TouchWheel: Enabling Flick-and-Stop Interaction on the Mouse Wheel

Sunmin Son, Jingun Jung, Auejin Ham, and Geehyuk Lee

School of Computing, Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea

ARTICLE HISTORY
Compiled January 3, 2023

ABSTRACT
The mouse wheel may induce physical fatigue by requiring users to clutch the wheel repeatedly when scrolling long distances. To address this problem, some mouse products provide a free-spinning mode in which users can spin the wheel to initiate inertial scrolling. However, the time and effort required to toggle the free-spinning mode may lower the usability of the mouse wheel. To enable inertial scrolling without the overhead of mode switching, we present TouchWheel, a physical mouse wheel with touch sensitivity and a virtual wheel model that enables users to perform flick-and-stop operations as they do on touchscreens. A user experiment revealed that participants could actively use TouchWheel's flick-and-stop operation according to their different scrolling needs. For tasks that frequently required long-distance scrolling, TouchWheel outperformed a normal mouse wheel in task completion time and required fewer clutching and mode-switching actions than the normal mouse wheel and a spin-enabled mouse wheel. For tasks that required short-distance scrolling, TouchWheel performed similarly to other baseline options.

KEYWORDS
Scrolling, mouse wheel, flick-and-stop, touch sensitive wheel

1. Introduction

Scrolling is a basic user interface concept that allows users to interact with long content that does not fit on a limited physical screen (Byrne, John, Wehrle, & Crow, 1999). Well-designed scroll interaction techniques allow users to be effective and efficient; therefore, various efforts have been made to introduce new scrolling techniques and to improve existing scrolling techniques, including flick-based (Aliakseyeu, Irani, Lucero, & Subramanian, 2008; Zimmerman & Martino, 2004), ring-based (Moscovich & Hughes, 2004; Quinn & Cockburn, 2009; G. Smith, schraefel, & Baudisch, 2005), force-based (Antoine, Malacria, & Casiez, 2017; Heo & Lee, 2011), and wheel-based techniques (Cockburn, Quinn, Gutwin, & Fitchett, 2012; K. Hinckley, Cutrell, Bathiche, & Muss, 2002; K. P. Hinckley & Cutrell, 2007).

The mouse wheel is the primary interaction device for scrolling in the desktop environment. The wheel, placed between the two mouse buttons, enables users to

This work involved human subjects or animals in its research. Approval of all ethical and experimental procedures and protocols was granted by the KAIST Bioethics Review Committee under Approval No. KH2019-179.

CONTACT Geeyhyuk Lee. Email: geehyuk@kaist.ac.kr

perform pointing and scrolling seamlessly. However, it requires repeated clutching actions when users need to scroll a long distance, which can lead to physical fatigue. To address this problem, some mouse products provide a free-spinning mode, in which users can spin the mouse wheel so that scrolling may continue even after their finger leaves the wheel (Logitech, 2016). Although the free-spinning mode is effective for long scrolling, it requires users to do explicit mode switching by pressing a button on the mouse. Given that scrolling interaction usually requires mixed uses of short and long scroll operations, the need to switch modes explicitly may have a negative impact on usability.

The touchscreen interface has the same needs of mixed uses of short and long scrolling but solves it in a way that does not require explicit mode switching. On a touchscreen, users use a flick operation to initiate an inertial scrolling and a light touch to stop it. The touchscreen can determine whether users perform a drag or flick operation based on the fingertip movement feature and does not require users to do explicit mode switching. Enabling such "flick-and-stop" operations on the mouse wheel would be a better solution than providing a separate free-spinning mode because users would be able to mix short and long scrolling operations seamlessly without needing to be aware of the current mode of the wheel.

There can be two main approaches to enable flick-and-stop operations on the mouse. One is to replace the physical wheel with a small touchpad, and the other is to make the physical wheel detect flick-and-stop operations. The first approach was proposed by the early studies on a touch mouse (Balakrishnan & Patel, 1998; Villar et al., 2009) and is used by some of the mouse products with a touchpad on the market (Chou, 2016). With a touchpad, users can flick and stop the same way as they do on the touchscreen. The second approach is enabling flick-and-stop operations on the physical wheel. A physical wheel may have advantages over the touchpad emulation of a physical wheel in respect of affordance and ergonomics. Some users may not want to trade a physical wheel for its touchpad emulation to have flick-and-stop operations, even if the touchpad emulation can provide comparable usability. Therefore, we explored this second approach in this study even though the first approach already enables users to use flick-and-stop operations on the mouse.

The implementation details of a flick operation on the touchscreen vary from system to system, but all systems use the terminal speed of the fingertip at the moment it leaves the touchscreen to detect a flick operation; they declare a flick operation if the terminal speed of the finger exceeds a certain small threshold (Quinn, Malacria, & Cockburn, 2013). Using a terminal speed is simple but intuitive and consistent with real-world physics. Suppose a finger moves an object placed on a horizontal plane. The object moves at the same speed as that of the fingertip while it is in contact with the fingertip. When it is released from the fingertip, its future speed is determined solely by its speed at the moment it is released, without being affected by its speed before the release moment, however large or small it may be. The same physics holds for the case where the fingertip rotates a wheel; the future rotation speed of the wheel is determined solely by its rotational speed at the moment it is released. Therefore, the implementation of a flick operation on the wheel, if it is intended to provide a user experience similar to that of a flick operation on the touchscreen, should use the terminal rotational speed of the wheel.

TouchWheel is a mouse wheel augmented by touch sensitivity and a virtual wheel model. Touch sensitivity is needed to detect the exact moment of a finger release so that it can measure the terminal speed of the wheel. A virtual wheel with virtual inertia starts to rotate with an initial rotational speed determined by the terminal

speed. Its rotation continues even after the real wheel stops due to friction and can be used to implement inertial scrolling. The same touch sensitivity is used to implement a stop operation; users can touch the wheel lightly to stop the virtual wheel rotation as they do on the touchscreen to stop inertial scrolling.

We implemented a TouchWheel prototype and conducted a user experiment to verify the effectiveness of TouchWheel in this study. The results showed that participants could use flick-and-stop operations on the wheel according to their needs in representative scrolling tasks. In addition, TouchWheel outperformed the normal mouse wheel in terms of task completion time and the number of clutching actions and outperformed a spin-enabled mouse in terms of the required number of discrete actions – clutching and mode switching actions – for tasks that require high-speed scrolling.

In the following section, we review some related work and describe the design of the TouchWheel prototype. We then describe the design and results of the aforementioned user experiment in detail. Finally, we conclude the study with the limitations of the study, the future possibilities of TouchWheel, and the contributions of this study.

2. Related work

We first review prior research efforts to improve scroll interactions to support the importance of scrolling interaction. We then review the history of the mouse wheel and research efforts to improve it. Finally, we review existing methods to enable flick-and-stop actions on mouse devices and point out their limitations.

2.1. Scrolling interaction

Scrolling is an important operation when interacting with electronic content. According to Byrne et al. (1999), users spent approximately forty minutes scrolling in a five-hour recording of web-browsing sessions. Accordingly, many researchers have introduced novel ideas to improve scrolling interaction. In the traditional WIMP environment, the mouse wheel is the major interface for scrolling interaction. We review the mouse wheel interface separately in the next section.

Zimmerman and Martino (2004) presented the concept of flick scrolling (or inertial scrolling) for scroll interaction on the touchscreen, and many studies have been conducted to improve flick scrolling on touchscreen devices. For example, Aliakseyeu et al. (2008) presented and evaluated several different mapping functions for flick-based scrolling interaction. They found that the performance of flick scrolling was comparable to that of scrolling using a scrollbar. Flick scrolling has been used in a wide variety of touchscreen devices, including iPhones and Android devices. Quinn et al. (2013) used reverse-engineering methods to expose the scrolling transfer functions of Apple iOS and Google Android. Their results showed that the systems implemented scrolling transfer functions with acceleration and cumulative gain algorithms. The concept of flicking has also been extensively studied and utilized for the manipulation of graphical objects in tabletop interfaces. For instance, Wu and Balakrishnan (2003) described a "flick and catch" technique in which an object is thrown once it is dragged up to a certain speed and is caught when touched again.

