Conductive Inkjet Printed Passive 2D TrackPad for VR Interaction

Chuhan Gao
University of Wisconsin - Madison
chuhan@cs.wisc.edu

Xinyu Zhang
University of California San Diego
xyzhang@ucsd.edu

Suman Banerjee
University of Wisconsin - Madison
suman@cs.wisc.edu

ABSTRACT

Mobile virtual reality (VR) headsets, such as Google Cardboard and Samsung GearVR, can reuse a smartphone as near-eye display to create immersive experience. But such devices barely support any user interaction, even for simple tasks such as menu selection and single-character input. In this paper, we design *Inkput*, a simple passive interface attached to the unexploited backside of the headset to enable touch sensing. Inkput is a piece of paper substrate with carbon ink patterns printed atop. It leverages the column of electrodes near the edge of the smartphone touchscreen to sense multi-touch on the 2D space, and is even able to locate finger hovering. Our experiments demonstrate that Inkput can precisely detect touch positions with *mm*-level precision. Our case studies in actual VR applications also verify that Inkput can support common VR interactions and can even outperform high-end handheld controllers in terms of efficiency.

KEYWORDS

Mobile virtual reality; VR interaction; Inkjet printing; Touch sensing

ACM Reference Format:
Chuhan Gao, Xinyu Zhang, and Suman Banerjee. 2018. Conductive Inkjet Printed Passive 2D TrackPad for VR Interaction. In *The 24th Annual International Conference on Mobile Computing and Networking (MobiCom '18), October 29-November 2, 2018, New Delhi, India.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3241539.3241546

MobiCom '18, October 29-November 2, 2018, New Delhi, India

ACM ISBN 978-1-4503-5903-0/18/10…$15.00
https://doi.org/10.1145/3241539.3241546

1 INTRODUCTION

The increasingly higher computation capability as well as advanced pose-tracking technologies [1, 2] have made smartphones a self-contained virtual reality (VR) device. The smartphones can be slotted into a variety of headsets, such as Google Cardboard [3], Daydream [4] and Samsung GearVR [5], which form a low-cost head-mounted-display (HMD), offering mobile VR experience to users anywhere, anytime.

One fundamental drawback of such mobile VR is the poor interaction capability. Since the smartphone screen is chambered in the headset and fully occupied by display, no intuitive touch interface is available, which severely limits how the user interacts with the virtual world. Many mainstream mobile VR systems attempt to alleviate this drawback using *head-rotation* and/or an *external handheld controller*. The *head-rotation* approach employs the smartphone's inertial measurement units (accelerometer, gyroscope and compass). The user can rotate her head to navigate a crosshair cursor and select UI elements through a countdown timer. But head-rotation can be cumbersome even for simple utilitarian interactions such as navigating a menu and selecting items. On the other hand, the *external controller* can have physical click-buttons, built-in motion sensors, or a mini-touchpad with swiping sensing capabilities [4, 5], to ease the selection. However, navigating a cursor with such an external controller is far from convenient or expressive for certain operations, such as text entry, where one or more fingers are necessary to interact naturally with the application, similar to the way one would interact with ordinary touchscreens.

Alternative hand tracking technologies may approach the ideal and enable natural interaction within the virtual environment through hand/finger gestures. Examples include the gesture gloves which instrument the hand [6], Leap Motion [7] which instruments the HMD with infrared sensors/cameras, HTC VIVE's Lighthouse [8] and Oculus Rift [9] which instrument the environment with optical tracking basestations, *etc.* Despite the high tracking accuracy, these technologies are much more costly than the mobile VR HMD itself. They entail heavy instrumentation overhead, and are hardly practical for mobile VR systems such as Google Cardboard.

In this paper, we propose *Inkput*, a lightweight and passive paper-based interface to enable multi-finger touch sensing with millimeter-level precision. As illustrated in Fig. 1, Inkput

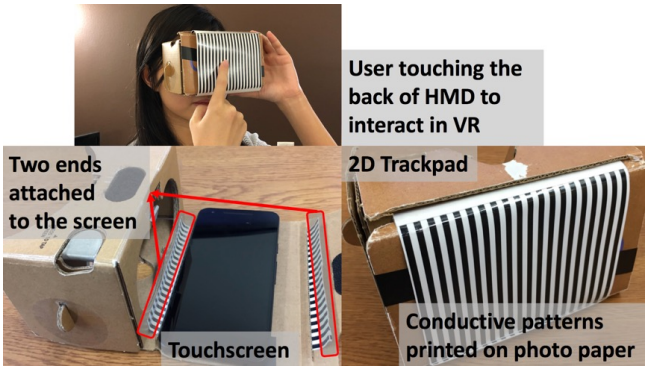


Figure 1: Google Cardboard integrated with Inkput.

is a piece of paper with conductive stripe patterns printed atop. A thin edge of the paper is attached to the edges of a smartphone touchscreen, which extends its interaction area to the backside of the HMD. Touches on the paper "propagates" to the touchscreen, and can be localized by Inkput's sensing algorithms running inside the smartphone. Inkput does not need any hardware/firmware modification to the mobile device, nor does it require any additional infrastructure. It can be produced by a home printer, ordinary photo paper and cheap *carbon ink*.

Despite its conceptual simplicity, Inkput faces three significant practical challenges. ***(a)*** *How to sense 2D touch position on a fully passive printed extension?* Modern capacitive touchscreens comprise a 2D grid of *touch sensing units*, *i.e.*, the electrodes. To avoid blocking the display, Inkput only attaches its conductive pattern with the column of electrodes on the long edge of the touchscreen. This single column of electrodes needs to sense touches on the 2D paper. To meet this challenge, we use conductive carbon ink to print extension stripes, each in contact with the smartphone edge. Different finger positions, even along the same stripe, can generate different resistance/capacitance which leads to different readings on the electrodes.

(b) *How to detect multi-finger touch?* The ability to detect multiple simultaneous touches is crucial for complex gestures such as zooming. With normal touchscreen, the positions of multiple touches can be detected by identifying local maximums in the 2D electrode grid output. In Inkput, the effects of *multiple touches* can mix together on the same stripe, and must be disentangled by a single electrode. In addition, since the finger is in proximity of multiple stripes, it may be *capacitively coupled* to multiple electrodes on the screen. Such interference can make the local maximums even harder to find. Inkput resolves these issues using a model-based optimization mechanism that takes advantage of the interference among stripes to estimate which stripes and which positions are touched.

(c) *How to detect hovering?* Since the VR user's eyes are blinded by the HMD, she sees the exact touch position indicated on the screen, only after touch occurs. Thus, a wrong point may be touched before she realizes. Inkput alleviates such a usability issue by detecting the *finger hovering* position. It first distinguishes touch and hovering based on their intrinsically different stripe interference models. Then, it fuses the interference model for hover into the optimization framework, to estimate hover positions.

We have prototyped Inkput using ordinary home printer and photo-papers, and implemented the above 2D touch/hover sensing mechanisms on Android phones. Our experiments demonstrate that Inkput can localize touches on the printed extension with millimeter (mm) level accuracy. Specifically, for single-finger touch, the average error is around 1.7 mm, and only increases to 5 mm for 4 simultaneous touches. Inkput also achieves mm-level accuracy for hover position detection. It incurs negligible sensing latency, and can output the results at 25 Hz even for multi-touch. We further conducted two case studies to verify the usability of Inkput in a practical setup. In a VR text-entry application, Inkput achieves comparable input efficiency as the heavily instrumented HTC VIVE, at even lower error rate. We also repurpose Inkput as a paper-keyboard to extend a smartphone touchscreen, and observe that it achieves close to zero error rate.

The idea of extending capacitive touch interface through conductive stripes has been explored by previous work [10–15], among which ExtensionSticker [10] is closest to Inkput. However, these systems adopt silver nanoparticle ink with near-zero resistance, and hence cannot locate touch points along a stripe. Consequently, they can only detect 1D gestures, *i.e.*, tapping on one stripe or swiping across multiple stripes. Inkput represents the first to realize 2D touch sensing and hovering detection, using resistive carbon ink augmented with sensor signal processing algorithms. To this end, Inkput makes the following technical contributions:

(a) Proposing a simple paper-printable interface that borrows tiny edges of the smartphone touchscreen, and realizes 2D touch sensing on the unexploited backside of a mobile VR headset.

(b) Designing effective mechanisms to realize highly accurate and reliable multi-touch sensing and hover sensing capabilities on the printed extension.

(c) Integrating Inkput with VR applications, and verifying its performance and usability through comprehensive micro-benchmarks and use case studies.

2 RELATED WORK

Many VR and augmented reality (AR) applications rely on vision/optical tracking as an input method. Advanced VR systems [8, 9] use external tracking stations to locate an HMD instrumented with optical transceivers. Alternative

self-contained tracking solutions may use infrared or cameras to track finger/hand gestures in the air [7, 16–18]. However, the high computation cost of such vision-based approaches limits the frame rate of tracking, which degrades the user experience in interaction-rich VR applications. In addition, cameras can quickly drain the smartphone battery along with the power-hungry display and computationally intensive VR applications.

Existing mobile VR systems usually resort to a bruteforce way of controlling the device at a general UI level (*e.g.*, menu configuration and text input). The user has to execute such operations on the touchscreen before wearing the HMD. To change the settings the user has to take off the HMD again. Besides touching/selecting utilitarian UI elements, some VR applications create a virtual representation of user's hands, allowing interaction with virtual objects with arms' reach (assuming hand position can be precisely tracked). However, unlike real objects, no haptic feedback is provided to the user. So these approaches do not offer much usability advantage compared with Inkput, which uses the HMD backside itself to provide passive haptic feedback. Eye gaze tracking [19] may help navigate a cursor in VR, but it incurs high computation overhead like other computer vision solutions, and suffers from the same inconvenience as head-rotation.

