Vidgets: Modular Mechanical Widgets for Mobile Devices

CHANG XIAO, Columbia University
KARL BAYER, Snap Inc.
CHANGXI ZHENG, Columbia University
SHREE K. NAYAR, Snap Inc.

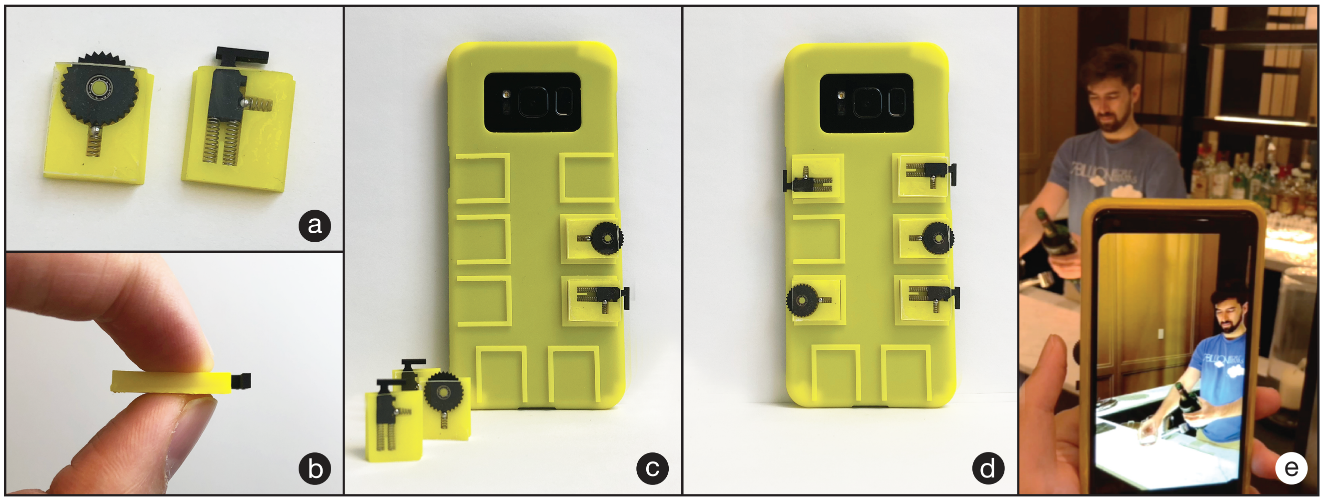


Fig. 1. **Vidgets widgets.** We design a family of physical widgets that are modular, power efficient, (a) compact, and (b) thin. (c) These widgets can be attached to a smartphone's protective case in a modular fashion. (d) The widgets can be used to construct a wide range of user interfaces and adapt to individual's preferences. (e) As just one example, here we use a Vidgets knob to zoom a camera's field of view and a Vidgets 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.

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 widgets' nonlinear mechanical response shifts the device slightly and quickly, and this subtle motion can be detected by the accelerometer commonly equipped 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 so that the resulting accelerometer signals become 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. Vidgets 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. We demonstrate their use in three applications: photo capture with single-handed zoom, control of mobile games, and making a playable mobile music instrument.

Authors' addresses: Chang Xiao, Columbia University, chang@cs.columbia.edu; Karl Bayer, Snap Inc. kbayer@snap.com; Changxi Zheng, Columbia University, cxz@cs.columbia.edu; Shree K. Nayar, Snap Inc. snayar@snap.com.

0730-0301/2019/7-ART100 $15.00
https://doi.org/10.1145/3306346.3322943

CCS Concepts: • **Human-centered computing** → *Haptic devices.*

Additional Key Words and Phrases: mechanical widget, accelerometer

ACM Reference Format:
Chang Xiao, Karl Bayer, Changxi Zheng, and Shree K. Nayar. 2019. Vidgets: Modular Mechanical Widgets for Mobile Devices. *ACM Trans. Graph.* 38, 4, Article 100 (July 2019), 12 pages. https://doi.org/10.1145/3306346.3322943

1 INTRODUCTION

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 [Chu and Wong 2011; Lee and Oulasvirta 2016; Zaman et al. 2010]. 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* [Siek et al. 2005].

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 *Vidgets*, a family of mechanical widgets, specifically a set of push buttons and rotary knobs, that satisfies all these desiderata.

Our design of Vidgets 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. 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 Vidgets is illustrated in Fig. 1, featuring a number of attributes:

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[1]. 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. 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).

[1]See the survey published at Statista (www.statista.com).

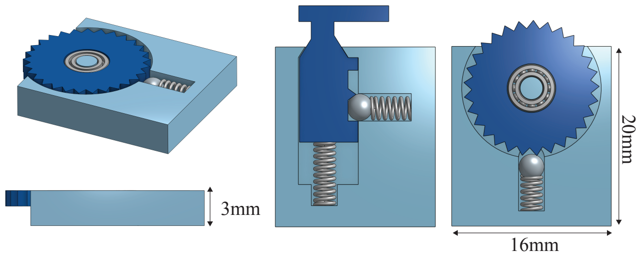


Fig. 2. **Vidgets design.** The basic design of Vidgets is simple. We use a spring to push a steel ball against a tooth (to make a button shown in the middle) or a circle of teeth (to make a knob shown on the right). For the button, we use an additional spring on the bottom to reset. As we will analyze in §2, 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 power consumption. Consisting of merely mechanical structures, Vidgets require no power supply nor wired connection to the phone. They rely only on the device's accelerometer, which is always on at all time [Google Inc. 2018], 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 §4).