Scrolling via ring gestures is another key touchscreen technique. Moscovich and Hughes (2004) proposed a virtual scroll ring that mapped a circular finger motion onto vertical scrolling. G. M. Smith and schraefel (2004) designed Radial Scroll Tool that provided a circular scrolling widget divided into multiple segments. Furthermore,

G. Smith et al. (2005) introduced Curve Dial technique that enabled eyes-free ring scrolling by taking advantage of the fact that curvature is independent of location. Tu, Ren, Tian, and Wang (2014) showed that ring scrolling is more advantageous than flick scrolling when using a thumb or pen.

Another approach to improve scrolling on a touchscreen is the use of force as an additional input. Quinn and Cockburn (2009) presented a list selection interface for touch/pen devices called Zoofing, which combined pressure-based zooming and flick-based scrolling. This was shown to overcome the relative weakness of the flick-based scrolling technique in selecting a target from a long list. Heo and Lee (2011) presented a scrolling technique that utilized shear force on a touchscreen to obviate the need for clutching the pointing device while scrolling. Antoine et al. (2017) presented ForceEdge, a force-based technique in which users can control the speed of auto-scrolling on the edge.

2.2. Mouse wheel and efforts to improve them

Since its inception, the mouse wheel has been the major input device to support convenient scrolling in the desktop user interface. A mouse with a wheel was first introduced by Ohno, Fukaya, and Nievergelt (1985), but the wheel in this case was used not for scrolling but for controlling the orientation of the cursor. A mouse with a wheel for scrolling was first introduced by Dan Venolia with Apple in 1989 (Buxton, 2011). A thumb wheel on the mouse was used for scrolling in 2D applications and to move in the z-direction in 3D applications. Soon, the Genius EasyScroll mouse (1995) appeared as the first commercial wheel mouse, but the Microsoft Intellimouse (1996) fully popularized the use of the mouse wheel (Atwood, 2007). Zhai, Smith, and Selker (1997) compared a wheel mouse with other scrolling input devices in terms of scrolling efficiency. A wheel mouse was shown to be inferior to other devices in a task where participants needed to select a single prominent target in long documents successively.

Many subsequent studies followed to improve the efficiency of a mouse wheel in tasks that require long-distance scrolling. For instance, K. Hinckley et al. (2002) proposed an acceleration algorithm for a scrolling wheel. The algorithm uses a transfer function to boost the scroll gain when the wheel speed exceeds a certain threshold, enabling efficient long-distance scrolling. K. P. Hinckley and Cutrell (2007) proposed a cumulative gain algorithm that accelerated the scrolling speed when a mouse wheel was rotated repeatedly. Quinn, Cockburn, Casiez, Roussel, and Gutwin (2012) found that the Microsoft IntelliPoint (8.20.468 on Windows) mouse drivers implemented a similar cumulative gain algorithm in a reverse-engineering study of mouse wheel transfer functions. Cockburn et al. (2012) proposed a scroll transfer function that enabled both precise scrolling and rapid document traversal by making the scroll gain dependent on document length. Another approach to improve the mouse wheel is to enable users to flick on the mouse to do inertial scrolling, which is reviewed in the next section.

2.3. Inertial scrolling with the mouse wheel

One of the approaches to facilitate long-distance scrolling is inertial scrolling, which is based on the metaphor of the simple physics of an inertial object; an object with inertia continues to move even after an external force is removed. Users are now familiar with this interface concept through the everyday use of the touchscreen interface. Three approaches to enable inertial scrolling with the mouse wheel are found in the

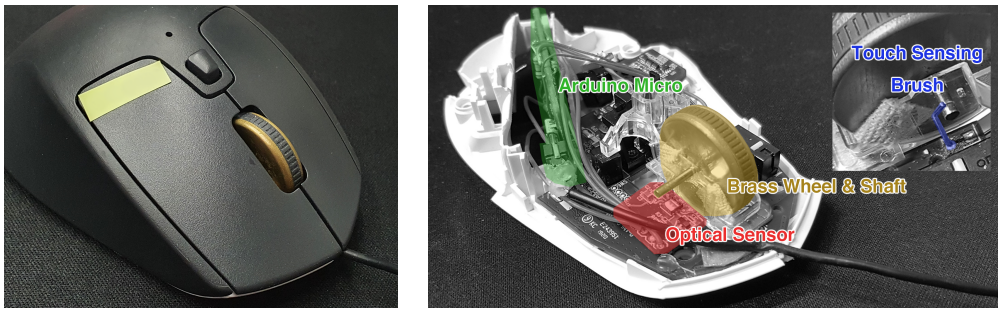
literature and/or in computer products: replacing a physical wheel with a touchpad (Balakrishnan & Patel, 1998; Chou, 2016; Villar et al., 2009), adding a free-spinning mode to a physical mouse wheel (Logitech, 2016), and using a software emulation based on the peak speed of the wheel (SmoothScroll, 2019).

The first approach is used by some of the touch mouse products on the market. See Chou (2016) for a review of recent touch mouse products. Microsoft Arc Touch Mouse (Microsoft, 2020) enables a flick operation by utilizing a touch-sensitive strip that acts like a touchpad. It also provides haptic feedback to imitate the detent feeling of an ordinary mouse wheel. Apple's Magic Mouse (AppleMagicMouse, 2015) has a smooth multi-touch surface that allows users to use a flick operation to initiate inertial scrolling. This approach may allow users to use the same flick-and-stop operations that they do on the touchscreen. An obvious downside is that users do not have a physical wheel anymore. We discuss reasons for users not to want to give up a physical wheel below.

First, a physical wheel may provide better affordance. Users can notice its presence and tell how to use it immediately. A touch mouse may lack such affordance and pose a hurdle to new users. Second, a physical wheel has an ergonomic advantage. A finger motion on the wheel is constrained to the wheel plane and is mostly one-dimensional. On the other hand, a finger motion on the touchpad is two-dimensional. A finger motion on the wheel is a guided motion with a lower degree of freedom. Another difference is in the friction that the finger experiences. The rolling friction of the wheel is almost independent of the finger force on the wheel. Therefore, users can manipulate the wheel without stick-and-slip friction, even when they rest their finger on the wheel with its full weight. Third, users may want a physical wheel even if its function can be achieved equally well using a touchpad emulation. An electronic device design that users choose is not determined by usability only but also by other values, including aesthetics.

The second method was implemented by some of the Logitech mouse products, such as the M720 Triathlon (Logitech, 2016). They have a scroll wheel that switches between normal and free-spinning modes. In the free-spinning mode, the wheel spins like a frictionless flywheel, enabling rapid and continuous scrolling. A clear advantage is that it allows users to continue to have a physical wheel, which may be important for them for the aforementioned reasons. A downside is that it requires explicit mode switching, which may negatively affect the usability of the mouse.

An example of the third approach is SmoothScroll (SmoothScroll, 2019), which initiates an inertial scrolling when the peak speed of the mouse wheel exceeds a certain threshold. This has the advantage of being able to be applied to an ordinary wheel mouse but has the limitation of not being consistent with users' mental model for a flick operation. As a flick action is detected not by the terminal speed but by the peak speed of the finger rotating the wheel, users cannot use a flick action as they do in the real world. For instance, users may want to perform slow but continuous scrolling with successive light flick operations. They may also want to quickly move the page up and down without activating inertial scrolling. Such interactions should be possible according to real-world physics that they know but are not possible when a flick operation is determined by the peak rotation speed of the wheel. Another limitation of this approach is that it does not provide an intuitive way to stop inertial scrolling. In the previous two approaches, as well as in the case of TouchWheel, users can stop an inertial scrolling by touching the wheel or the touchpad lightly. In the case of SmoothScroll, users have to rotate the wheel in the opposite direction to stop an ongoing inertial scrolling.



(a) (b)

Figure 1. TouchWheel prototype: (left) external and (right) internal structures.

3. TouchWheel

TouchWheel is an augmented mouse wheel that can recognize flick-and-stop operations. TouchWheel consists of a touch-sensitive wheel and a software module implementing a virtual wheel model. We describe them below.

3.1. TouchWheel mouse

Figure 1 shows the mouse prototype with a touch-sensitive wheel. It was based on an off-the-shelf mouse (Logitech M720 Triathlon). We modified the mouse devices in two ways. First, we added touch sensitivity by replacing the plastic wheel with a conductive wheel made of brass, which was electrically connected to a touch sensor via a brush. The brass wheel was 3D-printed, and the touch sensor was implemented using the digital I/O pins of an Arduino board (Arduino Micro) and the Arduino Capacitive Sensing Library (Badger, 2018). Second, we replaced the original rotary encoder with a high-resolution optical rotary encoder (PAT9125EL, PixArt Imaging). This replacement was required because the original encoder output an event every 15° rotation, which is too sparse for a reliable estimation of the instantaneous wheel speed required by the virtual wheel model.