Inkput is inspired by the back-of-phone interaction techniques [20–23], which augment the devices' backside with physical buttons or an additional touch pad. Despite the cost and form-factor limitations, they have shown the feasibility of mobile interaction and input, purely based on users' sense of *proprioception*, *i.e.*, ability to touch or select body parts/accessories without seeing the fingers. Facetouch [24] extended the principle to mobile VR HMD. But again, it needs to add an additional touchscreen and wire it with the smartphone inside the HMD.

Many wireless sensing techniques have been proposed recently that use acoustic [25–29] or RF [30–34] signals to identify hand gestures or even track finger movement. Although the mean tracking accuracy can reach below 1 cm in controlled settings, these approaches are severely limited by their reliability in complicated environments. In particular, frequent body and device movement create sophisticated multipath reflections that compromise the signal strength or phase measurements. Consequently, in practical use cases, even with limited body movement, the 90-percentile accuracy of state-of-the-art wireless solutions can only reach a few centimeters at best [27, 31], far from enough for touch interactions with mobile VR. Besides, none of such systems is capable of multi-finger touch sensing. Motion sensors on the smartphone have also been used to sense touches on fixed locations [35, 36], but cannot afford 2D continuous tracking. RIO is another recently proposed system that detects touch on RFID tags [37]. Although the tags are extremely cheap, RIO requires a dedicated reader, which typically costs several

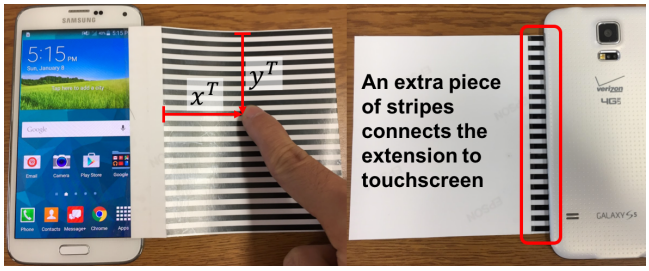


Figure 2: Inkput attaches a printed paper with conductive stripes to the edge of the touchscreen. An extra piece of printed stripes connects the paper and touchscreen so that printed pattern faces the same direction as the screen.

hundred dollars. It also does not provide the capability to detect touches continuously along the conductive material, thus unable to support 2D touch sensing.

Recent work exploited conductive ink as a lightweight medium to extend touch sensing interfaces. ExtensionSticker [10, 11], in particular, attaches a series of thin conductive ink stripes to extend a touchscreen, similar to Inkput. Yet it can only recognize "on/off" touch on a low-resistance stripe. In contrast, Inkput realizes 2D multi-touch and hover sensing, by locating the exact touch point along each carbon ink stripe and resolving coupling between stripes. Wiethoff *et al.* [15] draw conductive ink on physical objects and turn them into tangible interfaces. Holman *et al.* [14] wire resistive patterns to a microprocessor to create a keyboard-like touch device. Many other systems used different conductive medium together with dedicated capacitive sensing circuits to detect interactions [12, 13, 38, 39]. Operating at a larger scale, Wall++ [40] instruments capacitive sensors in a wall, turning the entire wall into a large 2D touch/proximity sensor.

3 OVERVIEW

Fig. 2 illustrates how Inkput acquires touch sensing capabilities by borrowing the edge of the smartphone touchscreen. The primary physical components of Inkput are a series of conductive stripes printed on normal photo paper. In Inkput's coordination system, the top left corner of the printed paper extension is the origin point. The objective is to precisely sense a touch position (x^T, y^T) within this 2D space. Inferring y^T is relatively easy, because touches on the stripe only affect the electrodes in its close contact. On the other hand, to determine x^T, we print carbon-ink stripes with relatively high resistance (over $100k\Omega$). The resistance between the finger and the touchscreen changes with the x^T, resulting in different output values on the touchscreen electrode, which can be utilized together with the signal processing components of Inkput to infer touch/hover positions.

Before putting Inkput into use, a simple one-time calibration is needed to parameterize its sensing algorithms.

Specifically, a user needs to swipe across the stripes, which allows Inkput to obtain two models: (a) *Stripe resistance curve*, which characterizes how touchscreen output changes with the resistance between finger and screen, *i.e.*, x^T (Sec. 5.2); (b) *Stripe interference pattern*, which characterizes how a nearby stripe is affected when one is touched. These two models are used together to perform touch and hover detection, as well as a model-based optimization to identify multiple touches (Sec. 6.2, 7). Note that the calibration does not need to be repeated unless the user switches to a new smartphone, or the stripes' geometry/material changes. In addition, we find through experiments that the two models do not vary across users, so the calibration does not need to be repeated for different users, either.

Due to its low-cost, easy-to-build features, Inkput can easily be produced at home with normal printer and cheap, consumer-grade conductive ink. Beyond the use case of back-of-HMD touch sensing for mobile VR, Inkput can serve as a generic solution to extend the interaction area of mobile devices (Sec. 12.3). The passive and lightweight printed sheet can be a separate smartphone accessory or integrated with flip case, and serve as a convenient touchscreen extension anytime, anywhere.

4 PROTOTYPING OF INKPUT

4.1 Touch Sensing Device

We prototype Inkput on two Android smartphones: Nexus 5 (N5) and Galaxy S5 (S5). Both have capacitive touchscreens that are commonly used by many other phone models. Underneath the screen are touch sensing electrodes organized in a 27×15 (28×16) grid for N5 (S5). Each electrode (sensing unit) measures the capacitance variation upon finger touch. Aside from touch, hover detection is naturally supported by touchscreen hardware. Finger hovering causes slight variations of the measured capacitance (explained in Sec. 9.4), and therefore a close hover can be detected by hardware, although the detectable range is usually short. Hover detection is usually disabled by the driver as reporting measured small capacitance variation might cause more false positives frequently in touch detection.

Both N5 and S5 implement the Synaptics touchscreen controller, which controls the touchscreen hardware and extracts the RAW touch sensing data. The RAW data is then processed by the touchscreen controller driver (de-noising, phantom removal, interpolation, *etc.*) and estimated touch positions are reported to the OS kernel. In order to acquire the RAW data as input to Inkput, we use a modified Synaptics touchscreen driver that supports RAW data reporting [41], which we integrated into N5 and S5's kernel. The driver was initially found in the Moto X 2013's kernel source, named Synaptics DSX, and by enabling the test-reporting function, the RAW data become accessible.

4.2 Conductive Inkjet Printing

Conductive inkjet printing has been widely used in fast prototyping electronics due to its low-cost and reconfigurability. The conductive ink (typically made of silver or carbon) is inkjet-printed on either photo paper or specialized substrates to form conductive wires. Different types of ink differ in terms of conductivity, dielectric properties, surface tension, *etc.*, which affect the printed circuit's performance, especially under high frequency.

Inkput only uses conductive patterns to create resistors operating at very low frequency, as the scanning frequency of touchscreen is only 10-50 Hz. The only property that matters is the conductivity of the ink. We use a type of *conductive carbon ink* which has a sheet resistance of around $4.5k\Omega/sq$ and can be printed on normal photo paper [42]. We will explain the choice of conductivity level in Sec. 5.

To produce conductive patterns, we simply fill the carbon ink into the cartridge of a home inkjet printer, Epson Stylus C88+ [43]. Today's home printers fall in one of two categories: *(i)* Inkjet printers produce images by dropping spots of ink onto the paper. *(ii)* Laser printers do this by scanning laser beams across photoreceptors. The latter are more popular in office environments due to faster printing speed, while inkjet printers are more often used to print photos or image-heavy documents due to higher image quality. Since the carbon ink is in liquid form under room temperature, it needs to be used on inkjet printers.

The Epson Stylus C88+ costs around $120. The carbon ink from NovaCentrix [44] costs around $100 for 50 ml, and the Epson photo paper we use costs $10 per 20 8×10 sheets. We find that 50 ml of ink could produce more than 500 sheets of extensions used by Inkput (Fig. 2). Therefore, each extension only costs around $0.4 worth of conductive ink and paper. The cost can be further reduced under massive production.

Finally, to integrate the printed extension with touchscreen, we fix the extra piece of printed stripes to the touchscreen edge using tapes. This only occupies a tiny width (3 *mm*) of the screen, but suffices to trigger the edge column of electrodes. Although we use tape for the connection, a more elegant solution can be an additional folding structure in the VR HMD, which physically touches the smartphone when it is mounted. In addition, note that the touchscreen electrodes are not visible to users, and *Inkput does not require each stripe to be aligned with each individual electrode* because the y^T it resolves is a continuous variable (Sec. 6.2).

5 DETECTING TOUCH POSITION ALONG A SINGLE STRIPE

5.1 A Primer on Capacitive Touchscreen

A touchscreen generally contains a grid of sensors underneath an insulating layer, such as glass or polymer [45]. Among a variety of sensing approaches, including resistive

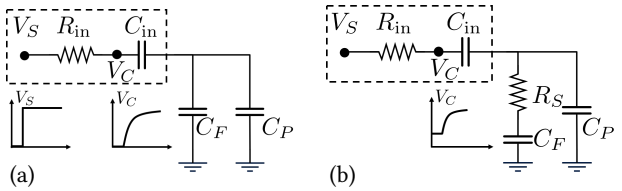


Figure 3: Equivalent circuit model of touch sensing hardware and charging voltage before/when touched (a) original touchscreen electrode; (b) with Inkput stripe.