1.1 Related Work

Our work is related to an area generally known as around-device interaction [Kratz and Rohs 2009]. The idea is to extend the interaction possibilities beyond the mobile device itself and include the space around it. Most of the proposed solutions requires additional sensors and other hardware attached or connected to the mobile device. For example, HoverFlow [Kratz and Rohs 2009] uses infrared proximity sensors to track hand and finger positions. SideSight [Butler et al. 2008] uses an array of infrared sensors and LED lights for tracking objects in proximity. And ShiftIO [Strasnick et al. 2017] utilizes reconfigurable tactile elements attached on the edge of a mobile device to enable physical control and tactile feedback. MagGetz [Hwang et al. 2013], Bianchi et al. [2013] and Liang et al. [2014] detecting the magnetic field generated by 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 increasingly small form factor of a mobile device. Our work makes no such assumption.

Another line of work aims to enable user interaction on the back of the device by, for example, attaching a keypad or touch pad on the backside [Baudisch and Chu 2009; Hiraoka et al. 2003; Li et al. 2008;

Sugimoto and Hiroki 2006] or placing a camera that looks at the backside [Matsushima et al. 2017; Wigdor et al. 2007]. However, we notice that a majority of mobile products such as smartphones now use protective cases that cover almost entirely the backside of the device. It is unclear how those backside sensors can be used together with the case. In contrast, our Vidgets widgets can be attached to the backside of the case itself, thereby retaining the protection provided by the case.

Instead of using additional sensors, researchers have also sought to enrich user interaction using sensors already existing on a mobile device. One idea is to use microphones. Physical objects can be designed to emit specific, detectable sounds. During interaction, a microphone captures the sounds, and by analyzing the sound pattern or frequency, the mobile device is able to recognize the object [Li et al. 2016] and user interactions [Harrison et al. 2012; Murray-Smith et al. 2008; Savage et al. 2015; Xiao et al. 2014]. However, microphone signals are likely contaminated by ambient noise, which can become a prominent issue in a noisy environment. Laput et al. [2015] 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 have 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 [Laput et al. 2016] customizes the OS kernel of a smartwatch to enable its accelerometer to sample at 4kHz and thereby recognize hand gestures. Unfortunately, kernel customization is not always possible on locked systems such as iOS and on most of the Android devices. We show that the default sampling rate (400Hz) is sufficient for users interacting with Vidgets. In addition, Zhang el al. [2016; 2015] used signals from the accelerometer, gyroscope, and microphone all together to enable additional tapping and sliding inputs. They also rely on a machine-learning based algorithm for the recognition, although it seems 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. [2018] 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 dating back to 1870s [Norman 2013]. While the switch mechanisms (e.g., [Cherry GmbH 2018; Harris 1978]) 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 the user's finger would interact with them, which motivates our design.

Contributions. In summary, we contribute:

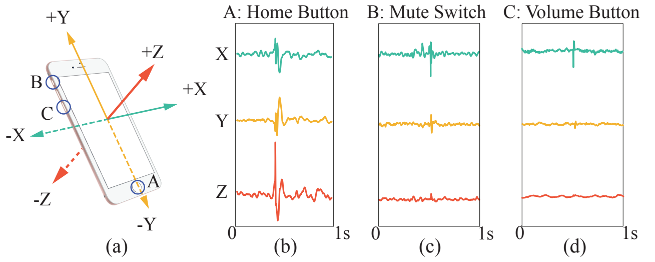


Fig. 3. **Accelerometer signals.** (a) An accelerometer measures the device acceleration in the device's local frame of reference. We use it to record X-, Y-, and Z-channel signals when pressing three buttons on the phone (labeled by A, B, and C, and indicated in the circles). (b,c,d) The recorded accelerometer 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.

- the design of a family of mechanical widgets that can form a rich set of physical user interfaces for mobile devices,
- a physics-based model that guides and justifies the choice of widget design,
- a computational algorithm that analyzes accelerometer signals and identifies which user interaction events and on which widgets they are triggered on, and
- three applications that demonstrate the use of Vidgets widgets in practical scenarios.

2 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.

2.1 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. 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.

2.1.1 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. 4). 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 [Cherry GmbH 2018; Kim et al. 2018; Rempel et al. 1994]. 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.