For the same reason, we removed the detent mechanism around the wheel to avoid fluctuations in the speed of the wheel. In place of the detent mechanism, we added fabric spacers between the wheel and the supporting frame. The rotational friction of the wheel was set to be similar to that of a typical mouse wheel; the wheel stops almost immediately when the flicking finger releases the wheel. The same Arduino board used for implementation of touch sensing was used for reading the data from the optical encoder and sending the wheel's touch state and rotational speed to the PC every 30 ms via a serial communication link.

3.2. Virtual wheel model

Figure 2 shows the virtual wheel model that outputs the rotational speed of a virtual wheel based on the touch state and speed information of the physical wheel of the TouchWheel mouse. The virtual wheel model has two internal states, referred to as Free and Engaged. It enters the Engaged state when the finger touches the wheel and returns to the Free state when a user's finger leaves the wheel. In the Engaged state, the rotational speed of the virtual wheel ($\hat{\omega}$) is a function of the rotational speed

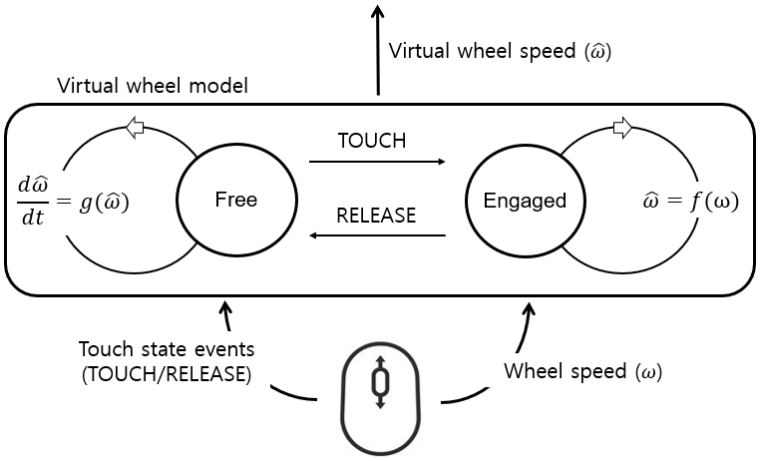


Figure 2. State diagram of the virtual wheel model.

of the physical wheel (ω): $\hat{\omega} = f(\omega)$. For example, when $\hat{\omega} = f(\omega) = a\omega$ ($a \geq 0$), $\hat{\omega}$ is proportional to ω. In the Free state, the virtual wheel is not subject to the physical wheel, and its rotation is determined by its own dynamics: $d\hat{\omega}/dt = g(\hat{\omega})$. For example, when $d\hat{\omega}/dt = g(\hat{\omega}) = -b\hat{\omega}$ ($b \geq 0$), $\hat{\omega}$ exponentially converges to 0; that is, $\hat{\omega} = \hat{\omega}(t_0)e^{-b(t-t_0)}$, where t_0 is the start time of the current Free state.

Figure 3 shows input-output examples of the virtual wheel model when $f(\omega) = a\omega$ and $g(\hat{\omega}) = -b\hat{\omega}$ ($b \geq 0$). Figure 3a corresponds to the case of dragging the wheel. The user rotates the wheel while his/her finger is touching it. In this case, the speed of the virtual wheel is the same as that of the physical wheel according to the model. Figure 3b corresponds to the case of performing flick-and-stop operations on the wheel. The user rotates the wheel, and his/her finger releases the wheel before the wheel stops. The speed of the virtual wheel is the same as that of the physical wheel during the first Engaged state but starts to decrease exponentially according to $d\hat{\omega}/dt = -b\hat{\omega}$ (solid line) from the moment the finger leaves the wheel. When the finger touches the wheel again, the speed of the virtual wheel becomes 0 as the speed of the physical wheel is already 0 at that moment due to friction. The constant b determines the exponential decay rate of the virtual wheel speed. In this regard, the constant b may be called the "friction coefficient" of the virtual wheel. The dashed line in the same graph corresponds to the extreme case of $b = 0$. The virtual wheel continues to rotate without friction and stops only when the user touches it.

The optimal choice of the friction coefficient b may differ for different scrolling tasks. It may also be a configurable parameter that users can choose as they choose an acceleration parameter for their mouse devices.

4. User Experiment

The goal of the experiment is twofold. The first was to verify whether users would be able to effectively use the flick-and-stop and drag operations enabled by TouchWheel. We analyzed participants' behaviors in three scrolling tasks that required different amounts of long-distance scrolling for this goal. The second goal was to quantify the benefits of TouchWheel with respect to other baseline conditions.

As our motivation for TouchWheel was to present a usable substitute for a free-

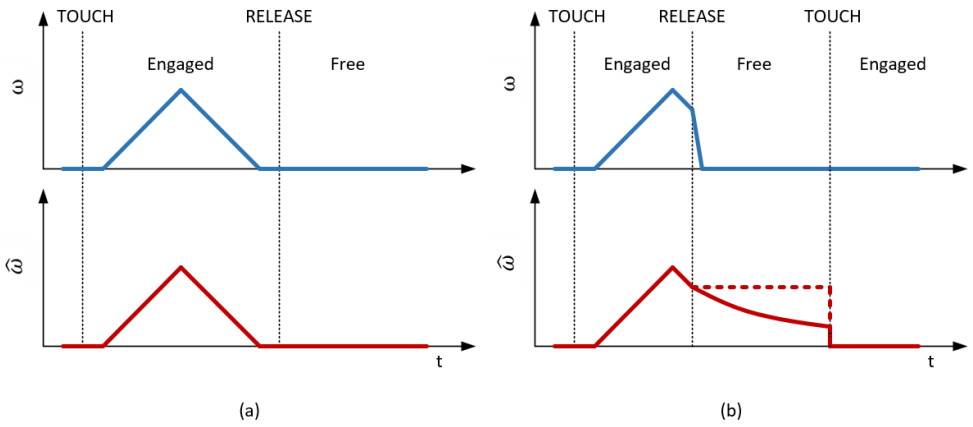


Figure 3. Input and output examples of the virtual wheel model. (a) The user drags the wheel. (b) The user performs flick-and-stop operations.

spinning mode, we chose the following three baseline conditions: an ordinary wheel mouse (Normal), a free-spinning wheel mouse (FreeSpin), and a wheel mouse toggling between normal mode and free-spinning mode (Toggle). The experiment used a within-subject design where the independent variable was $WheelType$, which is Normal, FreeSpin, Toggle, or TouchWheel. We describe the four $WheelType$ conditions, the three experimental tasks, and the quantitative performance metrics chosen for quantifying the benefits of TouchWheel in the following sections.

4.1. Apparatus

We prepared mouse devices for the following four $WheelType$ conditions: Normal, FreeSpin, Toggle, and TouchWheel. As different mouse form factors may influence the results of the experiment, we prepared all of the mouse devices based on the same mouse product: the Logitech M720 Triathlon mouse (Logitech, 2016). For the Normal condition, we modified a Logitech M720 Triathlon mouse by replacing the original rotary encoder with one extracted from a Logitech B100 mouse. The replacement encoder had 24 detents per revolution (15°/detent). The original wheel was also replaced with a custom 3D-printed wheel that fit the replacement encoder. For the FreeSpin and Toggle conditions, we used an unmodified Logitech M720 Triathlon mouse. In the FreeSpin condition, participants were only allowed to operate the mouse in the free-spinning mode. Contrarily, in the Toggle condition, participants were allowed to toggle freely between the normal and free-spinning modes. For the TouchWheel condition, we used the TouchWheel mouse prototype, which was also based on a Logitech M720 Triathlon mouse, as described in the previous section. The rotational speed output from the model was integrated to estimate the rotation amount, and a scroll-up or scroll-down event was generated whenever the rotation exceeded a threshold. The threshold was chosen as 15° so that the same amount of wheel rotation resulted in the same scrolling distance across all conditions.