[46], surface capacitive [45], and optical [47] technologies, capacitive sensing is most widely today due to its high efficiency, robustness, sensitivity and low cost [48]. Capacitive touchscreen is used by more than 92% touch sensing devices shipped in 2014, and predicted to be 98% in 2018 [49].

When a conductive object is in close proximity or direct contact, the touchscreen's capacitive sensing electrodes are coupled with it [50]. The variation of capacitance is detected by the electrodes by measuring the change of charging time of an inner capacitor. Fig. 3 (a) illustrates an equivalent cascade *RC* circuit model for one electrode, where the voltage of inner capacitor, denoted by V_C, can be written as,

$$V_C = V_S(1 - e^{-\frac{C_{\text{in}}+C_F+C_P}{C_{\text{in}}(C_F+C_P)R_{\text{in}}}t}) \quad (1)$$

where V_S is a fixed supply voltage; R_{in}, C_{in} are the inner resistor and capacitor; C_F and C_P are finger capacitance and parasitic capacitance, respectively; and t represents the time since V_S is supplied [51]. Parasitic capacitance exists in every AC circuit due to proximity between conductive components. To detect touch, a touchscreen controller periodically transmits a probing signal to scan through all the electrodes, and measures the capacitance through the charging time τ, *i.e.*, the time it takes for V_C to increase to $\delta_C V_S$, where δ_C is a threshold ($0 < \delta_C < 1$). τ can be represented as [51],

$$\tau = \frac{C_{\text{in}}(C_F + C_P)R_{\text{in}}}{C_{\text{in}} + C_F + C_P} \ln(\frac{1}{1-\delta_C}) \quad (2)$$

5.2 Detecting Touch on a Conductive Stripe

For simplicity, we start with the basic setup where one end of a single stripe physically contacts one electrode on the touchscreen. When touching the stripe, it can be regarded that a resistor R_S is connected between finger capacitor C_F and the sensor, as illustrated in Fig.3 (b). The R_S value depends on the length of the stripe in between the finger tip and the screen, as well as the resistivity of the printed stripe. For a printed thin film conductor, the resistance can be calculated as $\rho_s \frac{L}{W}$, where L, W are its length and width, respectively; and ρ_s represents sheet resistance, which characterizes the resistivity of the film [51]. More specifically, ρ_s describes the relation between resistance and the length/width of the pattern, given a certain thickness. ρ_s is usually fixed for a certain material and printing technique. For example, a sheet

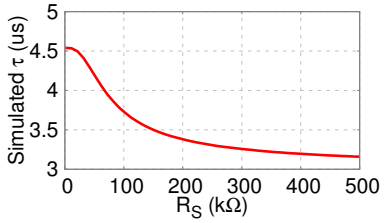


Figure 4: Simulation on impact of R_S on charging time.

resistance of $4.5k\Omega/sq$ means the film has $4.5k\Omega$ resistance for a square shape pattern. Based on the resistance model, we know that the overall *resistance is only affected by the ratio between L and W*, thus any similar square patterns would have the same resistance.

Impact of R_S on charging time. Adding an additional resistor in series with the finger capacitor decreases the charging time. Without R_S, the initial value of V_C is zero during the charging, while adding R_S lets V_C have a non-zero initial state, which reduces the time for V_C to reach the threshold. Increasing R_S further increases the initial voltage of V_C, thus decreasing the charging time. However, as R_S becomes too large, the C_P connected in parallel becomes dominant, so the charging time will decrease much more slowly.

To validate the analysis, we use numerical simulation to quantify the time it takes for V_C in Fig. 3 (b) to reach $0.7V_S$. As the exact implementation of commodity touchscreens is unknown, we set the parameters of the simulated circuit such that the output (charging time) is sensitive to capacitance change caused by finger touch (10 pF to 100 pF). Specifically, R in is set to $2M\Omega$. C_{in}, C_F and C_P are set to 100 pF, 50 pF and 10 pF, respectively. The results in Fig. 4 show that indeed *a one-to-one mapping exists between τ and R_S, and the relation is monotonic.*

Estimating x^T from the output of touch sensing unit. To model the relation between the touchscreen output and touch position x^T along the stripe, Inkput first performs a one-time calibration to get a ***stripe resistance curve***. Since a monotonic relation exists between the touch sensing unit's output and R_S/x^T, Inkput can infer x^T from the stripe resistance curve and the touch sensor readings.

In order to validate the rationale, we print a single stripe and connect it to one touch sensing unit on the S5 phone. We touch different positions along the stripe and measure the actual resistance R_S using a LCR meter. The touch sensing output is normalized so that it equals zero when not touched. Moreover, since the stripe has uniform width and resistivity, the position x^T should be equal to R_S divided by a constant. From the results in Fig. 5 (a), we see that the touchscreen's output decreases monotonically with the increase of stripe resistance (and hence X_T), which fits our analysis and simulation above. We expect similar observation to be made on other smartphones as well. In general, Inkput should behave

similarly as long as the touchscreen uses capacitive techniques, which has been dominate and is unlikely to change significantly in near future. Note that certain touchscreen hardware may not necessarily output the charging time τ directly, and may instead measure a scaling function. But this does not change the monotonic relation, and the effect can be naturally handled in the one-time calibration.

Deriving the proper resistivity and geometry of the stripe. The resistivity and total resistance of the stripe is crucial to the performance of Inkput. The resistivity determines the spatial resolution, *i.e.*, the minimal distance between two distinguishable touch positions along a stripe. Although the measured charging time decreases monotonically with R_S, the touch sensor's ADC has a limited resolution. Moreover, the touchscreen output has minor fluctuations due to noise (*e.g.*, unstable finger contact). Therefore, two adjacent touch points need to have sufficiently different R_S to be distinguished. For a stripe with uniform resistivity, the resistance difference should be proportional to $\rho_s D$, where D is the distance between the two touch points. As a result, *increasing the resistivity of the printed conductive stripe would help improve its spatial resolution.*

On the other hand, *the resistivity should not be too large, in order to limit the total resistance of the stripe.* Recall that the touch position does not affect the measured charging time much when R_S becomes too large (Fig. 4). An extremely large R_S would make the initial value of V_C during charging exceed the detection threshold $\delta_C V_S$, rendering the touch points near end the stripe end not measurable.

Our empirical results in Fig. 5 (a) show that the overall resistance should not exceed $200k\Omega$ to ensure sufficient sensitivity. We set the length of each strip to 90 mm to fit the dimension of Google Cardboard. Based on the conductivity of our carbon ink ($4.5k\Omega/sq$ sheet resistance), we set the width of each stripe to 3 mm, which leads to a total resistance of $130k\Omega$ to $140k\Omega$, well below the $200k\Omega$ requirement. For other touchscreen models and printing material, the stripes' geometry can be determined in a similar way.

In addition, the gap between stripes needs to be small enough, so that the *finger tip always has direct contact to one or two stripes whenever touch occurs.* Our prototype sets the gap to 3 mm by default.

5.3 Detecting Two Touch Points on a Single Stripe

Note that one sensing unit is not capable of detecting more than one touches on that stripe. To solve this problem, we connect both ends of the stripe to the two long edges of the touch screen (Fig. 1). Together, the two connected sensing units provide two readings, which reflect the impacts of *up to two touches* on the stripe. The underlying rationale is that the touchscreen sensor is coupled with the finger

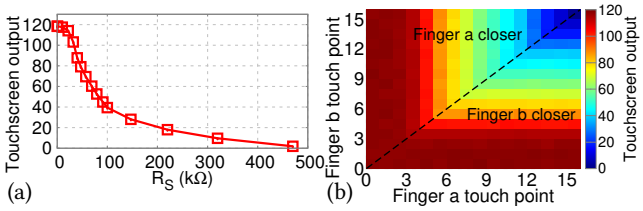


Figure 5: Touchscreen sensor output with (a) one single touch (b) two simultaneous touches.

that is physically closer, which has smaller stripe resistance between itself and the sensor.

To verify the effect, we randomly touch a series of points along a stripe with two fingers. Fig. 5 (b) plots the touchscreen output, where the axes indicate the touch positions of two fingers, and separation between adjacent touch points is 6 mm. The region below the dashed line indicates finger b is closer to the touchscreen, and vice versa for finger a. It can be observed that in both regions, *the touchscreen output is only determined by the touch that is closer to the electrode.* In addition, we find that the relation between touch position and output is exactly the same as Fig. 5 (a), which means stripe resistance curve can be used directly when both ends of the stripe are connected to the screen.

6 INTERFERENCE AMONG NEIGHBORING STRIPES

6.1 Interference Pattern

Ideally, the touch on each stripe would only affect the touch sensing unit in direct contact with the stripe. However, we find in measurement that nearby stripes suffer from non-trivial *coupling effect.* Fig. 6 plots the touch sensors' output as a single stripe connected to sensing unit #9 is touched. It can be observed that the nearby 6 touch sensing units also have non-zero output. The main reason is that the finger is in close proximity and coupled to the adjacent stripes as well through equivalent capacitors, which can also be counted as parasitic capacitance, as depicted in Fig. 8. It is equivalent that an additional capacitor C_C is connected between R_S and C_F in Fig. 3 (b). The combined capacitance of C_C and C_F in series is smaller than C_F, leading to a shorter charging time τ. So the outputs of nearby touch sensors are lower than the center one's. There also exists parasitic capacitance between the end of the stripe and nearby touch sensing units, which affects the reading in the same way.

