Acquiring Targets in the Velocity Domain: Toward Predictive Modeling of Virtual Tossing

Michael Mcguffin, Pierre Dragicevic, Luc Tremblay

Tech Report: Acquiring Targets in the Velocity Domain: Toward Predictive Modeling of Virtual Tossing

Michael J. McGuffin
Department of Software and IT Engineering
École de technologie supérieure, Montreal, Canada
michael.mcguffin@etsmtl.ca

Pierre Dragicevic
INRIA
Paris, France
dragice@lri.fr

Luc Tremblay
Faculty of Physical Education and Health
University of Toronto, Toronto, Canada
luc.tremblay@utoronto.ca

ABSTRACT

Tossing, throwing, or flicking objects in a user interface or virtual environment can be used as a faster, lower-precision alternative to traditional pointing, however there is currently no predictive model of user performance with tossing. We report experimental measurements of performance in a 1D tossing task from which a predictive model is derived. We consider a simplified form of tossing where a virtual object on a horizontal surface is accelerated and released, and then decelerates under friction, coming to rest at some final position. The distance traveled after release is determined by the release velocity as well as by the friction model used. To abstract away the details of the friction model, our experiment measures the ability of users to accelerate and release a virtual object in 1D (using a mouse) with a given target velocity, with target velocities varying from 6.25 cm/s to 1 m/s. Results indicate that there is a linear relationship between the target release velocity and the standard deviation of the release velocity achieved by the user. We also propose an automatic release technique (instead of requiring the user to manually release using a mouse button) that significantly improves precision. The model derived from our experiment predicts that a user should be able to toss at three different target speeds (effectively tossing toward target locations at three different distances) with an error rate under 4%. We also predict that having four or more targets in the same direction would cause the error rate to rise above 10%. Design implications for integrating tossing into graphical user interfaces are discussed.

Author Keywords

Throwing, flinging, tossing, flicking, sliding, pushing, pointing, Fitts' law, interaction design

INTRODUCTION

Tossing, or gently throwing objects at a relatively low speed, is an everyday activity in the real world, used to save effort when precise placement is not required. Paper documents are tossed onto work surfaces, sometimes in rough piles or groupings, and sometimes to create an intentionally informal arrangement. Objects are tossed from one person to another to save time, and items to be sorted, packaged, or disposed of may be tossed into a basket, box, or recycling bin, at times providing a fun challenge and yielding a sense of satisfaction when the toss is successful. Tossing or throwing are a central element of many sports, and it is not unusual for trained athletes to achieve a low error rate in tossing a ball toward some target. For example, several dozen of the professional basketball players listed at nba.com have a "free throw percentage" above 80%, some as high as 90%[1].

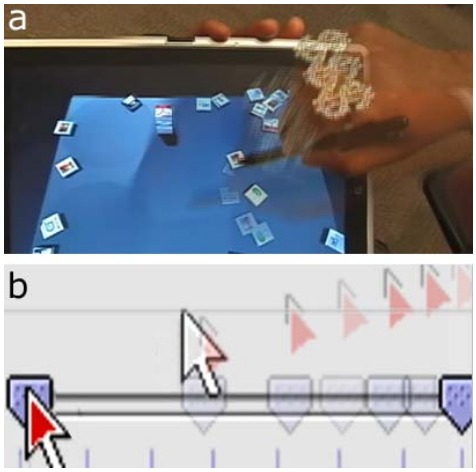

Figure 1. Two examples of tossing in GUIs. *Top:* In the physics-based desktop environment BumpTop, an icon is tossed towards a group of related icons [1]. *Bottom:* An inertial mouse cursor is used to rapidly set a slider to its maximum value [15].

