Relevance and Applicability of Hardware-independent Pointing Transfer Functions

Raiza Hanada, Damien Masson, Géry Casiez, Mathieu Nancel, Sylvain Malacria

Relevance and Applicability of Hardware-independent Pointing Transfer Functions

Raiza Hanada
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL
Lille, France
raiza.hanada@gmail.com

Damien Masson
Cheriton School of Computer Science, University of Waterloo
Waterloo, Canada
dmasson@uwaterloo.ca

Géry Casiez*
Univ. Lille, CNRS, Inria, Centrale Lille, UMR 9189 CRIStAL
Lille, France
gery.casiez@univ-lille.fr

Mathieu Nancel
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL
Lille, France
mathieu.nancel@inria.fr

Sylvain Malacria
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL
Lille, France
sylvain.malacria@inria.fr

ABSTRACT

Pointing transfer functions remain predominantly expressed in pixels per input counts, which can generate different visual pointer behaviors with different input and output devices; we show in a first controlled experiment that even small hardware differences impact pointing performance with functions defined in this manner. We also demonstrate the applicability of "hardware-independent" transfer functions defined in physical units. We explore two methods to maintain hardware-independent pointer performance in operating systems that require hardware-*dependent* definitions: scaling them to the resolutions of the input and output devices, or selecting the OS acceleration setting that produces the closest visual behavior. In a second controlled experiment, we adapted a baseline function to different screen and mouse resolutions using both methods, and the resulting functions provided equivalent performance. Lastly, we provide a tool to calculate equivalent transfer functions between hardware setups, allowing users to match pointer behavior with different devices, and researchers to tune and replicate experiment conditions. Our work emphasizes, and hopefully facilitates, the idea that operating systems should have the capability to formulate pointing transfer functions in physical units, and to adjust them automatically to hardware setups.

CCS CONCEPTS

• **Human-centered computing → Pointing**.

KEYWORDS

transfer function, computer mouse, pointing task

*Also with Institut Universitaire de France.

1 INTRODUCTION

Pointing using a computer mouse or a finger on a touchpad is one of the most common tasks on desktop and laptop computers. Their relative displacements are converted to cursor translations through a transfer function that defines the relationship between the motor input speed and the on-screen cursor speed. These functions can have a dramatic impact on pointing performance and usability [8], yet their design remains seldom documented, especially in operating systems (OSs) [7]. More importantly, major OSs define their transfer functions using virtual units like pixels and mouse counts, rather than physical units of speed and distance. The "same" pointing transfer function can therefore produce different observable pointer behaviors when used with input or output devices with different resolutions.

Thus, even with standard devices like mice and touchpads, the choice of a transfer function is important, and not always a matter of maintaining default settings. As of today, the resolutions of commercial mice can span from 400 to 16,000 counts per inch (CPI) and above[1], so the same physical movement can be sensed as drastically different values. Displays can pack 70 to more than 200 dots per inch (DPI)[2], so a cursor displacement expressed in pixels can look very different from one screen to another. In effect, a user may experience different pointer behaviors when switching from one computer setup to another, even when they are configured to the same "acceleration" setting. Many end-users are facing this situation, either because they frequently interact with multiple computers, switch between OSs on the same computer, or simply use several monitors with different pixel densities. They can sometimes adapt to these environments with changing constrains, but within some limits that remain to be documented in the context of pointing transfer functions.

The effect of input and output resolution on the pointing task, which remains to be quantified in terms of performance and usability, does not only impact end-users but also the HCI research. Experiment descriptions too seldom report these functions accurately, nor input and output resolutions, which puts their replicability, their

[1] https://www.titanwolf-gaming.de/admiral-1
[2] Sometimes also referred as pixels per inch (PPI).

design, and possibly some of their findings into question. For instance, two distinct pointing studies using *e.g.* "the default transfer function of Windows 10" as baselines can end up comparing very different pointer behaviors, and infer conclusions about candidate pointing mechanisms based on inconsistent or suboptimal settings that most users would not use by default. Similarly, two studies observing users' adoption of mouse vs. keyboard-based command selection in linear menus, but "silently" using different pointing transfer functions, can yield different results and reach different conclusions since the cost of selecting a command using the mouse will be different in each [26, 38]. In other words, the combination of transfer function and hardware can be a confounding variable in an experiment. Worse, failing to precisely report this information prevents reviewers and practitioners from assessing the possibility that these conditions were unrealistic or inconsistent in the first place.

We argue that pointing transfer function settings should have the ability to behave consistently across hardware setups, so users do not need to adjust to new environments. We also argue that baseline conditions in controlled experiments should depict interface behaviors that could realistically be expected in real setups; that includes pointing transfer functions, which should be reported with enough details to allow for replication across studies even when the exact same hardware is not available. A solution already proposed in previous work [7] is to systematically report transfer functions independently from hardware resolutions, *i.e.* using physical units. However, while feasible in research contexts with dedicated libraries, or by using input and output hardware with the exact same resolutions, this is not trivial in daily use: a user switching from a 100 to a 200 DPI display might not be able (or willing) to find the exact equivalent function among the available settings in their OS.

We here define *hardware-independent* pointing transfer functions as functions whose domain and codomain (input and output) are expressed in physical units, e.g. centimeters per second, radians per milliseconds, etc. Such formulations[3] are "independent of hardware" in that they describe the consistently observable behavior of a cursor regardless of the system, devices and settings on which they are implemented. By opposition, *hardware-dependent* pointing transfer functions have at least one of their input or output domains expressed in so-called virtual units, typically counts or pixels, whose physical expression requires a conversion factor (e.g. the display's pixel density, or the input device's polling rate).

In this paper, we investigate the relevance and applicability of such *hardware-independent* pointing transfer functions. We first examine the effects of input and output hardware characteristics on the performance and usability of common transfer functions. We also demonstrate the feasibility of using functions expressed in physical units, with or without the means to impose a custom function to the OS. In a first study, we show that even small variations in hardware resolutions can significantly impact user performance when acquiring targets using the "same" pointing transfer function. More precisely, we report the effects of using common transfer function settings in two realistic combinations of input and output resolutions, and show that performance and subjective usability vary in each hardware setup – and follow distinctly different trends – even with small differences in resolutions. In a second study, we investigate whether transfer function behaviors can be replicated across different input and output resolutions, simply by scaling them for equivalence in physical units. Finally, we present two web applications that allow users and researchers to replicate a transfer function from one environment to another, and simplify the implementation and comparison of hardware-independent pointing transfer functions.

Our work strengthens the argument in favor of expressing pointing transfer functions in physical units, contributes to a better understanding of the consequences of selecting "default" functions as baselines in controlled experiments, and presents solutions for designing and selecting equivalent transfer functions across different hardware setups.

2 RELATED WORK

2.1 How pointing transfer functions work

Pointing transfer functions on modern OSs all work the same way: a physical motion performed by the user is sensed by an input device (typically moving a mouse controller on a desk), which measures raw relative displacements in counts and sends them to the computer at a given (capped) frequency. From these input signals, the computer calculates a relative displacement of the cursor in pixels using a predefined function. There are three ways, traditionally, to define these functions.

First, the relationship between input and output displacements can be a constant multiplier, also called the control-to-display (CD) gain. This CD gain is often defined in display pixels per input count, so two mouse movements of the same distance and using the same gain value, but using mice with different CPI, will result in different cursor displacements. This may appear counter-productive, as computer mice with higher resolutions should measure hand movements with greater precision [35], not speed the cursor up. Similarly, for the same distance in pixels, a display with a higher pixel density will result in visually smaller cursor displacements compared to a display with a lower density.

Second, the CD gain can be a function of the input displacement's amplitude. In the HCI literature, such "gain functions" describe the CD gain as a function of input speed, typically favoring low gains at low speeds and high gains at high speeds [8, 27] in order to boost both accuracy and speed. Note however that most OSs define their gains as a function of input *displacement*, not speed; gain is then a function of the number of measured counts within a certain time frame. This brings input rate into play: most computer mice send events at up to 125 Hz, but this rate can increase up to 1000 Hz for gaming mice. For a given (spatial) input resolution and movement of the device, a higher input frequency results in more input events with fewer reported counts each. If the transfer

[3]Note that there is an important semantic distinction to be made between the *formulation* of a transfer function and its *implementation*. The hardware-*independent* formulation of a function requires an implementation that is aware of all the relevant conversion factors and resolutions to convert, e.g., the sensed input counts into the right amplitude of pixel displacement; in other terms, its implementation needs to be "hardware-aware". Conversely, a hardware-*dependent* formulation, i.e. expressed in counts and pixels, can be implemented as-is without knowledge ("agnostic") of the hardware – but result in different behaviors. The adjectives "hardware-(in)dependent" in this paper relate to *formulations*, not implementations.

function is nonlinear, which is often the case [7], these events will be assigned smaller gains from the function and result in smaller cursor displacements.

Third, and finally, the term "transfer function" is traditionally reserved for functions that define the cursor's displacement *directly* as a function of input displacements, without using a multiplicative gain. The two mechanisms are functionally equivalent as long as both functions are continuous: any gain function $G(x)$ can be assigned an equivalent function $F(x) = x \cdot G(x)$.

Casiez and Roussel systematically characterized the functions available in modern operating systems (Windows, macOS, Xorg) [7] and found that all systems, with the exception of macOS which sometimes takes into account the resolution of Apple input devices, use hardware-*dependent* CD-gain functions, *i.e.* defined in virtual units. Other systems assume at best that the input and output resolutions are constant and equal to 400 CPI for input and 96 DPI for output.

