possibilities for this part of the system are limited only by the imagination. In this thesis, I provide two small applications to give a sense of what the system can do.

In the following sections, I describe each component in detail.

# Chapter 3

# Operation

The previous chapter laid out the three main components of the system: the query process, the localization process, and the application library. This chapter describes in detail the operation of the the first two components: how the system communicates with the tag to determine its wireless channel, and how the system uses these wireless channels to localize the tag. The next chapter deals with the remaining component.

At a high level, the system performs the following steps to determine the position of an RFID tag:

1. Power up and query the RFID tag.

2. Simultaneously, transmit a narrowband low-power signal outside of the ISM band, which is also backscattered due to the tag antenna's wide frequency response.

3. Decode the tag's response to determine reflective and non-reflective states, and compute a narrowband channel estimate.

4. Repeat #1–3 at several different out-of-band frequencies to stitch together a wideband channel estimate.

5. Use the wideband channel to estimate coarse distances from each TX/RX antenna pair.

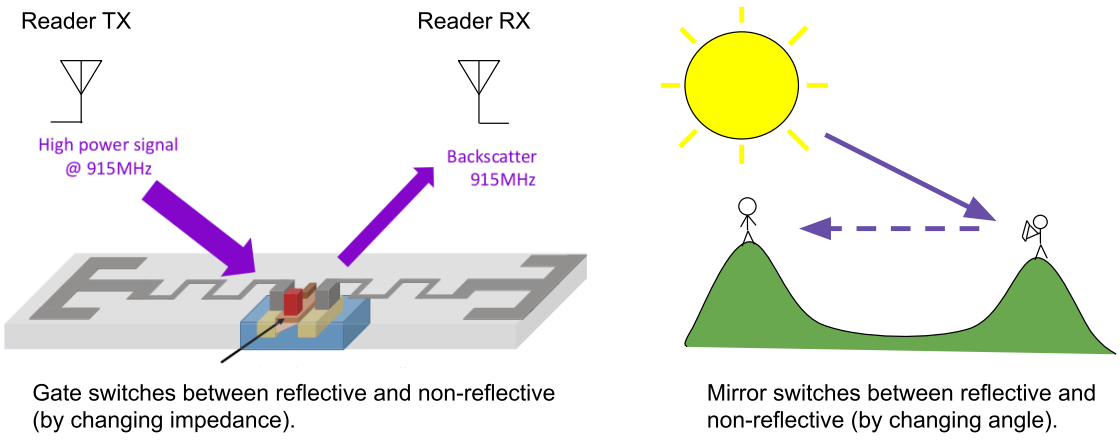


Figure 3-1: The Principle of Backscatte[1]

6. Use a super-resolution algorithm to compute a fine distance from each TX/RX antenna pair.

7. Perform multilateration on the fine distances to estimate location.

The following subsections describe these steps in greater detail.

## 3.1 Backscatter Model

Before I describe how the RFID positioning system works, let me provide a primer on how RFID communication works. This section provides a brief introduction to RFID and introduces the problem of decoding addressed in the next section.

A major part of the appeal of RFID tags is their low power consumption; many tags require no battery and are entirely passive. This capability derives from their use of backscatter communication, as depicted in Figure 3-1. Rather than transmitting their own radio signal, RFID tags switch their antenna impedance to communicate with the reader. This switching between "reflective" and "non-reflective" states, like a signal mirror reflecting the sun's light, enables communication by redirecting external power rather than expending internal power.

Backscatter presents its own challenges. The observable change from the tag's backscattering may be several orders of magnitude weaker than the signal transmitted

[1]Figure on the left adapted from [16] with permission.

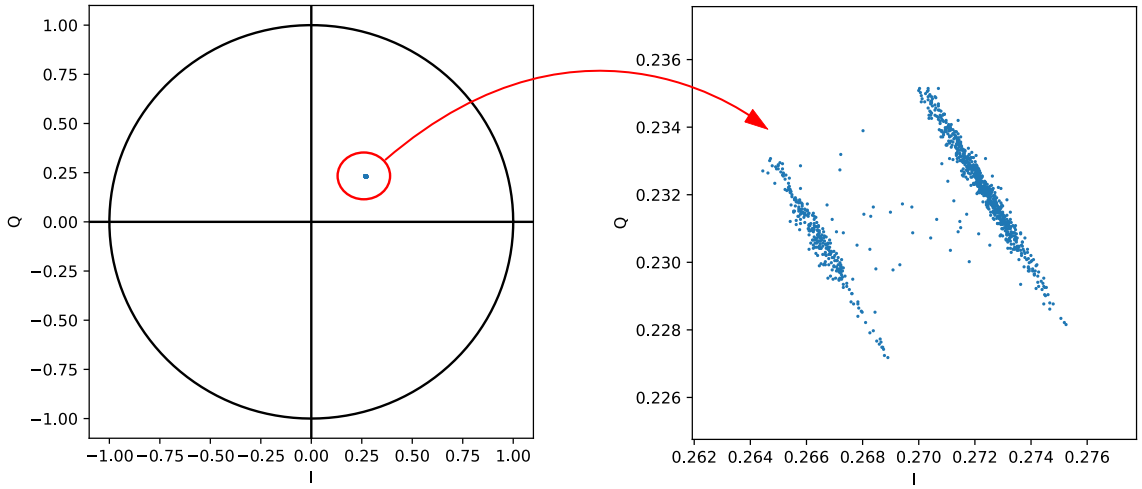


Figure 3-2: Backscatter in the Complex Plane

by the reader, leading to the classic self-jamming problem. This can be mitigated by estimating and subtracting the transmitted signal in hardware, but, as Buettner and Wetherall demonstrated, reads can range up to 6 meters even without this special hardware.[8]

As shown in Figure 3-2, the two tag states (reflective and non-reflective) correspond to two clusters of samples in the complex plane. The difference between those clusters is the tag's channel, which provides physical information about the tag's location relative to the antennas. To localize, we need to compute the channel, which means we need to compute the difference between the reflective and non-reflective states, which means we need to identify which received samples correspond to each state.

Assuming single-tap channels for simplicity, we can model the system's reception of the tag backscatter as:

$$y[m] = (h + s[m]h_t)x[m] + w[m]$$

where $h$ is the underlying environmental channel (defined as the midpoint between the tag's two states for convenience), $h_t$ is the tag's channel, $s[m]$ is the tag's current state (e.g. 1 for reflective, -1 for non-reflective), and $w[m]$ is the noise. $x[m]$ is the transmitted signal, which is a pure tone; once modulated to baseband, we can

disregard it as a constant factor.

In the next two sections, I describe how we can decode the received packet (i.e., recover $s[m]$) as well as estimate the channel over the duration of a packet $h_t$ in order to use it for localization.

## 3.2 Communication

The previous section introduced backscatter and explained the information needed to localize the tag. This section explains how we obtain this information, determining when the tag is reflective or non-reflective (that is, $s[m]$) in order to estimate the channel $h_t$.

### 3.2.1 RFID Coding Schemes

An RFID tag communicates with a reader by switching between reflective and non-reflective states. The EPC Gen2 protocol specifies two coding schemes for tag-to-reader communication: FM0 and Miller. FM0 supports higher data rates at the potential expense of decoding accuracy (more bit errors); Miller, which uses longer symbols for each bit, allows for lower error rates at the expense of the data rate.[2] For simplicity, we consider only FM0 in this thesis. FM0 allows for greater maximum throughput while still allowing for a reasonable spectrum of throughput-accuracy tradeoffs, since we can set the backscatter link frequency (BLF) independently of the encoding scheme.

FM0 encodes bits as in Figure 3-3. '0' and '1' are the same length ($1/BLF$). '0' flips in the middle, while '1' does not. Bits may appear in either orientation, and the amplitude always flips at bit boundaries. The rest of this section considers how to accurately decode FM0 data.

[2]There are three variants of Miller, but all of them are more redundant than FM0.

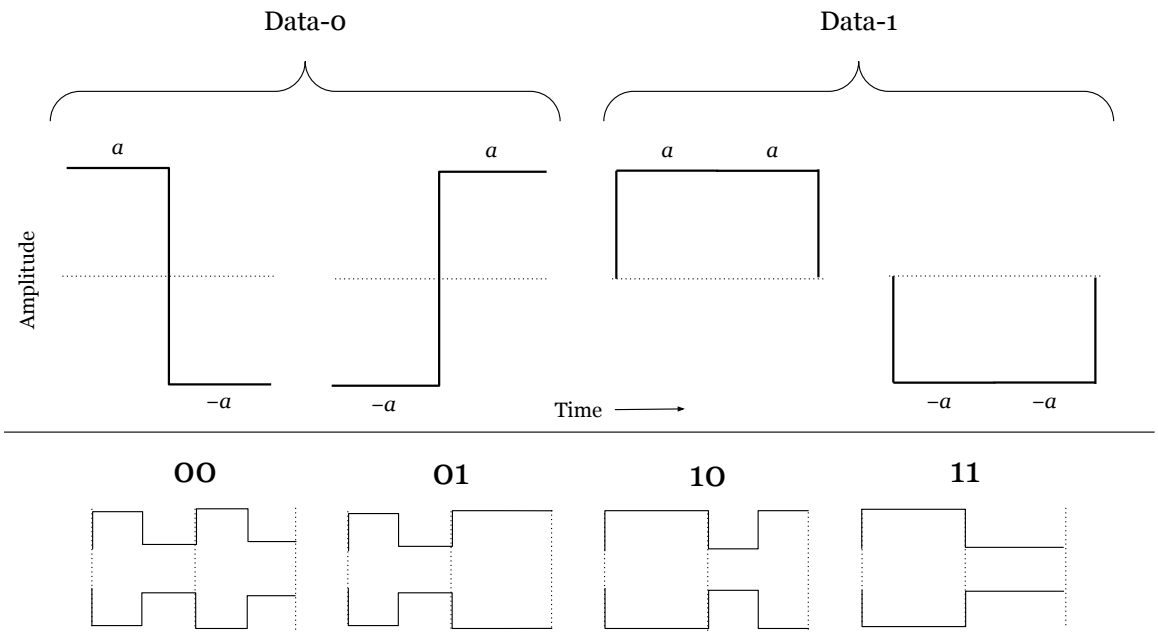


Figure 3-3: In FM0, there are four symbols (top), two each for data-0 and data-1, as each may appear flipped. FM0 requires that there always be a flip between adjacent bits, as shown in the enumeration of possible two-bit sequences (bottom).

### 3.2.2 Packet Parameter Estimation

Before we can decode the bits, we must first determine where they are. Thus, we must perform packet detection to find the start of the packet and estimate the BLF to determine bit boundaries.

The EPC Gen2 specification includes a Tag $\rightarrow$ Reader preamble for packet detection [1]. The preamble includes a violation of the normal FM0 encoding rule to avoid confusion with valid bit-sequences; by correlating with the preamble, we can determine the where the packet begins. Performing the correlation requires that we already have a template for the preamble. The specification gives us the content of the preamble but not the precise BLF of the packet, which (per the specification) may vary from what the Reader requested. This poses a particular problem for longer packets (as in EPC packets, or RN16 packets encoded with Miller rather than FM0).

To decode accurately, we need to estimate both the start of the packet and the backscatter link frequency. Together, these provide us with windows for each reflective/non-reflective state in the packet, which we can then average and decode as

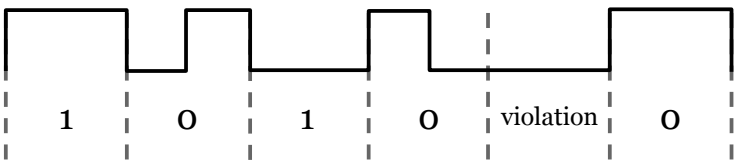


Figure 3-4: FM0 Packet Preamble. The violation (the missing flip between the fourth and fifth bits) allows the preamble to be distinguished from normal FM0 bit sequences.

described in the next section. To estimate these packet parameters, I use a technique akin to those found in GPS systems, which search a 2D space of times and frequencies to find the best match for the received data.[9]

This approach raises two questions:

1. What are the bounds of the search in each dimension?

2. How do we score each point in the search space?

The EPC Gen2 specification answers the first question [1, Table 6-16]. The time $T_1$ between the end of the reader's command and the start of the tag's response is bounded by the formula:

$$MAX(RTcal, 10T_{pri})\times(1-|FrT|)-2\mu s \leq T_1 \leq MAX(RTcal, 10T_{pri})\times(1+|FrT|)+2\mu s$$

where $T_{pri} = 1/BLF$, $FrT$ is frequency tolerance, and $RTcal$ is determined by the reader. For a BLF of 40 kHz and an RTcal of $72\mu s$, this amounts to a range of $67.12\mu s$ to $76.88\mu s$. Additionally, the specification bounds the search space in frequency, specifying a frequency tolerance of $\pm 4\%$ for 40 kHz BLF (38.4 kHz to 41.6 kHz) [1, Table 6-9].

