

Bridging the Gap Between People, Mobile Devices, and the Physical World

Chang Xiao

Submitted in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy  
under the Executive Committee  
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2021

© 2021

Chang Xiao

All Rights Reserved

## Abstract

Bridging the Gap Between People, Mobile Devices and the Physical World

Chang Xiao

Human-computer interaction (HCI) is being revolutionized by computational design and artificial intelligence. As the diversity of user interfaces shifts from personal desktops to mobile and wearable devices, yesterday's tools and interfaces are insufficient to meet the demands of tomorrow's devices. This dissertation describes my research on leveraging different physical channels (e.g., vibration, light, capacitance) to enable novel interaction opportunities.

We first introduce *FontCode*, an information embedding technique for text documents. Given a text document with specific fonts, our method can embed user-specified information (e.g., URLs, meta data, etc) in the text by perturbing the glyphs of text characters while preserving the text content. The embedded information can later be retrieved using a smartphone in real time.

Then, we present *Vidgets*, a family of mechanical widgets, specifically push buttons and rotary knobs that augment mobile devices with tangible user interfaces. When these widgets are attached to a mobile device and a user interacts with them, the nonlinear mechanical response of the widgets shifts the device slightly and quickly. Subsequently, this subtle motion can be detected by the Inertial Measurement Units (IMUs), which is commonly installed on mobile devices.

Next, we propose *BackTrack*, a trackpad placed on the back of a smartphone to track fine-grained finger motions. Our system has a small form factor, with all the circuits encapsulated in a thin layer attached to a phone case. It can be used with any off-the-shelf smartphone, requiring no power supply or modification of the operating systems. BackTrack simply extends the finger

tracking area of the front screen, without interrupting the use of the front screen.

Lastly, we demonstrate *MoiréBoard*, a new camera tracking method that leverages a seemingly irrelevant visual phenomenon, the moiré effect. Based on a systematic analysis of the moiré effect under camera projection, MoiréBoard requires no power nor camera calibration. It can easily be made at a low cost (e.g., through 3D printing) and ready to use with any stock mobile device with a camera. Its tracking algorithm is computationally efficient and can run at a high frame rate. It is not only simple to implement, but also tracks devices at a high accuracy, comparable to the state-of-the-art commercial VR tracking systems.

## Table of Contents

Acknowledgments . . . . .	1
Section 1 Overview . . . . .	1
<b>Chapter 1 FontCode: Mobile Interaction through Text Document</b>	<b>5</b>
Section 2 Introduction and Related Work . . . . .	6
2.1 Information Embedding . . . . .	8
Section 3 Method Overview . . . . .	11
3.1 Message Embedding . . . . .	11
3.2 Message Retrieval . . . . .	13
Section 4 Glyph Recognition . . . . .	14
4.1 Pixel Image Preprocessing . . . . .	14
4.2 Network Structure . . . . .	15
4.3 Network Training . . . . .	16
4.4 Constructing The Glyph Codebook . . . . .	17
4.5 Embedding and Retrieving Integers . . . . .	19
Section 5 Error Correction Coding . . . . .	21
5.1 Challenges of the Coding Problem . . . . .	21

5.2	Chinese Remainder Codes . . . . .	22
5.2.1	Hamming Distance Decoding . . . . .	23
5.2.2	Chinese Remainder Codes . . . . .	24
5.3	Improved Error Correction Capability . . . . .	28
Section 6	Perceptual Evalutation . . . . .	30
Section 7	Applications . . . . .	33
7.1	Application I: Format-Independent Metadata . . . . .	33
7.2	Application II: Imperceptible Optical Codes . . . . .	34
7.3	Application III: Encrypted Message Embedding . . . . .	34
7.4	Application IV: Text Document Signature . . . . .	35
Section 8	Results and Validation . . . . .	38
8.1	Main Results . . . . .	38
8.2	Validation . . . . .	39
8.3	Perceptual Evaluation . . . . .	43
Section 9	Discussion and Future Work . . . . .	46
<b>Chapter 2</b>	<b>Vidgets: Modular Mechanical Widgets for Mobile Devices</b>	<b>48</b>
Section 10	Introduction and Related Work . . . . .	49
10.1	Around-device Interaction . . . . .	52
Section 11	Theory of Operation . . . . .	55
11.1	Cause of Subtle Motion . . . . .	55

11.2 Resistance Force of a Tactile Button . . . . .	55
11.3 Generation of Accelerometer Signal . . . . .	56
11.4 Physics-Based Force Model . . . . .	58
11.5 Widget Design . . . . .	60
Section 12 Signal Processing at Runtime . . . . .	63
12.1 Segmentation . . . . .	64
12.2 Event Detection . . . . .	65
Section 13 Applications . . . . .	67
Section 14 Evaluation . . . . .	71
14.1 Accuracy . . . . .	71
14.2 Response Time . . . . .	73
14.3 Robustness . . . . .	74
14.4 Discussion about Layout and Robustness . . . . .	76
Section 15 Discussion and Future Work . . . . .	79
<b>Chapter 3 BackTrack: 2D Back-of-device Interaction Through Front Touchscreen</b>	<b>81</b>
Section 16 Introduction and Related Work . . . . .	82
16.1 BackTrack Overview . . . . .	84
16.2 Back-of-device Interaction . . . . .	86
Section 17 A Capacitive Circuit Model for BackTrack . . . . .	90
17.1 BackTrack Circuit Model and Analysis . . . . .	91

Section 18 BackTrack Design . . . . .	94
18.1 Electrode Design . . . . .	94
18.2 Electrode Connection Pattern . . . . .	96
18.3 Switch Design . . . . .	99
18.4 Prototype Details . . . . .	100
Section 19 Applications . . . . .	102
19.1 Avoiding Occlusion using BoD Interaction . . . . .	102
19.2 BoD Authentication . . . . .	104
19.3 Joint Interaction with Touches on the Front . . . . .	104
Section 20 User Evaluations . . . . .	106
20.1 Finger Motion Performance . . . . .	106
20.2 Usability . . . . .	108
Section 21 Discussion . . . . .	110
<b>Chapter 4 MoiréBoard: Low-cost and Accurate Camera Tracking for Mobile VR</b>	<b>111</b>
Section 22 Introduction and Related Work . . . . .	112
22.1 Camera Tracking . . . . .	114
Section 23 Camera Tracking using Moiré Effect . . . . .	117
23.1 The Geometry of Moiré Fringes . . . . .	117
23.2 Moiré Pattern under Camera Projection . . . . .	118
23.3 Calibration-free Camera Tracking . . . . .	121

23.4 MoiréBoard Design . . . . .	125
23.5 Accuracy Analysis . . . . .	126
Section 24 Evaluation . . . . .	128
24.1 Synthetic Scene Evaluation . . . . .	128
24.2 Real Scene Evaluation . . . . .	131
Section 25 Discussion and Future Work . . . . .	135
<b>Chapter 5 Epilogue</b>	<b>137</b>
Epilogue . . . . .	139
References . . . . .	153

## **Acknowledgements**

Throughout my journey as a Ph.D. student, I have received countless support and assistance. I would like to thank everyone who helped me through this adventure.

I would like to thank my advisor Prof. Changxi Zheng, whose expertise was invaluable in formulating the research questions and methodology. You always encourage me to explore a wide range of topics. Your high standard also helped me develop a taste of conducting high-quality research. I would also like to thank Prof. Shree K. Nayar, for his guidance on how to be an excellent researcher and all the opportunities I received to further my research.

I would like to thank my committee members for their great suggestions on improving this dissertation. Without their guidance, I would not have made it. Thank you, Prof. Steven Feiner, Prof. Brian Smith, Prof. Carl Vondrick and Prof. Andrés Monroy-Hernández.

I would like to further acknowledge all my collaborators, colleagues, and friends from the university and industry, not only for their support to my research, but also for the time they spent with me to fulfill my wonderful life as a Ph.D. student. These individuals include, but are not limited to, Karl Bayer, Pengyu Chen, Gabriel Cirio, Peter Chen, Raymond Fei, Dingzeyu Li, Henrique Maia, Yuchen Mo, Zahra Montazeri, Da Tang, Jian Wang, Rundi Wu, Ruilin Xu, Cheng Zhang, Shuang Zhao, Pi Zhen and Peiling Zhong.

I would also like to thank all staffs of Computer Science Department for taking care of the logistics. Thank you, Anne, Cindy, Elias, Ivy, Jessica, Rob, Twinkle.

I would like to give a special thanks to all my basketball teammates, it is always fun and stress-relieving to play basketball with you after work.

Finally, I would like to thank my father Rixin Xiao and mother Yonghong Chen, for their wise counsel and sympathetic ears. You are always there for me. Also, I could not have completed my dissertation without the support from Zixiaofan Yang. You have been extremely supportive throughout the entire process and made countless efforts to help me get to this point.

## Section 1: Overview

It has been over thirty years since the invention of the first commercially available mobile device (The Motorola DynaTAC 8000X). Since then, mobile devices have been regarded to serve as a tool for better connecting individual people. With the advancement of novel sensing and interaction techniques, today’s mobile devices have become much more powerful than thirty years ago. Besides mobile phones, the set of today’s mobile devices also have been extended to a wide range of smart devices, including smartphones, smartwatches, and smartglasses. Those modern mobile devices are not only designed for communication, but also open the door for users to better sense and interact with the physical world. For example, using mobile Augmented Reality (AR) systems, we can easily scan the scene structure and render virtual objects like furniture on top of the real scene. This appears to be very helpful for indoor room design.

To better allow users to sense the physical world, modern mobile devices are equipped with various types of sensors, such as depth cameras, IMUs, and microphone arrays. These sensors provide abundant information about the environment, thereby enabling users to have a rich set of interaction opportunities regarding the physical space. In addition, many sensor fusion algorithms have been proposed [36, 162, 4, 69] to further enhance the interactivity of mobile devices. However, many previous efforts to advance mobile devices not paying much attention to the physical world. It would be interesting to engineer objects in the physical space which will allow the mobile devices to better sense and interact with the surrounding environment. This motivates my research on engineering the physical world to enhance the interaction of mobile devices.

On the other hand, with ever-changing mobile devices, users are constantly adapting and learning how to use new technologies. However, a good device should also consider and adapt to the user’s behavior. Thus, it is essential to engineer mobile devices with a high awareness of accessibility and customizability. To this end, I proposed new back-of-device interaction methods, which

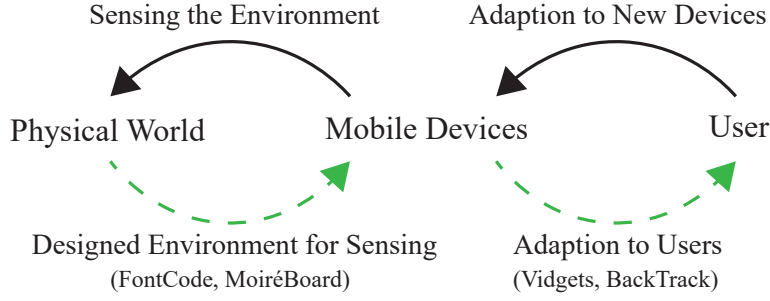


Figure 1.1: **The cycle of users, mobile devices and the physical world.** Black arrows indicate traditional flow with respect to the advancement of mobile devices. Green dashed arrows indicate my research direction.

enable new interaction methods on mobile devices in a more convenient and configurable way.

Fig. 1.1 shows the connection among users, mobile devices and the physical world.

This dissertation is structured as follows:

**Chapter 1** introduces a new information embedding technique for text documents called *FontCode*. Given a text document with specific fonts, our method embeds user-specified information in the text by perturbing the glyphs of text characters while preserving the text content. We devise an algorithm to choose unobtrusive yet machine-recognizable glyph perturbations, leveraging a recently developed generative model that alters the glyphs of each character continuously on a font manifold. We then introduce an algorithm that embeds a user-provided message in the text document and produces an encoded document whose appearance is minimally perturbed from the original document. We also present a glyph recognition method that recovers the embedded information from an encoded document stored as a vector graphic or pixel image, or even on a printed paper. In addition, we introduce a new error-correction coding scheme that rectifies a certain number of recognition errors. Lastly, we demonstrate that our technique enables a wide array of applications, especially as a text document metadata holder, an unobtrusive optical barcode, a cryptographic message embedding scheme, and a text document signature.

**Chapter 2** presents a family of mechanical widgets, *Vidgets*, specifically push buttons and rotary knobs that augment mobile devices with tangible user interfaces. When these widgets are attached to a mobile device and a user interacts with them, the nonlinear mechanical response

of widgets shifts the device slightly and quickly, and this subtle motion can be detected by the accelerometer commonly installed on mobile devices. We propose a physics-based model to understand the nonlinear mechanical response of widgets. This understanding enables us to design tactile force profiles of these widgets, thereby making the resulting accelerometer signals easy to recognize. We then develop a lightweight signal processing algorithm that analyzes the accelerometer signals and recognizes how the user interacts with the widgets in real time. Widgets are low-cost, compact, reconfigurable, and power efficient. They can form a diverse set of physical interfaces that enrich users' interactions with mobile devices in various practical scenarios.

**Chapter 3** introduces a trackpad placed on the back of a smartphone to track fine-grained finger motions called *BackTrack*. BackTrack has a small form factor, with all the circuits encapsulated in a thin layer attached to a phone case. It can be used with any off-the-shelf smartphone, requiring no power supply or modification of the operating systems. BackTrack simply extends the finger tracking area of the front screen, without interrupting the use of the front screen. It also provides a switch to prevent unintentional touch on the trackpad. All these features are enabled by a battery-free capacitive circuit, part of which is a transparent, thin-film conductor coated on a thin glass and attached to the front screen. To ensure accurate and robust tracking, the capacitive circuits are carefully designed. BackTrack is based on a circuit model of capacitive touchscreens, justified through both physics-based finite-element simulation and controlled laboratory experiments. We conduct user studies to evaluate the performance of BackTrack. We also demonstrate its use in a number of smartphone applications.

**Chapter 4** demonstrates a novel camera tracking system, *MoiréBoard*. Device tracking is an essential building block in a myriad of HCI applications. For example, commercial VR devices are equipped with dedicated hardware such as laser-emitting beacon stations, to enable accurate tracking of VR headsets. However, this hardware remains costly. On the other hand, low-cost solutions such as IMU sensors and visual markers are available, but they suffer from large tracking errors. In this work, we bring high accuracy and low cost together to present MoiréBoard, a new camera tracking method that leverages a seemingly irrelevant visual phenomenon, the moiré effect.

Based on a systematic analysis of the moiré effect under camera projection, MoiréBoard requires no extra power nor camera calibration. It can easily be made at a low cost (e.g., through 3D printing) and ready to use with any stock mobile device with a camera. Its tracking algorithm is computationally efficient and can run at a high frame rate. It is not only simple to implement, but also tracks devices at a high accuracy, comparable to the state-of-the-art commercial VR tracking systems.

# **Chapter 1**

## **FontCode: Mobile Interaction through Text Document**

## Section 2: Introduction and Related Work

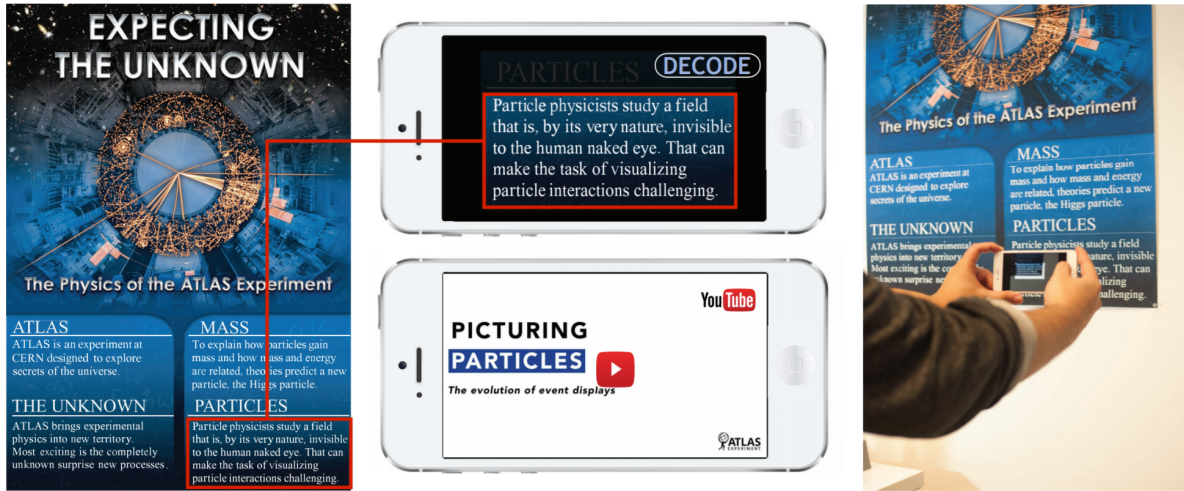


Figure 2.1: **Augmented poster.** Among many applications enabled by our FontCode method, here we create a poster embedded with an unobtrusive optical barcode (left). The poster uses text fonts that look almost identical to the standard Times New Roman, and has no traditional black-and-white barcode pattern. But our smartphone application allows the user to take a snapshot (right) and decode the hidden message, in this case, a Youtube link (middle).

Information embedding, the technique of embedding a message into host data, has numerous applications: Digital photographs have metadata embedded to record such information as capture date, exposure time, focal length, and camera’s GPS location. Watermarks embedded in images, videos, and audios have been one of the most important means in digital production to claim copyright against piracies [16]. And indeed, the idea of embedding information in light signals has grown into an emerging field of visual light communication (e.g., see [70]).

In all these areas, information embedding techniques meet two desiderata: (i) the host medium is minimally perturbed, implying that the embedded message must be minimally intrusive; and (ii) the embedded message can be robustly recovered by the intended decoder even in the presence of some decoding errors.

Remaining reclusive is the information embedding technique for text documents, in both digital

and physical form. While explored by many previous works on digital text steganography, information embedding for text documents is considered more challenging to meet the aforementioned desiderata than its counterparts for images, videos, and audios [3]. This is because the “pixel” of a text document is individual letters, which, unlike an image pixel, cannot be changed into other letters without causing noticeable differences. Consequently, existing techniques have limited information capacity or work only for specific digital file formats (such as PDF or Microsoft Word).

We propose *FontCode*, a new information embedding technique for text documents. Instead of changing text letters into different ones, we alter the *glyphs* (i.e., the particular shape designs) of their fonts to encode information, leveraging the recently developed concept of font manifold [24] in computer graphics. Thereby, the readability of the original document is fully retained. We carefully choose the glyph perturbation such that it has a minimal effect on the typeface appearance of the text document, while ensuring that glyph perturbation can be recognized through Convolutional Neural Networks (CNNs). To recover the embedded information, we develop a decoding algorithm that recovers the information from an input encoded document—whether it is represented as a vector graphics file (such as a PDF) or a rasterized pixel image (such as a photograph).

Exploiting the features specific to our message embedding and retrieval problems, we further devise an error-correction coding scheme that is able to fully recover embedded information up to a certain number of recognition errors, making a smartphone into a robust FontCode reader (see Fig. 22.1).

**Applications** As a result, FontCode is not only an information embedding technique for text documents but also an unobtrusive tagging mechanism, finding a wide array of applications. We demonstrate four of them. (i) It serves as a metadata holder in a text document, which can be freely converted to different file formats or printed on paper without loss of the metadata—across various digital and physical forms, the metadata is always preserved. (ii) It enables to embed in a text unobtrusive optical codes, ones that can replace optical barcodes (such as QR codes) in artistic designs such as posters and flyers to minimize visual distraction caused by the barcodes.

(iii) By construction, it offers a basic cryptographic scheme that not only embeds but also encrypts messages, without resorting to any additional cryptosystem. And (iv) it offers a new text signature mechanism, one that allows to verify document authentication and integrity, regardless of its digital format or physical form.

**Technical contributions** We propose an algorithm to construct a glyph codebook, a lookup table that maps a message into perturbed glyphs and ensures the perturbation is unobtrusive to our eyes. We then devise a recognition method that recovers the embedded message from an encoded document. Both the codebook construction and glyph recognition leverage CNNs. Additionally, we propose an error-correction coding scheme that rectifies recognition errors due to factors such as camera distortion. Built on a 1700-year old number theorem, the Chinese Remainder Theorem, and a probabilistic decoding model, our coding scheme is able to correct more errors than what block codes based on Hamming distance can correct, outperforming their theoretical error-correction upper bound.

## 2.1 Information Embedding

In this section, we briefly review the previous work related to FontCode.

**Font manipulation** Our FontCode system perturbs glyphs using the generative model by Campbell and Kautz [24]. There are also some other methods that create fonts and glyphs either with computer-aided tools [127] or automatic generation.

Metafont [81] creates parametric fonts and was used to create most of the Computer Modern typeface family. Later, Shamir and Rappoport [132] proposed a system that generate fonts using high-level parametric features and constraints to adjust glyphs. This idea was extended to parameterize glyph shape components [66]. Other approaches generate fonts by deriving from examples and templates [87, 140], similarity [101] or crowdsourced attributes [113]. Recently, Phan et al. [117] utilize a machine learning method trained through a small set of glyphs to synthesize typefaces that have a consistent style.

**Font recognition** Automatic font recognition from a photo or image has been widely studied [9, 71, 120]. These methods identify fonts by extracting statistical and/or typographical features of the document. Recently in [29], the authors proposed a scalable solution leveraging supervised learning. Then, Wang et al. [149] improved font recognition using Convolutional Neural Networks. Their algorithm can run without resorting to character segmentation and optical character recognition methods. In our work, we use existing algorithms to recognize text fonts of the input document, and further devise an algorithm to recognize glyph perturbation for recovering the embedded information. Unlike the existing font recognition methods that identify fonts from a text of many letters, our algorithm aims to identify glyph perturbation for individual letters.

**Text steganography** Our work is related to digital steganography (such as digital watermarks for copyright protection), which has been studied for decades, mostly focusing on videos, images, and audios—for example, we refer to [28] for a comprehensive overview of digital imaging steganography. However, digital text steganography is much more challenging [3], and is thus much less developed. We categorize existing methods based on their features (see Table 2.1):

Methods based on cover text generation (CTG) hide a secret message by generating an *ad-hoc* cover text which looks lexically and syntactically convincing [151, 150]. However, this type of steganography is unable to embed messages in existing text documents. Therefore, it fails to meet the attribute that we call cover text preservation (CP), and has limited applications.

The second type of methods exploits format-specific features (FSF). For example, for Microsoft Word document, Bhaya et al. [14] particularly assigned each text character a different but visually similar font available in Word to conceal messages. Others hide messages by changing the character scale and color or adding underline styles in a document [136, 116]. However, these changes are generally noticeable. More recent methods exploit special ASCII codes and Unicodes that are displayed as an empty space in a PDF viewer [27, 125]. These methods are not format independent (FI); they are bounded to a specific file format (such as Word or PDF) and text viewer. The concealed messages would be lost if the document was converted to a different format or even opened

Category	Previous work	Attribute			
		CP	FGC	FI	PP
CTG	[150, 3]		✓	✓	✓
SP	[20, 79, 5, 57]	✓		✓	✓
FSF	[116, 14, 27, 125]	✓	✓		
<b>Our work</b>		✓	✓	✓	✓

Table 2.1: **A summary** of related text steganographic methods. CP indicates cover text preservation; FGC indicates fine granularity coding; FI indicates format-independent; PP indicates printed paper.

with a different version of the same viewer. Also, the concealed message cannot be preserved when the document is printed on paper (PP) and photographed later.

More relevant to our work is the family of methods that embed messages via what we call structural perturbations (**SP**). Line-shift methods [5, 20] hide information by perturbing the space between text lines: reducing or increasing the space represents a 0 or 1 bit. Similarly, word-shift methods [79, 20] perturb the spaces between words. Others perturb the shape of specific characters, such as raising or dropping the positions of the dots of “i” and “j” [20]. These methods cannot support fine granularity coding (FGC), as they encode 1 bit in every line break, word break or special character that appears sparsely. Our method provides FGC by embedding information in individual letters, and thereby has a much larger information capacity. In addition, all previous methods demonstrate the retrieval of hidden messages from *digital document files only*. It is unclear to what extent they can decode from real photos of the text.

Table 2.1 summarizes these related work and their main attributes. Like most of these works, our method aims to perturb the appearance of the text in an unobtrusive way. Our method is advantageous, as it provides more desired attributes.

## Section 3: Method Overview

Our FontCode system embeds in a text document any type of information as a bit string. For example, an arbitrary text message can be coded into a bit string using the standard ASCII code or Unicode<sup>1</sup>. We refer to such a bit string as a *plain message*.

In a text document, the basic elements of embedding a plain message are the *letters*, appearing in a particular font. Our idea is to perturb the glyph of each letter to embed a plain message. To this end, we leverage the concept of *font manifold* proposed by Campbell and Kautz [24]. Taking as input a collection of existing font files, their method creates a low-dimensional font manifold for every character—including both alphabets and digits—such that every location on this manifold generates a particular glyph of that character. This novel generative model is precomputed once for each character. Then, it allows us to alter the glyph of each text letter in a subtle yet systematic way, and thereby embed messages.

It is worth noting that our method does not depend on specifically the method of [24], and other font generative models can also be used to generate glyph perturbations in our work. In this paper, when there is no confusion, we refer to a location  $u$  on a font manifold and its resulting glyph interchangeably.

### 3.1 Message Embedding

Our message embedding method consists of two steps, (i) precomputation of a codebook for processing all documents and (ii) runtime embedding of a plain message in a given document.

During precomputation, we construct a codebook of perturbed glyphs for typically used fonts. Consider a font such as Times New Roman. Each character in this font corresponds to a specific

---

<sup>1</sup>See the character set coding standards by the Internet Assigned Numbers Authority (<http://www.iana.org/assignments/character-sets/character-sets.xhtml>)

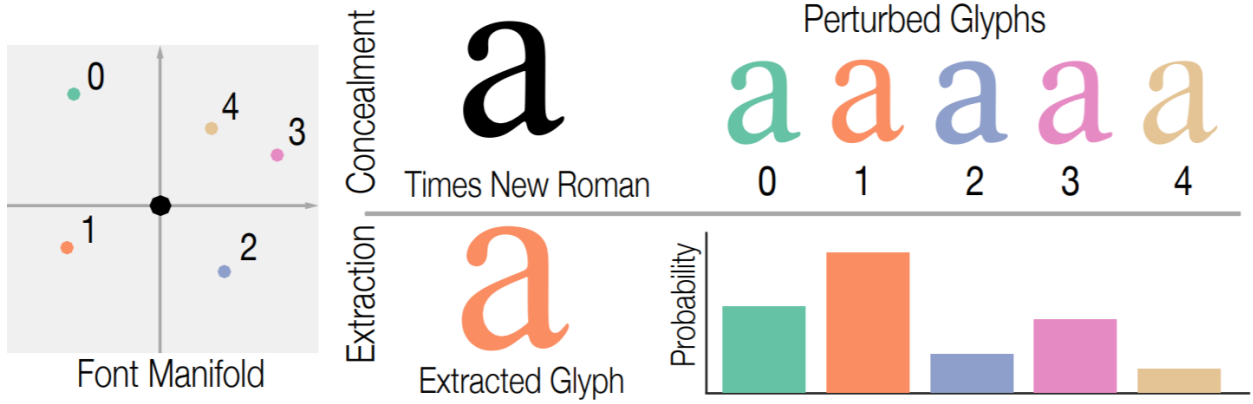


Figure 3.1: **Embedding and extraction.** Here we sample 5 points around the Times New Roman on the manifold (left), generating the perturbed glyphs to embed integers (top-right). We embed “1” in letter “a” using the second glyph (in orange) in the perturbation list. In the retrieval step, we evaluate a probability value (inverse of distance) by our CNNs (bottom-right), and extract the integer whose glyph results in the highest probability.

location,  $\bar{u}$ , on the font manifold. We identify a set of locations on the manifold as the *perturbed glyphs* of  $\bar{u}$  and denote them as  $\{u_0, u_1, \dots\}$  (see Fig. 3.1). Our goal in this step is to select the perturbed glyphs such that their differences from the glyph  $\bar{u}$  of the original font is almost unnoticeable to our naked eyes but recognizable to a computer algorithm. The sets of perturbed glyphs for all characters with typically used fonts form our codebook.

At runtime, provided a text document (or a text region or paragraphs), we perturb the glyph of the letter in the document to embed a given plain message. Consider a letter in an original glyph  $\bar{u}$  in the given document. Suppose in the precomputed codebook, this letter has  $N$  perturbed glyphs, namely  $\{u_0, u_1, \dots, u_{N-1}\}$ . We embed in the letter an integer  $i$  in the range of  $[0, N)$  by changing its glyph from  $\bar{u}$  to  $u_i$  (see Fig. 3.1-top). A key algorithmic component of this step is to determine the embedded integers for all letters such that together they encode the plain message.

In addition, we propose a new error-correcting coding scheme to encode the plain message. This coding scheme adds certain redundancy (i.e., some extra data) to the coded message, which, at decoding time, can be used to check for consistency of the coded message and recover it from errors.

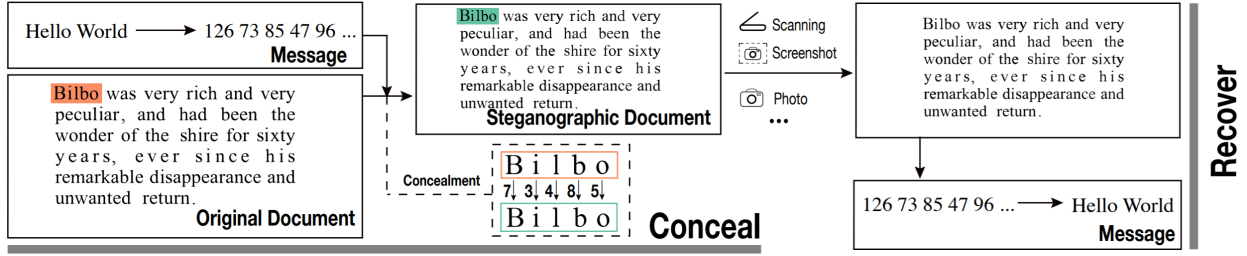


Figure 3.2: **Overview.** (left) Our embedding method takes an input message and a text document. It encodes the message into a series of integers and divides the letters into blocks. The integers are assigned to each block and embedded in individual letters. (right) To recover the message, we extract integers by recognizing the glyphs of individual letters. Then, the integers are decoded into the original plain message.

### 3.2 Message Retrieval

To retrieve information from a coded text document, the first step is to recover an integer from each letter. For each letter in the document, suppose again this letter has  $N$  perturbed glyphs in the codebook, whose manifold locations are  $\{u_0, u_1, \dots, u_{N-1}\}$ . We recognize its glyph  $u'$  in the current document as one of the  $N$  perturbed glyphs. We extract an integer  $i$  if  $u'$  is recognized as  $u_i$  (see Fig. 3.1-bottom). Our recognition algorithm works with not only vector graphics documents (such as those stored as PDFs) but also rasterized documents stored as pixel images. For the latter, the recognition leverages convolutional neural networks.

The retrieved integers are then fed into our error correction coding scheme to reconstruct the plain message. Because of the data redundancy, even if some glyphs are mistakenly recognized (e.g., due to poor image quality), those errors will be rectified and we will still be able to recover the message correctly. The embedding and retrieval process is summarized in Fig. 3.2.

## Section 4: Glyph Recognition

We start by focusing on our basic message embedding blocks: individual letters in a text document. Our goal in this section is to embed an integer number in a single letter by perturbing its glyph, and later retrieve that integer from a vector graphics or pixel image of that letter. In the next section, we will address what integers to assign to the letters in order to encode a message.

As introduced in §3.1, we embed an integer in a letter through glyph perturbation, by looking up a precomputed codebook. Later, when extracting an integer from a letter, we compute a “distance” metric between the extracted glyph and each perturbed glyph in  $\{u_0, \dots, u_{N-1}\}$  in the codebook; we obtain integer  $i$  if the “distance” of glyph  $u_i$  is the smallest one.

Our recognition algorithm supports input documents stored as vector graphics and pixel images. While it is straightforward to recognize vector graphic glyphs, pixel images pose significant challenges due to camera perspectives, rasterization noise, blurriness, and so forth. Our initial attempt used various template matching methods (e.g., [43]), but they easily become error-prone when image quality and camera perspective are not ideal.

In this section, we first describe our algorithm that decodes integers from rasterized (pixel) glyphs. This algorithm leverages convolutional neural networks (CNNs), which also allow us to systematically construct the codebook of perturbed glyphs. Next, we describe the details of embedding and extracting integers, as well as a simple algorithm for recognizing vector graphic glyphs.

### 4.1 Pixel Image Preprocessing

When a text document is provided as a pixel image, we use the off-the-shelf optical character recognition (OCR) library to detect individual letters. Our approach does not depend on any par-

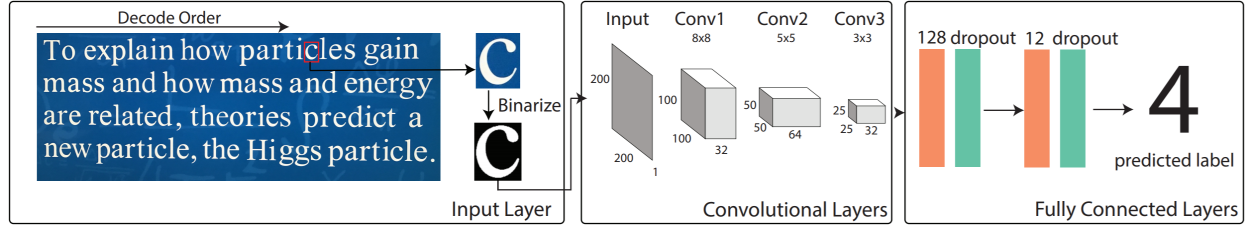


Figure 4.1: **Network architecture.** Our CNN takes as input a 200×200, black-and-white image containing a single letter (left). It consists 3 convolutional layers (middle) and 2 fully connected layers (right). It outputs a vector, in which each element is the estimated probability of recognizing the glyph in the input image as a perturbed glyph in the codebook.

ticular OCR library. In practice, we choose to use Tesseract [135], one of the most popular open source OCR engines. In addition to detecting letters, OCR also identifies a bounding box of every letter on the pixel image.

To recognize the glyph perturbation of each letter using CNNs, we first preprocess the image. We crop the region of each letter using its bounding box detected by the OCR. We then binarize the image region using the classic algorithm by Otsu [115]. This step helps to eliminate the influence caused by the variations of lighting conditions and background colors. Lastly, we resize the image region to have 200×200 pixels. This 200×200, black-and-white image for each letter is the input to our CNNs (Fig. 4.1-left).

## 4.2 Network Structure

We treat glyph recognition as an image classification problem: provided an image region of a letter which has a list of perturbed glyphs  $\{u_0, u_1, \dots\}$  in the codebook, our goal is to classify the input glyph of that letter as one from the list. Therefore, we train a CNN for each letter in a particular font.

Thanks to the image preprocessing, we propose to use a simple CNN structure (as illustrated in Fig. 4.1), which can be quickly trained and evaluated. The input is a 200×200, black-and-white image containing a letter. The CNN consists of three convolutional layers, each followed by a ReLU activation layer (i.e.,  $f : x \in \mathbb{R} \mapsto \max(0, x)$ ) and a 2×2 max pooling layer. The kernel size of the three convolutional layers are 8×8×32, 5×5×64, and 3×3×32, respectively. The

convolutional layers are connected with two fully connected (FC) layers, arranged in the following form:

$$\rightarrow \text{FC}(128) \rightarrow \text{Dropout} \rightarrow \text{FC}(N) \rightarrow \text{Softmax} \rightarrow \text{Output}.$$

Here “Dropout” indicates a 0.5-dropout layer, and  $\text{FC}(m)$  is an  $m$ -dimensional FC layer. In the second FC layer,  $N$  is the number of perturbed glyphs in the codebook for the letter in a certain font. After passing through a softmax layer, the output is an  $N$ -dimensional vector indicating the probabilities (or the inverse of “distance”) of classifying the glyph of the input letter as one of the perturbed glyphs in the list. The glyph of the letter is recognized as  $u_i$  if its corresponding probability in the output vector is the largest one.

### 4.3 Network Training

**Training data** Our CNNs will be used for recognizing text document images that are either directly synthesized or captured by digital cameras. Correspondingly, the training data of the CNNs consist of synthetic images and real photos. Consider a letter whose perturbed glyphs from a standard font are  $\{u_0, \dots, u_{N-1}\}$ . We print all the  $N$  glyphs on a paper and take 10 photos with different lighting conditions and slightly different camera angle. While taking the photos, the camera is almost front facing the paper, mimicking the scenarios of how the embedded information in a document would be read by a camera. In addition, we include synthetic images by rasterizing each glyph (whose vector graphics path is generated using [24]) into a  $200 \times 200$  image.

**Data augmentation** To train the CNNs, each training iteration randomly draws a certain number of images from the training data. We ensure that half of the images are synthetic and the other half are from real photos—empirically we found that the mix of real and synthetic data leads to better CNN recognition performance. To reduce overfitting and improve the robustness, we augment the selected images before feeding them into the training process. Each image is processed by the following operations:

- Adding a small Gaussian noise with zero mean and standard deviation 3.

- Applying a Gaussian blur whose standard deviation is uniformly distributed between 0 (no blur) and 3px.
- Applying a randomly-parameterized perspective transformation.
- Adding a constant border with a width randomly chosen between 0 and 30px.

We note that similar data augmentations have been used in [149, 29] to enrich their training data. We therefore refer to those papers for more details of data augmentation. Lastly, we apply the preprocessing routine described in §4.1, obtaining a binary image whose pixel values are either 0 or 1.

**Implementation details** In practice, our CNNs are optimized using the Adam algorithm [80] with the parameters  $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and a mini-batch size of 15. The network was trained with  $10^5$  update iterations at a learning rate of  $10^{-3}$ . Our implementation also leverages existing libraries, Tensorflow [1] and Keras [31].

#### 4.4 Constructing The Glyph Codebook

The CNNs not only help to recognize glyphs at runtime, but also enable us to systematically construct the glyph codebook, a lookup table that includes a set of perturbed glyphs for every character in commonly used fonts (such as Times New Roman, Helvetica, and Calibri). Our codebook construction aims to satisfy three criteria: (i) The included glyph perturbation must be perceptually similar; their differences from the original fonts should be hardly noticeable to our eyes. (ii) The CNNs must be able to distinguish the perturbed glyphs reliably. (iii) Meanwhile, we wish to include as many perturbed glyphs as possible in the codebook for each character, in order to increase the capacity of embedding information (see §5).

To illustrate our algorithm, consider a single character in a given font. We use its location  $\bar{u}$  on the character’s font manifold to refer to the original glyph of the font. Our construction algorithm first finds a large set of glyph candidates that are perceptually similar to  $\bar{u}$ . This step

uses crowdsourced perceptual studies following a standard user-study protocol. We therefore defer its details until §6, while in this section we denote the resulting set of glyphs as  $C$ , which typically has hundreds of glyphs. Next, we reduce the size of  $C$  by discarding glyphs that may confuse our CNNs in recognition tests. This is an iterative process, wherein each iteration takes the following two steps:

**Confusion test.** We randomly select  $M$  pairs of glyphs in  $C$  ( $M = 100$  in practice). For each pair, we check if our CNN structure (introduced in §4.2) can reliably distinguish the two glyphs. In this case, since only two glyphs are considered, the last FC layer has two dimensions. We train this network with only synthetic images processed by the data augmentations. We use synthetic images generated by different augmentations to test the accuracy of the resulting CNN. If the accuracy is less than 95%, then we record this pair of glyphs in a set  $\mathcal{D}$ , one that contains glyph pairs that cannot be distinguished by our CNN.