2.2 The importance of accurately reporting pointing conditions

In a nutshell, modern operating systems do not define pointing transfer functions independently of their input and output devices. Defining transfer functions in physical units has already been argued for in previous work, but even with such hardware-independent definitions, OSs still require functions defined in pixels and counts. In effect, they ignore input and output resolutions. For this reason, when running an experiment involving an indirect pointing device, it is critical to report all information necessary to replicate the experimental settings, including the operating system version, and the input and output devices' resolutions.

Despite the publication of a research paper specifically dedicated to highlighting this issue in 2011 [7], reporting these elements systematically in experiment descriptions is still not standard (see *e.g.* [1, 10, 19, 21, 23, 25, 31]) which can make their replication impossible. Fortunately, several projects provide enough details about the input and output devices' resolution, operating system, and mouse configuration settings to allow exact replication of a transfer function [9, 28, 30, 34, 39, 41]. Others [2, 22, 33] use the `libpointing` library developed to help define transfer functions and ease their replication. That being said, providing accurate OS and resolution details for replication might not always be sufficient for two main reasons.

First, comparing results between different studies remains complicated. Take as an example two studies comparing the performance of modern transfer functions. (1) In [7], significant differences were found in the performance of the default cursor acceleration settings of Xorg, macOS, and Windows 7. (2) In [8], the lowest setting of the Windows XP/Vista was found to be 14% slower than a subset of all remaining XP/Vista settings, including the default one. Because the functions of Windows 7 and XP/Vista are the same, combining the result of these studies could allow us to compare different Windows settings (1) with the macOS and Xorg settings (2) by using the default Windows setting as a reference. However (1) was run with a 400 CPI mouse and a 98.5 DPI monitor, and (2) with a 1600 CPI mouse and a 100 DPI monitor, so a direct comparison would be questionable.

Second, while some research projects accurately report pointing conditions, replicating an experiment setup exactly is not always feasible because some hardware may no longer be available, arduous to procure, or because some OSs are no longer running. Therefore, it is important to investigate the feasibility of interacting with transfer function defined in a hardware-independent way, and to generalize the reporting of hardware-independent functions themselves.

2.3 Hardware-independent pointing transfer functions for desktop computers

Casiez and Roussel's work [7] inform us that current OSs pointing transfer functions ignore the resolution of input and output devices, and suggest that different hardware configurations could result in different visual pointing behaviors. However, they do not document what amount of such difference would significantly impact user performance. Conversely, in their discussion section, Casiez and Roussel recommend describing custom nonlinear functions *"using figures or tables with physical units mapping the device speed to the cursor speed or CD gain."*; but it remains unclear if such a transfer function would yield similar performance in practice, when operated with hardware devices of different resolutions. For instance, the precision at which a mice movement is measured will significantly differ between a 400 DPI and a 1600 DPI devices, which could affect user performance even with the same function expressed in physical units.

To summarize, the following questions remain unanswered. First, we know that two different hardware configurations can result in different pointer behaviors, but does that difference impact user performance? Second, if user performance is impacted, how could hardware-independent transfer functions be implemented in practice, and would existing transfer functions yield similar performance when scaled to different hardware resolutions? We investigate these two research questions in the following sections.

3 STUDYING THE IMPACT OF HARDWARE CONFIGURATION ON POINTING PERFORMANCE

Current OSs express transfer functions using virtual units (*e.g.* pixels and counts), which can result in different pointer behaviors when input and output resolutions change. To better understand the consequences of this phenomenon, we designed a first controlled experiment in which we investigate how small differences in hardware configurations can impact user performance and subjective ratings in pointing tasks with different OS transfer functions.

3.1 Apparatus

We evaluated two hardware configurations, that we treated as a between-subject factor. The first group of participants (referred as group $800_{\text{cpi}}122_{\text{dpi}}$) used an 800 CPI optical USB mouse Logitech M-B58[4] running at 125 Hz, and the native 27 inches monitor of a late 2015 iMac Retina 5K running macOS Mojave, using a 60 Hz display with resolution set to 2880 × 1620 pixels (122 DPI). The second group (group $1000_{\text{cpi}}93_{\text{dpi}}$) used a 1000 CPI optical USB mouse (Dell

[4]Datasheet available in: https://pricecat.be/en-sg/p/logitech/930995-0600/mice-premium+optical+wheel+mouse+b58-44252.html

MS111-L[5]) also running at 125 Hz, and a 23.8 inches Dell monitor (Dell P24157H) running at 60 Hz with display resolution set to 1920 × 1080 (93 DPI) under Ubuntu 18.04. Both configurations ran the same experimental platform built with Node.js and running in Google Chrome v.72.

We used the `libpointing` toolkit with PointingServer[6] in order to retrieve raw mouse events and override the system transfer function with the ones tested in this experiment.

The experiment was run on a plywood desk bearing only the mouse, the monitor, a keyboard that was used to type the URL of the experiment, a printout of the qualitative survey, and a pen. Elements were disposed in a way that did not disturb participants' arm movements while selecting the targets. We did not use a mousepad.

3.2 Task and procedure

Participants were seated in front of a computer with a given Hardware configuration and invited to fill out a demographic questionnaire. They were then instructed to perform series of 2D reciprocal pointing tasks with 13 targets positioned along a circle, similar to the target geometry proposed in the ISO 9241-9 standard [17].

For each trial, participants had to select a target highlighted in green on a black background by moving the mouse cursor over it, then pressing the left mouse button for confirmation. Cursor displacement was controlled by different Transfer Functions during the experiment. Participants were instructed to select the targets as quickly and accurately as possible. If participants clicked outside of the target, it turned red for 200 ms to inform them of the error, but participants still had to select the target to carry on with the study. The next trial only started when the current target was successfully selected. Once selected, the color of the target changed back to the neutral color (gray) and the next target was highlighted (green). A progress bar was displayed at the bottom of the screen to indicate the overall study progress.

Note that participants were not instructed to avoid clutching, and we did not set up any apparatus [8] to count or characterize it. Clutching is an inherent component of indirect pointing that does not systematically correlate with poor performance [28], nor disappears with high gains. We posit that artificially filtering it out brings no clear benefit, and could hide some relevant aspects of the studied phenomenon.

3.3 Design

The study used a mixed-model design, with Hardware configuration as between-subject factor, and Block, target Width and distance (Dist), and Transfer Function as within-subject factors. Hardware configuration was treated as a between-subject factor to maintain reasonable study duration while allowing us to vary task difficulty.

To gather performance data from a realistic set of pointing tasks, we varied the distance between targets Dist (8 and 24 cm) and their diameter Width (0.3 and 1.9 cm), resulting in four levels of IDs ranging from 2.38 to 6.34 bits. Dist and Width are varied for the sake of increasing external validity, their effect will not be analyzed in detail.

Figure 1: Operational system settings compared in the study. MS Windows 10 v. 1909 (left) and OS X 10.6.7 (right) available acceleration settings on each system's mouse speed slider.

We tested 8 different Transfer Functions, four from Microsoft Windows 10 v. 1909 and four from macOS 10.6.7, as provided in the `libpointing` library[7]. In the remainder of this paper, we refer to the transfer function settings as ordinal integers corresponding to the position of the slider in the mouse configuration panel in their respective OS (Figure 1), with 0 (zero) corresponding to the default setting: the 6th tick in Windows 10 and the 4th tick in macOS. We consider this setting ordinal because, to our knowledge, there is no clear linear relationship between the function levels in these OSs. We obtain a range of WIN-5 to WIN+5 for Windows, and MAC-3 to MAC+6 for macOS. The fact that the range of slider positions are different between OSs bears no meaning in itself.

For our study we selected the default (WIN0, MAC0) and maximum (WIN+5, MAC+6) settings available in each OS. Our intention was to select one setting lower than default for each OS, but our preliminary tests with 800 CPI mice on a 122 DPI display revealed that the default macOS setting feels already quite slow with such a standard configuration, while lower settings of Windows remained usable. We therefore selected WIN-2, and WIN+2 as a halfway setting between default and maximum. For macOS, we selected MAC+2 and MAC+4 as equidistant steps between default (MAC0) and maximum (MAC+6)[8].

Note that OS and setting level are not distinct parameters of our study: the transfer functions of different OSs were designed separately, possibly with different mice and displays in mind, and what constitutes "default", "minimum", "maximum", or the amplitude of a "tick", is not comparable across OSs. In most of this study's analyses, the combinations of OS × setting are treated as distinct, individual Transfer Function levels with no relation to each other.

Each combination of Transfer Function, Width, and Dist were repeated in 4 consecutive Blocks of 13 trials each, to reveal learning or fatigue. Participants could take breaks between blocks. After completing the task for a given Transfer Function, participants were invited to provide subjective ratings about their perceived *Precision*, *Speed*, *Fatigue*, and degree of *Control* on 7-point Likert scales. The ordering of Width and Dist combinations was

[5]https://pricecat.be/en-sg/p/dell/330-9456/mice-0799471741689-ms111-13416047.html

[6]PointingServer npm page: https://www.npmjs.com/package/pointingserver

[7]The eight transfer functions corresponded to the URIs windows:7?slider=-2, windows:7?slider=0, windows:7?slider=2, windows:7?slider=5, osx:mouse?setting=0.6875, osx:mouse?setting=1.00, osx:mouse?setting=2.00, osx:mouse?setting=3.00 in https://github.com/INRIA/libpointing. Note that the transfer function in Windows 10 v. 1909 is the same as in Windows 7.