The second question, of how to score each parameter in the search space, is less clear-cut. Correlation with the preamble makes sense and is typically considered sufficient for packet detection, but it leaves out the information provided by the rest of the packet, the contents of which are unknown. However, we know that regardless of the encoded bits, the sign must flip at bit boundaries, and we can use this to calculate correlation with the entire packet despite not knowing its contents.[3]

[3]Note that taking the sign of the amplitude difference at each bit boundary is essentially equiv-

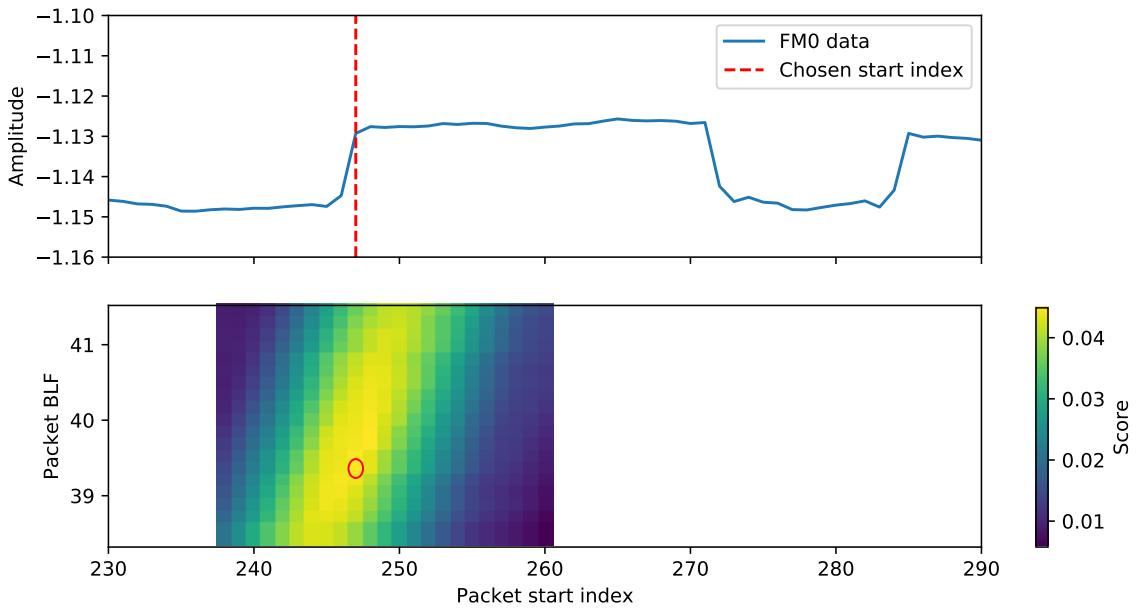


Figure 3-5: Packet Parameter Estimation. On top, the FM0 data; on the bottom, the results of the parameter scoring function, with the chosen parameters circled.

First, we estimate the channel by taking the inner product of the signal with the known preamble (at the candidate offset and frequency). Then we process the rest of the packet, projecting the difference at bit boundaries (which are guaranteed to flip) on to the channel estimate. Depending on the result of this projection (positive or negative), we add the difference or its negation on to the channel estimate. After finishing the packet, the magnitude of the channel estimate, which represents the separation between the presumed reflective and non-reflective states, provides a score. We select the parameter pair with the highest score as the best estimate of the packet parameters.

Fig. 3-5 shows the results of this process. The heatmap shows the amplitude of the search space over BLF and start index, where navy blue a low score and yellow indicates a high score. The figure also shows how the system selects the correct (BLF,index) pair by choosing the highest value in the search space.

Pythonic pseudocode for this process is included in Listing 3.1. Note that a `margin` argument is included which specifies how many samples to ignore at the

alent to deciding the packet contents, as discussed in the next section.

edges of half-bits (as these may occur when the tag is switching between reflective and non-reflective states and weaken the channel estimate).

This approach to parameter estimation uses the entire packet, estimates both the packet start and frequency, and works well in practice. Aside from channel noise, the only thing that could throw off the windows is variation within the packet, but this is tightly limited by the specification [1, Table 6-9] to $\pm 2.5\%$, and (since our BLF estimate is estimated from the whole packet, rather than the preamble alone) the error does not accumulate.

### 3.2.3 Maximum Likelihood FM0 Decoder

Now that we know where the packet is and where the bits are within the packet, we need to decode those bits. The definition of FM0—0 flips in the middle and 1 doesn't—suggests a simple strategy for decoding: look in the middle of each bit and see if it flips. In this section, I demonstrate that, under Additive White Gaussian Noise (AWGN), this decoding strategy is not optimal, and I introduce a simple implementation of the optimal decoder. For simplicity, I assume we are given a real FM0 signal with zero mean and one sample per half-bit.

First, consider the simple approach of checking for a flip in the middle of each bit. Given a real FM0 signal with zero mean, this amounts to sign comparison between the first and second half of each bit. As we assume the bit consists of two samples, this is a sign comparison between the two samples. With this decoding strategy, an error occurs if EITHER noise pushes the first sample OR the second sample to other side of zero (flipping) their sign, but not if it pushes both over.

But this naive approach throws away useful information. In particular, it does not make use of the fact that flips are guaranteed at bit boundaries; equivalently, it fails to take into account the fact that FM0 is actually a stateful encoding. As described in the EPC Gen2 specification, FM0 actually has four states, with transitions depicted in 3-6.

We can take advantage of this statefulness to build a more robust decoder. The flip at the boundary between bits, beyond serving as a convenience for clock synchro-

```
def estimate_parameters(data, packet_length, margin):
    "Estimate the start index and frequency of an FM0 packet."

    # Half-bits in the preamble; 1 = reflective, -1 = non-reflective.
    preamble = [1, 1, -1, 1, -1, -1, 1, -1, -1, -1, 1, 1, -1]
    best_score = -infinity
    best_parameters = (0, 0)
    for blf in FREQ_CANDIDATES:
        step = SAMPLE_RATE / blf
        for i in OFFSET_CANDIDATES:
            channel = 0
            # Compute channel estimate from preamble.
            # This requires the the mean of the preamble is zero;
            # i.e., reflective and non-reflective states are balanced.
            for j in range(len(preamble)):
                start = round(j * step / 2 + i + margin)
                end = round((j + 1) * step / 2 + i - (margin - 1))
                channel += preamble[j] * mean(data[start:end])

            # Refine estimate using bit boundaries in the rest of the packet.
            for b in range(packet_length):
                # Average the half-bit before the boundary.
                left_start = round((6.5 + b) * step + i + margin)
                left_end = round((7 + b) * step + i - (margin-1))
                left_sample = mean(data[left_start:left_end])
                # Average the half-bit after the boundary.
                right_start = round((7 + b) * step + i + margin)
                right_end = round((7.5 + b) * step + i - (margin-1))
                right_sample = mean(data[right_start:right_end])
                # Project difference on to channel estimate.
                diff = right_sample - left_sample
                proj = diff.real * channel.real + diff.imag * channel.imag
                # Update channel estimate.
                channel += sign(proj) * diff

            score = abs(channel)
            if score > best_score:
                best_parameters = (blf, i)
                best_score = score

    return best_parameters
```

Listing 3.1: Packet Parameter Estimation

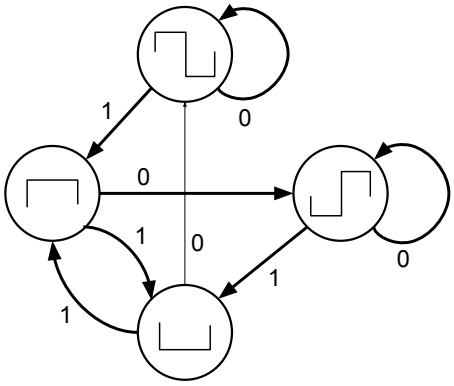


Figure 3-6: FM0 State Machine. Edges indicate transitions based on the next encoded bit. Inside each state is the corresponding encoder output.

nization, can serve as a useful constraint in decoding. Looking at the two samples inside any bit, there are four possibilities allowed by the encoding: HIGH-HIGH (1), LOW-LOW (1), HIGH-LOW (0), and LOW-HIGH (0). But looking at the two samples at a bit boundary, there are only two possiblities allowed by the encoding: HIGH-LOW and LOW-HIGH. Intuitively, because there are fewer legal possibilities *between* bits than within them, it's more reliable to decode the bit boundaries rather than the bits themselves. Once the bit boundaries have been decided, these directly imply the contents of the bits. For example, if a HIGH-LOW boundary is followed by a LOW-HIGH boundary, then the inside must (1).

The above passage is intended to provide an informal description of the optimal decoder and intuition for why it outperforms the simple decoder. A more rigorous analysis follows, assuming Additive White Gaussian Noise (AWGN).

First, consider the naive FM0 decoder, which detects whether there is a flip in the middle of the bit. Assume we have one sample per window. Call the amplitude of the signal $a$, the noise added to the first sample $-n_1$, and the noise added to the second sample $n_2$. Assume we are decoding coherently, so that the noise orthogonal to the FM0 signal is discarded; the channel has noise power $N_0$, but only $N_0/2$ is relevant to our decoding, so $n_0, n_1 \sim \mathcal{N}(0, N_0/2)$.

Detecting a flip means registering if the first and second samples have the same sign (data-1) or opposite signs (data-0). There are four cases to consider, as depicted

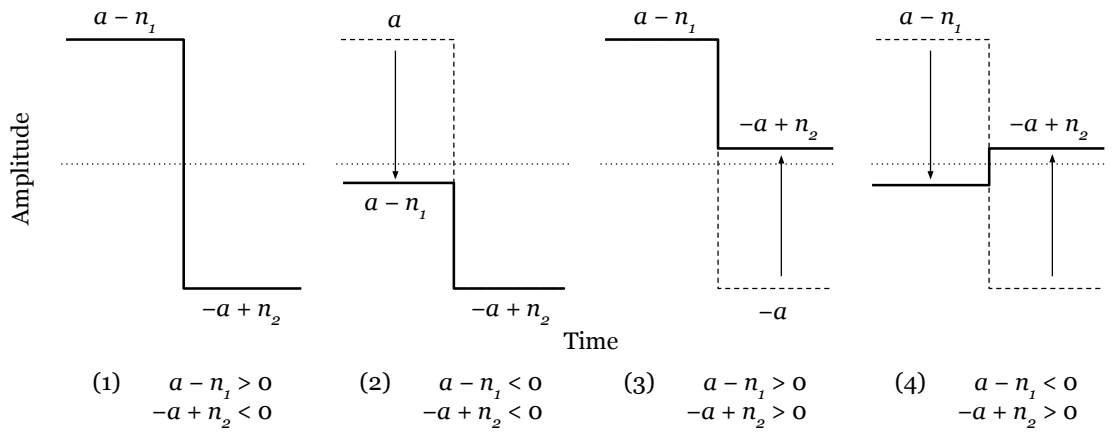


Figure 3-7: Naive FM0 Decoder: Four cases

in Fig. 3-7.[4] In case (1), the effect of noise is small on both samples ($n_1 < a$ and $n_2 < a$), then we see the samples are positive and negative, respectively. The samples have opposite signs, so we correctly classify the bit as a 0. In case (2), we experienced more noise on the first bit, so $n_1 > a$ while $n_2 < a$. Then $a - n_1 < 0$ and $-a + n_2 < 0$, so we observe that both samples are negative and misclassify the bit as a 1. We get the same result in case (3), where $n_2 > a$ while $n_1 < a$: both samples are positive, so we decode a 1. In case (4), the noise is large on both samples, so that $n_1 > a$ and $n_2 > a$. In this case, both samples flip from their "intended" signs and we decode a 0 again.[5]

Let $p$ be the probability that $n_1 > a$. As $n_1$ and $n_2$ are drawn from the same distribution independently, this is also the probability that $n_2 > a$. An error occurs if either $n_1 > a$ or $n_2 > a$, but not both. $P(n_1 > a \vee n_2 > a) = p + p - p^2$; to account for the "but not both", we subtract out $P(n_1 > a \wedge n_2 > a) = p^2$ to get $P(n_1 > a \oplus n_2 > a) = 2p - 2p^2 = 2p(p-1)$, which is the probability of a decoding error with this scheme. $n_1, n_2 \sim \mathcal{N}(0, N_0/2)$, so $p = P(n_1 > a) = Q(\frac{a}{\sqrt{N_0/2}})$ and the bit error rate is $2Q(\frac{a}{\sqrt{N_0/2}})(1 - Q(\frac{a}{\sqrt{N_0/2}}))$.

We can quantify the performance of this decoding scheme in terms of SNR (signal-to-noise ratio) and BER (bit error rate); SNR is a common metric for measuring the quality of an information channel, and determining BER as a function of SNR is a

[4]WLOG, we assume the bit in question is actually a data-0.

[5]In this case, we make two mistakes due to noise but they cancel out; we're so wrong, we're right.

