Pointer Acceleration Evaluation in libinput

Peter Hutterer
Red Hat

Revision 1.0 - 2014-09-12

Abstract

This paper describes a userstudy performed to evaluate three different pointer acceleration methods for libinput. The analysis shows the difference between the methods is largely neglegible, with objective data being statistically significant in only a few cases and subjective data based on a participant questionnaire not meeting statistical significance.

1 Revision History

2014-09-16 - Revision 1.0 Initial publication

Contents

2 Introduction

A pointer acceleration method is a function mapping input speed of a device to cursor speed in pixels. The faster one moves the mouse, the further the cursor moves per "mickey" (a 1 device-unit movement). For example, an input delta of 1 may result in a 1 pixel movement, an input delta of 10 may result in a 30 pixel movement. All common windowing systems provide some pointer acceleration methods for mouse-like input devices, but the exact algorithm differs between the systems. [6]

libinput[1] is the input stack for some popular upcoming Wayland compositors and thus in charge of pointer acceleration on a Wayland-based desktop. A previous analysis of pointer acceleration in libinput [7] showed some potential areas for improvement for the pointer acceleration currently employed in version 0.4.0. To this effect, two new pointer acceleration methods were developed and evaluated in a userstudy.

The three pointer acceleration methods used in this study were nicknamed:

smooth: a shortening of the term "smooth and simple" used in its documentation [1], this method is used in libinput 0.5 as well as in the X.Org stack since ca 2008.

stretched: a modification of smooth with roughly the same profile, but the maximum acceleration is applied at a higher speed. This method, developed by Hans de Goede, was very promising in informal testing.

linear: a linear acceleration method with a roughly similar speed-to-acceleration profile as the first two. This method was developed to test if a simple function (effectively 3 lines of code) could achieve similar results, as the more complex "smooth" and "stretched" methods.

Figure 1 shows the different acceleration profiles of the three methods. As the graph shows, the profile is roughly identical and the main difference is how quickly the maximum acceleration factor is reached.

The input data expected by all three methods is in units/ms. Touchpad devices are normalised to 400 dpi, other devices are left a their native resolution. It is impossible to detect in software what resolution a generic mouse supports, so any acceleration method differs between devices. This is intended by the manufacturer, high-resolution devices are sold as "faster" for this reason.

[1] `http://www.freedesktop.org/wiki/Software/libinput/`

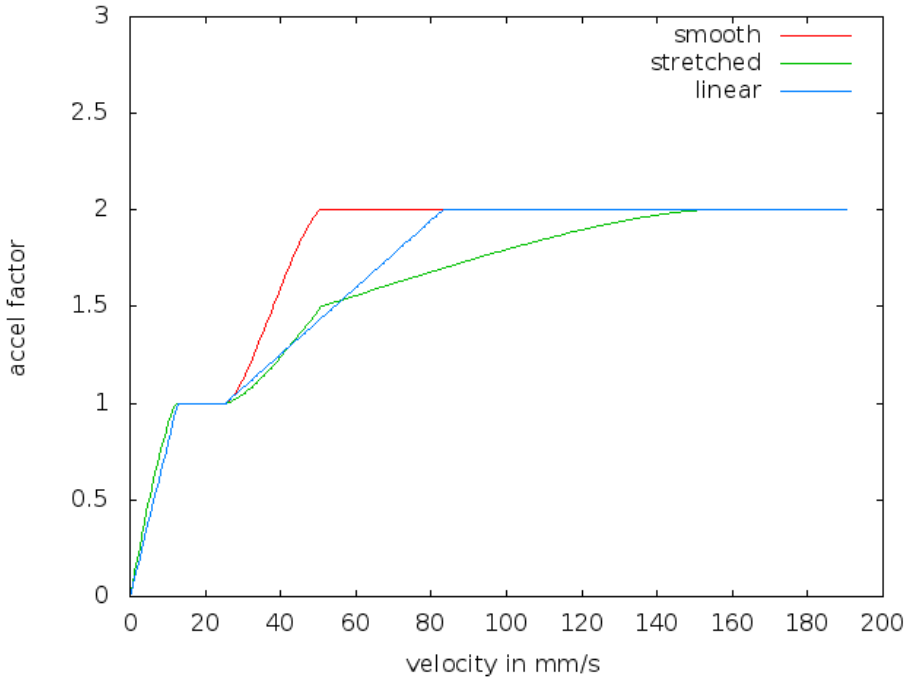

Figure 1: The three pointer acceleration methods

libinput employs additional effects such as delta softening [7] in addition to the pointer acceleration method. For this study, only the pointer acceleration method was modified.

3 Study description

For user evaluation, a tool was built on libinput (see Appendix B). The tool displays a full-screen white window with a round green target (see Figure 2). Instructions to participants were given via GTK dialog boxes, otherwise the study was unsupervised and self-guided. More information about the study was available on a website the participants were initially directed to. The website also provided the installation instructions and the upload form for the resulting data set.

The task required participants to click on a round target with a radius of 15, 30 and 45 pixels. Targets were grouped, each "set" consisted of 15 targets of the same size. At the start of each set an additional 16th target was shown in the centre of the screen to ensure the pointer location at the start of each set was roughly identical. Data recording started once that first target was clicked and events were recorded until the last of the 15 subsequent targets

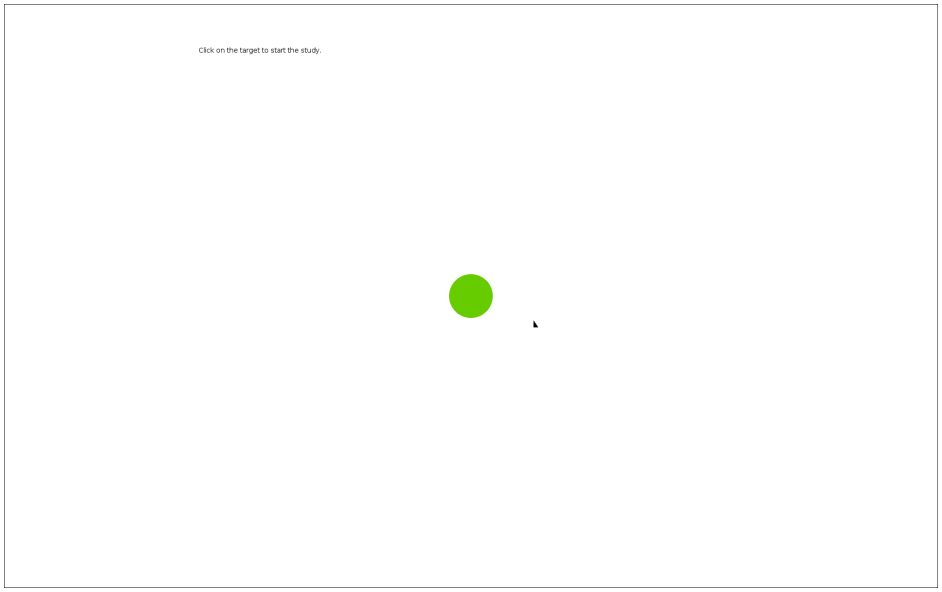

Figure 2: Screenshot of the study tool with the first target (radius 45px)

was clicked.