For this experiment, we needed to determine the constant b, which determines the exponential decay rate of the virtual wheel speed. We first studied the exponential decay constant used in the implementation of a flick operation on a touchpad or touchscreen. Quinn et al. (2013) found that the decay constant differs depending on the device or the operating system. For example, it was 2.006 s^{-1} for iOS ScrollView

and 10.47 s^{-1} for iPhone WebView. We considered these values first but concluded that they were too large for TouchWheel. This result may be due to differences in device form factors and interaction skills. We also considered the decay rate of the mouse wheel speed when the wheel was in the free-spinning mode but concluded that it was too small for TouchWheel. This result may be due to the difference between the mental models of spinning and flicking operations. Therefore, we conducted a pilot test with four participants to choose the best b-value for TouchWheel. We considered five values: 4, 2, 1, 0.5, and 0.25 s^{-1}. Finally, we chose 1 s^{-1}, because it resulted in the best task completion times and subjective ratings.

4.2. Tasks

We designed three scrolling tasks, SortedList, UnsortedList, and MultipleTargets, as shown in Figure 4. The three tasks were carefully designed to observe users' scrolling behaviors under different application requirements, as described below.

SortedList: The list contains 800 lines that each hold an integer ranging from 001 to 800. Each integer occurs exactly once in the list. The lines are sorted in ascending order. A single line is picked randomly as the target line and is previewed to the right of the list. We expected participants to scroll long distances with a general idea of the target line's location. The task is completed when the target line is selected.

UnsortedList: The list contains 800 lines that each hold an integer ranging from 001 to 800. Each integer occurs exactly once in the list. The lines are shuffled randomly. A single line is picked randomly as the target line and is previewed to the right of the list. The target line is highlighted with a red background so that participants can easily notice it when it appears while scrolling the list. We expected participants to scroll long distances without knowing when the target line would appear. The task is completed when the target line is selected.

MultipleTargets: A digit between 1 and 9, inclusive, is picked randomly as the target digit. The list is populated with a total of 60 lines: 3 target lines that hold the target digit and 57 lines that do not hold the target digit. For example, if the target digit is 1, the list will contain 3 target lines with the value of 1, while the other 57 lines will contain evenly distributed values ranging from 2 to 9. The target digit is previewed to the right of the list. The background of target line turns red when selected. We expected participants to scroll short distances slowly while examining each line. The task is completed when all three target lines are selected.

In all tasks, the scroll gain was set as 3 lines/event; the list scrolls up or down by three lines for every scroll-up or down event from the mouse. In the early studies on the mouse wheel interface, Zhai et al. (1997) used a scroll gain of 1 line/event, but K. Hinckley et al. (2002) pointed out that it is fatiguing and impractical to scroll hundreds of lines at the gain of 1 line/event and used both 1 line/event and 3 lines/event. In our case, the choice of the gain was not critical because we could adjust the difficulty of the task by changing the size of the selection zone. Therefore, we could have used either 1 line/event or 3 lines/event, but finally chose 3 lines/event because scrolling at 1 line/event in the Normal condition felt overly slow and tiring. The choice was in favor of the Normal condition, which was the first baseline condition of the experiment.

In all tasks, a translucent green box was overlaid at the center of the list to indicate

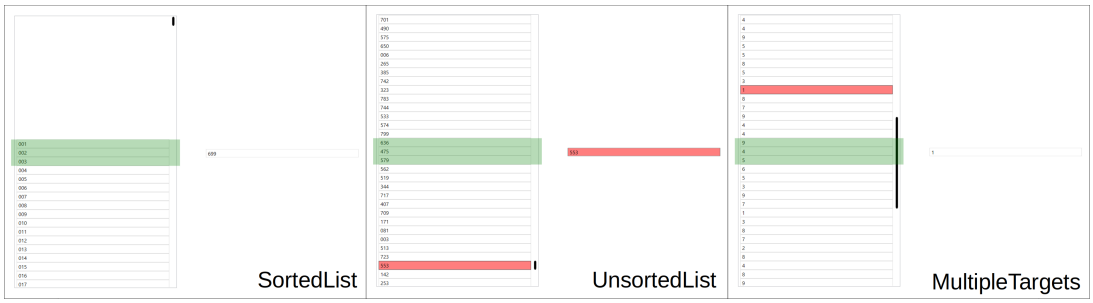


Figure 4. Three scrolling tasks designed for the experiment.

a selection zone. The height of the selection zone was set to three lines because the scroll gain was 3 lines/event. (If 1 line/event was chosen as the scroll gain, the height of the selection zone could have been set to one line.) To prevent the selection zone from containing multiple target lines in the MultipleTargets task, the target lines in the MultipleTargets task were arranged so that two adjacent target lines were at least three lines apart from each other.

The mouse cursor was hidden during the task so that participants could focus on scrolling and selecting operations. In order to make a selection, participants were instructed to scroll the target item into the selection zone and click the left mouse button. To motivate participants to select targets carefully and accurately, a pleasant sound was played by the experiment interface when a target was successfully selected. On the other hand, a red flashing animation with a beeping sound was played when an incorrect selection was made.

4.3. Performance metrics

The performance metrics that we chose as dependent variables were task completion time ($CompletionTime$), the number of errors ($Errors$), the number of discrete actions ($DiscreteActions$), the NASA Task Load Index (TLX), and preference ($Preference$).

$CompletionTime$ was measured from the moment the task type and target were revealed to a participant until the moment the target was successfully selected. In the MultipleTargets task, the time was measured until all three targets were selected. $Errors$ were defined as the number of all mouse clicks made when the target was not within the selection zone.

$DiscreteActions$ included clutching actions on the wheel and toggling actions for switching the wheel modes. The number of clutching actions, such as replacing the mouse and replacing the finger on the touchpad, is often used as a metric to reflect manual effort in the use of input devices (Nancel, Vogel, & Lank, 2015). In addition to clutching actions, toggling actions in the Toggle condition require an effort comparable to or higher than clutching the wheel. $DiscreteActions$ accounts for the manual effort of both actions. $DiscreteActions$ were counted by analyzing the recorded videos of all sessions.

4.4. Procedure

Twelve participants (five women) aged 19 to 26 (average age = 21.4) were recruited for the experiment. The experiment was carried out one by one and each participant

was paid 13,000 KRW (approximately 10.6 USD).

The participant first went through a tutorial where the experimenter explained the four wheel interfaces and the three types of tasks. During the tutorial, the participant practiced performing a single trial of each task using each wheel interface (12 trials). Then, the participant performed three blocks of the experiment. In each block, each of the four interfaces was used to perform the three different tasks, three trials each. This resulted in 36 trials per block ($4 \times 3 \times 3$). Therefore, a total of 120 trials ($12 + 36 \times 3$) were performed throughout the experiment. During a block, one interface was used to perform all three tasks (9 trials) before switching to the next interface. During those 9 trials, the task type was randomized so that the participant could not predict the next task. The order of the interfaces used was counterbalanced using a balanced Latin square.

We imposed balancing constraints on the target positions to balance task difficulty among the interface conditions. In the SortedList and UnsortedList tasks, targets appeared between positions 100 and 800, with an average of 450 for each interface condition. In the MultipleTargets task, each of the three targets appeared within each third (20-item section) of the list. To equalize the task difficulty among participants, the target locations were determined beforehand and the same target locations were used for all participants.

After the first block, first impressions of each interface were collected through a semi-structured interview. During the final block, NASA TLX was performed to evaluate the task load of the four interfaces. When all three blocks were completed, the preferred interface for each task and additional comments were obtained through another short interview. The experiment lasted approximately an hour for each participant.

5. Results

We evaluated the effectiveness of TouchWheel by analyzing the user behaviors in the Toggle and TouchWheel conditions. We then computed the metrics $CompletionTime$, $Errors$, $DiscreteActions$, TLX, and $Preference$, in the final block of the four interface conditions. We excluded data from the first two blocks because we wanted to compare the conditions after the participants became accustomed to the use of the different scrolling interfaces and tasks. For effectiveness, $CompletionTime$, $DiscreteActions$, and TLX, we performed one-way repeated measures ANOVA (rm-ANOVA) or the Friedman test for parametric or nonparametric data, respectively. We used the Holm–Bonferroni correction for post hoc tests. For $Errors$, we performed a goodness-of-fit Chi-square test to compare error rates between scrolling interfaces. For $Preference$, we did not perform any statistical tests and included the interview schedule and a summary of responses in the appendix for reference.

