Controlling Widgets with One Power-up Button

Daniel Spelmezan, Caroline Appert, Olivier Chapuis, Emmanuel Pietriga

▶ **To cite this version:**

Daniel Spelmezan, Caroline Appert, Olivier Chapuis, Emmanuel Pietriga. Controlling Widgets with One Power-up Button. Proceedings of the 26th ACM Symposium on User Interface Software and Technology, Oct 2013, Saint Andrews, United Kingdom. pp.71-74, 10.1145/2501988.2502025 . hal-00864209v2

HAL Id: hal-00864209
https://hal.science/hal-00864209v2

Submitted on 14 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Controlling Widgets with One Power-up Button

Daniel Spelmezan[1,2] **Caroline Appert**[2,1] **Olivier Chapuis**[2,1] **Emmanuel Pietriga**[1,3]

[1] INRIA
91405 Orsay, France

[2] Univ Paris-Sud & CNRS (LRI)
91405 Orsay, France

[3] INRIA Chile (CIRIC)
7561211 Santiago, Chile

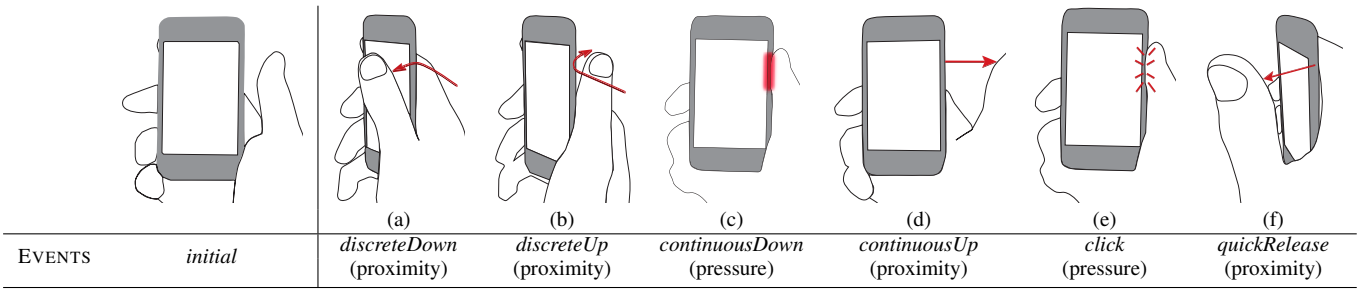


Figure 1. Gestures performed on and around the *Power-up Button*.

ABSTRACT

The Power-up Button is a physical button that combines pressure and proximity sensing to enable gestural interaction with one thumb. Combined with a gesture recognizer that takes the hand's anatomy into account, the Power-up Button can recognize six different mid-air gestures performed on the side of a mobile device. This gives it, for instance, enough expressive power to provide full one-handed control of interface widgets displayed on screen. This technology can complement touch input, and can be particularly useful when interacting eyes-free. It also opens up a larger design space for widget organization on screen: the button enables a more compact layout of interface components than what touch input alone would allow. This can be useful when, e.g., filling the numerous fields of a long Web form, or for very small devices.

Author Keywords

Mobile device; Proximity input; Pressure input

ACM Classification Keywords

H.5.2 [Information Interfaces and Presentation]: User Interfaces - Input devices and strategies (e.g., mouse, touchscreen)

INTRODUCTION

Touch input enables users to interact with display surfaces in a straightforward manner, by directly manipulating content and controlling widgets with their fingers. This simple interaction paradigm has favored the wide adoption of touch input on mobile devices. However, it has some limitations. First, accommodating control widgets reduces the area dedicated to the display of actual content, a resource that is already scarce on small devices. Second, users face occlusion and precision issues inherent to direct finger input [9]. This makes touch input inadequate in some contexts. For instance, users cannot operate their device eyes-free even for simple operations. Browsing songs in a music player, discarding an incoming phone call, or using the device as a remote controller to perform basic actions on a public display, requires taking the device out of one's pocket and looking at it.