**Updating glyph candidate set.** Next, we update the glyph candidates in  $C$  to avoid the recorded glyph pairs that may confuse the CNNs while retaining as many glyph candidates as possible in  $C$ . To this end, we construct a graph where every node  $i$  represents a glyph in  $C$ . Initially, this graph is completely connected. Then, the edge between node  $i$  and  $j$  is removed if the glyph pair  $i\ j$  is listed in  $\mathcal{D}$ , meaning that  $i$  and  $j$  are hardly distinguishable by our CNN. With this graph, we find the maximum set of glyph that can be distinguished from each other. This amounts to the classic maximum clique problem on graphs. Although the maximum clique problem has been proven NP-hard (see page 97 of [73]), our graph size is small (with up to 200 nodes), and the edges are sparse. Consequently, many existing algorithms suffice to find the maximum clique efficiently. In practice, we choose to use the method of [83]. Lastly, the glyphs corresponding to the maximum clique nodes form the updated glyph candidate set  $C$ , and we move on to the next iteration.

This iterative process stops when there is no change of  $C$ . In our experiments, this takes up to 5 iterations. Because only synthetic images are used as training data, this process is fully automatic and fast. But it narrows down the glyph candidates conservatively—two glyphs in  $C$  might still

be hardly distinguishable in real photos. Therefore, in the last step, we train a CNN (as described in §4.3) using both synthetic images and real photos to verify if the CNN can reliably recognize all glyphs in  $C$ . Here the last FC layer of CNN has a dimension of  $|C|$  (i.e., the size of  $C$ ). At this point,  $|C|$  is small (typically around 25). Thus, the CNN can be easily trained. Afterwards, we evaluate the recognition accuracy of the CNN for each glyph in  $C$ , and discard the glyphs whose recognition accuracy is below 90%. The remaining glyphs in  $C$  are added to the codebook as the perturbed glyphs of the character. A fragment of our codebook for Times New Roman is shown in Fig. 4.2. Note that different characters have a different number of perturbed glyphs in the codebook. This difference poses a technical challenge when designing the message decoding algorithm in §5.

#### 4.5 Embedding and Retrieving Integers

**Embedding** Now we can embed integers into a text document in either vector graphic or pixel image format. First, we extract from the input document the text content and the layout of the letters. Our method also needs to know the original font  $\bar{u}$  of the text. This can be specified by the user or automatically obtained from the metadata of a vector graphic document (such as a PDF). If the document is provided as a pixel image, recent methods [29, 149] can be used to recognize the text font.

In order to embed an integer  $i$  in a letter of font  $\bar{u}$ , we look up its perturbed glyph list  $\{u_0, \dots, u_{N-1}\}$  in the precomputed codebook, and generate the letter shaped by the glyph  $u_i$  [24]. We then scale the glyph to fit it into the bounding box of the original letter (which is detected by the OCR library for pixel images), and use it to replace the original letter of the input document.

**Retrieval** To retrieve integers from a pixel image, we extract the text content and identify regions of individual letters using the OCR tool. After we crop the regions of the letters, they are processed in parallel by the recognition algorithm: they are preprocessed as described in §4.1 and then fed into the CNNs. An integer  $i$  is extracted from a letter if the output vector from its CNN has the  $i$ -th

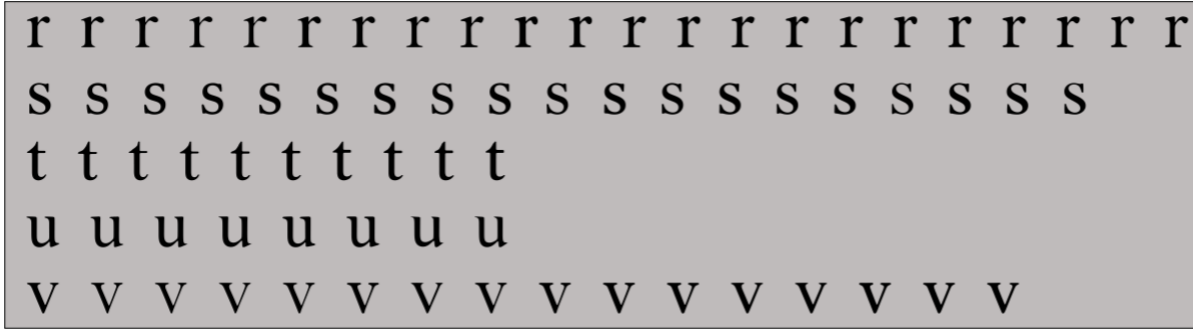


Figure 4.2: **A fragment of codebook.** A complete codebook for all alphabetic characters in lower case for Times New Roman can be found in the supplemental document.

element as the largest.

If the input document is provided as a vector graphics image (e.g., as a PDF file), glyph recognition of a letter is straightforward. In our codebook, we store the outlines of the perturbed glyphs as polylines. In the integer embedding stage, these polylines are scaled to replace the paths of the original glyphs. In the retrieval stage, we compute the “distance” between the letter’s glyph polyline, denoted as  $f$ , in the document and every glyph  $u_i$  in the perturbed glyph list of the codebook: we first scale the polylines of  $u_i$  to match the bounding box of  $f$ , and then compute the  $L_2$  distance between the polyline vertices of  $u_i$  and  $f$ . We recognize  $f$  as  $u_j$  (and thus extract the integer  $j$ ) if the distance between  $f$  and  $u_j$  is the minimum.

## Section 5: Error Correction Coding

After presenting the embedding and retrieval of integer values over individual letters, we now focus on a plain message represented as a bit string and describe our coding scheme that encodes the bit string into a series of integers, which will be in turn embedded in individual letters. We also introduce an error-correction decoding algorithm that decodes the bit string from a series of integers, even when some integers are incorrect.

Algorithms presented in this section are built on the coding theory for noise channels, founded by Shannon's landmark work [133]. We refer the reader to the textbook [97] for a comprehensive introduction. Here, we start with a brief introduction of traditional error-correcting codes, to point out their fundamental differences from our coding problem.

### 5.1 Challenges of the Coding Problem

The most common error-correction coding scheme is *block coding*, in which an information sequence is divided into blocks of  $k$  symbols each. We denote a block using a  $k$ -tuple,  $\mathbf{r} = (r_1, r_2, \dots, r_k)$ . For example, a 128-bit binary string can be divided into 16 blocks of 8-bit binary strings. In this case,  $k = 8$ , and  $r_i$  is either 0 or 1. In general,  $r_i$  can vary in other ranges of discrete symbols. A fundamental requirement of error-correction coding is that (i) the number of possible symbols for each  $r_i$  must be the same, and (ii) this number must be a prime power  $p^m$ , where  $p$  is a prime number and  $m$  is a positive integer. In abstract algebraic language, it requires the blocks to be in a *Galois Field*,  $\text{GF}(p^m)$ . For example, the aforementioned 8-bit strings are in  $\text{GF}(2)$ . Most of the current error-correction coding schemes (e.g., Reed-Solomon codes [121]) are built on the algebraic operations in the polynomial ring of  $\text{GF}(p^m)$ . At encoding time, they transform  $k$ -tuple blocks into  $n$ -tuple blocks in the same Galois field  $\text{GF}(p^m)$ , where  $n$  is larger than  $k$ . At decoding

time, some symbols in the  $n$ -tuple blocks can be incorrect. But as long as the total number of incorrect symbols in a block is no more than  $\lfloor \frac{n-k}{2} \rfloor$  regardless of the locations of incorrections, the coding scheme can fully recover the original  $k$ -tuple block.

In our problem, a text document consists of a sequence of letters. At first glance, we can divide the letter sequence into blocks of  $k$  letters each. Unfortunately, these blocks are ill-suited for traditional block coding schemes. For example, consider a five-letter block,  $(C_1, C_2, \dots, C_5)$ , with an original font  $\bar{u}$ . Every letter has a different capacity for embedding integers: in the codebook,  $C_i$  has  $s_i$  glyphs, so it can embed integers in the range  $[0, s_i)$ . However, in order to use block coding, we need to find a prime power  $p^m$  that is no more than any  $s_i, i = 1 \dots 5$  (to construct a Galois field  $\text{GF}(p^m)$ ). Then every letter can only embed  $p^m$  integers, which can be significantly smaller than  $s_i$ . Perhaps a seemingly better approach is to find  $t_i = \lfloor \log_2 s_i \rfloor$  for every  $s_i$ , and use the 5-letter block to represent a  $T$ -bit binary string, where  $T = \sum_{i=1}^5 t_i$ . In this case, this binary string is in  $\text{GF}(2)$ , valid for block coding. Still, this approach wastes much of the letters' embedding capacity. For example, if a letter has 30 perturbed glyphs, this approach can only embed integers in  $[0, 16)$ . As experimentally shown in §8, it significantly reduces the amount of information that can be embedded. In addition, traditional block coding method is often concerned with a noisy communication channel, where an error occurs at individual *bits*. In contrast, our recognition error occurs at individual *letters*. When the glyph of a letter is mistakenly recognized, a chunk of bits becomes incorrect, and the number of incorrect bits depends on specific letters. Thus, it is harder to set a proper relative redundancy (i.e.,  $(n - k)/n$  in block coding) to guarantee successful error correction.

## 5.2 Chinese Remainder Codes

To address these challenges, we introduce a new coding scheme based on a 1700-year old number theorem, the *Chinese remainder theorem* (CRT) [76]. Error-correction coding based on Chinese remainder theorem has been studied in the field of theoretical computing [51, 17], known as the *Chinese Remainder Codes* (CRC). We adopt CRC and further extend it to improve its error-

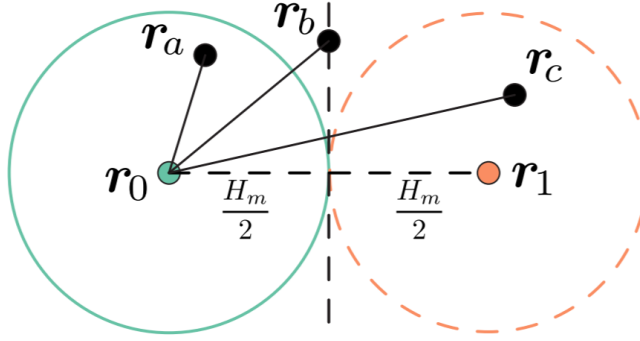


Figure 5.1: **Analogy** of Euclidean distance on 2D plane to gain intuition of Theorem 1. Here  $r_0$  and  $r_1$  represent two codewords having the minimum Hamming distance  $H_m$ . Suppose  $r_0$  is recognized as a code vector  $r$ . If the distance  $|r - r_0| < H_m/2$  (e.g.,  $r_a$ ), it is safe to decode  $r$  into  $r_0$ , as no other codeword is closer to  $r$ . If  $|r - r_0| > H_m/2$ , there might be multiple codewords having the same distance to  $r$  (e.g., when  $r$  is  $r_b$ ), and  $r$  may be mistakenly decoded as  $r_1$  (e.g., when  $r$  is  $r_c$ ).

correction ability.

**Problem definition** Given a text document, we divide its letter sequence into blocks of  $n$  letters each. Consider a block, denoted as  $\mathbb{C} = (C_1, C_2, \dots, C_n)$  in an original font  $\bar{u}$ . Our goal for now is to embed an integer  $m$  in this  $n$ -letter block, where  $m$  is in the range  $[0, M)$ . We will map the plain message into a series of integers later in this section. Formally, we seek an encoding function  $\phi : m \rightarrow r$ , where  $r = (r_1, \dots, r_n)$  is an  $n$ -vector of integers. Following the coding theory terminology, we refer  $r$  as a *codeword*. The function  $\phi$  needs to ensure that every  $r_i$  can be embedded in the letter  $C_i$ . Meanwhile,  $\phi$  must be injective, so  $r$  can be uniquely mapped to  $m$  through a decoding function  $\phi^+ : r \rightarrow m$ . Additionally, the computation of  $\phi$  and  $\phi^+$  must be fast, to encode and decode in a timely fashion.

### 5.2.1 Hamming Distance Decoding

We now introduce *Hamming Distance decoding*, a general decoding framework for block codes, to pave the way for our coding scheme.

**Definition 1 (Hamming Distance)** Given two codewords  $u$  and  $v$ , the Hamming Distance  $H(u, v)$  measures the number of pair-wise elements in which they differ.

For example, given two codewords,  $\mathbf{u} = (2, 3, 1, 0, 6)$  and  $\mathbf{v} = (2, 0, 1, 0, 7)$ ,  $H(\mathbf{u}, \mathbf{v}) = 2$ .

Now we consider glyph recognition errors. If the integer retrieval from a letter's glyph is incorrect, then the input  $\tilde{\mathbf{r}}$  of the decoding function  $\phi^+$  may not be a valid codeword. In other words, because of the recognition errors, it is possible that no  $m$  satisfies  $\phi(m) = \tilde{\mathbf{r}}$ . To distinguish from a codeword, we refer to  $\tilde{\mathbf{r}}$  as a *code vector*.

The Hamming distance decoding uses a decoding function  $\phi^+$  based on the Hamming distance:  $\phi^+(\tilde{\mathbf{r}})$  returns an integer  $m$  such that the Hamming distance  $H(\phi(m), \tilde{\mathbf{r}})$  is minimized over all  $m \in [0, M)$ . Intuitively, although  $\tilde{\mathbf{r}}$  may not be a valid codeword, we decode it as the integer whose codeword is closest to  $\tilde{\mathbf{r}}$  under the measure of Hamming distance. Let  $H_m$  denote the *minimum* Hamming distance among all pairs of valid codewords, namely,

$$H_m = \min_{\substack{i, j \in [0, M-1] \\ i \neq j}} H(\phi(i), \phi(j)). \quad (5.1)$$

The error-correction ability of Hamming distance decoding is bounded by the following theorem [97],

**Theorem 1** *Let  $E$  denote the number of incorrect symbols in a code vector  $\tilde{\mathbf{r}}$ . Then, the Hamming distance decoding can correctly recover the integer  $m$  if  $E \leq \left\lfloor \frac{H_m-1}{2} \right\rfloor$ ; it can detect errors in  $\tilde{\mathbf{r}}$  but not recover the integer correctly if  $\left\lfloor \frac{H_m-1}{2} \right\rfloor < E < H_m$ .*

An illustrative understanding of this theorem using an analogy of Euclidean distance is shown in Fig. 5.1. This theorem holds regardless of the encoding function  $\phi$ , as long as it is a valid injective function.

### 5.2.2 Chinese Remainder Codes

We now introduce the following version of the Chinese Remainder Theorem, whose proof can be found in [126].

**Theorem 2 (Chinese Remainder Theorem)** *Let  $p_1, p_2, \dots, p_k$  denote positive integers which are*

mutually prime and  $M = \prod_{i=1}^k p_i$ . Then, there exists an injective function,

$$\phi : [0, M) \rightarrow [0, p_1) \times [0, p_2) \times \dots [0, p_k),$$

defined as  $\phi(m) = (r_1, r_2, \dots, r_n)$ , such that for all  $m \in [0, M)$ ,  $r_i = m \bmod p_i$ .

This theorem indicates that given  $k$  pairs of integers  $(r_i, p_i)$  with all  $p_i$  being mutually prime, there exists a unique non-negative integer  $m < \prod_{i=1}^k p_i$  satisfying  $r_i = m \bmod p_i$  for all  $i = 1 \dots k$ . Indeed,  $m$  can be computed using the formula,

$$m = CRT(r, p) = r_1 b_1 \frac{P}{p_1} + \dots + r_n b_n \frac{P}{p_n}, \quad (5.2)$$

where  $P = \prod_{i=1}^k p_i$ , and  $b_i$  is computed by solving a system of modular equations,

$$b_i \frac{P}{p_i} \equiv 1 \pmod{p_i}, \quad i = 1 \dots k, \quad (5.3)$$

using the classic Euclidean algorithm [126].

If we extend the list of  $p_i$  to  $n$  mutually prime numbers ( $n > k$ ) such that the  $n - k$  additional numbers are all larger than  $p_i$  up to  $i = k$  (i.e.,  $p_j > p_i$  for any  $j = k+1 \dots n$  and  $i = 1 \dots k$ ), then we have an encoding function  $\phi$  for non-negative integer  $m < \prod_{i=1}^k p_i$ :

$$\phi(m) = (m \bmod p_1, m \bmod p_2, \dots, m \bmod p_n). \quad (5.4)$$

This encoding function already adds redundancy: because  $m$  is smaller than the product of any  $k$  numbers chosen from  $p_i$ , we can compute  $m$  from any  $k$  of the  $n$  pairs of  $(r_i, p_i)$ , according to the Chinese Remainder Theorem. Indeed, as proved in [51], the minimum Hamming distance of the encoding function (5.4) for all  $0 \leq m < \prod_{i=1}^k p_i$  is  $n - k + 1$ . Thus, the Hamming decoding function of  $\phi$  can correct up to  $\lfloor \frac{n-k}{2} \rfloor$  errors by Theorem 1.

Bit stream	010111 10111010 1000 1001001
Integer sequence	23 186 8 73
Letter seq.	Bilbo was ve ry ric h and very...

Figure 5.2: **An example** of determining  $m_t$ : (top) The bit string representation of the plain message; (bottom) Each block of letters is highlighted by a color; (middle)  $m_t$  values assigned to each block.

**Encoding** We now describe our encoding algorithm based on the encoding function  $\phi(m)$  in (5.4).

- **Computing  $p_i$ .** Suppose that the letter sequence of a document has been divided into  $N$  blocks, denoted as  $\mathbb{C}_1, \dots, \mathbb{C}_N$ , each with  $n$  letters. Consider a block  $\mathbb{C}_t = (C_1, \dots, C_n)$ , where  $C_i$  indicates a letter with its original font  $u_i$ , whose integer embedding capacity is  $s_i$  (i.e.,  $C_i$ 's font  $u_i$  has  $s_i$  perturbed glyphs in the codebook). We depth-first search  $n$  mutually prime numbers  $p_i, i = 1 \dots n$ , such that  $p_i \leq s_i$  and the product of  $k$  minimal  $p_i$  is maximized. At the end, we obtain  $p_i, i = 1 \dots n$  and the product of  $k$  minimal  $p_i$  denoted as  $M_t$  for each block  $\mathbb{C}_t$ . Note that if we could not find mutually prime numbers  $p_i$  for block  $\mathbb{C}_t$ , we simply ignore the letter whose embedding capacity is smallest among all  $s_i$  and include  $C_{n+1}$  to this block. We repeat this process until we find a valid set of mutually prime numbers.
- **Determining  $m_t$ .** Given the plain message represented as a bit string  $\mathbb{M}$ , we now split the bits into a sequence of chunks, each of which is converted into an integer and assigned to a block  $\mathbb{C}_t$ . We assign to each block  $\mathbb{C}_t$  an integer  $m_t$  with  $\lfloor \log_2 M_t \rfloor$  bits, which is sequentially cut from the bit string  $\mathbb{M}$  (see Fig. 5.2).
- **Embedding.** For every block  $\mathbb{C}_t$ , we compute the codeword using the CRT encoding function (5.4), obtaining  $r = (r_1, \dots, r_n)$ . Each  $r_i$  is then embedded in the glyph of the letter  $C_i$  in the block  $\mathbb{C}_t$  as described in §4.5.

**Decoding** At decoding time, we recognize the glyphs of the letters in a document and extract integers from them, as detailed in §4.5. Next, we divide the letter sequence into blocks, and repeat the algorithm of computing  $p_i$  and  $M_t$  as in the encoding step, for every block. Given a block  $\mathbb{C}_t$ , the extracted integers from its letters form a code vector  $\tilde{\mathbf{r}}_t = (\tilde{r}_1, \dots, \tilde{r}_n)$ . We refer to it as a code vector because some of the  $\tilde{r}_i$  may be incorrectly recognized. To decode  $\tilde{\mathbf{r}}_t$ , we first compute  $\tilde{m}_t = CRT(\tilde{\mathbf{r}}_t, \mathbf{p}_t)$  where  $\mathbf{p}_t$  stacks all  $p_i$  in the block. If  $\tilde{m}_t < M_t$ , then  $\tilde{m}_t$  is the decoding result  $\phi^+(\tilde{\mathbf{r}}_t)$ , because the Hamming distance  $H(\phi(\tilde{m}_t), \tilde{\mathbf{r}}) = 0$ . Otherwise, we decode  $\tilde{\mathbf{r}}_t$  using the Hamming decoding function: concretely, since we know the current block can encode an integer in the range  $[0, M_t)$ , we decode  $\tilde{\mathbf{r}}_t$  into the integer  $m_t$  by finding

$$m_t = \phi^+(\tilde{\mathbf{r}}) = \arg \min_{m \in [0, M_t)} H(\phi(m), \tilde{\mathbf{r}}). \quad (5.5)$$

As discussed above, this decoding function can correct up to  $\lfloor \frac{n-k}{2} \rfloor$  incorrectly recognized glyphs in each block. Lastly, we convert  $m_t$  into a bit string and concatenate  $m_t$  from all blocks sequentially to recover the plain message.

**Implementation details** A few implementation details are worth noting. First, oftentimes letters in a document can carry a bit string much longer than the given plain message. To indicate the end of the message, we attach a special chunk of bits (end-of-message bits) at the end of each plain message, very much akin to the end-of-line (newline) character used in digital text systems. Second, in practice, blocks should be relatively short (i.e.,  $n$  is small). If  $n$  is large, it becomes much harder to find  $n$  mutually prime numbers that are no more than each letter's embedding capacity. In practice, we choose  $n = 5$  and  $k = 3$  which allows one mistaken letter in every 5-letter block. The small  $n$  and  $k$  also enable brute-force search of  $m_t$  in (5.5) sufficiently fast, although there also exists a method solving (5.5) in polynomial time [51, 17].

### 5.3 Improved Error Correction Capability

Using the Chinese Remainder Codes, the error-correction capacity is upper bounded. Theorem 1 indicates that at most  $\lfloor \frac{n-k}{2} \rfloor$  mistakenly recognized glyphs are allowed in every letter block. We now break this theoretical upper bound to further improve our error-correction ability, by exploiting specific properties of our coding problem. To this end, we propose a new algorithm based on the maximum likelihood decoding [97].

In coding theory, maximum likelihood decoding is not an algorithm. Rather, it is a decoding philosophy, a framework that models the decoding process from a probabilistic rather than an algebraic point of view. Consider a letter block  $\mathbb{C}$  and the code vector  $\tilde{\mathbf{r}}$  formed by the extracted integers from  $\mathbb{C}$ . We treat the true codeword  $\mathbf{r}$  encoded by  $\mathbb{C}$  as a *latent variable* (in statistic language), and model the probability of  $\mathbf{r}$  given the extracted code vector  $\tilde{\mathbf{r}}$ , namely  $\mathbb{P}(\mathbf{r}|\tilde{\mathbf{r}})$ . With this likelihood model, our decoding process finds a codeword  $\mathbf{r}$  that maximizes the probability  $\mathbb{P}(\mathbf{r}|\tilde{\mathbf{r}})$ , and decodes  $\mathbf{r}$  into an integer using the Chinese Remainder Theorem formula (5.2).

When there are at most  $\lfloor \frac{n-k}{2} \rfloor$  errors in a block, the Hamming decoding function (5.5) is able to decode. In fact, it can be proved that when the number of errors is under this bound, there exists a unique  $m \in [0, M_t)$  that minimizes  $H(\phi(m), \tilde{\mathbf{r}})$  [97], and that when the number of errors becomes  $\lfloor \frac{n-k}{2} \rfloor + 1$ , there may be multiple  $m \in [0, M_t]$  reaching the minimum (see Fig. ?? for an intuitive explanation). The key idea of breaking this bound is to use a likelihood model to choose an  $m$  when ambiguity occurs for the Hamming decoding function (5.5).

Consider a block with  $\lfloor \frac{n-k}{2} \rfloor + 1$  errors, and suppose in (5.5) we find  $N_c$  different integers  $m_i$ , all of which lead to the same minimal Hamming distance to the code vector  $\tilde{\mathbf{r}}$ . Let  $\mathbf{r}_i = \phi(m_i), i = 1 \dots N_c$ , denote the corresponding codewords. We use another subscript  $j$  to index the integer elements in code

	$\tilde{\mathbf{r}}$	$\tilde{r}_1$	$\tilde{r}_2$	$\tilde{r}_3$	$\tilde{r}_4$	$\tilde{r}_5$
Code Vector		2	3	1	7	6
	$\mathbf{r}_1$	$r_{11}$	$r_{12}$	$r_{13}$	$r_{14}$	$r_{15}$
Codeword 1		2	3	4	5	6
	$\mathbf{r}_2$	$r_{21}$	$r_{22}$	$r_{23}$	$r_{24}$	$r_{25}$
Codeword 2		4	3	5	7	6

vectors and codewords. For every  $\mathbf{r}_i$ , some of its integer elements  $r_{ij}$  differs from the corresponding integer elements  $\tilde{r}_j$ . Here  $r_{ij}$  can be interpreted as the index of the perturbed glyphs of the  $j$ -th letter. We denote this glyph as  $\mathbf{u}_{ij}$  and the letter’s glyph extracted from the input image as  $\mathbf{f}$ . If  $r_{ij}$  is indeed the embedded integer, then  $\mathbf{f}$  is mistakenly recognized, resulting in a different glyph number  $\tilde{r}_j$ .

We first model the probability of this occurrence, denoted as  $\mathbb{P}(\tilde{r}_j|r_{ij})$ , using our “distance” metric. Intuitively, the closer the two glyphs are, the more likely one is mistakenly recognized as the other. Then the probability of recognizing a codeword  $\mathbf{r}_i$  as a code vector  $\tilde{\mathbf{r}}$  accounts for all inconsistent element pairs in  $\mathbf{r}_i$  and  $\tilde{\mathbf{r}}$ , namely,

$$\mathbb{P}(\tilde{\mathbf{r}}|\mathbf{r}_i) = \prod_{j, r_{ij} \neq \tilde{r}_j} \frac{g(\mathbf{u}_{ij}, \mathbf{f})}{\sum_{k=1}^{s_j} g(\tilde{\mathbf{u}}_j^k, \mathbf{f})}. \quad (5.6)$$

Here  $g(\cdot, \cdot)$  is (inversely) related to our distance metric between two glyphs: for pixel images,  $g(\mathbf{u}_{ij}, \mathbf{f})$  is the probability of recognizing  $\mathbf{f}$  as  $\mathbf{u}_{ij}$ , which is the  $r_{ij}$ -th softmax output from the CNN of the  $i$ -th letter, given the input glyph image  $\mathbf{f}$ ; for vector graphics,  $g(\mathbf{u}_{ij}, \mathbf{f}) = 1/d(\mathbf{u}_{ij}, \mathbf{f})$ , the inverse of the vector graphic glyph distance. The denominator is for normalization, where  $\tilde{\mathbf{u}}_j^k$  iterates through all perturbed glyphs of the  $j$ -th letter in the codebook. For pixel images, the denominator is always 1 because of the unitary property of the softmax function. Lastly, the likelihood  $\mathbb{P}(\mathbf{r}_i|\tilde{\mathbf{r}})$  needed for decoding is computed by Bayes Theorem,

$$\mathbb{P}(\mathbf{r}_i|\tilde{\mathbf{r}}) = \frac{\mathbb{P}(\tilde{\mathbf{r}}|\mathbf{r}_i)\mathbb{P}(\mathbf{r}_i)}{\mathbb{P}(\tilde{\mathbf{r}})}. \quad (5.7)$$

Here,  $\mathbb{P}(\tilde{\mathbf{r}})$  is fixed, and so is  $\mathbb{P}(\mathbf{r}_i)$  as all codewords are equally likely to be used. As a result, we find  $\mathbf{r}_i$  that maximizes (5.7) among all  $N_c$  ambiguous codewords, and decode  $\tilde{\mathbf{r}}$  using  $\mathbf{r}_i$ .

In our experiments, we show that the proposed decoding scheme indeed improves the error-correction ability. For our implementation wherein  $n = 5$  and  $k = 3$ , in each block we are able to correct up to **2** errors, as opposed to 1 indicated by Theorem 1, thereby increasing the error tolerance from 20% to 40%.

## Section 6: Perceptual Evaluation

We now describe our crowd-sourced perceptual studies on Mechanical Turk (MTurk). The goal of the studies is, for each character in a particular font, to find a set of glyphs that are perceptually similar to its original glyph  $\bar{u}$ . When constructing the glyph codebook, we use these sets of glyphs to initialize the candidates of perturbed glyphs.

The user studies adopt a well-established protocol: we present the MTurk raters multiple questions, and each question uses a two-alternative forced choice (2AFC) scheme, i.e., the MTurk raters must choose one from two options. We model the rater’s response using a logistic function (known as the Bradley-Terry model [19]), which allows us to learn a perceptual metric (in this case the perceptual similarity of glyphs) from the raters’ response. We note that this protocol has been used previously in other perceptual studies (e.g., [113, 145]), and that we will later refer back to this protocol in §8.3 when we evaluate the perceptual quality of our results. Next, we describe the details of the studies.

**Setup** We first choose on the font manifold a large region centered at  $\bar{u}$ , and sample locations densely in this region. In practice, all font manifolds are in 2D, and we choose 400 locations uniformly in a squared region centered at  $\bar{u}$ . Let  $\mathcal{F}$  denote the set of glyphs corresponding to the sampled manifold locations. Then, in each MTurk question, we present the rater a pair of glyphs randomly selected from  $\mathcal{F}$ , and ask which one of the two glyphs looks closer to the glyph of the original font  $\bar{u}$ . An example is shown in Fig. 6.1. We assign 20 questions to each MTurk rater. Four of them are control questions, which are designed to detect untrustworthy raters. The four questions ask the rater to compare the same pair of glyphs presented in different order. We reject raters whose answers have more than one inconsistencies among the control questions. At the end, about 200 raters participated the studies for each character.

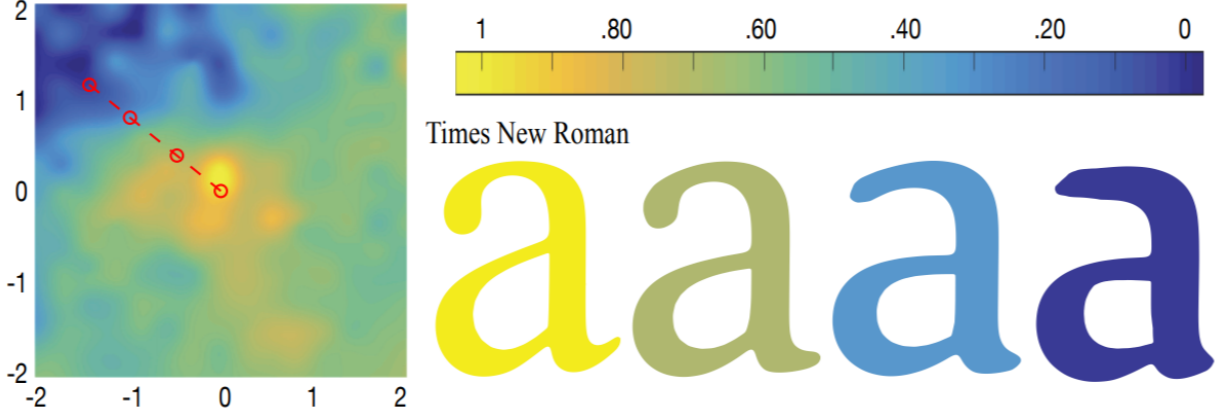


Figure 6.1: **MTurk result.** (left) The font manifold of the letter “a” color-mapped using the perceptual similarity value of each point to the standard Time New Roman font. (right) Sampled glyphs that correspond to the manifold locations indicated by red circles. The colors indicate their perceptual similarity to the Times New Roman.

<p><b>Instructions:</b></p> <p>Your task is to select the most similar font pair from the two pairs of font images.</p> <p>The two images on the left column are always identical fonts.</p>	<p><b>Question1</b></p> <table style="width: 100%; text-align: center;"> <tr> <td style="width: 50%;"><input type="radio"/></td> <td style="width: 25%;"></td> <td style="width: 25%;"></td> </tr> <tr> <td style="border-top: 1px solid #ccc; border-bottom: 1px solid #ccc;"><input type="radio"/></td> <td style="border-top: 1px solid #ccc; border-bottom: 1px solid #ccc;"></td> <td style="border-top: 1px solid #ccc; border-bottom: 1px solid #ccc;"></td> </tr> </table>	<input type="radio"/>			<input type="radio"/>		
<input type="radio"/>							
<input type="radio"/>							

Figure 6.2: MTurk user interface.

**Analysis** From the raters’ response, we estimate a scalar value  $s_i = s(\mathbf{u}_i; \bar{\mathbf{u}})$ , the perceptual similarity value of the glyph  $\mathbf{u}_i$  to the original glyph  $\bar{\mathbf{u}}$ , for every glyph  $\mathbf{u}_i \in \mathcal{F}$ . Let a set of tuples  $\mathcal{D} = \{(\mathbf{u}_i, \mathbf{u}_j, u, q)\}$  record the user study responses, where  $\mathbf{u}_i$  and  $\mathbf{u}_j$  are randomly selected glyphs being compared,  $u$  is the MTurk rater’s ID, and the binary value  $q$  is the rater’s choice:  $q = 1$  indicates the rater judges  $\mathbf{u}_i$  perceptually closer to  $\bar{\mathbf{u}}$ , and  $q = 0$  otherwise. We model the likelihood of the rater’s choice using a logistic function,

$$p(q = 1 | \mathbf{u}_i, \mathbf{u}_j, u) = \frac{1}{1 + \exp(r_u(s_i - s_j))}, \quad (6.1)$$

where the scalar  $r_u$  indicates the rater's reliability. With this model, all the similarity values  $s_i$  and user reliability values  $r_u$  are obtained by minimizing a negative log-likelihood objective function,

$$E(\mathbf{s}, \mathbf{r}) = - \sum_k \left[ q^k \ln p(q = 1 | \mathbf{u}_i^k, \mathbf{u}_j^k, u^k) + (1 - q^k) \ln p(q = 0 | \mathbf{u}_i^k, \mathbf{u}_j^k, u^k) \right], \quad (6.2)$$

where  $k$  indices the MTurk response in  $\mathcal{D}$ ,  $\mathbf{r}$  stacks the raters' reliability values, and  $\mathbf{s}$  stacks the similarity values for all  $\mathbf{u}_i \in \mathcal{F}$ . The  $s_i$  values are then normalized in the range of  $[0, 1]$ .

We learn the similarity values for the glyphs of every character with a font  $\bar{\mathbf{u}}$  independently. Fig. 6.1 visualizes the perceptual similarity of glyphs near the standard Times New Roman for the character “a”. Lastly, We iterate through all the pre-sampled glyphs  $\mathbf{u}_i \in \mathcal{F}$ , and add those whose similarity value  $s_i$  is larger than a threshold (0.85 in practice) into a set  $C$ , forming the set of perceptually similar glyphs for constructing the codebook.

## Section 7: Applications

Our method finds many applications. In this section, we discuss four of them, while referring to the supplemental video for their demonstrations and to §8 for implementation summaries.

### 7.1 Application I: Format-Independent Metadata

Many digital productions carry *metadata* that provide additional information, resources, and digital identification [55]. Perhaps most well-known is the metadata embedded in photographs, providing information such as camera parameters and copyright. PDF files can also contain metadata<sup>1</sup>. In fact, metadata has been widely used by numerous tools to edit and organize digital files.

Currently, the storage of metadata is *ad hoc*, depending on specific file format. For instance, a JPEG image stores metadata in its EXIF header, while an Adobe PDF stores its metadata in XML format with Adobe’s XMP framework [2]. Consequently, metadata is lost whenever one converts an image from JPEG to PNG format, or rasterizes a vector graphic document into a pixel image. Although it is possible to develop a careful converter that painstakingly preserves the metadata across all file formats, the metadata is still lost whenever the image or document is printed on paper.

Our FontCode technique can serve as a means to host text document metadata. More remarkably, the metadata storage in our technique is *format-independent*. Once information is embedded in a document, one can freely convert it to a different file format, or rasterize it into a pixel image (as long as the image is not severely downsampled), or print it on a piece of paper. Throughout, the glyphs of letters are preserved, and thereby metadata is retained (see video).

---

<sup>1</sup>If you are reading this paper with Adobe Acrobat Reader, you can view its metadata by choosing “File→Properties” and clicking the “Additional Metadata” under the “Description” tab.

## 7.2 Application II: Imperceptible Optical Codes

Our FontCode technique can also be used as optical barcodes embedded in a text document, akin to QR codes [42]. Barcodes have numerous applications in advertising, sales, inventory tracking, robotics, augmented reality, and so forth. Similar to QR codes that embed certain level of redundancy to correct decoding error, FontCode also supports error-correction decoding. However, all existing barcodes require to print black-and-white blocks and bars, which can be visually distracting and aesthetically imperfect. Our technique, in contrast, enables not only an optical code but an unobtrusive optical code, as it only introduces subtle changes to the text appearance. Our retrieval algorithm is sufficiently fast to provide point-and-shoot kind of message decoding. It can be particularly suitable for use as a replacement of QR codes in an artistic work such as a poster or flyer design, where visual distraction needs to be minimized. As a demonstration, we have implemented an iPhone application to read a hidden message from coded text (see video).

## 7.3 Application III: Encrypted Message Embedding

Our technique can further encrypt a message when embedding it in a document, even if the entire embedding and retrieval algorithms are made public. Recall that when embedding an integer  $i$  in a letter  $c$  of a glyph  $\bar{u}$ , we replace  $\bar{u}$  with a glyph chosen from a list of perturbed glyphs in the codebook. Let  $\mathbb{L}_c = \{\mathbf{u}_0, \dots, \mathbf{u}_{N_c-1}\}$  denote this list. Even though the perturbed glyphs for every character in a particular font are precomputed, the order of the glyphs in each list  $\mathbb{L}_c$  can be arbitrarily user-specified. The particular orders of all  $\mathbb{L}_c$  together can serve as an encryption key.

For example, when Alice and Bob<sup>2</sup> communicate through encoded documents, they can use a publicly available codebook, but agree on a private key, which specifies the glyph permutation of each list  $\mathbb{L}_c$ . If an original list  $\{\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \dots\}$  of a letter is permuted into  $\{\mathbf{u}_{p_0}, \mathbf{u}_{p_1}, \mathbf{u}_{p_2}, \dots\}$  by the key, then Alice uses the glyph  $\mathbf{u}_{p_i}$ , rather than  $\mathbf{u}_i$ , to embed an integer  $i$  in the letter, and Bob decipheres the message using the same permuted codebook. For a codebook that we precomputed

---

<sup>2</sup>Here we follow the convention in cryptography, using Alice and Bob as placeholder names for the convenience of presenting algorithms.

for Times New Roman (see the supplemental document), if we only consider lowercase English alphabet, there exist  $1.39 \times 10^{252}$  different glyph permutations in the codebook; if we also include uppercase English alphabet, there exist  $5.73 \times 10^{442}$  glyph permutations. Thus, without resorting to any existing cryptographic algorithm, our method already offers a basic encryption scheme. Even if others can carefully examine the text glyphs and discover that a document is indeed embedding a message, the message can still be protected from leaking.