On each successful click within the target radius, a new target appeared on one out of twelve possible locations, arranged in a grid of 4x3 with grid points 300 pixels apart. The location of the target was randomly selected but was never on the same location twice in a row.

Each participant was tested for two acceleration methods, each acceleration method had 6 sets of 15 targets (2 sets per target size, order randomised). The two acceleration methods were randomly selected on startup, throughout the study they were simply referred to as "first" and "second" acceleration method with no further detail provided.

Before the first set, participants were given a training session with 10 targets to familiarise themselves with the task. After 6 sets, participants were informed the pointer acceleration method had changed to the second method and another 10-target training session had to be completed. At the end of each set participants were given the chance for a rest. As said earlier, no events were recorded during these breaks, event recording continued with a click on the first target.

After the completion of all 12 sets, users were given a questionnaire about themselves and their experience with the two acceleration methods. Once completed, users were asked to save the file and upload it to the study website.

For a short introduction into basic statistical concepts, see Appendix A.

4 Study participants

Invitation emails were sent to three Red Hat-internal mailing lists with a link to the study description. One list was a developer-specific list, the other two lists were generic lists. As Red Hat employees, all recipients are expected to be familiar with Linux-based operating systems and the majority are more technical than the average user. The data collected does not make it possible to identify who took part in the study.

44 participants submitted results, 7 left-handed, 37 right-handed (no ambidextrous option was provided in the questionnaire). Gender distribution was 38 male, 6 female. Mean age was 33.3 years (SD 6.7) and participants had a mean 21.2 years of experience with mouse-like input devices (SD 4.9) and used those devices an average 58.1 hours per week (SD 20.0).

RPM packages were provided for Fedora 19, 20, 21 and rawhide, instructions to build from source were provided as well. Build instructions were given for a specific git tag which was also the head of the git branch for the duration of the study to avoid any inconsistencies. It is not possible to identify whether RPMs or a source build was used from the submitted data. The study recorded the kernel version via uname. Based on that, the distribution was Fedora 19 - 4 participants, Fedora 20 - 37 participants, Fedora 21 - 3 participants.

As all participants are familiar with Linux systems and thus exposed to the smooth acceleration method on their workstations, a bias towards the smooth acceleration method was expected.

5 Study data

Data collected included relative movement deltas for each event and absolute pointer position. Figure 3 shows a reconstruction of one participant's cursor movement. Data files were manually checked and verified, three files were discarded:

- One file had no motion events and only button events. This indicates a bug in the user study tool.
- One file was dropped as extreme outlier. Instead of a straight movement from one target to the next, at least two of the targets movements across the whole screen, as if shaking the mouse or performing some other task. One movement from 800/600 to 800/300 took more than 1300 events and several thousand pixels of movement (see Figure 4).

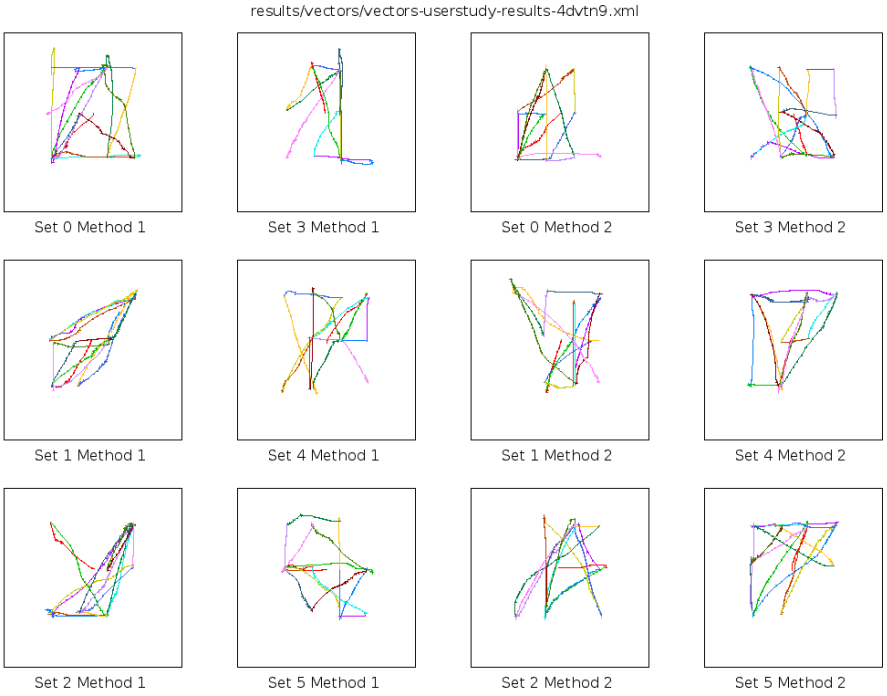

Figure 3: Cursor movement of one participant for all 15 targets of all 12 sets

- One file was dropped because of extreme delays between events. Several targets had multi-second pauses, suggesting the participant was otherwise occupied. Some of the pauses are longer than the average time-to-click for a target (e.g. three different target showed delays of 8, 6 and 9 seconds); indeed 7 out of the 10 slowest time-to-click targets came from this file.

This leaves 41 data files, with 82 sets of data. Two data files were generated from a locale using commas as a decimal separators instead of periods. These files were modified using a standard search/replace function.

The distribution of methods in these sets was: 27 for smooth, 25 for stretched and 30 for linear. Table 1 shows the distribution for each combination and how often the respective method was the first method.

The central unit of measurement was the so-called "Index of Difficulty" (ID): $ID = distance\text{-}of\text{-}target/width\text{-}of\text{-}target$. The ID gives an indication on how difficult it is to hit the target; a large target very close is easier to hit than a small target that is some distance away. Figure 5 illustrates the basic principle.