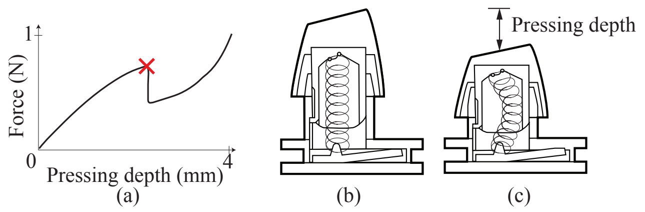


Fig. 4. **Nonlinear mechanical switch.** Many mechanical switch has a nonlinear force response. Here we illustrate the force nonlinearity using the IBM's buckling spring switch (b), a classic switch design historically used on IBM's Model M keyboard. Its force response curve is qualitatively plotted 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.

2.1.2 *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. 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. 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. 5-a). Suppose that the user's thumb is pressing a button with a force F_p while other fingers and the palm provide support, and the support forces are denoted by $\hat{F}_1, ..., \hat{F}_5$. 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. 4). 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 §2.2) it takes about 10ms for F_b to drop, while our finger's response time to tactile stimuli is on the order of 100ms [Ng and Chan 2012]. 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. 6).

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

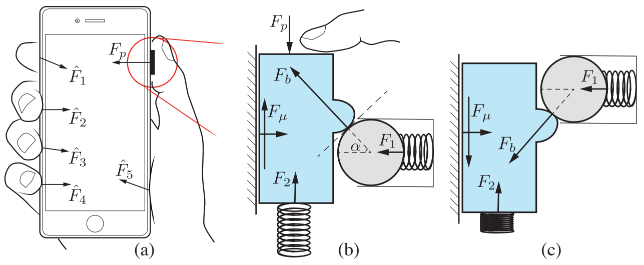


Fig. 5. **Force analysis.** (a) When a phone is held by a hand and a button (in the red circle) is pressed, a set of forces are applied on the phone to keep the phone almost still. The pressing force F_p from the thumb is exerted to the button's switch mechanism shown in (b) (after a 90°rotation), where 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.

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 are interested in happens within less than 50ms (Fig. 6-b,c). Therefore, a simple high-pass filter can separate it from the signals due to locomotion. This detail will be discussed later in §5.3.

2.2 Physics-Based Force Model

We now introduce a physics-based model that describes the resistance force profile of our Vidgets 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. 5-b, which also depicts the Vidgets button design in Fig. 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 Vidgets knob design showing in Fig. 2-c (after rotated by 90°). 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

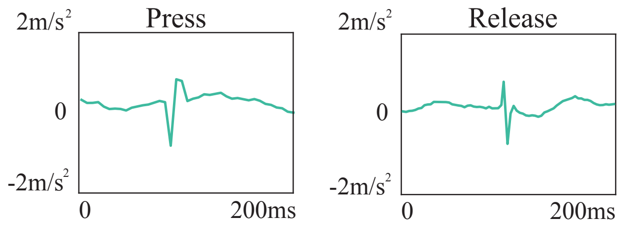


Fig. 6. **A close look** of the X-axis signal when pressing and releasing the volume button on a smartphone. On the left is the signal generated by pressing the button down (downstroke), and on the right is the signal from releasing the button (upstroke). Notice the short durations of these signals, 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.

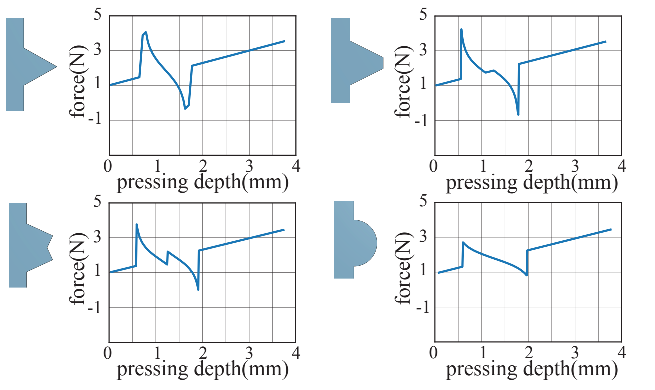


Fig. 7. **The force profile zoo.** Here we show four different designs of the tooth shape used in Vidgets buttons (Fig. 2-b) and their estimated resistance force profiles. As we vary the tooth shape, the force profile changes considerably.

must be zero, meaning

$$F_p + F_2 + F_b \sin\alpha + \tau \cdot \mu \cdot F_b \cos\alpha = 0, \tag{1}$$

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 α is the contact angle (see Fig. 5 for the notation). In addition, μ denotes the Coulomb friction coefficient between the slider and the container, and τ 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

$$F_1 + F_b \cos\alpha = 0, \tag{2}$$

where F_1 is the force from side spring. Combining (1) and (2) yields an expression of the widget's resistance force F_r, namely,

$$F_r = -F_p = F_2 - F_1 \tan\alpha - \tau \cdot \mu \cdot F_1. \tag{3}$$

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 (3) because F_1, F_2, and α 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 α 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:

$$\begin{gathered}[x_c - x(t_0)]^2 + [y_c - y(t_0) + d]^2 = R^2, \\ x'(t_0)\,[x_c - x(t_0)] + y'(t_0)\,[y_c - y(t_0) + d] = 0.\end{gathered} \tag{4}$$