In user interfaces, simplified or metaphorical forms of tossing have been used as an alternative to pointing at a distant location, where users may drag, gesture, or flick in a given direction [16, 5, 23, 13, 37, 2] (Figure 1). As pointed out in [37], sliding or flicking objects is a fast, easy-to-understand action and a natural extension of normal drag-and-drop actions, possibly giving it an advantage over other techniques for pointing at distant targets. One system [1] even simulates collisions between the tossed object and obstacles, causing the object to bounce off walls or displace other objects previously placed on a virtual surface. Virtual tossing can also

[1] In a free throw, the throw line is about 420 cm from the center of the net, which has a diameter of 45 cm, and the basketball has a diameter of 24 cm, yielding an index of difficulty of at most $\log_2(\frac{420}{45-24}+1) \approx 4.4$ bits, ignoring the fact that small errors can still result in the ball falling through the net.

be useful if the user wants to quickly throw, sort, or arrange objects, and precision is not required or not desired.

Unfortunately, there is still no predictive model of tossing that could be used to derive design guidelines for user interfaces. With a 2D input device and 2D user interfaces, we are particularly interested in tossing tasks comparable to the sport of *curling*, in which players slide and accelerate an object on a horizontal surface and then release the object, which then decelerates due to friction. (This is also called flicking [37].) Many open questions exist related to such a task. If the user wishes to toss toward some target, how small can the target be made before the error rate becomes unacceptable? How many different directions can a user toss in? In a given direction, how many different distances can a user toss toward? Is there a tradeoff between movement time and precision that can be made in tossing, as with traditional pointing? Can tossing performance be predicted from the index of performance in Fitts' law [18, 29]? How should the release of a tossed object be triggered? What is the maximum speed with which a user can toss? How does the index of performance (or bandwidth) of tossing compare to that of traditional pointing?

Regarding the question of how many directions a user can toss in, previous work suggests an answer. In [28], error rates rose above 5% if there were as many as 12 directions to stroke in within a single-level marking menu. Similarly, in [16], 90% of strokes were within $\approx 15^\circ$ of the target direction, corresponding to an error rate of 10% if there are 12 discrete directions to stroke in. In this previous work, users did not need to perform the strokes with any particular velocity, whereas in tossing they would need to do so, making tossing in different directions at least as difficult. Hence 12 items could be taken as an upper bound on the number of directions for tossing. The other questions in the previous paragraph are investigated by the current work, by considering a simplified form of tossing. The distance traveled during a toss depends on (1) the offset in position that is created during the acceleration stage, (2) the velocity at the time of release, and (3) the kind of friction simulated during deceleration. The first and third factors are abstracted away if we focus on the question of how well users can toss with a given velocity. In other words, how well can users "acquire" targets in the velocity domain? We performed an experimental study of such performance in 1D, resulting in answers to the previous questions. The major result of our study is a predictive model of the standard deviation in the user's release velocity. This allows us to predict the optimal placement of a given number of targets in the velocity domain, and also predict the error rate that would result with such targets. We also discuss issues in integrating tossing into traditional point-and-click user interfaces.

BACKGROUND

Fitts' law [18, 33, 29] predicts the time to acquire a target with a rapid, aimed pointing movement. In the Shannon formulation of Fitts' law [29], given the distance D to reach the target, and the width W of the target measured along the axis of motion, the average movement time MT required is

$$MT = a + b \underbrace{\log_2\left(\frac{D}{W}+1\right)}_{ID} \quad (1)$$

where a and b are empirically determined constants that vary with factors such as the particular pointing device used. The logarithm is referred to as the index of difficulty ID which is in bits.

Aimed, rapid movements involve a speed/accuracy tradeoff, reflected in MT being an increasing function of the ratio D/W in Equation 1. Thus, if the distance D to a target is increased by some factor, the selection can still be performed in the same time (i.e. with greater speed) if the accuracy requirement is reduced (i.e. W is increased) by the same factor. Also observe that, if the target's size W is reduced by a factor of 1/2, this results in an increase of approximately 1 bit in ID, and a corresponding incremental increase of MT.

One way to interpret Equation 1, and the task it models, is that the user must home in on the target by reducing the initially large error (or "noise") in the cursor's position until the error is small enough that the cursor falls within the target. Every reduction by 1/2 of this error requires b seconds, and corresponds to transmitting 1 bit of information. The number of such reductions necessary is $\log_2(A/W) \approx ID$, and transmitting ID bits requires bID seconds in total. The remaining a seconds in Equation 1 can partly be explained as reaction time and/or the time necessary to complete the selection with a button press. More detailed and accurate explanations of the mechanisms behind Fitts' law have been developed [32, 33], however the foregoing is a useful first approximation.