#### 7.4 Application IV: Text Document Signature

Leveraging existing cryptographic techniques, we augment FontCode to propose a new digital signature technique, one that can authenticate the source of a text document and guarantee its integrity (thereby protecting from tampering). This technique has two variations, working as follows:

**Scheme I** When Alice creates a digital document, she maps the document content (e.g., including letters, digits, and punctuation) into a bit string through a cryptographic hash function such as the MD5 [123] and SHA [44]. We call this bit string the document’s *hash string*. Alice then chooses a private key to permute the codebook as described in §3.1, and uses the permuted codebook to embed the hash string into her document. When Bob tries to tamper this document, any change leads to a different hash string. Without knowing Alice’s private key, he cannot embed the new hash string in the document, and thus cannot tamper the document successfully. Later, Alice can check the integrity of her document, by extracting the embedded hash string and comparing it against the hash string of the current document.

**Scheme II** The above algorithm allows only Alice to check the integrity of her document, as only she knows her private key. By combining with *asymmetric cryptography* such as the RSA algorithm [124], we allow everyone to check the integrity of a document but not to tamper it. Now, the codebook is public but not permuted. After Alice generates the hash string of her document, she encrypts the hash string using her private key by an asymmetric cryptosystem, and obtains

an *encrypted string*. Then, she embeds the encrypted string in the document using the FontCode method, while making her public key available to everyone. In this case, Bob cannot tamper the document, as he does not have Alice’s private key to encrypt the hash string of an altered document. But everyone in the world can extract the encrypted string using the FontCode method, and decipher the hash string using Alice’s public key. If the deciphered hash string matches the hash string of the current document, it proves that (i) the document is indeed sourced from Alice, and (ii) the document has not been modified by others.

**Advantages** In comparison to existing digital signatures such as those in Adobe PDFs, our method is *format-independent*. In contrast to PDF files whose signatures are lost when the files are rasterized or printed on physical papers, our FontCode signature is preserved regardless of file format conversion, rasterization, and physical printing.

**Further extension** In digital text forensics, it is often desired to not only detect tampering but also locate where the tampering occurs. In this regard, our method can be extended for more detailed tampering detection. As shown in our analysis (§8.2), in a typical English document, we only need about 80 letters to embed (with error correction) a string of 128 bits, which is the length of a hash string resulted from a typical cryptographic hash function (e.g., MD5). Given a document, we divide its text into a set of segments, each with at least 80 letters. We then compute a hash string for each segment and embed the encrypted strings in individual segments. This creates a fine granularity of text signatures, allowing the user to check which text segment is modified, and thereby locating tampering occurrences more precisely. For example, in the current text format of this paper, every two-column line consists of around 100 letters, meaning that our method can identify tampering locations up to two-column lines in this paper. To our knowledge, digital text protection with such a fine granularity has not been realized.

**Discussion about the storage** We close this section with a remark on the memory footprint of the documents carrying messages. If a document is stored as a pixel image, then it consumes no

additional memory. If it is in vector graphics format, our current implementation stores the glyph contour polylines of all letters. A document with 376 letters with 639 bits encoded will consume 1.2M memory in a compressed SVG form and 371K in a compressed JPEG format. PDF files can embed glyph shapes in the file and refer to those glyphs in the text. Using this feature, we can also embed the entire codebook in a PDF, introducing about 1.3M storage overhead, regardless of the text length. In the future, if all glyphs in the codebook are pre-installed on the operating system, like the current standardized fonts, then the memory footprint of vector graphic documents can be further reduced, as the PDF and other vector graphic file format are able to directly refer to those pre-installed fonts.

## Section 8: Results and Validation

We now present the results and experiments to analyze the performance of our technique and validate our algorithmic choices. Here we consider text documents with English alphabet, including both lower- and upper-case letters, while the exact method can be directly applied to digits and other special characters. We first present our main results (§8.1), followed by the numerical (§8.2) and perceptual (§8.3) evaluation of our method.

### 8.1 Main Results

We implemented the core coding scheme on an Intel Xeon E5-1620 8 core 3.60GHz CPU with 32GB of memory. The CNNs are trained with an NVidia Geforce TITAN X GPU. Please see our accompanying video for the main results. A side-by-side comparison of an original document with a coded document is shown in Fig. 8.1.

**Metadata viewer** We implemented a simple text document viewer that loads a coded document in vector graphics or pixel image format. The viewer displays the document. Meanwhile, it extracts the embedded metadata with our decoding algorithm and presents it in a side panel.

**Unobtrusive optical codes** We also implemented an iPhone application, by which the user can take a photo of an encoded text displayed on a computer screen or printed on paper. The iPhone application interface allows the user to select a text region to capture. The captured image is sent through the network to a decoding server, which recovers the embedded message and sends it back to the smartphone.

Pippin looked out from the shelter of Gandalf's cloak. He wondered if he was awake or still sleeping, still in the swift moving dream in which he had been wrapped so long since the great ride began. The dark world was rushing by and the wind sang loudly in his ears. He could see nothing but the wheeling stars, and away to his right vast shadows against the sky where the mountains of the South marched.

Pippin looked out from the shelter of Gandalf's cloak. He wondered if he was awake or still sleeping, still in the swift moving dream in which he had been wrapped so long since the great ride began. The dark world was rushing by and the wind sang loudly in his ears. He could see nothing but the wheeling stars, and away to his right vast shadows against the sky where the mountains of the South marched.

Figure 8.1: **(top)** an input text document. **(bottom)** the output document that embeds a randomly generated message.

**Embedding encrypted message** Our implementation allows the user to load an encryption key file that specifies the permutation for all the lists of perturbed glyphs in the codebook. The permutation can be manually edited, or randomly generated—given a glyph list of length  $n$ , one can randomly sample a permutation from the permutation group  $\mathbb{S}_n$  [131] and attach it to the key.

**Text document signature** We use our technique to generate a MD5 signature as described in §???. Since the MD5 checksum has only 128 bits, we always embed it in letters from the beginning of the text. Our text document viewer can check the signature and alert the user if the document shows as tampered.

## 8.2 Validation

**Information capacity** As described in §5, we use  $n = 5$  and  $k = 3$  in our error-correction coding scheme. In every 5-letter block, if the mutually prime numbers are  $p_i, i = 1 \dots 5$ , then this block can encode integers in  $[0, p_1 p_2 p_3)$ , where  $p_1, p_2$ , and  $p_3$  are the smallest three numbers among  $p_i, i = 1 \dots 5$ . Thus, this block can encode at most  $\lfloor \log_2(p_1 p_2 p_3) \rfloor$  bits of information.

To estimate the information capacity of our scheme for English text, we randomly sample

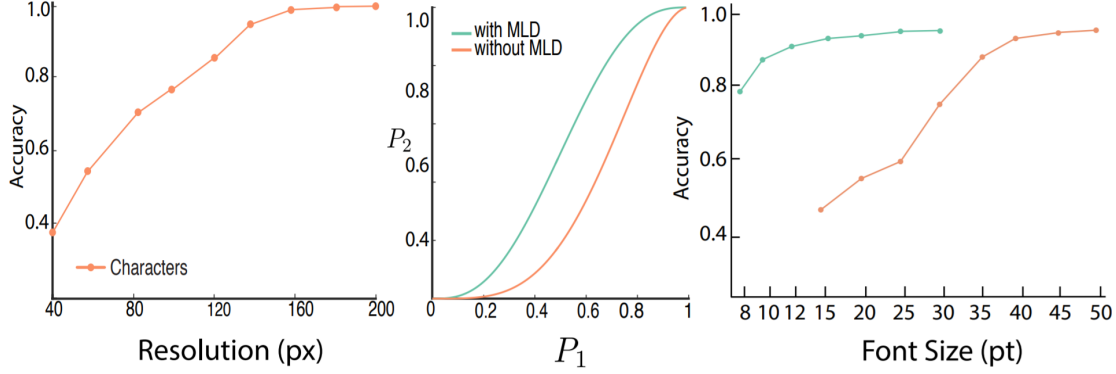
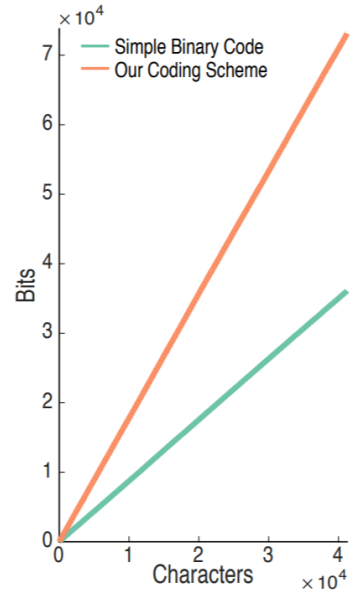


Figure 8.2: **(left)** The accuracy of our CNN decoder changes as the resolution in height of the letters increases. **(middle)** Theoretical improvement of maximum likelihood decoding (green) over the Chinese Remainder coding (orange). Please see the main text for the definition of  $P_1$  and  $P_2$ . **(right)** The accuracy of our CNN decoder trained with two different font sizes (green curve for 15pt fonts and orange curve for 30pt fonts) printed on paper. Please see the main text for the details of experiment setup.

5 characters from the alphabet to forming a block. The characters are sampled based on the widely known English letter frequencies (e.g., “e” is the most frequently used while “z” is the least used) [47]. We compute the average number of bits that can be encoded by a character. The result is **1.77**, suggesting that on average we need 73 letters to encode a 128-bit MD5 checksum for the application of digital signature.

Next, we compare our coding scheme with the simple approach discussed in §5.1. Recall that our method can correct at least one error in a block of 5 letters, which amounts to correcting  $\log_2(\max s_i)$  bits out of the total  $\sum_{i=1}^5 \log_2 s_i$  bits. The simple approach in §5.1 can store  $\sum_{i=1}^5 \lfloor \log_2 s_i \rfloor$  bits of information. But in order to correct one recognition error of the letter with the standard linear block codes, it needs to spend  $2 \lceil \log_2(\max s_i) \rceil$  bits for adding redundancy, leaving  $\sum_{i=1}^5 \lfloor \log_2 s_i \rfloor - 2 \lceil \log_2(\max s_i) \rceil$  bits to store information. We compare it with our method using *The Lord of The Rings, Chapter I* as our input text, which contains in total 41682 useful letters (9851 words). As shown in the adjacent figure, as the number of letters increases, our method can embed significantly more information.



**Decoding accuracy** We first evaluate the glyph recognition accuracy of our CNNs. For every character, we print it repeatedly on a paper with randomly chosen glyphs from the codebook and take five photos under different lighting conditions. Each photo has regions of around 220px×220px containing a character. We use these photos to test CNN recognition accuracy, and for all characters, the accuracy is above 90%.

We also evaluate the decoding accuracy of our method. We observed that decoding errors are mainly caused by image rasterization. If the input document is in vector graphics format, the decoding result is fully accurate, as we know the glyph outlines precisely. Thus, we evaluate the decoding accuracy with pixel images. We again use *The Lord of The Rings, Chapter 1* to encode a random bit string. We rasterize the resulting document into images with different resolutions, and measure how many letters and blocks can be decoded correctly. Figure 8.2-left shows the testing result.

We also theoretically estimate the decoding robustness of our maximum likelihood decoding method. Suppose the probability of correctly recognizing the glyph of a single letter is a constant  $P_1$ . The probability  $P_2$  of correctly decoding a single 5-letter block can be derived analytically: if we only use Chinese Remainder Decoding algorithm,  $P_2$  is  $\binom{5}{1}P_1^4(1 - P_1)$ . With the maximum likelihood decoding,  $P_2$  becomes  $\binom{5}{2}P_1^3(1 - P_1)^2$ . The improvement is visualized in Fig. 8.2-middle.

The construction of the codebook needs to tradeoff recognition accuracy for the embedded information capacity. This is validated and illustrated in Fig. 8.2-right, where we show the recognition accuracy of a text document printed on a paper with fixed 600dpi. The orange and green curve correspond to two recognizers trained with different setups: for the orange curve, the recognizer is trained using characters printed with 30pt size on paper, and each character in the training data is captured with a resolution of 200px×200px; for the green curve, the recognizer is trained using characters printed with 15pt size, and each character is captured with a resolution of 100px×100px. Not surprisingly, the recognition accuracy drops as we reduce the printing font size. The latter recognizer (green curve) has higher accuracy in comparison to the former (orange curve). But this

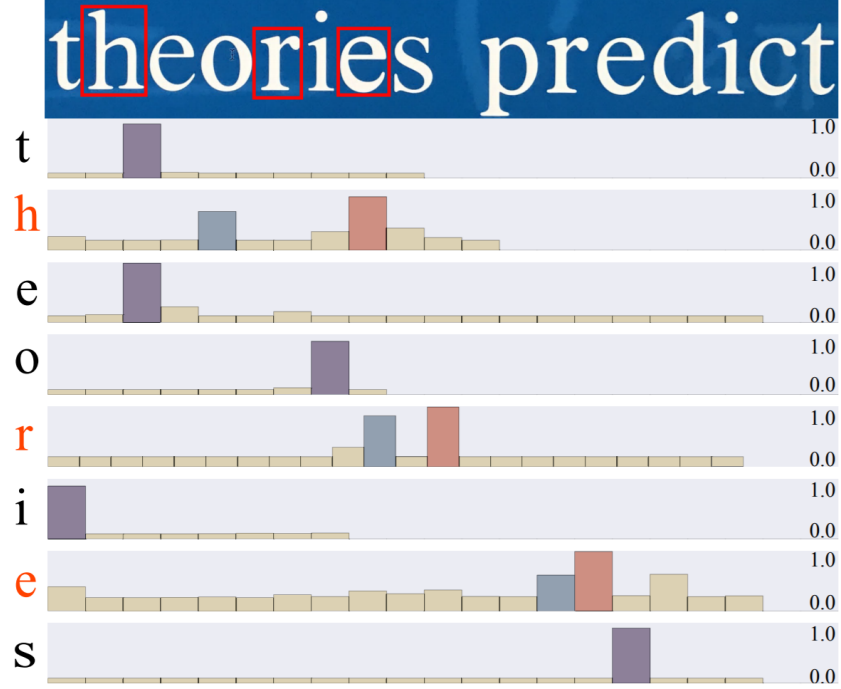


Figure 8.3: **Decoding Probability.** (top) A small region of photo to be decoded, where red boxes indicate recognition errors. (bottom) Each row visualizes the probabilities of recognizing input glyphs (from the image) as the perturbed glyphs. Along the x-axis, every bar corresponds to a perturbed glyph in the codebook. The values on the y-axis are the output from our CNNs. When an error occurs, the blue and red bars indicate the discrepancy between the correctly and incorrectly recognized glyphs.

accuracy improvement comes with a price of the decreased codebook size. For example, for character “g”, its number of perturbed glyphs in the codebook drops from 23 to 14. This indicates that if one needs to embed messages in a document that is expected to print in small fonts, then they needs to construct codebook more conservatively in order to retain the recognition accuracy.

**Performance** We use tensorflow[1] with GPU support to train and decode the input. It takes 0.89 seconds to decode a text sequence with 176 letters (30 blocks). Our encoding algorithm is running in a single thread CPU, taking 7.28 seconds for the same length of letters.

**Error correction improvement** In §5.3, we hypothesize that the probability of recognizing an input pixel glyph  $f$  as a glyph  $u$  in the codebook is proportional to the softmax output of the CNNs. We validated this hypothesis experimentally. Let  $g(u, f)$  denote the softmax output value

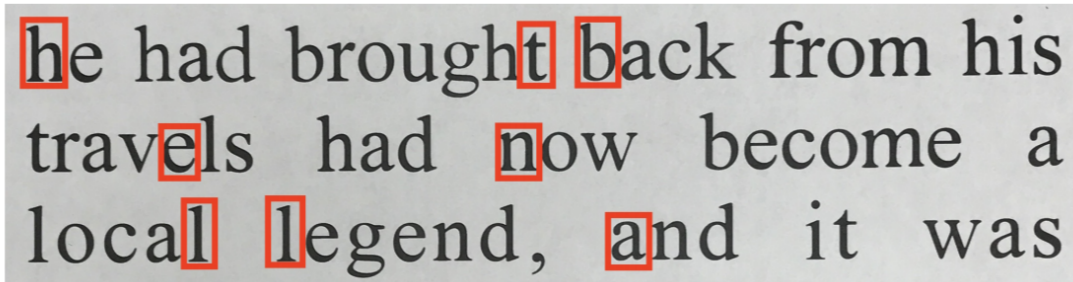


Figure 8.4: **Error correction.** We decode information from an input photo. Red boxes indicate where recognition errors occur. While there exist one block contains two errors, our decoding algorithm still successfully decodes.

of recognizing  $f$  as a perturbed glyph  $u$ . As an example shown in Fig. 8.3, when  $f$  is mistakenly recognized as  $u$  as opposed to its true glyph  $u^*$ , the probability values  $g(u, f)$  and  $g(u^*, f)$  are both high (although  $d(u, f)$  is higher) and close to each other, indicating that  $f$  may be recognized as  $u$  or  $u^*$  with close probabilities. Thus, we conclude that using a likelihood model proportional to  $g(u^*, f)$  is reasonable.

We also extensively tested our decoding algorithm using text document photos under different lighting conditions and various camera perspectives. We verified that it can successfully decode all 5-letter blocks that have at most 2 errors. A small snapshot of our tests is shown in Fig. 8.4.

### 8.3 Perceptual Evaluation

To evaluate the subjective distortion introduced by perturbing the glyphs of a document, we conducted two user studies on MTurk. Both studies follow the standard 2AFC protocol which is described in §6 and has been used in other contexts.

*Study A* assesses the perceptual distortion of a perturbed glyph with respect to a standard font. We prepare a set of paragraphs, and the glyphs of each paragraph are from one of the six categories: (1) the standard Times New Roman; (2-5) the perturbed glyphs from four glyph codebooks, in which the thresholds used to select the perceptually similar glyph candidates are 0.95, 0.85 (i.e., the value we use in §6), 0.75, and 0.65, respectively; and (6) a different font (Helvetica). The font size of each paragraph ranges from 25pt to 60pt, so the number of letters in the paragraphs varies. In each 2AFC question, we present the MTurk rater three short paragraphs: one is in standard

Times New Roman, the other two are randomly chosen from two of the six categories. We ask the rater to select from the latter two paragraphs the one whose font is closest to standard Times New Roman (shown in the first paragraph). We assign 16 questions of this type to each rater, and there were 169 raters participated. An example question is included in the supplemental document.

After collecting the response, we use the same model (6.1) to quantify the perceptual difference of the paragraphs in each of the six categories with respect to the one in standard Times New Roman. In this case,  $s_i$  in (6.1) is the perceptual difference of a category of paragraphs to the paragraphs in standard Times New Roman. As shown in Fig. 8.5, the results suggest that the glyphs in our codebook (generated with a threshold of 0.85) lead to paragraphs that are perceptually close to the paragraphs in original glyphs—much closer than the glyphs selected by a lower threshold but almost as close as the glyphs selected by a higher threshold (i.e., 0.95).

*Study B* assesses how much the use of perturbed glyphs in a paragraph affects the aesthetics of its typeface. We prepare a set of paragraphs whose glyphs are from one of the 12 categories: We consider four different fonts (see Fig. 8.6). For each font, we generate the glyphs in three ways, including (1) the unperturbed standard glyph; (2) the perturbed glyphs from our codebook using a perceptual threshold of 0.85; and (3) the perturbed glyphs from a codebook using a threshold of 0.7. In each 2AFC question, we present the MTurk rater two short paragraphs randomly chosen from two of the 12 categories, and ask the rater to select the paragraph whose typeface is aesthetically more pleasing to them. We assign 16 questions of this type to each rater, and there were 135 participants. An example question is also included in the supplemental document.

Again, using the logistic model (6.1), we quantify the aesthetics of the typeface for paragraphs in the aforementioned three categories of glyphs. Fig. 8.6 shows the results, indicating that, while the aesthetics are different across the four fonts, paragraphs using glyphs from our codebook (a threshold of 0.85) are aesthetically comparable to the paragraphs in standard glyphs, whereas using glyphs selected by a lower perceptual threshold significantly depreciates the typeface aesthetics.

We note that in order to identify untrustworthy raters, in both user studies we include four control questions in each task assigned to the raters, in a way similar to those described in §6.

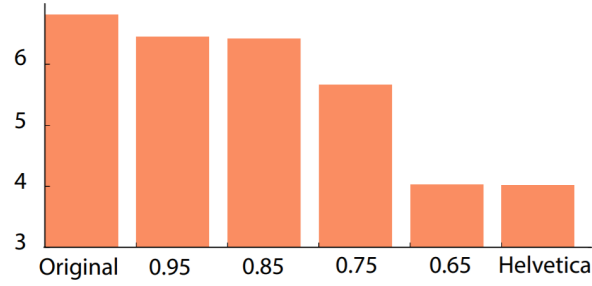


Figure 8.5: **User Study A.** From left to right, the bars correspond to the user study result (see main text for details) for original font glyphs, 0.95, 0.85 (used in our examples), 0.75, 0.65, another different font, respectively.

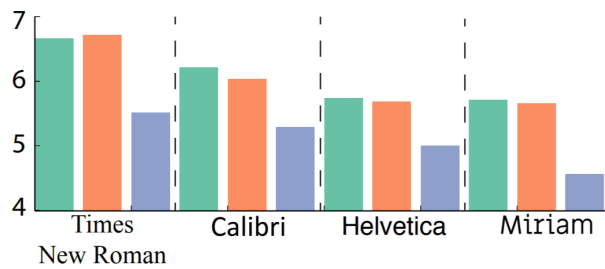


Figure 8.6: **User Study B.** We conduct user studies for four different fonts simultaneously, and learn the scores of aesthetics when different types of glyphs are used: Green bars indicate the original glyphs, orange bars indicate the perturbed glyphs from our codebook, and purple bars indicate the perturbed glyphs chosen by a lower perceptual threshold (i.e., 0.7).

## Section 9: Discussion and Future Work

We have introduced a new technique for embedding additional information in text documents. Provided a user-specified message, our method assigns each text letter an integer and embeds the integer by perturbing the glyph of each letter according to a precomputed codebook. Our method is able to correct a certain number of errors in the decoding stage, through a new error-correction coding scheme built on two algorithmic components: the Chinese Remainder coding and the maximum likelihood decoding. We have demonstrated our method with four applications, including text document metadata storage, unobtrusive optical codes on text, symmetric-key encryption, and format-independent digital signatures.

**Limitations** Currently we only consider standard fonts such as regular Times New Roman, but not their variants such as Times New Roman Bold Italic. But we can treat those variants as independent standard fonts and include their perturbed glyphs in the codebook. Then, our method will work with those font variants.

When extracting messages from a rasterized text document (e.g., a photograph), letters in the document must have sufficient resolution. When printed on paper, the text document must have sufficient large font size for reliable message retrieval. We rely on the OCR library to detect and recognize characters. However, we cannot recover any OCR detection error. If a character is mistakenly recognized by the OCR, the integer embedded in that character is lost, and our error-correction scheme may not be able to recover the plain message since different characters may have different embedding capacities. Nevertheless, in our experiments the OCR library always recognizes characters correctly.

If a part of the text is completely occluded from the camera or contaminated by other inks, the embedded message is lost, as our decoding algorithm needs to know how the text is split into

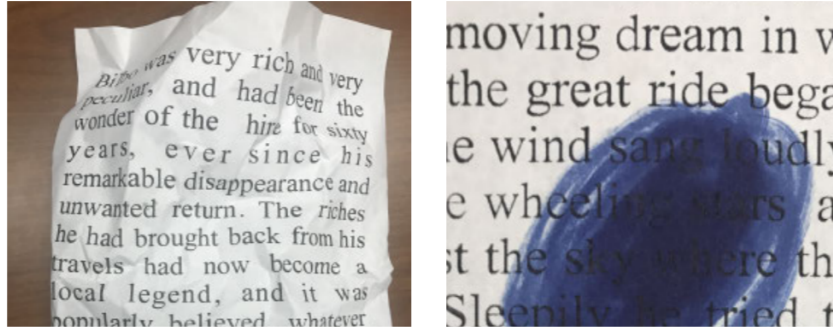


Figure 9.1: **Failure cases.** If a printed text document is heavily crumpled (left) or contaminated by other colors (right), our method will fail to recover the embedded message.

blocks. Similarly, if the document paper is heavily crumpled or attached to a highly curved surface, our method will fail, because our training data was collected on flat surface. Fig. ?? illustrates two cases wherein our method cannot recover the embedded message. In the future, we hope to improve our coding scheme so that it is able to recover from missing letters in the text as well.

In general, the idea of perturbing the glyphs for embedding messages can be applied to any symbolic system, such as other languages, mathematical equations, and music notes. It would be interesting to explore similar embedding methods and their applications for different languages and symbols. Particularly interesting is the extension to logographic languages (such as Chinese), in which the basic element for message embedding is a logogram (i.e., a written character). The number of logograms in a logographic language is much more than the size of English alphabet. For example, the number of commonly used Chinese characters is about 3500. Therefore, it would be expensive to straightforwardly extend our approach to build a codebook of thousands of characters.

Currently, our approach is not optimized to reduce the storage overhead introduced by embedding a hidden message and the codebook. How to compress a vector graphic document with embedded messages is an interesting venue for future work.

Lastly, while our method is robust to format conversion, rasterization, as well as photograph and scan of printed papers, it suffers from the same drawback that almost all text steganographic methods have: if a text document is completely retyped, the embedded information is destroyed.

## **Chapter 2**

# **Vidgets: Modular Mechanical Widgets for Mobile Devices**

## Section 10: Introduction and Related Work

The quest to have the maximum possible screen size and thinnest possible bezel is dominating today’s smartphone market. Both Apple and Samsung removed the Home button and HTC has removed all physical buttons from their latest smartphones. Hardware widgets such as buttons on mobile devices appear to be going extinct.

Yet, there are still many aspects of physical widgets that their digital counterparts have not improved on, or even equalled: Physical widgets give us a reassuring feeling of control, a touching sense of the phone’s orientation, and are provably more efficient for applications such as gaming [90, 32, 162]. Physical widgets also often complement software widgets, in cases where it is just inconvenient for the user to touch the screen—for example, when the user’s hands are wet or covered by gloves, or when the user is playing a game or watching a full-screen video: any finger touching the screen would occlude the displayed content and disrupt the user. Moreover, restricted by the often small touchscreen size, software widgets such as buttons might be too compact to be touched precisely—an issue generally known as the *fat finger problem* [134].

Therefore, a natural question is how to reconcile physical widgets with the current trend of mobile device design. We argue that to utilize physical widgets on modern mobile devices, several desiderata are of importance. **1)** Physical widgets must be easy to manufacture and have a low cost. **2)** Just like the mobile application’s icon button can be dragged and relocated on the screen, physical widgets should be able to be repositioned, added, and/or removed to meet each individual’s preference—a left-handed person will certainly prefer a different user interface layout from a right-handed person. **3)** Physical widgets should have small form factors since smartphones themselves are becoming thinner and lighter. And **4)** they must be power efficient, ideally requiring no power input at all. In other words, they should be completely passive.

In this work, we introduce *Vidgits*, a family of mechanical widgets, specifically a set of push

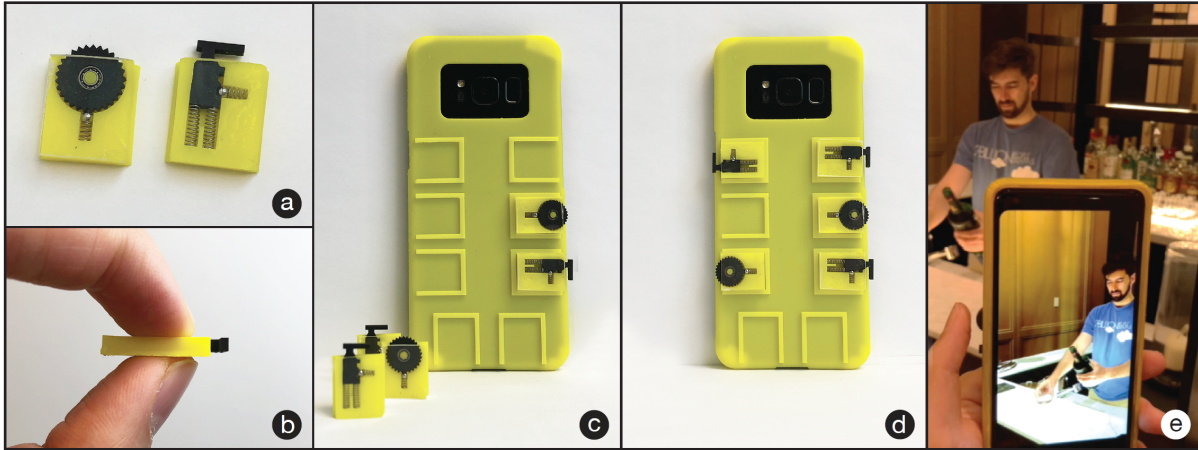


Figure 10.1: **Widgets widgets.** We design a family of physical widgets that are modular, power efficient, (a) compact, and (b) thin. (c) These widgets can be attached to a smartphone’s protective case in a modular fashion. (d) The widgets can be used to construct a wide range of user interfaces and adapt to individual’s preferences. (e) As just one example, here we use a Widgets knob to zoom a camera’s field of view and a Widgets button to trigger the capture. In this way, the user can easily take a photo with a desired focus using a single hand, without needing another hand to pinch on the screen. We demonstrate several more applications in the paper.

buttons and rotary knobs, that satisfies all these desiderata.

Our design of Widgets is based on a key observation. When the user presses a tactile button on a mobile device, the button’s resistance force is nonlinear with respect to the depth it was pressed. The complex interplay between this nonlinear force and the finger’s muscle force causes a small shift of the device. This shift, albeit subtle and transient, can be detected by the accelerometer commonly equipped on mobile devices, and the detected signal is largely shaped by the button’s resistance force profile.

In light of this, we design a set of mechanical buttons and knobs shown in Fig. 10.2. Our basic idea is to use a spring to push a steel ball against a set of teeth that can slide as a button or rotate as a knob. We propose a simple physics-based model to analyze how this design affects its resistance force profile. Gaining insight from this model, we identify a few design parameters that render the user interaction events easily recognizable and the widgets distinguishable from each other. This is achieved through a lightweight computational algorithm that analyzes the accelerometer signals. The use of Widgets is illustrated in Fig. 22.1, featuring a number of attributes:

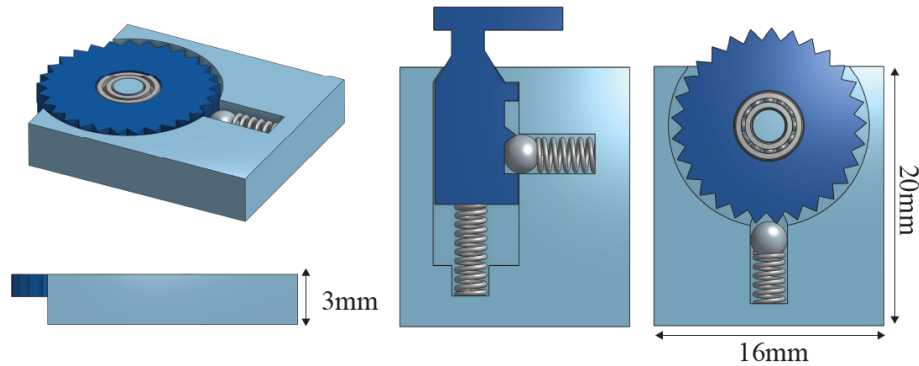


Figure 10.2: **Vidgets design.** The basic design of Vidgets is simple. We use a spring to push a steel ball against a tooth (to make a button shown in the middle) or a circle of teeth (to make a knob shown on the right). For the button, we use an additional spring on the bottom to reset. As we will analyze in §11, this simple setup produces a nonlinear resistance force. By changing the design parameters, we are able to tune the resistance force profiles, and obtain a set of variants for each type of widgets. This design has a small form factor; each Vidgets widget is as large as a finger nail.

- **Low cost.** The widgets are assembled using 3D-printed components and mass-produced springs, balls, and ball bearings, all of which are low-cost. The total cost of each widget is about 70¢ (USD).
- **Small form factor.** The widgets are compact. Our prototype is 20×16×3mm in size, the size of a finger nail.
- **Reconfiguration.** Instead of squeezing the widgets in the already crowded mobile hardware layout, Vidgets can be attached to the *device's protective case*, with no direct touch to the device. Today, about 80% of the smartphone users use protective cases<sup>1</sup>. Thus, the case offers a natural “real estate”, where the widgets can be attached as individual modules. With a number of slots preallocated on the back side of the case (see Fig. 22.1-c), the user can choose which widgets to insert into which slots to customize the user interface layout. If needed, the layout can be reconfigured to adapt to a specific application or individual's preference (see video).
- **Low power consumption.** Consisting of merely mechanical structures, Vidgets require

<sup>1</sup>See the survey published at Statista ([www.statista.com](http://www.statista.com)).

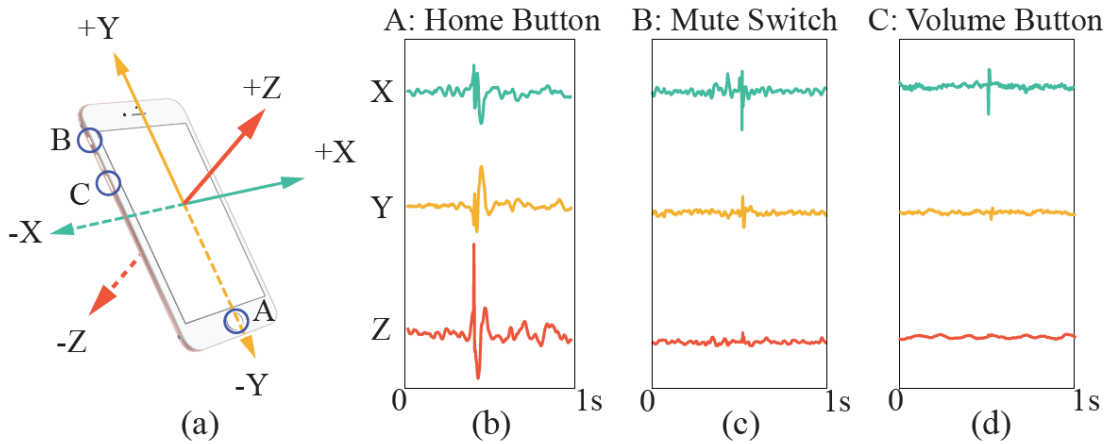


Figure 10.3: **Accelerometer signals.** (a) An accelerometer measures the device acceleration in the device’s local frame of reference. We use it to record X-, Y-, and Z-channel signals when pressing three buttons on the phone (labeled by A, B, and C, and indicated in the circles). (b,c,d) The recorded accelerometer signals along three channels are plotted. These plots show that when different buttons are pressed, the recorded signals differ. Therefore, there is a good chance to recognize button events from the signals.

no power supply nor wired connection to the phone. They rely only on the device’s accelerometer, which is always on at all time [53], and consumes power as low as a few milliwatts—significantly lower than other mobile sensors such as WiFi radios, Bluetooth, and microphones (none of which are always on).

- **Diverse functionality.** Thanks to Vidgets’ modularity, a diverse set of user interfaces can be configured. We demonstrate how Vidgets can be used in several practical scenarios: to enable easier single-handed photo capture, to construct a mobile game controller, as well as to customize a smartphone into a playable music instrument (see §13).

## 10.1 Around-device Interaction

Our work Vidgets is related to an area generally known as around-device interaction [84]. The idea is to extend the interaction possibilities beyond mobile devices and include space around it. Most of the proposed solutions require additional sensors and other hardware attached or connected to a mobile device. For example, HoverFlow [84] uses infrared proximity sensors to track hand and finger positions. SideSight [23] uses an array of infrared sensors and LED lights for tracking

objects in proximity. And ShiftIO [137] utilizes reconfigurable tactile elements attached to the edge of a mobile device to enable physical control and tactile feedback. MagGetz [67], Bianchi et al. [15] and Liang et al. [96] detect the magnetic field generated by a user’s interaction on their customized controllers attached around the device. In general, the assumption (and expectation) is that with time those additional sensors will be miniaturized to fit in the small form factor of a mobile device. Our work makes no such assumption.

Instead of using additional sensors, researchers have also sought to enrich user interaction using sensors that already exist on a mobile device. One idea is to use microphones. Physical objects can be designed to emit specific, detectable sounds. During the interaction, a microphone captures the sounds, and by analyzing the sound pattern or frequency, the mobile device is able to recognize the object [94] and user interactions [60, 130, 108, 158]. However, microphone signals are likely to be contaminated by ambient noise, which can become a prominent issue in a noisy environment. Laput et al. [85] instead used a smartphone speaker to actively emit a sound, and redirected the sound to a microphone through an acoustic wave guide. User interactions with the wave guide can be recognized from the received sound. This approach is more robust against noise, but it has to occupy both the microphone and speaker when it is in use. Moreover, mobile device microphones consume more power than accelerometers and are not always on.

Previous works have also explored the use of accelerometers for user interaction. ViBand [86] customizes the OS kernel of a smartwatch to enable its accelerometer to sample at 4kHz and recognize hand gestures. Unfortunately, kernel customization is not always possible on locked systems such as iOS and most Android devices. We show that the default sampling rate (400Hz) is sufficient for users interacting with Widgets. In addition, Zhang et al. [163, 164] used signals from the accelerometer, gyroscope, and microphone all together to enable additional tapping and sliding inputs. They also relied on a machine-learning based algorithm for the recognition, although it seemed that collecting training data for their method requires a considerable effort.

In comparison to previous works, we focus on user interaction with physical widgets such as buttons and knobs rather than trackpads or gestures. Our approach relies on the on-device

accelerometer only, which is more power efficient. While our method also needs to collect training data for recognizing user interaction, the training process is lightweight: the user only needs to use each widget a few times.

Recently, Piovarči et al. [118] proposed a physics-based data-driven model to understand stylus-surface interaction for designing digital drawing tools. Our work shares a similar spirit of modeling tangible forms, but focuses on buttons and knobs. Our design is also partially inspired by the rich design of mechanical keyboards [112]. While the switch mechanisms (e.g., [59, 30]) used on the keyboard are too bulky to be directly transferred to mobile devices, their nonlinear resistance force profiles pose an intriguing question of how a user’s finger would interact with them. This motivates our design.

## Section 11: Theory of Operation

This section first presents our understanding of why pressing a mechanical button generates accelerometer signals to motivate the design of Vidgets. We then propose a physics-based model to analyze the force profiles of Vidgets widgets, which in turn guides our design choices.

### 11.1 Cause of Subtle Motion

When the user presses a mechanical button or turns a knob, the interplay between the finger's muscle group and widget's mechanical response causes the device to slightly shift. As demonstrated in Fig. 10.3, this subtle motion can be captured by the device's accelerometer. To understand the signal generation process, we start by describing how a mechanical button resists pressing.