[8]Note that unlike macOS transfer functions that are all non-linear, Windows provides both linear and non-linear transfer functions (depending on whether the "Enhance pointer precision" box is checked or not). In this paper, we systematically refer to the non-linear versions when referring to Windows transfer functions.

randomized. Presentation of Transfer Function was counterbalanced across participants using a Latin Square design. The study lasted approximately 45 minutes for each participant.

We had for each hardware configuration 8 (Transfer Function) × 2 (Dists) × 2 (Widths) × 4 (Blocks) × 13 (trials) = 1664 trials per participant.

3.4 Participants

We recruited 32 participants that we split in two groups of 16, each group assigned to one of two Hardware (mouse+screen) configurations. All participants were daily computer users, and had normal or corrected-to-normal vision. The participants were free to operate the pointing device using the hand they felt most comfortable with.

Participants of the first group (4 female, 12 male) were 31.6 years old on average (24 to 43, SD=5.98). Five reported using a touchpad as main pointing device, eleven declared using mice more often. Two reported using Xorg systems more often, four reported using Windows systems more often, and ten declared using macOS more.

In the second group (2 female, 14 male), the participants were 23.4 years old on average (21 to 40, SD=4.83). Only one participant declared using a touchpad as main pointing device, the remaining fifteen reported using mice more often. Five participants declared using Xorg systems more, nine declared using Windows systems more, and two declared using macOS more.

3.5 Results

3.5.1 Data pre-processing. Our main dependent measures were *Error* rate, measured as the proportion of trials wherein the target was not correctly selected on the first try (regardless of later attempts); the Movement Time (*MT*) elapsed between the appearance of the target and its successful selection; and the number of target *Re-entries*, *i.e.* the number of times the cursor entered the target minus one [24]. Note that for *MT* we did not follow the ISO 9241-9 analysis recommendations regarding throughput as a performance metric. Olafsdottir et al. [29] documented issues of replicability and invariance with the classic throughput formulation, and recent work [14] criticized the arbitrary nature of its 4% error assumption.

In the following, *Error* rates are summarized using arithmetic means; *MT* is summarized using geometric means, as recommended for pointing studies and skewed datasets in general [37]; *Re-entry* was also right-skewed, so unlike in [24], we summarized it as geometric means. For *Error* and *MT*, statistical analyzes were performed using mixed-effect models with Block, Transfer Function, and Hardware as factors, and Participant treated as a random factor using the REML procedure of the SAS JMP package. For *Re-entry* and *Error rate*, because the normality assumption of the residuals was violated (Shapiro-Wilk p<0.001), we used an Aligned Rank Transform [40] using the *ARTool* package in R[9] and post-hoc analyses were done using *ART-C* [11]. Other post-hoc tests are all-pairwise comparisons using Tukey's HSD tests when there are more than two levels, and t-tests otherwise.

In each hardware configuration group, we found a significant effect of Block on completion time *MT* ($F_{3,45} = 11.9, p < 0.0001$ in the $800_{\text{cpi}}122_{\text{dpi}}$ group, $F_{3,45} = 21.6, p < 0.0001$ in the $1000_{\text{cpi}}93_{\text{dpi}}$ group): *MT* in Block 0 (means 1034 and 1062 ms, respectively) are in both cases significantly different from the *MT* of Blocks 1, 2, and 3 (mean times 1021 ms and below in the $800_{\text{cpi}}122_{\text{dpi}}$ group, 1035 and below in the $1000_{\text{cpi}}93_{\text{dpi}}$ group). We found no significant effect of Block on *Error* rate. For that reason, we discard Block 0 from subsequent analyses.

3.5.2 Errors. Participants made selection errors in 4.8% of trials in the $800_{\text{cpi}}122_{\text{dpi}}$ group, and 6.7% in the $1000_{\text{cpi}}93_{\text{dpi}}$ group. We found no effect of Hardware on *Error* rate, but a significant effect of Transfer Function in the $1000_{\text{cpi}}93_{\text{dpi}}$ group ($F_{7,210} = 6.8$, $p < 0.0001$): the function WIN+5 caused significantly more errors overall (mean 10.6%) than any other (means 6.9% and below). There was no interaction effect.

3.5.3 Movement time. Conversely, we found that *MT* was significantly affected by Hardware ($F_{1,208.1} = 18.4, p < 0.0001$), Transfer Function ($F_{7,207.9} = 11.1, p < 0.0001$), and the interaction between Hardware and Transfer Function ($F_{7,207.9} = 14.4, p < 0.0001$). Participants in the $800_{\text{cpi}}122_{\text{dpi}}$ group (mean 961 ms) were significantly faster overall than in the $1000_{\text{cpi}}93_{\text{dpi}}$ group (mean 1080 ms). Considering Transfer Functions separately, Hardware had a significant effect on *MT* for MAC+4 ($p < .05$, simple effect size 74 ms), WIN0 ($p < .01$, 96 ms), MAC+6 ($p < .0001$, 182 ms), WIN+2 ($p < .0001$, 305 ms), and WIN+5 ($p < .0001$, 338 ms).

Considering Hardware groups separately, we observe very distinct patterns in *MT* performance (Figure 2).

In the $800_{\text{cpi}}122_{\text{dpi}}$ group, higher settings in each OS had a distinct advantage: MAC0 was significantly slowest (mean 1140 ms), and MAC+2 and WIN-2 (means 1038 ms) were significantly slower than WIN+5 (965 ms), WIN0 (962 ms), and WIN+2 (949 ms). MAC+4 (1026 ms) and MAC+6 (1020 ms) were in between, and significantly different only from MAC0 on one end and WIN+2 on the other.

In the $1000_{\text{cpi}}93_{\text{dpi}}$ group, it is the lower settings of each OS that obtained better performance: MAC+6 (1096 ms) and WIN+5 (1093 ms) were significantly slower than MAC0 (1016 ms), MAC+2 (1011 ms), WIN-2 (971 ms), and WIN0 (954 ms). WIN+2 (1039 ms) and MAC+4 (1031 ms) were in between, only slower (significantly) than WIN0.

This trend is confirmed by the very similar results we obtain when we analyze macOS and Windows separately. In the $800_{\text{cpi}}122_{\text{dpi}}$ condition MAC0 (mean 1140 ms) is significantly slower than all other MAC settings (means between 1020 and 1038 ms) and WIN-2 (1038 ms) is significantly slower than all other WIN settings (means between 949 and 965 ms). In the $1000_{\text{cpi}}93_{\text{dpi}}$ condition, MAC+6 is significantly slower than all other MAC settings (1011 to 1031 ms), and WIN+5 (1093 ms) and WIN+2 (1039 ms) are significantly slower than WIN-2 (971 ms) and WIN0 (954 ms).

3.5.4 Target re-entry. Both Hardware ($F_{1,30} = 23.4, p < 0.0001$) and Transfer Function ($F_{7,210} = 25.27, p < 0.0001$) significantly affected the number of target *Re-entry*, with an interaction effect between Transfer Function and Hardware ($F_{7,210} = 2.34, p < .05$). The functions WIN+5 and WIN+2 (mean 1.16 in both cases) caused significantly more *Re-entries* than all other functions (means 1.07 to 1.11) except WIN0 (mean 1.15). Similarly, the $800_{\text{cpi}}122_{\text{dpi}}$ group saw significantly fewer *Re-entries* than the $1000_{\text{cpi}}93_{\text{dpi}}$ group (means 1.09 vs. 1.14).

[9] https://cran.r-project.org/web/packages/ARTool/

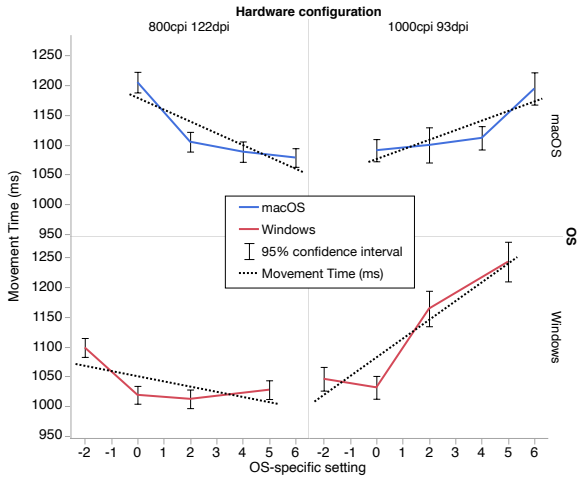

Figure 2: Movement Time without errors (Y, non-zeroed) as a function of OS-specific transfer function settings (X), for each hardware configuration (top) and OS (right). Error bars represent 95% confidence intervals, and the trend lines are for illustration purposes. Note the upward trend in the $800_{cpi}122_{dpi}$ group and downward trend in the $1000_{cpi}93_{dpi}$ group.

3.6 Subjective feedback

We treated participants' responses to our Likert-scale questionnaires as ordinal data, and assessed their similarity across Hardware and Transfer Functions using Pearson's chi-squared test. In all that follows, higher scores are better: faster for perceived *Speed*, more precise for *Precision*, *less* of a cause for *Fatigue*, and providing a better sense of *Control*.

Perceived *Precision* was significantly affected by Transfer Function ($\chi^2 = 73.9, p < 0.01$) with WIN+5 receiving more negative ratings overall (mean ratings 2.3 vs. [3.3-4.2]), and by Hardware × Transfer Function ($\chi^2 = 125.3, p < 0.01$) in which WIN+5 did slightly worse in $1000_{cpi}93_{dpi}$ (2.1) than in $800_{cpi}122_{dpi}$ (2.4), and MAC0 and WIN-2 fared better in $1000_{cpi}93_{dpi}$ (resp. 4.7 and 4.3) than in $800_{cpi}122_{dpi}$ (3.6 each).