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 (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 (4)

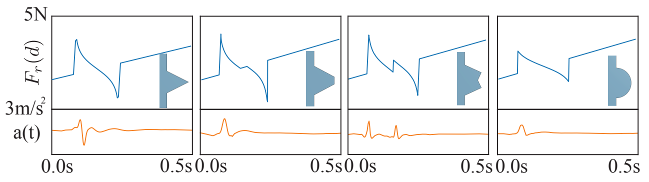


Fig. 8. **Force profile vs. accelerometer signals.** Here we show the force profiles of four tooth shapes (the same as Fig. 7) and their accelerometer signals when these tooth shapes are used in a Vidgets button. Notice the correlation and temporal alignment between the estimated actuation points of 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.

depend on the pressing depth d, both α and F_1 are functions of d, and so is F_r.

2.3 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. 2). Figure 7 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 §2.1.2).

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. 7 leads to different acceleration signals shown in Fig. 8. The correlation between the estimated actuation points and the recorded signals (shown in Fig. 8) 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. 9-a, where the button has no teeth at all. In that case, the actuation point vanishes and the 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. 8 and Fig. 9).

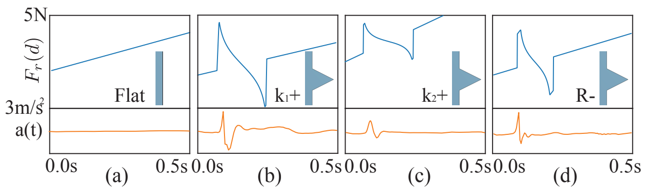


Fig. 9. **Force profile dependence on design parameters**. The top row shows the estimated force profiles when various designs are used in the button mechanism. The bottom row shows the corresponding accelerometer signals when those designs are used. (a) With no teeth, the force is linear, and no signal is detected by the accelerometer. In (b), (c), and (d), we use the same tooth shape as shown in the first plot of Fig. 7, 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−).

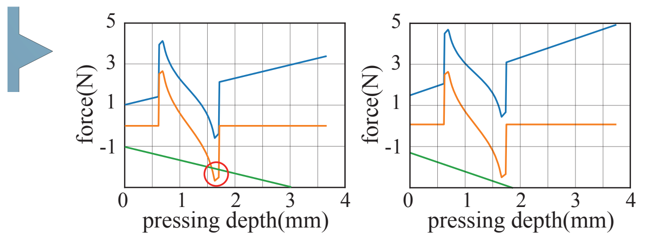


Fig. 10. **Button reset.** Given a tooth shape shown on the left, blue curves are the estimated force profile $F_r(d)$. The orange curves indicate the vertical component $F_b \sin\alpha$ of the force exerted by the ball. and the green curves indicate the negated force from the bottom spring (i.e., $-F_2$). (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.

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 signals (Fig. 9-c).

Equally important is guaranteeing that the button resets itself when it is released. This is illustrated in Fig. 10, wherein the orange curve indicates the vertical force (i.e., $F_b \sin\alpha$) exerted by the steel ball (shown in Fig. 5). 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. 5-c). The green curve in Fig. 10 shows the bottom spring's force. If it intersects with the orange curve (Fig. 10-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. 7). 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 §3.2.

3 SIGNAL PROCESSING AT RUNTIME

Vidgets' 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. 11. The first step identifies time regions in which Vidgets 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.

3.1 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 Vidgets usage events may happen. Each event region has a length of 50ms. We choose this time length according to [Killourhy and Maxion 2009], 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 [Felzenszwalb et al. 2010], 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 Δ 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.

3.2 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.

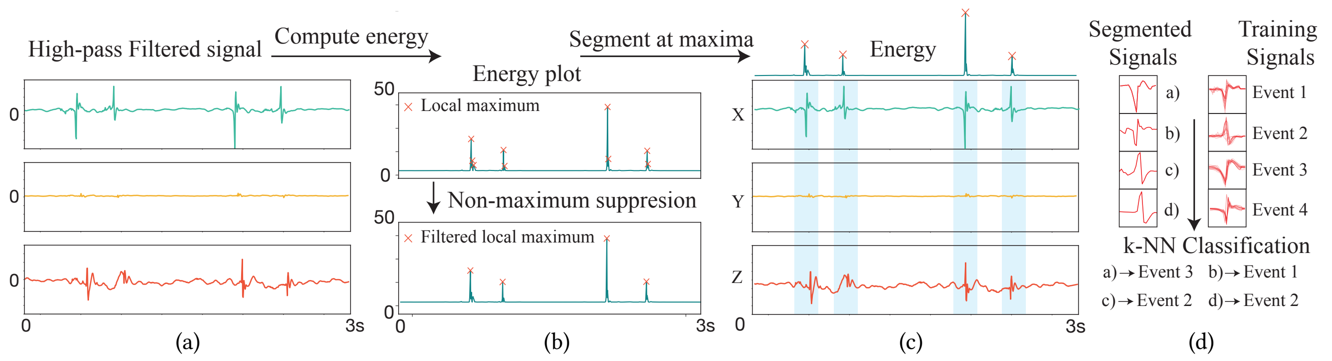


Fig. 11. **Overview of runtime detection.** Our detection algorithm takes as input the raw accelerometer signals. After applying a high-pass filter (a), the signals are processed through non-maxima suppression (b) and segmented at the local maximum (c). The segmented signal is then classified using k-NN classification to determine what Vidgets interaction event, if any, occurs (d).