### 11.2 Resistance Force of a Tactile Button

Among numerous designs of mechanical buttons, one type of buttons, called *tactile button*, is of particular relevance to our Vidgets design. Widely used in keyboards, tactile button has a nonlinear force response curve with respect to the pressing depth (see Fig. 11.1). When the button is being pressed, its resistance force increases nearly linearly, until a certain pressing depth is reached (e.g., typically 1.5~2mm for keyboard). At that point, referred as the *actuation point*, the resistance force drops dramatically. Historically, the force profile at the actuation point is designed for fast keyboard typing, allowing the user to quickly switch keys without pressing the key all the way down [30, 78, 122]. It turns out that similar force profiles also play a key role for the generation of accelerometer signals that we will harness, as elaborated next.

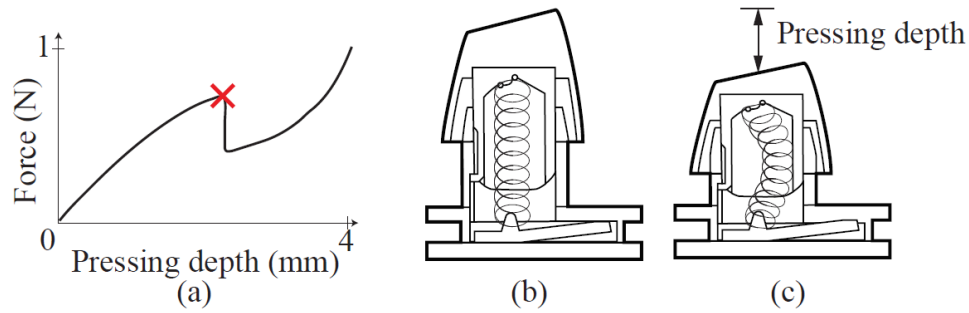


Figure 11.1: **Nonlinear mechanical switch.** Many mechanical switch has a nonlinear force response. Here we illustrate the force nonlinearity using the IBM’s buckling spring switch (b), a classic switch design historically used on IBM’s Model M keyboard. Its force response curve is qualitatively plotted in (a). As the switch is being pressed, the resistance force increases until the actuation point (indicated by the red cross mark) is reached. At that point, the resistance force drops dramatically. This force nonlinearity inspires our design of Vidgets.

### 11.3 Generation of Accelerometer Signal

The nonlinear force response does not by itself produce detectable signals for the accelerometer, although one might imagine that pressing a button may always cause elastic vibrations (or waves) to propagate through the device body to the accelerometer. We confirm that the internal vibration is not the cause of accelerometer signals through experiments: we press a smartphone button while clamping the phone firmly on a table. In this case, there are still elastic vibrations, but no signal is detected by the accelerometer. The signal emerges only when one presses the button while holding the phone in the hand.

Moreover, notice in Fig. 10.3 that the accelerometer measures accelerations along X-, Y-, and Z-direction (in the phone’s local frame of reference) separately. We find that the three channels of signals depend only on the button’s pressing direction, but not the orientation of the widget’s internal mechanism. For example, if we horizontally flip the button mechanism in Fig. 10.2-a such that the steel ball pushes against the button from a different side, the resulting signal remains largely unchanged.

In fact, it is the interplay between the user’s fingers and the widget’s force response that produces the accelerometer signals. As an example, consider a phone hold by a user’s hand (see Fig. 11.2-a). Suppose that the user’s thumb is pressing a button with a force  $F_p$  while other fingers

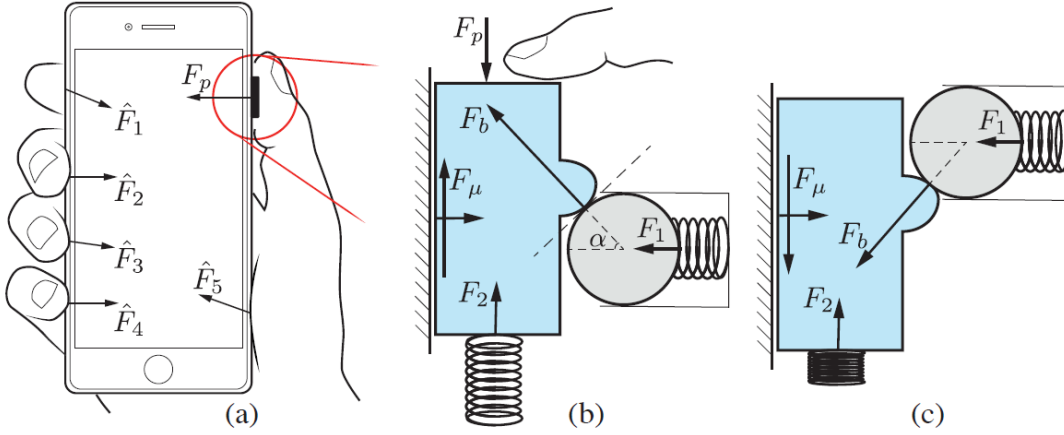


Figure 11.2: **Force analysis.** (a) When a phone is held by a hand and a button (in the red circle) is pressed, a set of forces are applied on the phone to keep the phone almost still. The pressing force  $F_p$  from the thumb is exerted to the button's switch mechanism shown in (b) (after a  $90^\circ$  rotation), where the notation will be used in the main text. (c) After the button is pressed down, the spring on the bottom pushes the button up to reset. At this point,  $F_2$  needs to be sufficiently large to overcome the resistance from  $F_b$ .

and the palm provide support, and the support forces are denoted by  $\hat{F}_1, \dots, \hat{F}_5$ . If the button is pressed slowly (or quasi-statically), then  $F_p$  needs to balance against the button's resistance force  $F_b$ . Meanwhile, to keep the phone still, the net force on the phone needs to be nearly zero. Thus, in this case we have the relationship,  $F_b \approx -F_p \approx \sum_{i=1}^5 \hat{F}_i$ . As the button is being pressed deeper, this relationship remains satisfied until the actuation point is reached (recall Fig. 11.1). At that point,  $F_b$  drops immediately, but it takes a longer time for the finger muscles to adjust and rebalance against  $F_b$ . For example, in our estimation (described in §11.4) it takes about 10ms for  $F_b$  to drop, while our finger's response time to tactile stimuli is on the order of 100ms [110]. During the response time, the total force on the phone is imbalanced, causing the phone to accelerate (and decelerate). This acceleration, although in a very short time, is detected by the accelerometer (see Fig. 11.3).

*Remark.* In a real scenario, our hand will never hold a phone completely still. As a result, the signal recorded by the accelerometer also includes the acceleration (and deceleration) of the hand and body locomotion such as walking. We note that the acceleration due to body locomotion occurs at a time scale much larger than the finger-button interaction. The acceleration signal we

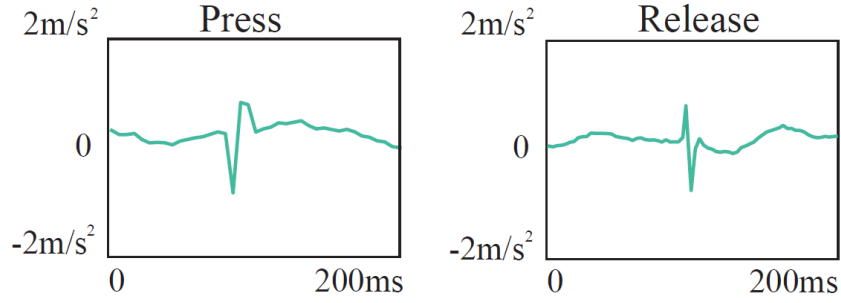


Figure 11.3: **A close look** of the X-axis signal when pressing and releasing the volume button on a smartphone. On the left is the signal generated by pressing the button down (downstroke), and on the right is the signal from releasing the button (upstroke). Notice the short durations of these signals, typically under 50ms. It is also worth noting that the two signals are opposite with respect to each other, in the sense that downstrokes begin with a negative acceleration, while upstroke begins with a positive acceleration—a feature useful for distinguish downstrokes from upstrokes.

are interested in happens within less than 50ms (Fig. 11.3-b,c). Therefore, a simple high-pass filter can separate it from the signals due to locomotion. This detail will be discussed later in §14.3.

#### 11.4 Physics-Based Force Model

We now introduce a physics-based model that describes the resistance force profile of our Widgets designs. This model is not meant to predict the accelerometer signal. After all, it is very challenging, if not impossible, to accurately model the finger’s muscle reaction to the widget resistance. Rather, our goal is to use this model to understand the widget force profile near the actuation point and inform our design choices.

Consider the setup and notation shown in Fig. 11.2-b, which also depicts the Widgets button design in Fig. 10.2-b. Let  $k_1$  and  $k_2$  be the coefficients of springs on the side and bottom, respectively. Note that if the bottom spring vanishes ( $k_2 = 0$ ), the same diagram reflects the Widgets knob design showing in Fig. 10.2-c (after rotated by  $90^\circ$ ). Therefore, the force model derived here can be applied to both types of widgets. When the widget is in a quasi-static state, its net force must be zero, meaning

$$F_p + F_2 + F_b \sin \alpha + \tau \cdot \mu \cdot F_b \cos \alpha = 0, \quad (11.1)$$

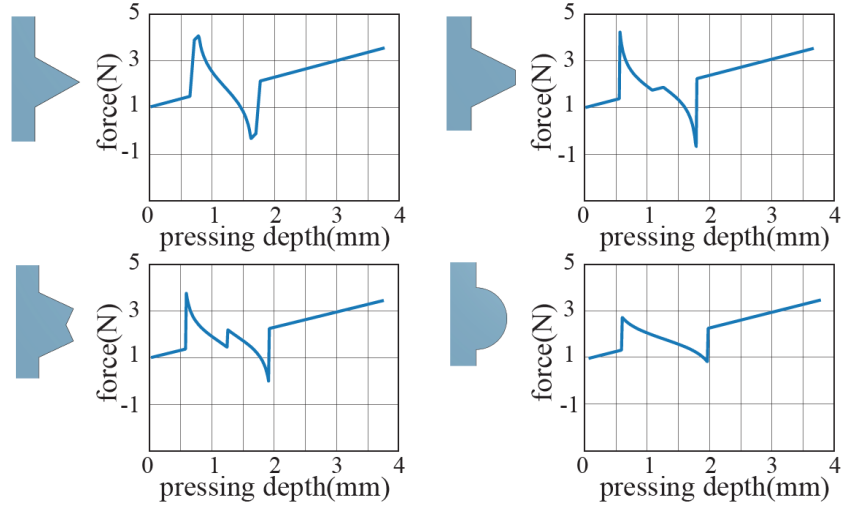


Figure 11.4: **The force profile zoo.** Here we show four different designs of the tooth shape used in Vidgets buttons (Fig. 10.2-b) and their estimated resistance force profiles. As we vary the tooth shape, the force profile changes considerably.

where  $F_p$  is the pressing force exerted by the user,  $F_2$  is the force from the bottom spring,  $F_b$  is the contact force between the ball and slider, and  $\alpha$  is the contact angle (see Fig. 11.2 for the notation). In addition,  $\mu$  denotes the Coulomb friction coefficient between the slider and the container, and  $\tau$  indicates the sliding direction:  $\tau = -1$  when the button is being pressed, and  $\tau = 1$  when it is being released. Here we neglect the frictional force between the ball and slider, as the mass-produced steel ball is sufficiently smooth. Meanwhile, the force balance of the ball is written as

$$F_1 + F_b \cos \alpha = 0, \quad (11.2)$$

where  $F_1$  is the force from side spring. Combining (11.1) and (11.2) yields an expression of the widget's resistance force  $F_r$ , namely,

$$F_r = -F_p = F_2 - F_1 \tan \alpha - \tau \cdot \mu \cdot F_1. \quad (11.3)$$

The widget's force profile  $F_r(d)$  is the resistance force  $F_r$  changing with respect to the pressing depth  $d$ . It can be evaluated using (11.3) because  $F_1$ ,  $F_2$ , and  $\alpha$  all depend on  $d$ . In particular,  $F_2(d) = k_2 \cdot d$  if we let  $d = 0$  correspond to the rest state of the bottom spring.  $F_1$  and  $\alpha$  are

evaluated as follows. Suppose that the boundary of the slider (when not pressed) is given by the parameterization  $(x(t), y(t))$ ,  $t \in [0, 1)$ . If the slider is pressed down by a distance  $d$ , then the ball-slider contact point  $(x(t_0), y(t_0))$ , parameterized by  $t_0$ , can be computed by solving the following system of equations:

$$\begin{aligned} [x_c - x(t_0)]^2 + [y_c - y(t_0) + d]^2 &= R^2, \\ x'(t_0) [x_c - x(t_0)] + y'(t_0) [y_c - y(t_0) + d] &= 0. \end{aligned} \tag{11.4}$$

The first equation constrains the contact point at  $(x(t_0), y(t_0))$ , while the second equation requires the tangent direction of the ball to align with that of the slider at the contact point. Here the Y-coordinate of the ball,  $y_c$ , is fixed, as it moves horizontally. Therefore, solving (11.4) yields the values of  $t_0$  and the ball's X-coordinate  $x_c$ . Finally, we obtain  $\alpha = \arctan \frac{y(t_0) - y_c}{x(t_0) - x_c}$  and  $F_1 = k_1 \cdot (x(t_0) - L_0)$ , where  $L_0$  is the rest length of the side spring. Note that since the equations in (11.4) depend on the pressing depth  $d$ , both  $\alpha$  and  $F_1$  are functions of  $d$ , and so is  $F_r$ .

## 11.5 Widget Design

The physics-based model allows us to estimate the widget's force profile of a specific design. For instance, an important design choice is the shape of the teeth (to provide resistance against the push from the ball) on the button and the knob (Fig. 10.2). Figure 11.4 shows force profiles of four teeth shapes. By changing the teeth's shape, we are able to tune the position of the actuation point as well as how quickly the resistance force drops after the actuation point—the main cause of acceleration signals (as discussed in §11.3).

The force profile at the actuation point directly affects the accelerometer signals when we press the button or rotate the knob. For example, the four tooth shapes in Fig. 11.4 leads to different acceleration signals shown in Fig. 11.5. The correlation between the estimated actuation points and the recorded signals (shown in Fig. 11.5) is evidence that our force model is able to inform the performance of the widgets. Those force profiles also contrast starkly to the case shown in Fig. 11.6-a, where the button has no teeth at all. In that case, the actuation point vanishes and the

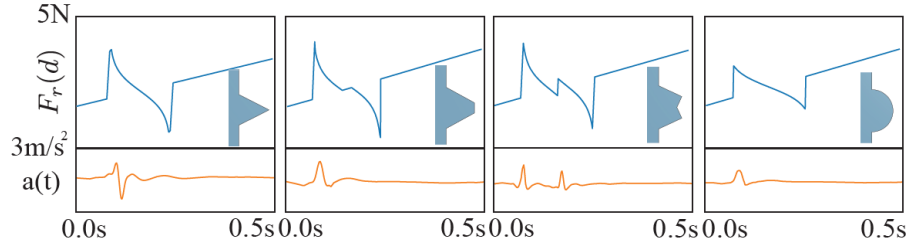


Figure 11.5: **Force profile vs. accelerometer signals.** Here we show the force profiles of four tooth shapes (the same as Fig. 11.4) and their accelerometer signals when these tooth shapes are used in a Vidgets button. Notice the correlation and temporal alignment between the estimated actuation points of the force profiles and the recorded signals. For example, a quicker force drop at an actuation point results in a stronger accelerometer signal, and dual force drops result in two peaks in the signal.

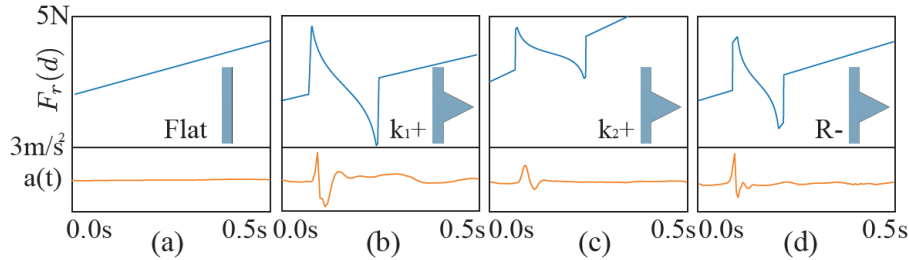


Figure 11.6: **Force profile dependence on design parameters.** The top row shows the estimated force profiles when various designs are used in the button mechanism. The bottom row shows the corresponding accelerometer signals when those designs are used. (a) With no teeth, the force is linear, and no signal is detected by the accelerometer. In (b), (c), and (d), we use the same tooth shape as shown in the first plot of Fig. 11.4, but change other design parameters, including an increase of side spring ( $k_1+$ ) and bottom spring ( $k_2+$ ) as well as a reduction of the steel ball radius ( $R-$ ).

accelerometer captures no signal.

**Design choices** A few factors are crucial to the widget's performance. First, we wish to receive strong accelerometer signals when the widget is in use, because the signals are likely contaminated by noise. This can be achieved by deepening the teeth or strengthening the side spring, as predicted by our force model and confirmed by the recorded signals (Fig. 11.5 and Fig. 11.6).

It is also interesting to note that the effect of the bottom spring is counter-intuitive. Intuitively, one might expect a stronger bottom string to produce a stronger signal. However, our force model predicts that a stronger bottom spring tends to flatten the force profile near the actuation point, therefore weakening the accelerometer signals. This prediction is also confirmed by our recorded

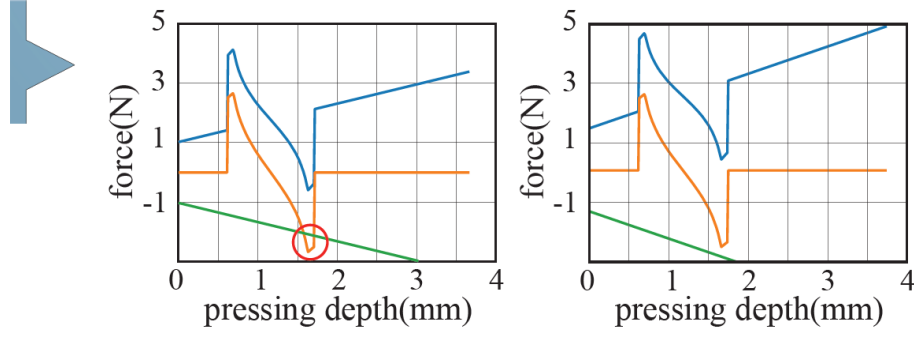


Figure 11.7: **Button reset.** Given a tooth shape shown on the left, blue curves are the estimated force profile  $F_r(d)$ . The orange curves indicate the vertical component  $F_b \sin \alpha$  of the force exerted by the ball, and the green curves indicate the negated force from the bottom spring (i.e.,  $-F_2$ ). (Left) If  $F_2$  is insufficient, the green curve intersects with the curve of  $F_b \sin \alpha$  (orange curve), meaning the button can not reset. (Right) To ensure the button reset, we must increase the spring coefficient  $k_2$  to eliminate the intersection between orange and green curves.

signals (Fig. 11.6-c).

Equally important is guaranteeing that the button resets itself when it is released. This is illustrated in Fig. 11.7, wherein the orange curve indicates the vertical force (i.e.,  $F_b \sin \alpha$ ) exerted by the steel ball (shown in Fig. 11.2). After the button is pressed down, the steel ball tends to block it from resetting, since at that point  $F_b \sin \alpha$  is downward (Fig. 11.2-c). The green curve in Fig. 11.7 shows the bottom spring's force. If it intersects with the orange curve (Fig. 11.7-left), the spring force will not be sufficient to overcome the blocking force, and as a result, the button will be stuck by the ball. Therefore, our force model also helps check if a particular design is mechanically sound.

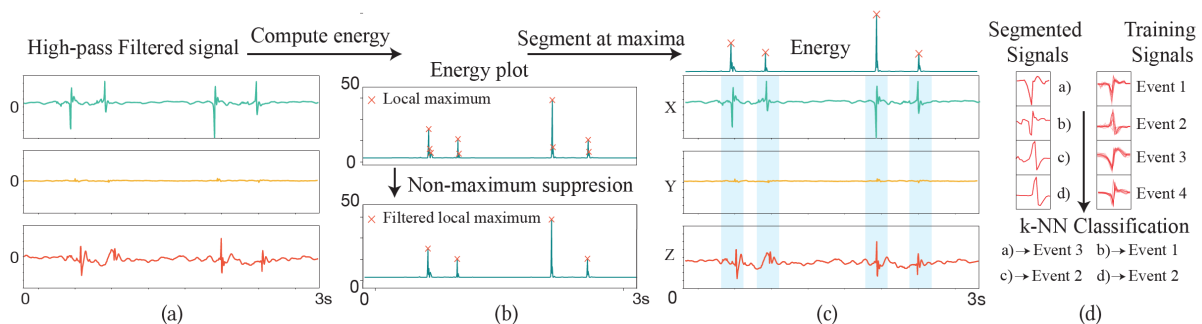
Through these analyses, we identify four distinct tooth shapes that result in different force profiles (Fig. 11.4). The difference among these force profiles in turn leads to disparate signals captured by the accelerometer when they are in use. We appreciate the difference because it makes the widgets easily distinguishable from each other using the signals, especially when they are located near each other in the phone's case. We will discuss the runtime widget recognition in §12.2.

## Section 12: Signal Processing at Runtime

Vidgits' design allows us to process smartphones' accelerometer signals in a simple manner. This section presents a lightweight signal processing algorithm that runs in real time on mobile devices. The goal is to identify which user interaction events (i.e., downstroke, upstroke, or knob rotation) triggered, and on which widget they are triggered on.

On most mobile devices, the accelerometer samples at a rate of hundreds of hertz. For example, the Android device that we use in our examples works at 400Hz. Our algorithm runs at 25 Hz, although a higher rate is also possible. Each time it runs, it loads the accelerometer's samples from the past 0.5sec and detects user interaction events from those samples.

The detection algorithm has two steps, summarized in Fig. 16.2. The first step identifies time regions in which Vidgits usage events may occur. The second step then analyzes each time region to recognize the event type and the widget that produced the event. For each widget, we also maintain a timestamp that indicates the time of the most recent event produced by that widget. These timestamps will be used to cull events in the current time window but already detected in previous runs. We now elaborate the two steps.



**Figure 12.1: Overview of runtime detection.** Our detection algorithm takes as input the raw accelerometer signals. After applying a high-pass filter (a), the signals are processed through non-maxima suppression (b) and segmented at the local maximum (c). The segmented signal is then classified using k-NN classification to determine what Vidgits interaction event, if any, occurs (d).

## 12.1 Segmentation

For each time window, the accelerometer supplies three channels of signals, corresponding to the accelerations along X-, Y-, and Z-axes. The output of this step is a set of time regions, called *event regions*, wherein Widgets usage events may happen. Each event region has a length of 50ms. We choose this time length according to [77], which reports that the typical time duration for pressing a mechanical keyboard is 50~300ms. We use the lower bound for the finest time granularity.

First, we apply a high-pass filter to the three channels, respectively, removing the signal components lower than 20Hz. This is because each output event region (of 50ms length) resolves signal components higher than 20Hz, and this filter effectively removes acceleration components introduced by other motions such as the human body locomotion.

Next, we estimate the total power  $P^i = \sum_j (x_j^i)^2$  of each channel  $i$ , where  $j$  indexes the individual samples. We then choose the channel with the highest power (or the largest signal-to-noise ratio) to analyze. This is motivated by the fact that the user's interaction with the widget almost always generates a dominant signal along one direction—for example, the pressing direction for a button or the sliding direction for a knob. We denote the samples of that channel as  $\tilde{x}_j$ .

Then, we identify event regions. Inspired by a commonly used object recognition technique in computer vision [46], we apply non-maxima suppression to the samples  $\tilde{x}_j$ . Concretely, we locate all the local maxima of the power samples (i.e.,  $\tilde{x}_j^2$ ) over time, and include their time indices in the set  $\mathcal{T} = \{t_i\}_{i=1}^K$ . Next, we iteratively choose a time  $t$  from  $\mathcal{T}$  and include the time region  $[t - \Delta, t + \Delta]$  as one of the output event regions, where  $\Delta$  is set to be 25ms. In each iteration, we choose from the current set  $\mathcal{T}$  the time that has the highest power sample (i.e.,  $t = \arg \max_{t \in \mathcal{T}} \tilde{x}_t^2$ ), and remove all the time points that are in the range of  $[t - \Delta, t + \Delta]$  from  $\mathcal{T}$ . The output of this process is a number of event regions.

## 12.2 Event Detection

Next, we recognize which type of event, if any, occurs in each event region. The event could be a press or release of a button, or a (single tooth “tick”) rotation of a knob toward either of the two rotational directions. If any of these events happen, we also need to identify the specific widget that produces it. An detected event region could also be a false positive in which no event occurs. Our goal here is to recognize all these possibilities.

As discussed in §11.1, a mechanical widget produces accelerometer signals through the user’s finger muscles reacting to the widget’s mechanical response. In general, the signal generation depends on how the mobile device is held, where the widgets are, and what type of widget are in use. Our analysis in §11 is to design the widgets such that their resulting signals are easily recognizable. Indeed, a lightweight recognition algorithm suffices for the recognition.

**Training** After the widgets are mounted on the device (e.g., on its protective case), they are calibrated through a simple training process. In this process, the user is asked to use each widget five times in the poses they are comfortable with. Each use is guided by the system (e.g., a mobile application) so it knows what event is expected to happen. For each use, the system identifies the event region (as described above in the segmentation step), which includes 20 samples (400Hz in 50ms) in three accelerometer channels. This process results in a  $20 \times 3$  vector for each training event. Each vector is associated with the event type and a widget ID and is stored for runtime use.

**Recognition** At runtime, after we identify the event regions from the segmentation step, we recognize each event region by performing standard k-nearest neighbor (k-NN) classification [109]. This algorithm computes the classic Dynamic Time Warping (DTW) similarity [13] between the samples in each event region and the pre-stored calibration vectors, and classifies the region via a majority vote. The DTW similarity is more robust than  $L_2$  distance for comparing two time series signals. Even if one signal (e.g., the signal in a event region) is time shifted or scaled, the DTW distance remains invariant whereas the  $L_2$  distance changes significantly. If the minimum similar-

ity between the event region samples and all calibration vectors is larger than a threshold, we treat that event region as a false positive and ignore it. The choice of the threshold and its validation will be discussed later in §14.3. We will show that this simple recognition algorithm runs in real time on mobile devices while providing a high recognition accuracy.

**Remark** Even if two widgets are exactly the same, as long as they are located at different positions on the phone case, our k-NN recognition algorithm is still able to distinguish them. This is because for widgets at various locations, we tend to use different fingers or poses to interact with them, that is, the muscle forces used in the interactions are different. If the two widgets are closely located (e.g., next to each other), we have to rely on different widget designs to make sure they generate distinguishable signals. For this purpose, the various tooth shapes (such as those in Fig. 11.4) are particularly helpful. Thereby, the various designs of Widgets widgets ensure that they are always recognizable regardless of their intended layout on the phone case.

## Section 13: Applications

Vidgits are building blocks of a diverse set of user interfaces and thereby offer some unique advantages. To bring Vidgits into concrete use scenarios, consider, for example, a simple case wherein the user is wearing a glove, and thus can not interact with the touchscreen. A Vidgits interface is able to save the user from this hassle, allowing them to interact with the device without taking off the glove (see video). In what follows, we demonstrate three concrete applications: photo capture, gaming, and music playing. Throughout, we will discuss the features brought by Vidgits.

**Zoom-in/out of photo capture.** When capturing a photo or video with a smartphone, oftentimes we need to zoom the camera's field of view in or out to focus on a target scene. Perhaps ironically, in most camera applications such a simple and common task requires two hands to accomplish: one hand holds the phone while two fingers from the other hand pinch on the screen to zoom in or out. This becomes rather inconvenient if one hand is occupied—think about trying to capture a photo while holding a cup of coffee in another hand.

By attaching a push button and a rotary knob on the phone's protective case, as in Fig. 13.1, we are able to construct an one-handed interface for zooming and capturing photos. We place a knob near the thumb to zoom in and out and a button to trigger the capture.

Remarkably, this interface can easily accommodate left-handed and right-handed users. Thanks to the Vidgits' modularity, one can reposition the button and knob on the phone case to fit their personal preference, as shown in the video.

*Discussion.* Although Vidgits button will cause a subtle shift of the phone when taking photos, no negative impact is observed on the captured photo quality. We believe that this is due to short time scale of the shift and the help of optical image stabilization system already existed in most of



Figure 13.1: **Camera zoom.** When taking a photo or video, the user can zoom the field of view in and out using a Vidgets knob (in white box) and trigger the shot using a Vidgets button (in white circle), all with single hand. In contrast, most of the current smartphone cameras require two hands to zoom in/out (with the second hand’s fingers pinching the screen), which is not always feasible. In addition, the knob and button can be easily repositioned to accommodate both left-handed and right-handed users (see video).

the smartphones on the market. After all, handshake and subtle motions are almost unavoidable even when we capture a photon with the touchscreen button.

**Mobile game controller** Another important application of Vidgets is mobile gaming. Currently, most mobile games rely on graphical user interfaces (GUIs) such as a set of software buttons showing on the screen to receive user input (e.g., shooting bullets). Yet, the GUI for mobile gaming suffers from a few limitations. Not only does it take a part of the already limited screen space away from displaying interesting visual content, certain areas of the screen will also be occluded whenever the user touches the GUI. As the game becomes intense, the player’s hand sweats, and the wetness further deteriorates the sensitivity of the touchscreen. Indeed, it is these reasons that motivate some companies to make *mobile trigger buttons*<sup>1</sup>, a joystick-like accessory that can be clipped on the mobile screen for game control, although they are generally bulky, still consume some screen space, and support only a particular set of games and smartphone models.

<sup>1</sup>For example, see the products from this [Amazon link](#).



Figure 13.2: **Mobile gaming.** (top) When controlling a game on a mobile device, the player’s fingers often occlude the display (e.g., in red circles) and cause disruption. (down) Vidgets interface (in white circles) enables the player to control game more efficiently without occluding the screen.

Using Vidgets, we can easily assemble a mobile game controller that adapts to a specific game and provide a more natural way of control, without sacrificing any screen space (see Fig. ?? and the video).

**Virtual instrument.** Wang [147] demonstrated the concept of “The iPhone’s Magic Flute”, aiming to re-imagine an ancient acoustic instrument, flute, in the context of modern mobile technology for expressive music-making. Their demonstration is truly impressive, while their flute interface is a GUI displayed on the screen. Motivated by previous studies showing that tactile feedback provides more efficient handheld input than touchscreens [162, 21, 65], we extend Wang’s concept and introduce the “smartphone’s saxophone”, which has tactile keys made of Vidgets buttons. Different combinations of the keys are mapped to produce different music notes, allowing the user to play a melody.

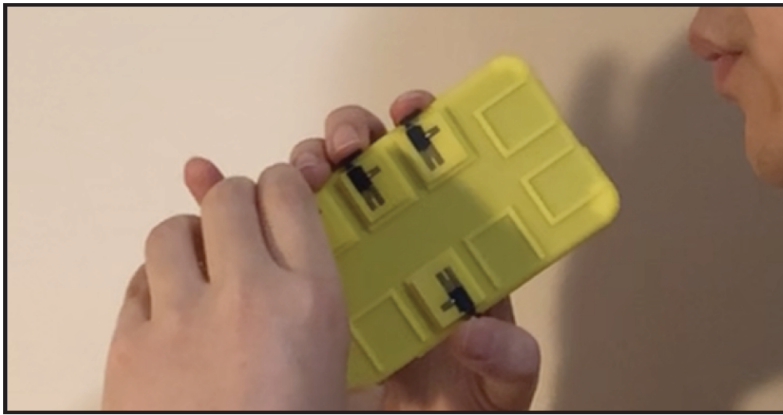


Figure 13.3: **Saxophone app.** We demonstrate a 4-key “smartphone’s saxophone”. Each Vidgets button serves as a saxophone key. By detecting the downstroke and upstroke of each Vidgets button, this smartphone application can recognize which keys (and key combinations) are pressed and play the corresponding music notes.

## Section 14: Evaluation

We now report our experiments toward understanding Vidgets’ accuracy, robustness, and system responsiveness. We also discuss some rules of thumb that affect Vidgets’ robustness.

**Hardware** All our experiments are conducted on a Samsung Galaxy S8 smartphone, which equips a STMicroelectronics LSM6DSL inertial measurement unit. The same series of accelerometers has been widely used in many other popular smartphones and wearable devices, including the Google Pixel 2/2XL, Samsung Gear S3, Samsung Galaxy Note9, and others. The maximum sampling rate obtained through the Android API is 400Hz [53].

### 14.1 Accuracy

**Participants** To understand the accuracy of using Vidgets widgets, we recruited 13 participants, including six females. Their mean age is 24.3, and two are left-handed.

**Procedure** We use an 8-widget layout on the smartphone case, as shown in Fig. 14.1-left. This layout includes six push buttons and two rotary widgets. Each widget can produce two events: downstroke/upstroke for the button widgets and forward/backward rotation for the knob widgets. In total, there are  $8 \times 2 = 16$  distinct user interaction events in this layout.

At the beginning of the study, we asked each participant to perform a calibration procedure involving five pushes for each button (which generate five downstrokes and five upstrokes) and five forward and backward rotations for each knob. The signals collected in this step (after segmentation) are used by our k-NN user interaction event classification.

Afterwards, we ask the participants to press each button around ten times and rotate each knob around ten times in each direction. These interactions are performed in a random order,

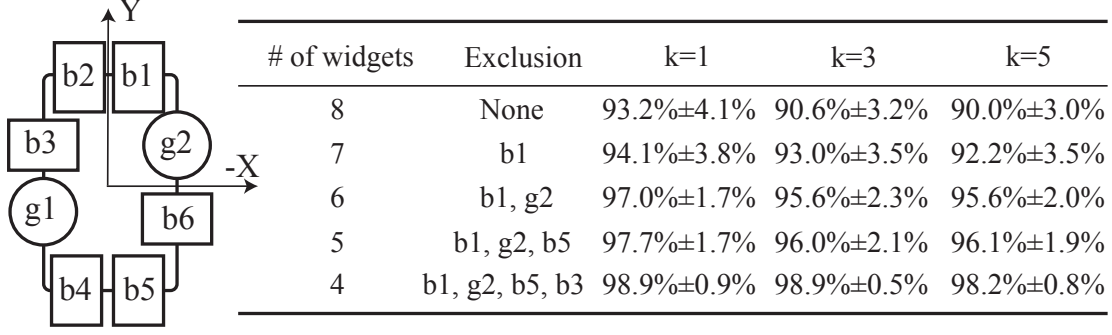


Figure 14.1: **(left)** We use eight widgets, including six buttons and two knobs, on a smartphone case. Their relative positions are indicated by boxes (for buttons) and circles (for knobs). We use ‘b#’ and ‘g#’ (where # is a number) as the widget IDs. **(right)** We report the recognition accuracy when using different  $k$  values in the k-NN algorithm. In general,  $k = 1$  yields higher accuracy than higher  $k$  values. On each row from top to bottom, we report the accuracy when increasingly more widgets are disabled (i.e., fewer widgets are in use). The second column indicates the widgets that are excluded in each test. As fewer widgets are used, the recognition accuracy increases.

and each participant repeats these tasks for three runs. We do not restrict how the participants hold the phone—we simply ask them to use the poses they feel most comfortable with and keep them consistent across the three runs. The collected signals are then processed by our k-NN-based recognition algorithm, and the reported events are compared with the true events to evaluate the algorithm’s recognition accuracy (reported in Fig. 14.1). We also evaluated the accuracy when choosing different k-NN parameters, and found that  $k = 1$  in the k-NN algorithm yields higher accuracy than other  $k$  values. The confusion matrix when  $k = 1$  is reported in Fig. 14.5.

Another trend that we observed is that as we use fewer widgets, the widget events become easier to recognize. We note that in most cases, such as those described in §13, we need only up to four widgets, meaning that we are able to achieve an accuracy above 98%. In general, our recognition accuracy seems higher than previously reported passive interaction widgets [86, 130], although the underlying operation principles are different.

**Other classification algorithms** We tested other classification algorithms including support vector machine (SVM) and neural networks. We found that, given the limited training dataset (five trials per widget event), both the SVM and neural network have worse performance than k-NN classification. If we increase the size of the training dataset to 30 trials per widget event, the ac-

curacy of SVM becomes comparable to k-NN, and so does the neural network’s accuracy if we tune its hyperparameter values carefully. Since limiting the required amount of user calibration is crucial for the usefulness of the widgets in practice, we conclude that the k-NN algorithm is the most suitable one for detecting Vidgets usage events.

**Discussion** Our 8-widgets evaluation is to 1) demonstrate that many widgets can be used simultaneously on a phone case, and 2) show the trend that the accuracy increases when fewer widgets are used.

It is worth noting that although we record the data first and then run the processing algorithm, the processing algorithm is exactly the same as what we run in our realtime demo (i.e., we used the same algorithm to process the same data). In other words, the results have no difference from what we get on the device in realtime. By using this test scheme, we can easily collect accuracy statistics from multiple users and multiple use cases within a single device.

The performance of certain interaction scenarios can be inferred from our experiment in Fig15. For example, the camera app uses g1, g2, b3, and b6. We can account for the accuracy of these widgets and ignore data from others, and get a 98.2% accuracy.

## 14.2 Response Time

We also measured the responsiveness of our system on a Samsung Galaxy S8 phone. The measurement is based on the time delay from the beginning of a Vidgets usage event to the point at which the event is detected by our signal processing algorithm. We estimate the event’s start time using the starting time of the segmented event region (i.e.,  $t - \Delta$  in §??). Note that this is a conservative estimation, as the true starting time of an event is always later than that value. We found that the time delay is less than 56ms in average, far below the well-known 100ms limit for people recognizing a system as acting instantaneously [106, 26].

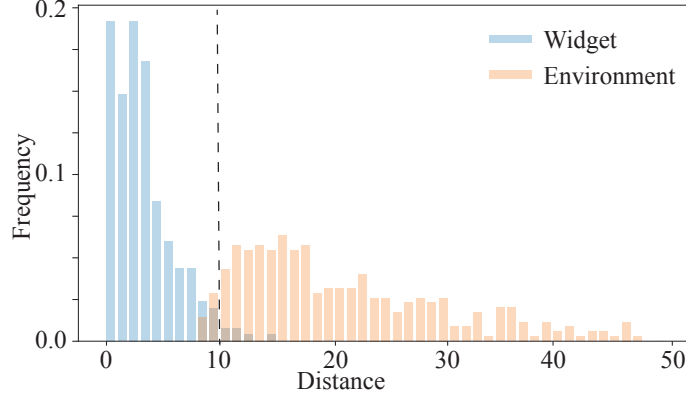


Figure 14.2: **Eliminating false events.** We collect signals generated by other events from the daily life (e.g., walking, taking the subway, and gaming), and plot (in red) the distribution of k-NN distances between those events and the calibration vectors. In comparison, we also plot (in blue) the distribution of k-NN distances between true widget events and the calibration vectors. It is evident that the two distributions are well separated. Thus, our algorithm is able to distinguish the widget events from other random events.