Perceived *Speed* was significantly affected by Hardware ($\chi^2 = 27.6, p < 0.0001$), Transfer Function ($\chi^2 = 74, p < 0.01$), and Hardware × Transfer Function ($\chi^2 = 160.8, p < 0.0001$). Pointing in the $800_{cpi}122_{dpi}$ group (mean 3) was perceived as slower than in the $1000_{cpi}93_{dpi}$ group (3.9); remember however that this was a between-subject factor, so this could boil down to group differences. Regarding Transfer Function, MAC0 received the lowest *Speed* ratings overall (2.1), and MAC+6 the highest (4.5), the other functions ranging [3-3.9]. Finally, the interaction effect can be explained by much lower *Speed* ratings in $800_{cpi}122_{dpi}$, in particular for MAC0 (mean 1), MAC+2 and WIN-2 (2.1 each), than in $1000_{cpi}93_{dpi}$ (minimum mean 3.3).

Reported *Fatigue* was significantly affected only by Hardware × Transfer Function ($\chi^2 = 141.1, p < 0.001$). MAC0 and MAC+2 were felt to cause more *Fatigue* in $800_{cpi}122_{dpi}$ (mean 1.9 each) than in $1000_{cpi}93_{dpi}$ (resp. 3.3 and 3.1), while WIN+2 caused more *Fatigue* in $1000_{cpi}93_{dpi}$ (2.6) than in $800_{cpi}122_{dpi}$ (3.3).

Finally, sense of *Control* was significantly affected by Hardware ($\chi^2 = 13, p < 0.05$, mean 3 for $800_{cpi}122_{dpi}$ vs. 3.4 for $1000_{cpi}93_{dpi}$) and by Hardware × Transfer Function ($\chi^2 = 118.9, p < 0.05$), in which MAC0 and WIN-2 induced a greater sense of *Control* in $1000_{cpi}93_{dpi}$ (resp. 4.3 and 4.4) than in $800_{cpi}122_{dpi}$ (resp. 2.9 and 3.1). Again, main effects of Hardware need to be considered with a pinch of salt, considering that it was a between-subject factor.

Overall these subjective reports do not diverge from our objective reports: *Control* and perceived *Precision* follow *Error* rates, and perceived *Speed* is a good indicator of *MT*.

3.6.1 Discussion. We compared the performance of the same set of OS pointing function settings on two hardware configurations, with the same input and output frequencies but different input and output resolutions. The two input resolutions, 800 and 1000 CPI, are very common in commercial mice yet rather close to each other; the 93 and 122 DPI display resolutions arguably do not differ much either. Yet we found that these changes do affect performance when using hardware-dependent transfer functions, as the hardware configuration had a significant effect on *MT* for 5 tested transfer functions out of 8.

For Windows functions, the settings with steady movement time are in the range [WIN0, WIN+5] for the $800_{cpi}122_{dpi}$ resolutions, and in the range [WIN-2, WIN-0] for the $1000_{cpi}93_{dpi}$ resolutions. For macOS functions, we observe a similar trend, with the range [MAC+2, MAC+6] providing steady performance in the $800_{cpi}122_{dpi}$ resolutions, and the range [MAC0, MAC+4] in the $1000_{cpi}93_{dpi}$ resolutions. Considering how transfer functions work on current systems, it can appear surprising that the best range of settings for the $1000_{cpi}93_{dpi}$ resolutions is lower than the ones for the $800_{cpi}122_{dpi}$ resolutions: the same movement will generate input events with more counts with 1000 DPI than with 800 DPI, but that phenomenon is counterbalanced by a reversed order in pixel densities. We can infer that this boils down to a stronger impact of the difference in pixel densities (31%) than of mouse resolutions (25%). Finally, the higher error rate and number of target re-entries obtained for WIN+5 can be explained by the relatively high gain used at low speed (around 5 pixels/count), possibly making precise movements more difficult.

4 CONVERTING AND USING HARDWARE-INDEPENDENT TRANSFER FUNCTIONS

The previous study confirms that input and output hardware resolutions can affect the pointing performance of a given OS acceleration setting, because transfer functions are expressed in mouse counts and display pixels rather than physical units. This further motivates the use of hardware-independent transfer functions. Such a transfer function describes the speed components of a cursor's behavior in physical units, *i.e.* the amount of visual displacement of the cursor as a function of the physical motion (speed) of the user's limb(s). Reporting such a function from an existing setup requires four main elements:

The hardware-dependent transfer function F_H or gain function G_H Both usually use as input the unit given by the input device: 'counts' for mice and touchpads, pixels for touchscreens, radians [27] or quaternions [15] for rotation-based devices, etc. The output of F_H is typically screen pixels, or pixels/input-unit for G_H. These functions are typically expressed as lookup tables with linear interpolation (see *e.g.* [7, 22]), but can also be fully algebraic (*e.g.* [27]). Obtaining a precise description of the OS's function can be challenging, but tools [7] and methods [32] exist to reverse-engineer them. For classic OSs, the `libpointing` library[10] documents the functions corresponding to the available user settings. Converting between F and G is done trivially by dividing or multiplying the function's output by its input: $F(x) = G(x) \cdot x$.

The input resolution R_i. It expresses the relationship between input units and their physical equivalent. For mice it is for instance expressed in counts per inch (CPI). Some devices have variable CPIs that can be adjusted with additional software, or even on the device itself. In translation-based input, we recommend using counts per millimeters (CPMM), with 1 CPMM = 25.4 CPI.

The input period T_i (or the time differential between each input event). This is necessary to express the user's motion as a speed, typically in milliseconds.

The output's pixel density R_o. This is the output equivalent to input resolution[11] and is expressed in dots (pixels) per distance units, often per inch (DPI). We recommend using dots per millimeters (DPMM), with 1 DPMM = 25.4 DPI.

The hardware-independent expression $F_{\not H}$ of an algebraic transfer function F_H used in a given setup (R_i, T_i, R_o) is

$$F_{\not H}(x) = \frac{F_H(x \cdot R_i \cdot T_i)}{R_o} \tag{1}$$

with x expressed in physical units of speed and $F_{\not H}(x)$ in physical units of distance. When F_H is expressed as a lookup table of the form `{input: output}`, the input values need to be divided by $R_i \cdot T_i$ and the output values by R_o to obtain the equivalent hardware-independent table.

The opposite process, *i.e.* expressing a hardware-independent function $F_{\not H}$ in terms that can be interpreted by a hardware-*dependent* system with a given setup (R_i, T_i, R_o), is

$$F_H(x) = F_{\not H}\left(\frac{x}{R_i \cdot T_i}\right) \cdot R_o \tag{2}$$

with x expressed *e.g.* in mouse counts and $F_H(x)$ in pixels. With a lookup table, input and output need to be multiplied by respectively $R_i \cdot T_i$ and R_o.

5 INVESTIGATING THE FEASIBILITY OF HARDWARE INDEPENDENT TRANSFER FUNCTIONS

In this second study, we investigate whether expressing a baseline transfer function in physical units makes it behave consistently across different input and output configurations, and in doing so we explore two methods to implement hardware-independent functions within hardware-*dependent* systems.

5.1 Apparatus

We used a Logitech G9 gaming optical USB mouse[12] running at 125 Hz and with resolutions ranging from 400 to 3200 CPI, controlled by a dedicated software[13]. We used the same iMac Retina 5K 27" monitor as in the previous study, but we used two resolutions, 122 DPI (2880 × 1620 pixels) and 87 DPI (2048 × 1152), controlled using the Display settings of macOS. The experimental platform was the same as the one used in the first study, using the `libpointing` toolkit [7] to get the raw mouse events and control the system's transfer functions.

5.2 Method

As a baseline transfer function, we chose WIN+2 used with a 800-cpi mouse and a 122-dpi monitor because it provided among the best pointing performance and subjective ratings in the previous study. Using Equation (1), we produced a hardware-independent version of this function as it behaved in the $800_{\text{cpi}}122_{\text{dpi}}$ setup. For hardware configurations, we used the combinations of {400, 800, 1600, 3200} CPI as input, and {87, 122} DPI as output. For each CPI×DPI combination, we used two distinct Methods to implement our baseline in macOS.

5.2.1 Scaled. The first Method, named scaled, consists in directly converting the baseline transfer function into the input and output units of the target hardware setup, using Equation (2) with the baseline function and target input and output resolutions. Thus, transfer functions generated using the scaled method should behave exactly the same as the baseline under all hardware conditions (Figure. 3).

We label these transfer functions as $\text{X}_{\text{CPI}}\text{Y}_{\text{DPI}}$SCALED.

5.2.2 Closest. The second Method, named closest, aims to test a more practical solution that can already be applied in everyday operating systems. It consists in selecting the OS built-in transfer function setting that would produce, from a user perspective and with the current CPI and DPI settings, the closest physical and visual behavior from the baseline (see Figure 3).

There exist, to our knowledge, no recognized method or metric to assess the "distance" between two pointing transfer functions in terms of usability or performance, so we decided to rely on human subjective assessments. For each hardware setup, the closest transfer function was identified as follows. We recruited four authors of this submission to act as subjective assessors. For each CPI×DPI combination used in this study, except 800cpi×122dpi which corresponds to our baseline, they were asked to assess which Windows non-linear acceleration setting (candidate function) "felt" closest to the baseline, by repeating the 2D reciprocal pointing tasks from the previous study. The candidate functions were tested alternatively with the baseline function to maintain a behavior reference. Assessors could switch back and forth between the baseline and the

[10] Libpointing github page: https://github.com/INRIA/libpointing

[11] Note that the term 'display resolution' is by convention the number of distinct pixels in each dimension that can be displayed on a display device.