Various techniques have been proposed to make pointing easier, for example, by making targets [31] (or the cursor [21, 10]) larger, by bringing targets closer to the cursor, or by making it easier to move the cursor toward a target (§4.2 in [31] contains an overview of several techniques up to around 2005). For example, some techniques enable the cursor to "jump" onto targets [22, 4]. Others make temporary proxies of distant targets available nearby the cursor [5, 7]. The techniques drag-and-throw, push-and-throw, and variants [23, 12, 13] have names suggesting they involve actions akin to tossing, but as pointed out in [37], these techniques do not actually involve imparting a velocity with a rapid drag or flick gesture.

More relevant to the current work is previous work on flicking [16, 34, 37, 2]. [16] and [34] measured the movement time, angle, and length of flick gestures, comparing them to alternate input techniques, however these flick gestures were not used to impart velocity on a virtual object. Reetz et al. [37] measured movement times and error rates in a target acquisition task, comparing three techniques: flicking (imparting velocity on a virtual object, with friction), pointing within a "radar" view of proxy targets, and superflicking (a form of flicking allowing corrections after release). Flicking was found to be the fastest technique, but also the most error

prone, with an error rate above 45% in some conditions. The current work differs in several ways: first, [37] does not report the details of the friction model used, whereas our work is independent of any friction model; second, [37] used an input device sampled at 50 Hz, whereas we use a mouse that reports its position at 500 Hz; third, [37] did not investigate a possible relationship between error rate and target distance or target size or index of difficulty; and fourth, [37] reports error rates for each condition, but not the distribution of final positions (from which an *effective width* [29] could be calculated, to more precisely quantify the user's precision). Aliakseyeu et al. [2] investigated the use of flicking for performing 1D scrolling of a view (where the flick gesture is used to give a velocity to the view), and proposed interesting variants on flicking. However, their work did not focus on the characteristics of individual flick gestures, and instead measured total time to scroll to a given target with one or more flicks. Finally, unlike [37, 2], the current work does not experimentally investigate tossing techniques that allow corrections after release, focusing instead on the more fundamental questions mentioned in the introduction that have not been addressed as much, or at all, in previous work.

Among older literature, two studies [26, 24] investigated tracking tasks where users had to capture moving targets, using either position or velocity control, and proposed predictive models of user performance. In the position control conditions, users had to move the pointing device with a speed matching the target's speed, making the task comparable to that in the current work. However, our work differs in the range of speeds investigated (speeds up to 1 m/s in our work, versus [26] where a joystick was moved at a target speed of up to about 7.5°/s, or about 6.5-13 mm/s assuming the stick was 5-10 cm long) as well as in the details of the task (in our work, the user must simply accelerate and release at a given target velocity, rather than first acquire a moving target and then match its velocity for a minimum interval of time).

MODELING FRICTION AND DECELERATION

Before discussing our experimental study, it is useful to first consider the kinds of friction that might be used in simulating tossing, and what their effect would be. Consider the 1-dimensional motion of an object of mass m sliding over a horizontal planar surface. The user accelerates the object, and then releases it at time $t = 0$ with velocity $v = v_0$, after which a negative friction force f causes deceleration.

Normally, a physical model might assume a *kinetic friction* force $f = -\mu_k mg$ where g is the gravitational acceleration and μ_k is the coefficient of kinetic friction. The constant acceleration due to this friction is $a = f/m = -\mu_k g$. Solving the equations of motion, the instantaneous velocity $v = v_0 + at$ and the object's position $x = v_0 t + \frac{1}{2}at^2$. Assuming the object stops when $v = 0$, the total distance traveled will be $D = \frac{1}{2\mu_k g}v_0^2$. Thus D is proportional to v_0^2. This is the type of friction used in the "multi-flick-friction" technique of [2].