### 14.3 Robustness

Body locomotion and environmental vibration such as car motion also generate accelerometer signals. As discussed in §12.2, we avoid detecting such false positive events by enforcing a distance (i.e., DTW similarity) threshold in the k-NN feature space. If the signal from a segmented event region is not within this threshold distance from any of the training data signal, we discard that event region. To justify the choice of the threshold, we ask participants to record accelerometer signals in 30 minutes of their daily life, including the scenarios of walking, taking a bus or subway, and playing a mobile game using the touchscreen. As shown in Fig. 14.4, accelerometer signals generated by these daily life activities all have distinct characteristics from those Vidgets usage events.

Moreover, given the segmented event regions from those unrelated daily events, we plot the distribution of their k-NN distances (i.e., the minimum DTW distance between the signal in the event region and all the calibration vectors) in Fig. 14.2, together with the distance distribution of true widget event regions. It shows that the distance distribution of true widget events is well separated from that generated by other types of events. By setting the distance threshold to 10, we reject 95.1% of the false positive events when using the 8-widget interface shown in Fig. 14.1-left.

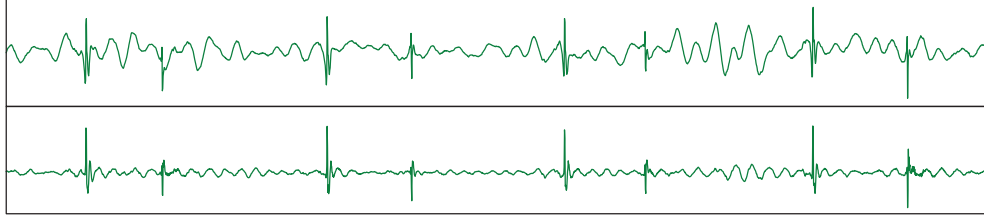


Figure 14.3: **(top)** We recorded accelerometer signals when using the widgets on a running subway train. **(bottom)** The noise introduced by the train motion can be largely eliminated through a high-pass filter, since the widget interaction events occur at a much shorter time scale.

More specifically, during the 30 minutes \* 13 participants = 390 minutes time period (including standing in a bus and running subway), we detected 122 false positive responses, and 116 of them are correctly rejected by our algorithm. This means that there are only 6 false positives for the entire 390 minutes.

We also tested our system in a very dynamic environment—a running subway with 5 participants. The test was performed on a running subway in a US city. The average speed is about 27km/h and the top speed is 89km/h according to the subway’s website. For this test, our participants performed the calibration in a static environment as usual, but evaluated the system’s accuracy by asking the participants to perform the test when they are on a running train. Although the train’s motion introduces additional signals, they can be largely cleaned through a high-pass filter as presented in §12.1. Figure 14.3 shows an example of the recorded signals before and after we apply the high-pass filter. When using the 8-widget interface shown in Fig. 14.1-left, we observed a slight drop in accuracy from 93.2% to 89.1%.

**Interaction speed** Widgets have no restriction on how quickly the user should interact with them. Since each user might have a different interaction speed, we wish to understand if the interaction speed would impact Widgets performance. To this end, we asked the participants to change their interaction speeds intentionally. We found that the widget’s accelerometer signal pattern is largely independent from the interaction speed, probably because the time scale of the event (less than 50ms) is much shorter than the highest user interaction frequency. The fastest user in our experiments can press a Widgets button 4 times per second. This corresponds to 8 events per second

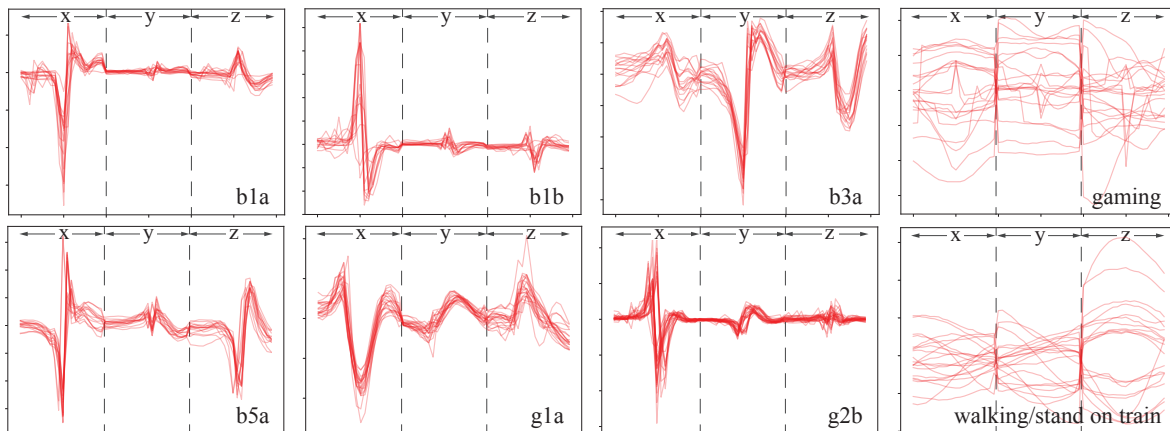


Figure 14.4: **Visual differences between widget signals and others.** In the first three columns, each plot visualizes 20 accelerometer signals generated by a widget event. The event is indicated by the ID in the bottom-right corner of the plot (in the same way as those used in Fig. 14.1). Each curve is a concatenation of the three channels (X-, Y-, and Z-channel) of accelerometer signals after align them on the time axis. It can be seen that each individual widget event produces signals sharing similar characteristics. In the last column, the two plots show the signal curves in the same way but for other events collected when the user is playing a mobile game (top-right) and walking/taking a subway train (bottom-right).

(including both the downstrokes and upstrokes), and a 125ms gap between the events, significantly larger than the 50ms detection window we use. Therefore, our algorithm has no difficulty to response to even the fastest user. Nevertheless, a Widgets knob can be rotated very fast. We found that if the user rotate the knob faster than 50mm/s, the event segmentations start to overlap and the detection will fail. In practice, this rarely happens though.

#### 14.4 Discussion about Layout and Robustness

We close the section by discussing a few rules of thumb that we learned throughout the experiments on where to place multiple widgets for maximal robustness. First, a widget generates the strongest motion along the finger pressing (for buttons) or sliding (for knobs) direction, called the operation direction. Therefore, aligning the operation direction with one of the accelerometer axes could produce stronger signals and better SNRs. For the same reason, if there are multiple widgets, it is desired to arrange them in a way such that their operation directions align with different axes of the accelerometer—thus, it becomes easier to distinguish the signals produced by different wid-

gets. If two widgets have to be placed closely and their operation directions are the same, then it is better to use tooth shapes that have clearly distinct resistance force profiles, such as those shown in Fig. 11.4.

	b1a	b1b	b2a	b2b	b3a	b3b	b4a	b4b	b5a	b5b	b6a	b6b	g1a	g1b	g2a	g2b
b1a	0.94	0	0	0	0	0	0	0	0	0	0	0	0	0.062	0	0
b1b	0	0.73	0	0.067	0	0	0	0	0	0.067	0	0	0.067	0	0.067	0
b2a	0	0	0.87	0.13	0	0	0	0	0	0	0	0	0	0	0	0
b2b	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
b3a	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
b3b	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
b4a	0	0	0	0	0	0	0.92	0.077	0	0	0	0	0	0	0	0
b4b	0	0	0	0	0	0	0	0.92	0	0.077	0	0	0	0	0	0
b5a	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
b5b	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
b6a	0	0	0	0	0	0	0	0	0	0	0.92	0.083	0	0	0	0
b6b	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
g1a	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
g1b	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
g2a	0.087	0	0	0	0	0	0	0	0	0	0	0	0	0.043	0.87	0
g2b	0	0.034	0	0	0	0	0	0.034	0	0	0	0	0.034	0	0	0.9

Figure 14.5: **Confusion matrix** for detecting user interaction events on 8 widgets. The layout of the widgets is shown in Fig. 14.1-left, and the k-NN classification used here has  $k = 1$ . In each of the row and column label, the first letter indicates the widget type ('b' for button, 'g' for knob), and the next number indicates the widget position corresponding to Fig. 14.1-left, and the last letter indicates the event type: for button, 'a' means downstroke and 'b' means upstroke; for knob, 'a' means forward rotation, and 'b' means backward rotation.

## Section 15: Discussion and Future Work

We have presented Vidgets, a set of modular mechanical widgets that can be used to construct a wide range of physical user interfaces for mobile devices. During their use, Vidgets widgets cause a slight shift of the device which can be detected by the on-device accelerometer. Thus, they are fully passive and power efficient. Vidgets are also low-cost and have small form factors. These features make them ideal for working in tandem with modern mobile devices. For example, they can be attached to the device's protective cases to customize personalized user interfaces and ease user interactions in many applications. Given the fact that more and more factories have removed most of the physical button on their mobile products, Vidgets provide an extensible option for tactile input on the commonly used phone cases. Because Vidgets are modular (on the phone case) and low-cost, the user can choose to use them whenever and however they need, and renew them if necessary.

**Limitations and future work** A potential limitation of Vidgets widgets is processing simultaneous user interaction events. In theory, if two events from different widgets occur simultaneously, the resulting signal would be a mix of both, and it will be hard to decompose it. In practice, though, we found this to hardly be a problem, because the time length for each event region is 50ms, and in our tests, it is hard for the user to trigger user interaction events within a time window as short as 50ms, even when the user intentionally tries to do so. However, when the user constantly rotates a knob while simultaneously pressing a button, the multiple interaction events may confuse the detection algorithm.

We observed a slight drop in detection accuracy when the widgets are used in a dynamical environment such as a running subway. It is possible that this drop can be reduced if the accelerometer can sample at a higher rate. In fact, most inertia measurement units on current mobile devices are

able to sample at a higher rate [86], yet we are limited by the APIs exposed from the operational system. Therefore, how to process accelerometer signals at a higher sampling rate is a worthy direction of future research.

Since our algorithm can distinguish downstrokes of Widgets buttons from upstrokes, the mobile application is able to know when the user is “holding” a button. But recognizing “holding” state depends on correctly detecting the downstroke event. If a downstroke is missed, the “holding” state would be completely lost. This is a limitation in comparison to the standard switch mechanism in which the state of a electronic circuit is changed when the switch is pressed down, after which an electronic signal is triggered continually as long as the switch is being hold.

Lastly, while we demonstrated the design of push buttons and rotary knobs, other types of mechanical widgets can be realized using a similar operational principle. An interesting future work would be to explore a wider range of widgets including sliders, joysticks, and more.

## **Chapter 3**

# **BackTrack: 2D Back-of-device Interaction Through Front Touchscreen**

## Section 16: Introduction and Related Work

Since the celebrated release of the first iPhone in 2007, the touchscreen has been a first-class citizen on a mobile device. Serving as the major interface both for displaying visual content and for taking user input, the touchscreen largely defines how we use the device. In many aspects, however, it also rigidifies our use: we are supposed to grip our phone in one hand, with four fingers responsible for holding it and our thumb and possibly even our other hand interacting with the touchscreen. We must constantly move our thumb out of the way of the screen to avoid occluding its visual content, while our other four fingers stay idle most of the time. In this format, the thumb is always engaged; the other four are always unengaged.

To address this rigidity and to liberate the finger from touching and occluding the screen, HCI community has become interested in Back-of-Device (BoD) interaction, and various systems have been proposed. For example, by taking advantage of the sensors on the phone, such as the camera [104], microphone [158, 163], and inertial measurement unit [156, 54, 86], the user can use their fingers that would normally stay idle to tap the backside of the phone and interact with the phone. BoD interactions aim to enable the user to use the device in a more flexible way, by engaging more fingers and alleviating the occlusion of the front screen. Yet, most of the existing BoD interactions have a limited vocabulary, supporting only finger tapping or 1D swiping (see more discussion in §22.1).

In this work, we propose *BackTrack*, a BoD trackpad that tracks fine-grained finger motions in 2D (see Fig. 16.1). In stark contrast to previous works, not only does our system enrich user interaction on the back of the device, it also offers remarkable advantages in its daily use. It requires no power supply, no wireless or port connection to the device, and no modification of the operating system—the entire system is encapsulated in a thin layer on the back of a phone case, ready to use with any off-the-shelf smartphones and tablets.

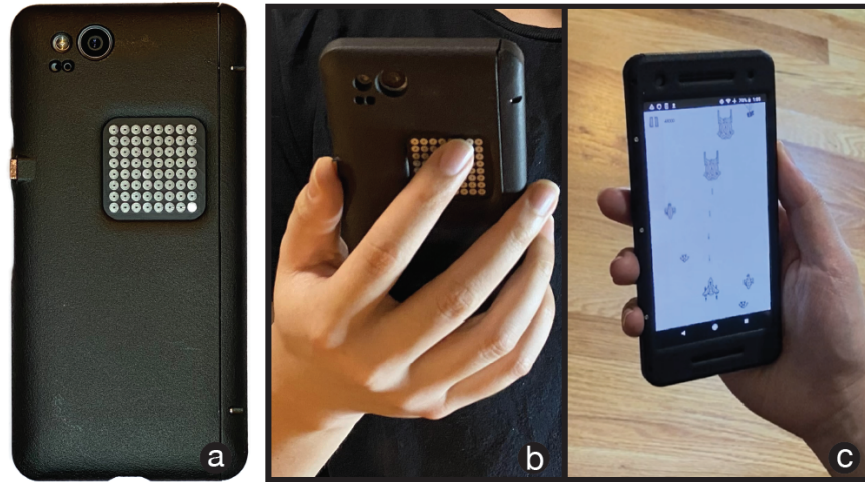


Figure 16.1: **BackTrack**. (a) Our BackTrack has a compact and clean design that provides a 2D trackpad on the back of the device. The entire system is encapsulated in a phone case that resembles a conventional case, and requires no power supply. (b) The user can interact with the phone by operating on the back-of-device trackpad. (c) In many apps, this type of interactions avoid occluding the screen.

To design such a trackpad on the back of the device, a natural starting point is the trackpad already on the device, that is, the front touchscreen. Nearly all touchscreens in today’s mobile devices are based on *capacitive sensing* techniques. When an electrical conductor, such as a finger, comes into contact with the touchscreen, it distorts the local electric field at that point. This distortion is measured (by the touchscreen’s driver) as a change in capacitance, and if strong enough, it is registered as a touch event. Under this working principle, a touch event can be triggered by any conductor of sufficient charge and surface area, not just a finger. For example, one can extend a certain point of the front touchscreen with a long conductive wire; a finger touching at the other end of the wire will trigger a touch event at that point (Fig. 16.2-a). Indeed, this is the idea behind several works that aim for off-the-device user interactions [161, 75, 148], although none of them address 2D finger tracking (see §22.1 for more discussion).

Our BoD trackpad exploits the idea of extending the front screen through conductive wires. But when it comes to 2D tracking, this general idea must be meticulously thought out, to materialize a design that is accurate, robust, and practical for its daily use. In the rest of this section, we

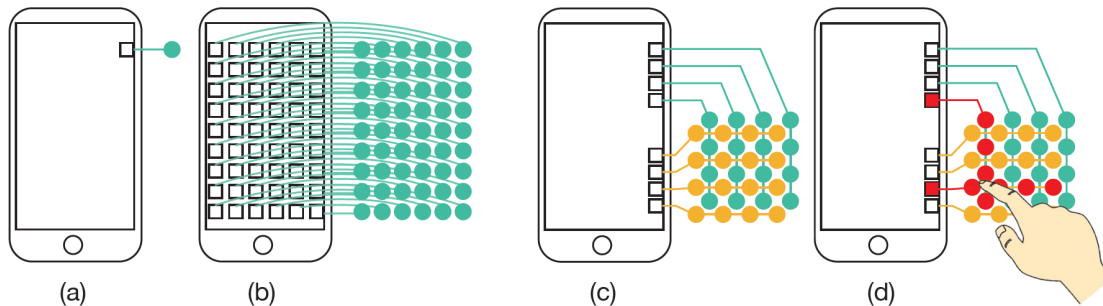


Figure 16.2: **BackTrack Overview.** (a) When a touchscreen is connected to a conductive wire, a finger touching at the other end of the wire (i.e., the green dot) will trigger a touch event (under the square) on the screen. (b) A naïve solution attempting to build a 2D trackpad is by connecting every (green) point on the trackpad to a distinct point on the screen through a conductive trace. But it is almost impossible to route all the traces from the front screen to the back with no intersection. (c) In our BackTrack design, electrodes on the trackpad are divided into two groups (green and yellow). In the first group (green), electrodes on each column are connected by a trace to a distinct electrode on the front screen. In the second group (yellow), electrodes on each row are connected to another distinct electrode on the front. (d) When a finger touches the BoD trackpad, it will trigger two touch events on the front screen (shown in red). The two touch events indicate, respectively, the column index of the green electrode and the row index of the yellow electrode touched by the finger. From the triggered row and column indices, we can recover the 2D coordinate of the touch location.

introduce the basic design of BackTrack and then discuss the considerations that drive our design choices toward a practical 2D tracking system.

## 16.1 BackTrack Overview

The core task of a trackpad is to sense the 2D position of a finger touch. A naïve approach would be a simple extension of the idea above: connect every point on the trackpad to a corresponding point on the front touchscreen through conductive traces (see Fig. 16.2-b). Thereby, the finger position on the trackpad is one-to-one mapped to individual touch events on the front screen. However, this approach only affords an exceedingly low tracking resolution. If the resolution is  $N \times N$ , it will consume  $N^2$  points on the front screen and require  $N^2$  traces. To build a compact BoD trackpad, it is very hard, if not impossible, to route those  $N^2$  traces without any intersection. In addition to being intersection-free, any pair of traces must be well separated. Otherwise, the close proximity of two traces introduces additional capacitance which will interfere with the

measurement of touch-induced capacitance changes.

Instead, we realize that the front screen is not just a touchscreen but a *multitouch* screen; it can sense capacitance changes at multiple locations simultaneously. Leveraging this feature, we can reduce the design complexity significantly. BackTrack requires only  $2N$  traces for a trackpad with resolution  $N \times N$ , as shown and described in Fig. 16.2-c. A finger touching on the BoD trackpad triggers two touch events at the same time on the front screen (see Fig. 16.2-d); those two events encode the  $x$ - and  $y$ -coordinate of the touching point, respectively.

In our approach, we place  $2N$  electrodes near the edge of the front screen so that they can be easily traced to the BoD trackpad (Fig. 16.2-c). The electrodes are made of indium tin oxide (ITO) coated on a thin glass (0.8 mm thickness). The ITO glass is fully transparent, covering the front screen just like a screen protector. As a result, the entire front screen continues functioning as it does without the ITO glass.

**Design considerations** While the basic idea behind BackTrack seems simple, the devil is in the details. Our BackTrack design must address several challenges arising from its use in practice:

- i) Our goal is to allow BackTrack to be readily used on any off-the-shelf mobile devices. To this end, we do *not* assume the user can modify the touchscreen's driver in the kernel space. Nor do we assume the applications using BackTrack have access to the touchscreen's raw capacitance signals. Applications must function with unmodified operating system's application programming interfaces (APIs) solely. This goal sets us apart from any previous BoD systems that require kernel access (see §22.1). Therefore, we need to ensure finger touches on the BoD trackpad to produce strong capacitance changes that will be robustly registered by the unmodified driver as touch events. This goal challenges our design. We must judiciously choose design parameters such as electrode's size and shape and connection pattern.
- ii) BackTrack encodes a BoD finger touch location in two touch events on the front screen. The encoding scheme, determined by how the trackpad electrodes are connected (Fig. 16.2-b), have many variations (see Fig. 18.2 for a few examples). The electrodes' connection pattern

will affect the robustness of finger motion tracking. We therefore need to find one that is robust to commonly used finger motions (see §18.2).

- iii) While the BoD trackpad enriches user interaction, it is also prone to accidental touches—after all, the backside is where fingers are positioned to hold the device. To prevent unintentional touches, we introduce a switch on the side of the device (Fig. 22.1-a). BackTrack is activated only when the user positions their thumb on the switch. To design such a switch, care must be taken. Both mechanical and analogue switches suffer from limitations on their form factor, analogue switches would also require a power supply (see §18.3). Interestingly, the capacitive nature of the touchscreen allows us to devise an alternative design that is battery-free and as thin as the copper foil.

To justify these design choices, we develop a capacitive circuit model that helps us analyze how design parameters affect BackTrack performance. Our design informed by this model is further backed by finite-element simulation of the BoD trackpad as well as controlled laboratory experiments. In the end, we demonstrate a rich set of user interaction features enabled by BackTrack in commonly used applications, including the web browser, mobile game, file manager, passcode input, and others. We also conduct user studies to evaluate BackTrack’s performance and ease of use.

## 16.2 Back-of-device Interaction

Most relevant to our work, BackTrack, is the area generally known as Back-of-Device (BoD) interaction. Early work in this area aimed to use BoD user interaction to avoid finger occlusion of the front screen [138, 153, 64, 84]. More recently, BoD interactions have been used on small-screen devices [11, 157], to augment front-screen operations [36, 12, 100, 159], and for authentication [38, 92, 37].

Most BoD techniques build electronic hardware and connect it to a mobile device through a cable or wireless connection. For instance, *BlindSight* [95] incorporates a 12-key pad on the back

of a phone; *InfiniTouch* [89] builds a capacitive sensing array around the entire phone to sense finger presence and position; and Hakoda et al. [58], after analyzing the range of index finger motion on the back of the device, employed a small photoreflector to track finger motion. In all these techniques, a power supply is needed to drive their electronic sensors. As a result, these techniques require either a cable connection to the phone to draw power or an external battery.

The requirement of a power supply is a notable limitation. To overcome this limitation, other methods take advantage of the already existing sensors on the phone. Along this direction, the mainstream idea is to leverage physical (in contrast to electronic) channels to communicate between the finger and device. Various sensing channels such as light sensed by the camera [104, 154], sound sensed by the microphone [158, 163, 139, 146], and vibration sensed by the inertial measurement unit (IMU) [156, 54, 86, 166] have been explored for BoD interaction. However, recognizing intended user interactions from physical raw signals is not as straightforward as using capacitive sensors. More importantly, those signals are often susceptible to noise: a microphone will record any environmental noise such as human voices, and IMU sensors will capture signals produced by movements like walking and hand motion, apart from the intended inputs from finger gestures.

In contrast, BackTrack is an electronic (capacitive) sensor specifically designed for BoD finger tracking. It thus enjoys the advantages of capacitive techniques—fast, robust to environmental noise, requiring a simple recognition algorithm. Additionally, as a fully passive sensor, it requires no power supply and no port or wireless connection to the phone.

**Capacitive sensing** has been extensively studied for decades [91]. Today, used on almost every mobile device, it has become a technological staple of our daily life. It is nearly impossible to enumerate the rich literature of this field here. We, therefore, refer the reader to the survey by Grosse-Puppenthal et al. [56] for a thorough review of capacitive sensing in HCI. Here, we only review the works that are closest to ours, and that also extend the capacitive touchscreen with conductive materials.

*Clip-on Gadgets* [161] attaches conductive tactile buttons on some locations of a touchscreen.

	IS	FT	OS	DM
Clip-on Gadgets [161]	0D	No	Yes	No
ExtensionSticker [75]	1D	Yes	Yes	No
FlexTouch [148]	2D	No	No	No
Ohmic-Sticker [69]	2.5D	No	No	No
<b>BackTrack</b>	2D	Yes	Yes	Yes

Table 16.1: **Comparison with previous work.** **IS** indicates interaction space; **FT** indicates whether the method can track finger motions; **OS** indicates if the system can be used without modifying the driver in the operating system’s kernel space; **DM** indicates if the system provides a switch mechanism to prevent unintentional touches.

When the user presses a tactile button, it triggers a touch event on the screen. Here, the goal is to provide the user with tactile feedback when they operate on the screen. However, unlike BackTrack, the attached buttons will occlude the screen, and the interactions only takes place at places where the buttons are attached.

*ExtensionSticker* [75] further develops this approach by attaching a 1D array of conductive lines to the touchscreen. It allows the user to swipe or tap on a surface outside the actual screen area. But it is limited to 1D operations, as a touch on the other end of the conductive lines is mapped one-to-one to a location on the screen.

*Ohmic-Sticker* [68, 69] is based on an interesting phenomenon. When a conductor is attached to a certain point on the touchscreen, the signal of the touchscreen at that point is affected by the resistance of the conductor. Based on this observation, the authors built an input device similar to a TrackPoint [7]. With a further extension, it can also sense input forces, serving as a 2.5D input device. A TrackPoint, however, is fundamentally different from a trackpad. Some research has concluded that the usability of TrackPoint is more limited than a trackpad [155, 7].

*FlexTouch* [148] extends the touchscreen to a large area. With access to the touchscreen’s raw capacitance signal, the authors trained a machine-learning model to recognize certain human poses on a yoga mat. But it is unclear how their method can be applied to track finger motions in BoD applications.

Both Ohmic-Sticker [68, 69] and FlexTouch [148] require access to the raw signals of capac-

itive sensors. This is a notable requirement; for off-the-shelf mobile devices, this means kernel hack, which is not always feasible. Thus, unlike BackTrack, these approaches may not be useable on devices in which raw signals are hard to retrieve.

Lastly, none of these methods provide a switch mechanism, that allows a user to enable/disable the sensor at their will. This may not be an issue for their target applications. But for BoD interactions, unintentional touches on the back are almost unavoidable. Thus, it is highly desired, if not required, to include a switch to activate the BoD trackpad.

Table 16.1 summarizes the technical features of our method and closely related works.

## Section 17: A Capacitive Circuit Model for BackTrack

We now present a capacitive circuit model of our BackTrack design. This model will guide our choices of various design parameters discussed in the next section.

**Background on capacitive touchscreen** Capacitive touchscreen senses the touch of a conductive object, such as a finger, by measuring capacitance change in each pair of electrodes (called *transmit* and *receive* electrodes [56]) placed underneath the touchscreen (see Fig. 17.1). The transmit and receive electrodes form two capacitors in parallel,  $C_{TR}$  and  $C_M$ , contributed by electric fields in the air and screen glass, respectively (Fig. 17.1-a). This is effectively a resistor–capacitor (RC) circuit. Driven by a square wave generator, if the transmit electrode has a voltage switching from 0 (low voltage) to  $V$  (high voltage) at  $t = 0$ , the voltage at receive electrode changes over time in the following way:

$$V_r(t) = V \left( 1 - e^{-\frac{t}{R_0 C}} \right), \quad (17.1)$$

where  $C = C_{TR} + C_M$  is the effective capacitance of the circuit. Equation (17.1) indicates that the receive electrode voltage will reach  $V (1 - e^{-1})$  after time  $\tau_{idle} = R_0 C$ .

When a finger comes into contact with the screen,  $C_M$  stays unchanged but  $C_{TR}$  is altered (Fig. 17.1-b). Two effective capacitors are formed: one between the transmit electrode and the finger, and another between the receive electrode and the finger. As a result, the total capacitance between the transmit and receive electrodes increases, because the finger, as an electric conductor, effectively reduces the distance between the two electrodes. Let  $C_1$  denote the total effective capacitance, and ignore the finger resistance ( $R_{body}$  in Fig. 17.1-b), as it is typically small. Now, receive electrode voltage will reach  $V (1 - e^{-1})$  after time  $\tau_{idle} = R_0 C_1$ .

This analysis shows that the time delay ( $\tau_{act}$  and  $\tau_{idle}$ ) is proportional to the system's effective capacitance. Thus, instead of measuring the capacitance change directly, the touchscreen driver

monitors the change in time delay  $\tau$  on every rising edge of the square wave generator. Only when the change is sufficiently large (e.g.,  $\tau_{\text{act}} - \tau_{\text{idle}}$  introduced by a finger touch) does the driver detect a touch event, and notify the operating system.

### 17.1 BackTrack Circuit Model and Analysis

In our BackTrack design, a number of indium-tin-oxide (ITO) electrodes are placed on top of the touchscreen glass (recall Fig. 16.2-a). Each ITO electrode, together with the transmit and receive electrodes underneath it, forms two effective capacitors,  $C_{\text{TR1}}$  and  $C_{\text{TR2}}$ , respectively (see Fig. 17.1-c). Meanwhile, the ITO electrode is connected to the electrodes on the BoD trackpad, which introduces some additional resistance and capacitance. The circuit model of this setup is shown in Fig. 17.1-c. This is also an RC circuit, in which the time delay for the receive electrode to reach  $V(1 - e^{-1})$  is

$$\tau_{\text{idle}} = R_0 \left[ C_M + \frac{(C_{\text{elec}} + C_{\text{TR1}}) \cdot C_{\text{TR2}}}{C_{\text{elec}} + C_{\text{TR1}} + C_{\text{TR2}}} \right], \quad (17.2)$$

where the resistance and capacitance notations are shown in Fig. 17.1-c. Here in the derivation, we ignore the small resistance  $R_{\text{elec}}$  between the ITO electrode and the BoD electrodes, as its contribution is negligible.

Moreover, Figure 17.1-d illustrates the case in which a finger touches some electrodes on the BoD trackpad. Since those electrodes are connected to an ITO electrode on the front screen, the finger introduces additional resistance and capacitance in the RC circuit. Now, the time delay experienced by the receive electrode becomes into

$$\tau_{\text{act}} = R_0 \left[ C_M + \frac{(C_{\text{elec}} + C_{\text{body}} + C_{\text{TR1}}) \cdot C_{\text{TR2}}}{C_{\text{elec}} + C_{\text{body}} + C_{\text{TR1}} + C_{\text{TR2}}} \right]. \quad (17.3)$$

We can simplify Eq. (17.3) further by assuming  $C_{\text{TR1}} = C_{\text{TR2}} = C_T$  due to the symmetry between transmit and receive electrodes. Now, when a finger touches the BoD trackpad, the touchscreen

driver captures a change of the time delay, which is expressed as

$$\tau_{\text{act}} - \tau_{\text{idle}} = R_0 \left[ \frac{(C_{\text{elec}} + C_{\text{body}} + C_T) \cdot C_T}{C_{\text{elec}} + C_{\text{body}} + 2C_T} - \frac{(C_{\text{elec}} + C_T) \cdot C_T}{C_{\text{elec}} + 2C_T} \right] = f(C_{\text{elec}} + C_{\text{body}}) - f(C_{\text{elec}}), \quad (17.4)$$

where  $f(\cdot)$  is a function defined to aid our analysis coming next, namely,

$$f(C) := R_0 \left[ \frac{(C + C_T) \cdot C_T}{C + 2C_T} \right]. \quad (17.5)$$

**Analysis** The time delay change  $\tau_{\text{act}} - \tau_{\text{idle}}$  is what the touchscreen driver monitors. Only when it is large enough, the finger touch on the BoD trackpad is recognized as a touch event on the front screen. Therefore, to achieve robust trackpad response, the BackTrack design must aim to increase  $\tau_{\text{act}} - \tau_{\text{idle}}$ .

Also notice that  $f(C)$  is a monotonically increasing function of  $C$ , as indicated by its derivative with respect to  $C$ , that is,

$$f'(C) = \frac{C_T^2 R_0}{(C + 2C_T)^2} > 0 \text{ for all } C. \quad (17.6)$$

This analysis suggests two guidelines for choosing BackTrack's design parameters. **i)** To increase  $\tau_{\text{act}} - \tau_{\text{idle}}$ , which is equivalent to  $f(C_{\text{elec}} + C_{\text{body}}) - f(C_{\text{elec}})$ , we need to increase  $C_{\text{body}}$  (for increasing the first term) and reduce  $C_{\text{elec}}$  (for reducing the second term). **ii)** Once  $C_{\text{body}}$  and  $C_{\text{elec}}$  are set, another way of increasing  $f(C_{\text{elec}} + C_{\text{body}}) - f(C_{\text{elec}})$  is by increasing the rate of change from  $f(C_{\text{elec}})$  to  $f(C_{\text{elec}} + C_{\text{body}})$ , that is, the derivative  $f'(C)$ . This suggests that we should seek to increase  $C_T$ , as  $f'(C)$  is monotonically increasing with respect to  $C_T$  (one can verify that the derivative of  $f'(C)$  with respect to  $C_T$  is always positive).

The two guidelines help us justify our BackTrack design, as described in the next section.

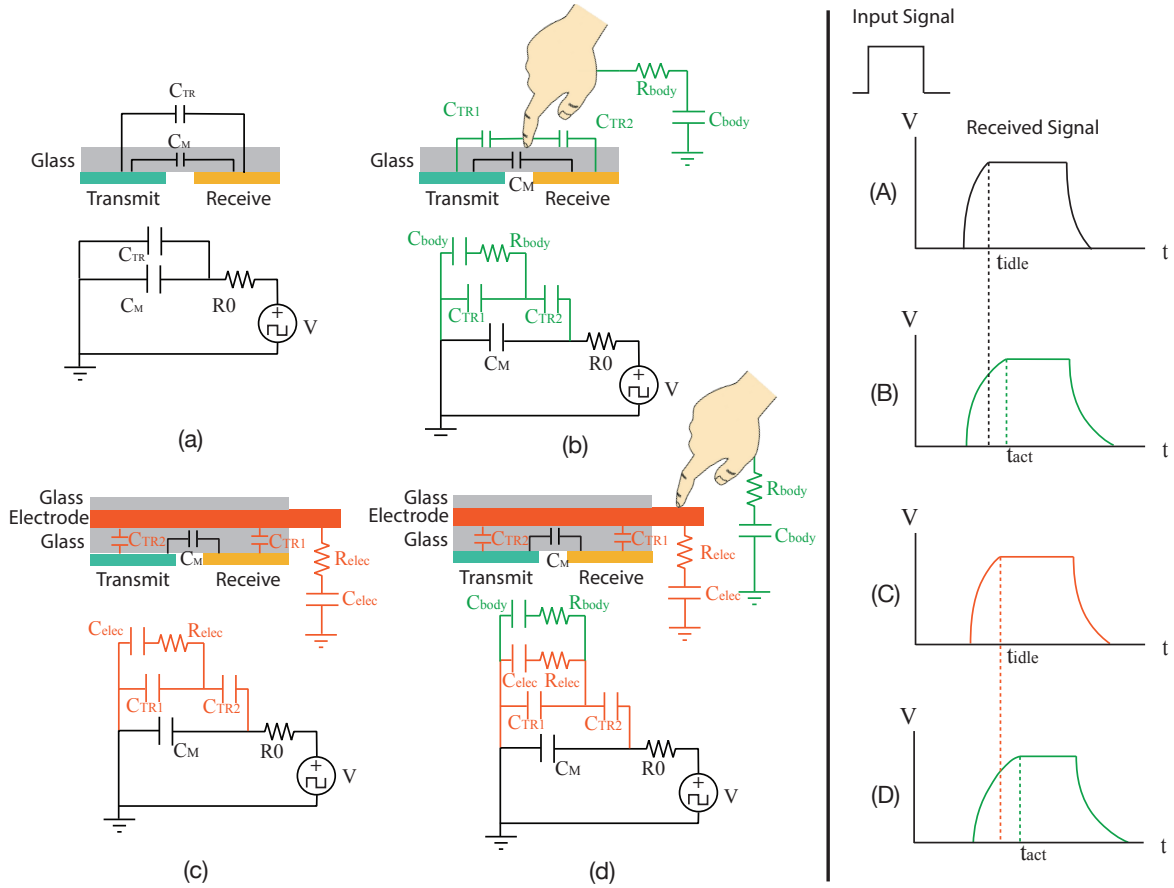


Figure 17.1: **Circuit models.** (a) A capacitive circuit model of the conventional touchscreen. Driven by a square wave generator (top-right), the voltage at the receive electrode changes as in (A) on the right. (b) When a finger comes into contact with the touchscreen, the effective circuit changes, and the voltage signal at the receive electrode takes a longer time to reach its maximum (B). (c) The circuit model of our BackTrack design in absence of finger touches. Here it shows the model related to a single ITO electrode on the front screen. Different ITO electrodes are separate from each other. (d) The circuit model of BackTrack in presence of a finger touch. (C, D) on the right show the voltage signals on the receive electrode, corresponding to the models in (c) and (d), respectively.

## Section 18: BackTrack Design

### 18.1 Electrode Design

**Electrodes on the BoD trackpad** are specified by their shape, size, and spacing  $d$  (see Fig. 18.1). Choices of these parameters determine the effective capacitance  $C_{\text{elec}}$  in Fig. 17.1-c. According to the design guidelines in §17, we wish to reduce  $C_{\text{elec}}$ . But the reduction of  $C_{\text{elec}}$  must balance two conflicting factors: On the one hand, electrodes with a small size are preferred, because smaller electrodes introduce less capacitance. On the other hand, as the electrode reduces its size, it becomes harder for a fingertip to firmly contact the electrode, making the trackpad less reliable. Empirically, we found that a diameter of 3mm is a proper compromise.

Additional capacitance is also introduced due to the proximity of unconnected electrodes. Recall the electrode layout shown in Fig. 18.1-a. Electrodes in each horizontal (and vertical) line are connected, and across different lines they are separate. As a result, electrodes close to but separate from each other form another effective capacitor  $C_{\text{cros}}$ , one that arises in the circuit model in parallel to  $C_{\text{elec}}$  (Fig. 18.1-b). This effectively increases the electrode's capacitance  $C_{\text{elec}}$ , and thus we choose electrode shape and spacing with the aim of reducing  $C_{\text{cros}}$ .

To this end, we use COMSOL, a commercial finite-element simulation tool, to predict  $C_{\text{cros}}$  under different electrode configurations. As reported in Fig. 18.1, we consider both square and circular electrodes each with difference spacings. The simulation shows that circular electrodes lead to a smaller  $C_{\text{cros}}$  than square electrodes, and as the spacing increases,  $C_{\text{cros}}$  reduces (Fig. 18.1-c). Also note that if the spacing  $d$  is too large, the electrodes would become too sparse to be contacted reliably. Therefore, we choose to use circular electrodes with a spacing  $d = 3.1\text{mm}$ .

**Electrodes on the front screen** are mainly specified by their size; their shape and spacing are not crucial. This is because the frontal electrodes are not meant to be directly touched by the

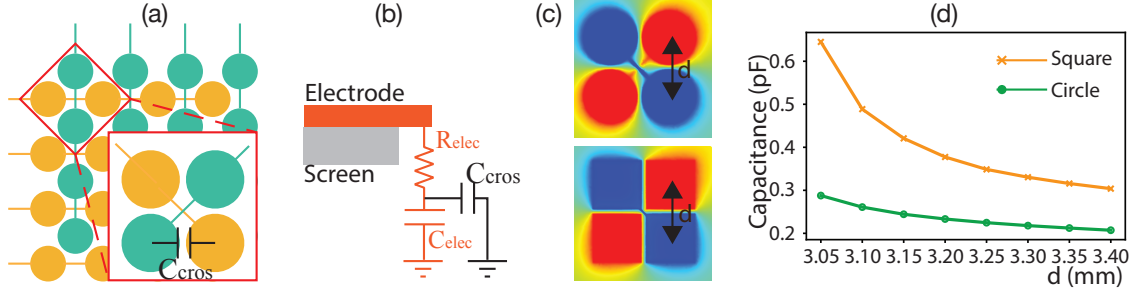


Figure 18.1: **Simulation of electrode coupling.** (a) Electrodes that are close by but separate from each other (such as the green and yellow ones in the red box) introduce additional capacitance  $C_{\text{cros}}$ , which will arise in our circuit model in parallel to  $C_{\text{elec}}$  shown in (b). Note that here (b) is the circuit model related to one line of connected electrodes (e.g., a row of yellow ones), not to the entire set of electrodes. In (c), We visualize the electric field near the electrodes highlighted in (a) (after applying a  $45^\circ$  rotation). This is simulated using finite-element method in COMSOL. Here we consider two electrode shapes, circular (top) and square (bottom). (d) We plot the change of  $C_{\text{cros}}$  with respect to the parameter  $d$  (where  $d$  is defined in (c)).

finger, and thus they can be sparsely placed near the edge of the front screen . The size of a frontal ITO electrode affects the capacitance  $C_T$  between the ITO electrode and the transmit (and receive) electrode underneath the touchscreen (Fig. 17.1-c). Enlarging the size will increase  $C_T$ . According to the analysis in §17, to improve the trackpad robustness, we need to increase  $C_T$  and thus enlarge the electrode size.