As discussed in §2.1, 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 §2 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 20×3 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 [Nasrabadi 2007]. This algorithm computes the classic Dynamic Time Warping (DTW) similarity [Berndt and Clifford 1994] 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 similarity 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 §5.3. 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. 7) are particularly helpful. Thereby, the various designs of Vidgets widgets ensure that they are always recognizable regardless of their intended layout on the phone case.

4 APPLICATIONS

Vidgets are building blocks of a diverse set of user interfaces and thereby offer some unique advantages. To bring Vidgets 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 Vidgets 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 Vidgets.

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. 12, 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 Vidgets' modularity, one can reposition the button and knob on the phone case to fit their personal preference, as shown in the video.

Discussion. Although Vidgets 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

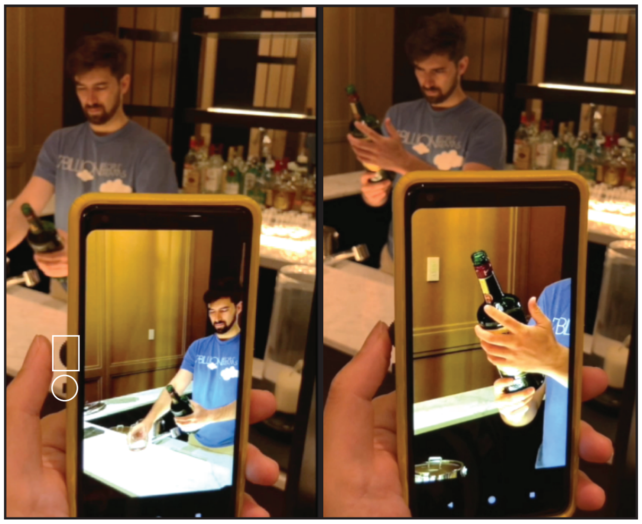


Fig. 12. **Camera zoom.** When taking a photo or video, the user can zoom the field of view in and out using a Vidgets knob (in white box) and trigger the shot using a Vidgets button (in white circle), all with single hand. In contrast, most of the current smartphone cameras require two hands to zoom 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).

scale of the shift and the help of optical image stabilization system already existed in most of 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*[2], 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.

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. 13 and the video).

Virtual instrument. Wang [2009] 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,

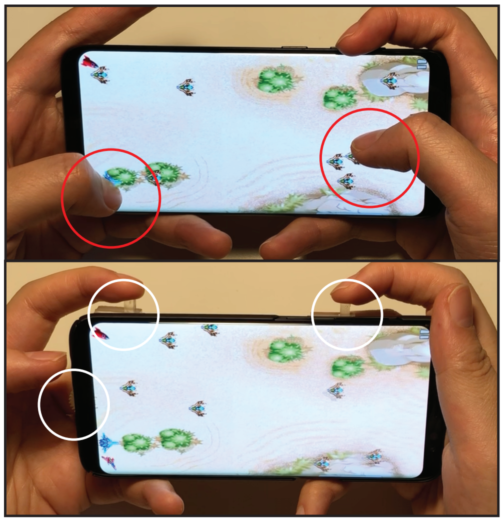


Fig. 13. **Mobile gaming.** (top) When controlling a game on a mobile device, the player's fingers often occlude the display (e.g., in red circles) and cause disruption. (down) Vidgets interface (in white circles) enables the player to control game more efficiently without occluding the screen.

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 [Brewster et al. 2007; Hoggan et al. 2008; Zaman et al. 2010], we extend Wang's concept and introduce the "smartphone's saxophone" (see Fig. 14 and the supplementary video), 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.

5 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

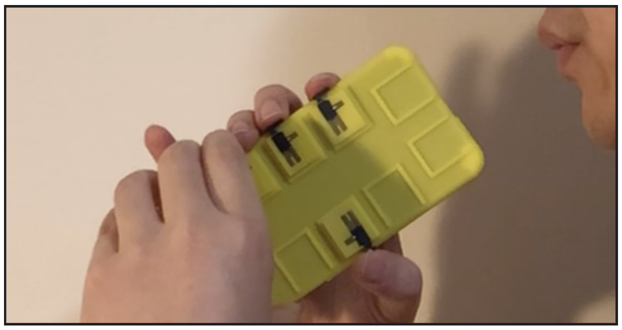
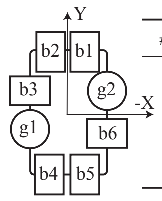


Fig. 14. **Saxophone app.** We demonstrate a 4-key "smartphone's saxophone". Each Vidgets button serves as a saxophone key. By detecting the downstroke and upstroke of each Vidgets button, this smartphone application can recognize which keys (and key combinations) are pressed and play the corresponding music notes.