A number of alternatives to touch input have been investigated, that expand interaction to the back of the device [1, 5, 9], to its sides [6, 10], and to the space above and around it [2, 4, 7, 8]. Many of these alternatives, however, are not very practical, no matter how interesting they might be. Instrumenting the finger with additional hardware [4] is inconvenient; around-the-device interaction [2, 7] does not allow users to both hold and operate the device with a single hand. Introducing physical buttons on the device's sides does not cause such issues, but their limited expressive power makes them better at complementing touch input rather than at providing full interface control [6]. One option would be to increase the number of such buttons, but this approach is impractical for small devices [5]. Another option is to extend the expressive power of those buttons.

Our *Power-up Button* enables users to perform numerous actions using a single controller located on the side of the device adjacent to the thumb. The controller does not occupy more space than a regular button would, yet provides a much higher level of expressive power compared to scroll wheels and joysticks that were used in some commercial devices. The *Power-up Button* consists of one proximity sensor superimposed on one continuous pressure sensor. Combined with the gesture recognizer we developed, it allows users to trigger a set of four discrete events and two continuous events using simple gestures performed on and around the button while operating the device with a single-hand (Figure 1). A single *Power-up Button* enables users to control, e.g., a music player eyes-free: pause and resume playback, adjust volume settings, and navigate both within and across songs.

D. Spelmezan, C. Appert, O. Chapuis, and E. Pietriga. Controlling Widgets with One Power-up Button. In UIST '13: Proceedings of the 26th ACM Symposium on User Interface Software and Technology, 71-74, ACM, 2013.

© ACM, 2013. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version will be published in *UIST '13*, October 8–11, 2013, St. Andrews, United Kingdom. http://doi.acm.org/10.1145/2501988.2502025

We first give descriptions of our hardware prototype and event recognizer, that enables the *Power-up Button* to discriminate 2D gestures even though the underlying technology is based on one-dimensional sensing capabilities. We then explain how events were consistently mapped to actions that allow users to control all widgets of an application, providing an alternative to touch input when the latter is not practical; and conclude with directions for future work.

HARDWARE PROTOTYPE

Figure 2 shows the hardware prototype. The sensing hardware comprises four components: a force-sensitive resistor (Interlink Electronics FSR400 Short), an infrared proximity sensor (Vishay VCNL4000) that is mounted on a Sparkfun breakout board, an Arduino Pro Mini, and a custom-designed printed circuit board. Our sensor layering is similar to that of [3, 8], with one significant difference: our hardware detects continuous pressure input instead of switch-on/off events, and continuous proximity input instead of hovering.

The casing for the pressure sensor was produced by a 3D printer (Figure 2-a). This casing has a gap of 2 mm where the sensor and a thin piece of rubber are inserted. The rubber uniformly translates the applied pressure to the sensor's active area, which has a diameter of 5 mm.

The proximity sensor has an integrated ambient light sensor and signal processing circuit for suppressing ambient light by signal modulation. Its nominal sensing range is 200 mm. This sensor is placed inside a casing that was produced by a laser cutter. At the top of the casing is a small window (5×5 mm), which is centered above the proximity sensor at a distance of 2 mm (Figure 2-b). The window constrains the sensor's cone-shaped detection zone to an angle of $\sim \pm 20°$, and decreases the detection range to $\sim$50 mm for the user's thumb.

The printed circuit board (Figure 2-c) is powered by an iPod touch 4G (iOS 5.0). This board has an op-amp based circuit that linearizes the output of the pressure sensor. The Arduino samples sensor data at 25 Hz and sends the measurements to the iPod over the serial port. The iPod applies a low-pass filter and normalizes the output from both sensors.

The output of the proximity sensor depends on the properties of the sensed object, including reflectivity, size, material, color, and inclination inside the detection zone. To calibrate the sensor for our setup we measured the actual distance to one of the author's thumb at intervals of 5 mm, and linearly interpolated between the sensor readings.