5.1. Effectiveness of TouchWheel

The first goal of the experiment was to verify whether users could use the flick-and-stop and drag operations enabled by TouchWheel, according to their intentions. First, for the Toggle condition, we analyzed participants' use patterns of flick and drag operations according to the type of task. Similarly, for the TouchWheel condition, we analyzed participants' use patterns of flick-and-stop and drag operations according to the type of task in order to compare with those of the Toggle condition. We believed that if the usage patterns of flick-and-stop and drag operations on TouchWheel were

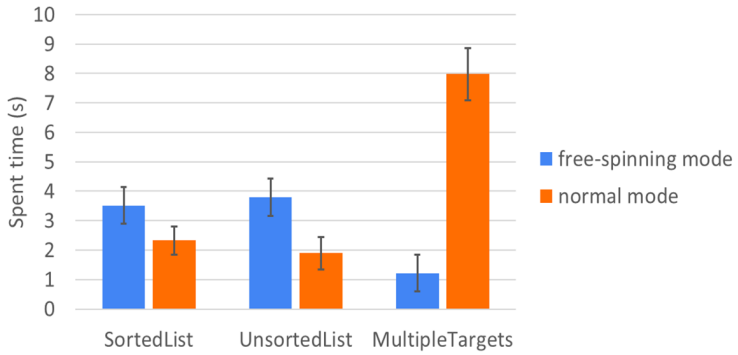


Figure 5. The average amount of time participants spent in each of the normal and free-spinning modes for each of the three tasks in the Toggle condition. The intervals on the bars represent standard errors.

similar to the usage patterns of free-spinning and normal modes on Toggle, then participants could use the flick-and-stop and drag operations of TouchWheel effectively. Therefore, we analyzed the experiment videos and measured the following two metrics.

(1) The average amount of time in seconds that participants spent in each mode (normal or spin) for each of the three tasks in the Toggle condition.
(2) The average number of flick-and-stop operations that participants performed for each of the three tasks in the TouchWheel condition.

We expected the first metric to reveal the different needs of long-distance scrolling in the three different tasks. The results are shown in Figure 5. As we expected when we designed the tasks, participants spent more time in the free-spinning mode than in the normal mode while they performed the SortedList and UnsortedList tasks. In contrast, they spent most of the task time in the normal mode when they performed the MultipleTargets task. The proportions of the task time spent in the free-spinning mode were 0.58, 0.65, and 0.15 on average for the SortedList, UnsortedList, and MultipleTargets tasks, respectively. A Friedman test showed that the differences in the average values are statistically significant ($\chi^2(2) = 14.00, p < 0.001$). Pair-wise Wilcoxon signed rank tests with Holm–Bonferroni correction showed statistically significant differences between SortedList and MultipleTargets ($z = -2.668, p < 0.05$) and between UnsortedList and MultipleTargets ($z = -2.666, p < 0.05$). It can be concluded that the SortedList and UnsortedList tasks demanded long-distance scrolling more than the MultipleTargets task.

It may then be expected that participants must have used flick-and-stop operations more frequently in the SortedList and UnsortedList tasks than in the MultipleTargets task if they found the flick-and-stop operation enabled by TouchWheel to be effective for long-distance scrolling. The second metric defined above turned out to be consistent with this expectation, as shown in Figure 6. The proportion of flick-and-stop operations among all wheel operations were 0.47, 0.62, and 0.02 on average for the SortedList, UnsortedList, and MultipleTargets tasks, respectively. A Friedman test showed that the differences in the average values are statistically significant ($\chi^2(2) = 21.83, p < 0.001$). Pair-wise Wilcoxon signed rank tests with Holm–Bonferroni correction showed that there were statistically significant differences between every pair of the three conditions, that is, between SortedList and UnsortedList ($z = -2.714, p < 0.01$), between SortedList and MultipleTargets ($z = -3.061, p < 0.01$), and between UnsortedList and MultipleTargets ($z = -3.063, p < 0.01$). It may be concluded that participants

Figure 6. The average number of flick-and-stop and drag operations that participants performed for each of the three tasks in the TouchWheel condition. The intervals on the bars represent standard errors.

actively adopted the flick-and-stop operations enabled by TouchWheel when the task required long-distance scrolling. This conclusion is more meaningful as the differences were from the final block of the experiment. In the final block, they must have been less influenced by the experimenters' motivation and adopted flick-and-stop operations according to the task's demand.

An encouraging observation during this data analysis was that two participants who did not use the free-spinning mode in the Toggle condition at all used flick-and-stop operations in the TouchWheel condition as actively as other participants. This observation, though anecdotal, suggests the relatively lower adoption barrier of the flick-and-stop interaction than that of the free-spinning mode.

5.2. Task completion time

Figure 7 shows the task completion times ($CompletionTime$) of the three types of tasks under the four interface conditions. For the SortedList task, a one-way repeated measures ANOVA (rm-ANOVA) showed a statistically significant effect of $WheelType$ on $CompletionTime$ ($F_{3,33} = 18.72, p < 0.001$). Post hoc tests using the Holm–Bonferroni correction showed that $CompletionTime$ was higher in the Normal condition than in other conditions. For the UnsortedList task, the Friedman test was used because of normality violation. The results showed a statistically significant effect of $WheelType$ on $CompletionTime$ ($\chi^2(2) = 7.60, p < 0.05$). A post hoc analysis using Wilcoxon signed-rank tests with a Holm–Bonferroni correction showed statistically significant differences between Normal and FreeSpin ($Z = -2.67, p < 0.01$) and between Normal and TouchWheel ($Z = -3.059, p < 0.01$). For the MultipleTargets task, an rm-ANOVA showed a statistically significant effect of $WheelType$ on $CompletionTime$ ($F_{3,33} = 3.40, p < 0.05$). However, post hoc tests with the Holm–Bonferroni correction indicated no statistically significant difference between the conditions.

5.3. Number of errors

Table 1 shows the total selection errors for each interface condition during the final block. The error events were not frequent; therefore, we summed the number of errors for all three tasks and for all participants to obtain statistically significant numbers ($Errors$). During the final block, the participants were asked to select a target 180

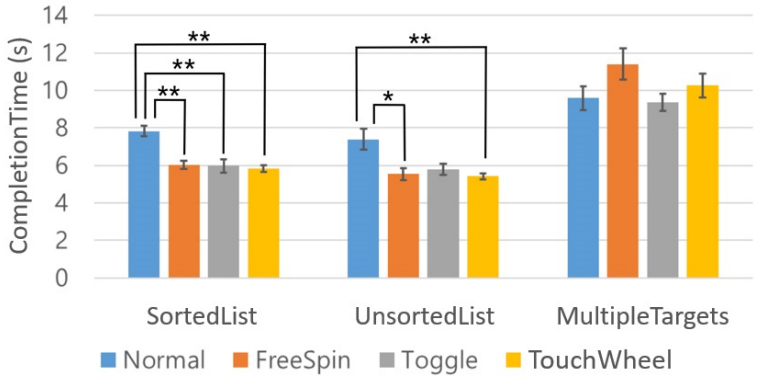


Figure 7. Average task completion times of the three types of tasks for the four interfaces. "*" and "**" indicate $p < 0.05$ and $p < 0.01$, respectively, and the intervals on the bars represent standard errors.

	Interface			
	Normal	FreeSpin	Toggle	TouchWheel
Errors	2	10	2	0

Table 1. Number of errors (sum over all participants and all tasks) during the final block.

times for each interface condition. Therefore, the ten errors for *FreeSpin* in the table correspond to an error rate of 5.6%. A goodness-of-fit Chi-square test showed that the error rates are not uniform over the four interfaces ($\chi^2_3 = 16.86, p < 0.001$), suggesting a statistically meaningful difference, at least between FreeSpin and TouchWheel.

The high error rate in the FreeSpin condition may be due to an error that the participants committed while they moved their finger away from the wheel to press the left button. Immediately before a mouse click, the participants unintentionally rotated the frictionless wheel, causing the target to be misaligned from the selection zone. Similar errors were also observed when participants used the frictionless mode of Toggle to select the target. Most participants reported difficulty using FreeSpin to scroll precisely. In particular, two participants said FreeSpin felt uncomfortable because of frequent selection errors. Errors occasionally occurred in the Normal condition when participants misclicked the left button while trying to clutch the mouse wheel rapidly. Errors did not occur when the participants used TouchWheel.

5.4. Number of discrete actions

Figure 8 shows the average number of discrete actions per trial (*DiscreteActions*). For the SortedList task, an rm-ANOVA indicated statistically significantly different means ($F_{3,33} = 167.3, p < 0.001$). Post hoc tests using the Holm–Bonferroni correction indicated that *DiscreteActions* in Normal was significantly higher than in FreeSpin ($p < 0.001$), Toggle ($p < 0.001$), and TouchWheel ($p < 0.001$). Additionally, *DiscreteActions* in Toggle was statistically significantly higher than in TouchWheel ($p < 0.05$).