[2]For example, see the products from this Amazon link.



# of widgets	Exclusion	k=1	k=3	k=5
8	None	93.2%±4.1%	90.6%±3.2%	90.0%±3.0%
7	b1	94.1%±3.8%	93.0%±3.5%	92.2%±3.5%
6	b1, g2	97.0%±1.7%	95.6%±2.3%	95.6%±2.0%
5	b1, g2, b5	97.7%±1.7%	96.0%±2.1%	96.1%±1.9%
4	b1, g2, b5, b3	98.9%±0.9%	98.9%±0.5%	98.2%±0.8%

Fig. 15. (**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.

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 [Google Inc. 2018].

5.1 Accuracy

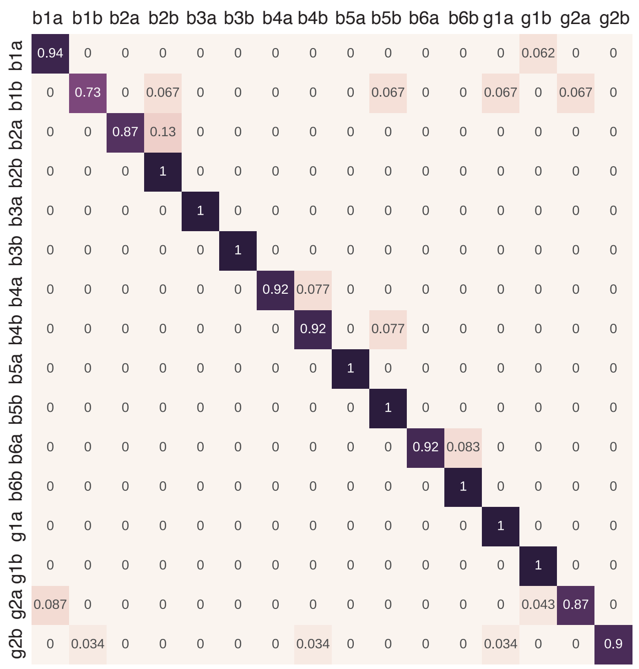
Participants. To understand the accuracy of using Vidgets widgets, we recruited 13 participants, including six females. Their mean age is 24.3, and two are left-handed.

Procedure. We use an 8-widget layout on the smartphone case, as shown in Fig. 15-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 8×2=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, 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. 15). 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. 16.

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 §4, 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 [Laput et al. 2016; Savage et al. 2015], although the underlying operation principles are different.



	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

Fig. 16. **Confusion matrix** for detecting user interaction events on 8 widgets. The layout of the widgets is shown in Fig. 15-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. 15-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.

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 accuracy 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

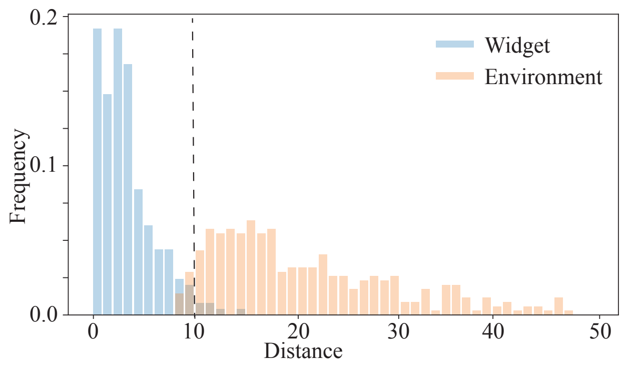


Fig. 17. **Eliminating false events.** We collect signals generated by other events from the daily life (e.g., walking, taking the subway, and gaming), and plot (in red) the distribution of k-NN distances between those events and the calibration vectors. In comparison, we also plot (in blue) the distribution 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.

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.

5.2 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 §3.1). 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 [Card et al. 1991; Miller 1968].

5.3 Robustness

Body locomotion and environmental vibration such as car motion also generate accelerometer signals. As discussed in §3.2, 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. 19, 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 (recall §3.1), 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. 17, together

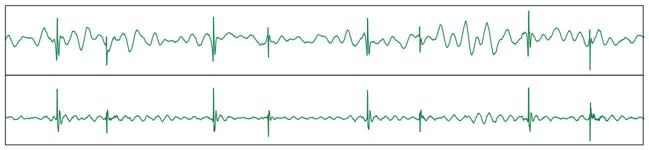


Fig. 18. (**top**) We recorded accelerometer signals when using the widgets on a running subway train. (**bottom**) The noise introduced by the train motion can be largely eliminated through a high-pass filter, since the widget interaction events occur at a much shorter time scale.

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. 15-left. 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 §3.1. Figure 18 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. 15-left, we observed a slight drop in accuracy from 93.2% to 89.1%.

Interaction speed. Vidgets 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 Vidgets 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 Vidgets button 4 times per second. This corresponds to 8 events per second (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 Vidgets 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.