Although the center stripe still generates the highest output, the interference among nearby stripes will cause problems when multiple close-by stripes are touched. In Fig. 7, for example, two stripes are touched, with the same x^T, and the effects of these two touches are mixed together, making it difficult to identify whether there are two or three touch points. More importantly, even if we assume two touches generate two local maximum outputs, it is non-trivial to estimate x^T

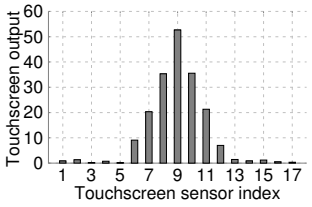


Figure 6: Touchscreen output as a single stripe is touched.

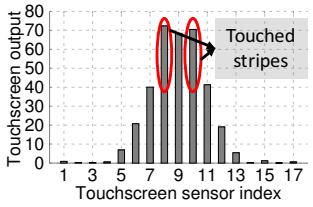


Figure 7: Touchscreen output as two stripes are touched.

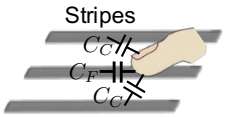


Figure 8: Cause of interference to adjacent stripes.

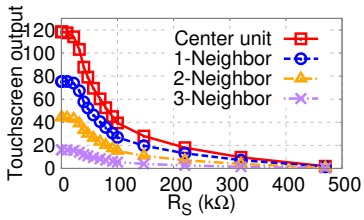


Figure 9: Touch sensor output under nearby stripe interference.

for each, since the mixed effects of two touches corrupt the stripe resistance curve, *i.e.*, the touchscreen output for #8 is also affected by touch on #10.

In Inkput, we use a ***stripe interference pattern*** to characterize the coupling between neighboring stripes and subsequently resolve the multi-touch. Stripe interference pattern consists of $K+1$ curves, including the stripe resistance curve of the center sensing unit, and the touch sensing outputs of its $K/2$ degree neighbors as a function of R_S. The model is obtained during the one-time calibration phase. In Inkput, we empirically set $K = 6$ as up to 3-degree neighbors are affected by touching on one stripe (Fig. 6). Fig. 9 plots the stripe interference pattern measured on the S5 phone. We see that the neighbors' output curves have the same trend as center one's stripe resistance curve, except that the amplitudes are lower. For different phone models and printing materials, this same measurement is needed as a *one-time calibration.*

6.2 Model-Based Optimization Algorithm for Multi-Touch Detection

To resolve the multi-touch positions, we design an optimization algorithm based on the model of interference pattern. Suppose there are N touches, positions denoted by $(x^T(i), y^T(i)), i = 1, 2 \cdots N$. Let vector $\vec{A}$ represent the outputs of the electrodes on the edge of the touchscreen. To determine the values of N and $x^T(i), y^T(i)$, we leverage two observations to make the solution extremely efficient.

Observation 1: We find in experiments that the interference patterns of simultaneous touches are added together in the touchscreen output, which can be expressed as,

$$\vec{A} = \sum_{i=1}^{N} \vec{I_T}(x^T(i), y^T(i)) \tag{3}$$

where $\vec{I_T}(x^T, y^T)$ represents the output when a single finger touches position $(x^T(i), y^T(i))$. When multiple fingers touch nearby stripes, they are all capacitively coupled to the touchscreen in a similar way, generating similar interference patterns with different amplitudes due to different touch positions x^T. Eq. (3) imposes a linear constrain to the unknown variables N and $(x^T(i), y^T(i)), i = 1, 2, \cdots, N$.

Observation 2: One or multiple nearby touches generate a *cluster* of outstanding values in the touchscreen output $\vec{A}$. In addition, since each touch affects only a few most adjacent stripes on both sides, the two ends of each cluster are usually the interference readings rather than actual touch. In our Inkput prototype, we find that touches separated by no more than 3 stripes apart can usually form such a cluster.

Based on observation 1, we formulate an optimization problem that derives the number and positions of touches, $N, x^T(i)$ and $y^T(i)$, by minimizing the error between the measured output $\vec{A}_{mes}$, and the predicted output $\vec{A}_{pred}$ based on the model in Eq (3), *i.e.*,

$$\begin{aligned} \underset{N, x^T(i), y^T(i)}{\text{minimize}} \quad & \left\| \vec{A}_{mes} - \vec{A}_{pred} \right\|^2 \\ \text{subject to} \quad & \vec{A}_{pred} = \sum_{i=1}^{N} \vec{I_T}(x^T(i), y^T(i)) \end{aligned} \tag{4}$$

To solve the problem more efficiently, we leverage observation 2 to narrow down the solution space near the center of each cluster, through the following steps.

It's worth noting that our circuit analytic model described in Sec. 5.1 is not coherent with observation 1. Due to lack of information on the Synaptics touchscreen controller's implementation, our analytic model is only an approximation. Although being able to fit the experimental observation in Sec. 5, the model does not well characterize the observation in multi-touch scenario, which is a limitation.

(i) Detecting cluster: Inkput detects cluster using a computationally efficient heuristic: It looks for d ($d \geq 9$) consecutive outputs in $\vec{A}$ with amplitudes larger than a threshold δ_I. Each touch generates a cluster with width $K + 1$ (Fig. 6). Since each stripe is only 3 mm—much narrower than finger tip—we assume two different touches need to be separated by at least one stripe's width; otherwise they are regarded as one single touch. Each touch sensor grid on the screen is 4 mm by 4 mm. In this case, a cluster generated by two touches should have a width of at least 9.

To determine a proper δ_I, we reuse the stripe interference pattern (Fig. 9). In our Inkput prototype, a 90 mm stripe has $140k\Omega$ total resistance. We can observe from Fig. 9 that when the center stripe is touched at the position with $140k\Omega$

resistance, the sensor output of 3-degree neighbor is around 6. Therefore, we set $\delta_I = 5$ in our implementation. This process of parameter tuning can be automated during the one-time calibration.

(ii) Narrowing down solution space for $y^T(i)$ and N: When a cluster with width d is detected, Inkput only searches for touches in the center $d - 2\delta$ sensing units of each cluster, and ignores the Δ ones near the the cluster edge which are primarily interference. In this way, we reduce the search spaces of N from d to $d - 2\Delta$ possibilities. Δ can be set empirically and equals 3 in our prototype.

Given these steps, we add two more constrains to problem (4): *First* , the continuous variable $y^T(i)$ is within the range of center $d - 2\Delta$ sensing units of a cluster (with width d), where $i = 1 \ldots, N$. *Second*, we have $N \leq d - 2\Delta$. The resulting problem is a quadratic optimization problem, which can be solved efficiently with Conjugated Gradient method [52]. In our Inkput prototype, we use a Java quadratic programming solver running on Android. Among a handful of commonly used optimization solver, we chose to use CPLEX, due to its support for quadratic programming [53]. We will show through experiments (Sec. 10) that the computational overhead is negligible even when running on a smartphone.

We emphasize that Inkput does not require each stripe to be perfectly aligned with the (invisible) electrode on the touchscreen. In case when a stripe connects to two electrodes, a touch may trigger high output to two center electrodes. Nonetheless, the optimization is unaffected, as the corresponding change in stripe interference pattern has been accounted for in the calibration phase. And the optimization will solve for the continuous variable y^T which naturally runs an interpolation between the stripes' y-axis locations.

7 HOVERING DETECTION

On ordinary hovering-enabled touchscreens, the hovering position can be determined in the same way as touching. Yet on the same position, hover generates much lower amplitude. This can be again explained by the circuit model (Sec. 6.1). Hovering is equivalent to connecting an extra parasitic capacitor between the finger tip and the conductive sensing medium. This capacitor is connected in series with the finger capacitor, which decreases the overall capacitance and hence the touch sensor's charging time C_{in}.

In Inkput, detecting hovering above the printed stripes faces a unique challenge: either hovering above a position with small stripe resistance, or touching a position with large stripe resistance, could result in the same small output value.

To distinguish them solely based on the touchscreen output, we leverage an observation that, due to different coupling medium with the stripes (air vs. carbon layer), touch and hovering lead to different stripe interference pattern. To elucidate the difference, we measure the stripe interference pattern in the same way as in Fig. 6 and Fig. 9. The

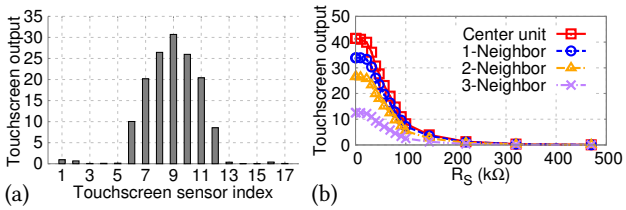


Figure 10: Stripe interference pattern of hovering (a) snapshot (b) complete model.

results (Fig. 10) show that the interference patterns of touch and hover show similar trends with R_S, but the amplitude of the center stripe is lower for hovering. Similar to touch, we obtain the stripe interference pattern for hovering in the calibration phase, and denote the vector of sensor output as $I_H(x^T, y^T)$ for hovering position (x^T, y^T).

To detect multiple simultaneous hovers, we formulate an optimization problem similar to (4), but add a binary variable α_T indicating touch ($\alpha_T = 1$) or hover ($\alpha_T = 0$). The predicted touchscreen output can be rewritten as,

$$\vec{A}_{pred} = \sum_{i=1}^{N} [\alpha_T \cdot \vec{I_T}(x^T(i), y^T(i)) + (1 - \alpha_T) \cdot \vec{I_H}(x^T(i), y^T(i))]$$

In principle we could assign a separate α_T for each touch/hover to detect simultaneous touch and hover. However, we find doing so increases computation time, and yields sub-optimal accuracy due to drastically larger variable space. Since simultaneous touch and hovering is rarely used in practical interaction, we do not implement it in Inkput.