Retrospectively, the study was not ideally suited for evaluation based on ID. The targets were aligned on a grid and the ID based on the pointer

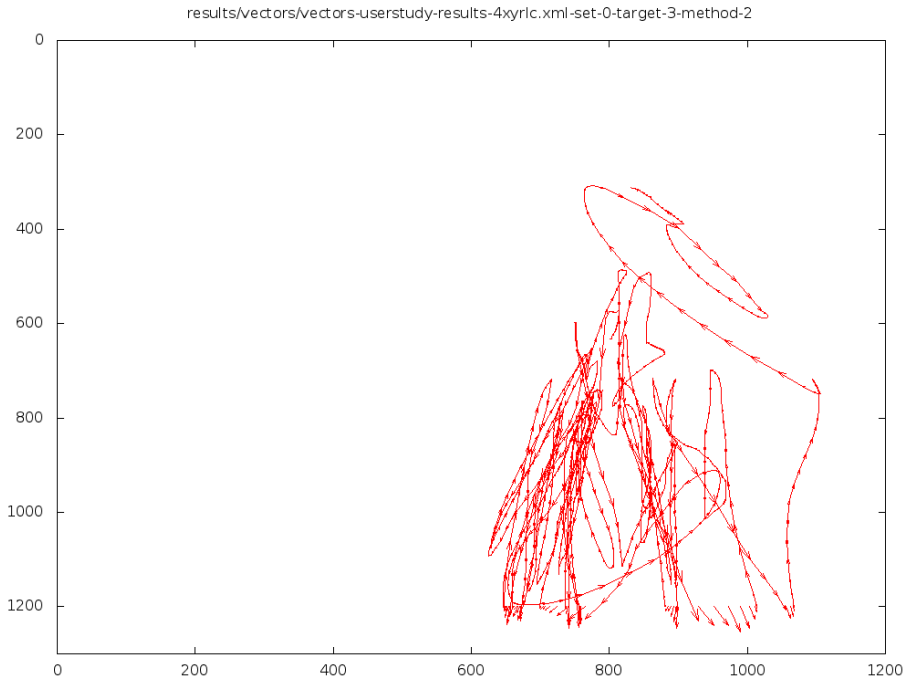

Figure 4: Cursor movement for a single target in the discarded data file

Figure 5: Illustration of the Index of Difficulty for a target

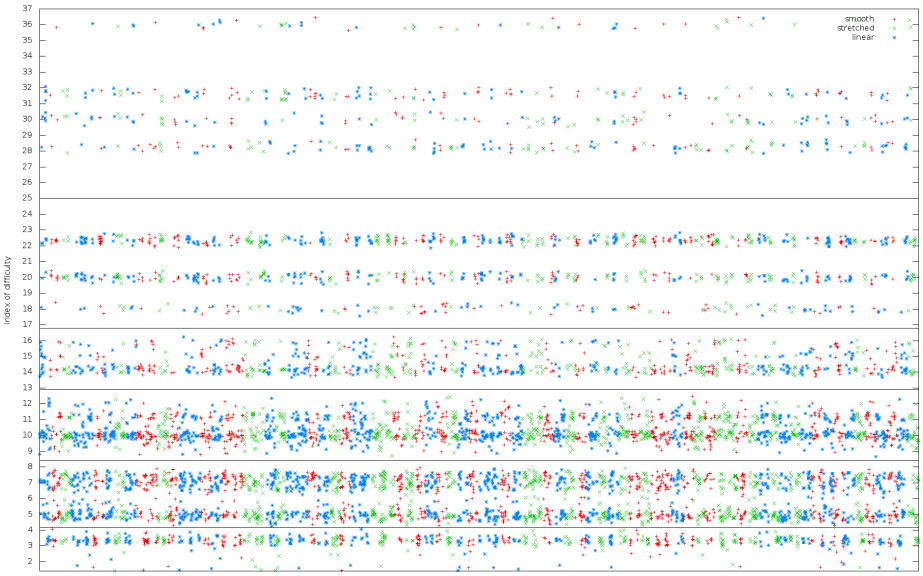

Figure 6: ID for each target with the group divider lines shown

position was highly variable. Figure 6 shows the ID for all targets. As is visible in the graph, there are few clear dividing lines to categorise the targets based on their ID. For the evaluation the targets were grouped into specific ID groups: ID < 4.2, ID < 8.4, ID < 12.9, ID $< 16.9 <$ ID < 25 and ID > 25. The numbers were selected because of the clear gaps between the ID clusters (see Figure 6). This division results in uneven group sizes, Figure 7 shows the number of targets for each method in each ID group based on this division.

The highest ID was 36.44, corresponding to a 15px radius target 1093 pixels away, the lowest ID was 1.45, corresponding to a 45px radius target 130 pixels away. The mappings of ID groups to distance per target size are shown in Table 2.

Informal testing of different ID groups did not yield significantly different results.

Table 1: Distribution of pointer acceleration methods

	smooth second	stretched second	linear second
smooth first	-	4	7
stretched first	7	-	8
linear first	9	6	-

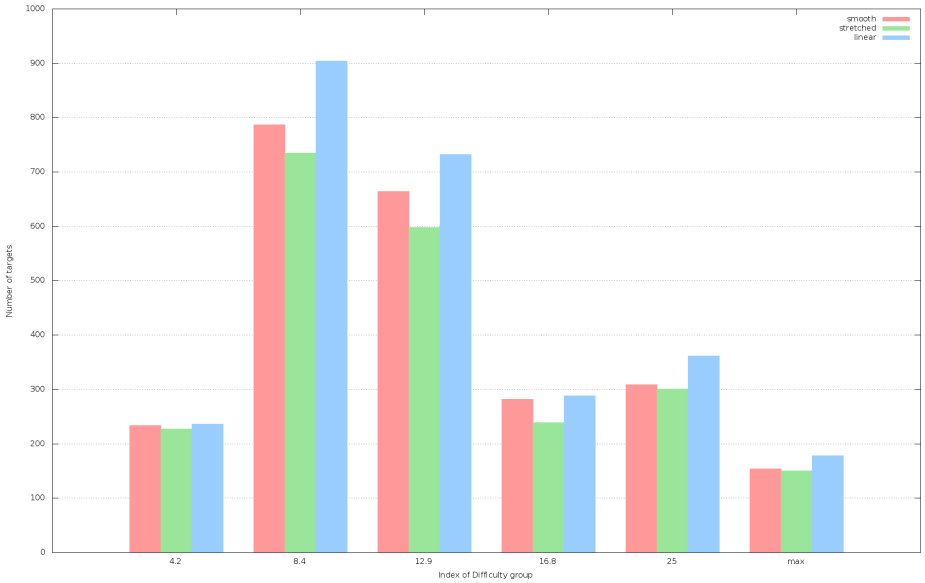

Figure 7: Number of targets per ID group

Table 2: Maximum distance in px for each target radius for each ID group

ID group	15px	30px	45px
4.2	126	252	378
8.4	252	504	756
12.9	387	774	1161
16.8	504	1008	1512
25	750	1500	2250
max	...	...	...