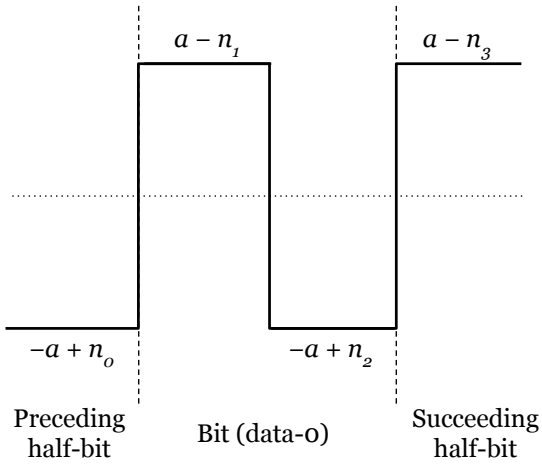


Figure 3-8: An FM0-encoded bit and its neighboring half-bits, with noise.

standard way to quantify the performance of a decoding scheme. For further reference, see [21]. For this naive decoding scheme, the SNR (defined as signal power divided by the noise power per symbol) is $a^2/N_0$, and thus the BER, as a function of SNR, is $2Q(\sqrt{2SNR})(1 - Q(\sqrt{2SNR}))$.

Now let's consider the maximum likelihood decoder. There are a few ways of looking at it. One way is the view the task still in terms of checking the middle of the bit for a flip, but now with error-correction via the boundary-flip constraint. Let's zoom out from the bit we've looking at to see around it:

Because FM0 guarantees sign flips at bit boundaries, we expect that $-a + n_0$ and $a - n_1$ have opposite signs. If that's not the case, one sample must have pushed to the wrong side of the line by noise, and we should correct for it. But which is it—the second sample of the preceding bit, or the first sample of this bit? Since noise values closer to 0 are more likely (with a Gaussian distribution), it's more likely that whichever sample has the smaller magnitude is the one that has been pushed over the line. Thus, before we use the samples in the current bit to decode it, we can correct them against the adjacent samples.

After we've applied the correction, we're back where we were before: we have two boolean values. If exactly one is wrong, the decoded bit is wrong; if zero or both are wrong, the decoded bit is right. We end up again with $2p(1 - p)$, where $p$ is

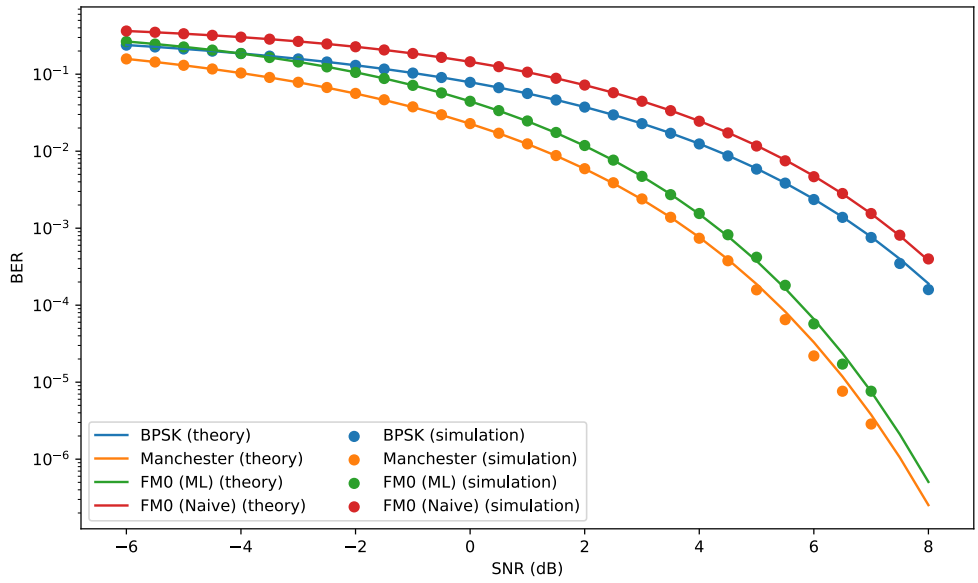


Figure 3-9: BER vs. SNR in theory and simulation. Simulation conducted by encoding $2^{20}$ random bits, injecting the appropriate amount of Gaussian noise for the desired SNR, and decoding.

the probability of one of our values being flipped. What is $p$ now, after boundary-correction?

With correction, what matters is that $-a+n_0 < a-n_1$. They can have the same sign, but as long as $-a+n_0 < a-n_1$, we can correctly decide that the first sample of the bit, $a-n_1$, should be considered HIGH. We'll make an error only if the relative ordering of the samples is switched, so that $-a+n_0 > a-n_1$. Rearranging terms shows that this occurs when $n_0+n_1 > 2a$. As $n_0, n_1 \sim \mathcal{N}(0, N_0/2)$, the sum $n_0+n_1 \sim \mathcal{N}(0, N_0)$. Thus, $p = P(n_0+n_1 > 2a) = Q(\frac{2a}{N_0})$. Plugging this into the expression above, we have an overall error probability of $2p(1-p) = 2Q(\frac{2a}{N_0})(1-Q(\frac{2a}{N_0}))$, which in terms of SNR is $2Q(2\sqrt{SNR})(1-Q(2\sqrt{SNR}))$. Compared to the naive decoder, this yields the same error rates with half the SNR, providing an effective gain of $10\log_{10}2 \approx 3dB$ over that decoding scheme.

These relationships are summarized in Fig. 10. Experimental results are included in section 5.1.

A few points worth mentioning before moving on:

- FM0 is stateful, and can be viewed as a simple convolutional code. In general, the maximum likelihood decoder for a convolutional code is the Viterbi decoder, which employs dynamic programming. In the case of FM0, this can be simplified to the scheme described above, since any half-bit can constrain only the half-bit on the other side of its neighboring boundary.

- In other words, the contents of any FM0 bit are not independent of the contents of neighboring bits, due to the guaranteed flip at boundaries—but the half-bits on either side of a boundary are independent of the rest of the transmission.

- The naive decoding scheme is equivalent to BPSK decoding the half-bits followed by XORing non-overlapping bit pairs, while the ML decoding scheme is equivalent to Manchester decoding the boundaries followed by XORing overlapping bit pairs. The gain provided by the ML scheme over the naive scheme corresponds to the gain provided by Manchester over BPSK.

## 3.3 Localization

### 3.3.1 Wideband Channel Estimation

Obtaining channel information across a wide bandwidth is essential to localizing accurately. TurboTrack [15] used OFDM and software radios with a large instantaneous bandwidth to get channel estimates spanning 100 MHz; unfortunately, a large instantaneous bandwidth is both expensive and strictly limited by hardware. TurboTrack's predecessor, RFind [16], instead stitched together a wideband channel estimate from many narrowband estimates by retuning the RF frontend between queries. Unfortunately, this process was relatively slow, and produced roughly one location estimate every 6.4 seconds, making it unsuitable for interactive applications.

This system follows RFind's frequency-hopping approach (heavily optimized, as discussed in Secs. 3.4.1 and 4.2.2), enabling it to use a large bandwidth on cheaper hardware. While powering up and querying the RFID at 925 MHz, the system simultaneously transmits a low-power signal at another frequency: 800 MHz, then 820,

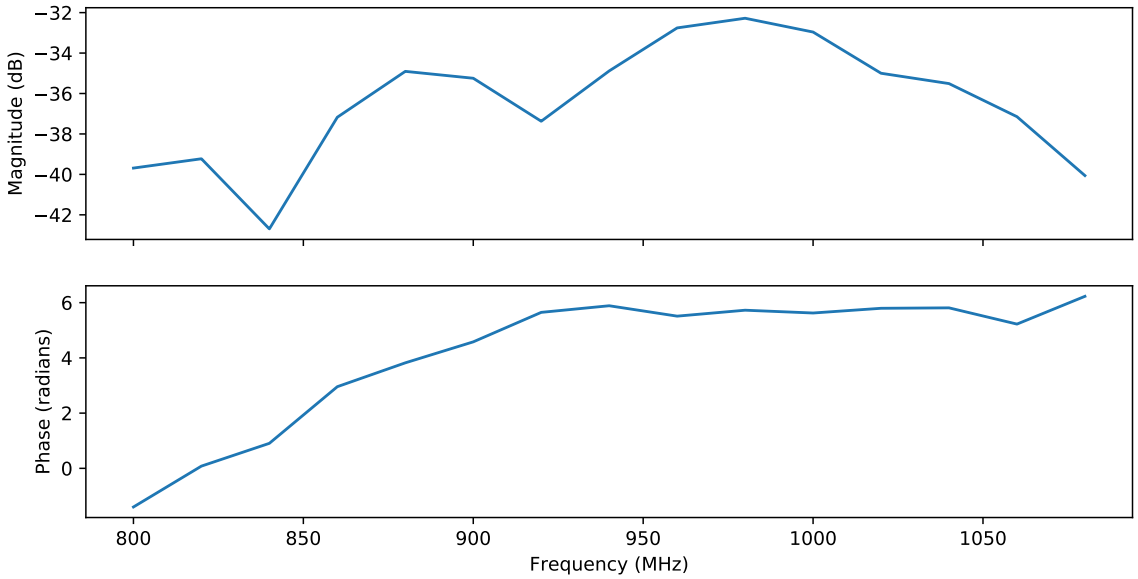


Figure 3-10: A wideband channel estimate, stitched together via frequency-hopping

840, and so on up to 1100 MHz.

At each out-of-band frequency, we decode the tag's in-band response (as described in Sec. 3.2) and use it to determine the portions of the signal where the tag is reflective or non-reflective ($s[m]$, from Eq. 1). We then average the corresponding samples in the out-of-band signal to obtain two channel estimates, corresponding to the reflective and non-reflective states of the tag. We subtract one from the other to obtain $h_t$, a narrowband estimate of the tag's wireless channel.

Computing narrowband estimates at all 15 out-of-band frequencies yields a wideband channel estimate $h_t[k]$, for $k = 0 \ldots 14$.

This process must be performed once with the tag at a known location to obtain calibration data. Subsequent channel estimates are divided by the calibration estimate, which corrects for the uneven frequency responses of the antennas (the system's and the tag's) and the part of the channel from the SDR to the antennas (which provides no information as to the tag-antenna distance). After division, the quotient corresponds to the *difference* between the tag's distance and its distance at calibration, so we add the calibration distances back in the next step of the process.

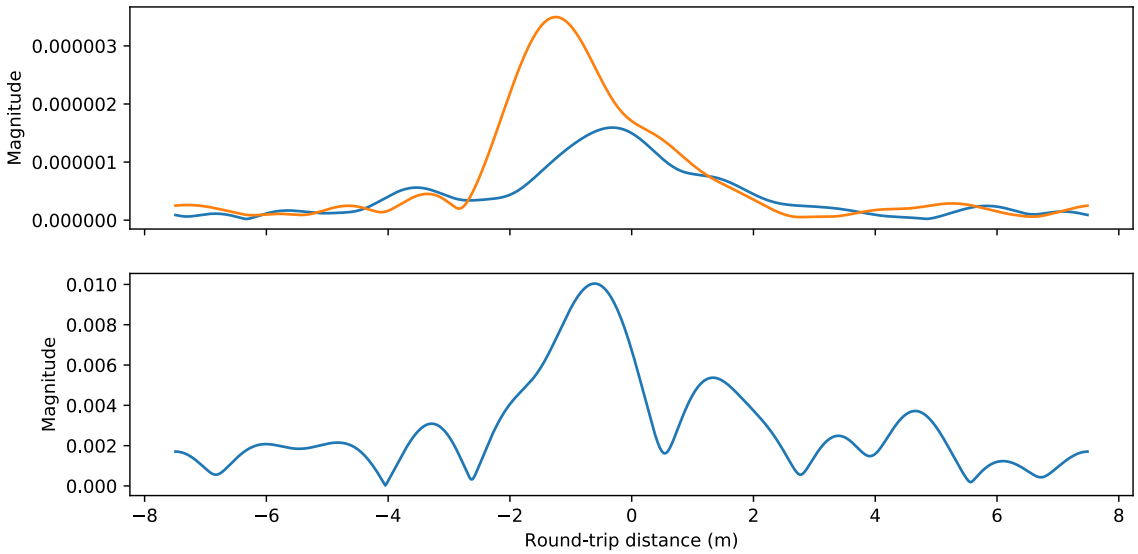


Figure 3-11: The IFFTs of two channel estimates captured at different distances (top), and the IFFT of one divided by the other (bottom). Division is deconvolution in the time domain, so the peak in the bottom plot corresponds to the difference in distance between the two measurements.

### 3.3.2 Distance Estimation

The wideband channel estimates, covering a bandwidth of 300 MHz, can provide us with a round-trip distance resolution of $\frac{c}{BW} = \frac{2.997\times10^8 m/s}{3\times10^8 Hz} \approx 1m$ (amounting to a one-way resolution of 0.5m), but this is insufficiently accurate for fine human control. Obtaining centimeter-scale distance resolution from this alone would require a bandwidth of $\frac{c}{.02m} = 15GHz$, which is infeasible for both technical and regulatory reasons. Instead, this system makes the most of its bandwidth by employing the same super-resolution technique developed in recent wideband RFID localization systems:

1. Perform an interpolated IFFT on the wideband channel estimate to obtain a coarse distance estimate.

2. Use the coarse distance estimate with the IFFT to obtain filtered phases.

3. Use the filtered phases to estimate a fine distance estimate.

The filtered phases are then used to bridge the gap to fine resolution on the assumption of sparsity of multipath near the tag. The essence of this technique was

introduced and described in prior work [16, Section 4.2], but it is sufficiently important to this system that I describe it here and highlight some optimizations.