Improving hovering detection range and accuracy. Note the sensitivity of hover detection depends on the electrode hardware. We find that N5 can detect a hover of up to 3 mm above the printed stripes, while S5 supports up to 5 mm. This is because most smartphone touchscreens do not natively support high-sensitivity hovering detection. Here we argue and prove that the detectable hover range can be increased with some simple parameter changes in the touchscreen electrodes. Specifically, we simply need to increase the charging time induced by a hover (Sec. 5.1, Eq. 2). This can be realized by increasing the inner resistor R_{in} of the electrodes on the screen, which "magnifies" the detectable charging time proportionally.

We conduct a proof of concept study to demonstrate the feasibility. Specifically, we prototype a touch sensing unit following the equivalent circuit model (Fig. 3) which contains inner resistor R_{in} and capacitor C_{in} connected in series with a voltage source. C_{in} is then connected with a printed stripe to serve as touch sensor. C_{in} is set to 100 pF. We use an Arduino board to provide the power source and measure charging time. Fig. 11 plots the measured maximum hover range corresponding to a detectable charging time.

We observe that the hover range increases significantly with R_{in}. Although different touchscreens have different implementations, it can be expected that the hover detection sensitivity can be enhanced with equivalent modifications.

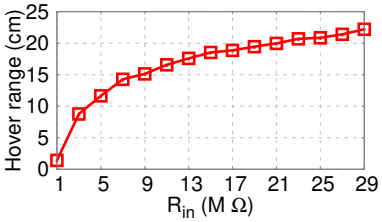


Figure 11: Proof of concept study of increasing hover range.

In fact, certain touchscreens for mobile devices already support up to 35 *mm* of hovering range [54, 55]. Meanwhile, hovering detection accuracy can also be improved as the measured output becomes less sensitive to noise. The cost of increasing R_{in} is increased charging time for both touch and hover detection, which increases detection latency.

8 ONE-TIME CALIBRATION

The purpose of calibration is to obtain the stripe resistance curve and stripe interference pattern. Since these two models primarily depend on the configuration of touchscreen and the ink stripes, the calibration only needs to be performed once when an Inkput extension is printed. During the calibration, the user uses a single finger to touch different positions on each stripe, Inkput then runs a cubic Hermite interpolator on the sample points to recover the stripe resistance curve. In our experiment, we find that 5 equally spaced samples on each 90 *mm* stripe (around 140 kΩ) are sufficient for recovering the curve. Meanwhile, Inkput stores the touchscreen outputs of up to 3-degree neighbors for each touch point, and establishes the complete stripe interference pattern with the same interpolation. Then, the user needs to repeat the swiping process, but with hovering, to obtain the stripe interference pattern for hover.

The calibration points can be marked out during printing. Suppose there are 6 calibration points per stripe. Then the user simply needs to swipe across all the stripes 6 times, and similarly for hovering. The whole process takes no more than 2 minutes, which is a negligible overhead. In addition, it's a one-time process that does not need to be repeated unless a new touchscreen device is used, or the conductivity of the printed stripes changes. In practice, we find environmental factors such as temperature and humidity do not change conductivity of the ink. Once the patterns are printed, their electric property stays stable for a long time.

9 EVALUATION

We use two Android phones, Nexus 5 (N5) and Galaxy S5 (S5), for the experiments. By default, the printed extension is made up of 20 stripes, each 90 *mm* long, 3 *mm* wide, with 3 *mm* gaps. Both ends of the stripes are connected to the touchscreen unless otherwise specified. For experiments with multiple users, we recruited 10 volunteer participants (4 females, 6 males between the ages 23-28). None of them were provided with any benefits or rewards. They were instructed to interacted with the printed extensions as they would normally interacted with a touchscreen, but were untold regarding the technical details of Inkput, nor did they have any practice on Inkput or similar systems prior to the user study.

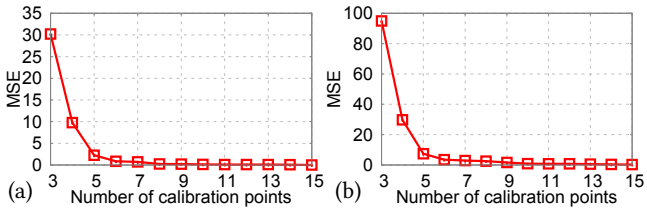


Figure 12: The MSE of (a) stripe resistance curve (b) stripe interference pattern with different number of calibration points.

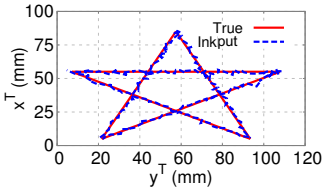
9.1 Calibration Performance

We first evaluate how many calibration points are needed to estimate the stripe resistance curve and stripe interference pattern accurately. The accuracy is quantified by the Mean Square Error (MSE) between ground truth curves and calibrated curves, obtained by interpolation of 20 points, and 3 to 15 points, respectively. Fig. 12 shows that, for both stripe resistance curve and stripe interference pattern, the MSE decreases with the number of calibration points. Interestingly, it flattens out as the number exceeds 5, due to the fact that the curves of these two models are relatively smooth. Therefore, around 6 calibration points should suffice to reduce the calibration overhead without compromising accuracy.

9.2 Single Touch Tracking Accuracy

To evaluate the accuracy of single-finger touch sensing, we first use normal non-conductive ink to draw ground-truth line/curve patterns on the printed extension. We then ask the 10 users to use a finger to follow the pattern. Fig. 13 shows an example star pattern, in contrast to the touch trajectory estimated by Inkput. We repeat the experiment across 5 other patterns with similar size. Fig. 14 plots the error CDF across all patterns drawn by each user. Clearly, *Inkput can track the finger positions very accurately among all the users.* The median error is only around 1.7 *mm*, 90-percentile around 4 *mm*, and even 99-percentile is only around 5.5 *mm*, barely larger than half of the finger tip width.

We next investigate the *spatial error distribution* across the touch area. We divide the printed extension into 5 *mm* by 5 *mm* grids, and place the finger tip on each grid for 10 times. For this particular experiment, we only connect one end of the stripes to the touchscreen to characterize the impact of stripe resistance between touch point and the screen. The average detection errors are plotted in Fig. 15, which shows that *the error is slightly higher at the end of the*



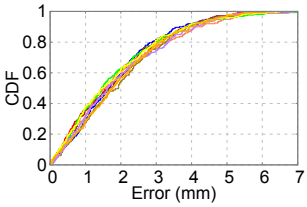


Figure 13: Patterns computed by Inkput and ground truth.

Figure 14: Finger tracking error of 10 participants.

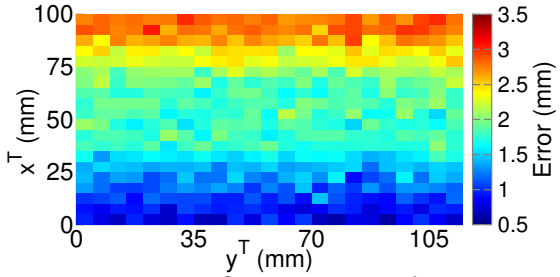
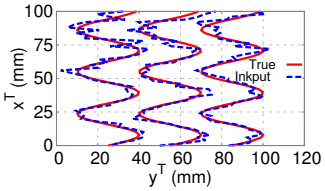


Figure 15: Average finger position detection error.



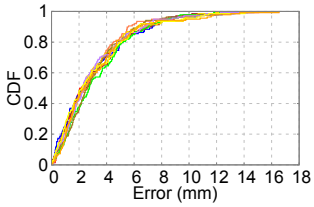


Figure 16: Multi-touch patterns computed by Inkput and ground truth.

Figure 17: Multi-finger tracking error of 10 participants.

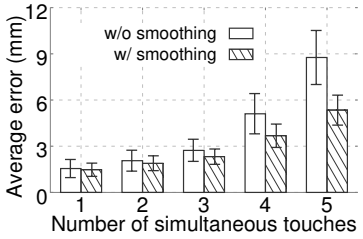


Figure 18: Average touch detection error with different number of touches. Error bars indicate standard deviation.

stripe that is far away from the screen. The reason is that the stripe resistance curve is relatively flat as x^T increases, and the same level of noise would have larger impact at the far end of the stripe. Nonetheless, even in this region, the mean error is only around 3 *mm*, while in the majority parts of the printed extension, the mean error falls below 2.5 *mm*.

One interesting question is how resilient Inkput is to *variation of touch pressure*. Unlike touchscreen that can locate the center of a touch, Inkput essentially detects the contact point along a stripe that is the closest to the screen (Sec. 5.3). So, pressing harder would make the detected position appear closer. In practice, however, we find natural touches barely cause any errors, and even extreme force (which is unlikely to occur in real use cases) only causes up to 3 *mm* deviation. On the other hand, we find that stripe resistance curve and stripe interference pattern do not exhibit measurable changes across people, so Inkput's performance remains consistent for different users (Fig. 14).

9.3 Multi-Touch Tracking Accuracy

To evaluate the accuracy of multi-touch detection, we ask the 10 participants to draw a series of patterns that contain 2 to 5 curves, using two or more fingers simultaneously. We make sure no more than 2 curves cross the same stripe simultaneously, because Inkput can only discriminate up to 2 simultaneous touch points on the same stripe.

Fig. 16 shows an example pattern which contains three curves, and Fig. 17 plots the error CDF across all patterns for each user. We can see that the accuracy of multi-touch tracking is slightly lower than single touch (median 2.4 *mm*).

To quantitatively evaluate touch detection accuracy with different number of simultaneous touches, we have each participant touch multiple points simultaneously. Fig. 18 shows that the average error increases from 1.7 *mm* to 8.8 *mm* as touch points increases from 1 to 5. Although we calibrate the stripe interference pattern in advance, the exact patterns presented in application phase may experience variations due to noise and how the user places the finger. As more fingers touch simultaneously, the errors of each finger's stripe interference pattern are added together, which eventually increases the error of touch position estimation.