One-way Analysis of Variance (ANOVA, see Appendix A) was performed separately between the methods (i.e. smooth vs stretched, smooth vs linear, stretched vs linear). As ANOVA requires equal-sized sample sets, the two data arrays were cut to be of equal length before each analysis. For example, comparing smooth and stretched in the ID max group shortened the smooth dataset to 150 elements. The order of targets was randomised after reading in the files (Python's `random.shuffle` with a `random.seed(123456789)` [2]). Thus, the targets that were dropped before comparison were random, but the resulting sample sets were consistent across the analysis of various factors. ANOVA was performed using SciPy's `scipy.stats.f_oneway` function. [3]

ANOVA is a common analysis method for this type of study. One-way ANOVA requires only a single factor to change in the sample sets. For this study this factor was the pointer acceleration method, but all participants

Table 3: Number of targets per acceleration method per ID group

ID group	smooth	stretched	linear
4.2	234	227	236
8.4	787	735	904
12.9	664	598	732
16.8	282	239	288
25	309	301	362
max	154	150	178

Table 4: Device types (as classified by the participants)

mouse	28
trackball	3
touchpad	7
pointing stick	3

executed the study on their hardware. This introduces a wide variety of hardware with different features (see Table 4). The expectation was that a large enough sample set would neutralise hardware differences and even allow for hardware-specific analysis of the pointer acceleration method. A sample set of 41 does not neutralise hardware differences or allow for finer-grained analysis. Any results presented in the following must be viewed in that light.

6 Study Results

The following factors were analysed:

- Time to click on target
- Movement efficiency
- Overshoot

6.1 Time to click on target

Time to click on a target was measured as the time between displaying the target and clicking on it. This does not take reaction time into account, but there is no reliable way of measuring reaction time in this setup.

The mean times varied by ID and method used, but overall increased as the ID increased (see Figures 8 and 9). The mean times for each method and ID group shown in Table 5.

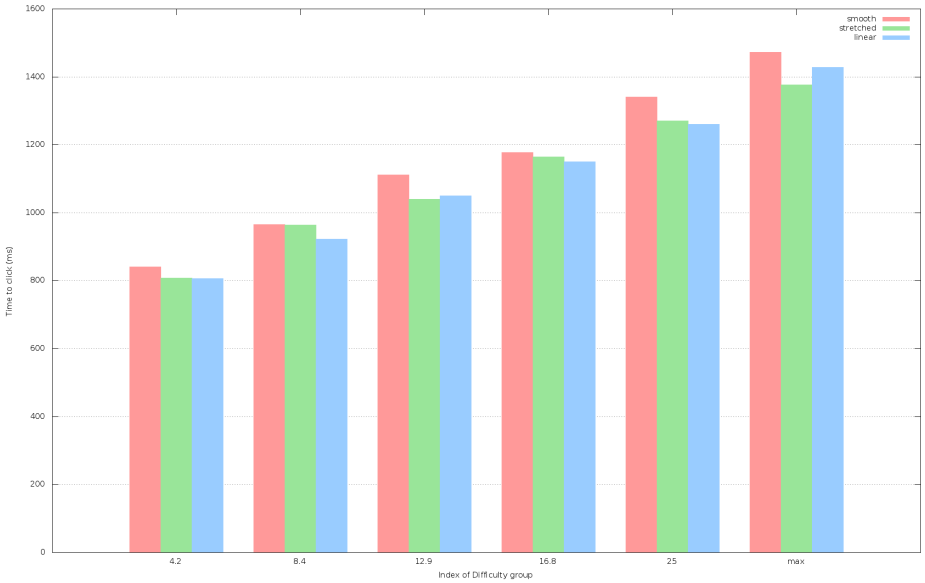

Figure 8: Mean time to click on a target

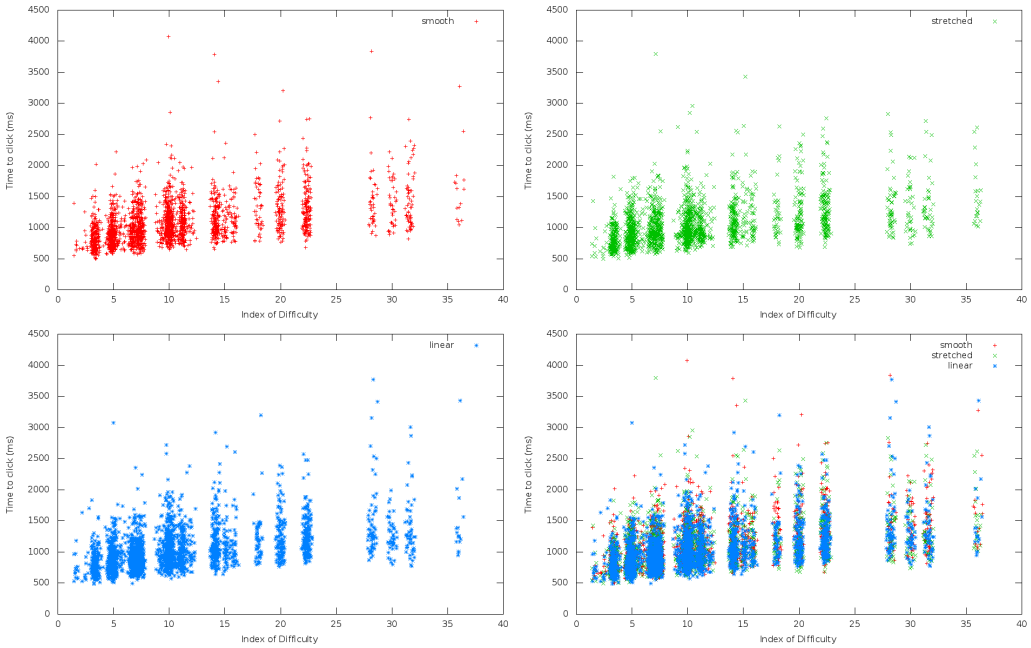

Figure 9: Click times for each target

Table 5: Mean time-to-click in ms for each method and target size

ID group	smooth	stretched	linear
4.2	843 SD(220)	808 SD(197)	811 SD(219)
8.4	963 SD(257)	964 SD(284)	911 SD(228)
12.9	1105 SD(317)	1041 SD(298)	1035 SD(288)
16.8	1179 SD(365)	1164 SD(366)	1152 SD(348)
25	1344 SD(396)	1271 SD(402)	1264 SD(351)
max	1476 SD(471)	1376 SD(441)	1402 SD(516)

The smooth method appears slower than the other two in most ID groups, linear and stretched appear to be fairly similar. However, the differences are only statistically significant in the following cases:

- ID 8.4: linear is faster than smooth and stretched
- ID 12.9: linear and stretched are faster than smooth
- ID 25: linear and stretched are faster than smooth

In all other combinations, there is no statistically significant difference between the three methods.

6.2 Efficiency of movement

The most efficient path from the cursor position to the target is a straight line. However, most movements do not follow that straight line for a number of reasons. One of these reasons is caused by human anatomy - the wrist's rotatry action makes straight movements difficult. Other reasons *may* be deficiencies in the pointer acceleration method. To measure the efficiency, we calculated the distance to the target (i.e. the straight line) and compared that to all the deltas added up to the total movement. Figure 10 illustrates that principle. Note that the distance is to the center of the target, whereas the actual movement may be to any point in the target. So for short distances and large targets, a movement may be less than the distance to the target.