The dimensions of the prototype button is $20 \times 16 \times 11$ mm ($L \times W \times H$), which matches the size of the large breakout board where the proximity sensor is mounted. It could actually be made much smaller in more advanced prototypes, as the sensing unit is tiny ($3.95 \times 3.95 \times 0.75$ mm).

GESTURES AND INTERACTION EVENTS

To demonstrate the potential of the *Power-up Button*, we designed a set of gestures that can be discriminated using the technology described above when the button is operated using the thumb of the hand that holds the device (Figure 2). We considered both motor and physiological aspects, ending up with a set of gestures that are easy to perform and that will not cause too much fatigue even when performed in sequence. To control the *Power-up Button*, users move their thumb within the device's plane towards or away from the button (Figure 1-(c,d,e)), and out of the device's plane towards either the front or the rear of the device (Figure 1-(a,b,f)). We implemented a recognizer to discriminate between these two-dimensional gestures with one-dimensional sensing capabilities only.

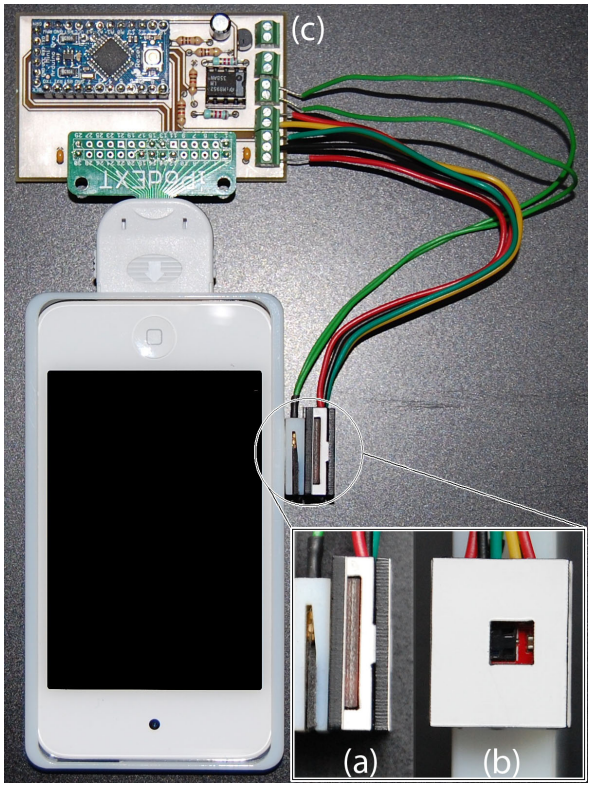


Figure 2. Our prototype, equipped with a button for sensing pressure and proximity. The sensors are placed inside custom-built plastic casings. (a) Side view of the pressure and proximity sensors. (b) Front view of the window above the proximity sensor. (c) Circuit board.

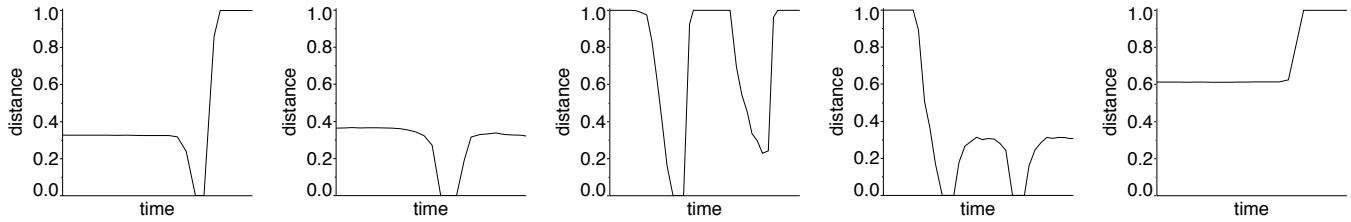
Gesture Recognizer

Our recognizer is implemented as a finite state machine in which final states are one of the following interaction events: *discreteDown*, *discreteUp*, *click*, *quickRelease*, *continuousDown* and *continuousUp*. Transitions are triggered by input variations on the pressure and proximity sensors. For the two continuous events, a first-order control is initiated as soon as the corresponding state is reached, letting users adjust the rate by varying the pressure intensity or the distance to the button.