However, the accuracy of 2 to 3 simultaneous touches (2.1 *mm* and 2.7 *mm* average error) is more than enough to make Inkput fit for most sophisticated multi-finger gestures (*e.g.*, pinch to zoom in/out, or rotating objects), mostly used for control and gestural interactions.

To make touch sensing more resilient to noise, a standard solution is *spatial smoothing*, *i.e.*, use filters or curve-fitting to estimate the trajectory along consecutive touch points. To test the effectiveness of this approach in Inkput, we apply a median filter with a widow size of 7 points on the estimated curves in Fig. 13 and Fig. 16. This window only spans 0.16 second when the touchscreen scans at a typical frequency of 18 Hz. The latency is unlikely to affect user experience. Fig. 18 plots the error after filtering. We find that the *average error significantly decreases (e.g., from 9 mm to 5 mm for 5-finger touch) even with this simple spatial smoothing*. Note that the smoothing is best applicable for applications that involve multi-finger gestural interaction, which continuous finger movement trajectories.

Finally, to evaluate Inkput's ability to resolve close-by touches, we touch the extension with two fingers. The two touch points have the same x^T (position along the stripe), and are separated by 1.5 *cm* and 2 *cm* along y^T direction. The average detection errors under 1.5 *cm* and 2 *cm* separation

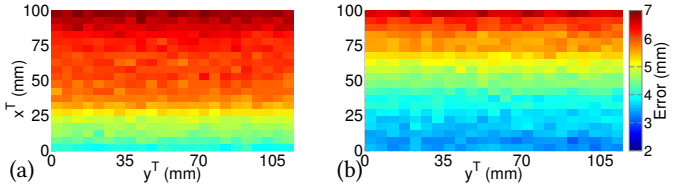


Figure 19: Hover detection error on 5 *mm* by 5 *mm* grids of (a) N5 (b) S5

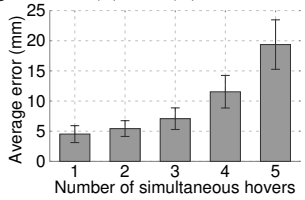


Figure 20: Average hover detection error with multiple simultaneous hovers (S5). Error bar indicates standard deviation.

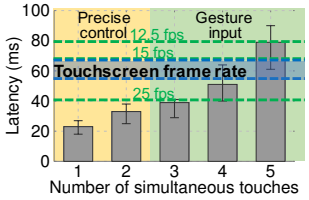


Figure 21: Average processing latency. Error bar shows 90-percentile minimum and maximum.

are 3.5 *mm* and 3.1 *mm*, respectively, demonstrating that the Inkput's model-based optimization framework is able to distinguish and detect close-by touches.

9.4 Hover Detection & Tracking Accuracy

To evaluate hover detection accuracy, we first have the participants hover a finger 3 *mm* above the $5mm \times mm$ grids. Fig. 19 plots the measured average error for N5 and S5. Here we also only connect one end of the stripes to the screen. We find that S5 provides slightly higher hover detection accuracy than N5. For S5, the hover detection error is around 4.5 *mm* in the center of the printed extension, and around 6.1 *mm* for N5. Although not as accurate as touch detection, the performance of hover detection is still able to provide user the hint of where the finger is about to touch, which is especially beneficial for back-of-the-phone touch sensing in mobile VR. Note that Inkput automatically distinguish touch and hover, taking advantage of their different stripe interference pattern. In this same set of measurements, 97.1% of the hovers are correctly detected, while in only 2.9% of the cases they are misclassified as touches.

Next we evaluate the case where multiple fingers are hovering above the printed extension. The setup is the same as the multi-touch experiment, except that the finger tips are up to 5 *mm* in the air above the stripes. Fig. 20 shows that average error increases from 4.7 *mm* to 11 *mm* as the hover points increase from 1 to 5. Thus, the current version of Inkput may not be suitable for hovering detection applications that require very precise tracking of 4 or more fingers.

10 PROCESSING LATENCY

Apart from accuracy, latency is another crucial aspect of touch and hover detection. It determines the output frame rate of the detected interaction events and positions. We found the touchscreens of both N5 and S5 provide around 15 to 18 frames of output per second.

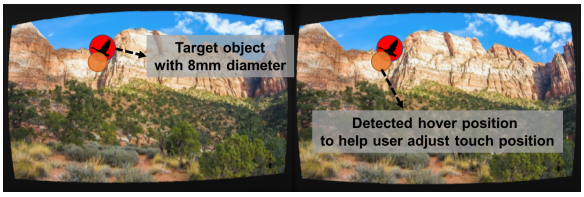


Figure 22: Panoramic view displayed with a 8 *mm* diameter target showing up in random locations. Participants are instructed to touch the back of the phone to select the target.

Inkput's maximum frame rate is bounded by the touchscreen's frame rate, as it relies on the output from touchscreen to infer interaction on the stripes. Recall that Inkput takes the output from the touchscreen and runs an optimization framework for detection. Given a set of output, Inkput needs to finish all the processing before the next set of output from touchscreen arrives, in order to achieve the same detection frame rate as the touchscreen.

To evaluate the achievable frame rate, we measure the average processing time from Inkput obtains the output from touchscreen till it successfully estimates the touch and hover positions as the 10 participants use 1 to 5 fingers to draw various patterns on the printed stripes. The average processing time is plotted in Fig. 21. The results demonstrate that *Inkput is able to achieve the same average output frame rate as the original touchscreen (13-15 frames/s) even under 4 simultaneous touches*, which account for most of the input/control commands for mobile VR. Even when the touch points increase to 5, Inkput is still able to provide 10 frames of output per second. As we have explained in Sec. 6.2, the optimization problem can be solved with low latency on smartphone because Conjugated Gradient method can obtain the optimal solution efficiently. As the number of touches increases, the computation time increases as a result of larger variable space. Specifically, since the touch points increases, we need to solve the x^T and y^T for each finger, which increases the dimension of the variable space.

We anticipate that inputs requiring precise and timely control, such as typing or object control in VR gaming, typically involves one to two fingers, while inputs with three or more fingers are generally gestures, such as pinching or rotating objects which can tolerate relatively high latency. Inkput's processing speed is thus sufficient for these practical use cases.

11 POWER CONSUMPTION

We measure the power consumption of Inkput using a Monsoon Power Monitor [56] on Galaxy S5. S5 itself consumes 983.7 ± 21.3 mW when the screen is on but all background applications turned off. When Inkput is enabled, we have each user continuously draw patterns with arbitrary number

of fingers for 60 seconds, and the measured power consumption is 1131.5 ± 40.4 mW, only increasing the average power consumption by less than 150 mW. The additional power cost is negligible compared to that of the display and rendering-heavy games themselves. The majority of the cost is attributed to the computation, which can be further reduced by optimizing the Java signal processing components.

12 USE CASE STUDY

In this section, we introduce case studies of 3 applications that we developed on top of Inkput, to validate its usability as an interaction technique.

12.1 Back-of-the-HMD VR Gaming

To verify Inkput for mobile VR, we first integrate the printed extension with Google Cardboard. As shown in Fig. 1, two edges of the extension are connected to the two long edges of the smartphone hosted inside Cardboard. With the help of Inkput, one can interact with the virtual world in the similar way of using a touchscreen. For instance, the user can drag and place a virtual object, or can perform multi-finger gestures such as zooming in/out.

We now provide a quantitative study to show how the hover detection can improve user experience when integrated with touch detection. We have implemented an Android VR application that mimics a hunting game, as shown in Fig. 22. The application displays a target that is 8 *mm* in diameter and randomly located on the screen. The user wears the Cardboard and navigates a finger on the backside to "shoot" the target as quickly as she can. The red circle in Fig. 22 illustrates the target whereas the orange circle is the hovering position detected by Inkput and displayed to the user to help her adjust the touch position. We compare the target points and the ones actually touched by the user, with and without enabling hover detection. Fig. 23 is a scatter plot of the points for 4 participants, and Fig. 24 shows the error CDFs of all 10 participants.

In Fig. 23, we observe that the 4 participants accurately touched the target area in only 20%, 18%, 30% and 12% of the time, with touch sensing alone. When augmented with hovering detection, the accuracy is increased to up to 98%. The user who achieved highest accuracy is the one that used the system multiple times, implying that Inkput's usability improves over time. Among all the 10 participants, the median touch errors ranges from 7 to 15 *mm*, and the 80-percentile error can be as high as 17 *mm* for some users. On the other hand, with hovering detection enabled, the mean touch errors reduce to 2 to 3 *mm* for all users. The results also implies that the human sense of proprioception can only help very coarse-grained finger navigation for pre-touch selection. It needs to be combined with hovering detection to make the back-of-HMD interaction truly usable.

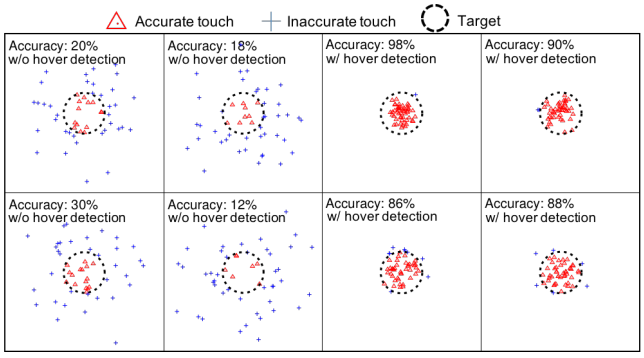


Figure 23: Actual touch positions relative to the target of 4 test participants. Left 4 graphs show the results without hover detection. Right 4 graphs show results with hover detection helping participants adjust touch position.