The efficiency was calculated as movement-path/distance and then normalised to a percent value. A value of 10 thus means the movement path was 10% longer than the straight line to the target centre. Figure 11 shows the mean of these additional distances per ID group.

Figure 11 shows that stretched seems to perform better than smooth and linear in all but one ID group and smooth performing worse than linear in all but ID group 4.2. Looking at the actual values however shows that the large standard deviation prevents statistical significance.

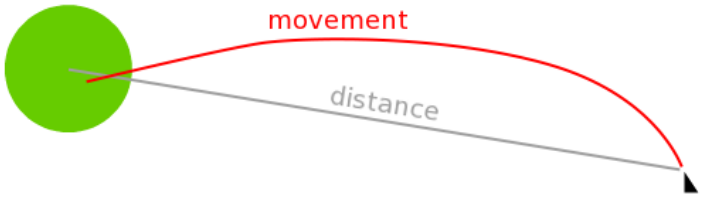

Figure 10: Distance to target vs. movement path

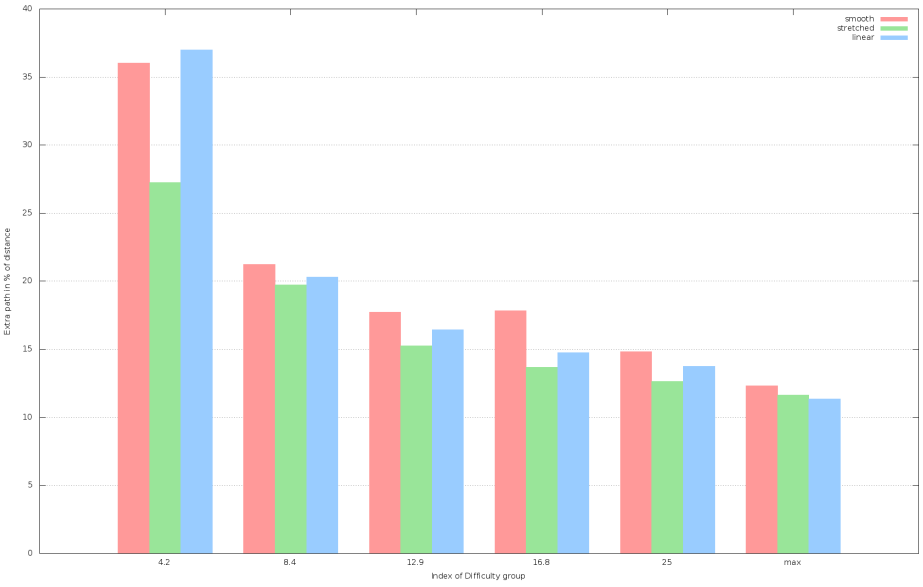

Figure 11: Mean of extra distance covered

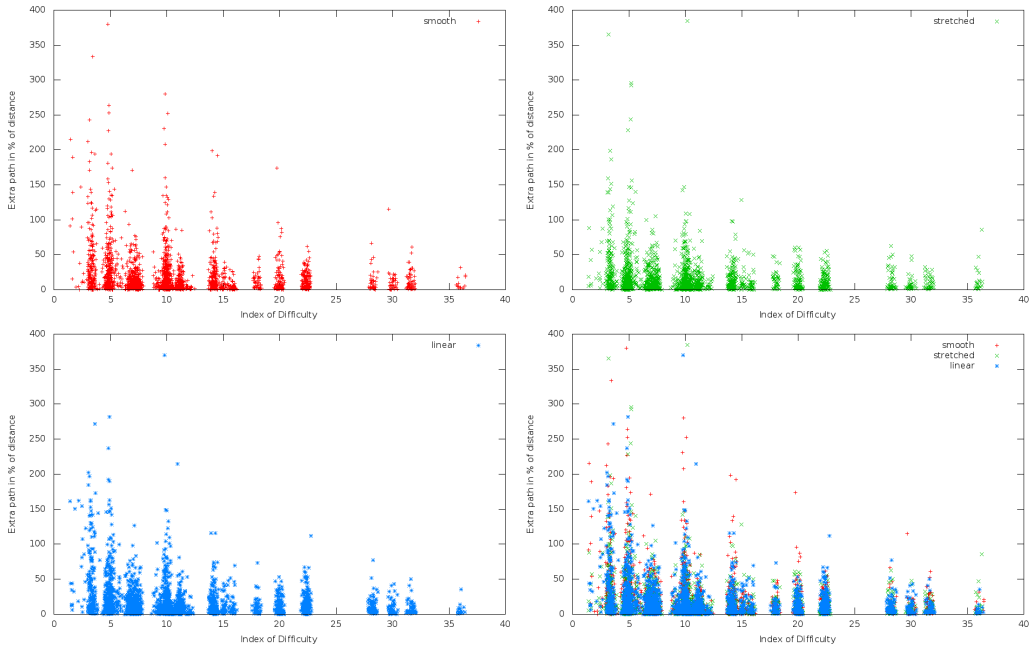

Figure 12: Extra distance for each target

The differences are only statistically significant in the following cases:

- ID 4.2: stretched is more efficient than smooth and linear

In all other combinations, there is no statistically significant difference between the three methods.

Figure 11 shows that when measured in percent of the distance to the target, the extra distance is higher for low-ID targets than for high-ID targets. Curiously, the extra path when measured in pixels remains almost constant across the ID range, as Figures 13 and 14 show. The differences between the methods are only statistically significant in the following cases:

Table 6: Mean extra path in % of the distance

ID group	smooth	stretched	linear
4.2	36.2 SD(50.9)	27.2 SD(42.1)	38.5 SD(51.0)
8.4	19.6 SD(30.3)	19.7 SD(48.9)	19.1 SD(30.3)
12.9	17.7 SD(29.8)	15.3 SD(24.7)	16.0 SD(29.8)
16.8	17.3 SD(26.8)	13.7 SD(17.2)	14.2 SD(26.8)
25	14.8 SD(17.4)	12.6 SD(13.1)	14.1 SD(17.4)
max	12.2 SD(14.7)	11.6 SD(13.6)	10.6 SD(14.7)