Alternative models of friction lead to other dependencies of D on v_0. With a *fluid friction* drag force $f = -bv$ proportional to v, we have $a = (-b/m)v$, from which it follows that $v = v_0 e^{-(b/m)t}$ and $a = dv/dt = \frac{-b}{m}v_0 e^{-(b/m)t}$ and $x = \frac{m}{b}v_0(1 - e^{-(b/m)t})$. In this case, technically the velocity tends toward zero without ever reaching it, however we can still calculate that the total distance traveled is $D = \frac{m}{b}v_0$ which is proportional to v_0.

The distinction noted above between kinetic friction and fluid friction should be kept in mind in the following section, and will be returned to again in our discussion of Future Directions for this work.

TARGETS IN THE VELOCITY DOMAIN, AND (-1)TH-ORDER CONTROL

As already mentioned in the introduction, in the tossing tasks performed in our experimental study, the details of the type of friction used are abstracted away by having the user toss with a given target velocity. Each target velocity would normally map to a specific target distance (depending on the friction model used). However, instead of displaying a graphical representation of the position of the virtual object in space as it is tossed and gradually decelerates towards a target location, we display a cursor whose x coordinate is proportional to the mouse's sideways speed. Targets for tossing tasks are displayed directly in the velocity domain, and the user can see immediately whether the cursor is on the target or not, without having to wait for a virtual object to decelerate and stop. Although this is unlike users' real-world experience with physical tossing, it has the advantages of giving the user immediate feedback as to their performance, of allowing the user to move on to their next action faster, and is independent of the friction model used.

Two observations can be made regarding this. First, previous work has investigated both position control input ([26, 24, 37], for example) and velocity control input [26, 24], but in both cases the output has been displayed in the spatial domain, showing the spatial positions of the cursor and target(s). The terms zero-order control and first-order control [25] are also used to describe these, since the position of the input device is used to control either the position or velocity of the cursor, respectively. However, in our experimental tossing tasks, output is displayed in the velocity domain, and the *velocity* of the mouse controls the *position* of the cursor. Hence, our tossing tasks involve (-1)th-order control, by analogy. Our second observation is that, if we were to assume a fluid friction model, where the distance D traveled by the tossed object is proportional to the release velocity v_0, then there is simple linear mapping between the spatial and velocity domains. This means that displaying output in the velocity domain is almost equivalent to displaying output in the spatial domain, so the tossing task in our experimental study is not so different from physical tossing.

HYPOTHESIS

As mentioned in the background section, Fitts' law can be interpreted as stating that every reduction by 1/2 of the error in the cursor's position requires a constant amount of time b. In fact, one of the simplest lower-level models of aiming movements, from which Fitts' law can be derived, is the deterministic iterative-corrections model [14, 27]. In this

model, the aiming movement is composed of several submovements, each of which brings the limb (or cursor) closer to the target center by a constant ratio, and each of which requires a constant time. The distance remaining to the target's center after each submovement can be thought of as the error associated with the submovement, and is proportional to the length of the submovement. This error is also proportional to the average velocity of the submovement (since each submovement takes the same amount of time). A more sophisticated and recent model, from which Fitts' law can again be derived, is the stochastic optimized-submovement model [32], which assumes that the endpoints of the first (or primary) submovement have a distribution with a standard deviation that is proportional to the average velocity of the first submovement (see also [17] for a review). Such a relationship has been confirmed in subsequent work, for example in [40]. On a related note, a linear relationship has been found between the peak velocity achieved during the primary submovement and the target distance; this is discussed and reviewed in [4].

Turning to movements for tossing, [16, 34, 37] found average movement times for flick gestures varying between 154 ms [34] and 284 ms [16]. These movement times are close to the minimum time required for a user to able to make corrections based on visual feedback (the threshold is estimated to be somewhere between 100 ms [8] and 200 ms [29, §5.1]). This should greatly limit, or even preclude, the possibility of closed-loop corrections to the movement, implying that the movement is ballistic or nearly ballistic. In discussing the submovements involved in traditional pointing, [16] suggests that "[the] first movement or primary submovement may correspond to the flick gesture." If this is the case, that a flick or toss movement has the characteristics of the primary submovement in an aiming movement and is performed too quickly to allow for corrective submovements, then we should expect the "error" or standard deviation in the endpoint position of a toss movement to be proportional to the movement's average speed. Taking this one (speculative) step further, we hypothesize that for the tossing task performed in our experimental study, the standard deviation in the release velocity achieved by the user will be proportional to the target release velocity.