[12] Datasheet available in: https://pricecat.be/en-sg/p/logitech/910-000175/mice-g9+laser+mouse-914257.html

[13] Logitech Gaming Software v. 9.02.22 https://support.logi.com/hc/en-gb/articles/360025298053-Logitech-Gaming-Software

currently tested function, as well as between tested functions. Once confident about the choice of the closest setting to the baseline for a given CPI×DPI combination, assessors reported it then moved on to the next CPI×DPI configuration.

Finally, we aggregated the selections of all assessors, which were highly consistent. More precisely, the four assessors chose the same candidate function for all CPI×DPI combinations, except 800_87 wherein one assessor selected WIN-1 instead of the majority WIN0, and 3200_87 wherein one assessor chose WIN-5 instead of the majority WIN-4, resulting in an overall Kappa-Fleiss [12] agreement score of 0.82. Kappa-Fleiss scores between 0.81 and 1 can be considered as "almost perfect" [20], so we selected as CLOSEST functions the candidate functions that received the most votes in each CPI×DPI combination (see Table 1). We label these transfer functions as $X_{CPI}Y_{DPI}$CLOSEST.

We must report, however, that while in most cases the choice of a closest function boiled down to finding a setting that behaved mostly the same as the baseline, matching behaviors was at times impossible to achieve. This is notably the case for $400_{CPI}122_{DPI}$CLOSEST because Windows settings are capped to WIN+5, but WIN+5 still felt noticeably slower than the baseline in this setup. A hypothetical WIN+6 or WIN+7 setting may have felt closer to the baseline, but we did not want to stray from existing settings.

CPI (input)	DPI (output)	Setting selected as closest
400	122	WIN+5
400	87	WIN+5
800	122	WIN+2 (baseline)
800	87	WIN0
1600	122	WIN-2
1600	87	WIN-3
3200	122	WIN-4
3200	87	WIN-4

Table 1: Windows 10 acceleration settings selected as producing the closest cursor behavior to the baseline according our subjective assessors.

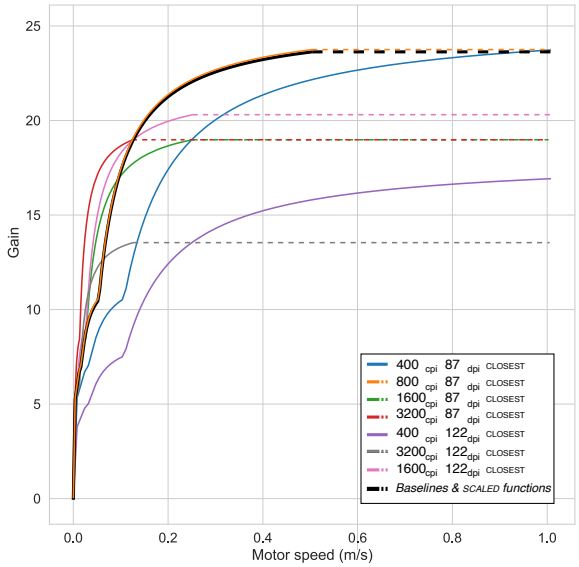

Figure 3: Gain functions used in the experiment, expressed in gain (Y, unitless) per physical unit (X, m/s). The functions of the form $A_{CPI}B_{DPI}$CLOSEST are represented in physical units using Equation (1) with $R_i = A/25.4$, $R_o = B/25.4$, and $T_i = 8$ ms. Both baseline conditions ($800_{CPI}122_{DPI}$SCALED, $800_{CPI}122_{DPI}$CLOSEST) and all SCALED functions are superimposed (black line). Transfer functions in `libpointing` are expressed in accordance with the HID standard, which encodes input signal in 8-bit integers and is therefore limited to 127 levels [3, p.61]. For higher input values, we maintained the highest gain value of each function (dotted lines).

5.3 Design

Independent variables were organized as follows. Participants performed all trials for a given DPI (screen resolution), then for the other; order of DPIs alternated between participants. For each DPI, the set of Transfer Functions obtained by crossing both Methods and all four CPIs was presented one after the other; the order of the 8 Transfer Functions was counterbalanced using a Latin Square design. Each combination of Transfer Function, Width, and Dist were repeated in 2 consecutive Blocks of 13 trials each. The ordering of Width and Dist combinations was randomized. Participants could take breaks between blocks.

After completing all trials for a given Transfer Function, participants were invited to report subjective ratings regarding their perceived *Precision*, *Speed*, *Fatigue*, and degree of *Control* on 7-point Likert scales.

Our design therefore had 2 display resolutions DPI (87, 122) × 4 mouse resolutions CPI (400, 800, 1600, 3200) × 2 Methods (SCALED, CLOSEST) × 2 distances (8 and 24 cm) × 2 widths (0.3 and 1.9 cm) × 2 blocks × 13 trials = 1664 trials per participant. The study lasted approximately 45 minutes for each participant.

5.4 Task

The task was similar to Study 1, except that the circular targets were presented in a 1D reciprocal pattern (left-right) rather than in a circular pattern. We chose a 1D task to reduce any source of variance related to the direction of movement [21] and increase the likelihood of observing significant differences between the functions, which would contradict our assumption that the functions perform similarly.

5.5 Participants

We recruited 16 unpaid participants (2 female, 14 male). The participants were 28.9 years old on average (SD=6.8, min=23, max=46). Three participants declared using a touchpad as main pointing device, ten declared using mice more often, and three declared using both. Participants' main operating systems were XOrg (6 participants), Windows (4), and macOS (6).

5.6 Results

5.6.1 Data pre-processing. Data was treated and analyzed the same way as in the previous study. Our main dependent measures were *Error* rate, measured as the proportion of trials wherein the target was not correctly selected on the first try (regardless of later attempts); the Movement Time (*MT*) elapsed between the appearance of the target and its successful selection; and the number of Target *Re-entries*. As in the previous study, *Error* rates are summarized using arithmetic means, and *MT* and *Re-entries* are summarized using geometric means [37]. Statistical analyzes were performed using mixed-effect models with Transfer Function and Block as factors, and Participant treated as a random factor using the REML procedure of the SAS JMP package. When assumptions of normality were violated (*i.e.* for *Re-entries* and *Error rate*), we used an Aligned Rank Transform [40] using the *ARTool* package in R. Unless specified otherwise, post-hoc tests are all-pairwise comparisons Tukey's HSD tests when there are more than two levels, and t-tests otherwise.

We found no significant effect of Block on *Error* rate (4.7% in Block 0, 4.1% in Block 1) nor on completion time *MT* (948 ms in Block 0, 943 ms in Block 1). We therefore kept both blocks in all subsequent analyses.

5.6.2 Error rate. We found no significant effect of the Transfer Function on *Error* rate. Trials with errors are ignored in the analysis of MT.

5.6.3 Movement time. We found no significant effect of Transfer Function on *MT*. Despite this, we note in Fig. 4 that the condition with the highest average *MT* was $122_{\text{CPI}}400_{\text{DPI}}$closest, *i.e.* the condition for which the subjective assessors could not choose a setting higher than WIN+5 when selecting the Windows function with the closest behavior to the baseline (see Table1 and Fig. 3).

In other terms, with the possible exception of the hardware condition in which the choice of Windows settings was capped, all of the scaled and closest functions were not significantly different from each other.

5.6.4 Target re-entry. The means of target *Re-entry* ranged from 1.07 to 1.18 per condition. We found no main effect of Block nor Transfer Function and no interaction effects.

5.7 Baseline comparison

Our previous analysis did not reveal any significant effect of Transfer Function on error rate or movement time. Interestingly, the largest difference for movement time, yet not significant, was in a closest condition wherein the ideal setting was (likely) higher than the highest available. This is a first hint that the closest and scaled methods both provide suitable surrogate functions to replicate an existing cursor behavior.

To quantify *how closely* these surrogate functions match the performance of a baseline function, we ran equivalence tests (TOST - two one-sided tests) between the baseline function and every other functions, with decreasing tolerances until a threshold was found. In these tests we aggregated both baseline conditions $800_{\text{CPI}}122_{\text{DPI}}$scaled and $800_{\text{CPI}}122_{\text{DPI}}$closest into a single reference dataset, rather than choose one artificially, since they are exactly the same function

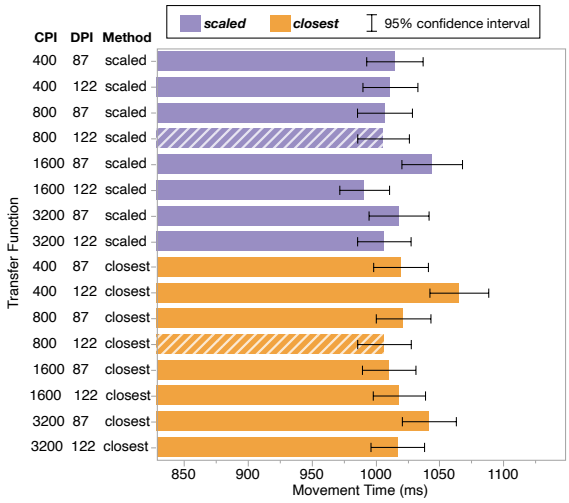

Figure 4: Overall Mean Movement Time (Y) for each Transfer Function (X). The baseline function is highlighted with hatching.

both in principle and in implementation. We found that with a minimum tolerance of 80 ms, all tested functions were equivalent to the baseline. Taking away $122_{\text{CPI}}400_{\text{DPI}}$closest, which was chosen by lack of a higher available setting, the minimum tolerance that makes all remaining functions equivalent to the baseline goes down to 60 ms, and to a minimum of 22 ms when we consider individual functions separately ($3200_{\text{CPI}}122_{\text{DPI}}$scaled in this case).