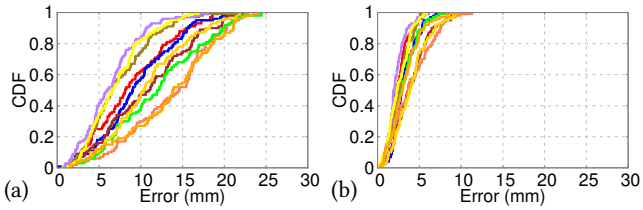


Figure 24: User touch position error CDF (a) without hover detection (b) with hover detection.

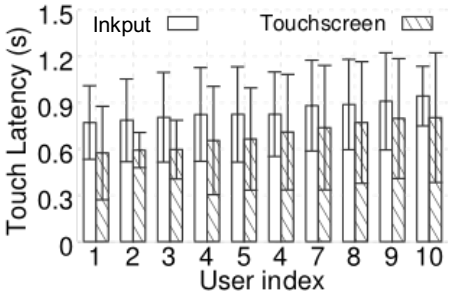


Figure 25: Display to touch latency. Error bars indicate standard deviation.

In order to evaluate whether the user can get used to back-of-the-HMD interaction for VR, we also measure the *display-to-touch latency*, defined as the time between the target is displayed on the screen and the user touches the backside. The latencies are collected in the above experiments with hovering detection enabled. As a comparison, we run a benchmark experiment, where the participants touch the smartphone touchscreen directly to select the target. The average latencies and *std.* of the 10 users are plotted in Fig. 25. It can be observed that the latency of Inkput is only slightly higher than using the touchscreen in the normal way, and the highest average latency among all participants is only around 0.93 s.

12.2 Back-of-the-HMD VR Text Input

Typing in VR has always been a challenging task. Most commodity VR solutions [8, 9] can only track the external handheld controllers' coordinates/orientations, or track the orientation of the headset [3–5]. There exists no elegant way to enable the users to type in the same way as typing on a touchscreen or real keyboard. For example, in Daydream VR, the user has to rotate the head to navigate a cursor on a virtual keyboard, and click a button on the controller to make a keystroke. Similar method is used for most mobile VR headsets. For high-end VR that can precisely track the handheld controllers (*e.g.*, HTC VIVE and Oculus Rift), there are two solutions. SteamVR [57] allows users to manipulate a virtual laser beam with the handheld controller to navigate to virtual keys, and click a button to confirm. Another type of solution is to use two controllers as drum sticks, and click on the keyboard like hitting the drums. Examples include Drum Keyboard [58] and Cutie Key [59], both based on the HTC VIVE controllers.

In contrast, Inkput can exploit the back of the HMD as a keyboard input interface similar to a normal touchscreen. We have prototyped such a VR keyboard for Google Cardboard, and Fig. 26 shows the application view. The orange circle is the detected hover position displayed on top of the keyboard to help users avoid mis-clicking. To evaluate the keyboard performance, we ask the participants to wear the Cardboard, and type on the back of the phone in their most comfortable way. As a comparison, we run the same experiment using Daydream VR (using the SteamVR keyboard), HTC VIVE (the Cutie Keys), and head-rotation input method for Daydream VR. We require the user to type the same set of sentences using each device. The average typing speed and error rate plotted in Fig. 27, where the error bar represents *std.* Error rate is defined the percentage of letters that the user typed wrongly across one experiment.

Inkput and Cutie Key achieve the highest typing speed with around 16 words/minute. The two other approaches that require users to steer a cursor using either controller or head-rotation only achieves up to 6.5 words/minute. In terms of accuracy, Inkput has an error rate of 11%, in comparison to 19% (Cutie Key), 5% (SteamVR keyboard), and 8% (Daydream). The last two are relatively more accurate, primarily because typing speed is fundamentally limited by the way of interaction. As a result of faster typing speed, Inkput inevitably experiences higher error rate for letter input, but it still significantly outperforms the HTC VIVE based Cutie Key, which is an unfamiliar input method for most people who are used to typing through touch.

12.3 Keyboard Extension for Smartphones

We show that Inkput's printed extension can be customized into a large keyboard, which can save the small touchscreen real-estate and avoid the notorious finger occlusion problem.

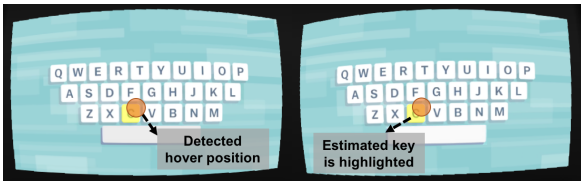


Figure 26: VR keyboard enabled by Inkput. Orange circle indicates detected hover position, which is displayed in the application to help user calibrate touch position.

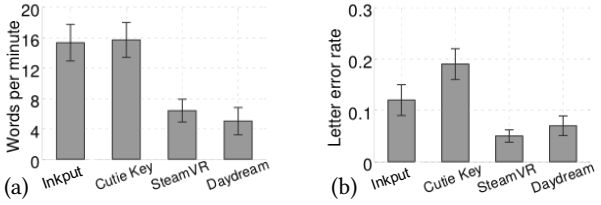


Figure 27: Performance of typing on VR keyboard enabled by Inkput and other approaches. (a) Average typing speed. (b) Letter error rate. Error bars indicate standard deviations.

This large size extension can be folded and fit into a flip case when not used.

To create a large extension, the layout of the stripes need to be adjusted. The length of both N5 and S5's touchscreen is around 11 *cm*. We bend the stripes as they reach out, so that the interaction area can be extended along the y^T direction, as shown in Fig. 28. Since the design obviously increases the overall length of each stripe, we increase the stripe width accordingly to reduce its total resistance to around $200k\Omega$. Although the layout of the stripes are different, the working principles of Inkput remain the same, so neither the calibration procedure nor the optimization framework need any modification.

We integrate this large extension with a S5 phone case that has a flip cover. The extension is folded to fit into the flip cover, and can be put into use by unfolding anytime. The keyboard layout cannot be directly printed on the extension, because the keys become unrecognizable when overlapping with the stripes. We thus overlay a normal paper-keyboard printout on top of Inkput's paper substrate. Fig. 28 shows the integrated system.

Unfortunately, this extra piece of paper between the conductive stripes and finger would affect the touch sensor readings, and accordingly changes both stripe resistance curve and stripe interference pattern. To elucidate the effect, we measure the stripe resistance curve of a stripe on the extension with and without the keyboard layout sheet covered, and plot the results in Fig. 29. It can be observed that touchscreen has weaker output with the keyboard sheet, due to the fact that finger is not contacting the stripes directly. We find through measurement that stripe interference pattern experiences similar changes. Fortunately, the general trend

Figure 28: Customized printed extension to extend the interaction area of smart phone. The extension is integrated with a flip cover phone case, and can be folded to fit inside the cover. Keyboard layout printed on normal paper attached on top of conductive stripes.

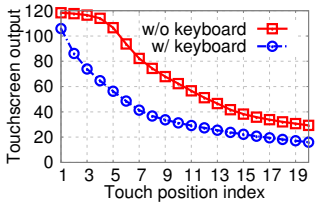


Figure 29: Stripe resistance curve affected by the keyboard sheet.

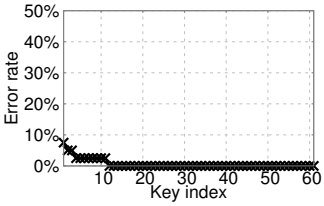


Figure 30: Key detection error rate.

and monotonicity of these two models remain unchanged. *Thus, the slight parametric changes due to the additional paper layer can be totally captured through a recalibration.* Besides, the user only needs to touch two example keys, on the top left and right corners of the layout, in order to figure out the relative position/orientation of the keyboard sheet with respect to the stripes.

To evaluate the performance of the Inkput keyboard, we touch each key 40 times following a random order. The error rates for all the keys are plotted in Fig. 30. Out of the 61 keys, 50 experiences 0% errors, and only one outlier key has an error rate above 5%. Therefore, Inkput suffices as an accurate and reliable way of extending the conventional touchscreen-based mobile interaction.

13 DISCUSSION

Customizing the printed extension. It is straightforward to customize the dimensions of stripes as well as the paper substrate in Inkput, to satisfy different application requirements. For example, mobile VR requires the printed extension to be of similar size with the VR headset, while using Inkput as a extended keyboard for smartphone should allow much larger interaction area. Touch detection can work properly as long as the printed stripes have uniform resistivity, and have end-to-end resistance of around $200k\Omega$ (Sec. 5.2). Therefore, to increase the size of printed extension while not increasing the stripe resistance, we ought to increase the width of each stripe. Note that increasing the length and width proportionally does not change the stripe resistance (which equals $\rho_s \frac{L}{W}$). As we described in Sec. 5.2, the stripe can be as long as 133 mm with a width of 3 mm in order to not exceed $200k\Omega$ of total resistance.

Broader use cases for VR. Our VR keyboard case study shows that back of the phone interaction, although being a new technique, can be easily familiarized by the users. To date, most mobile VR applications are limited to passive viewing experience due to the lack of effective interaction. With Inkput's 2D touch sensing ability, a wide range of applications and games that are only possible on high end VR systems can now be enabled on mobile VR. Examples include *Human Medical Scan* and *Robot Repair*, available on SteamVR for HTC VIVE, where the player needs to select and drag different objects with two controllers simultaneously to precise locations to perform inspection and other operations [60, 61]. Such interactions are infeasible for the head-rotation based systems like Daydream VR. Many other VR games (*e.g.*, *Xortex* [60, 61]) require users to perform selection tasks as efficient as on touchscreens. We have demonstrated that Inkput's accuracy and latency can uniquely satisfy these needs. In addition, Inkput's user experience can be further significantly improved if it can be integrated with VR HMD or smartphone case by the vendor, which makes the printed extension even more low-cost.