Events and Pressure

To recognize *click* and *continuousDown* events, the *Power-up Button* analyzes the profile of the normalized pressure intensity that the thumb applies to the device. When the user starts pressing the force sensor, the state machine waits for a short amount of time (500 ms)[1] to decide if the pressure is released or sustained. Releasing the pressure before the timeout occurs triggers a *click* event if the initial force exceeded 1N. If

[1]This value minimizes accidental activation of rate-based control.



(a) *discreteDown* (b) *discreteUp* (c) *discreteDown*$^+$ (d) *discreteUp*$^+$ (e) *quickRelease*

Figure 3. Proximity profiles for discrete events (time scale = 1 sec). *discreteDown*$^+$ **and** *discreteUp*$^+$ **show two consecutive gestures (time scale = 1.5 sec).**

the pressure is sustained, the state machine triggers *continuousDown*, enabling pressure-dependent *continuous* control.

Events and Proximity

The *Power-up Button* analyzes the profile of the normalized distance between the device and the thumb. The distance is $d = 1$ if the thumb is too far away from the sensor's detection zone (`OutOfRange`), $d = 0$ if the sensor is occluded (`Touch`), and $d \in]0..1[$ if the thumb is close to the button (`Prox`). In addition, our algorithm examines how the distance changes over time ($\Delta d = d_t - d_{t-1}$, uniform sampling frequency) to detect changes in direction ($\Delta d > 0$ and $\Delta d < 0$) and gesture speed.

This low-level input allows our recognizer to identify the *discreteDown*, *discreteUp*, *quickRelease* and *continuousUp* gestures. The main challenge lies in discriminating orthogonal gestures *discreteDown* and *discreteUp*, as the proximity sensor only captures the orthogonal distance between the thumb and the edge of the device. We call these gestures orthogonal because the thumb is leaving the device's plane either towards the front of the device (*discreteDown*) or towards the rear of the device (*discreteUp*). Figure 3-(a,b) shows typical sensor readings for these gestures. Both gestures must begin in the proximity range (`Prox`), touch the button (`Touch`) and leave the device's plane in one direction or the other. This final part of the movement can be discriminated because of the anatomy of the hand, that makes moving the thumb away from the user (adduction) harder than moving it towards her (abduction). This results in sensing `OutOfRange` for *discreteDown* gestures and `Prox` for *discreteUp* gestures.

The recognizer for *discreteDown* and *discreteUp* accepts several profiles for a single gesture, making the repetition of such gestures in clockwise and counter-clockwise directions easier to perform. The initial position for both gestures becomes optional, allowing users to start their gesture either in proximity range `Prox` (Figure 3-(a,b)) or `OutOfRange` (Figure 3-(c,d). For *discreteDown*, users can either touch the button or pass close enough to it in mid-air (Figure 3-(c)). Below are typical sequences of low-level events that trigger discrete navigation events ($\nearrow$) in our state machine:

Two *discreteDown* performed in a row (Figure 3-(c)):
$\texttt{OutOfRange} \rightarrow \texttt{Touch} \rightarrow \texttt{OutOfRange} \nearrow$
$\rightarrow \texttt{Prox}_{(\Delta d \neq 0)} \rightarrow \texttt{OutOfRange} \nearrow$

Two *discreteUp* performed in a row (Figure 3-(d)):
$\texttt{OutOfRange} \rightarrow \texttt{Touch} \rightarrow \texttt{Prox}_{(\Delta d > 0)} \rightarrow \texttt{Prox}_{(\Delta d \leq 0)} \nearrow$
$\rightarrow \texttt{Touch} \rightarrow \texttt{Prox}_{(\Delta d > 0)} \rightarrow \texttt{Prox}_{(\Delta d \leq 0)} \nearrow$

Figure 3-e shows that the *quickRelease* gesture is easily recognized based on a speed threshold at which the thumb must leave the $\texttt{Prox}_{(\Delta d=0)}$ without touching the button ($\Delta d > s$ with $s = 0.5 \cdot (1 - d_{t_{initial}})$). Finally, the *continuousUp* gesture is recognized when the thumb remains in the detection zone for more than 500 ms.