Owing to normality violation, a Friedman test was performed for the UnsortedList task. The results indicated a significantly different means ($\chi^2(2) = 28.98$, $p < 0.001$). A post hoc analysis with Wilcoxon signed-rank tests with a Holm–Bonferroni correction showed that *DiscreteActions* in Normal for the SortedList task was significantly higher than in FreeSpin ($p < 0.01$), Toggle ($p < 0.01$), and TouchWheel ($p < 0.01$).

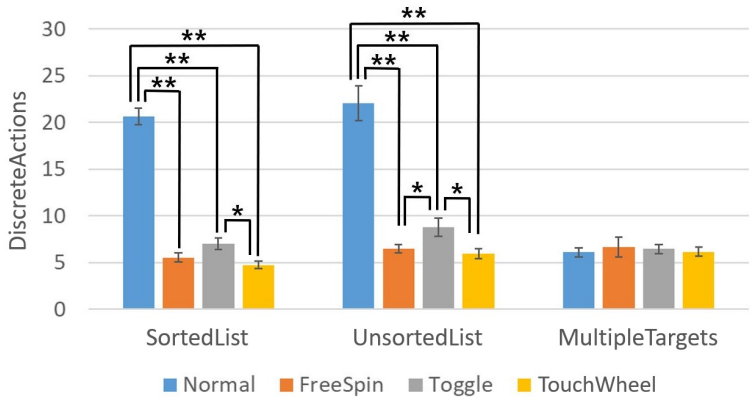


Figure 8. Average number of discrete actions per trial in the final block. "*" and "**" indicate $p < 0.05$ and $p < 0.01$, respectively, and the intervals on the bars represent standard errors.

Additionally, *DiscreteActions* of Toggle was significantly higher than that of FreeSpin ($p < 0.05$) and TouchWheel ($p < 0.05$). There were no statistically significant differences in *DiscreteActions* for the MultipleTargets task.

When participants were required to scroll long distances, for example, in the SortedList and UnsortedList tasks, Normal resulted in statistically significantly higher *DiscreteActions* than the other three conditions. Furthermore, Toggle resulted in statistically higher *DiscreteActions* than TouchWheel for both SortedList and UnsortedList tasks. Toggle required toggle actions between the two wheel modes, and therefore required higher *DiscreteActions* than TouchWheel.

The UnsortedList task differed from the SortedList task in that the target location was unknown to the participants. Therefore, when using a flick operation, they found it difficult to stop near the target. Instead, they usually scrolled beyond the target and then returned to the target. This explains the slightly higher *DiscreteActions* of the flick scrolling interfaces for the UnsortedList task than for the SortedList task.

5.5. NASA TLX

Figure 9 shows the weighted NASA TLX scores (TLX). An rm-ANOVA test indicated statistically significant differences ($F_{3,33} = 10.23$, $p < 0.001$). Post hoc tests using the Holm–Bonferroni correction indicated that Normal had a significantly higher TLX than those of Toggle ($p < 0.05$) and TouchWheel ($p < 0.05$) and that FreeSpin had a significantly higher TLX than those of Toggle ($p < 0.05$) and TouchWheel ($p < 0.01$). This means that the participants' perceived workloads of Normal and FreeSpin were higher than those of Toggle and TouchWheel. In other words, participants felt that a significant amount of additional effort was required when using Normal or FreeSpin than when using Toggle or TouchWheel.

5.6. Preference

In the post-experiment interview, participants were asked to indicate their preferred scrolling interface for each task (Table 2). TouchWheel was chosen as the preferred interface most frequently for the SortedList and UnsortedList tasks. However, for the MultipleTargets task, seven participants preferred Normal, five preferred Toggle, and

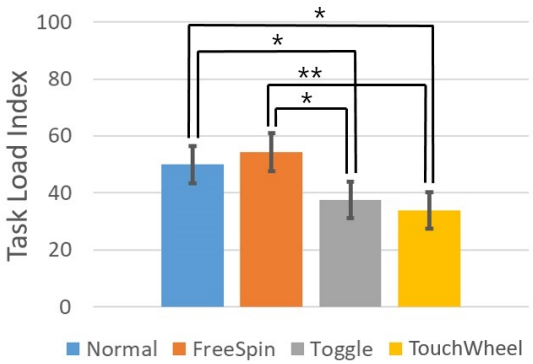


Figure 9. Weighted NASA TLX scores of the four interfaces. "*" and "**" indicate $p < 0.05$ and $p < 0.01$, respectively, and the intervals on the bars represent standard errors.

Task	Interface			
	Normal	FreeSpin	Toggle	TouchWheel
SortedList	0	3	2	**7**
UnsortedList	0	2	4	**6**
MultipleTargets	**7**	0	5	0

Table 2. Frequency of most preferred scrolling interface by task. The largest number for each task is marked in bold.

none preferred TouchWheel. The main reason for not preferring TouchWheel for the MultipleTargets task was the lack of detent feedback. In the case of TouchWheel, the wheel rotated with constant friction but without detent feedback. In contrast, Normal and Toggle in the normal mode provided detent feedback for every 15° rotation, which helped prevent slippage when selecting a target (most participants used Toggle in the normal mode for the MultipleTargets task).

5.7. Summary

TouchWheel was successfully able to support flick-and-stop skills that improved task completion time compared to Normal for the SortedList and UnsortedList tasks. These improvements were comparable to or better than those of FreeSpin and Toggle. Moreover, TouchWheel resulted in a significantly lower number of errors than FreeSpin because of FreeSpin's frictionless wheel issue. This result is also in accordance with the significantly lower NASA TLX score of TouchWheel compared to FreeSpin.

Additionally, TouchWheel resulted in a significantly lower number of discrete actions than Toggle for the SortedList and UnsortedList tasks. Although Toggle was generally well received, it was evaluated negatively because of its mode-switching experience. During the post-experiment interview, ten out of twelve participants sympathized with the need for the two modes of Toggle, but five of them said that it was uncomfortable to press the toggle button. The absence of a statistically significant difference in task completion time between Normal and Toggle in the UnsortedList task might have been due to the overhead of pressing the toggle button in the Toggle condition.

6. Discussion

6.1. TouchWheel with a dynamic decay rate

We used a fixed value of 1 as the decay rate of the virtual wheel for the TouchWheel condition, which was determined based on the result of a preliminary experiment. In our post-experiment interview, we asked the participants if they were satisfied with the decay rate when using TouchWheel. The responses were divided as follows: three participants wanted a higher decay rate (faster deceleration), seven participants liked the current settings, and two participants wanted a lower decay rate (slower or no deceleration). In fact, changing the decay rate can result in a trade-off. If the decay rate is increased, the number of overshoots may be reduced, but the number of discrete actions when scrolling long distances may increase. In contrast, if the the decay rate is decreased, long distances may be reached with a lower number of discrete actions, but the number of overshoots may increase. TouchWheel may require a different optimal decay rate for different application situations; therefore, the decay rate may be adjusted dynamically depending on the application context. For example, document length-dependent gain (Cockburn et al., 2012) used the length of a document to control the maximum attainable gain level of scrolling. Longer documents can be traversed quickly because they result in higher gain values. Similarly, TouchWheel may use a document-length-dependent decay rate.

6.2. Infinite flick

An extreme case in which the decay rate is zero may pose a new possibility. This case is illustrated by the dashed line in Figure 3. In this case, a flick action results in scrolling without slowing down; therefore, we refer to this interaction as an infinite flick. The interaction may be similar to the powered Flick-and-Brake proposed by Baglioni, Malacria, Lecolinet, and Guiard (2011). An infinite flick may allow users a unique experience that is impossible with a physical wheel.

To test the concept of an infinite flick, we implemented it and evaluated it in a small workshop with 12 participants. To enter an infinite flick state, the user had to flick on the mouse wheel twice within a short period (300 ms). In the infinite flick state, the user could pause scrolling by touching the mouse wheel, but when the touch is released from the wheel, scrolling will resume at the same speed as before. Thus, by touching the mouse wheel occasionally, the user may pause the scrolling of the list intermittently. This behavior was expected to be useful when the user wants to scan the contents of a list for a short period before moving on. To escape from an infinite flick state, the user could either double-tap the wheel or scroll the wheel slightly in the opposite direction.