### Motivation

To understand this procedure, consider the use of phases in localization. In an ideal environment with no multipath, measured phases correspond directly to the line-of-sight (LOS) distance. If the distance relative to the calibration distance was $d$, then the phase $\phi$ at wavelength $\lambda$ would be $\frac{2\pi d}{\lambda} \mod 2\pi$. This would allow computing precise distances (limited by the precision of the tuning and phase measurement), but it would result in ambiguities at each multiple of $\lambda$. Measuring another phase $\theta$ at a slightly larger wavelength, $\lambda + \epsilon$, would eliminate much of that ambiguity, because each possible $(\phi, \theta)$ combination would recur only once every $\lambda/\epsilon$ repetitions of the longer wavelength, making distances ambiguous at multiples of $\frac{\lambda^2}{\epsilon} + \lambda$ rather than $\lambda$.[6]

This approach would work just fine and simplify localization greatly: rather than many hops over a wide bandwidth, we could make do with just two closely-spaced channel estimates. Staying in band and querying at 900 and 901 MHz would grant us $\frac{c}{901MHz - 900MHz} \approx 300m$ of unaliased range with accuracy limited only by our precision in measuring phase. So, why doesn't every RFID reader come with fantastic localization capabilities?

The problem is multipath. All of the above works accurately, quickly, and with a narrow bandwidth *on the assumption* that phases correspond to the line-of-sight distance. In practice, they do not: the signal takes many paths on its route from the TX antenna to the RX antenna, bouncing off objects and walls, each path traversing a different distance, each producing a different phase at the receiver. The result is a garbled sum of complex exponentials with a phase that may differ greatly from that of the LOS path.

A system entirely reliant on phase would need vanishingly little bandwidth, but

[6] This is the principle behind aliasing in general; more tightly spaced samples reduce ambiguity and allow for a larger unaliased domain.

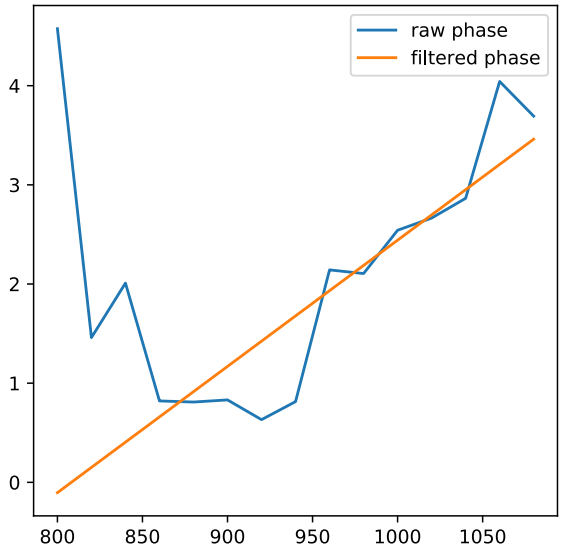


Figure 3-12: Phases before and after filtering

it would have no way to deal with this problem. A system that didn't rely on phase wouldn't have this problem in the first place, but it would require an enormous bandwidth. The super-resolution algorithm allows this system to strike a happy medium: use a wide bandwidth to filter out much of the multipath inteference from the phases, and then use the filtered phases to estimate distance.

### Filtering Phases

RFind presented the following phase-filtering transform, with a proof of its ability to suppress multipath:

$$\theta_k = \angle \sum_{i=1}^{K} h_k e^{j\frac{2\pi}{c}(f_i - f_k)d_0^c}$$

where $d_0^c$ is the coarse distance estimate corresponding to the LOS peak in the IFFT.[7]

An intuitive interpretation of this procedure is zeroing out everything except the LOS path in the time domain, in order to suppress the effect of multipath from

[7]The LOS peak is estimated as the earliest peak—the first peak that occurs at a distance greater than the minimum possible distance, which is the distance between the TX and RX antennas. In practice, the IFFT has sidelobes due to windowing, so the actual peak chosen is the earliest one above a certain threshold, relative to the largest peak (which might not correspond to the LOS).

non-line-of-sight paths on phases in the frequency domain. More precisely:

1. Take the interpolated IFFT of the wideband channel estimate, as mentioned before.

2. Find the peak corresponding to the LOS path and zero-out every other bin in the IFFT.

3. Take the FFT of this modified array and return the phases of the first 15 bins.

This interpretation suggests that we can use the value from the IFFT peak directly rather than re-processing the wideband channel estimate. And this special case, in which we take the FFT of an array with only non-zero value, is well-known: the FFT of an impulse is constant across frequency, while the FFT of a delay has constant magnitude and linear phase. By the shift theorem, $x[n-l] \leftrightarrow e^{\frac{-j2\pi kl}{N}} X[k]$. Given the IFFT as $x[n]$ and the index of the LOS peak as $i$, we can compute the filtered phases directly as:

$$\theta_k = \angle \left( x[i] e^{\frac{-j2\pi ki}{N}} \right) \equiv \angle x[i] - 2\pi k \frac{i}{N}$$

**Fine Distance Estimation**

Now we can find the distance estimate that best matches the filtered phases $\theta_k$. RFind proceeded with a clustering optimization algorithm that grouped distances from each frequency together and computed their cluster error as deviation from the mean distance. But with the knowledge that the filtered phases are necessarily linear, we can solve for the best distance directly.

Denote the first frequency of the hops $s$, and the step size between hops $t$. Then the $kth$ frequency is $s + kt$, and the $kth$ wavelength $\lambda_k = \frac{c}{s+kt}$. With the phase $\theta_k = \angle x[i] - 2\pi k \frac{i}{N}$, we then compute the distance modulo $\lambda_k$ as

$$d_k \equiv \lambda_k \frac{\theta_k}{2\pi} = \frac{c\left(\angle x[i] - 2\pi k \frac{i}{N}\right)}{2\pi(s+kt)} \mod 2\pi$$

Replacing the $\mod 2\pi$ with an integer phase ambiguity term $n$ yields:

$$d_k = \frac{c\left(\angle x[i] - 2\pi k\frac{i}{N} + 2\pi n\right)}{2\pi(s+kt)} = \frac{c\left(\frac{\angle x[i]}{2\pi} - k\frac{i}{N} + n\right)}{s+kt}$$

The question is then, what choice of $n$ brings the distances for each frequency $d_k$ into agreement best? The ideal would be perfect agreement, where $d_k$ is constant across all frequencies. It turns out we can find this by solving for $n$ constant across all frequencies ($\frac{dn}{dk} = 0$):

$$n = \frac{d_k(s+kt)}{c} + k\frac{i}{N} - \frac{\angle x[i]}{2\pi}$$

$$\frac{dn}{dk} = \frac{d_k t}{c} - \frac{i}{N} = 0$$

$$d_k = \frac{c}{t}\frac{i}{N}$$

$$n = \frac{i(s+kt)}{tN} - k\frac{i}{N} - \frac{\angle x[i]}{2\pi} = \frac{si}{tN} - \frac{\angle x[i]}{2\pi}$$

This choice of $n$ yields $d_k = \frac{c}{t}\frac{i}{N}$, which is independent of $k$ and thus constant across all frequencies (corresponding to 0 cluster error). Of course, this distance is exactly the coarse distance estimate: $\frac{c}{t}$ is the unaliased range (the range the IFFT encompasses) and $\frac{i}{N}$ is the normalized index of the LOS peak in the IFFT.

But this exact solution for $n$ is not an integer, and we need an integer number of wavelengths to reconstruct a distance estimate from phase. The difference between each $d_k$ will only grow as we move away from the exact solution, so the lowest-error cluster can be found directly by rounding $n$ to the nearest integer and averaging the corresponding $d_k$s.

One last step remains. The nearest integer $n$ isn't always the best distance estimate, and sometimes the coarse estimate is right between two clusters, such that noise can easily cause to system to vacillate between them. To avoid this, the system employs a simple Markov model of motion at the end of distance estimation, using the

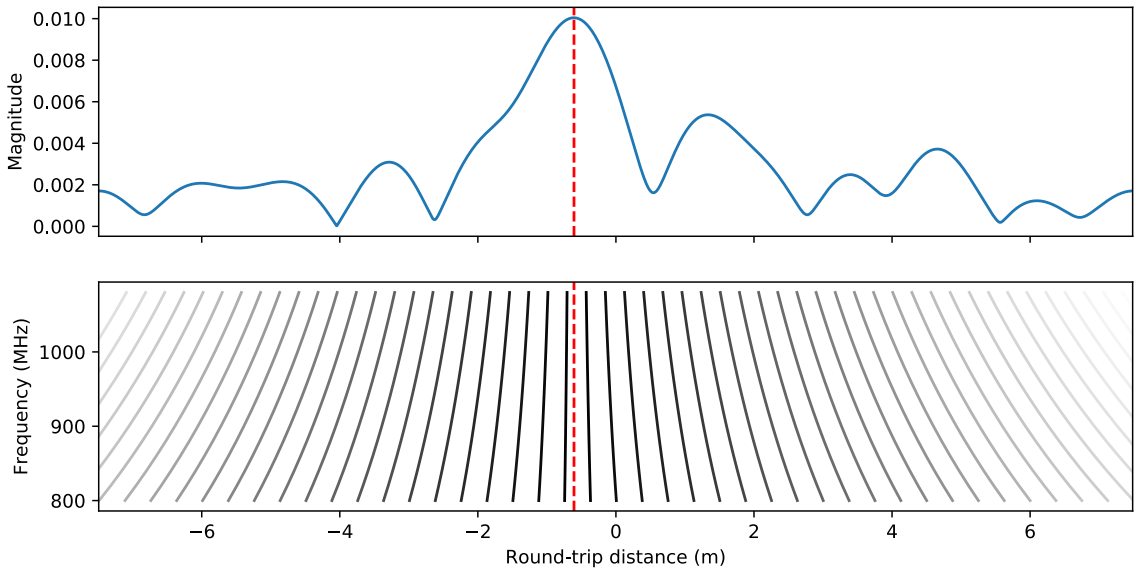


Figure 3-13: The IFFT magnitude (top), with the coarse distance estimate indicated by the dashed line. The phase clusters (bottom, gray lines) deviate more further from the coarse estimate. The line closest to the coarse estimate has the least error and is the best choice for our final distance estimate.

best two distance candidates (the floor and ceiling of the exact solution for $n$). For each new measurement, it uses the forward algorithm to determine the probabilities for the new candidates and picks the highest one to pass in to the next stage.[8]

### 3.3.3 Location Estimation

Once we have distance estimations for every out-of-band TX/RX antenna pair, we combine them through multilateration to compute a 2D or 3D location estimate (depending on the configuration). Unlike previous systems, which computed exact solutions for pairwise intersections with a fixed antenna configuration, this system treats multilateration as an optimization problem and employs a standard non-linear least squares solver [2] to compute the final location.

For $N$ out-of-band receivers, there are $N$ residuals $R$. $R_k$ denotes the Euclidean distance between the candidate point $p$ and the edge/surface of the ellipse/spheroid

[8]If the new probabilities are too low (indicating a disjuncture between consecutive measurements, possibly due to the tag exiting or entering the operating region), the system resets the probabilities to avoid getting stuck.

with focal points at the TX antenna and the $k$th RX antenna and round-trip distance $d_k$, so:

$$R_k(p) = |\vec{p} - \vec{a}_{TX}| + |\vec{p} - \vec{a}_{RX,k}| - d_k$$

The non-linear least squares solver numerically finds a point $p$ that minimizes the sum of the squared residuals $S$:

$$S(\vec{p}) = \sum_{k=0}^{N-1} R_k(\vec{p})$$

In the critically constrained case, there may be ambiguous solutions; these can be eliminated with appropriate bounds on the search space (e.g. the constraint $y > 0$, corresponding to directional antennas).

This approach has the virtues of not being tied to a particular antenna configuration, code reuse between the 2D and 3D cases, and good performance when the problem is overconstrained (by e.g. having more out-of-band receivers than spatial dimensions).

## 3.4 Tracking

The previous sections described the steps necessary to compute one location estimate. This section bridges the gap between getting a single location estimate for a single tag and getting many location estimates for many tags. Interactive applications typically require low latency ($<$ 100ms) to produce a responsive, usable experience, and this section consider how to achieve this.