Controlling Widgets

To evaluate the expressive power of our button, we demonstrate how the set of interaction events it generates can be used to control any widget of an existing graphical application. Table 1 lists the set of widgets commonly available on small devices[2] and how we can use *Power-up Button* events to fully control them. As we do not rely on touch input to detect which widget owns the focus, interaction events must also be dedicated to navigation in the widget hierarchy.

- *click* selects a widget. For a group of widgets, this event drills down to widgets that are located one level deeper in the widget hierarchy (navigation from parent to child).
- *discreteUp* and *discreteDown* increase and decrease the current value of a widget by one unit. For a group of widgets, these events move the input focus to the previous or next widget that are located at the same level in the widget hierarchy (navigation between siblings).
- *continuousUp* and *continuousDown* increase and decrease the current value of a widget using rate-based control.
- *quickRelease* moves up to widgets located one level up in the widget hierarchy (navigation from child to parent).

Figure 4 illustrates a scenario in which a user sets the value of a radio button that is not visible in the current viewport, which is scrollable. At the beginning, the viewport is outlined in blue to indicate that the input focus is set to the scrollbar. A *click* turns the outline of the viewport red (scrolling enabled) so that the user can bring the set of radio buttons in the viewport through the use of *continuousUp* and *continuousDown* events. She then *clicks* to move the input focus from the scrollbar to the first widget that is visible in the viewport (i.e., the group of radio buttons) and *clicks* again to access the level where she can control the set of individual buttons. She then navigates to the second radio button with a *discreteDown* event and *clicks* to select it. Finally, *quickRelease* commits the button selection and brings the input focus back to the group of radio buttons.

[2]Our implementation uses the iOS widget set. A *stepper* is an iOS widget that contains *plus* and *minus* buttons to set a discrete value.

EVENTS	BUTTON	STEPPER	CHECK BOXES AND RADIO BUTTONS	COMBO BOX	SLIDER	SCROLLBAR
click	invoke		select option	show list box	begin tracking	begin scrolling
quickRelease				dismiss list box	end tracking	end scrolling
discreteUp		increase value	highlight prev. option	highlight prev. list item		
discreteDown		decrease value	highlight next option	highlight next list item		
continuousUp					increase value*	scroll upward*
continuousDown					decrease value*	scroll downward*

* *(rate-based, speed depends on pressure level / proximity)*

Table 1. Interaction events and their mappings to navigation actions for widget control.

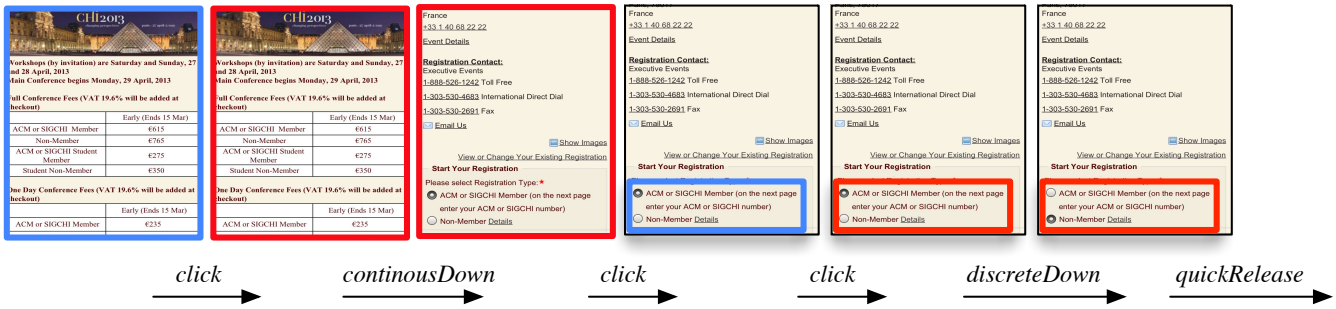


Figure 4. Selecting radio button Non-Member.