In other terms, the transfer functions obtained through the scaled and closest methods are equivalent to the chosen baseline with a 60-ms tolerance, when closest is not capped by available settings.

5.8 Subjective feedback

We treated participants' responses to our Likert-scale questionnaires as ordinal data, and assessed their similarity across Transfer Functions using Pearson's chi-squared test. None of our subjective measures (*i.e.* perceived *Speed*, *Precision*, *Fatigue*, and sense of *Control*) was significantly affected by Transfer Function.

5.9 Generalization of results

The scaling method simply consists in expressing a hardware-independent transfer function in virtual units (counts, pixels) corresponding to specific input and output resolutions, or in other words, in constructing a hardware-dependent version of a hardware-independent function. In agreement with our assumptions, no significant difference was found between the baseline and any of the scaled transfer functions. The equivalence tests confirmed and further quantified this finding. Based on these results and on how the technique works, we hypothesize that the scaling method could be generalized to other transfer functions (notably from macOS) as well as to other device resolutions, as long as it remains within reasonable limits that guarantee sufficient control to the user [8, 27].

CPI (Input)	DPI (Output)	Area between curves	Discrete Frechet	Arc-length	DTW	Assessors' choices
400	122	WIN+5	WIN+2	WIN+2	WIN+2	WIN+5
400	87	WIN+5	WIN 0	WIN 0	WIN 0	WIN+5
800	87	WIN 0	WIN 0	WIN 0	WIN 0	WIN 0
1600	122	WIN-1	WIN 0	WIN 0	WIN 0	WIN-2
1600	87	WIN-3	WIN 0	WIN 0	WIN 0	WIN-3
3200	122	WIN-4	WIN+3	WIN+3	WIN+3	WIN-4
3200	87	WIN-4	WIN 0	WIN 0	WIN 0	WIN-4

Table 2: Windows transfer functions with smallest distance to the baseline function when measured with different similarity measures. Cells are colored in green when the transfer functions are the same as the one selected by the assessors.

However, the generalizability of the CLOSEST method remains an open question. Average movement time was highest with the $400_{\text{CPI}}122_{\text{DPI}}$CLOSEST condition , and was also rated slower than all other functions by our participants. This result illustrates a conceptual limitation of this method: in extreme cases such as a low-resolution (400 CPI) mouse used on a high-resolution (122 DPI) display, typical OS settings need to stray far away from the use-cases they were initially designed for, and a suitable surrogate function might not be available among the available settings. This phenomenon, while non significant here, was however predictable, as assessors clearly expressed the limited suitability of the WIN+5 setting for this condition even though it was the best available.

On the other hand, no significant difference was found in average movement time or target re-entry for all CLOSEST transfer functions – which, as a reminder, were determined subjectively. In order to support a partial generalization of the CLOSEST method, we decided to investigate objective measures of transfer function similarity that could be used to replace the subjective assessors process that we followed prior to this experiment. We first chose four similarity measures[14]: Area between two curves in 2D space [18], Discrete Frechet distance [13], Arc-length distance between curves[6], and Dynamic Time Warping [4]. Then, for each combination of input and output resolutions, we computed the distance measured by each similarity measure between the baseline and all other transfer functions.

Table 2 reports the transfer function closest to the baseline for each similarity measure, as well as whether it agrees with the assessors' consensus. As can be seen, the area between two curves yields very similar results, with only one disagreement for $1600_{\text{CPI}}122_{\text{DPI}}$CLOSEST. As such, we tentatively propose that the area between two curves could be used as a default indicator of transfer function similarity, to quickly obtain functions that would not "feel" so different in behavior from other Windows 10 settings with different device resolutions. It is however too early to advertise this as a bulletproof selection method. We do not recommend to select a CLOSEST transfer function with a 2D-curve-area distance greater than 0.53. This value, somewhat arbitrary, is the difference between our baseline and the $400_{\text{CPI}}87_{\text{DPI}}$CLOSEST function in our study, *i.e.* the largest distance with a condition that was not found significantly different from the baseline. We cannot yet formally recommend to use this method for macOS transfer functions either, given that we did not test them in this study, and that the macOS function profiles are quite distinct from Window's.

6 HOMOGENEIZING POINTER BEHAVIOR ACROSS DIFFERENT HARDWARE CONFIGURATIONS

We built two web-based tools[15] to produce or suggest pointing transfer functions that yield similar pointer behaviors – and performance – across different hardware conditions. Each tool relies on one of the two methods investigated in the previous study, namely SCALED and CLOSEST, focusing on their specific use-cases as described in the following two scenarios.

6.1 Scaling tool: replicating HCI experimental conditions

We were first interested in situations where HCI researchers might want to tune and replicate conditions from previously conducted experiments, but using different hardware configurations.

Let's say that Alice, an HCI researcher, wants to replicate a pointing study initially conducted with a 1D reciprocal pointing task, but this time using a 2D pointing task. Unfortunately, Alice's lab does not own a display with the same pixel density as the display used in the initial study. She starts the *Scaling* tool and specifies in the leftmost part (Figure 5-Step 1) the source configuration, that is, the information relative to the hardware and transfer function that were used (and reported!) in the original study. Once the source configuration is specified, Alice fills the top-right part (Figure 5-Step 2) of the interface with the information relative to *her* hardware configuration. Then, by clicking the "Click to download the transfer function" button (Figure 5-Step 3), she downloads the .dat file of a transfer function that replicates the pointing behavior of the initial study, on her own hardware, ready to be used with the `libpointing` library by following the corresponding instructions[16].

The Scaling tool was implemented as a web page using a Node.js 12.6 server and a Python 3 script. The resulting .dat file contains a lookup table of the source transfer function expressed as counts and pixels in the new input and output resolutions, and computed using the same method as described in section 4. The Scaling tool can be used regardless of the source and destination hardware (input and output resolutions) and software (Operating System, linear or non-linear function) configurations. Note that this tool could also be used for

[14] We used the python library `similaritymeasures 0.4.3` [18].

[15] ns.inria.fr/loki/tftools/

[16] https://github.com/INRIA/libpointing/wiki/Custom-Functions

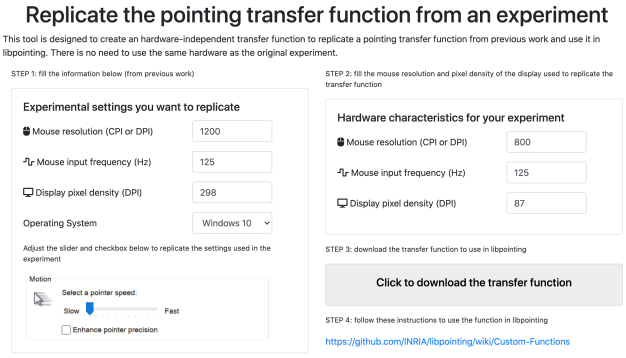

Figure 5: User interface of the Scaling tool. Users specify source hardware and software configuration (Step 1), as well as destination hardware configuration (Step 2). Scaling tool generates a transfer function that can be downloaded (Step 3) and used with libpointing to replicate the source pointing behavior with the new hardware.

other scenarios, for instance to set the exact same transfer function between two Windows computers operated by a single end-user, but that would require said end-users to implement a specific piece of software using the corresponding method[17] of the `libpointing` library, to bypass the default transfer function and apply the new one.

6.2 Closest tool: assisting end-users to set up several configurations

Using the Scaling tool is not always possible, for instance for people with no programming skills, or when they cannot or do not want to install a background software that would modify pointing behavior. Therefore, we built a second tool with end-users in mind, that can be used to configure two different computers with relatively similar pointing transfer functions, only using available OS pointer settings.

Bob is a middle-aged office worker who, because of the COVID-19 situation, is constrained to split his work time between his usual work station at the office, and a new one set at home under the circumstances. Each of these stations is equipped with a desktop computer running Microsoft Windows, a monitor, a keyboard, and a mouse controller. Unfortunately, the monitors and mouse controllers are of different models and resolutions. Therefore, Bob wants to change the pointer setting of his new home computer to obtain a cursor behavior similar to the one at work. He starts the *Closest* tool and specifies in the left part (Figure 6-Step 1) the required information about his office workstation: input and output device resolutions, and pointing setting of the computer. He then enters the information relative to the mouse and display for his home computer (Figure 6-Step 2), and possibly the Windows version if it differs. As a result, the *Closest* tool indicates which setting should be selected in his home OS to obtain a mouse behavior as similar as possible to his home computer's (Figure 6-Step 3).

[17] https://github.com/INRIA/libpointing/blob/master/pointing/transferfunctions/windows/winSystemPointerAcceleration.cpp#L90

Warning messages can be displayed if the configurations are too different.

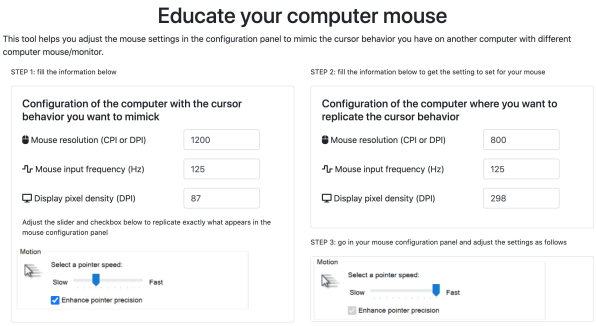

Figure 6: User interface of the Closest tool. Users specify source hardware and software configuration (Step 1), as well as destination hardware configuration (Step 2). Closest tool suggests which settings to select on the destination computer (Step 3).