### 3.4.1 Framerate Optimizations

As noted in 3.3.1, this system stitches together a wideband channel estimate from several narrowband estimates, which requires hopping the out-of-band TX/RX from one frequency to another and querying the tag to get an FM0 packet at each frequency. This approach is inherently slower than one based on a large instantaneous bandwidth,

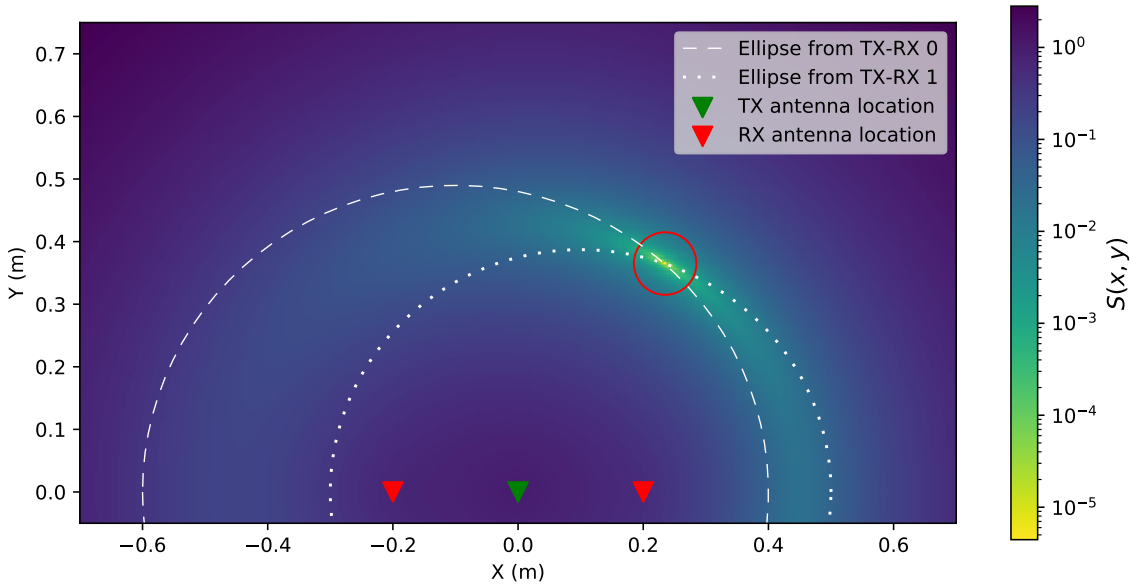


Figure 3-14: Multilateration with two out-of-band receivers. The function to minimize is shown as a log heatmap. The optimal point (corresponding to the intersection of the ellipses) is circled in red. Note that there is another solution with $y$ negated, but it is ignored because it is behind the antennas.

which can compute a full wideband estimate from a single packet. This difference accounts for some of the performance difference between RFind (which computes one location every 6.4 seconds) and TurboTrack (which computes 300 locations per second). I found that I was able to optimize the hopping approach considerably; this is discussed in Sec. 4.2.2. In this section, I discuss higher-level optimizations relating to the EPC Gen2 protocol.

RFind and TurboTrack generated a single query and repeated it indefinitely; this means that each query started a new communication session with the tag and had to use a full-length Query command (and a Select, in multi-tag scenarios), and each command was preceded and succeeded by a long charge gap, as if powering up the tag for the first time. BackTrack instead treats each round of hops as a single round of communication with a tag; it keeps that tag powered, and rather than repeating a Query command, it issues a Query once to begin the session and follows it up with QueryAdjusts.[9] This improves throughput because QueryAdjusts are shorter

[9]Unfortunately, the even-shorter QueryRep command is unsuitable for this purpose, as it causes the tag's internal counter to underflow and prevents the tag from responding to repeated queries in

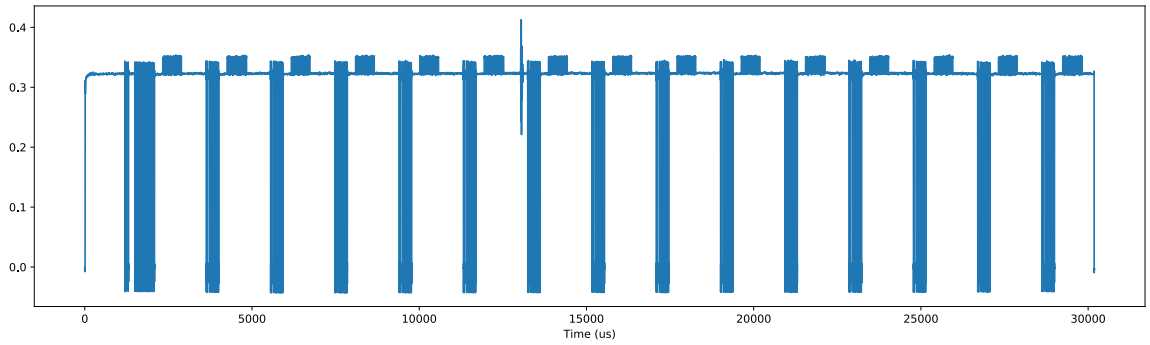


Figure 3-15: One full round of communication: one Query, followed by 14 QueryAdjusts

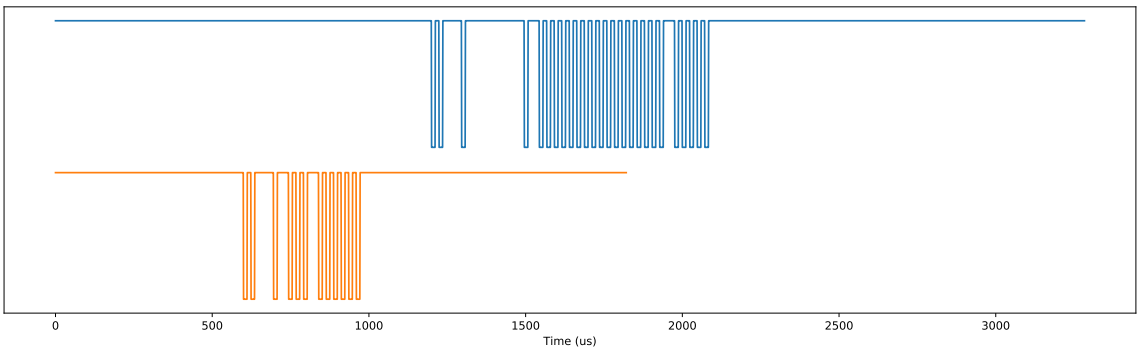


Figure 3-16: The repeated transmission for TurboTrack (top), vs. the repeated QueryAdjust (bottom)

than Queries: the packet contains 9 bits, rather than 22 [1, Table 6-28], and it omits calibration pulses required to initiate a round. BackTrack also does away excessive charge gaps: it powers the tag up at the start of the round and maintains power for the duration of the round. These efforts reduce the average duration of one query-response from 3.284 ms down to 1.896 ms, increasing throughput from about 305 queries per second up to 527.

Further protocol-level optimizations are possible (such as increasing the frequency of reader commands or tag responses), but these would likely trade-off other desirable metrics such as range or accuracy for only marginal benefit—and, following the hopping optimizations described in Sec. 4.2.2, the main bottleneck is tuning time rather than communication time.

the same session.

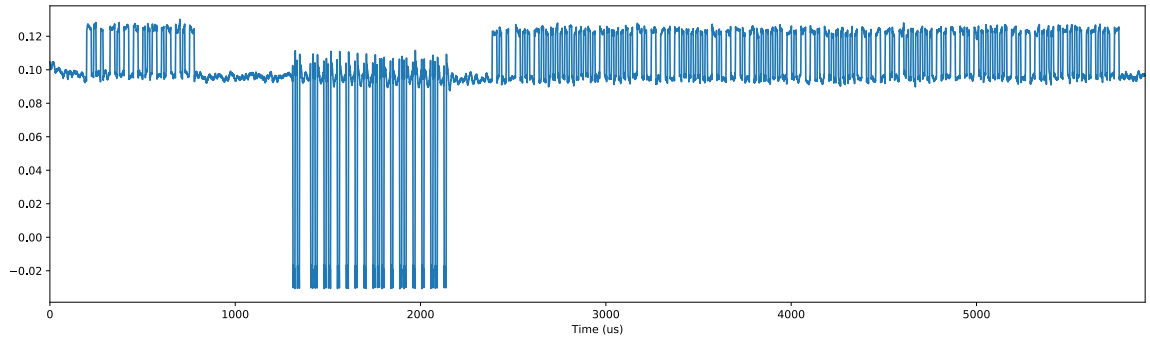


Figure 3-17: Reading a tag: RN16, ACK, EPC

### 3.4.2 Multi-tag

An essential capability of RFID localization—and an advantage over other radio-sensing technology—is that RFID tags can be targeted individually. This means that they can be queried and localized one at a time without requiring the system disentangle one tag's response from another (or from other objects in environment). BackTrack supports tracking an arbitrary number of tags, with protocol-level optimization to reduce the impact on throughput, and provides utilities to ease working with multiple tags.

To work with multiple tags, we must first determine each tag's identifier ("EPC", for Electronic Product Code). Previous localization systems relied on a separate RFID reader to handle this aspect of communication, but this system handles it itself. The main challenge here is in decoding a tag's RN16 and replying with an ACK in a timely fashion; the EPC Gen2 specification puts a tight deadline on this turnaround of $500\mu s$.[1, Table 6-1] To make this deadline reliably, I implemented the turnaround functionality in C++ (see 4.2) and tweaked the SDR network settings to reduce latency. The end result is the API function `backtrack.get_epc()`, which queries the tag, decodes the RN16, replies with an ACK, and decodes the tag's EPC for future reference.

That EPC can then be passed to other functions. `backtrack.calibrate(epc)` estimates the tag's channel at a known location and saves it with the tag's EPC. `backtrack.init(epcs)` takes a list of EPCs, loads the calibration data for each, and

computes minimal selectors for future queries. The protocol allows specifying which tags should respond with the Select command, which can filter tags by their EPC. In particular, the Select command can be used to filter on any substring of the full 96-bit EPC. The system takes advantage of this to make the Select commands as short as possible for the set of tags to be tracked, which improves throughput when tracking multiple tags.

# Chapter 4

# Implementation & Evaluation

## 4.1 Hardware

My experimental setup consisted of the hardware depicted in Fig. 4-1: three USRP N210s [18] with SBX [17] daughterboards in my apartment. One USRP handled in-band RX/TX; one handled out-of-band RX/TX; and the third provided an extra out-of-band RX channel for 2D trilateration.

I used three MT-242025 antennas. I connected one of these to a ZAPD-2-21-3W-S+ power combiner which in turn connected to the in-band TX and out-of-band TX. I connected another antenna to a ZAPD-2-21-3W-S+ power splitter which in turn connected to in-band RX and one out-of-band RX. I connected to the last antenna directly to the other out-of-band RX. The USRP N210s were synchronized via an OctoClock-G and connected to a NETGEAR GS110MX network switch, which in turn connected to my laptop, a System76 Galago Pro with an Intel i7-7500U processor. I used SMARTRAC Belt paper tags in my experiments, adhered to objects such as paper cups, plastic bottles, cardboard boxes, and whiteboard markers.

## 4.2 Software

The system is implemented primarily in Python, using the Python 3 bindings for the USRP's UHD driver (version 3.15.0.0) to control the radios. Some components

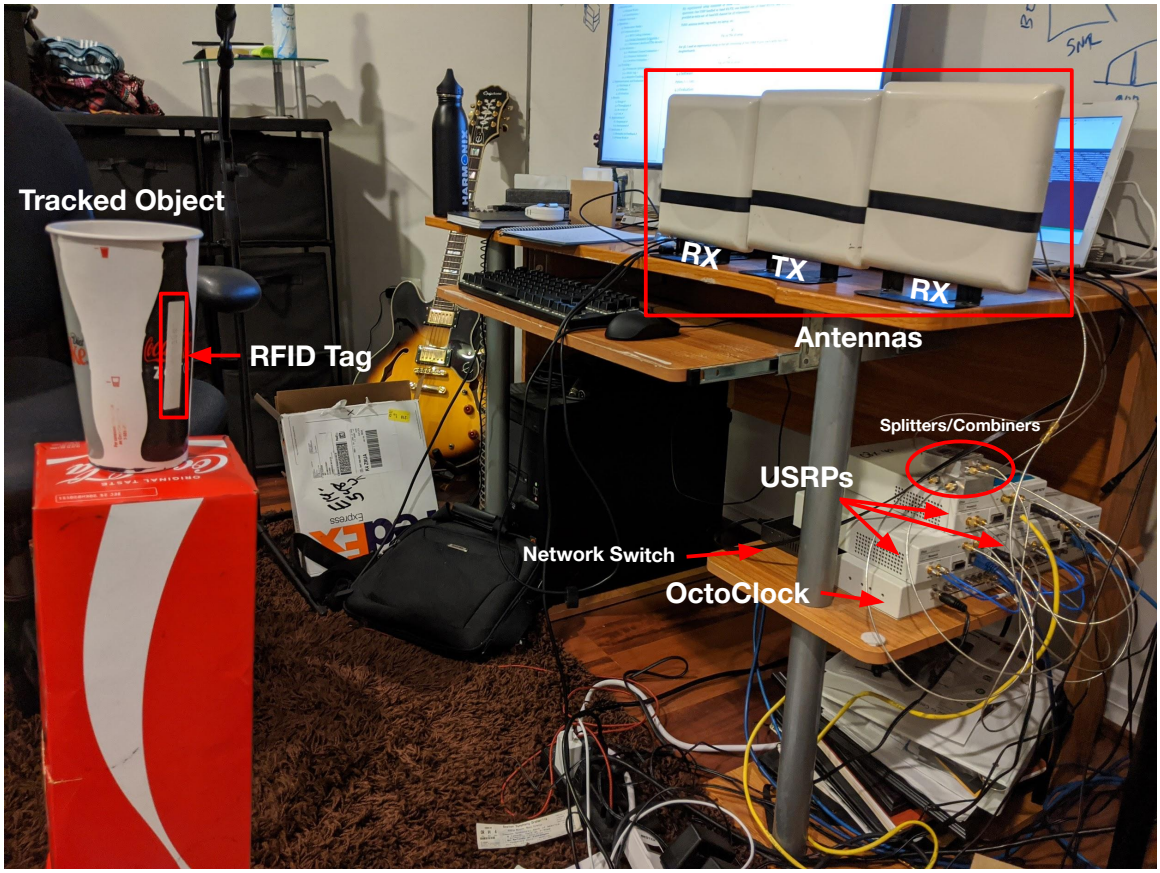


Figure 4-1: Home setup

of the system are implemented in C++ for speed; these include decoding, channel estimation, and the Query-ACK turnaround. The C++ components are called directly from Python, using the pybind11 library (which Ettus uses to provide UHD bindings). Among other libraries, the system makes extensive use of `numpy`, `scipy`, and `matplotlib` (for debugging and generating the figures here). The applications are built with `pygame`, `PyAudio`, and `pynput`.