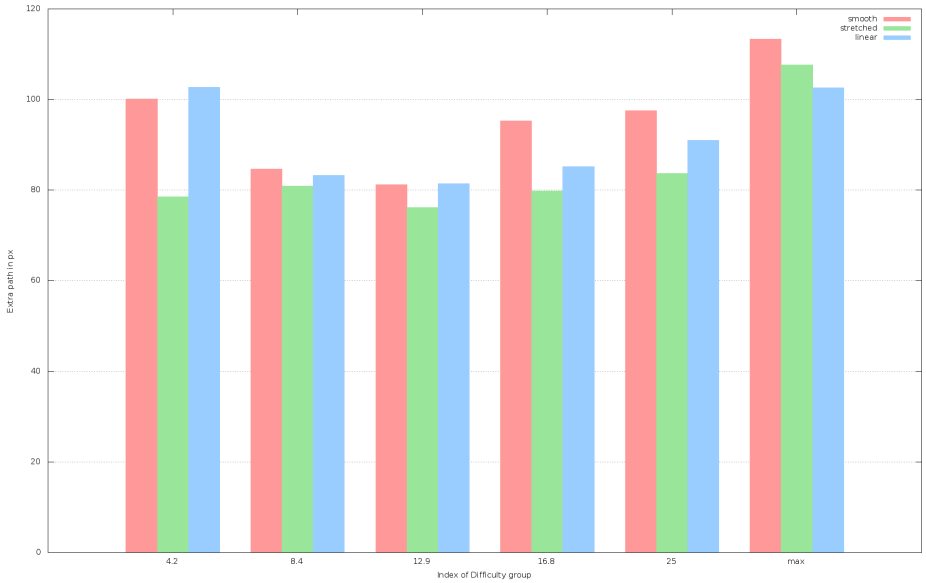

Figure 13: Mean of extra distance in pixels

- ID 4.2: stretched is more efficient than linear

In all other combinations, there is no statistically significant difference between the three methods.

6.3 Overshoot

Somewhat similar to the efficiency of movement, the overshoot is the distance the pointer has moved *past* the target. It was calculated by drawing a line perpendicular to the direct path from the pointer position to the target's far side. If the pointer moves past this line, the user has overshot the target. The maximum distance between the line and the pointer shows how much the user has overshot the target. Figure 15 illustrates this principle.

Overshoot was calculated in pixels, as % of the distance and as % of the actual path taken. Unsurprisingly, the graphs are largely identical so only the overshoot in pixels is provided here (see Figure 16).

As the ID increases, the amount of overshooting increases too. Again the three pointer acceleration methods are largely the same, though linear seems to be slightly less affected by overshoot than smooth and stretched. The differences are only statistically significant in the following cases:

- ID 4.2: if measured as percentage of distance, stretched has less overshoot than linear.

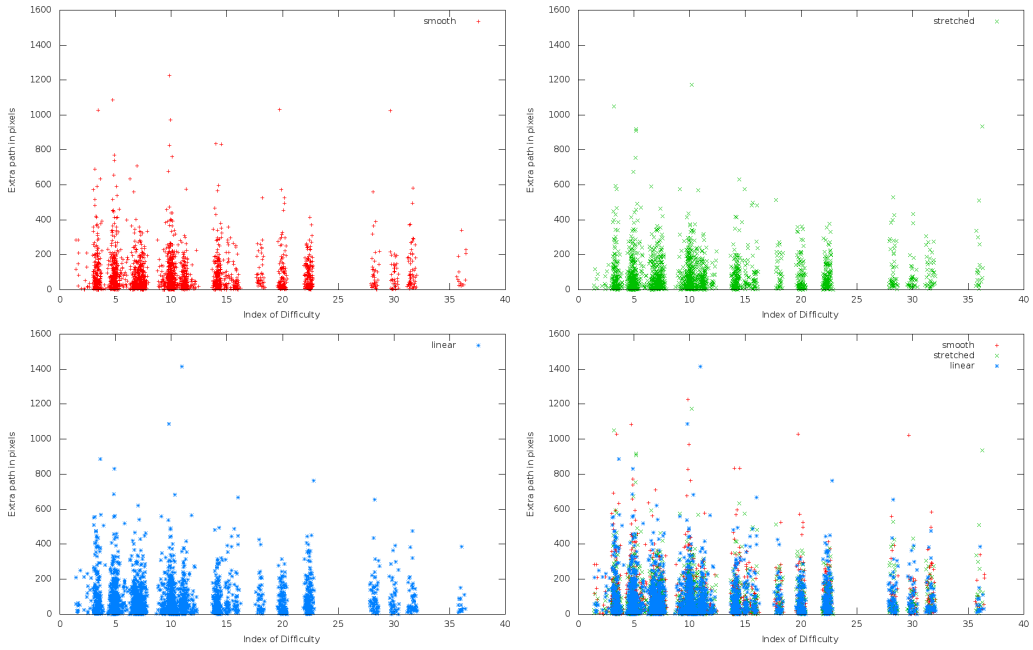

Figure 14: Extra distance in pixels per target

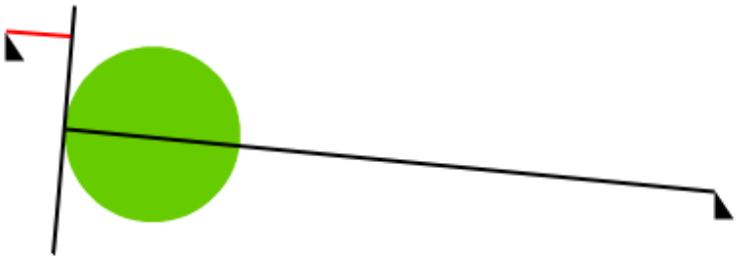

Figure 15: Illustration of pointer overshooting the target. The red line shows the amount the pointer has overshot the target.

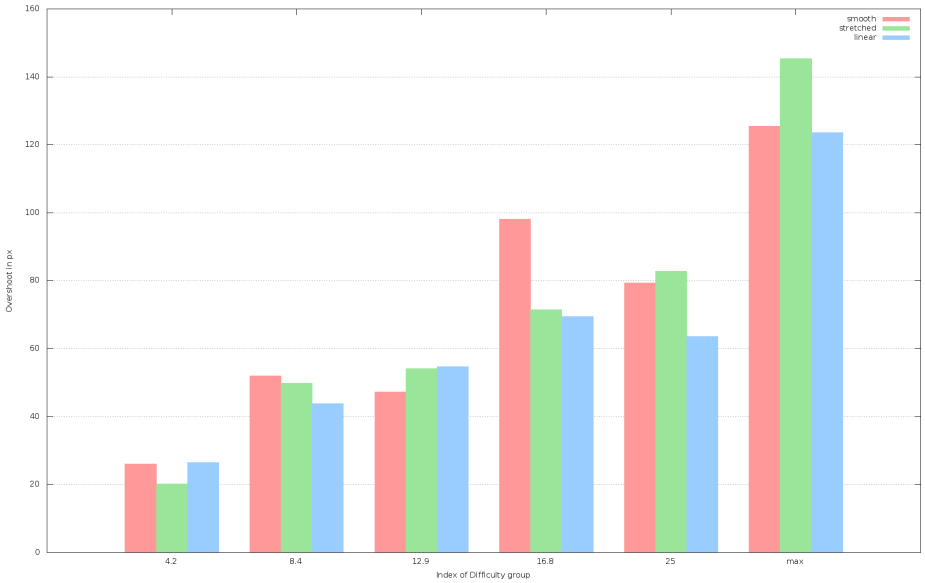

Figure 16: Overshoot in pixels by ID group

Table 7: Overshoot in pixels by ID group

ID group	smooth	stretched	linear
4.2	26.1 SD(44.0)	20.1 SD(35.3)	27.1 SD(44.0)
8.4	50.0 SD(95.1)	49.8 SD(96.6)	41.0 SD(95.1)
12.9	48.0 SD(105.7)	54.1 SD(116.9)	48.9 SD(105.7)
16.8	93.5 SD(145.3)	71.5 SD(132.9)	61.4 SD(145.3)
25	79.5 SD(145.5)	82.7 SD(133.1)	73.4 SD(145.5)
max	122.7 SD(191.0)	145.4 SD(211.6)	110.9 SD(191.0)

- ID 8.4: if measured as percentage of movement path, linear has less overshoot than smooth.
- ID 16.8: if measured as percentage of distance, stretched and linear have less overshoot than smooth.
- ID 16.8: if measured as percentage of distance, linear has less overshoot than smooth.
- ID 16.8: if measured in pixels, linear has less overshoot than smooth.