5.4 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

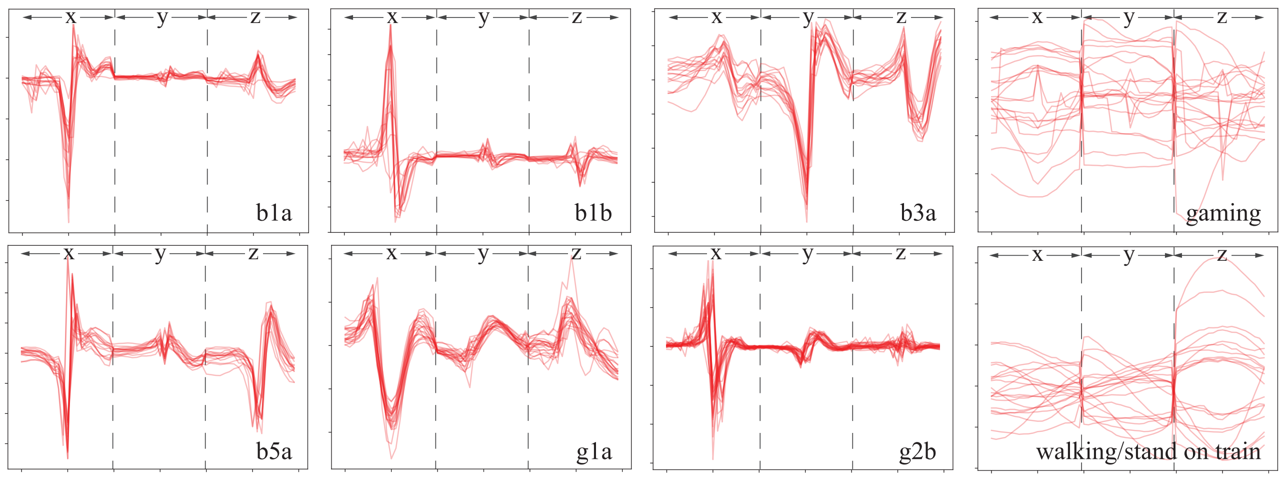


Fig. 19. **Visual differences between widget signals and others**. In the first three columns, each plot visualizes 20 accelerometer signals generated by a widget event. The event is indicated by the ID in the bottom-right corner of the plot (in the same way as those used in Fig. 15). Each curve is a 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).

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 widgets. 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. 7.

6 CONCLUSION 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 [Laput et al. 2016], 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 Vidgets 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.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback, as well as Brian Smith and Lydia Chilton for their help on paper revision, Zixiaofan Yang for video recording and Henrique Maia for video narration. This work was supported by Snap Inc. It was done while Chang Xiao was an intern at Snap.

REFERENCES

Patrick Baudisch and Gerry Chu. 2009. Back-of-device Interaction Allows Creating Very Small Touch Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1923–1932.

Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series.. In *KDD workshop*, Vol. 10. Seattle, WA, 359–370.

Andrea Bianchi and Ian Oakley. 2013. Designing tangible magnetic appcessories. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. ACM, 255–258.

Stephen Brewster, Stephen Brewster, Faraz Chohan, and Lorna Brown. 2007. Tactile feedback for mobile interactions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 159–162.

Alex Butler, Shahram Izadi, and Steve Hodges. 2008. SideSight: multi-"touch" interaction around small devices. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '08)* (proceedings of the acm symposium on user interface software and technology (uist '08) ed.). Association for Computing Machinery, Inc. https://www.microsoft.com/en-us/research/publication/sidesight-multi-touch-interaction-around-small-devices/

Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. 1991. The Information Visualizer, an Information Workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*. ACM, New York, NY, USA, 181–186.

Cherry GmbH. 2018. Cherry MX Switches. https://www.cherrymx.de/en.

Kimberly Chu and Chui Yin Wong. 2011. Mobile input devices for gaming experience. In *User Science and Engineering (i-USEr), 2011 International Conference on*. IEEE, 83–88.

Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. 2010. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence* 32, 9 (2010), 1627–1645.

Google Inc. 2018. Sensor Overview of Android system. https://developer.android.com/guide/topics/sensors/sensors_overview.

Richard Hunter Harris. 1978. Buckling spring torsional snap actuator. US Patent 4,118,611.

Chris Harrison, Robert Xiao, and Scott Hudson. 2012. Acoustic Barcodes: Passive, Durable and Inexpensive Notched Identification Tags. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 563–568.

Shigeo Hiraoka, Isshin Miyamoto, and Kiyoshi Tomimatsu. 2003. Behind touch, a text input method for mobile phones by the back and tactile sense interface. *Information Processing Society of Japan, Interaction 2003* (2003), 131–138.

Eve Hoggan, Stephen A Brewster, and Jody Johnston. 2008. Investigating the effectiveness of tactile feedback for mobile touchscreens. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1573–1582.

Sungjae Hwang, Myungwook Ahn, and Kwang-yun Wohn. 2013. 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, 411–416.

Kevin S Killourhy and Roy A Maxion. 2009. Comparing anomaly-detection algorithms for keystroke dynamics. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP international conference on*. IEEE, 125–134.