The Closest tool was implemented using the same Node.js 12.6 and Python 3 technology as the Scaling tool. Suggested transfer functions correspond to functions with the lowest distance found using the area-between-two-curves metric with the specified input and output resolutions.

Two different types of warnings can be displayed. First, if the specified resolutions or transfer functions differ from the ones tested in our second experiment, users are warned that the suggested closest function has not been validated by human assessors. Second, if the difference between the initial and suggested functions is considered "too large", *i.e.* higher than 0.53 for the reason presented in section 5.9, users are warned that the function might yield a noticeably different pointing behavior.

7 DISCUSSION AND CONCLUSION

7.1 Summary of findings

Pointing transfer functions remain predominantly expressed in pixels per input counts, which can generate different visual pointer behaviors with different input and output devices. In this work, we investigate the relevance and applicability of "hardware-independent" pointing transfer functions. In a first study, we document the effects that even small differences in hardware resolutions can have on the performance and usability of common transfer functions, and confirm that hardware resolution impacts the range of transfer function settings that yield best performance. In a second study, we demonstrate the applicability of hardware-independent transfer functions defined in physical units, by adapting a baseline function to different hardware setups via two methods: scaling it to the new input and output resolutions, or selecting the OS acceleration setting that produces the closest visual behavior. The resulting functions provided equivalent performance when used with different screen and mouse resolutions. Finally, we describe two software tools, with accompanying use-cases, that can help researchers and end-users take advantage of hardware-independent transfer functions.

7.2 Experimental decisions

Several decisions had to be made during this project that impacted our studies' design. One of them lies in the design of our first study, which investigates how different hardware configurations impact performance in pointing tasks with different OS transfer functions. In order to ensure that the study was of acceptable duration while maintaining interesting and relevant conditions, we used a mixed-model design with hardware configuration as a between-subject factor, and ended up with an average age higher in one group than in the other (respectively 31.6 years vs. 23.4 years old on average). However, previous work suggests that age does not play a significant role on pointing performance in precuing tasks, *i.e.* when the location of the targets is known before visually identifying them [16], as was our case here. It is therefore unlikely that the observed effects of hardware be caused by age differences, especially since our findings confirm previous work.

7.3 Limitations

While we believe that the SCALED method can be trusted with most transfer function profiles and hardware resolutions, the CLOSEST method is strongly dependent on the available functions in the target OS. It also remains to be shown that it works as well when the source and target functions have different shapes, as do *e.g.* Windows and macOS functions. In effect, CLOSEST was only tested on a subset of available Windows 10 transfer functions, and on a limited number of input and output resolutions. In an attempt to generalize this result, we tested different similarity metrics and identified one that yields extremely close suggestions to our own subjective assessments. We tentatively use this metric in the corresponding software tool, but with warnings when suggestions do not match our tested conditions. Future work should assess how well this method works with macOS and Xorg transfer functions, across operating systems, and in more various settings than mouse and desktop. Further research should also be conducted to explore alternative similarity metrics that can represent usage and user perception more faithfully, *e.g.* balancing the relative frequencies of binned input speeds vs. their contribution to the overall movement after the function was applied.

Our results are valid for the Windows 10 transfer functions we tested, which have remained the same since 2011 [7] at the time of writing this article. We relied on [7]'s extensive details about the end-to-end calculation of transfer functions in different OSs. However, different or new OS architectures could introduce minute differences in the way raw mouse input is processed, or introduce radically different transfer functions, that would require further validation of this article's findings.

Mouse controllers remain one of the main pointing devices, but touchpads are also increasingly common. They are almost systematically equipped on laptop computers, and available as external input devices. Theoretically, pointing with touchpads works in a similar fashion as mice, with finger displacement on the surface being converted into count events sent to the system at a given frequency, and transformed through a dedicated gain function into pointer translations [7]. However, the input resolution of touchpads is not only impacted by the hardware resolution of the sensing surface, but also by how touch contacts are interpreted by the device, converted into blob displacements on its surface, to finally being converted to counts. While there is no reason to believe that our methods to adapt a baseline function to a different hardware setup would not scale to interaction on touchpads, future work should confirm this. Moreover, it is important to note that unlike mouse controllers, the physical size of a touchpad impacts its tracking range and as such, touchpads of different sizes may result in different pointing strategies from the users [5, 28].

Little research has been conducted on how users adjust to pointing transfer functions over long periods of time. Unlike purely perceptual alterations such as upside-down goggles [36] to which the human perceptual and motor systems can be shown to adapt, pointing transfer functions can impact the output's minimum and maximum achievable speed and accuracy (assuming bounded motor capabilities). We thus posit that there exist situations in which users cannot reach similar levels of pointing performance when switching between two transfer functions, regardless of how long they practice. We also note that, while some adaptation happens when switching between transfer functions, it is unclear whether long-term adaptation applies exclusively to performance: it could be, for example, that users "optimize" comfort and accuracy at the expense of selection time. In our first experiment and in previous work, performance plateaued after the first block, but investigating this capability in detail is an important future area of research.

7.4 Identifying input and output resolution

Another practical limitation of this work is the fact that it is sometimes difficult for end-users to rapidly determine the exact resolution and input frequency of an input device, including computer mice. This information is rarely written on the device itself, and it remains virtually impossible to determine it programmatically. Users may also install additional third-part drivers that change the mouse input resolution without system notification, making this information even more tedious to find. Future work should investigate methods to either automatically determine input resolution, or at least make this information easier to find for end-users.

7.5 Practical takeaway message

We encourage OS designers and developers to consider revisiting the management of transfer functions in current OSs. Transfer functions should be able to adapt to the characteristics of input and output devices, for example providing more precision when the mouse or screen allows for it, but without sacrificing the general behavior of the cursor. From a given slider position in the mouse settings panel, users should experience consistent cursor behavior from one computer to another. This would help switching from one input device to another, or getting consistent cursor behavior when using multiple monitors with different pixel densities. Note that automatic adjustment to any change in physical display resolution is neither always feasible, nor systematically desirable: imagine a projected display, with a transfer function set to the liking of its user. Now imagine that the projector has to be pushed forward or backward by 30 cm for some reason, scaling all distances and targets equally, but keeping them roughly as visible as they were before the move. It is then debatable that the cursor should behave

the same *with respect to the physical space* (the projector screen) as opposed to the virtual space (the relative movements of the cursor within the available display). Because such cases do exist we do not argue for systematic transfer function scaling, but for the availability of hardware-independent transfer functions. If the input or output resolutions cannot be determined automatically, they should appear as settings in the configuration panel. Ultimately, adjusting mouse, screen, or cursor acceleration settings should remain a personal preference rather than a way to deal with different hardware configurations that users may not fully understand.

Finally, we encourage computer mouse manufacturers to follow the HID specification that allows specifying the device resolution in the HID descriptor [3, p.37], to make it readily available to the OS. We conducted an informal review of a number of computer mice, which revealed that this information is seldom available. At least the device resolution should be easily accessible to end users, for instance printed clearly on the device's case to facilitate the manual configuration of the transfer function.

8 ACKNOWLEDGEMENTS

This work was partially supported by the Agence Nationale de la Recherche (Causality, ANR-18-CE33-0010-01) and by the LAI ULille Réapp.

REFERENCES

[1] David Ahlström and Martin Hitz. 2013. Revisiting PointAssist and Studying Effects of Control-Display Gain on Pointing Performance by Four-Year-Olds. In *Proceedings of the 12th International Conference on Interaction Design and Children* (New York, New York, USA) *(IDC '13)*. Association for Computing Machinery, New York, NY, USA, 257–260. https://doi.org/10.1145/2485760.2485792

[2] Axel Antoine, Sylvain Malacria, and Géry Casiez. 2018. Using High Frequency Accelerometer and Mouse to Compensate for End-to-End Latency in Indirect Interaction. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (Montreal QC, Canada) *(CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3173574.3174183

[3] Mike Bergman, Tom Peurach, Tom Schmidt, Steve McGowan, Jodi Crowe, Robert Dezmelyk, Remy Zimmermann, Mike Van Flandern, Bob Nathan, Mike Davis, and Joe Rayhawk. 2001. *Device class definition for human interface devices (HID)*. Version 1.11. USB Implementers' Forum. https://www.usb.org/sites/default/files/hid1_11.pdf

[4] Donald J. Berndt and James Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (AAAIWS'94)*. AAAI Press, Seattle, WA, 359–370.