But the ITO electrode cannot be too large. Otherwise, it will be coupled with many transmit and receive electrodes underneath the screen, and will form many effective capacitors. In that case, whenever a finger touches the glass over that exceedingly large ITO electrode, the screen region affected by the finger touch will not depend on the fingertip any more, but depend on the area of the ITO electrode, which is much larger than the fingertip. Then, the finger touch will not be recognized by the driver as a touch event (recall that the phone will not respond to a contact from your palm). In short, the size of the frontal electrode must not be too large or too small. Empirically, we found that the size of 5mm×12mm (in rectangular shape) with 0.4mm spacing results in the best robustness. In that setting the front screen still functions as expected.

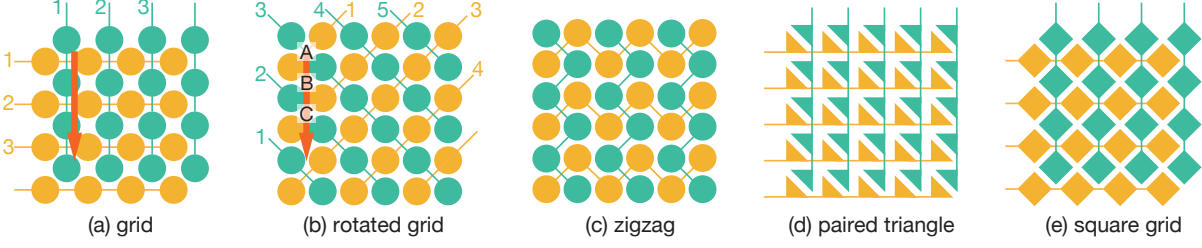


Figure 18.2: **Electrode connection patterns.** We show five different patterns for connecting the trackpad electrodes, and refer them as (a) *grid*, (b) *rotated grid*, (c) *zigzag*, (d) *paired triangle*, and (e) *square grid* patterns. Different patterns encode the coordinate of touch location differently, and lead to different robustness for finger tracking. To understand the robustness difference, consider the grid (a) and rotated grid (b) patterns, and a finger swiping downward (red arrow). With the grid pattern, the triggered row and column indices are (1,1), (2,1), (3,1), ... When the finger swipes down on the rotated grid pattern from position A (shown in (b)) to C, the triggered index sequence can be either [(1,3), (1,2), (2,2)] or [(1,3), (2,3), (2,2)], depending on at the transition position B which electrode gets crossed first. This uncertainty is the source of jittering shown in Fig. 18.3.

## 18.2 Electrode Connection Pattern

BackTrack relies on two groups of electrodes to map the 2D coordinates of a touch location into two simultaneous touch events on the front screen (Fig. 16.2). This mapping is determined by the specific pattern that connects each group of electrodes, and it has many options. For example, Figure 18.2 shows five different patterns. For the purpose of finger tracking, each pattern leads to different robustness, which is illustrated in Fig. 18.2-a, b and their caption. In what follows, we first evaluate the robustness of various patterns using controlled laboratory experiments, and then propose an algorithm to further improve tracking robustness.

**Robustness evaluation** We consider four connection patterns, namely grid, rotated grid, zigzag, and paired triangle patterns (see Fig. 18.2-a, b, c, d). For the first three patterns, we have two variants: one with circular electrodes and another with square electrodes. For each pattern variant, we also consider two different spacings:  $d = 3.1\text{mm}$  and  $d = 3.4\text{mm}$  ( $d$  is defined in Fig. 18.1-b). In total, we evaluate 14 different methods of electrode connection.

For each method of electrode connection, we evaluate its robustness for two types of user interactions: tapping and swiping. All the evaluations are conducted by a robotic arm holding a conductive pen (whose size is comparable to a fingertip; see Fig. 18.3-a). In tapping test, we

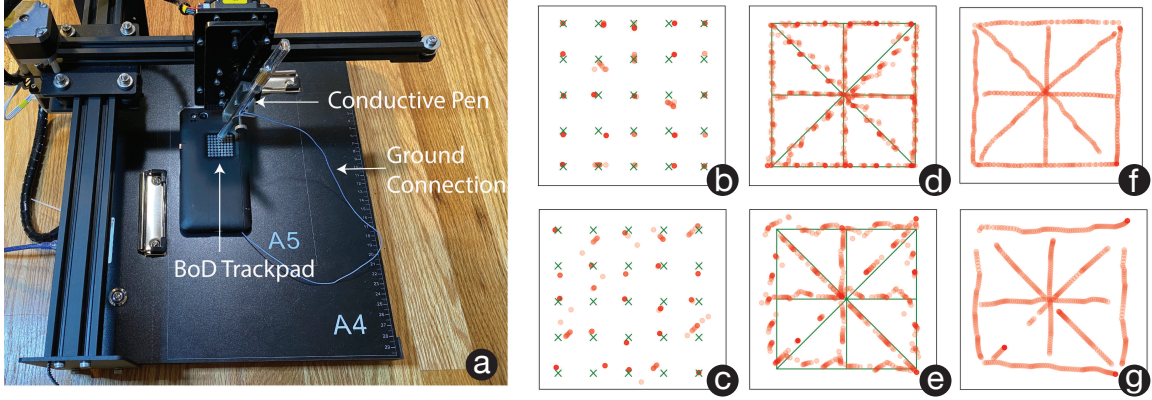


Figure 18.3: **Robotic tests.** (a) We use a robotic arm to test the robustness of various electrode shapes and connection patterns on the trackpad. The top row shows the results from the grid pattern in Fig. 18.2-a, and the bottom row shows the results from the rotated grid pattern in Fig. 18.2-b. (b, c) visualize the tapping test results: the green cross marks indicate the tapping positions by the robot, and red dots indicate the positions sensed by BackTrack system. (d, e) show the swiping test results: green lines are swiped by the robot, red dots indicate the sensed swiping trajectories. (f, g) are the swiping trajectories output from BackTrack system after applying the low-pass filter (18.1). program the robot to tap  $5 \times 5$  grid points on the trackpad (Fig. 18.3-b,c) and measure the mean square error between the sensed touch locations and true locations. In swiping test, we instruct the robot to swipe along horizontal, vertical, and diagonal lines (Fig. 18.3-d,e). We then measure the average distance from the sensed touch locations to their intended lines. We repeat each test 5 times and report the average error.

In addition, through an Android kernel hack, we are able to retrieve the raw signal value returned by the touchscreen driver. This raw value is proportional to the time-delay difference discussed in §17. We normalize the raw values in  $[0, 1]$  range, where 0 corresponds to the signal level in absence of touches, and 1 corresponds to the signal produced by a finger touching the front screen. With the normalization, in each experiment we collect and report the mean and standard deviation of the resulting raw signal. We also note that the kernel hacking is not needed to use BackTrack. Here we just leverage it for the evaluation of pattern robustness.

The evaluation results are reported in Table 18.1, from which two patterns stand out in terms of robustness, namely the grid and rotated grid patterns both with circular electrodes. The former performs the best for sensing finger tapping and swiping horizontally and vertically. The latter is more suitable for finger swiping along diagonal lines. The reason behind this is discussed in the

Conn. Pattern	Shape	Spacing(mm)	Avg. Signal	Tap Err.	Swipe H/V Err.	Swipe D Err.
Grid	Square	3.1	0.42±0.12	0.785	1.110	1.346
		3.4	0.47±0.11	0.832	1.090	1.361
	Circle	3.1	0.62±0.09	<b>0.672</b>	<b>0.729</b>	0.942
		3.4	0.64±0.10	0.753	0.769	1.102
Rotated grid	Square	3.1	0.69±0.17	0.788	1.113	0.641
		3.4	0.73±0.18	1.009	1.231	0.664
	Circle	3.1	0.73±0.15	0.767	0.844	<b>0.591</b>
		3.4	<b>0.75±0.16</b>	0.815	1.224	0.613
Paired triangle	Triangle	3.1	0.51±0.10	0.875	1.542	1.258
		3.4	0.53±0.11	0.963	1.825	1.724
Zigzag	Square	3.1	0.39±0.08	0.824	1.099	1.102
		3.4	0.42±0.08	0.863	1.527	1.060
	Circle	3.1	0.46±0.07	0.791	0.874	0.884
		3.4	0.47±0.09	0.742	0.948	0.861

Table 18.1: **Robotic test results** of experiments shown in Fig. 18.3-a. H/V represent Horizontal/Vertical and D represent Diagonal.

caption of Fig. 18.2. In practice, tapping and horizontal and vertical swiping are used more often, and therefore we use the grid pattern with circular electrodes in our BackTrack prototype.

*Remark.* Table 18.1 also reveals that for each pattern, 1) the circular electrodes produce stronger raw signal than the square electrodes, and 2) 3.4mm spacing leads to stronger response than 3.1mm spacing. Both observations agree with the conclusions we draw from the finite-element simulation described in §18.1 and Fig. 18.1.

**Temporal filter** As shown in Fig. 18.3 and discussed in Fig. 18.2, swiping along an arbitrary direction may result in a slightly jittered finger motion path. This is due to the limited trackpad resolution ( $N = 7$  in our tests). One can always reduce the jittering by increasing the resolution. In addition, a simple low-pass temporal filter can further reduce the jittering and improve the robustness, without changing the hardware. The low-pass filter is defined as

$$\mathbf{y}_i = \mathbf{y}_{i-1} + \alpha \cdot (\mathbf{x}_i - \mathbf{y}_{i-1}), \quad (18.1)$$

where the subscript indicates the sensing cycle in time,  $\mathbf{y}_i$  is the filtered touch location at timestep  $i$ ,  $\mathbf{x}_i$  is the sensed touch location at timestep  $i$  prior to the filtering, and  $\alpha$  is a parameter that controls the filter strength (in practice, we use  $\alpha = 0.1$ ). This filter is applied in realtime, and see Fig. 18.3 for its effect.

### 18.3 Switch Design

As discussed in §18.1, a BoD trackpad, if not carefully designed, will suffer from unintentional touches. Fingers accidentally touching the trackpad can interrupt the user’s interaction with the front screen. We now describe our switch design, with which the user can activate (and deactivate) the BoD trackpad at will.

**Option 1: Mechanical switch** Perhaps the most straightforward option is to use mechanical switch, not a single one but many of them. Recall that if the trackpad has a resolution  $N \times N$ , BackTrack uses  $2N$  traces to connect the electrodes on the front screen with those on the back ( $N = 7$  in our prototype). To fully disable the trackpad, we need  $2N$  switches, each placed on an individual trace to control its connection, and we need to switch them on or off simultaneously. This is indeed what an  $X$  Pole Single Throw (XPST) switch is designed for, where  $X$  is the number of connections to be controlled simultaneously. Unfortunately, it is extremely hard to design a compact XPST switch with a large  $X$ . Existing designs often have  $X$  up to 4, much smaller than our requirement.

**Option 2: Analogue switch** Alternatively, one may consider to use analogue switches. An analogue switch typically consists of a pair of transistors, switching between low and high resistance electrically with no moving parts. Often manufactured as integrated circuits, many analogue switches can be packed in a small form factor. But a key drawback is that they require power supply to function. This means that the BoD trackpad has to carry a battery or draw power from the mobile device. Either way, it is less practical. Moreover, the analogue switch itself also imposes sizeable capacitance in the system—this is confirmed in our experiments. Using them on each

trace increases  $C_{\text{elec}}$  (in Fig. 17.1-c) and thus renders the trackpad less reliable (recall the design guidelines in §17).

**Our proposal: embracing capacitance** Here we propose a new switch mechanism based on the insight developed in §17. We show that a novel use of a piece of copper foil can serve as a battery-free switch. First, in Eq. (17.4), if  $C_{\text{body}}$  approaches to zero, then  $f(C_{\text{elec}} + C_{\text{body}})$  approaches to  $f(C_{\text{elec}})$ , and the detected time delay difference  $\tau_{\text{act}} - \tau_{\text{idle}}$  becomes vanishing small regardless of the  $C_{\text{elec}}$  value. In other words, when  $C_{\text{body}}$  is sufficiently small, finger touches on the BoD electrodes will not be captured by the touchscreen driver—the trackpad becomes deactivated.

Now, our goal is to find a way of switching  $C_{\text{body}}$  between a low and high value. By definition,  $C_{\text{body}}$  is the capacitance between our body (i.e., the finger) and the ground. Here a subtle yet important detail is the notion of “ground”. It refers to the effective ground of the mobile device, which is not necessarily the physical ground. In fact, when we use a plastic phone case (just like most of us do), the phone’s ground is already insulated from the physical ground as well as our body. Because of this insulation,  $C_{\text{body}}$  is so small that the BoD trackpad is deactivated. To enable the trackpad, we simply need to connect the user’s body with the phone’s ground through a conductor, such as a piece of copper foil, so that  $C_{\text{body}}$  becomes large.

When implementing this switch, we leave a small hole on the side edge of the phone case, near where the thumb is usually positioned. Behind that hole, we place a piece of copper foil connecting to the phone’s shell (which is covered by the phone case). When the user wishes to use BackTrack, they just place their thumb over the hole to activate the trackpad. This simple design is battery-free and consumes almost no space. It can also be easily extended by using a momentary mechanical switch to provide haptic feedback or turning it into a toggle switch.

## 18.4 Prototype Details

We prototype BackTrack on Google Pixel 2, while the same work principle is also tested on other phone models, including LG Nexus 5, Samsung Galaxy S8, and iPhone 11 Pro Max. To



Figure 18.4: **BackTrack assembly.** (a) ITO electrodes are coated on the side of a thin glass. The ITO pattern will become fully invisible when attached to the phone’s screen. (b) A flex PCB is used for connecting the front ITO and BoD trackpad. (c) BoD trackpad is printed on a PCB. (d) The BoD trackpad is first put in a custom-designed phone case. We also attach to the BoD PCB a piece of copper foil, which will connect the BoD PCB with the phone’s ground after the phone is put into the case. This copper foil is also used as a capacitive switch (§18.3) to enable/disable BackTrack. (e) The ITO-coated glass is connected to the flex PCB by conductive epoxies. We then put the phone together with the glass into the phone case. (f) Lastly, we connect the flex PCB with the BoD PCB. (g) The final fully assembled BackTrack.

work with Google Pixel 2, we manufacture four parts: a phone case, a frontal glass coated with ITO electrodes, the side flex PCB, and the back PCB with the trackpad electrodes (see Fig. 18.4). The frontal glass carries 14 ITO electrodes, which are connected to the flex PCB using conductive epoxies. The other end of the flex PCB is connected to the back PCB. All the component are then encapsulated in a custom-designed phone case, which appears very much like a conventional phone case. The cost of the phone case<sup>1</sup>, the frontal glass with ITO coating, the side flex PCB, and the back PCB are \$4, \$10, \$3, and \$2, respectively.

Our BackTrack prototype has a trackpad with  $7 \times 7$  electrodes ( $N = 7$ ), a resolution lower than the front screen. This choice is not a limitation in our design—one can use a higher resolution of electrodes. Rather, this choice is justified by the limited range of finger motions when the finger is positioned on the back holding the device [58]. After all, BackTrack is not meant to replace the font screen, but to enrich it with an additional space for user interaction. Thus,  $7 \times 7$  electrodes on a small trackpad area suffice.

<sup>1</sup>Here we take into account the material cost for 3D printing the phone case.

## Section 19: Applications

BackTrack enriches the vocabulary for the user to interact with mobile devices. Now, putting it in the context of a few commonly used applications, we demonstrate several new user interaction features enabled by BackTrack.

### 19.1 Avoiding Occlusion using BoD Interaction

Finger touches on the front screen occludes the front screen. This occlusion can make user interaction inconvenient, sometimes even annoying. BackTrack liberates the finger from always interacting on the front screen, and thereby avoids finger occlusion. Here we demonstrate this advantage in three commonly used mobile apps.

**Gaming** Mobile gaming is probably where the user interaction suffers the most from finger occlusion. In a mobile game, the user must constantly move their fingers on the screen while paying attention to the gaming content shown on the screen. The occlusion is almost unavoidable. Yet, the occluded content (such as the enemy's location) may deserve the user's immediate attention to react in a timely fashion. Indeed, it is this dilemma that motivates many companies to make an external game controller for mobile devices.

BackTrack allows the user to control the game on the back of the device. For example, as shown in Fig. 22.1 and our supplementary video, in a shooting game that we implement, the movement of the aircraft is controlled by finger swiping on the back, and a bomb can be launched by double-tapping on the back. The screen, on the other side, is used exclusively for displaying the gaming content, with no occlusion.

**Web browser** Web browser on mobile devices relies on finger swiping to scroll up/down the webpage and tapping to click a hyperlink. Not only does the finger swiping occlude the displayed webpage, it may also cause accidental clicks on undesired hyperlinks. With BackTrack, the user can scroll a page by swiping on the BoD trackpad, and the front screen is only used for displaying and finger tapping on hyperlinks. In this way, the displayed webpage is less likely to be occluded; there is no confusion between scrolling and tapping. Moreover, the user can swipe left and right on the backside to go forward and back a page, again with no occlusion (see supplementary video).

**Photo capture** Today's camera app has a rich set of features. It allows the user, while capturing the photo, to adjust the field of view, set the focused region, change the contrast, apply a filter, to name a few. Many of these features require fingers, or even two hands, to operate on the screen, which at the same time displays a preview of the operation (e.g., the image filter effect). Therefore, it is almost unavoidable for the finger to occlude the preview.

BackTrack offers a natural place to perform some of these realtime adjustments without occluding the screen. For example, field-of-view zoom in/out can be performed by swiping up/down on the BoD trackpad, and image filter effects can be selected by swiping left/right on the back (see Fig. 19.1).

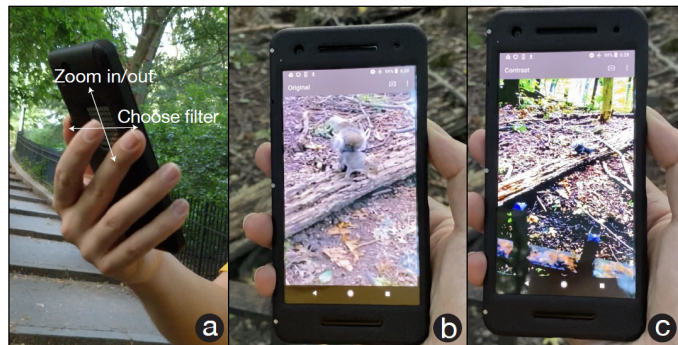


Figure 19.1: **Camera app.** With BackTrack, the user can swipe up/down on the BoD trackpad to zoom in/out (b), while swiping left/right to select a filter effect (c). Throughout, the user can see the image preview shown on the screen without any occlusion.

## 19.2 BoD Authentication

The very first operation after someone picks up their phone from the pocket is to unlock it. Indeed, mobile authentication is needed in numerous use scenarios, from unlocking the screen to making a payment. A common way of authentication is by inputting a PIN number. Today, even with the rise of other alternatives—such as fingerprint and face ID—PIN numbers continue serving as a reliable means of authentication, at least as a secondary choice when other alternatives fail to work: a fingerprint may not be accepted when the finger is wet, and face ID stops working when the user is wearing a mask (e.g., during the COVID-19 pandemic).

The problem of PIN authentication, however, is its vulnerability. It is not vulnerable as an authentication mechanism itself. Rather, it is vulnerable to simple attacks such as shoulder surfing. As suggested in many research [144], the PIN numbers typed on the front screen can be easily observed by a nearby person or camera.

Previous work [39] has suggested that authentication on the back of the device is more secure than inputting on the front screen. Such a BoD authentication can be easily implemented on BackTrack. Here we propose to split its trackpad area into four regions evenly, and finger tapping in individual regions represents different actions. In addition, we include four swiping actions (i.e., swipe left, right, up, and down). In total, there are eight distinct BoD actions. The authentication process requires the user to perform a certain number (4 or more) of actions sequentially. This sequence of actions is then mapped into a PIN number for authentication (see video for a demonstration).

## 19.3 Joint Interaction with Touches on the Front

So far, our demonstration uses BackTrack as an alternative to certain front-screen interactions to provide a better user experience. Here, on the other hand, we demonstrate the use of BackTrack in conjunction with the front screen interaction to enrich the expressiveness of finger motions. This is reminiscent of the keyboard augmentation of mouse interactions on a personal computer, where

holding the keys like **Ctrl** and **Shift** alters the semantics of the left-click and right-click mouse operations.

As a demonstration, we implement a file manager app (see the video). In this app, the user can open a file by tapping the file's icon on the front screen. To select the file without opening it, they can tap the file's icon while touching the *left* half of the BoD trackpad. Similarly, to rename the file, they can tap the file while touching the *right* half of the BoD trackpad. In this scenario, touching the left and right parts of the trackpad is semantically similar to pressing the **Shift** and **Ctrl** keys on a personal computer.

## Section 20: User Evaluations

We conduct user studies to understand the usability of BackTrack. In particular, we aim to address two questions: **i)** what type of finger motions the user can reliably perform on BackTrack? And **ii)** in specific applications, is BackTrack interaction preferable to front-screen interaction? 12 subjects (6 male, 6 female, aged 22 to 38, 2 left-handed) participated in these studies. They all have no experience of using BackTrack before the studies.

Our prototype is designed for the ergonomics of right-handed users—for example, the activation switch is placed on the right edge of the phone, and the BoD trackpad is positioned slightly toward the left to better host the index finger on right hand (see Fig. 22.1). We therefore ask the left-handed participants to use their right hand to interact with our BackTrack prototype, although this layout can be easily flipped to adapt to left-handed users.

### 20.1 Finger Motion Performance

Unlike the front-screen interaction, finger motions on the back of the device lack visual feedback; the user has little sense of where their finger touches. Further, the user's finger, while positioned on the back holding the device, has a limited range of motion. Thus, only a small area of BoD trackpad is needed and positioned at where the index finger is. Our first user study seeks to understand to what extent the user can perform finger actions on BackTrack.

We consider two types of commonly used finger motions on a trackpad, tapping and swiping. Our study consists of four tasks, including **i)** tapping a point on a  $2 \times 2$  grid, **ii)** tapping on a  $3 \times 3$  grid, **iii)** swiping along horizontal and vertical directions, and **iv)** swiping along horizontal, vertical, and diagonal directions (see Fig. 20.1-a,b,c,d).

Since all the subjects had never used BackTrack, when the user study begins, we inform the

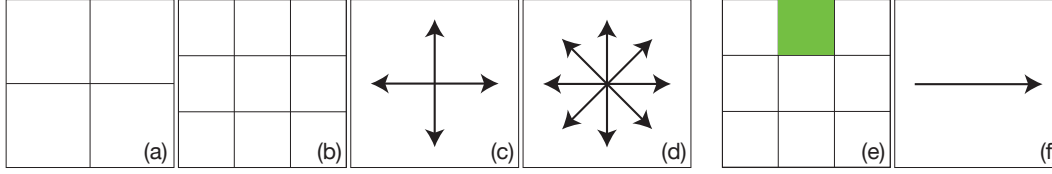


Figure 20.1: **Finger motions in user study I.** (a-d) We ask the subjects to perform four tasks. For example, the first task (a) is to tap one of the  $2 \times 2$  regions on the trackpad, and the fourth task (d) is to swipe along one of the eight directions. (e-f) We display on the front screen what finger motion to perform. Every time the target finger motion is randomly picked.

subjects that “there is a trackpad on the back of the phone” and ask them to use only their right hands and adjust their grip until they feel comfortable to operate with their index fingers. In each of the four tasks described above, we use the front screen to display what finger motion to perform (see Fig. 20.1-e,f). The subject then performs the required motion on the BoD trackpad. Each task repeats 50 times; every time a target finger motion is randomly chosen and displayed. Meanwhile, the front screen displays text which tells the subject if their *last* finger motion has been successfully received by the system. This is not a realtime feedback, but a feedback as an afterthought; it mimics the type of feedback that the user receives when using BoD interactions in a real-world app.

After finishing all four tasks, we ask the subjects to practice those tasks freely for 3 minutes, and then re-evaluate all the tasks. This is to understand how quickly the user can get familiar with our BackTrack interface.

**Results** We report the results in Fig. 20.2-a, where the accuracy (y-axis) is measured as the ratio of successfully recognized finger motions to the total number of motions. We found that with only 3-minutes of practice, the accuracies of all four tasks are improved significantly. This suggests that BackTrack is indeed easy to learn. Moreover, the accuracies of 4-way tapping and 4-way swiping are considerably higher than 9-way tapping and 8-way swiping (i.e., 99.16% vs. 88.54% for tapping, and 100% vs. 92.17% for swiping in the second trial). This suggests that the mobile app should use 4-way tapping and 4-way swiping as primary ways for the user interacting with BackTrack.

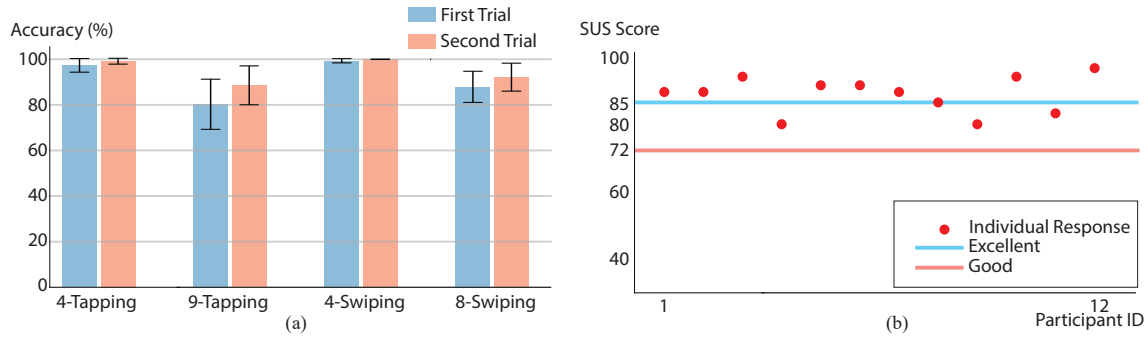


Figure 20.2: **User study results.** (a) For each of the four tasks in the first study, we report the finger motion accuracy measured as the ratio of successfully recognized finger motions to the total number of motions. The blue bars indicate human subject performance for the first time they use BackTrack, and the red bars show their performance after 3-minutes of practice. (b) The System Usability Scale (SUS) results reported by individual subjects.

## 20.2 Usability

After the first user study, we ask the subjects to use the mobile apps described in §19. In the browser app, the subjects are asked to browse the Wikipedia website and click hyperlinks they are interested in. In the camera app, the subjects are asked to capture images in an indoor environment with the freedom to adjust the field of view and apply image filters. And when playing the shooting game, the subjects are told to survive as long as possible. Each app is played for 5 minutes. Afterward, we ask the subjects to fill out the standard System Usability Scale (SUS) form [22].

**Results** The results are reported in Fig. 20.2-b. According to the SUS evaluation scale designed by Bangor et al. [10], BackTrack receives an average score of 87.5, which surpasses the standard score (85.58) for “excellent” usability. Only P4, P8, P9, and P11 rated our system below the “excellent” usability but still above the “good” usability.

Moreover, subjects generally agree that BackTrack is easy to use (10/12 strongly agrees), and 11 of them believe they will use it frequently (7/11 strongly agrees). In addition, subjects also believe that most users will learn to use BackTrack very quickly (8/12 strongly agrees). This again suggests that BackTrack is easy to use and requires little effort to learn.

**Discussion** Interestingly, we observe that some subjects, when using the web browser, tend to use not the BoD trackpad but the front screen to scroll up/down. But when playing the shooting game, no subject attempts to use the front screen for controlling their aircraft. We hypothesize that user habit is a factor here. When the user uses an app that they have been used to, they tend to use it in a way they are already familiar with. For apps that they have never used before, the user is more willing to adapt to the new ways of interaction.

We also ask the subjects for additional comments. P8 says “It would be nice to have the ability to adjust the position of the switch, as people may have different grip preferences and hand sizes”. Since our switch is as simple as a piece of copper foil (§18.3), we believe that there is better design of the phone case that would allow the user to adjust where the switch is—a subject for future research.

## **Section 21: Discussion**

BackTrack also shares the same limitations that other capacitive sensing techniques have. For example, it may not work properly when the user's fingers are wet or wear an insulator, such as a latex glove. It also lacks haptic feedback and does not yet support multitouch. It would be interesting to address these limitations as a future work.

To conclude, we have introduced BackTrack, a new sensing technique for 2D finger tracking on the back of the device. BackTrack is battery-free, requires no wireless or port connection to the device and no modification of the operating system. It can be readily used with any off-the-shelf mobile devices. In addition, we have presented a novel switch design that prevents unintentional, accidental touches on the BoD trackpad. Our switch is also battery free, and consumes almost no space. Our BackTrack design is guided by a capacitive circuit model, justified by physics-based simulation as well as controlled laboratory experiments, and evaluated in human subject studies.

## **Chapter 4**

# **MoiréBoard: Low-cost and Accurate Camera Tracking for Mobile VR**

## Section 22: Introduction and Related Work

Numerous HCI applications rely on a common building block: tracking a device’s spatial location in a fast and accurate way. Perhaps the most demanding use of device tracking is Virtual Reality (VR) and Augmented/Mixed Reality applications. Many commercial VR systems utilize dedicated hardware, such as laser and infrared beacon stations, to enable accurate device tracking, but often at high cost. For example, the Valve Index [35] and HTC Vive [41] system support device tracking with only a few millimeters deviation, but the cost for the entire system is nearly \$1,000 US dollars.

Other systems such as Google Cardboard aim to offer the user VR experience at low cost. Often repurposing the mobile phone as a VR headset without introducing additional hardware, these systems track the device using the on-device inertial measurement unit (IMU). The IMU sensor provides sufficient accuracy to track the device’s orientation when used in fusion [82]—some of the commercial VR devices even solely use IMU sensors to recover device orientation [88]. However, when it comes to position tracking, the IMU sensor falls short. This is because the sensor measures only the device’s acceleration, and requires double time integration to recover its position. As the tracking period increases, the time integration suffers from an increasingly larger drifting error.

Another option is to use visual markers [50], such as a checkerboard pattern, displayed on a planar area in the scene. When a camera captures the visual markers, camera location can be recovered from the pixel coordinates of the detected markers. The markers are fully passive, require no power, and cost as little as a piece of paper. Yet, the tracking accuracy is not on par with the active methods (such as the laser-emitting beacon-based solution [35, 34]). When the distance between the camera and visual markers reaches a few meters, the tracking error is on the order of centimeters, too large for many VR applications.

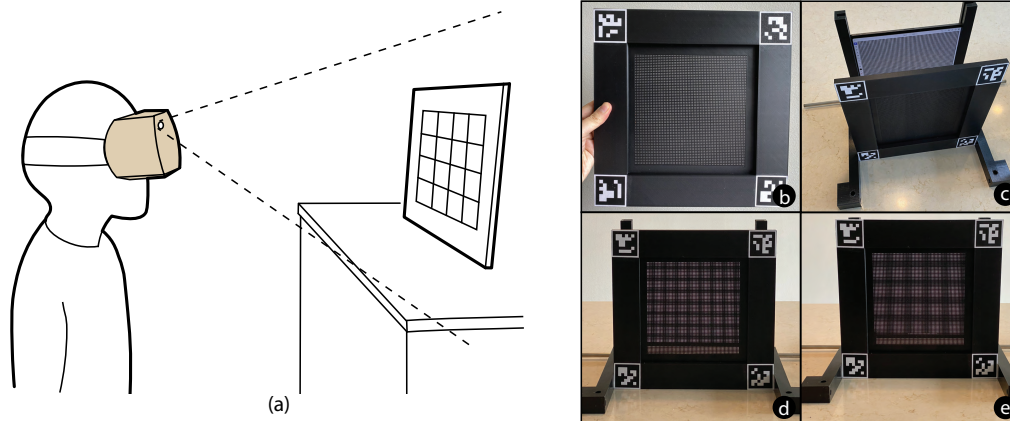


Figure 22.1: **MoiréBoard tracking.** (a) A typical application scenario: in a low-cost VR system such as Google Cardboard, MoiréBoard can be used to track camera position and provide tracking accuracy comparable to state-of-the-art VR tracking systems that are much more expensive. (b) 3D printed prototype: The total dimension is 30cm×30cm, with a 20cm×20cm fringe pattern area in the center. (c) We use an LED display (i.e., an iPad) to display the base layer of gratings, while the front layer is a 3D printed grating structure. The use of LED display is merely for the convenience of experiments (e.g., to test different design parameters). It can be replaced with another 3D printed structure or a printed paper. (d, e) When the camera views at different positions, the fringe pattern appears differently, providing clues for accurate position tracking. (Note: This figure may have aliasing artifacts when viewed on a computer screen due to PDF rendering and reduction of the image dimension. We recommend the reader to zoom-in the PDF and view it.)

All these solutions seem to suggest a performance-cost trade-off that one must take reluctantly. In this work, we challenge this trade-off. We propose a simple visual marker, called MoiréBoard, that enables accurate camera tracking at low cost. Our approach is as accurate as the beacon station system used in high-end VR devices, while maintaining a low cost—the markers can be easily made (e.g., using a 3D printer), and the tracking algorithm is computationally efficient, allowing for tracking at high frame rate (e.g., at 120FPS).

Our idea is to harness a seemingly irrelevant visual phenomenon, the *moiré effect* [18]. When two repetitive line gratings with different periods overlap, a bright-and-dark fringe pattern emerges at a larger period due to the interference of the two gratings (see Fig. 22.2). A remarkable ability of this fringe pattern is the amplification



Figure 22.2: **Moiré effect.** A fringe pattern emerges when two line gratings with different periods overlap.

of position changes: a subtle shift of the relative position of the two gratings will cause a notable change of the fringe pattern (see Fig. 22.1). In our MoiréBoard design, the two gratings stay fixed, but their projections on the camera’s image plane change as the camera moves. As a result, the camera captures a fringe pattern whose change can be easily discerned and measured.

Through a systematic analysis of this process, we derive a formula to recover the camera position from a detected fringe pattern. Unlike other visual-marker-based approaches which require camera calibration to recover intrinsic parameters (such as focal length) before the tracking starts, our formula is calibration-free. Our analysis further informs MoiréBoard’s design parameters—such as its physical size and grating periods. Our choice of these parameters is well justified by a theoretical accuracy analysis. Lastly, we prototype MoiréBoard using 3D printing, by which we compare our method with existing tracking methods, and demonstrate its accuracy, performance, and easiness to use.

## 22.1 Camera Tracking

The position tracking of VR/AR headsets has been extensively studied in the past few decades. Here we review the most relevant methods to MoiréBoard, namely optical tracking methods, while briefly discussing other methods that emerged recently.

**Optical tracking.** Currently, most commercial VR devices track headsets and controllers using optical signals. For example, the Valve Index [35] and HTC Vive [41] use the *Lighthouse* tracking system, relying on multiple laser base stations that emit infrared light beams. VR headsets and controllers then use the information of received IR signals with sensor fusions to estimate their 3D locations. According to Niehorster et al. [111], this is by far the most accurate tracking system used in commercial VR devices. However, due to the need for dedicated laser hardware, this system is very expensive. Sony PlayStation VR [99] adopts a different solution; the VR headsets and controllers emit visible light, which is received by a standalone camera at a fixed position; the camera then computes the device’s position. This approach reduces the cost of the system but

is not as accurate as the Lighthouse system. In addition, it may not work well in the presence of strong ambient light, which reduces the signal-to-noise ratio received by the camera.

To avoid the use of external beacon stations (and thereby reduce cost), some VR systems adopt Simultaneous Localization and Mapping (SLAM) [141, 129, 160, 25] for camera tracking, including the Oculus Quest 2 [45], Microsoft HoloLens/HoloLens 2 [105], and Magic Leap One [102]. Relying only on the camera capturing images, the SLAM algorithm aims to recover the scene structure and camera location simultaneously. However, the performance of SLAM largely depends on the scene structure, surface texture, and light condition [141]. Thus, it is not as reliable as beacon-based methods.

Instead of using dedicated commercial VR devices, many projects in both industry [52, 128] and academia [119, 4, 62] aimed to convert existing mobile devices to VR headsets at low cost. The most notable example is Google’s Cardboard platform [52]. On mobile devices, IMU [88, 82] sensors are often used to track the device’s orientation. But when it comes to tracking the device position, IMU sensors suffer from drifting errors. Mobile VR systems can also use SLAM for position tracking [107]. However, apart from the limitations of the SLAM technique in general, mobile devices often have more restricted computational budget, that further limits the frame rate of SLAM algorithm.

Another potential solution for mobile VR tracking is to use visual markers [50, 48, 40, 114, 74] displayed on an LCD screen or printed on a paper. Provided an image capturing visual markers, the tracking system first detects pixel coordinates of the visual markers, whose layout in 3D space is predefined. With this information, the camera position and orientation can be estimated using the *Perspective-n-Point* (PnP) [6, 93] algorithm. The PnP algorithm requires full knowledge of the camera’s intrinsic parameters (such as focal length), and thus the camera must be pre-calibrated [167]. The estimated camera position is sensitive to the detected marker locations. Even a deviation of few pixels would result in centimeter-level camera position error [72].

Our MoiréBoard can generally be viewed as a visual-marker-based tracking method. From this perspective, most relevant to our method is the work by Armstrong et. al [8, 152] called *Moiré*

*Phase Tracking* (MPT)<sup>1</sup>. The author introduced a visual marker composed of a glass substrate with printed film artwork bonded to both sides. Similarly, Tanaka et. al [142, 143] later introduced a visual marker design based on a microlens array. Both methods used the moiré effect—as the camera position changes, the captured marker pattern will also change, offering a clue for camera position estimation. Yet, lacking a systematic analysis of the pattern formation under the lens refraction, these methods relied on least square fitting for pose estimation. Thus, its tracking accuracy is limited.

On the contrary, we provide a full analysis of the geometry of moiré pattern under camera projection, allowing different applications can choose optimized moiré parameters based on our theory. Our method is accurate, comparable to the commercially available Lighthouse tracking system. Besides, it remains fully passive and low-cost. It is also computationally lightweight, allowing for tracking at a high frame rate. All these traits render it suitable for mobile VR applications.