Sunjun Kim, Byungjoo Lee, and Antti Oulasvirta. 2018. Impact Activation Improves Rapid Button Pressing. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 571, 8 pages.

Sven Kratz and Michael Rohs. 2009. 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 (MobileHCI '09)*. ACM, New York, NY, USA, Article 4, 8 pages.

Gierad Laput, Eric Brockmeyer, Scott E Hudson, and Chris Harrison. 2015. Acoustruments: Passive, acoustically-driven, interactive controls for handheld devices. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. ACM, 2161–2170.

Gierad Laput, Robert Xiao, and Chris Harrison. 2016. ViBand: High-Fidelity Bio-Acoustic Sensing Using Commodity Smartwatch Accelerometers. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 321–333.

Byungjoo Lee and Antti Oulasvirta. 2016. Modelling error rates in temporal pointing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 1857–1868.

Dingzeyu Li, David I.W. Levin, Wojciech Matusik, and Changxi Zheng. 2016. Acoustic Voxels: Computational Optimization of Modular Acoustic Filters. *ACM Trans. Graph. (SIGGRAPH 2016)* 35, 4 (2016). https://doi.org/10.1145/2897824.2925960

Kevin A Li, Patrick Baudisch, and Ken Hinckley. 2008. Blindsight: eyes-free access to mobile phones. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 1389–1398.

Rong-Hao Liang, Han-Chih Kuo, Liwei Chan, De-Nian Yang, and Bing-Yu Chen. 2014. 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, 365–372.

Nobutaka Matsushima, Wataru Yamada, and Hiroyuki Manabe. 2017. 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, 51–52.

Robert B Miller. 1968. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. ACM, 267–277.

Roderick Murray-Smith, John Williamson, Stephen Hughes, and Torben Quaade. 2008. Stane: Synthesized Surfaces for Tactile Input. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1299–1302.

Nasser M Nasrabadi. 2007. Pattern recognition and machine learning. *Journal of electronic imaging* 16, 4 (2007), 049901.

Annie WY Ng and Alan HS Chan. 2012. Finger response times to visual, auditory and tactile modality stimuli. In *Proceedings of the international multiconference of engineers and computer scientists*, Vol. 2. 1449–1454.

Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Constellation.

Michal Piovarči, David IW Levin, Danny M Kaufman, and Piotr Didyk. 2018. Perception-aware modeling and fabrication of digital drawing tools. *ACM Transactions on Graphics (SIGGRAPH 2018)* 37, 4 (2018), 123.

David Rempel, Jack Dennerlein, CD Mote Jr, and Thomas Armstrong. 1994. A method of measuring fingertip loading during keyboard use. *Journal of Biomechanics* 27, 8 (1994), 1101–1104.

Valkyrie Savage, Andrew Head, Björn Hartmann, Dan B. Goldman, Gautham Mysore, and Wilmot Li. 2015. Lamello: Passive Acoustic Sensing for Tangible Input Components. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. ACM, New York, NY, USA, 1277–1280.

Katie A Siek, Yvonne Rogers, and Kay H Connelly. 2005. Fat finger worries: how older and younger users physically interact with PDAs. In *IFIP Conference on Human-Computer Interaction*. Springer, 267–280.

Evan Strasnick, Jackie Yang, Kesler Tanner, Alex Olwal, and Sean Follmer. 2017. shiftIO: Reconfigurable Tactile Elements for Dynamic Affordances and Mobile Interaction. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 5075–5086. https://doi.org/10.1145/3025453.3025988

Masanori Sugimoto and Keiichi Hiroki. 2006. 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, 137–140.

Ge Wang. 2009. Designing Smule's Ocarina: The iPhone's Magic Flute. In *NIME*. 303–307.

Daniel Wigdor, Clifton Forlines, Patrick Baudisch, John Barnwell, and Chia Shen. 2007. Lucid touch: a see-through mobile device. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. ACM, 269–278.

Robert Xiao, Greg Lew, James Marsanico, Divya Hariharan, Scott Hudson, and Chris Harrison. 2014. 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, 67–76.

Loutfouz Zaman, Daniel Natapov, and Robert J Teather. 2010. Touchscreens vs. traditional controllers in handheld gaming. In *Proceedings of the international academic conference on the future of game design and technology*. ACM, 183–190.

Cheng Zhang, Anhong Guo, Dingtian Zhang, Yang Li, Caleb Southern, Rosa I. Arriaga, and Gregory D. Abowd. 2016. Beyond the Touchscreen: An Exploration of Extending Interactions on Commodity Smartphones. *ACM Trans. Interact. Intell. Syst.* 6, 2, Article 16 (Aug. 2016), 23 pages.

Cheng Zhang, Anhong Guo, Dingtian Zhang, Caleb Southern, Rosa Arriaga, and Gregory Abowd. 2015. BeyondTouch: Extending the Input Language with Built-in Sensors on Commodity Smartphones. In *Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI '15)*. ACM, New York, NY, USA, 67–77.