In all other combinations, there is no statistically significant difference between the three methods.

Table 8: The questions of the participant questionnaire

1	The first acceleration method felt natural
2	The first acceleration method allowed for precise pointer control
3	The first acceleration method allowed for fast pointer movement
4	The first acceleration method made it easy to hit the targets
5	I would prefer the first acceleration method to be faster
6	I would prefer the first acceleration method to be slower
7	The second acceleration method felt natural
8	The second acceleration method allowed for precise pointer control
9	The second acceleration method allowed for fast pointer movement
10	The second acceleration method made it easy to hit the targets
11	I would prefer the second acceleration method to be faster
12	I would prefer the second acceleration method to be slower
13	The two acceleration methods felt different
14	The first acceleration method was preferable over the second

7 Questionnaire results

In addition to the objectively measured data, subjective data was gathered from the participants through 14 questions (see Table 8). Answers were in in a 5-point Likert scale, ranging from "Strongly Disagree" (value -2) to "Strongly Agree" (value 2). Figure 17 figure below shows that comparatively few "strongly agree" and "strongly disagree" answers were given, hinting that differences between the methods were small.

The answers in the questionnaire are inconclusive. Figure 18 shows the answers for the first 12 questions, Table 9 lists the mean values for each question for each method. The differences between the answers are only statistically significant in the following case:

- Participants disagreed that smooth and stretched should be slower, but the disagreement was significantly stronger for the stretched case.

Disappointingly, Question 13 "The two acceleration methods felt different" likewise had an inconclusive result (see Table 10), with no statistical significance between the methods. Some participants noticed a difference, others did not.

Question 14 "The first acceleration method was preferable over the second" too had an inconclusive result (see Figure 19 and Table 11). While Figure 19 shows a slight preference to the smooth version when comparing smooth and linear, the difference is not statistically significant. The analysis was performed for an acceleration method order as shown in Table 10.

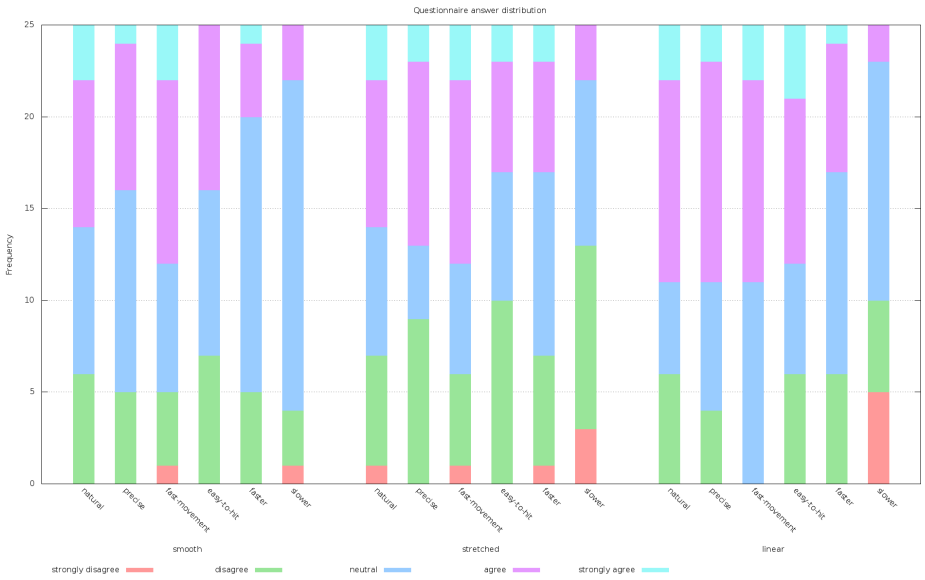

Figure 17: Distribution of answers in the questionnaire

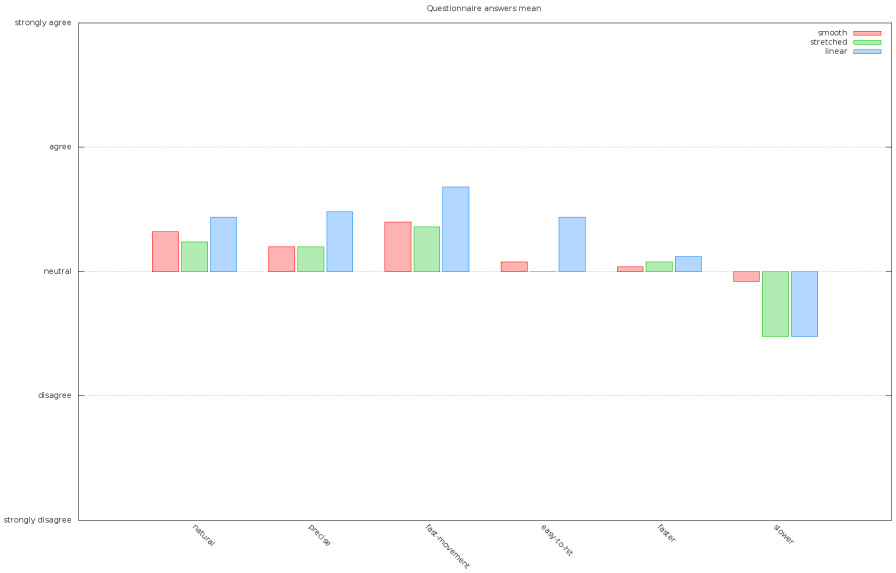

Figure 18: Mean answers for questions 1-12

Table 9: Answers to questions 1-12

question	smooth	stretched	linear
natural	0.32 SD(0.96)	0.24 SD(1.07)	0.44 SD(0.98)
precise	0.20 SD(0.80)	0.20 SD(1.02)	0.48 SD(0.85)
fast-movement	0.40 SD(1.02)	0.36 SD(1.05)	0.68 SD(0.68)
easy-to-hit	0.08 SD(0.80)	0.00 SD(0.98)	0.44 SD(1.02)
faster	0.04 SD(0.72)	0.08 SD(0.98)	0.12 SD(0.82)
slower	-0.08 SD(0.63)	-0.52 SD(0.85)	-0.52 SD(0.90)

Table 10: Mean answers to question 13

smooth vs. stretched	0.55 SD(0.78)
smooth vs. linear	0.45 SD(1.23)
stretched vs. linear	0.72 SD(0.75)

If a participant had the order of acceleration methods swapped, the Likert value was multiplied by -1. For example, if the participant had stretched as first method and smooth as second method, a "Strongly Agree" to Question 14 would be counted as "Strongly Disagree" when comparing smooth vs. stretched.