The exercise that consists of controlling all widgets is a demonstration of what the *Power-up Button* could do, pushed to the extreme. The more relevant use cases we envision involve a smaller set of controls like, e.g., discarding phone calls or controlling a music player. The latter basically consists of a slider for setting the volume, and a few buttons for navigating across and within songs. Considering only gestures relevant to the application or widget in focus can also help increase the recognizer's accuracy.

CONCLUSION AND FUTURE WORK

The *Power-up Button* is a physical button that combines pressure and proximity sensing to complement touch input on a mobile device for, e.g., controlling an application eyes-free. We present a set of simple gestures and a recognizer that let users manipulate an application on their mobile phone by moving the thumb of the hand that holds the device. We demonstrate that such a button alone can provide users with an expressive power high enough to control all usual widgets.

However, the *Power-up Button* has a few limitations. First, this technology captures mid-air gestures with one-dimensional sensing capabilities only. While it is an important advantage to be pluggable on any small device, it also means that the recognizer is potentially more permissive. Second, as with any gesture-based interaction, users must have clues to discover and learn possible gestures. This requires adding feedback to guide and assist novice users. However, informal testing with colleagues revealed that a short initial training period makes users both able to control the *Power-up Button* without triggering accidental events and to get a clear understanding about how to control a collection of widgets.

As future work, we plan to conduct controlled empirical studies to formally evaluate the two aspects mentioned above. We also plan to explore other gesture sets and other physical configurations for the button. For instance, the *Power-up Button* could be positioned on the back side of a mobile device to perform gestures with the index finger. It could also be placed on one of the sides of a mouse to enable users to operate it with the thumb while operating all over controls as usual.

REFERENCES

1. Baudisch, P., and Chu, G. Back-of-device interaction allows creating very small touch devices. In *Proc. CHI '09*, ACM (2009), 1923–1932.
2. Butler, A., Izadi, S., and Hodges, S. Sidesight: multi-"touch" interaction around small devices. In *Proc. UIST '08*, ACM (2008), 201–204.
3. Choi, S., Han, J., Lee, G., Lee, N., and Lee, W. Remotetouch: touch-screen-like interaction in the tv viewing environment. In *Proc. CHI '11*, ACM (2011), 393–402.
4. Harrison, C., and Hudson, S. E. Abracadabra: wireless, high-precision, and unpowered finger input for very small mobile devices. In *Proc. UIST '09*, ACM (2009), 121–124.
5. Harrison, C., and Hudson, S. E. Minput: enabling interaction on small mobile devices with high-precision, low-cost, multipoint optical tracking. In *Proc. CHI '10*, ACM (2010), 1661–1664.
6. Holman, D., Banerjee, A., Hollatz, A., and Vertegaal, R. Unifone: Designing for auxiliary finger input in one-handed mobile interactions. In *Proc. TEI '13*, ACM (2013).
7. Kim, J., He, J., Lyons, K., and Starner, T. The gesture watch: A wireless contact-free gesture based wrist interface. In *Proc. ISWC '07*, IEEE Computer Society (2007), 1–8.
8. Rekimoto, J., Ishizawa, T., Schwesig, C., and Oba, H. Presense: interaction techniques for finger sensing input devices. In *Proc. UIST '03*, ACM (2003), 203–212.
9. Wigdor, D., Forlines, C., Baudisch, P., Barnwell, J., and Shen, C. Lucid touch: a see-through mobile device. In *Proc. UIST '07*, ACM (2007), 269–278.
10. Wilson, G., Brewster, S. A., Halvey, M., Crossan, A., and Stewart, C. The effects of walking, feedback and control method on pressure-based interaction. In *Proc. MobileHCI '11*, ACM (2011), 147–156.