### 4.2.1 Protocol

In implementing the EPC Gen2 protocol to communicate with RFID tags, the system chooses a backscatter link frequency of 40 kHz and a Tari length of $24\mu s$. Unless otherwise specified, these are the values assumed in all protocol-related computations above.

### 4.2.2 Throughput

In order to achieve framerates suitable for interactive applications, BackTrack employs several optimizations to improve throughput. Some protocol-level optimizations were discussed in Sec. 3.4.1, but the most significant optimization concerns the use of the radio hardware. Whereas the RFind authors reported that they could switch to a new frequency once every 130 ms, Bell's work [6] suggests that, with retuning commands scheduled in advance, the N210 with an SBX daughterboard can achieve tuning rates more on the order of hundreds of microseconds rather than hundreds of millseconds. I confirmed this with my own setup and drastically reduced the time between hops. Further, BackTrack uses fewer hops than RFind; RFind used 22 hops over 220 MHz (spaced 10 MHz apart), while I use 15 hops over 300 MHz (spaced 20 MHz apart). Sampling in the frequency domain leads to aliasing in the time domain, but a step size of 20 MHz still allows $\frac{c}{20MHz} \approx 15m$ without aliasing (7.5m one-way).

Table 4.1: Library API

| Function | Description |
|---|---|
| `get_epc()` → `EPC`[1] | Read a tag and return its EPC (Electronic Product Code); this is used to identify the tag in other API functions.<br>Assumes there is only one tag in read range. |
| `calibrate(epc)` → `None` | Perform channel estimation on the given tag, and save the result as calibration data. Assumes tag is in the designated calibration location.<br>The user must call this once before tracking tags. (But once is enough, across multiple runs, restarts, etc.) |
| `init([epcs])` → `None` | Initialize the system and prepare to track the given tags. |
| `start()` → `bool` | Start the query and localization processes. |
| `stop()` → `bool` | Stop the query and localization processes. |
| `get_location(epc, block=False)`<br>→ `Location`[2] | Return the given tag's most recent location for the given tag. If `block` is true, waits for a fresh location.<br>This communicates with the localization process, pulling any location estimates since the last time the function was called.<br>The application is responsible for calling this function regularly to avoid pipe build-up. |

[1] An `EPC` is a big integer that stores the tag's 96-bit identifier.
[2] A `Location` is 2D or 3D floating-point vector relative to the origin, in meters.

### 4.2.3 Library

As mentioned in Ch. 2, BackTrack includes a library for acquiring location estimates for RFID tags. The application programming interface (API) implemented by this library consists is presented in Table 4.1. This API is quite minimal, but it suffices for the applications presented in Ch. 6.

## 4.3 Evaluation Environment

I began this project as an attempt to build off of TurboTrack, working with a large instantaneous bandwidth to get a high rate of tracking. The onset of COVID-19 in the Spring of 2020 necessitated relocating my experiments to my apartment, and I brought home a smaller setup consisting of USRP N210s, which lack the instantaneous bandwidth needed for a TurboTrack-like system; serendipitously, this led me to the discovery that their was plenty of room to optimize frequency-hopping approach, as discussed above. Moreover, as a result of this shift, I evaluated the system in my apartment bedroom (see Fig. 4-1)—a typical, furnished bedroom with plenty of multipath, including an 8'×4' metal whiteboard behind the antennas.

# Chapter 5

# Results

In this chapter, I present the results of evaluating key characteristics of the system via experiment.

## 5.1 Range

The operating range of an RFID system is constrained by two things: the ability to power up the tag, and the ability to decode the tag's response. In the home environment, I conducted experiments to characterize each of these, inspired by Buettner and Wetherall's evaluation of their USRP-based reader [8].

### 5.1.1 Powering up

In the first experiment, I tested the system's ability to power up the tag. I kept the tag at a fixed position and repeatedly queried and ACKed the tag, varying the TX gain to simulate different distances via the Friis equation. Unlike moving the tag to different distances, this experiment kept the impact of multipath fixed and was more feasible to perform in the close quarters of my bedroom. Success was defined as correctly decoding the tag's 128-bit EPC packet with no errors (which requires first powering up the tag, decoding the RN16, and replying with a valid ACK), and the tag was queried 20 times at each gain setting.

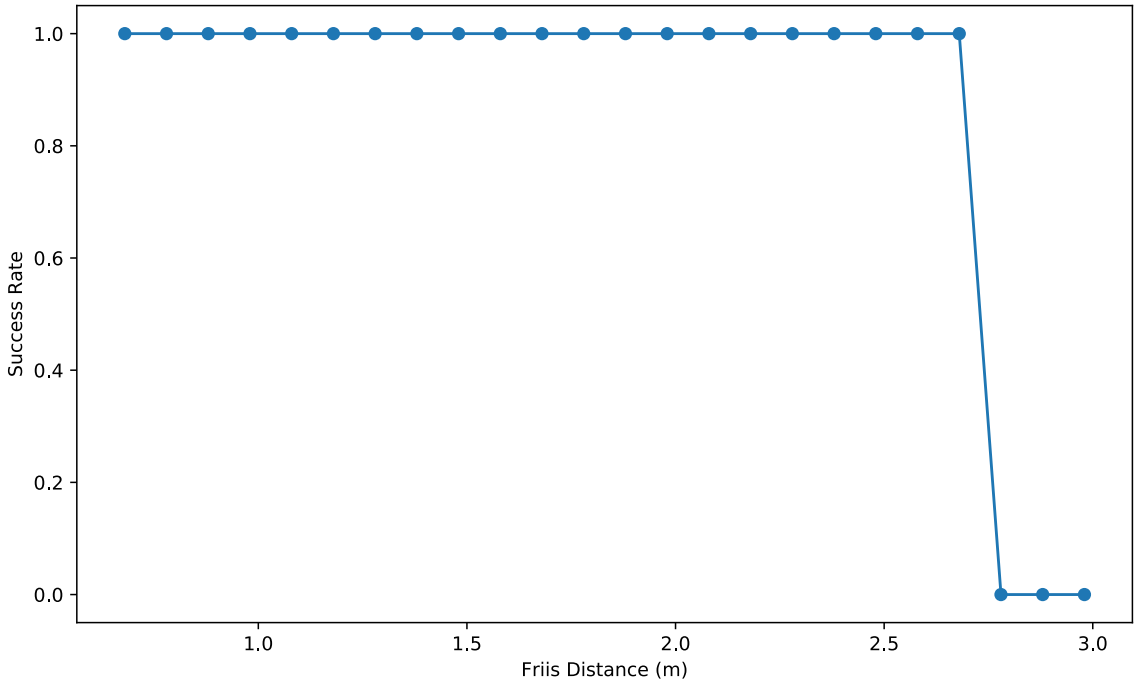


Figure 5-1: Success in identifying a tag versus distance (simulated by attenuation).

I chose gains corresponding to 10cm distance of increments, starting at the actual distance of 68cm. The results are shown in Fig. 5-1. As shown in the plot, there is a sharp cutoff where the success rate drops quicky from 100% to 0%. This mirrors the effect found by Buettner and Wetherall [8].

Note that the power-up range could be extended within an external amplifier (omitted in my setup) or antennas with higher gain; these extensions were not relevant in my physically constrained testing environment.

### 5.1.2 Decoding

In the second experiment, I tested the system's ability to correctly identify the tag under increasing levels of noise. I recorded one EPC packet with the tag in a fixed location, and then repeatedly attempted to perform the full decoding procedure (identifying packet parameters, averaging half-bit windows, and performing FM0 decoding) with increasing levels of added Gaussian noise.

For this experiment, I calculated the signal-to-noise ratio per half-bit. Note that

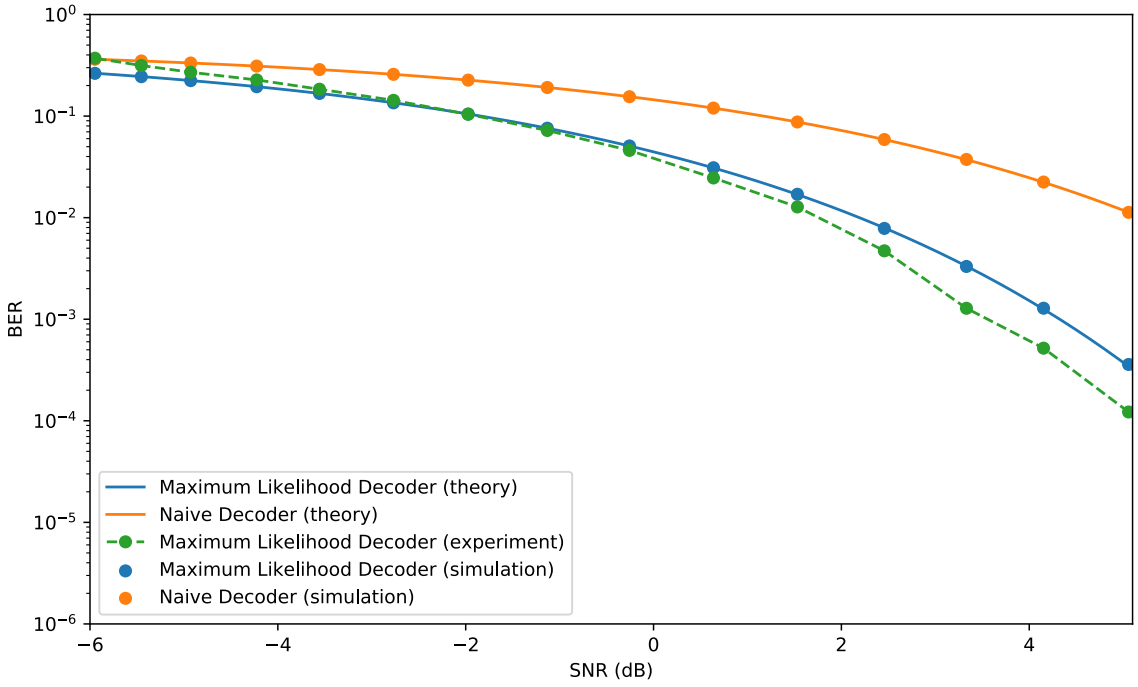


Figure 5-2: SNR vs. BER, in theory and in practice.

this yields a different SNR than calculating per-sample, as averaging multiple samples of Gaussian noise drawn from the same distribution effectively reduces the noise power.

The experimental results are plotted in Fig. 5-2 with the theoretical and simulated BER-SNR curves from section 3.2.3. Recall that these theoretical/simulated schemes are simpler than reality in that they do not have to perform any estimation of the packet's parameters before decoding—they assume we are given one sample per half-bit and that we are decoding coherently, with no need to estimate the channel. Nonetheless, the experimental BER-SNR curve matches that of the ML decoder reasonably well. It deviates by obtaining a slightly lower BER than expected at higher SNRs; this is likely due to the fact that, when the injected Gaussian noise is low, the original noise dominates, and the original noise is not perfectly Gaussian nor white. The original noise is largely phase noise from the hardware's local oscillators, and it may have more or less impact on the decoder's error rate depending on the alignment of the tag's channel and the direct TX-to-RX channel between the antennas.[1]

[1]An increase is noise that is orthogonal to the tag's channel lowers the SNR without causing

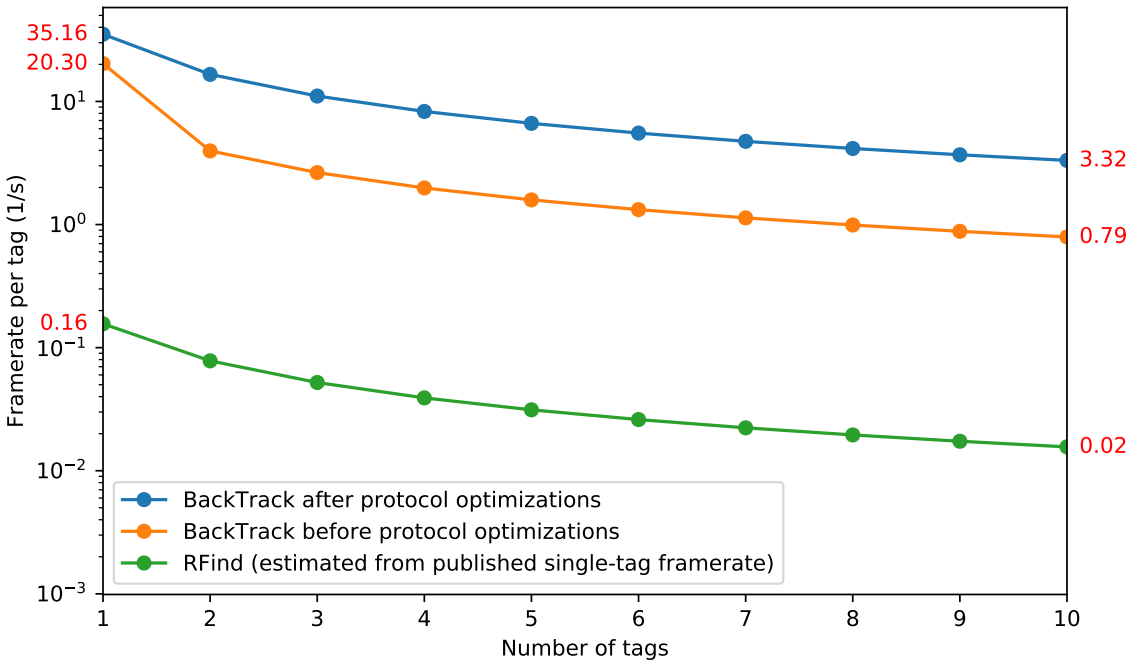


Figure 5-3: Framerate per tag vs. number of tags. Framerate is plotted on a log scale, with the linear value of notable points indicated in red.