[5] M. Camilleri, B. Chu, A. Ramesh, D. Odell, and D. Rempel. 2012. Indirect Touch Pointing with Desktop Computing: Effects of Trackpad Size and Input mapping on Performance, Posture, Discomfort, and Preference. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 56, 1 (2012), 1114–1118. https://doi.org/10.1177/1071181312561242 arXiv:https://doi.org/10.1177/1071181312561242

[6] J. Cao and J. Lin. 2008. A study on formulation of objective functions for determining material models. *International Journal of Mechanical Sciences* 50, 2 (2008), 193 – 204. https://doi.org/10.1016/j.ijmecsci.2007.07.003

[7] Géry Casiez and Nicolas Roussel. 2011. No More Bricolage!: Methods and Tools to Characterize, Replicate and Compare Pointing Transfer Functions. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Santa Barbara, California, USA) *(UIST '11)*. ACM, New York, NY, USA, 603–614. https://doi.org/10.1145/2047196.2047276

[8] Géry Casiez, Daniel Vogel, Ravin Balakrishnan, and Andy Cockburn. 2008. The Impact of Control-Display Gain on User Performance in Pointing Tasks. *Human–Computer Interaction* 23, 3 (2008), 215–250. https://doi.org/10.1080/07370020802278163

[9] Yuenkeen Cheong, Randa L. Shehab, and Chen Ling. 2013. Effects of age and psychomotor ability on kinematics of mouse-mediated aiming movement. *Ergonomics* 56, 6 (2013), 1006–1020. https://doi.org/10.1080/00140139.2013.781682

[10] A. Cockburn, D. Ahlström, and C. Gutwin. 2012. Understanding performance in touch selections: Tap, drag and radial pointing drag with finger, stylus and mouse. *International Journal of Human-Computer Studies* 70, 3 (2012), 218 – 233. https://doi.org/10.1016/j.ijhcs.2011.11.002

[11] Lisa A. Elkin, Matthew Kay, James J. Higgins, and Jacob O. Wobbrock. 2021. An Aligned Rank Transform Procedure for Multifactor Contrast Tests. arXiv:2102.11824 [stat.ME]

[12] Joseph L. Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological Bulletin* 76, 5 (1971), 378–382. https://doi.org/10.1037/h0031619 Place: US Publisher: American Psychological Association.

[13] M Maurice Fréchet. 1906. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Matematico di Palermo (1884-1940)* 22, 1 (1906), 1–72.

[14] Julien Gori, Olivier Rioul, and Yves Guiard. 2017. To Miss is Human: Information-Theoretic Rationale for Target Misses in Fitts' Law. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 260–264. https://doi.org/10.1145/3025453.3025660

[15] Faizan Haque, Mathieu Nancel, and Daniel Vogel. 2015. Myopoint: Pointing and Clicking Using Forearm Mounted Electromyography and Inertial Motion Sensors. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. ACM, New York, NY, USA, 3653–3656. https://doi.org/10.1145/2702123.2702133

[16] Morten Hertzum and Kasper Hornbæk. 2013. The Effect of Target Precuing on Pointing With Mouse and Touchpad. *International Journal of Human–Computer Interaction* 29, 5 (2013), 338–350. https://doi.org/10.1080/10447318.2012.711704

[17] ISO 9241-9 2000. *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 9: Requirements for non-keyboard input devices* (1 ed.). Technical Report. International Organization for Standardization, Geneva, CH.

[18] Charles F Jekel, Gerhard Venter, Martin P Venter, Nielen Stander, and Raphael T Haftka. 2019. Similarity measures for identifying material parameters from hysteresis loops using inverse analysis. *International Journal of Material Forming* 12, 3 (2019), 355–378.

[19] Heidi Horstmann Koester, Edmund LoPresti, and Richard C. Simpson. 2005. Toward Goldilocks' Pointing Device: Determining a "Just Right" Gain Setting for Users with Physical Impairments. In *Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility* (Baltimore, MD, USA) *(Assets '05)*. Association for Computing Machinery, New York, NY, USA, 84–89. https://doi.org/10.1145/1090785.1090802

[20] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33, 1 (1977), 159–174. http://www.jstor.org/stable/2529310

[21] Byungjoo Lee and Hyunwoo Bang. 2013. A kinematic analysis of directional effects on mouse control. *Ergonomics* 56, 11 (2013), 1754–1765. https://doi.org/10.1080/00140139.2013.835074

[22] Byungjoo Lee, Mathieu Nancel, Sunjun Kim, and Antti Oulasvirta. 2020. Auto-Gain: Gain Function Adaptation with Submovement Efficiency Optimization. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376244

[23] Ray F. Lin and Yi-Chien Tsai. 2015. The use of ballistic movement as an additional method to assess performance of computer mice. *International Journal of Industrial Ergonomics* 45 (2015), 71 – 81. https://doi.org/10.1016/j.ergon.2014.12.003

[24] I. Scott MacKenzie, Tatu Kauppinen, and Miika Silfverberg. 2001. Accuracy Measures for Evaluating Computer Pointing Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) *(CHI '01)*. Association for Computing Machinery, New York, NY, USA, 9–16. https://doi.org/10.1145/365024.365028

[25] Sylvain Malacria, Eric Lecolinet, and Yves Guiard. 2010. Clutch-Free Panning and Integrated Pan-Zoom Control on Touch-Sensitive Surfaces: The Cyclostar Approach. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2615–2624. https://doi.org/10.1145/1753326.1753724

[26] Sylvain Malacria, Joey Scarr, Andy Cockburn, Carl Gutwin, and Tovi Grossman. 2013. Skillometers: Reflective Widgets That Motivate and Help Users to Improve Performance. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology* (St. Andrews, Scotland, United Kingdom) *(UIST '13)*. Association for Computing Machinery, New York, NY, USA, 321–330. https://doi.org/10.1145/2501988.2501996

[27] Mathieu Nancel, Emmanuel Pietriga, Olivier Chapuis, and Michel Beaudouin-Lafon. 2015. Mid-Air Pointing on Ultra-Walls. *ACM Trans. Comput.-Hum. Interact.* 22, 5, Article 21 (Aug. 2015), 62 pages. https://doi.org/10.1145/2766448

[28] Mathieu Nancel, Daniel Vogel, and Edward Lank. 2015. Clutching Is Not (Necessarily) the Enemy. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems* (Seoul, Republic of Korea) *(CHI '15)*. Association for Computing Machinery, New York, NY, USA, 4199–4202. https://doi.org/10.1145/2702123.2702134

[29] Halla B. Olafsdottir, Yves Guiard, Olivier Rioul, and Simon T. Perrault. 2012. A New Test of Throughput Invariance in Fitts' Law: Role of the Intercept and of Jensen's Inequality. In *Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers* (Birmingham, United Kingdom) *(BCS-HCI '12)*. BCS Learning & Development Ltd., Swindon, GBR, 119–126.

[30] Y. H. Pang, E. R. Hoffmann, and R. S. Goonetilleke. 2019. Effects of Gain and Index of Difficulty on Mouse Movement Time and Fitts' Law. *IEEE Transactions on Human-Machine Systems* 49, 6 (2019), 684–691. https://doi.org/10.1109/THMS.2019.2931743

[31] Phillip T. Pasqual and Jacob O. Wobbrock. 2014. Mouse Pointing Endpoint Prediction Using Kinematic Template Matching. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. Association for Computing Machinery, New York, NY, USA, 743–752. https://doi.org/10.1145/2556288.2557406

[32] Philip Quinn, Sylvain Malacria, and Andy Cockburn. 2013. Touch Scrolling Transfer Functions. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2013)*. ACM Press, St. Andrews, United Kingdom, 61–70. https://doi.org/10.1145/2501988.2501995

[33] Sharan Ram, Anjan Mahadevan, Hadi Rahmat-Khah, Guiseppe Turini, and Justin G. Young. 2017. Effect of Control-Display Gain and Mapping and Use of Armrests on Accuracy in Temporally Limited Touchless Gestural Steering Tasks. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 61, 1 (2017), 380–384. https://doi.org/10.1177/1541931213601577

[34] Ridhan Riyal, Achyut D. Patel, Thilagan Murugesan, Giuseppe Turini, and Justin G. Young. 2015. Effect of control-display transfer function on pointing performance for a hand/finger based touchless gestural controls: a preliminary investigation. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 59, 1 (2015), 1085–1089. https://doi.org/10.1177/1541931215591155

[35] Nicolas Roussel, Géry Casiez, Jonathan Aceituno, and Daniel Vogel. 2012. Giving a Hand to the Eyes: Leveraging Input Accuracy for Subpixel Interaction. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (Cambridge, Massachusetts, USA) *(UIST '12)*. Association for Computing Machinery, New York, NY, USA, 351–358. https://doi.org/10.1145/2380116.2380162

[36] Pierre Sachse, Ursula Beermann, Markus Martini, Thomas Maran, Markus Domeier, and Marco R. Furtner. 2017. "The world is upside down" – The Innsbruck Goggle Experiments of Theodor Erismann (1883–1961) and Ivo Kohler (1915–1985). *Cortex* 92 (2017), 222–232. https://doi.org/10.1016/j.cortex.2017.04.014

[37] Jeff Sauro and James R. Lewis. 2010. Average Task Times in Usability Tests: What to Report?. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2347–2350. https://doi.org/10.1145/1753326.1753679

[38] S. Tak, P. Westendorp, and I. van Rooij. 2013. Satisficing and the Use of Keyboard Shortcuts: Being Good Enough Is Enough? *Interacting with Computers* 25, 5 (2013), 404–416.

[39] Yuntao Wang, Chun Yu, Yongqiang Qin, Dan Li, and Yuanchun Shi. 2013. Exploring the Effect of Display Size on Pointing Performance. In *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces* (St. Andrews, Scotland, United Kingdom) *(ITS '13)*. Association for Computing Machinery, New York, NY, USA, 389–392. https://doi.org/10.1145/2512349.2514911

[40] Jacob O. Wobbrock, Leah Findlater, Darren Gergle, and James J. Higgins. 2011. The Aligned Rank Transform for Nonparametric Factorial Analyses Using Only Anova Procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Vancouver, BC, Canada) *(CHI '11)*. Association for Computing Machinery, New York, NY, USA, 143–146. https://doi.org/10.1145/1978942.1978963

[41] Shota Yamanaka. 2019. Steering Performance with Error-Accepting Delays. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–9. https://doi.org/10.1145/3290605.3300800