**Acoustic tracking.** Recent years have also witnessed a significant rise in research interests on device tracking with acoustic signals. Although this is not directly related to our work, we refer the reader to the recent survey [98] and briefly highlight several recent works in HCI, including CAT [103], SoundTrack [165] and Millisonic [146]. The latter achieves the tracking accuracy of several millimeters, promising for VR tracking. However, these methods require additional acoustic hardware such as a microphone/speaker array, and they may suffer from drifting errors as the operating time increases [146]. Also, acoustic signals are generally prone to environmental noise; therefore, they are less reliable than optical signals. So far, most of the commercial VR tracking systems use optical tracking solutions.

---

<sup>1</sup>A commercialization of this work can be found in <https://www.metriainnovation.com/moire-phase-tracking>

## Section 23: Camera Tracking using Moiré Effect

We now present the core idea of MoiréBoard. Our approach is built on careful analysis of how moiré fringes are formed under a camera view (§23.1~23.3). Gaining insight from the analysis, we propose the design of a moiré pattern and a computational algorithm (§23.4). Together, they enable stable and accurate camera motion tracking.

### 23.1 The Geometry of Moiré Fringes

Moiré fringes from two superposed layers of repetitive structures have been well studied, and a general theory exists [18, 63]. Here to pave the way toward our core analysis (in §23.2), we introduce the geometry of a special type of moiré fringes.

The moiré fringes we consider here emerge from two superposed grid structures, which we refer to as grid A and grid B, respectively. Each grid can be viewed as a combination of two perpendicular line gratings (along  $x$ - and  $y$ -direction, respectively), and let the grating periods of grid A and grid B be  $T_a$  and  $T_b$  (see Fig. 23.1). Here we assume  $T_a > T_b$  without loss of generality. When the two grids superpose on each other, the interference between the gratings forms another grid pattern, generally known as the moiré *fringes*. It has been shown that the period  $T_m$  of the fringes (i.e., the grid cell size in Fig. 23.1a) depends on  $T_a$  and  $T_b$  in the following way,

$$T_m = \frac{T_a T_b}{T_a - T_b}. \quad (23.1)$$

This relation reveals an interesting property of moiré fringes: when  $T_a$  and  $T_b$  are close,  $T_a - T_b$  is so small that the fringe period  $T_m$  is much larger than  $T_a$  and  $T_b$  (due to the division in (23.1)). Remarkably, this property leads to an amplification of the relative motion between grid A and B, described as follows.

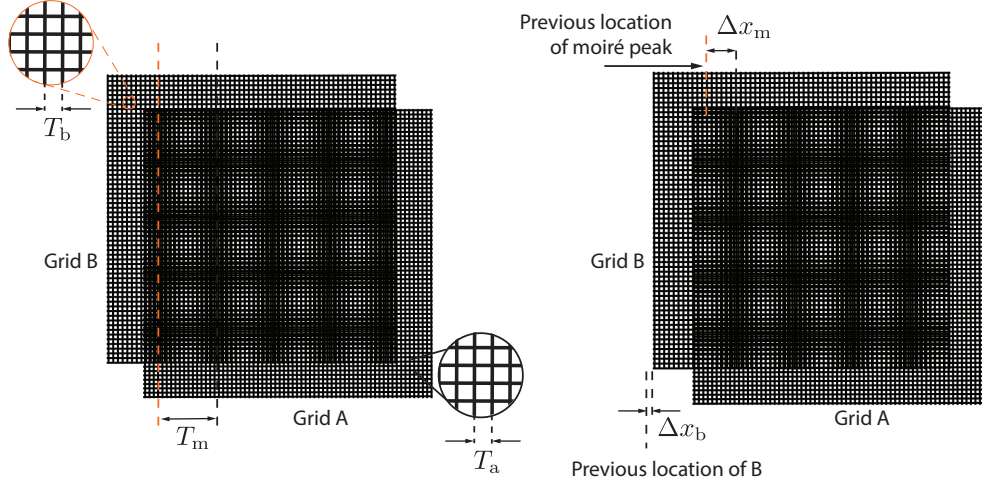


Figure 23.1: **Moiré fringes.** (left) Two grid patterns with grid size  $T_a$  and  $T_b$  are overlapped. The resulting moiré pattern is another grid pattern with grid size  $T_m$ . (right) Grid B is moved by  $\Delta x_b$  while grid A stays fixed. This causes the moiré pattern to be shifted by  $\Delta x_m$ , an amount much larger than  $\Delta x_b$  and thus much easier to detect robustly. Note that in this case the moiré fringes along the  $y$ -direction stay unchanged. Thereby, we can track camera motion along  $x$ - and  $y$ -directions separately.

Suppose grid B is translated by its period  $T_b$  while grid A stays fixed. After the translation, grid B appears the same as before due to its periodicity. Thus, the moiré fringes must be shifted by an entire period  $T_m$ . Formally, a small shift  $\Delta x_b$  of grid B will cause a large movement of the moiré fringes by the amount  $\Delta x_m$ , and the amplification ratio of their motions is

$$\frac{\Delta x_m}{\Delta x_b} = \frac{T_m}{T_b} = \frac{T_a}{T_a - T_b}. \quad (23.2)$$

Inspired by Eqs. (23.1) and (23.2), our idea is to harness moiré fringes as an amplifier lens, from which we can track even the subtle motion of grid B with high accuracy. The next step is to bridge the shift of grid B with camera motion so that we can track the camera.

## 23.2 Moiré Pattern under Camera Projection

We now examine moiré fringes under camera projection and motion—an analysis that to our knowledge has *not* been explored. The goal is to reveal how the camera motion can be recovered through the camera captured moiré fringes.

The camera projects a 3D point  $\mathbf{X}$  onto a 2D point  $\mathbf{x}$  on the image plane. In computer vision [6], this projection is expressed as  $\mathbf{x} = \mathbf{K} [\mathbf{R}|\mathbf{t}] \mathbf{X}$ , where  $\mathbf{X} = (x, y, z, 1)^T$  is a 3D point expressed by its homogeneous coordinate in the world frame of reference;  $\mathbf{x} = (x, y, 1)^T$  is the 2D homogeneous coordinate of the projected point on the image plane.  $\mathbf{X}$  is first transformed into the camera's local frame of reference by a  $3 \times 4$  matrix  $[\mathbf{R}|\mathbf{t}]$ , and then projected by a  $3 \times 3$  matrix  $\mathbf{K}$ . The transformation matrix  $[\mathbf{R}|\mathbf{t}]$ , known as the camera's *extrinsic* matrix, depends on the camera's orientation and location in 3D space. The  $\mathbf{K}$  matrix, known as the *intrinsic* matrix, encodes the camera's focal lengths and optical center, namely,

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \quad (23.3)$$

where  $f_x$  and  $f_y$  are the focal lengths along  $x$ - and  $y$ -direction of the image plane, respectively; and  $(c_x, c_y)$  indicates the camera's optical center.

In this work, we aim to recover camera position  $\mathbf{t}$  given a moiré fringe image, without the knowledge of camera intrinsics (i.e.,  $\mathbf{K}$ ). For recovering the rotation  $\mathbf{R}$ , we use the IMU sensors equipped on mobile devices. This is because IMU sensors offer sufficient accuracy for orientation tracking [88].

To start our analysis, consider two moiré layers: the first one, grid A, is located on the plane  $z = 0$ , and the second one, grid B, is placed on the plane  $z = h$  (see Fig. 23.2). Assume for now that the camera is located at  $(0, 0, t_z)$ , looking straight toward the  $z = 0$  plane (this assumption will be relaxed shortly). In this case, the camera's rotation matrix remains identity (i.e.,  $\mathbf{R} = \mathbf{I}$ ), and the grating period  $T_a$  on grid A appears on the captured image with the length  $\tilde{T}_a = T_a \frac{f}{t_z}$ , where  $f$  is the camera's focal length (assuming  $f_x = f_y = f$  for the simplicity of presentation). Similarly, the projected grating period  $\tilde{T}_b$  of grid B on the captured image is  $\tilde{T}_b = T_b \frac{f}{t_z - h}$ . Applying Eq. (23.1),

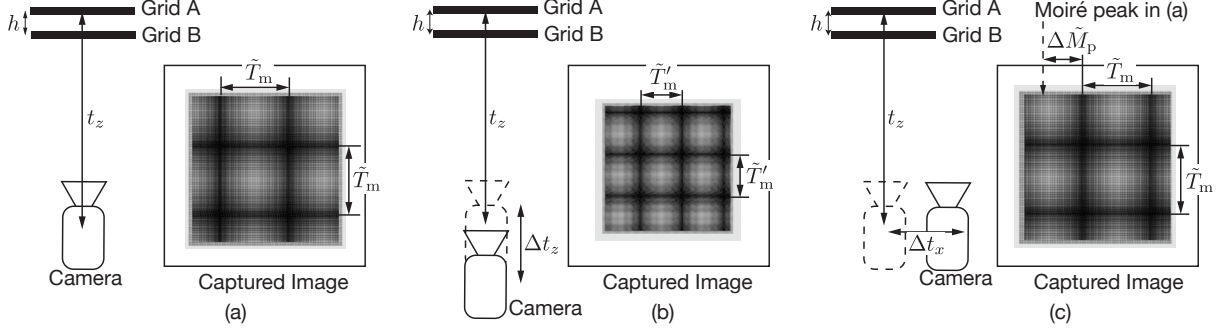


Figure 23.2: **Moiré fringes under camera projection.** In each subfigure, the left is a top-down view of the imaging setting, and the right is the captured image. **(a)** The camera looks straight towards the MoiréBoard. The captured moiré fringes have the period  $\tilde{T}_m$ . **(b)** When the camera moves further away, the period of captured moiré fringes changes into  $\tilde{T}'_m$ . **(c)** When the camera moves horizontally (starting from (a)), the captured moiré fringes are shifted by  $\Delta\tilde{x}_m$ , while their period remains unchanged.

we obtain the period of moiré fringes on the captured image (see Fig. 23.2), namely,

$$\tilde{T}_m = \frac{T_a T_b f}{T_a h - T_a t_z + T_b t_z}. \quad (23.4)$$

Suppose the camera is translated along  $x$ -direction to  $(t_x, 0, t_z)$  (see Fig. 23.2c). Under the camera projection, grid A and B will be translated by different amounts, as they are at different distances from the camera, namely  $\Delta\tilde{x}_a = -t_x \frac{f}{t_z}$  and  $\Delta\tilde{x}_b = -t_x \frac{f}{t_z - h}$ . This amounts to fixing grid A but shifting grid B by  $\Delta\tilde{x}'_b = \Delta\tilde{x}_b - \Delta\tilde{x}_a$ . According to Eq. (23.2), the moiré fringes on the captured image will be displaced by

$$\Delta\tilde{x}_m = \frac{t_x T_a f}{T_a h - T_a t_z + T_b t_z}. \quad (23.5)$$

Note that depending on the camera position  $t_z$ , the denominators in Eqs. (23.4) and (23.5) may be either positive or negative, and so are  $\tilde{T}_m$  and  $\Delta\tilde{x}_m$ . A positive  $\Delta\tilde{x}_m$  indicates that the fringes shift toward  $+x$  direction on the image plane as the camera moves toward  $+x$  direction in space; a negative  $\Delta\tilde{x}_m$  indicates fringes moving opposite to the camera's displacement. Also at a certain  $t_z$ , the denominators vanish. We will return to discuss this subtlety in §23.4.

Equations (23.4) and (23.5) lay the foundation of our camera tracking algorithm: in Eq. (23.4),

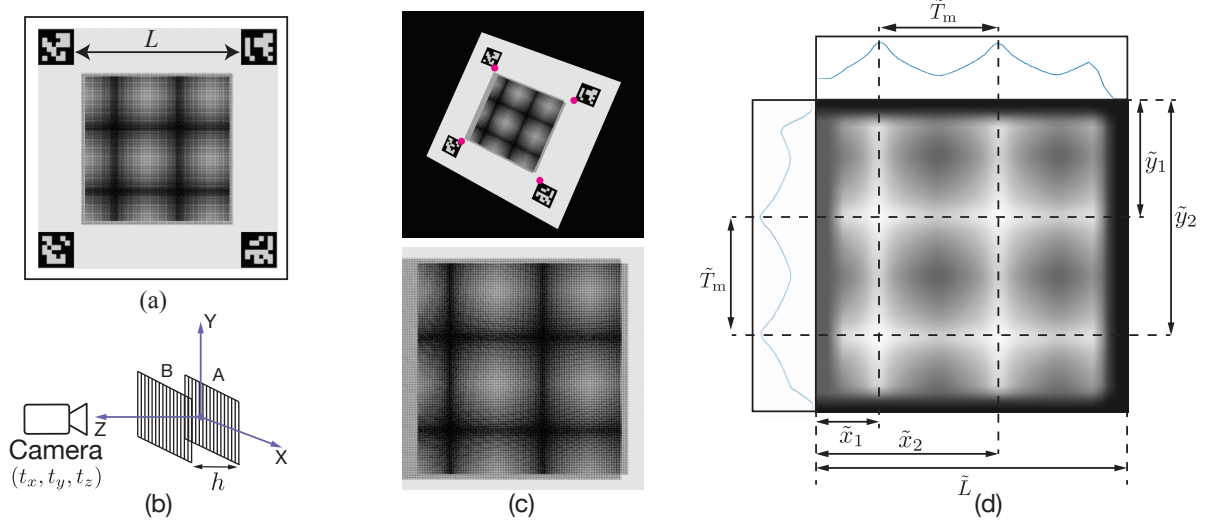


Figure 23.3: **Steps of our tracking algorithm.** (a) A MoiréBoard in 2D frontal view. (b) Coordinate system of our algorithm. Grid A is located at plane  $z = 0$ . Grid B is at  $z = h$ . (c) Top: An image of the MoiréBoard captured at a certain camera location. Inner corners of the four visual markers are detected and labeled as red dots. Bottom: We rectify perspective distortion based on the detected corner positions to convert the image into a frontal view image. (d) We apply a Gaussian blur filter to the image of (c)-bottom, and then measure  $\tilde{T}_m$ ,  $\tilde{x}_i$  and  $\tilde{y}_i$  on the resulting image.

$T_a$ ,  $T_b$ , and  $h$  are moiré grids' parameters known *a priori*;  $\tilde{T}_m$  can be measured on the captured image. If the focal length  $f$  is known, we can solve for  $t_z$ . Afterwards, we measure  $\Delta\tilde{x}_m$  from two captured images, and recover  $t_x$  using Eq. (23.5). However, to fully materialize this idea, two questions remain unanswered: (i) how to account for tilted camera (i.e.,  $\mathbf{R} \neq \mathbf{I}$ )? (ii) how to eliminate the need of the focal length  $f$  so that our camera tracking is calibration-free? We address both questions next.

### 23.3 Calibration-free Camera Tracking

Traditional camera tracking, whether marker-based [50, 48, 40] or markerless [33, 107], often require a calibration step to recover camera intrinsics (i.e.,  $\mathbf{K}$ ) before tracking starts [167, 61]. Not only does this calibration impose an additional step, it necessitates re-calibration whenever the camera's intrinsic parameters change (such as refocus). Our approach, at first glance, follows the same route, as both Eqs. (23.4) and (23.5) require the knowledge of the camera's focal length  $f$ .

**Elimination of  $f$**  A key observation to liberate our approach from calibration is that  $f$  simply scales  $\tilde{T}_m$  and  $\Delta\tilde{x}_m$  in Eqs. (23.4) and (23.5). Indeed, under camera projection, the physical distance between any two points (on a frontal plane) is scaled by  $f$ . In particular, if we place four visual markers on grid A—which are needed anyway to locate moiré fringes in a captured image (see Fig. 23.3a)—the physical distance  $L$  between two markers has an image-plane length  $\tilde{L} = f \frac{L}{t_z}$ . Since we can easily measure  $\tilde{T}_m$ ,  $\Delta\tilde{x}_m$ , and  $\tilde{L}$  from captured images, we take the ratios  $\alpha = \tilde{T}_m/\tilde{L}$  and  $\beta_x = \Delta\tilde{x}_m/\tilde{L}$ , which eliminate  $f$  from Eqs. (23.4) and (23.5). As a result, the camera position coordinate  $t_z$  and  $t_x$  can be recovered using the following formulas,

$$t_z = \frac{\alpha L T_a h}{T_a T_b + \alpha L (T_a - T_b)} \quad \text{and} \quad t_x = \frac{L}{T_a h} (\beta_x + n\alpha) (T_a h - T_a t_z + T_b t_z), \quad (23.6)$$

where  $n$  is an integer arising from the moiré fringes' periodicity: two displacements  $\Delta\tilde{x}_m$  and  $\Delta\tilde{x}_m + n\tilde{T}_m$  lead to the same fringe appearance on the image. In practice, we disambiguate the measurement of  $\Delta\tilde{x}_m$  by exploiting the camera motion's temporal coherence: choose an integer  $n$  such that the resulting  $t_x$  is closest to  $t_x$  from the last camera frame. (for the very first frame, we simply use  $n = 0$  and  $\beta_x = 0$ , which lead to  $t_x = 0$ ). In short, no camera's intrinsic parameters appear in Eq. (23.6), and thus no camera calibration is needed to recover  $t_x$  and  $t_z$ . To recover the  $y$ -coordinate  $t_y$ , we follow the same process as the  $t_x$  recovery but measure  $\beta_y$  along the image's  $y$ -direction.

*Remark.* The recovered camera position  $(t_x, t_y, t_z)$  is in a world frame of reference, with the unit that we use to measure  $L$  (such as millimeter). The world frame of reference is defined as follows. Its  $x$ - $y$  plane is the plane of grid A; its  $z$ -axis points toward grid B; and its origin is the camera's initial position (at frame 0) projected on the plane of grid A.

**Fringes under tilted camera** Our analysis so far assumes that the camera faces straight toward the MoiréBoard's grid plane. When the camera is tilted, the captured fringe pattern becomes distorted, not a square grid anymore. Interestingly, our analysis shown in Fig. 23.4 suggests a simple way to factor out the effect of camera rotation:

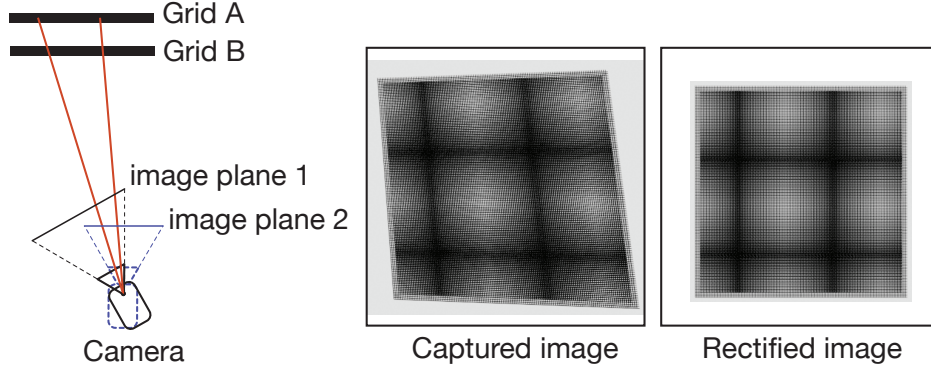


Figure 23.4: **Image rectification.** (left) Our key insight is that the moiré fringes (e.g., the peak bands indicated by the red lines) depend only on the positions of grid A and B and the camera location, but not the camera rotation. As the camera tilts, while its image plane has varying orientation, it captures the same set of fringe lines. This is a standard multi-view projection problem in computer vision [6]. One can construct a homography matrix to convert images across different views. In our algorithm, we convert the image captured by a tilted camera (middle) to the frontal view image (right).

From the captured image, we detect the position of the four visual markers, which are physically located on grid A to indicate the corners of a square region (Fig. 23.3a). On the image, however, the region indicated by the markers may not be a square due to the tilted camera distortion. But given the image-plane marker positions (i.e., red dots in Fig. 23.3c), we can construct a homography matrix to rectify the marker positions, restoring the marked region, including its image content, back to a perfect square (see Fig. 23.3). According to our analysis in Fig. 23.4, the rectified image is the same as the one captured by a camera at the same location but facing straight to the MoiréBoard’s grid plane. From there, we can apply Eq. (23.6) to recover the camera location.

Note that the homography transformation can scale the marked region arbitrarily. But our algorithm is agnostic to the image scale, as we only need the dimensionless ratios  $\alpha$  and  $\beta_x$  (and  $\beta_y$ ) in Eq. (23.6), not pixel lengths on the image plane. In practice, we always scale the marked region to 1000px×1000px.

**Outline of tracking algorithm** We now summarize our camera tracking algorithm:

1. Capture an image from the camera (Fig. 23.3c-top).

2. Detect the four corners of the visual markers (see red dots in Fig. 23.3c-middle) on the image. In practice, we use ArUco [50] for locating the corners. The side length  $L$  of the four corners in physical space is known *a priori*.
3. Use the detected corners to rectify the captured image to a frontal view image (Fig. 23.3b-bottom); ensure the size of the marked square region to have  $1000 \times 1000$  pixels. We then measure  $\tilde{L}$ , the side length (in pixels) of the four detected corners on the rectified image (Fig. 23.3d).
4. Detect peak bands of the captured moiré fringes: First, we convert the rectified image to a grayscale image and invert it (to detect moiré layers' constructive interference bands), followed by applying a Gaussian filter (see Fig. 23.3d). Next, we average the pixel intensities along  $x$ - and  $y$ -directions, respectively, reducing the image into two 1D images (see plots in Fig. 23.3d). From each 1D image, we locate its local peaks. There exist many peak detection algorithms. In practice, we apply a threshold to each 1D image to identify short intervals around the peaks, and then for each interval take the average (1D) pixel position (weighted by pixel intensities).
5. We measure the following quantities on the 1D images: (i)  $\tilde{T}_m$ , the averaged distance (in pixels) between two consecutive peaks. (ii)  $\tilde{x}_i$  and  $\tilde{y}_j$ : the peak locations on the horizontal and vertical 1D images, respectively. Here, the subscripts  $i$  and  $j$  indicate individual peaks.
6. For every peak  $i$ , compute the displacement of peak locations, namely  $\Delta\tilde{x}_m$ , with respect to frame 0. This is done by subtracting  $\tilde{x}_i$  in frame 0 from its value in the current frame. We then use Eq. (23.6) to compute  $t_z$  and  $t_x$  of the camera location. In this process, we enumerate  $n$  in (23.6) from -5 to 5, and for all possible values of  $t_x$ , we choose the one closest to the  $t_x$  value of the previous frame.
7. Meanwhile, we take the detected peak locations  $\tilde{y}_j$  along  $y$ -direction, and apply the same process as step (6) to recover  $t_y$  of the camera position.

## 23.4 MoiréBoard Design

The geometry of MoiréBoard is specified by its parameters  $T_a$ ,  $T_b$ , and  $h$ . We justify the choice of these parameters through closer analysis of Eqs. (23.4) and (23.5). First, when the camera is sufficiently far from the MoiréBoard (i.e.,  $t_z$  is large), Equation (23.5) leads to a negative  $\Delta\tilde{x}_m$ , which indicates that the fringes move in a direction opposite to the camera's moving direction. On the contrary, a small  $t_z$  leads to a positive  $\Delta\tilde{x}_m$  value, and the fringes moves in the same direction as the camera's moving direction. In between, when  $t_z = \frac{T_a h}{T_a - T_b}$ , the denominators of (23.4) and (23.5) vanish. This is the point at which  $T_a$  and  $T_b$  appear to have the same length under camera projection, and therefore the fringe pattern disappears, as indicated by the fringe period  $\tilde{T}_m \rightarrow \infty$ .

Figure 23.5 shows the relation of  $\alpha = \tilde{T}_m / \tilde{L}$  (i.e., the dimensionless fringe period) with respect to  $t_z$ . Although varying widely,  $|\alpha|$  must be smaller than 0.5 in practice. Otherwise, the MoiréBoard—which has a finite size  $L \times L$ —may not be large enough to display two peak bands at the same time, but we need at least two peaks for the measurement of  $\tilde{T}_m$  (recall Fig. 23.3d). Meanwhile, a peak band on the image has certain thickness. Formed by constructive interference of grating lines and due to the use of Gaussian filter (in step (4) of our algorithm), it will never appear as a sharp line. As a result, the fringe period  $\alpha$  can not be too small. In practice, we ensure  $|\alpha| > 0.1$  to allow for robust measurement of  $\tilde{T}_m$ .

As illustrated in Fig. 23.5, when  $|\alpha|$  is in the range  $[0.1, 0.5]$ , the camera motion is restricted in either range 1 (blue) or range 2 (orange). Specific values of these  $t_z$  ranges depend on the MoiréBoard's design parameters. In Table 23.1, we list nine sets of design parameters (i.e.,  $T_a$ ,  $T_b$  and  $h$ ), and their corresponding  $t_z$  ranges (for  $\alpha \in [0.1, 0.5]$ ). These parameters are chosen for typical VR applications, wherein the headset moves within a few meters. In practice, one can choose a specific set of parameters depending on their application needs.

ID	$T_a(mm)$	$T_b(mm)$	$h(cm)$	Working range (m)	Direction
1	3.1	3	10	0.9 to 2.1	S
2	3.1	3	16	1.5 to 3.5	S
3	3.2	3	10	0.6 to 1.3 or 2.1 to 4.0	S or O
4	3.2	3	16	1.0 to 1.7 or 3.3 to 4.0	S or O
5	2.6	2.5	10	0.8 to 1.9	S
6	2.6	2.5	16	1.4 to 3.1	S
7	3.0	2.5	10	0.6 to 1.0	O
8	2.7	2.5	10	0.7 to 1.2 or 1.6 to 4.0	S or O
9	2.7	2.5	20	1.4 to 2.3 or 3.2 to 4.0	S or O

Table 23.1: **MoiréBoard design parameters.** We list a set of MoiréBoard design parameters and their working ranges. Here we limit the camera’s  $t_z$  range up to 4m, as it suffices for most VR applications. ‘O’ indicates that the captured moiré fringes will move in the opposite direction to the camera (due to negative  $\Delta\tilde{x}_m$  in (23.5)), while ‘S’ indicates that the fringes will move in the same direction as the camera. This table can serve as reference designs when we deploy MoiréBoard in specific applications.

### 23.5 Accuracy Analysis

MoiréBoard is an order of magnitude more accurate than the traditional marker-based tracking methods. We now analyze our approach to understand its benefits, which we will also experimentally confirm in §24.

All vision-based tracking algorithms, including ours, are designed to map camera displacement to certain measurable image-space changes [6]. For instance, in traditional marker-based approaches (such as [167]), if a camera facing straight to the marker plane undergoes a lateral displacement  $t_x$ , then the captured image features will be shifted by  $f \frac{t_x}{t_z}$ , where  $f$  is the camera focal length, and  $t_z$  is the distance from the marker to the camera. But using MoiréBoard under the same situation, the captured features (which are moiré fringes) will be shifted by  $\Delta\tilde{x}_m$  defined in (23.5).

To increase the tracking sensitivity, one must enlarge the image-space changes for a given camera displacement. In traditional methods, this demands a larger focal length  $f$  (to increase  $f \frac{t_x}{t_z}$ ), but a larger  $f$  narrows the camera’s field of view, thus limiting the camera’s tracking range. In our approach, however, we can enlarge  $\Delta\tilde{x}_m$  by choosing proper  $T_a$ ,  $T_b$  and  $h$  without increasing  $f$ , and thereby keep the camera’s field of view open. Concretely, we can compare the sensitivity

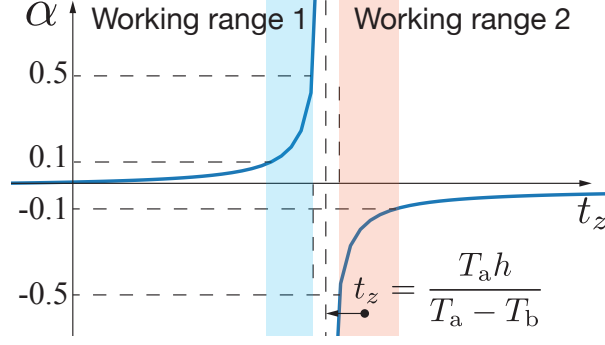


Figure 23.5: **The working range of MoiréBoard** is determined by  $\alpha$ : (i) we need at least two peak fringes to determine  $\tilde{T}_m$ , and this requires  $|\alpha| < 0.5$ ; (ii)  $|\alpha|$  must be large enough so that two fringe bands on the image plane can be clearly discerned, and this requires  $|\alpha| > 0.1$ . To satisfy these constraints, the camera's  $t_z$  must locate in either range 1 (blue region) or range 2 (orange region).

of our method with the traditional methods: under the same  $t_x$ ,  $t_z$  and  $f$ , the ratio  $\eta$  of the moiré fringe displacement to the image feature displacement in traditional methods is

$$\eta = |\Delta \tilde{x}_m| \frac{t_z}{t_x f} = \frac{|\alpha| L}{T_b}, \quad (23.7)$$

This sensitivity boost can be increased by using a larger MoiréBoard size  $L$ , and smaller grating period  $T_b$ , and it is independent from the distance between the camera and the MoiréBoard. To put it into context, consider a MoiréBoard with  $L = 300\text{mm}$  and  $T_b = 2.5\text{mm}$ . Recall  $|\alpha| \in [0.1, 0.5]$  from §23.4. According to (23.7), this MoiréBoard is **12~60×** more sensitive than the traditional methods for tracking camera motion in  $x$ - and  $y$ -directions.

## Section 24: Evaluation

**Prototyping details** There are many ways of making MoiréBoard. We prototype it in a simple way: 3D print the grating structure of grid B with a grating period  $T_b$ , and place it in front of an LED display (e.g., we use an iPad). The LED display merely serves as a quick way to display the grating structure of grid A, and can be replaced with other designs such as a piece of paper printed with the grating pattern. We use it for ease of experimenting with different grating structures (e.g., switch to a different  $T_a$ ). Figure 22.1 demonstrates the setup and the resulting moiré fringes.

### 24.1 Synthetic Scene Evaluation

Before evaluating MoiréBoard in real scenes, we test it in synthetic scenes, leveraging the full-fledged image rendering engine that can model camera and scene accurately. We use Blender [49] to set up a MoiréBoard (300mm×300mm) together with four visual markers, each of which has a size 50mm×50mm. We then use the Cycle rendering engine in Blender to synthesize photo-realistic images captured by a virtual camera (see Fig. 24.1). In this way, we know precisely where the camera is and can compute the locations of the moiré bands on captured images. With this ground-truth information, we can evaluate the accuracy at various stages of our algorithm.

$t_z$ (m)	Error (pixels)
1.0	$0.23 \pm 0.14$
1.5	$0.34 \pm 0.17$
2.0	$0.52 \pm 0.31$
2.5	$0.79 \pm 0.44$
3.0	$0.94 \pm 0.57$
3.5	$1.32 \pm 0.61$
4.0	$2.78 \pm 1.79$

Table 24.1: **Peak detection error** in pixels at different

$t_z$ .

**Peak detection accuracy** Our algorithm tracks the camera position by measuring image-plane peak locations (recall step (4) of our algorithm in §23.3). Thus, the tracking accuracy hinges on how accurately we detect the peaks. This motivates us to first examine the peak detection accuracy. To this end, we set up a MoiréBoard with  $T_a = 3.1$ mm,  $T_b = 3$ mm, and  $h = 100$ mm. We place the



Figure 24.1: **Synthetic images** rendered by Blender Cycle’s engine at four different camera positions. We set up a virtual scene and render camera captured images. In this way, we know the ground-truth camera locations, which allows us to evaluate our method’s tracking accuracy and compare it with the classic checkerboard-based approach.

virtual camera at different distances from the MoiréBoard, ranging from  $(0, 0, 1)\text{m}$  to  $(0, 0, 4)\text{m}$ , sampled every  $0.5\text{m}$ . At each camera location, we randomly rotate the camera 40 times, while ensuring that from each rotation the visual markers are visible. The camera images are rendered at a resolution of  $1920\text{px} \times 1080\text{px}$ . We then compare the detected peak locations with the ground-truth to evaluate the error.

Table 24.1 reports the peak detection errors at different  $t_z$  distances. For each  $t_z$ , we compute the average error of peak detection (in pixels) over all 40 camera rotations as well as the error deviation. The results show that the error becomes larger as  $t_z$  increases. This is because as the camera moves further away, the MoiréBoard appears in a smaller region on the image. When  $t_z < 3\text{m}$ , the detected peaks always have sub-pixel errors. This experiment suggests that a  $300\text{mm} \times 300\text{mm}$  MoiréBoard is suitable for camera tracking within a  $\sim 4\text{m}$  range, sufficient for most VR applications. For larger tracking ranges, one could use a larger size of MoiréBoard (e.g., a  $500\text{mm} \times 500\text{mm}$  MoiréBoard).

**Comparison with checkerboard-based tracking** With full knowledge of a scene setup, we can directly compare our method with other vision-based tracking methods. In particular, we consider the checkerboard-based tracking [167], which has become the standard tracking method in numerous applications, and has been implemented in many computer-vision toolboxes such as Matlab and OpenCV.

In this comparison, we use the same MoiréBoard setup as in the previous experiment. In checkerboard-based tracking, we use an  $8 \times 8$  checkerboard with the physical size  $400\text{mm} \times 400\text{mm}$ —the same size as our MoiréBoard plus its optical makers. The tracking algorithm is readily offered in

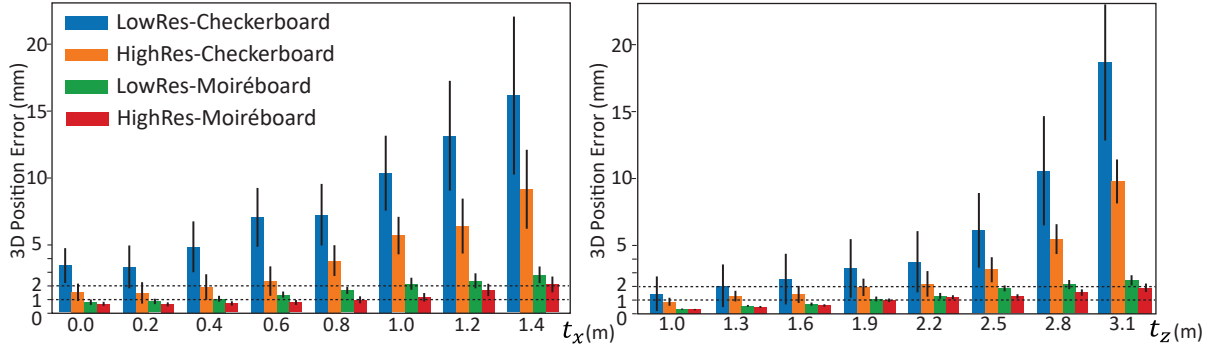


Figure 24.2: **Evaluation results on synthetic images. (Left)** Averaged 3D position error at different  $t_x$  ( $t_z=2m$ ). **(Right)** Averaged 3D position error at different  $t_z$ . We compare MoiréBoard to the checkerboard-based camera tracking, and test two image resolutions at high-res (1920px×1080px) and low-res (960px×720px). Thanks to the amplification effect of moiré fringes, our method is significantly more accurate than checkerboard. Its performance is also less affected by the image resolution.

the popular OpenCV library: we use `cv2.findChessboardCorners` to detect the pixel coordinates of checkerboard corners followed by `cv2.solvePnP` to obtain the camera position. The camera’s intrinsic parameters (such as its focal length) are known from Blender configuration.

We examine two scenarios: moving the camera along  $x$ - and  $z$ -directions. In the first scenario, we move the camera along  $x$ -direction from  $t_x = 0m$  to  $t_x = 1.4m$ , while fixing  $t_y = 0m$  and  $t_z = 2m$ . In this process, we sample camera locations every  $0.2m$ . In the second scenario, we move the camera from  $t_z = 1m$  to  $t_z = 3.1m$  and sample its position every  $0.3m$ . At every sampled location in both scenarios, we randomly choose 40 camera rotations and recover the camera location from both our method and the checkerboard method. To compare these two methods, we also consider two different image resolutions: *High-res* (1920px×1080px) and *Low-res* (960px×720px).

Figure 24.2 shows the 3D position errors at each camera position (averaged over 40 random rotations) resulted from both methods. This results indicate that **(i)** the tracking errors of MoiréBoard are an order of magnitude lower than the widely used checkerboard-based tracking; and **(ii)** MoiréBoard is much more robust to decreased camera resolution, thanks to its motion amplification through moiré effects.

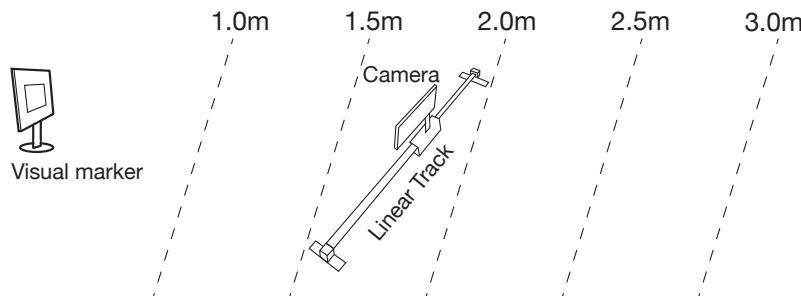


Figure 24.3: **Setup of real-scene evaluation.** The visual marker (i.e., MoiréBoard, checkerboard or Lighthouse base stations) is placed at a fixed location. The experiment area is divided into four regions. We repeat the experiments in each region by placing a linear slide that carries the camera moving along a random direction. Without knowing the camera’s ground-truth positions, we evaluate the tracking methods by measuring how well they recover the camera’s linear motion.

## 24.2 Real Scene Evaluation

Next, we conduct experiments in real scenes and compare MoiréBoard to Lighthouse 2.0 [35], the state-of-the-art commercial VR tracking system. Lighthouse 2.0 uses laser-emitting base stations to track the positions of VR headset and controllers. Unlike MoiréBoard, it is not a passive tracking system, and is much more expensive. We also compare our method to the checkerboard tracking method [167], as both are vision-based passive tracking methods.

In real scenes, all tracking methods produce errors, and the ground-truth camera positions are lacking. To measure the tracking errors, we leverage a motorized linear slide, which carries the camera (or VR headset) and translate it linearly in a controlled way. We place the linear slide along different directions. In each run, we collect a series of recovered camera positions and apply linear regression to fit a linear path. The error is measured as the average distance of all recovered position samples to the fitted linear path. In this way, no ground-truth camera positions are needed.

Our experiment setup is illustrated in Fig. 24.3. We fix the spatial location of MoiréBoard and checkerboard. When using a single base station of Lighthouse 2.0, we place it at the same location as the MoiréBoard and checkerboard. When using two base stations, we place the stations symmetrically: they are centered at the same location as the MoiréBoard and separated by 1 meter. To understand the tracking accuracy at different distances from the landmark (i.e., MoiréBoard,

checkerboard, or base stations), we divide the space into four regions (see Fig. 24.3). In each region, we place the linear slide along a random direction and evaluate the tracking errors of different methods (as described above); we repeat this process 10 times, each with a different slide direction.