Similarly, the real decoder underperforms theory at very low SNRs. This is likely due to the fact that we know the channel perfectly in theory, whereas in reality we must estimate it (and the other packet parameters) from noisy data.

## 5.2 Throughput

For throughput, the relevant metric is how many valid locations the system can produce per unit time. As discussed in 3.4.1, the system employs several optimizations to improve the query rate (and thus the framerate) over previous work.

Fig. 5-3 depicts the result of these optimizations. It estimates the performance of RFind in tracking multiple tags using the published single-tag latency ($\sim 6.4$ seconds per location). "BackTrack before protocol optimizations" refers to the performance of the system after optimizing the use of the radio hardware (hop rate) but before making any protocol-level optimizations, while "BackTrack after protocol optimizations"

a corresponding increase in BER, since the orthogonal noise is removed after projection onto the channel estimate.

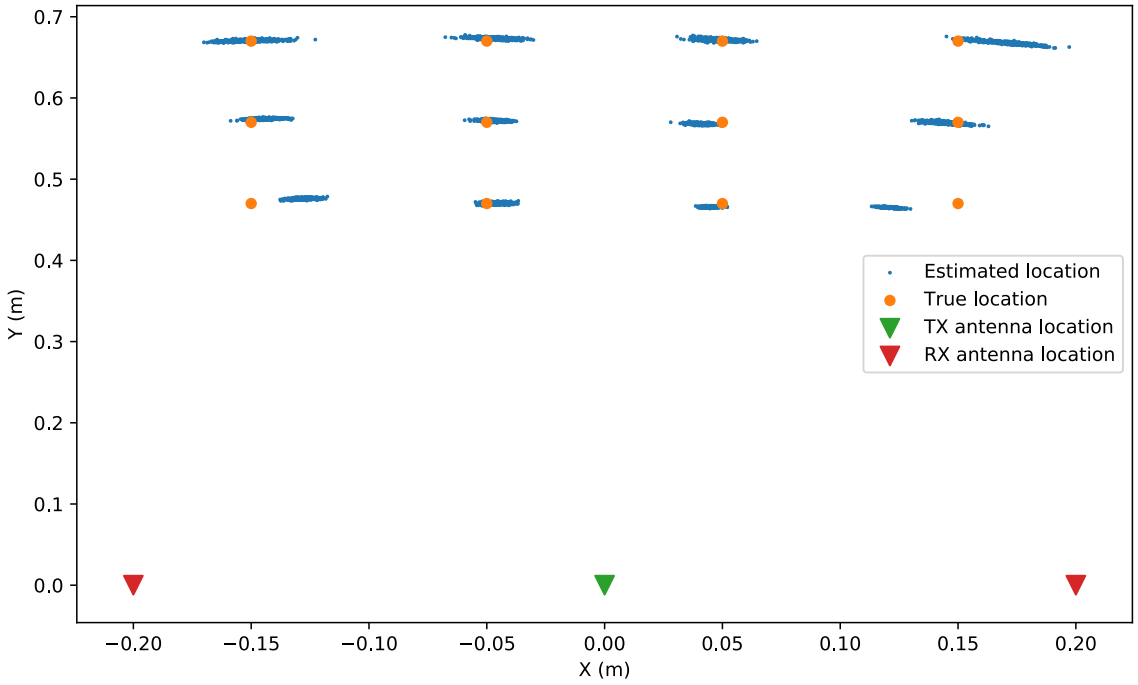


Figure 5-4: Accuracy experiment with stationary positions in 2D.

indicates the performance of the final system, after all optimizations.

This figure makes it clear that most of the improvement in framerate results from the decrease in time between hops, but protocol optimizations provide a considerable boost as well. It also highlights how quickly the per-tag framerate falls as the number of tags increases; I discuss some possible ways to mitigate the effect of this in interactive applications in Ch. 7.

## 5.3 Accuracy

In the home environment, I tested the system's accuracy by placing the tag in different static locations and recording the 2D locations output by the system. I marked 12 locations on a surface in four rows and three columns, spaced 10cm apart. I ran the system, moved the tagged object to the next marked location, and recorded the next 500 locations output by the system. The results (with points from all 12 locations) are shown in 5-4.

Overall, the mean distance error from the true location was 1.10cm, the median

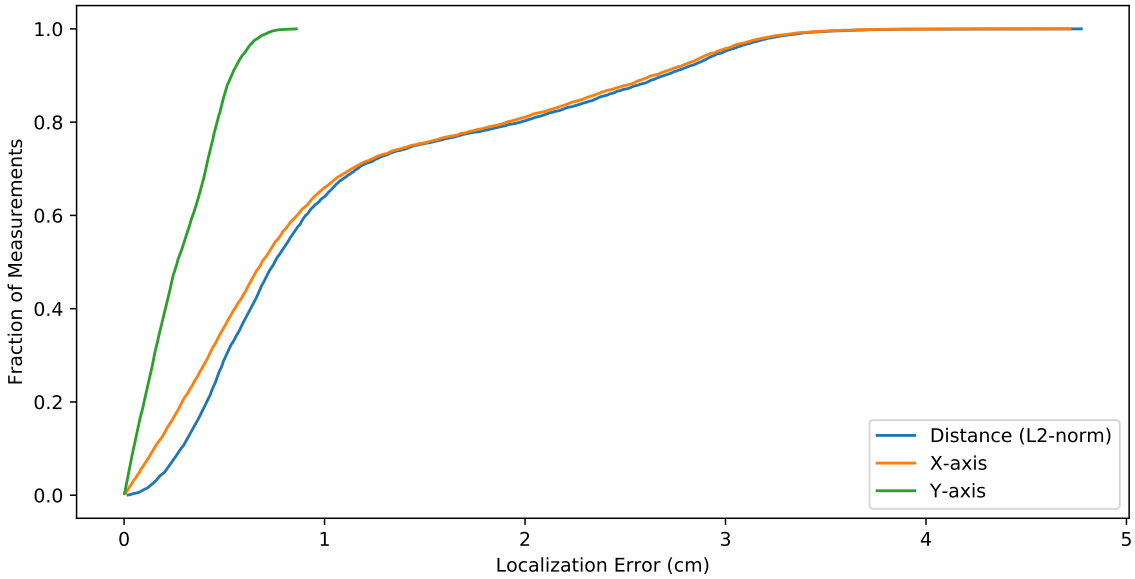


Figure 5-5: CDF of errors from the true locations, in centimeters.

was .752cm, the standard deviation was .894cm, and the maximum error was 4.77cm. As is apparent from Fig. 5-4, this error was unevenly distributed among the X and Y axes, with the X-axis error being about 16.8x the Y-axis error on average. This error data, broken down by distance and axis, is summarized in Fig 5-5.

The discrepancy of errors along the X and Y axes follows from the antenna arrangement: the RX and TX antennas in my home setup were arranged along the X axis, all with the same Y coordinate. Thus, the ellipses from each RX-TX pair share the same major axis, and a slight change in the distance estimate of one ellipse would result in a corresponding large change is the X coordinate of the intersection with the other ellipse. This also explains the increased spread as the points get farther from the antennas along the Y axis: at greater distances, the ellipses are larger and the intersection points are proportionally closer to the center of each ellipse along its major axis, where the curve is more horizontal and the derivative $\frac{dx}{dy}$ is maximized—so the same error in phase will produce a larger error in the X component of the intersection point.

# Chapter 6

# Applications

Recall that the ultimate goal of this work is to explore the feasibility of RFID localization for interactive systems. Up to this point, I have discussed the technical details of the system and how it achieves operating parameters suitable for interactive systems. In this chapter, I present two modest applications of the system which test the ultimate usability of the system and demonstrate its capabilities, built with the application library described in 4.2.[1]

## 6.1 Sequencer/Synthesizer

The first application is a combined musical sequencer/synthesizer. Some tags act as markers that can fill in slots in a musical sequencer. The X and Y coordinates of these tags are mapped to musical time (when should the sound play?) and sample selection (which sound should play?). Other tags control oscillators in a polyphonic synthesizer. The X and Y positions of these tags are mapped Theremin-style to volume and frequency, respectively.

Besides being a real-time system requiring fine-grained control, this example is intended to demonstrate the possibilities for assigning different behavior to different objects. In Fig. 6-1, for example, the cup serves as a different physical instrument than the bottle and marker: the user can "play" the cup directly, immediately chang-

[1] Videos of these applications can be found at `https://ijc8.me/research/backtrack`.

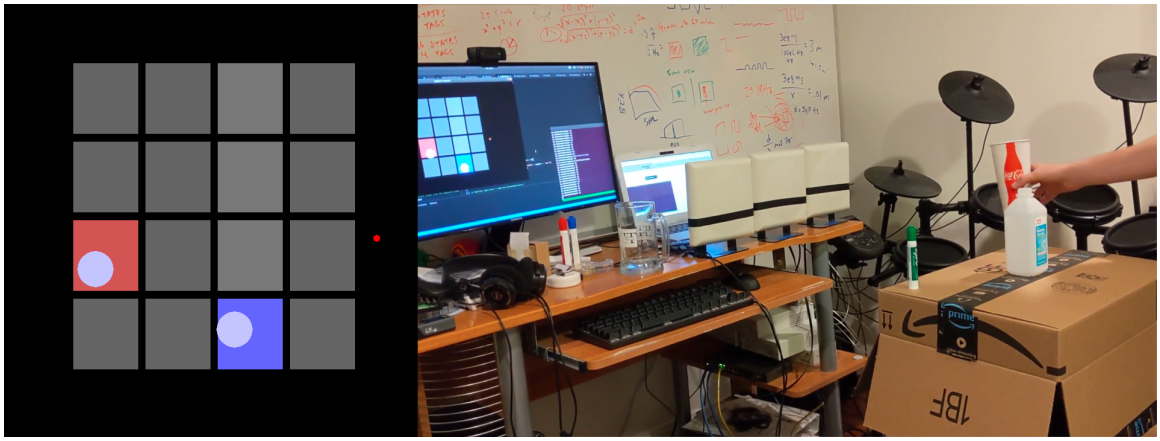

Figure 6-1: The Sequencer/Synthesizer in action. In the application (left), the large white circles represent the objects controlling the sequencer, while the small red dot represents the object controller the synthesizer. These objects, and the surrounding environment, are shown in the photo (right).

ing the note and how loud it is by moving the cup around, while the user can arrange the other objects to configure a rhythmic loop.

## 6.2 Multimapper

The second application acts as a sort of input translator. Many wonderful interactive applications already exist that are intended for use with the conventional input methods of keyboard and mouse. Inspired by MaKey MaKey [19], which maps electrical connections, this application maps RFID tag interactions to traditional keyboard and mouse events, enabling a user to control existing desktop applications by interacting with tagged objects in their environment.