EXPERIMENT

We conducted a controlled experiment to measure performance in a velocity-target acquisition task, which we will refer to simply a tossing task for brevity. Specifically, our experiment was designed to

- Measure and compare tossing and traditional pointing, in terms of movement time, precision, and index of performance.
- Identify a candidate model for predicting performance with tossing. For example, we would like to be able to predict movement time and/or precision of tossing as a function of target position and possibly target size.

Apparatus

The pointing device used was a Logitech G5 laser optical mouse that reports its position at 500 Hz with 2000 dpi ($\approx$ 0.01 mm per dot) resolution. Using a mouse rather than a tablet has the disadvantage that we cannot detect if the user lifts and repositions the mouse on the surface, however we designed our tasks to make such lifting unnecessary or impossible, and informal observation of participants indicated they did not lift the mouse in the middle of trials. Furthermore, since mice are an extremely common input device, we wanted our measurements of tossing performance to be performed with a device of similar mass, form factor, and sensing capabilities.

We experimented with different surfaces for the mouse, including a large "professional gaming" mouse pad, and found that simply using the mouse on a laminated desk surface seemed to produce the cleanest data.

The output device was an UltraSharp Dell 1800FP, 18.1 inch, 1280×1024 pixel LCD screen.

The mouse coordinates were polled at 500 Hz and smoothed using a moving average filter on a 20 ms window (i.e. containing 10 samples). Because the screen's refresh rate was limited to 75 Hz, we could not update the cursor's position on the screen at 500 Hz, hence a motion blur was simulated by displaying 6 to 7 alpha-blended cursors per frame, each with an alpha of ≈ 0.15 (Figure 2).

Figure 2. The stimuli displayed during one trial. *Left top:* the cursor begins at the starting position (cross hair) at the left. *Left middle and right:* the cursor midway between the starting position and the circular target at the right. The cursor is displayed as multiple alpha-blended copies, simulating a motion blur, to compensate for the screen's refresh rate being lower than the mouse's reporting rate. *Left bottom:* The cursor inside the target.

Task and Stimuli

A discrete target acquisition task was performed by participants using either traditional *pointing*, or *tossing*. Each trial involved moving a cursor from a starting position to a target that was located to the right of the starting position. The task was thus one dimensional — direction of motion was not varied.

At the start of each trial, the user first positioned the mouse at a location on the desk marked with tape, so that the physical starting position was approximately the same in each trial (this avoided the possibility that the user might run out of physical space in the middle of a trial and need to lift

and reposition the mouse). The user then pressed and held the mouse button to signal they were ready. The cursor then appeared at the starting location on the screen (and, in the case of the full experiment, this was followed by a pause called the *foreperiod*, used to prevent anticipatory movement initiation, as explained later). This was followed by the appearance of the target, after which the user began moving the mouse to acquire the target. The mouse button was held down throughout the trial, until the target was acquired. After target acquisition, there was a 1 second pause during which output was frozen to allow the user to see how close they got to the target, providing feedback to allow the user to adjust their performance in subsequent trials.

In the case of the *pointing* tasks, the physical displacement on screen of the cursor matched the physical displacement of the mouse, i.e. the gain or C:D ratio was 1, with no cursor acceleration.

In the case of the *tossing* tasks, displacements in the cursor corresponded directly to the velocity of the mouse. Hence, the x-coordinate of the cursor, measured with respect to the cursor's starting position, was proportional to the speed at which the mouse moved to the right. Accelerating the mouse toward the right caused the cursor to move to the right, whereas subsequently decelerating the mouse caused the cursor to move left, and if the mouse came to rest in a motionless position then the cursor returned to its starting position. The gain in this case was chosen so that the most extreme target to be acquired by the user (32 cm from the cursor's starting position on screen) would correspond to a mouse velocity of 1 m/s to the right; i.e., the C:D ratio was (1 m/s)/(32 cm) = 3.125 s^{-1}. Our own informal testing indicated that 1 m/s was close to the upper limit of limb movement, and [9] found a maximum limb speed of about 1.5 m/s.