14 CONCLUSION

Through the Inkput design, we have demonstrated the feasibility of expanding the interaction area of a smartphone using a simple paper-printed extension. The paper interface only occupies a tiny edge of the smartphone. But through proper design of the printed conductive patterns, along with careful processing of the touchscreen signals, Inkput can achieve millimeter-level precision when sensing multi-finger touches on the paper. Our prototype implementation verifies that Inkput can be extremely low-cost and incurs negligible detection latency. Our user studies on actual VR applications also demonstrate that it can approach the usability of an actual touchscreen. We believe Inkput can enable new interaction modalities for mobile VR.

ACKNOWLEDGEMENTS

We appreciate the anonymous reviewers for their insightful comments. This research was supported in part by a Google Faculty Research Award, Sony Research Award, and the US National Science Foundation through CNS-1345293, CNS-14055667, CNS-1525586, CNS-1555426, CNS-1629833, CNS-1647152, CNS-1719336, CNS-1506657, CNS-1518728, CNS-1343363, and CNS-1350039.

REFERENCES

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE robotics & automation magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[2] "Google project tango," https://get.google.com/tango/.

[3] "Google cardboard," https://vr.google.com/cardboard/.

[4] "Google daydream," https://vr.google.com/daydream/.

[5] "Samsung gear vr," http://www.samsung.com/global/galaxy/gear-vr/.

[6] "Hi5 vr glove," https://hi5vrglove.com.

[7] "Leap motion," https://www.leapmotion.com/.

[8] "Htc vive," https://www.vive.com/us/product/vive-virtual-reality-system/.

[9] "Oculus rift," https://www.oculus.com/rift/#oui-csl-rift-games=star-trek.

[10] 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*. ACM, 2015, pp. 1851–1854.

[11] ——, "Creating a mobile head-mounted display with proprietary controllers for interactive virtual reality content," in *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. ACM, 2015, pp. 35–36.

[12] S. Olberding, N.-W. Gong, J. Tiab, J. A. Paradiso, and J. Steimle, "A cuttable multi-touch sensor," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 245–254.

[13] K. Kato and H. Miyashita, "3d printed physical interfaces that can extend touch devices," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 2016, pp. 47–49.

[14] D. Holman, N. Fellion, and R. Vertegaal, "Sensing touch using resistive graphs," in *Proceedings of the 2014 conference on Designing interactive systems*. ACM, 2014, pp. 195–198.

[15] A. Wiethoff, H. Schneider, M. Rohs, A. Butz, and S. Greenberg, "Sketch-a-tui: low cost prototyping of tangible interactions using cardboard and conductive ink," in *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*. ACM, 2012, pp. 309–312.

[16] "Microsoft hololens," http://www.microsoft.com/microsoft-hololens/en-us.

[17] J. Song, G. Sörös, F. Pece, S. R. Fanello, S. Izadi, C. Keskin, and O. Hilliges, "In-air gestures around unmodified mobile devices," in *Proceedings of ACM symposium on User interface software and technology*, 2014.

[18] X. Chen, J. Schwarz, C. Harrison, J. Mankoff, and S. E. Hudson, "Air+touch: interweaving touch & in-air gestures," in *Proceedings of ACM symposium on User interface software and technology*, 2014.

[19] S. W. Greenwald, L. Loreti, M. Funk, R. Zilberman, and P. Maes, "Eye gaze tracking with google cardboard using purkinje images," in *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*. ACM, 2016, pp. 19–22.

[20] M. Sugimoto and K. Hiroki, "HybridTouch: An Intuitive Manipulation Technique for PDAs Using Their Front and Rear Surfaces," in *Proceedings of the Conference on Human-computer Interaction with Mobile Devices and Services (MobileHCI)*, 2006.

[21] 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.

[22] J. Scott, S. Izadi, L. S. Rezai, D. Ruszkowski, X. Bi, and R. Balakrishnan, "Reartype: text entry using keys on the back of a device," in *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*. ACM, 2010, pp. 171–180.

[23] 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*, 2008.

[24] J. Gugenheimer, D. Dobbelstein, C. Winkler, G. Haas, and E. Rukzio, "Facetouch: Enabling touch interaction in display fixed uis for mobile virtual reality," in *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. ACM, 2016, pp. 49–60.

[25] S. Gupta, D. Morris, S. Patel, and D. Tan, "Soundwave: using the doppler effect to sense gestures," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 1911–1914.

[26] W. Wang, A. X. Liu, and K. Sun, "Device-free gesture tracking using acoustic signals," in *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*. ACM, 2016, pp. 82–94.

[27] R. Nandakumar, V. Iyer, D. Tan, and S. Gollakota, "Fingerio: Using active sonar for fine-grained finger tracking," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 1515–1525.

[28] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 142–154.

[29] J. Wang, K. Zhao, X. Zhang, and C. Peng, "Ubiquitous keyboard for small mobile devices: harnessing multipath fading for fine-grained keystroke localization," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 2014, pp. 14–27.

[30] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 90–102.

[31] T. Wei and X. Zhang, "mtrack: High-precision passive tracking using millimeter wave radios," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. ACM, 2015, pp. 117–129.

[32] J. Wang, D. Vasisht, and D. Katabi, "Rf-idraw: virtual touch screen in the air using rf signals," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 235–246.

[33] F. Adib, Z. Kabelac, D. Katabi, and R. C. Miller, "3d tracking via body radio reflections." in *NSDI*, vol. 14, 2014, pp. 317–329.

[34] B. Chen, V. Yenamandra, and K. Srinivasan, "Tracking keystrokes using wireless signals," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2015, pp. 31–44.

[35] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion." *HotSec*, vol. 11, pp. 9–9, 2011.

[36] Z. Xu, K. Bai, and S. Zhu, "Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*. ACM, 2012, pp. 113–124.

[37] S. Pradhan, E. Chai, K. Sundaresan, L. Qiu, M. A. Khojastepour, and S. Rangarajan, "Rio: A pervasive rfid-based touch gesture interface," in *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM.

[38] R. Takada, B. Shizuki, and J. Tanaka, "Monotouch: Single capacitive touch sensor that differentiates touch gestures," in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 2016, pp. 2736–2743.

[39] S. Kratz, T. Westermann, M. Rohs, and G. Essl, "Capwidgets: tangile widgets versus multi-touch controls on mobile devices," in *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2011, pp. 1351–1356.

[40] Y. Zhang, C. J. Yang, S. E. Hudson, C. Harrison, and A. Sample, "Wall++: Room-scale interactive and context-aware sensing," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, 2018, p. 273.

[41] "Synaptics dsx driver," https://github.com/MotorolaMobilityLLC/kernel-msm/tree/lollipop-5.1.1-release-ghost/drivers/input/touchscreen.
[42] "Epson premium photo paper glossy," https://epson.com/For-Home/Paper/Photo/Premium-Photo-Paper-Glossy/m/S041667.
[43] "Epson stylus c88+ inkjet printer," https://epson.com/For-Home/Printers/Inkjet/Epson-Stylus-C88
[44] "Jr 700 carbon inkjet ink," https://store.novacentrix.com/product_p/910-0145-02.html.
[45] G. Barrett and R. Omote, "Projected-capacitive touch technology," *Information Display*, vol. 26, no. 3, pp. 16–21, 2010.
[46] E. So, H. Zhang, and Y.-s. Guan, "Sensing contact with analog resistive technology," in *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on*, vol. 2. IEEE, 1999, pp. 806–811.
[47] J. Y. Han, "Low-cost multi-touch sensing through frustrated total internal reflection," in *Proceedings of the 18th annual ACM symposium on User interface software and technology*. ACM, 2005, pp. 115–118.
[48] P. Nguyen, U. Muncuk, A. Ashok, K. R. Chowdhury, M. Gruteser, and T. Vu, "Battery-free identification token for touch sensing devices," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. ACM, 2016, pp. 109–122.
[49] C. Hsieh, "Touch-panel market analysis reports 2008-2014," *Technical report, DisplaySearch*, 2014.
[50] T. Grosse-Puppendahl, 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*. ACM, 2017, pp. 3293–3315.
[51] W. H. Hayt, J. E. Kemmerly, and S. M. Durbin, *Engineering circuit analysis*. McGraw-Hill New York, 1986.
[52] J. Nocedal and S. J. Wright, *Sequential quadratic programming*. Springer, 2006.
[53] "Cplex optimizer," https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer.
[54] K. Hinckley, S. Heo, M. Pahud, C. Holz, H. Benko, A. Sellen, R. Banks, K. O'Hara, G. Smyth, and W. Buxton, "Pre-touch sensing for mobile interaction," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 2869–2881.
[55] "Truetouch touchscreen controllers," http://www.cypress.com/products/truetouch-touchscreen-controllers.
[56] "Monsoon power monitor," https://www.msoon.com/online-store.
[57] "Steamvr," http://store.steampowered.com/steamvr.
[58] "Google daydream drum keyboard," https://developers.googleblog.com/2016/05/daydream-labs-exploring-and-sharing-vrs.html.
[59] "Cutie key," https://github.com/NormalVR/CutieKeys.
[60] "The lab (video game)," https://en.wikipedia.org/wiki/The_Lab_(video_game).
[61] "The lab on steam vr," http://store.steampowered.com/app/450390/The_Lab/.