The behavior of this application is specified by a configuration file, consisting of a simple list of rules. Rules specify what to do when certain tags enter or leave 2D regions of space. Configuration files are written in a standard, human-readable configuration language (YAML [7]). The user can choose which configuration to user when invoking Multimapper. Together, these allow the user to easily extend Multimapper to support different interactive applications without writing any code. As an example, the heavily-commented configuration for the drawing example is given

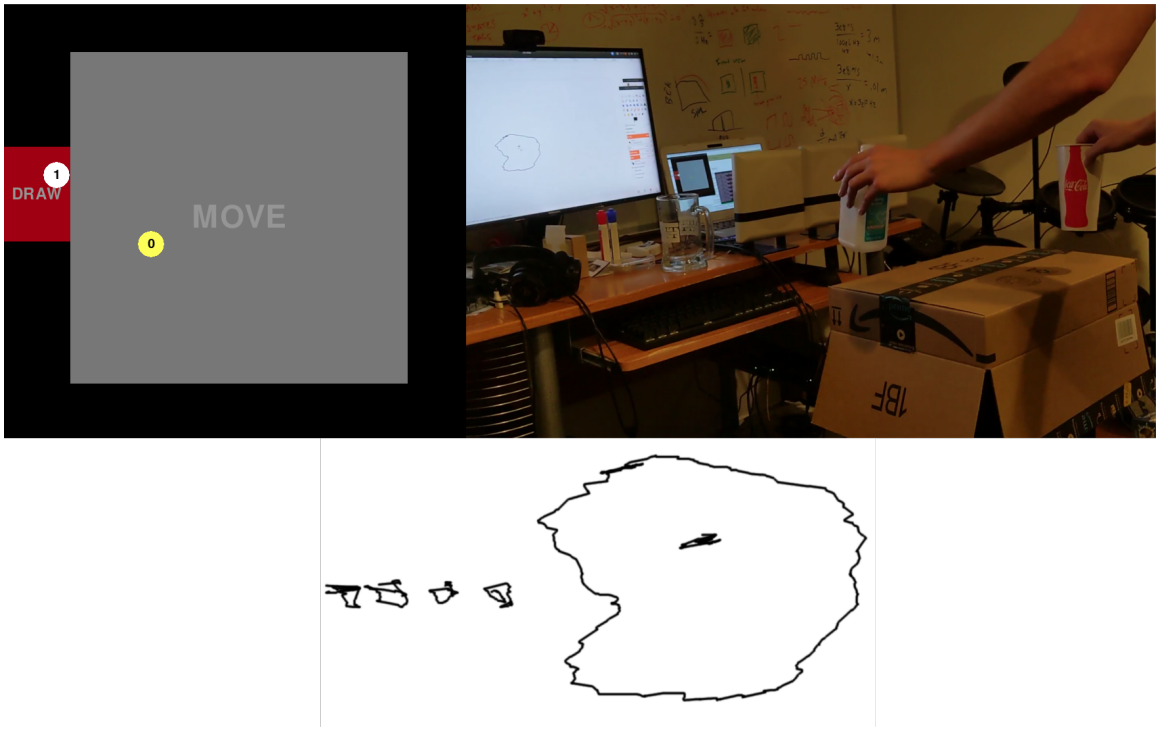


Figure 6-2: Multimapper configured for use with a drawing application. Screenshot of multimapper (top left), photo of the environment (top right), finished drawing (bottom).

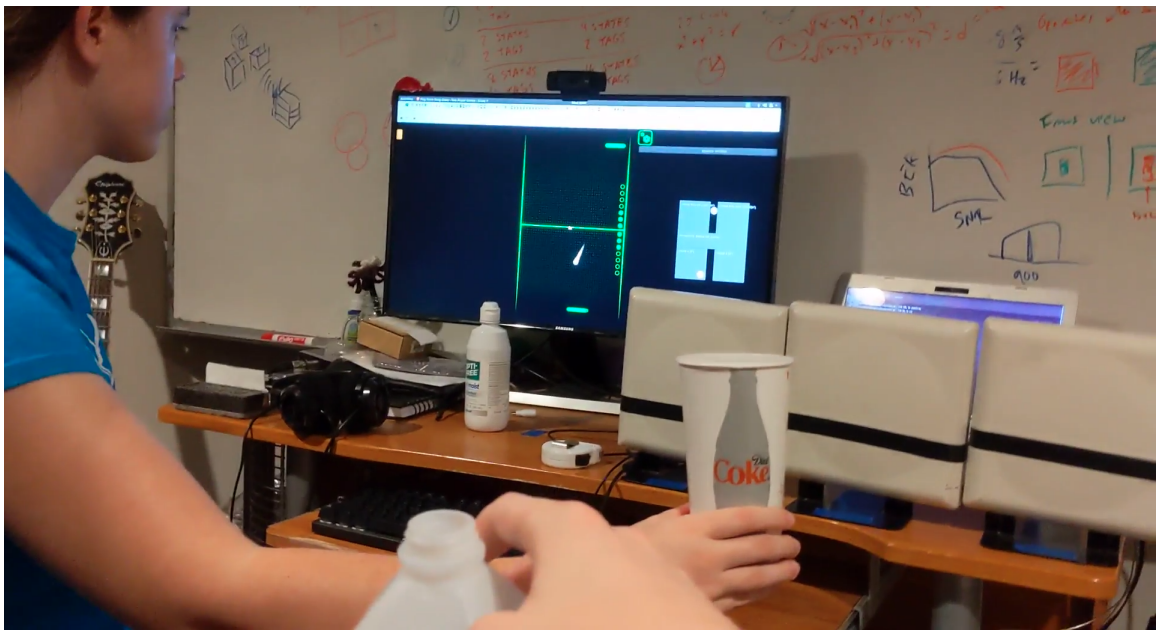

Figure 6-3: Multimapper configured for use with a two-player Pong clone. In the unmapped game, both paddles are controlled by keyboard inputs.

in listing 6.1.

```yaml
# Required: set of tags to track:
tags: [c0dec0dec0dec0dec0dec0de, 00000000000000000000000000]
smoothing: 10  # Optional: length of location history for smoothing estimates.
rules:  # Rule section.
  # The first two rules define how the tag 0000... controls when we're drawing,
  # while the last rule defines how the tag c0de... controls where we're drawing.
  - event: exit  # Required: what event triggers this rule? (0000...)
    # This rule fires when a matching tag exits the specified bounds.
    tag: 0*  # Optional: which tags should activate this rule?
    # Rectangular 2D region specified by opposite corners, in meters:
    bounds: [[-0.5, 0.3], [-0.3, 0.5]]
    action: mouseup  # Required: what happens when this rule triggers?
    # Argument specific to the action - here, which mouse button to release:
    target: left
  - event: enter  # This rule fires when a matching tag enters the bounds.
    tag: 0*
    bounds: [[-0.5, 0.3], [-0.3, 0.5]]  # Same region as the previous rule.
    action: mousedown
    target: left  # Which mouse button to press.
    # Optional appearance settings:
    color: "#990000"
    active_color: "#009900"
    label: DRAW
    label_size: 24
  - event: inside
    # This rule fires constantly when a matching tag is inside the bounds.
    tag: c*  # Match the other tag (c0de...).
    bounds: [[-0.3, 0.1], [0.7, 0.8]]
    action: mousemove
    # What region (in pixels) on the screen should we map the tag location to?
    target: [[0, 0], [1920, 1080]]
    label: MOVE
    label_size: 48
    color: [80, 80, 80]  # Alternate way of specifying color.
screen:
  width: 720  # Window dimensions (pixels).
  height: 720
  bounds: [[-0.5, 0], [0.9, 1]]  # Physical region to show in the window (meters).
  tag_radius: 20  # Size of marker drawn for each tag (pixels).
```

Listing 6.1: Multimapper Configuration for Drawing

# Chapter 7

# Conclusion

In this thesis, I explored the potential of RFID localization as an input modality for interactive systems. In this final chapter, I reflect on some of the challenges and opportunities for this technology going forward.

One obstacle between this system and real interactive deployments is cost. The system is built on software-defined radio equipment that costs several thousand dollars, which may make it infeasible for many use cases.[1] Luckily, as the system does not require a high sampling rate and new SDRs have arrived in the last few years, it is likely possible to move the system to a considerably cheaper platform without too much additional development time. I identified the bladeRF 2.0 as a promising candidate and performed initial experiments in powering up an RFID tag while transmitting an out-of-band signal, but more work is required to complete a bladeRF-based implementation and evaluate its performance.

One practical observation from testing the system is that there is more to be done for multitag usability. Though tags can be addressed independently, having multiple tags too close together tends to degrade accuracy; this may due to mutual coupling of the tag antennas, or due to the system's inability to separate out multipath with sufficient precision. This issue may not be significant in the supply chain/inventory management applications that RFID tags were intended for, but it is problematic

[1] Consider the impact of the Kinect, originally intended as a gaming peripheral, on HCI; not because body-tracking technology did not exist already, but because the Kinect offered it in a smaller, cheaper, easier-to-use package.

for small-scale interactive applications, in which a single user may be manipulating many objects within their reach. More investigation and modeling of close-proximity scenarios may provide insight into mitigating the impact on accuracy.

Looking ahead, the system is limited in another respect when tracking many objects. Because tags are queried one at a time in round-robin fashion, the framerate per tag quickly drops as the number of tags increases (see Fig. 5-3). The system's query rate could be used more effectively, by, for example, querying tags that are moving faster more often, to reduce the overall error. Another possibility is querying multiple tags (using a selector that applies to both) at once and detangling their collided responses, which could improve multitag framerates by a constant factor.

Finally, there is plenty of room to explore the applications of this system from an HCI perspective. One idea is to build out a platform for rapidly prototyping physical interfaces. Familiar interfaces, such as buttons, levers, dials, sliders, computer mice, touchscreens, and so on can all be seen as specialized position-tracking systems. A general-purpose position-tracking system could be used to implement many different conventional control schemes, and I believe RFID localization is well-suited to this purpose due to its accuracy, lack of line-of-sight constraints, and intrinsic ability to distinguish objects. The interface designer could assign a positioning modality to each tagged object: this rubber duck acts like a button, this book acts like a switch, this yo-yo acts like a slider, and so on. Such a system would convert small, wireless stickers into any kind of positional control, allowing the designer to instantly switch between them and quickly experiment with tangible alternatives.

# Bibliography


[1] Epc uhf gen2 air interface protocol. https://www.gs1.org/standards/epc-rfid/uhf-air-interface-protocol.

[2] scipy.optimize.least_squares. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html.

[3] Theremin. In Encyclopædia Britannica. Encyclopædia Britannica, inc., 2019.

[4] Radar. In Encyclopædia Britannica. Encyclopædia Britannica, inc., 2020.

[5] United States Food & Drug Administration. Radio frequency identification (rfid). https://www.fda.gov/radiation-emitting-products/electromagnetic-compatibility-emc/radio-frequency-identification-rfid.

[6] Richard Bell. Maximum supported hopping rate measurements using the universal software radio peripheral software defined radio. Proceedings of the GNU Radio Conference, 1(1), 2016.

[7] Oren Ben-Kiki, Clark Evans, and Ingy döt Net. Yaml ain't markup language. https://yaml.org.

[8] M. Buettner and D. Wetherall. A software radio-based uhf rfid reader for phy/mac experimentation. In 2011 IEEE International Conference on RFID, pages 134–141, 2011.

[9] Haitham Hassanieh, Fadel Adib, Dina Katabi, and Piotr Indyk. Faster gps via the sparse fourier transform. In Proceedings of the 18th Annual International Conference on Mobile Computing and Networking, Mobicom '12, page 353–364, New York, NY, USA, 2012. Association for Computing Machinery.

[10] Valentin Heun, James Hobin, and Pattie Maes. Reality editor: Programming smarter objects. In Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication, UbiComp '13 Adjunct, page 307–310, New York, NY, USA, 2013. Association for Computing Machinery.

[11] Haojian Jin, Jingxian Wang, Zhijian Yang, Swarun Kumar, and Jason Hong. Wish: Towards a wireless shape-aware world using passive rfids. In Proceedings of the 16th Annual International Conference on Mobile Systems, Applications,

and Services, MobiSys '18, page 428–441, New York, NY, USA, 2018. Association for Computing Machinery.

[12] Sergi Jordà, Günter Geiger, Marcos Alonso, and Martin Kaltenbrunner. The reactable: exploring the synergy between live music performance and tabletop tangible interfaces. In Tangible and Embedded Interaction, 2007.

[13] Hanchuan Li, Eric Brockmeyer, Elizabeth J. Carter, Josh Fromm, Scott E. Hudson, Shwetak N. Patel, and Alanson Sample. Paperid: A technique for drawing functional battery-free wireless interfaces on paper. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI '16, pages 5885–5896, New York, NY, USA, 2016. ACM.

[14] Hanchuan Li, Eric Brockmeyer, Elizabeth J. Carter, Josh Fromm, Scott E. Hudson, Shwetak N. Patel, and Alanson Sample. Paperid: A technique for drawing functional battery-free wireless interfaces on paper. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, CHI '16, pages 5885–5896, New York, NY, USA, 2016. ACM.

[15] Zhihong Luo, Qiping Zhang, Yunfei Ma, Manish Singh, and Fadel Adib. 3d backscatter localization for fine-grained robotics. In NSDI, 2019.

[16] Yunfei Ma, Nicholas Selby, and Fadel Adib. Minding the billions: Ultra-wideband localization for deployed rfid tags. In MobiCom '17, 2017.

[17] Ettus Research. Sbx. `https://www.ettus.com/`.

[18] Ettus Research. Usrp n210. `https://www.ettus.com/`.

[19] Jay Silver and Eric Rosenbaum. Makey makey. `https://makeymakey.com/`.

[20] Joshua R. Smith, Alanson P. Sample, Pauline S. Powledge, Sumit Roy, and Alexander Mamishev. A wirelessly-powered platform for sensing and computation. In Proceedings of the 8th International Conference on Ubiquitous Computing, UbiComp'06, page 495–506, Berlin, Heidelberg, 2006. Springer-Verlag.

[21] David Tse and Pramod Viswanath. Fundamentals of Wireless Communication. Cambridge University Press, USA, 2005.

[22] Deepak Vasisht, Jue Wang, and Dina Katabi. Rf-idraw: virtual touch screen in the air using rf signals. In S3@MobiCom, 2014.