During the workshop, the participants performed the same scrolling tasks used in the main user experiment. Many participants responded that infinite flick would be useful in situations that require large amounts of scrolling (e.g., shopping sites or webtoons). One participant commented that the mapping of two quick consecutive flick gestures for initiating an infinite flick seemed intuitive and reasonable because it matched the intention to keep scrolling. Another participant said that, although the touch-to-pause function felt unnatural initially, they imagined that it might be useful in many situations, such as when a long list needs to be skimmed in search of an item.

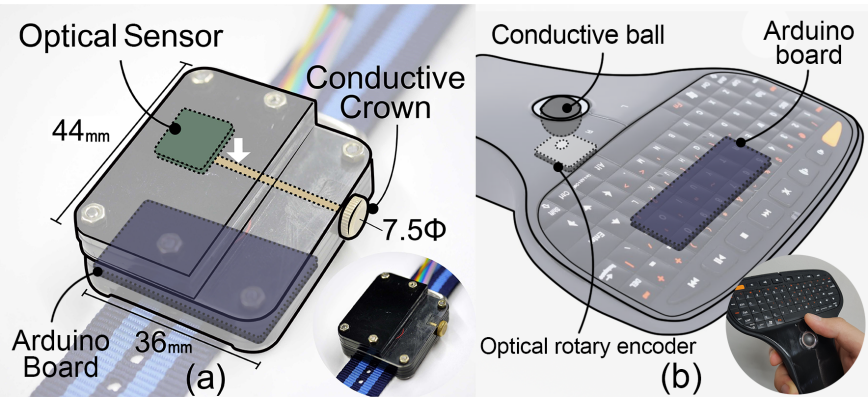


Figure 10. Applying the TouchWheel concept to other rotary input devices: (a) a smartwatch crown and (b) a mini trackball.

6.3. Applying TouchWheel to other rotary input devices

The concept of TouchWheel can be applied not only to the mouse wheel but also to other rotary input devices, such as a trackball and a smartwatch crown. A trackball allows pointing and scrolling inputs while requiring a small workspace compared to a mouse. However, trackballs are known to perform poorly when the device must be clutched to travel long distances (Accot & Zhai, 1999; Natapov & MacKenzie, 2010). Therefore, enabling flick scrolling on trackballs may resolve the clutching problem. Watch crowns have been used as an alternative to smartwatch touchscreens due to the "fat-finger" problem (Baudisch & Chu, 2009; Siek, Rogers, & Connelly, 2005). However, the small form factor of watch crowns makes it difficult to scroll a long distance. Thus, the flick scrolling functionality may also be useful for smartwatch crowns.

We implemented a trackball prototype by modifying the Lenovo Mini Wireless Keyboard N5901 and a watch crown prototype using a 3D-printed brass crown wheel, as shown in Figure 10. We collected user feedback about these TouchWheel-augmented devices in the same workshop where we tested the concept of an infinite flick. Participants performed scrolling tasks using our trackball and watch crown prototypes, with flick-and-stop enabled or disabled. All the participants responded that they preferred the flick-and-stop-enabled condition for the SortedList and UnsortedList tasks. Many participants expressed frustration when the flick-and-stop function was turned off. One participant said, "I thought the prototype was broken when the flick-and-stop function was turned off." Overall, the implementation of TouchWheel on other rotary input devices was well received.

6.4. TouchWheel with detent feedback

TouchWheel was most frequently preferred to other interfaces in the SortedList and UnsortedList tasks but not for the MultipleTargets task. Participant feedback showed this was primarily due to the lack of detent feedback. Detent feedback, which helped them prevent a slip while selecting a target, was a key preference factor when fast scrolling was not needed and clutching burden was not an issue. Our focus in this study was the problem of repeated clutching in fast scrolling and overlooked the importance of detent feedback for a stable target selection. We initially attempted to implement TouchWheel without removing detents. However, we faced an implementa-

tion difficulty. With detents, the rotational speed of the wheel oscillated significantly. With this oscillation, a stable estimation of the terminal speed of the wheel was difficult to achieve. Therefore, we decided to remove the detent structure without considering the importance of detent feedback in slow-scroll tasks. We believe that we could have used a smoothing algorithm to estimate a stable terminal speed despite the oscillation of the rotational speed. Alternatively, we could have used detent feedback emulation by providing vibration feedback, as some touch mouse products (Microsoft, 2020) do. There is a clear need for a follow-up study to investigate how TouchWheel, with detent feedback, could further improve the user experience of a mouse wheel.

7. Conclusion

We present TouchWheel, a novel, augmented mouse wheel that enables users to perform flick-and-stop operations for long-distance scrolling. Unlike a touch mouse, TouchWheel does not sacrifice the physical mouse wheel, while unlike a mouse wheel with a free-spinning mode, TouchWheel does not require explicit mode switching. Analysis of user behaviors in the main experiment revealed that participants could effectively use TouchWheel's flick-and-stop operation when long-distance scrolling was needed. Moreover, TouchWheel demonstrated advantages in terms of quantitative performance measures. In tasks that required long-distance scrolling, TouchWheel not only exhibited faster task completion times when compared to Normal, but also resulted in a lower number of discrete actions than that of Toggle. In addition, TouchWheel produced a lower error rate than FreeSpin by providing an appropriate amount of friction. TouchWheel may also be beneficial in domains other than the desktop environment, as its concept can be applied to other rotary input devices such as trackballs and smartwatch crowns. Future work involves improving TouchWheel by adding detent feedback to the wheel or by modifying the decay rate of the scrolling speed in the Free state.

References

Accot, J., & Zhai, S. (1999). Performance evaluation of input devices in trajectory-based tasks: an application of the steering law. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 466–472).

Aliakseyeu, D., Irani, P., Lucero, A., & Subramanian, S. (2008). Multi-flick: An evaluation of flick-based scrolling techniques for pen interfaces. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 1689–1698).

Antoine, A., Malacria, S., & Casiez, G. (2017). ForceEdge: Controlling autoscroll on both desktop and mobile computers using the force. In *Chi 2017.*

Apple. (2015). *Magic mouse 2.* Retrieved from `https://www.apple.com/shop/product/MLA02LL/A/magic-mouse-2-silver`

Atwood, J. (2007). *Meet the inventor of the mouse wheel.* Retrieved from `https://blog.codinghorror.com/meet-the-inventor-of-the-mouse-wheel/` (Online; accessed 2/27/2022)

Badger, P. (2018). *Capacitive sensing library.* Retrieved from `https://playground.arduino.cc/Main/CapacitiveSensor/`

Baglioni, M., Malacria, S., Lecolinet, E., & Guiard, Y. (2011). Flick-and-Brake: Finger control over inertial/sustained scroll motion. In *CHI '11 extended abstracts on human factors in computing systems* (pp. 2281–2286).

Balakrishnan, R., & Patel, P. (1998). PadMouse: Facilitating selection and spatial positioning for the non-dominant hand. In *Chi 1998.*

Baudisch, P., & Chu, G. (2009). Back-of-device interaction allows creating very small touch devices. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 1923–1932).

Buxton, W. (2011). *Human input to computer systems: Theories, techniques and technology.* Retrieved from `https://www.billbuxton.com/inputManuscript.html` (Online; accessed 3/23/2022)

Byrne, M. D., John, B. E., Wehrle, N. S., & Crow, D. C. (1999). The tangled web we wove: A taskonomy of WWW use. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 544–551).

Chou, J. R. (2016). An empirical study of user experience on touch mice. *Eurasia Journal of Mathematics, Science and Technology Education*, *12*(11), 2875–2885.

Cockburn, A., Quinn, P., Gutwin, C., & Fitchett, S. (2012). Improving scrolling devices with document length dependent gain. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 267–276).

Heo, S., & Lee, G. (2011). Force Gestures: augmenting touch screen gestures with normal and tangential forces. In *Proceedings of the 24th annual ACM symposium on user interface software and technology.*

Hinckley, K., Cutrell, E., Bathiche, S., & Muss, T. (2002). Quantitative analysis of scrolling techniques. In *Chi 2002.*

Hinckley, K. P., & Cutrell, E. B. (2007). *Distance-based accelerated scrolling.* Google Patents. (US Patent 7,173,637)

Logitech. (2016). *M720 triathlon.* Retrieved from `http://www.logitech.com/en-us/product/m720-triathlon`

Microsoft. (2020). *Arc touch mouse.* Retrieved from `http://www.microsoft.com/accessories/en-us/products/mice/arc-touch-mouse/rvf-00052`

Moscovich, T., & Hughes, J. F. (2004). Navigating documents with the virtual scroll ring. In *Proceedings of the 17th annual ACM symposium on user interface software and technology* (pp. 57–60).