Overall, the result of the Question 14 is inconclusive, with no decisive answers. Figure 19 shows a slight preference for the linear method compared to the stretched, and a slight preference to the smooth method compared to the linear one, but the standard deviations make the differences statistically insignificant.

8 Summary

The questionnaire was not able to identify a single pointer acceleration method as distinctively advantageous. The stretched and linear methods have slight advantages over the smooth methods in objective data methods. For the sheer simplicity of the linear method the recommendation for upstream is to switch to the linear method as default acceleration method.

Table 11: Mean answers to question 14

smooth vs. stretched	-0.09 SD(1.08)
smooth vs. linear	0.36 SD(1.15)
stretched vs. linear	-0.55 SD(0.65)

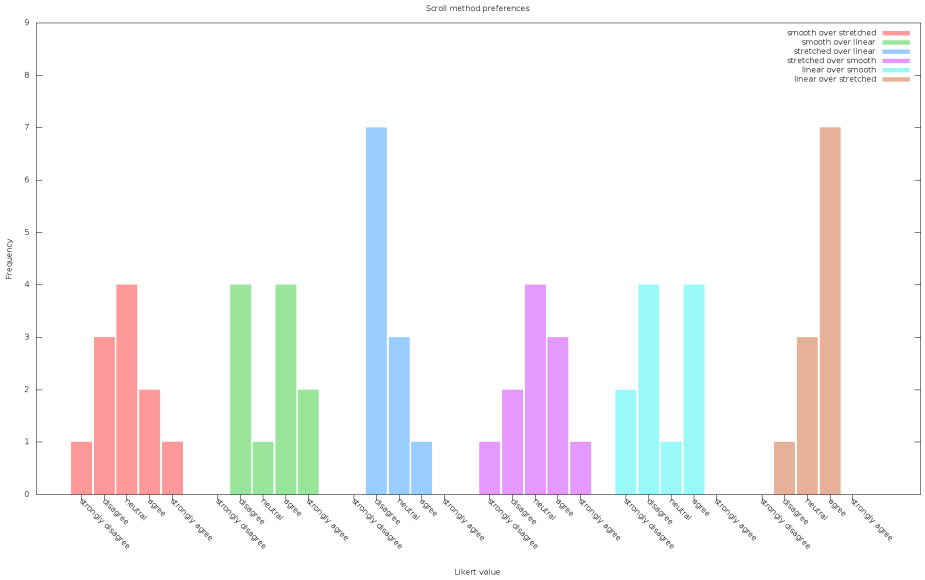

Figure 19: Likert counts for each method combination

9 Acknowledgements

Special thanks go to Benjamin Tissoires, Hans de Goede, Maírín Duffy for their help and comments during the study and all the participants in the trial and the user study.

References

[1] http://www.x.org/wiki/Development/Documentation/PointerAcceleration/. Retrieved Sep 14, 2014.

[2] https://docs.python.org/2/library/random.html. Retrieved Sep 14, 2014.

[3] http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.f_oneway.html. Retrieved Sep 15, 2015.

[4] http://en.wikipedia.org/wiki/F_test#One-way_ANOVA_example. Retrieved Sep 14, 2014.

[5] http://www.gnuplot.info. Retrieved Sep 14, 2014.

[6] Géry Casiez and Nicolas Roussel. 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*, UIST '11, pages 603–614, New York, NY, USA, 2011. ACM.

[7] Peter Hutterer. pointer acceleration in libinput - an analysis. http://who-t.blogspot.com.au/2014/07/pointer-acceleration-in-libinput.html, 2014. Retrieved Sep 14, 2014.

A Statistical concepts

A short foray into statistics to help explain the numbers presented in this paper.

The *mean* of a dataset is what many people call the average: all values added up divided by the number of values. As a statistical tool, the mean is easy to calculate but is greatly affected by outliers. For skewed datasets the *median* is be more helpful: the middle value of the data array (array[len/2]). The closer the mean and the median are together, the more symmetrical the distribution is.

The *standard deviation* (SD) describes how far the data points spread from the median. The smaller the SD the closer together are the data points. The SD is also used to estimate causality vs randomly induced sampling errors. Generally, if the difference between two items is more than 2 standard deviations, there's a 95% confidence that this is a true effect, not just randomness (95% certainty is a widel accepted standard in this domain). That 95% directly maps to the *p-value* commonly presented in other studies. A p-value of less than 0.05 equals a less than 5% chance of random factors causing the data differences. That translates into "statistically significant".

The *ANOVA* method is a standard statistical tool for studies such as the one presented here. . We're using one-way ANOVA only here, see Wikipedia for an example [4]. If multiple sets of samples differ in a only single factor (e.g. pointer acceleration method), the baseline is the so-called "Null-Hypothesis": "the factor has no influence, all results are the same on average". If the Null-Hypothesis can be rejected, the factor did actually change things. Otherwise, the factor did not change anything or the results are caused by random influences. The tools for ANOVA compare the mean value within each sets to the mean value differences across the sets and spit out a p-value. As above, a p-value of less than 0.05 means a greater than 95% confidence that the Null-Hypothesis can be rejected, i.e. the changing factor the measured differences. One peculiarity of ANOVA is that sample sets have to be the same size.

B Source Code

The source is available in the `wip/userstudy` branch of the git repository at `https://github.com/whot/libinput/` The source code used during the study is the git tag `userstudy-1.2` (sha `3769bd4cce`)[2] The source code used to generate graphs and numbers in this paper is the git tag `userstudy-1.2.1` (sha `cd5c5a1ba6`)[3]

The data files submitted by participants are available under the git tag `userstudy-1.2` (sha 763117c17d)[4] in the repository at `https://github.com/whot/libinput-userstudy-data/`

To build the tool and the analysis results, run

```
$ autoreconf -ivf
$ ./configure --with-userstudy-results=/path/to/results
$ make
```

Resulting data files and graphs are available in `tools/analysis/results`. The gnuplot program [5] is required to generate the graphs.

[2] `https://github.com/whot/libinput/tree/userstudy-1.2`
[3] `https://github.com/whot/libinput/tree/userstudy-1.2.1`
[4] `https://github.com/whot/libinput-userstudy-data/tree/userstudy-1.2`