The average tracking errors are reported in Table 24.3, where “MoiréBoard-1” and “MoiréBoard-6” indicate the use of MoiréBoard designs in the 1st and 6th row of Table 23.1, respectively. The results confirm that the tracking accuracy of MoiréBoard is an order of magnitude higher than the classic checkerboard-based approach. Our method is also much more accurate than Lighthouse 2.0 with a single base station. When Lighthouse 2.0 uses two base stations, both methods have comparable accuracy, although Lighthouse 2.0 requires power input to actively emit laser, and is much more costly. We also note that a single MoiréBoard has a more limited tracking range than Lighthouse 2.0 due to the reason discussed in Fig. 23.5. However, its tracking range can be easily extended by using two or more MoiréBoard designs (e.g., MoiréBoard-1 and MoiréBoard-6) in the same scene.

In terms of the computational cost, MoiréBoard only requires simple image processing operations before applying the tracking formula (23.6). Thereby, the algorithm is computationally lightweight, able to track at a much higher frame rate than the checkerboard approach (which needs to solve a PnP problem at every frame). The tracking frame rates are also reported in Table 24.3. We note that the image processing operations in our algorithm can be further accelerated using GPUs on mobile devices.

We also visualize the tracking errors of different methods in Fig. 24.4. From the plots, it is evident that **(i)** MoiréBoard tracking results match closely to the Lighthouse 2.0 with two base stations, and **(ii)** as the distance from the landmark increases, checkerboard-based tracking deteriorates quickly, while MoiréBoard stays closely with Lighthouse 2.0.

**VR application** In addition, we build a mobile VR application using an iPhone 11 Pro placed on a Head Mounted Device (HMD). We use the mobile phone’s camera to capture MoiréBoard in realtime and recover the HMD’s position. Together with the HMD’s orientation obtained from the

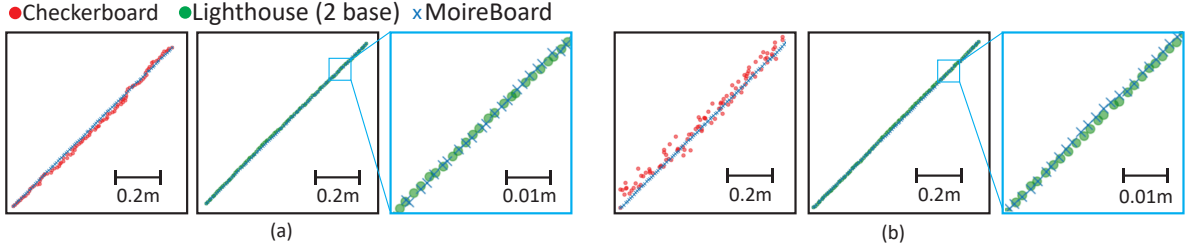


Figure 24.4: **Visualization of motion reconstruction.** Here we visualize the  $x$ - and  $z$ -coordinates of recovered camera positions in several experiments. Individual positions resulted from checkerboard, Lighthouse (with two base stations), and MoiréBoard are plotted in red, green, and blue, respectively, while the camera undergoes a linear motion. The cyan box shows a zoom-in view of the visualization. **(a)** Comparison in the  $t_z$  range of 1.0m to 1.5m. **(b)** Comparison in the  $t_z$  range of 2.5m to 3.0m. It is evident that the tracking results of MoiréBoard always stay closely to the state-of-the-art Lighthouse 2.0 system.

Tracking Method	Distance Range (m)				FPS
	1.0-1.5	1.5-2.0	2.0-2.5	2.5-3.0	
CheckerBoard	8.24	12.62	17.89	23.71	15.7
Lighthouse (1 Base)	5.62	7.98	9.03	13.31	60
Lighthouse (2 Base)	1.79	1.84	2.32	2.56	60
<b>MoiréBoard-1</b>	1.54	2.05	-	-	114.2
<b>MoiréBoard-6</b>	-	2.34	2.52	3.17	114.2

Table 24.2: **Real-scene tracking results.** We report the average tracking errors (in mm) of different methods as well as their tracking frame rate (in FPS). The checkerboard and MoiréBoard algorithms are implemented using iOS-OpenCV on an iPhone 11 Pro Max. Lighthouse algorithm is encapsulated in the hardware of Valve Index VR kit (Thus, its FPS may not reflect its true computational cost). Also note that our method can be further accelerated using the mobile device’s GPUs, since the performance bottleneck of our algorithm lies in the image processing tasks (such as applying the Gaussian filter), which are well suited for parallel processing.

phone’s IMU sensor, our VR application offers a full 6-DoF VR experience. We refer the reader to the supplementary video for more details.

Tracking Method	Distance Range (m)				FPS
	1.0-1.5	1.5-2.0	2.0-2.5	2.5-3.0	
CheckerBoard	8.24	12.62	17.89	23.71	15.7
Lighthouse (1 Base)	5.62	7.98	9.03	13.31	60
Lighthouse (2 Base)	1.79	1.84	2.32	2.56	60
<b>MoiréBoard</b>	1.54	2.05	2.52	3.17	114.2

Table 24.3: **Real-scene tracking results.** We report the average tracking errors (in mm) of different methods as well as their tracking frame rate (in FPS). The checkerboard and MoiréBoard algorithms are implemented using iOS-OpenCV on an iPhone 11 Pro Max. Lighthouse algorithm is encapsulated in the hardware of Valve Index VR kit (Thus, its FPS may not reflect its true computational cost). Also note that our method can be further accelerated using the mobile device’s GPUs, since the performance bottleneck of our algorithm lays in the image processing tasks (such as applying the Gaussian filter), which are well suited for parallel processing.

## Section 25: Discussion and Future Work

We present a new camera tracking method that utilizes moiré effects. Our method has high tracking accuracy, comparable to the state-of-the-art commercial VR tracking device. But our method is fully passive, requiring only two layers of repetitive structures, which can be made at low cost (such as printing on paper and 3D-printed meshes). It is therefore particularly suitable for low-cost VR systems such as Google Cardboard.

As a marker-based tracking method, MoiréBoard also shares some limitations with other marker-based approaches. When the camera moves at a high speed, the captured images may suffer from motion blur, which causes the image-plane measurement to become less robust. One solution to this issue is to use high frame rate to capture images and thereby reduce motion blur. Currently, most mobile devices can record video with a high FPS (usually up to 120FPS, with high-end devices such as the iPhone 11 can record at 240FPS). In our experiments, recording at 120FPS suffices in typical VR gaming scenarios.

Another limitation of marker-based tracking stems from the camera's limited field-of-view (FoV). The optical landmark may not always be visible on the captured image. One possible solution is to use a wide angle camera (e.g., a fisheye camera and 360° camera). But this demands a more sophisticated image processing algorithm to rectify images due to wide-angle lens distortion.

Also related to the image rectification is the need of four visual markers at the corners of MoiréBoard. Currently, we rely on the markers to convert captured moiré fringes into a frontal view. However, we know *a priori* that all the moiré peak bands in the frontal view form square grids. Based on this prior, it is possible to rectify the images directly from the moiré fringes, without the need of visual markers. Thereby, we can reduce the form factor of the MoiréBoard. But this will require additional attention to identify the region where MoiréBoard exists.

Lastly, different MoiréBoard designs will have different tracking ranges (recall Table 23.1).

We can use multiple MoiréBoard designs at the same time to extend the tracking range. Better yet we could use an LED display (such as an iPad) to *dynamically* change the grating pattern (such as  $T_a$ ) on grid A, and thereby choose the most suitable MoiréBoard designs based on the camera's current position.

## **Chapter 5**

## **Epilogue**

## Epilogue

This dissertation focuses on bridging the gap between people, mobile devices, and the physical world. The purpose of this dissertation is to develop a better sensing experience for mobile devices and allow users to access their devices in a convenient and frictionless way. Specifically, this dissertation introduces four novel interaction techniques, *FontCode*, *Vidgets*, *BackTrack*, and *MoiréBoard*. Besides the particular problems they individually address, they share the same design principle that all functionalities can solely be achieved by the internal sensors of a phone, without introducing any external hardware or modifying the OS kernel. This is important because all the proposed methods can directly be deployed on stock mobile devices, indicating a lower barrier between research prototypes and real productions.

Although all the techniques target on mobile devices, the value of these techniques and their theories will beyond the proposed applications. First, *FontCode* proposed a new coding algorithm that has both high coding capacity and error-correction mechanisms. It can be used as a general coding scheme for coding blocks with different numbers of variants. Second, *Vidgets* described a way to design tangible widgets with unique force feedback. It also provided a simple algorithm for IMU signal identification. Third, *BackTrack* introduced the relationship between the signal strength of a capacitive sensor and the grounding conditions. It could be used as a generalized principle for extending the sensing region of capacitive sensors. Last, *MoiréBoard* proposed a novel theory of moiré pattern under camera projection. This technique could potentially be used for many other applications like robot navigation.

In summary, we hope this dissertation delivers and deepens the current understanding of how

users, mobile devices, and the physical world interact with each other and what interaction opportunities existed between them. We also believe the systems described in this dissertation could inspire a wide range of future research beyond mobile devices.

## References

- [1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015.
- [2] A Adobe, “Manager’s introduction to adobe extensible metadata platform,” *The Adobe XML Metadata Framework*, 2001.
- [3] M. Agarwal, “Text steganographic approaches: A comparison,” *International Journal of Network Security & Its Applications (IJNSA)*, vol. 5, no. 1, 2013.
- [4] K. Ahuja, S. Pareddy, R. Xiao, M. Goel, and C. Harrison, “Lightanchors: Appropriating point lights for spatially-anchored augmented reality interfaces,” in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, 2019, pp. 189–196.
- [5] A. M. Alattar and O. M. Alattar, “Watermarking electronic text documents containing justified paragraphs and irregular line spacing,” in *Electronic Imaging 2004*, International Society for Optics and Photonics, 2004, pp. 685–695.
- [6] A. M. Andrew, “Multiple view geometry in computer vision,” *Kybernetes*, 2001.
- [7] C Armbrüster, C Sutter, and M Ziefle, “Notebook input devices put to the age test: The usability of trackpoint and touchpad for middle-aged adults,” *Ergonomics*, vol. 50, no. 3, pp. 426–445, 2007.
- [8] B. Armstrong, T. Verron, L. Heppe, J. Reynolds, and K. Schmidt, “Rgr-3d: Simple, cheap detection of 6-dof pose for teleoperation, and robot programming and calibration,” in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, IEEE, vol. 3, 2002, pp. 2938–2943.
- [9] C. Avilés-Cruz, R. Rangel-Kuoppa, M. Reyes-Ayala, A Andrade-Gonzalez, and R. Escarela-Perez, “High-order statistical texture analysis—font recognition applied,” *Pattern Recognition Letters*, vol. 26, no. 2, pp. 135–145, 2005.
- [10] A. Bangor, P. T. Kortum, and J. T. Miller, “An empirical evaluation of the system usability scale,” *Intl. Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

- [11] P. Baudisch and G. Chu, "Back-of-device interaction allows creating very small touch devices," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2009, pp. 1923–1932.
- [12] J. Bergstrom-Lehtovirta and A. Oulasvirta, "Modeling the functional area of the thumb on mobile touchscreen surfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 1991–2000.
- [13] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series.," in *KDD workshop*, Seattle, WA, vol. 10, 1994, pp. 359–370.
- [14] W. Bhaya, A. M. Rahma, and A.-N. Dhamyaa, "Text steganography based on font type in ms-word documents," *Journal of Computer Science*, vol. 9, no. 7, p. 898, 2013.
- [15] A. Bianchi and I. Oakley, "Designing tangible magnetic appcessories," in *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, ACM, 2013, pp. 255–258.
- [16] J. A. Bloom, I. J. Cox, T. Kalker, J.-P. Linnartz, M. L. Miller, and C. B. S. Traw, "Copy protection for dvd video," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1267–1276, 1999.
- [17] D. Boneh, "Finding smooth integers in short intervals using crt decoding," in *Proceedings of the 32nd ACM symposium on Theory of computing*, 2000, pp. 265–272.
- [18] G. Borgefors, H.-O. Peitgen, J. K. Tsotsos, A. Leonardis, R. Klette, K. Ikeuchi, I. Amidror, T. S. Huang, R. Deriche, and T. Jiang, *The Theory of the Moiré Phenomenon*. Springer London, 2009.
- [19] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952.
- [20] J. T. Brassil, S. Low, N. F. Maxemchuk, and L. O’Gorman, "Electronic marking and identification techniques to discourage document copying," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1495–1504, 1995.
- [21] S. Brewster, S. Brewster, F. Chohan, and L. Brown, "Tactile feedback for mobile interactions," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2007, pp. 159–162.
- [22] J. Brooke, "Sus: A "quick and dirty" usability," *Usability evaluation in industry*, p. 189, 1996.
- [23] A. Butler, S. Izadi, and S. Hodges, "Sidesight: Multi-"touch" interaction around small devices," in *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '08)*, Association for Computing Machinery, Inc., 2008.

- [24] N. D. F. Campbell and J. Kautz, “Learning a manifold of fonts,” *ACM Trans. Graph.*, vol. 33, no. 4, 91:1–91:11, Jul. 2014.
- [25] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, “Orb-slam3: An accurate open-source library for visual, visual-inertial and multi-map slam,” *arXiv preprint arXiv:2007.11898*, 2020.
- [26] S. K. Card, G. G. Robertson, and J. D. Mackinlay, “The information visualizer, an information workspace,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’91, New Orleans, Louisiana, USA: ACM, 1991, pp. 181–186, ISBN: 0-89791-383-3.
- [27] S. Chaudhary, M. Dave, and A. Sanghi, “Text steganography based on feature coding method,” in *Proceedings of the International Conference on Advances in Information Communication Technology & Computing*, ACM, 2016, p. 7.
- [28] A. Cheddad, J. Condell, K. Curran, and P. Mc Kevitt, “Digital image steganography: Survey and analysis of current methods,” *Signal processing*, vol. 90, no. 3, pp. 727–752, 2010.
- [29] G. Chen, J. Yang, H. Jin, J. Brandt, E. Shechtman, A. Agarwala, and T. X. Han, “Large-scale visual font recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3598–3605.
- [30] Cherry GmbH, *Cherry mx switches*, <https://www.cherrymx.de/en>, 2018.
- [31] F. Chollet *et al.*, *Keras*, <https://github.com/fchollet/keras>, 2015.
- [32] K. Chu and C. Y. Wong, “Mobile input devices for gaming experience,” in *User Science and Engineering (i-USER), 2011 International Conference on*, IEEE, 2011, pp. 83–88.
- [33] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette, “Real-time markerless tracking for augmented reality: The virtual visual servoing framework,” *IEEE Transactions on visualization and computer graphics*, vol. 12, no. 4, pp. 615–628, 2006.
- [34] H. Corporation, *Htc vive vr system*, <https://www.vive.com/us/>, 2017.
- [35] V. Corporation, *Valve index lighthouse tracking system*, <https://www.valvesoftware.com/en/index/base-stations>, 2019.
- [36] C. Corsten, B. Daehlmann, S. Voelker, and J. Borchers, “Backxpress: Using back-of-device finger pressure to augment touchscreen input on smartphones,” in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 4654–4666.
- [37] A. De Luca, M. Harbach, E. von Zezschwitz, M.-E. Maurer, B. E. Slawik, H. Hussmann, and M. Smith, “Now you see me, now you don’t: Protecting smartphone authentication

- from shoulder surfers,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 2937–2946.
- [38] A. De Luca, E. Von Zezschwitz, N. D. H. Nguyen, M.-E. Maurer, E. Rubegni, M. P. Scipioni, and M. Langheinrich, “Back-of-device authentication on smartphones,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2013, pp. 2389–2398.
  - [39] A. De Luca, E. von Zezschwitz, N. D. H. Nguyen, M.-E. Maurer, E. Rubegni, M. P. Scipioni, and M. Langheinrich, “Back-of-device authentication on smartphones,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’13, Paris, France: Association for Computing Machinery, 2013, 2389–2398, ISBN: 9781450318990.
  - [40] J. DeGol, T. Bretl, and D. Hoiem, “Chromatag: A colored marker and fast detection algorithm,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1472–1481.
  - [41] P. Dempsey, “The teardown: Htc vive vr headset,” *Engineering & Technology*, vol. 11, no. 7-8, pp. 80–81, 2016.
  - [42] A. Denso, “Qr code essentials,” *Denso Wave*, vol. 900, 2011.
  - [43] G. Dogan, J. Bernal, and C. R. Hagwood, “Fft-based alignment of 2d closed curves with application to elastic shape analysis,” in *Proceedings of the 1st International Workshop on Differential Geometry in Computer Vision for Analysis of Shape, Images and Trajectories*, 2015, pp. 4222–4230.
  - [44] D. Eastlake 3rd and P. Jones, *Us secure hash algorithm 1 (sha1)*, United States, 2001.
  - [45] L. Facebook Technologies, *Oculus quest 2*, <https://www.oculus.com/quest-2/>, 2020.
  - [46] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, “Object detection with discriminatively trained part-based models,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.
  - [47] N. Ferguson and B. Schneier, *Practical cryptography*. Wiley New York, 2003, vol. 23.
  - [48] M. Fiala, “Artag, a fiducial marker system using digital techniques,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, IEEE, vol. 2, 2005, pp. 590–596.
  - [49] B. Foundation, *Blender software*, <https://www.blender.org/>, 2021.

- [50] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, “Automatic generation and detection of highly reliable fiducial markers under occlusion,” *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014.
- [51] O. Goldreich, D. Ron, and M. Sudan, “Chinese remaindering with errors,” in *Proceedings of the 31st ACM symposium on Theory of computing*, 1999, pp. 225–234.
- [52] Google, *Google cardboard portable vr system*, <https://arvr.google.com/cardboard/>, 2016.
- [53] Google Inc., *Sensor overview of android system*, [https://developer.android.com/guide/topics/sensors/sensors\\_overview](https://developer.android.com/guide/topics/sensors/sensors_overview), 2018.
- [54] E. Granell and L. A. Leiva, “Less is more: Efficient back-of-device tap input detection using built-in smartphone sensors,” in *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*, 2016, pp. 5–11.
- [55] J. Greenberg, “Understanding metadata and metadata schemes,” *Cataloging & classification quarterly*, vol. 40, no. 3-4, pp. 17–36, 2005.
- [56] T. Grosse-Puppenthal, C. Holz, G. Cohn, R. Wimmer, O. Bechtold, S. Hodges, M. S. Reynolds, and J. R. Smith, “Finding common ground: A survey of capacitive sensing in human-computer interaction,” in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2017, pp. 3293–3315.
- [57] A. Gutub and M. Fattani, “A novel arabic text steganography method using letter points and extensions,” *World Academy of Science, Engineering and Technology*, vol. 27, pp. 28–31, 2007.
- [58] H. Hakoda, Y. Fukatsu, B. Shizuki, and J. Tanaka, “Back-of-device interaction based on the range of motion of the index finger,” in *Proceedings of the Annual Meeting of the Australian Special Interest Group for Computer Human Interaction*, 2015, pp. 202–206.
- [59] R. H. Harris, *Buckling spring torsional snap actuator*, US Patent 4,118,611, 1978.
- [60] C. Harrison, R. Xiao, and S. Hudson, “Acoustic barcodes: Passive, durable and inexpensive notched identification tags,” in *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST ’12, Cambridge, Massachusetts, USA: ACM, 2012, pp. 563–568, ISBN: 978-1-4503-1580-7.
- [61] E. E. Hemayed, “A survey of camera self-calibration,” in *Proceedings of the IEEE Conference on Advanced Video and Signal Based Surveillance, 2003.*, IEEE, 2003, pp. 351–357.

- [62] S. Heo, J. Han, S. Choi, S. Lee, G. Lee, H.-E. Lee, S. Kim, W.-C. Bang, D. Kim, and C. Kim, “Ircube tracker: An optical 6-dof tracker based on led directivity,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 577–586.
- [63] R. D. Hersch and S. Chosson, “Band moiré images,” *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 239–247, 2004.
- [64] S. Hiraoka, I. Miyamoto, and K. Tomimatsu, “Behind touch, a text input method for mobile phones by the back and tactile sense interface,” *Information Processing Society of Japan, Interaction 2003*, pp. 131–138, 2003.
- [65] E. Hoggan, S. A. Brewster, and J. Johnston, “Investigating the effectiveness of tactile feedback for mobile touchscreens,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM, 2008, pp. 1573–1582.
- [66] C. Hu and R. D. Hersch, “Parameterizable fonts based on shape components,” *IEEE Computer Graphics and Applications*, vol. 21, no. 3, pp. 70–85, 2001.
- [67] S. Hwang, M. Ahn, and K.-y. Wohn, “Maggetz: Customizable passive tangible controllers on and around conventional mobile devices,” in *Proceedings of the 26th annual ACM symposium on User interface software and technology*, ACM, 2013, pp. 411–416.
- [68] K. Ikematsu, M. Fukumoto, and I. Siio, “Ohmic-sticker: Force-to-motion type input device that extends capacitive touch surface,” in *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, 2019, pp. 1021–1030.
- [69] K. Ikematsu and I. Siio, “Ohmic-touch: Extending touch interaction by indirect touch through resistive objects,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 2018, pp. 1–8.
- [70] K. Jo, M. Gupta, and S. K. Nayar, “Disco: Display-camera communication using rolling shutter sensors,” *ACM Trans. Graph.*, vol. 35, no. 5, Jul. 2016.
- [71] M.-C. Jung, Y.-C. Shin, and S. N. Srihari, “Multifont classification using typographical attributes,” in *Document Analysis and Recognition, 1999. ICDAR’99. Proceedings of the Fifth International Conference on*, IEEE, 1999, pp. 353–356.
- [72] M. Kalaitzakis, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, “Experimental comparison of fiducial markers for pose estimation,” in *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2020, pp. 781–789.
- [73] R. M. Karp, “Reducibility among combinatorial problems,” in *Complexity of computer computations*, Springer, 1972, pp. 85–103.

- [74] H. Kato and M. Billinghurst, “Marker tracking and hmd calibration for a video-based augmented reality conferencing system,” in *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR’99)*, IEEE, 1999, pp. 85–94.
- [75] K. Kato and H. Miyashita, “Extensionsticker: A proposal for a striped pattern sticker to extend touch interfaces and its assessment,” in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2015, pp. 1851–1854.
- [76] V. J. Katz and A. Imhausen, *The Mathematics of Egypt, Mesopotamia, China, India, and Islam: A Sourcebook*. Princeton University Press, 2007.
- [77] K. S. Killourhy and R. A. Maxion, “Comparing anomaly-detection algorithms for keystroke dynamics,” in *Dependable Systems & Networks, 2009. DSN’09. IEEE/IFIP international conference on*, IEEE, 2009, pp. 125–134.
- [78] S. Kim, B. Lee, and A. Oulasvirta, “Impact activation improves rapid button pressing,” in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI ’18, Montreal QC, Canada: ACM, 2018, 571:1–571:8, ISBN: 978-1-4503-5620-6.
- [79] Y.-W. Kim, K.-A. Moon, and I.-S. Oh, “A text watermarking algorithm based on word classification and inter-word space statistics,” in *ICDAR*, 2003, pp. 775–779.
- [80] D. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, 2015.
- [81] D. Knuth, “The metafont book. addison-welsey,” *Reading Mass*, 1986.
- [82] M. Kok, J. D. Hol, and T. B. Schön, “Using inertial sensors for position and orientation estimation,” *arXiv preprint arXiv:1704.06053*, 2017.
- [83] J. Konc and D. Janezic, “An improved branch and bound algorithm for the maximum clique problem,” *proteins*, vol. 4, no. 5, 2007.
- [84] S. Kratz and M. Rohs, “Hoverflow: Expanding the design space of around-device interaction,” in *Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 2009, pp. 1–8.
- [85] G. Laput, E. Brockmeyer, S. E. Hudson, and C. Harrison, “Acoustruments: Passive, acoustically-driven, interactive controls for handheld devices,” in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM, 2015, pp. 2161–2170.
- [86] G. Laput, R. Xiao, and C. Harrison, “Viband: High-fidelity bio-acoustic sensing using commodity smartwatch accelerometers,” in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 2016, pp. 321–333.

- [87] V. M. K. Lau, “Learning by example for parametric font design,” in *ACM SIGGRAPH ASIA 2009 Posters*, ser. SIGGRAPH ASIA '09, Yokohama, Japan, 2009, 5:1–5:1.
- [88] S. M. LaValle, A. Yershova, M. Katsev, and M. Antonov, “Head tracking for the oculus rift,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2014, pp. 187–194.
- [89] H. V. Le, S. Mayer, and N. Henze, “Infinitouch: Finger-aware interaction on fully touch sensitive smartphones,” in *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 2018, pp. 779–792.
- [90] B. Lee and A. Oulasvirta, “Modelling error rates in temporal pointing,” in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, ACM, 2016, pp. 1857–1868.
- [91] S. Lee, W. Buxton, and K. Smith, “A multi-touch three dimensional touch-sensitive tablet,” *Acm Sigchi Bulletin*, vol. 16, no. 4, pp. 21–25, 1985.
- [92] L. A. Leiva and A. Català, “Bod taps: An improved back-of-device authentication technique on smartphones,” in *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*, 2014, pp. 63–66.
- [93] V. Lepetit, F. Moreno-Noguer, and P. Fua, “Epnnp: An accurate o (n) solution to the pnp problem,” *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.
- [94] D. Li, D. I. Levin, W. Matusik, and C. Zheng, “Acoustic voxels: Computational optimization of modular acoustic filters,” *ACM Trans. Graph. (SIGGRAPH 2016)*, vol. 35, no. 4, 2016.
- [95] K. A. Li, P. Baudisch, and K. Hinckley, “Blindsight: Eyes-free access to mobile phones,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2008, pp. 1389–1398.
- [96] R.-H. Liang, H.-C. Kuo, L. Chan, D.-N. Yang, and B.-Y. Chen, “Gaussstones: Shielded magnetic tangibles for multi-token interactions on portable displays,” in *Proceedings of the 27th annual ACM symposium on User interface software and technology*, ACM, 2014, pp. 365–372.
- [97] S. Lin and D. J. Costello, *Error control coding*. Pearson Education India, 2004.
- [98] M. Liu, L. Cheng, K. Qian, J. Wang, J. Wang, and Y. Liu, “Indoor acoustic localization: A survey,” *Human-centric Computing and Information Sciences*, vol. 10, no. 1, pp. 1–24, 2020.

- [99] S. I. E. LLC., *Sony playstation vr*, <https://www.playstation.com/en-us/ps-vr/>, 2016.
- [100] M. Löchtefeld, C. Hirtz, and S. Gehring, “Evaluation of hybrid front-and back-of-device interaction on mobile devices,” in *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia*, 2013, pp. 1–4.
- [101] J. Loviscach, “The universe of fonts, charted by machine,” in *ACM SIGGRAPH 2010 Talks*, ACM, 2010, p. 27.
- [102] I. Magic Leap, *Magic leap 1*, <https://www.magicleap.com/en-us/magic-leap-1>, 2018.
- [103] W. Mao, J. He, and L. Qiu, “Cat: High-precision acoustic motion tracking,” in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, 2016, pp. 69–81.
- [104] N. Matsushima, W. Yamada, and H. Manabe, “Attaching objects to smartphones back side for a modular interface,” in *Adjunct Publication of the 30th Annual ACM Symposium on User Interface Software and Technology*, ACM, 2017, pp. 51–52.
- [105] Microsoft, *Hololens*, <https://www.microsoft.com/en-us/hololens/hardware>, 2018.
- [106] R. B. Miller, “Response time in man-computer conversational transactions,” in *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, ACM, 1968, pp. 267–277.
- [107] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.
- [108] R. Murray-Smith, J. Williamson, S. Hughes, and T. Quaade, “Stane: Synthesized surfaces for tactile input,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’08, Florence, Italy: ACM, 2008, pp. 1299–1302, ISBN: 978-1-60558-011-1.
- [109] N. M. Nasrabadi, “Pattern recognition and machine learning,” *Journal of electronic imaging*, vol. 16, no. 4, p. 049 901, 2007.
- [110] A. W. Ng and A. H. Chan, “Finger response times to visual, auditory and tactile modality stimuli,” in *Proceedings of the international multiconference of engineers and computer scientists*, vol. 2, 2012, pp. 1449–1454.

- [111] D. C. Niehorster, L. Li, and M. Lappe, “The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research,” *i-Perception*, vol. 8, no. 3, p. 2041669517708205, 2017.
- [112] D. Norman, *The design of everyday things: Revised and expanded edition*. Constellation, 2013.
- [113] P. O’Donovan, J. Lībeks, A. Agarwala, and A. Hertzmann, “Exploratory font selection using crowdsourced attributes,” *ACM Trans. Graph.*, vol. 33, no. 4, 92:1–92:9, Jul. 2014.
- [114] E. Olson, “Apriltag: A robust and flexible visual fiducial system,” in *2011 IEEE International Conference on Robotics and Automation*, IEEE, 2011, pp. 3400–3407.
- [115] N. Otsu, “A threshold selection method from gray-level histograms,” *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.
- [116] J. Panda, N. Gupta, P. Saxena, S. Agrawal, S. Jain, and A. Bhattacharyya, “Text watermarking using sinusoidal greyscale variations of font based on alphabet count,” 2015.
- [117] H. Q. Phan, H. Fu, and A. B. Chan, “Flexyfont: Learning transferring rules for flexible typeface synthesis,” in *Computer Graphics Forum*, Wiley Online Library, vol. 34, 2015, pp. 245–256.
- [118] M. Piovarči, D. I. Levin, D. M. Kaufman, and P. Didyk, “Perception-aware modeling and fabrication of digital drawing tools,” *ACM Transactions on Graphics (SIGGRAPH 2018)*, vol. 37, no. 4, p. 123, 2018.
- [119] D. Pustka, J.-P. Hülß, J. Willneff, F. Pankratz, M. Huber, and G. Klinker, “Optical outside-in tracking using unmodified mobile phones,” in *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, IEEE, 2012, pp. 81–89.
- [120] R Ramanathan, K. Soman, L Thaneshwaran, V Viknesh, T Arunkumar, and P Yuvaraj, “A novel technique for english font recognition using support vector machines,” in *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom’09. International Conference on*, IEEE, 2009, pp. 766–769.
- [121] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, pp. 300–304, 1960.
- [122] D. Rempel, J. Dennerlein, C. Mote Jr, and T. Armstrong, “A method of measuring fingertip loading during keyboard use,” *Journal of Biomechanics*, vol. 27, no. 8, pp. 1101–1104, 1994.
- [123] R. Rivest, *The md5 message-digest algorithm*, United States, 1992.

- [124] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
- [125] S. G. Rizzo, F. Bertini, and D. Montesi, “Content-preserving text watermarking through unicode homoglyph substitution,” in *Proceedings of the 20th International Database Engineering & Applications Symposium*, ACM, 2016, pp. 97–104.
- [126] K. H. Rosen, *Elementary number theory*. Pearson Education, 2011.
- [127] L. Ruggles, “Letterform design systems,” Stanford University, Tech. Rep., 1983.
- [128] Samsung, *Samsung gear vr*, <https://www.samsung.com/global/galaxy/gear-vr/>, 2017.
- [129] M. R. U. Saputra, A. Markham, and N. Trigoni, “Visual slam and structure from motion in dynamic environments: A survey,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–36, 2018.
- [130] V. Savage, A. Head, B. Hartmann, D. B. Goldman, G. Mysore, and W. Li, “Lamello: Passive acoustic sensing for tangible input components,” in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ser. CHI ’15, Seoul, Republic of Korea: ACM, 2015, pp. 1277–1280, ISBN: 978-1-4503-3145-6.
- [131] Á. Seress, *Permutation group algorithms*. Cambridge University Press, 2003, vol. 152.
- [132] A. Shamir and A. Rappoport, “Feature-based design of fonts using constraints,” in *Electronic Publishing, Artistic Imaging, and Digital Typography*, Springer, 1998, pp. 93–108.
- [133] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, 1948.
- [134] K. A. Siek, Y. Rogers, and K. H. Connelly, “Fat finger worries: How older and younger users physically interact with pdas,” in *IFIP Conference on Human-Computer Interaction*, Springer, 2005, pp. 267–280.
- [135] R. Smith, “An overview of the tesseract ocr engine,” in *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, ser. ICDAR ’07, Washington, DC, USA: IEEE Computer Society, 2007, pp. 629–633, ISBN: 0-7695-2822-8.
- [136] I. Stojanov, A. Mileva, and I. Stojanovic, “A new property coding in text steganography of microsoft word documents,” 2014.
- [137] E. Strasnick, J. Yang, K. Tanner, A. Olwal, and S. Follmer, “Shiftio: Reconfigurable tactile elements for dynamic affordances and mobile interaction,” in *Proceedings of the 2017 CHI*

*Conference on Human Factors in Computing Systems*, ser. CHI '17, Denver, Colorado, USA: ACM, 2017, pp. 5075–5086, ISBN: 978-1-4503-4655-9.

- [138] M. Sugimoto and K. Hiroki, “Hybridtouch: An intuitive manipulation technique for pdas using their front and rear surfaces,” in *Proceedings of the 8th conference on Human-computer interaction with mobile devices and services*, ACM, 2006, pp. 137–140.
- [139] K. Sun, T. Zhao, W. Wang, and L. Xie, “Vskin: Sensing touch gestures on surfaces of mobile devices using acoustic signals,” in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 591–605.
- [140] R. Suveeranont and T. Igarashi, “Example-based automatic font generation,” in *International Symposium on Smart Graphics*, Springer, 2010, pp. 127–138.
- [141] T. Taketomi, H. Uchiyama, and S. Ikeda, “Visual slam algorithms: A survey from 2010 to 2016,” *IPSN Transactions on Computer Vision and Applications*, vol. 9, no. 1, pp. 1–11, 2017.
- [142] H. Tanaka, Y. Sumi, and Y. Matsumoto, “A high-accuracy visual marker based on a microlens array,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2012, pp. 4192–4197.
- [143] ———, “A visual marker for precise pose estimation based on lenticular lenses,” in *2012 IEEE International Conference on Robotics and Automation*, IEEE, 2012, pp. 5222–5227.
- [144] F. Tari, A. A. Ozok, and S. H. Holden, “A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords,” in *Proceedings of the second symposium on Usable privacy and security*, 2006, pp. 56–66.
- [145] K. Um, X. Hu, and N. Thuerey, “Perceptual evaluation of liquid simulation methods,” *ACM Trans. Graph.*, vol. 36, no. 4, 2017.
- [146] A. Wang and S. Gollakota, “Millisonic: Pushing the limits of acoustic motion tracking,” in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 2019, pp. 1–11.
- [147] G. Wang, “Designing smule’s ocarina: The iphone’s magic flute,” in *NIME*, 2009, pp. 303–307.
- [148] Y. Wang, J. Zhou, H. Li, T. Zhang, M. Gao, Z. Cheng, C. Yu, S. Patel, and Y. Shi, “Flex-touch: Enabling large-scale interaction sensing beyond touchscreens using flexible and conductive materials,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 3, no. 3, pp. 1–20, 2019.

- [149] Z. Wang, J. Yang, H. Jin, E. Shechtman, A. Agarwala, J. Brandt, and T. S. Huang, “Deepfont: Identify your font from an image,” in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM ’15, Brisbane, Australia: ACM, 2015, pp. 451–459, ISBN: 978-1-4503-3459-4.
- [150] P. Wayner, *Disappearing cryptography: information hiding: steganography & watermarking*. Morgan Kaufmann, 2009.
- [151] ———, “Mimic functions,” *Cryptologia*, vol. 16, no. 3, pp. 193–214, 1992.
- [152] J. T. Weinhandl, B. S. Armstrong, T. P. Kusik, R. T. Barrows, and K. M. O’Connor, “Validation of a single camera three-dimensional motion tracking system,” *Journal of biomechanics*, vol. 43, no. 7, pp. 1437–1440, 2010.
- [153] D. Wigdor, D. Leigh, C. Forlines, S. Shipman, J. Barnwell, R. Balakrishnan, and C. Shen, “Under the table interaction,” in *Proceedings of the 19th annual ACM symposium on User interface software and technology*, 2006, pp. 259–268.
- [154] P. C. Wong, H. Fu, and K. Zhu, “Back-mirror: Back-of-device one-handed interaction on smartphones,” in *SIGGRAPH ASIA 2016 Mobile Graphics and Interactive Applications*, 2016, pp. 1–5.
- [155] C.-F. Wu, C.-C. Lai, and Y.-K. Liu, “Investigation of the performance of trackpoint and touchpads with varied right and left buttons function locations,” *Applied ergonomics*, vol. 44, no. 2, pp. 312–320, 2013.
- [156] C. Xiao, K. Bayer, C. Zheng, and S. K. Nayar, “Vidgets: Modular mechanical widgets for mobile devices,” *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, pp. 1–12, 2019.
- [157] R. Xiao, G. Laput, and C. Harrison, “Expanding the input expressivity of smartwatches with mechanical pan, twist, tilt and click,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 193–196.
- [158] R. Xiao, G. Lew, J. Marsanico, D. Hariharan, S. Hudson, and C. Harrison, “Toffee: Enabling ad hoc, around-device interaction with acoustic time-of-arrival correlation,” in *Proceedings of the 16th international conference on Human-computer interaction with mobile devices & services*, ACM, 2014, pp. 67–76.
- [159] X. Xiao, T. Han, and J. Wang, “Lensgesture: Augmenting mobile interactions with back-of-device finger gestures,” in *Proceedings of the 15th ACM on International conference on multimodal interaction*, 2013, pp. 287–294.
- [160] G. Younes, D. Asmar, E. Shamma, and J. Zelek, “Keyframe-based monocular slam: Design, survey, and future directions,” *Robotics and Autonomous Systems*, vol. 98, pp. 67–88, 2017.

- [161] N.-H. Yu, S.-S. Tsai, I.-C. Hsiao, D.-J. Tsai, M.-H. Lee, M. Y. Chen, and Y.-P. Hung, “Clip-on gadgets: Expanding multi-touch interaction area with unpowered tactile controls,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 367–372.
- [162] L. Zaman, D. Natapov, and R. J. Teather, “Touchscreens vs. traditional controllers in handheld gaming,” in *Proceedings of the international academic conference on the future of game design and technology*, ACM, 2010, pp. 183–190.
- [163] C. Zhang, A. Guo, D. Zhang, Y. Li, C. Southern, R. I. Arriaga, and G. D. Abowd, “Beyond the touchscreen: An exploration of extending interactions on commodity smartphones,” *ACM Trans. Interact. Intell. Syst.*, vol. 6, no. 2, 16:1–16:23, Aug. 2016.
- [164] C. Zhang, A. Guo, D. Zhang, C. Southern, R. Arriaga, and G. Abowd, “Beyondtouch: Extending the input language with built-in sensors on commodity smartphones,” in *Proceedings of the 20th International Conference on Intelligent User Interfaces*, ser. IUI ’15, Atlanta, Georgia, USA: ACM, 2015, pp. 67–77, ISBN: 978-1-4503-3306-1.
- [165] C. Zhang, Q. Xue, A. Waghmare, S. Jain, Y. Pu, S. Hersek, K. Lyons, K. A. Cunefare, O. T. Inan, and G. D. Abowd, “Soundtrak: Continuous 3d tracking of a finger using active acoustics,” *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, pp. 1–25, 2017.
- [166] C. Zhang, J. Yang, C. Southern, T. E. Starner, and G. D. Abowd, “Watchout: Extending interactions on a smartwatch with inertial sensing,” in *Proceedings of the 2016 ACM International Symposium on Wearable Computers*, 2016, pp. 136–143.
- [167] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.