Nancel, M., Vogel, D., & Lank, E. (2015). Clutching is not (necessarily) the enemy. In *Conference on human factors in computing systems - proceedings* (Vol. 2015-April, pp. 4199–4202).

Natapov, D., & MacKenzie, I. S. (2010). The trackball controller: improving the analog stick. In *Proceedings of the international academic conference on the future of game design and technology* (pp. 175–182).

Ohno, K., Fukaya, K.-i., & Nievergelt, J. (1985). A five-key mouse with built-in dialog control. *SIGCHI Bulletin*, *17*(1), 29–34.

Quinn, P., & Cockburn, A. (2009). Zoofing!: faster list selections with pressure-zoom-flick-scrolling. In *Proceedings of the 21st annual conference of the australian computer-human interaction special interest group* (pp. 185–192).

Quinn, P., Cockburn, A., Casiez, G., Roussel, N., & Gutwin, C. (2012). Exposing and understanding scrolling transfer functions. In *Proceedings of the 25th annual ACM symposium on user interface software and technology* (pp. 341–350).

Quinn, P., Malacria, S., & Cockburn, A. (2013). Touch scrolling transfer functions. In *Proceedings of the 26th annual ACM symposium on user interface software and technology* (pp. 61–70).

Siek, K. A., Rogers, Y., & Connelly, K. H. (2005). Fat finger worries: how older and younger users physically interact with pdas. In *IFIP conference on human-computer interaction* (pp. 267–280).

Smith, G., schraefel, m. c., & Baudisch, P. (2005). Curve Dial: Eyes-free parameter entry for guis. In *CHI '05 extended abstracts on human factors in computing systems* (pp. 1146–1147).

Smith, G. M., & schraefel, m. c. (2004). The Radial Scroll Tool: Scrolling support for stylus-

or touch-based document navigation. In *Proceedings of the 17th annual ACM symposium on user interface software and technology* (pp. 53–56).

SmoothScroll. (2019). *Smoothscroll windows.* Retrieved from `http://www.smoothscroll.net/`

Tu, H., Ren, X., Tian, F., & Wang, F. (2014). Evaluation of flick and ring scrolling on touch-based smartphones. *International Journal of Human–Computer Interaction*, *30*(8), 643-653.

Villar, N., Izadi, S., Rosenfeld, D., Benko, H., Helmes, J., Westhues, J., ... Others (2009). Mouse 2.0: multi-touch meets the mouse. In *Proceedings of the 22nd annual ACM symposium on user interface software and technology* (pp. 33–42).

Wu, M., & Balakrishnan, R. (2003). Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *Proceedings of the 16th annual ACM symposium on user interface software and technology* (pp. 193–202).

Zhai, S., Smith, B. A., & Selker, T. (1997). Improving browsing performance: A study of four input devices for scrolling and pointing tasks. In *Human-computer interaction interact '97* (pp. 286–293).

Zimmerman, J., & Martino, J. A. (2004). *Touch-screen image scrolling system and method.* Google Patents. (US Patent 6,690,387)

Appendix

Short interview after block 1

1. Which finger do you usually use to operate the mouse wheel?
   - Index finger (5), middle finger (6), both (1)

2. What are your first impressions of Normal, TouchWheel, Freespin, and Toggle?
   - **Normal:** familiar(6), similar to mine (4), uncomfortable when scrolling long distances (4), ordinary (2), good for precise scrolling (2), comfortable (1)
   - **Freepsin:** uncomfortable and difficult for precise scrolling (9), good for fast scrolling (8), good overall (2), difficult to stop where I want it to (2)
   - **Toggle:** uncomfortable to press button to switch modes (6), used free-spinning mode for long distance scrolling and normal mode for precise scrolling (4), only used normal mode (2), only used free-spinning mode (2)
   - **TouchWheel:** comfortable (6), good for fast scrolling (4), good for all scrolling speeds (3), similar to touchscreen flick scrolling (2), good when stopping (2), different from touchscreen flick scrolling (1), scrolling speed a little too fast (1), comfortable to scroll up (1), likable smoothness (1)

Post-experiment interview

1. Please rank the scrolling interfaces according to your overall preference.
   - **First:** Toggle (6), TouchWheel (5), Freespin (1), Normal(0)
   - **Second:** Normal (4), Toggle (4), TouchWheel (3), Freespin (1)
   - **Third:** Normal (5), Freespin (3), TouchWheel (3), Toggle (1)
   - **Fourth:** Freespin (7), Normal (3), Toggle (1), TouchWheel (1)

2. Please indicate the most preferred scrolling interface, according to the task.
   - **SortedList:** TouchWheel (7), Toggle (2), Freespin (3), Normal (0)
   - **UnsortedList:** TouchWheel (6), Toggle (4), Freespin (2), Normal (0)

- **MultipleTargets:** Normal (7), Toggle (5), Freespin(0), TouchWheel (0)

3. What are the reasons for your preference?
   - **Normal:** very uncomfortable for scrolling long distances (6), best for MultipleTargets task (3), good for precise scrolling (3)
   - **Freespin:** difficult to control when precise scrolling (6), good for fast scrolling (4), good for SortedList task because the location of the target is known (2), best for all tasks (1), slower than TouchWheel (1)
   - **Toggle:** mode switching allows both fast and precise scrolling (6), mode switching is uncomfortable (2), better than TouchWheel because it gives feedback (1), good for fast scrolling (1)
   - **TouchWheel:** good for fast scrolling (3), easy to stop (2), immediate mode switching (2), better deceleration than Freespin (1), intuitive (1), mentally demanding (1), less preferred than Toggle because of no detent feedback (1), unclear position between Freespin and Normal (1), best overall without big issues (1)

4. How did Normal feel when compared to the mouse wheels you usually use?
   - Similar (6), same (4), ordinary (1), a little stiffer (1)

5. Did you switch modes when using Toggle? If so, how was that experience?
   - Yes, mode switching was okay (4)
     Yes, but the mode switching was uncomfortable (3)
     Yes, but only at the beginning of each task: free-spinning mode for SortedList and UnsortedList, normal mode for MultipleTargets (2)
     No, I only used normal mode because I didn't need the free-spinning mode (1)
     No, I only used normal mode because mode switching was uncomfortable (1)
     No, I only used free-spinning mode because mode switching was uncomfortable (1)

6. How was the decay rate of the scrolling speed when using TouchWheel's flick-and-stop operation?
   - Adequate (7), prefer faster deceleration (3), prefer slower deceleration (1), prefer no deceleration (1)

7. Was TouchWheel's flick-and-stop operation similar to the flick scrolling interaction in smartphones?
   - Similar (4), smartphones decelerate a little faster (3), difficult to compare (3), TouchWheel was more comfortable (1), similar for scrolling down but different for scrolling up (1)

8. What do you imagine an ideal mouse wheel would be like?
   - TouchWheel is closest to an ideal mouse wheel (4),
     Toggle is closest to an ideal mouse wheel (2),
     TouchWheel with detent (1),
     sensitive wheel, but with feedback (1),
     a mouse that can detect my different intentions (1),
     Toggle in normal mode allows inertial scrolling (1),
     Toggle with a more comfortable mode switching method (1),
     wheel that requires the least amount of contact (1)

About the authors

Sunmin Son received BS and MS degrees in Computer Science at KAIST. He is currently working as a research engineer in Beyless, Korea. His current research interests include AI models in autonomous vehicles and human-machine interaction.

Jingun Jung received a MS degree in Bio and Brain Engineering and a PhD degree in Computer Science from KAIST. He is currently working as an interaction designer at Samsung Electronics. His current research interests include home device interaction and accessibility.

Auejin Ham received a B.S. degree in Interdisciplinary Engineering from DGIST. He is currently pursuing a Master's degree at the School of Computing, KAIST. His current research interests include multi-modal interaction devices and techniques, data-driven interface design, and computational interaction.

Geehyuk Lee received BS and MS degrees in Physics from KAIST and a PhD degree in Electrical Engineering from University of Pennsylvania. He is currently a professor in the School of Computing, KAIST. His research interests include interaction devices and techniques for smart information devices, wearable computers, and smart environments.