For both *pointing* and *tossing* tasks, the location of the most extreme target (at 32 cm) was chosen so that the screen had a large margin of empty space to the right of the target, so that the cursor would remain visible in the event of overshooting.

Our design required the user to drag the mouse, i.e. with the mouse button held down, in trials for both pointing and tossing. This is because we feel that practical uses of tossing would likely require dragging, and we did not want our experimental comparison to give pointing an unfair advantage by allowing pointing to be done without dragging. (Previous research has shown that dragging results in a lower index of performance [30].)

For each of the tasks, we had to choose between presenting targets as either circles with a given size, or as crosshairs marking a point target with no size (Figure 3). Traditional pointing tasks are performed with a target with a given size, and participants are asked to acquire the target anywhere within its area. We could have given users the same goal in a tossing task using circular targets, however we were concerned that this would have only allowed us to measure an error rate indicating how often the user falls anywhere on the particular targets used in the experiment. For example, a large target might encourage the user to toss with less precision, preventing us from measuring their best possible performance. Ideally, we would instead prefer to ask users to toss as closely as possible to a given point target, and measure the resulting "best possible" distribution around the target. Such a distribution would provide us with more information, and should theoretically allow us to calculate the error rate resulting from choosing any given target size. However, to make the comparison between pointing and tossing fair, we wanted to use the same kind of targets in both tasks, and we did not know if using point targets in a traditional pointing task would result in behavior governed by Fitts' law. In the end, we made different choices of target type for the pilot and the full experiment, as explained later.

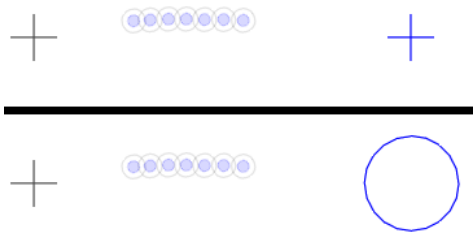

Figure 3. The cursor traveling from the starting position on the left toward a target on the right. There are two ways of displaying the target. *Top:* crosshair (i.e. point) target. *Bottom:* circular target.

Another issue we faced was how to complete a tossing action. In both pointing and tossing tasks, the goal was to *release* when the cursor was over, or as close as possible to, the target on screen. (In tossing, this meant the goal was to release when the mouse had reached the target velocity.) We decided to investigate two methods for release: *manual release*, where the user manually releases the mouse button to indicate that they have completed the desired physical motion, and *automatic release*, where the end of the input motion is determined automatically, regardless of when the user releases the mouse button.

To inform the design of a full experiment, including what kind of targets to use, we first performed a pilot experiment to gather some initial data.

Pilot Study

The pilot study involved 2 users, and involved both pointing and tossing, as well as circular targets and crosshair targets. There were four main conditions: Pointing, with Manual release, at crosshair targets (PM-+); Pointing, with Manual release, at circular targets with a 2 cm diameter (PM-O); Tossing, with Manual release, at crosshair targets (TM-+); and Tossing, with Automatic release, at crosshair targets (TA-+). The first two conditions were to test if target type would make a difference in traditional pointing, and the last two were to test the automatic release in tossing.

In tossing, automatic release occurred when the peak (i.e. maximum) velocity was detected in the input motion, and this peak velocity was used as the input velocity for the toss. (Note that peak velocity is also used by [4] for predictively

completing movements, however their work is aimed at improving pointing rather than tossing, and they do not examine the standard deviation of the peak velocity as a function of target distance as we do.) Detecting this peak in real time was done simply by waiting for the velocity to decrease repeatedly over a small number N of samples (with $N \approx 5$), and assuming that the velocity encountered N samples earlier is the peak for the whole motion. Such an automatic release might yield better performance than manual release for four reasons. First, requiring the user to release the button when they reach their desired velocity might interfere with the execution of their movement. Second, using the peak velocity of the user might make it easier for the user to achieve larger target speeds. Third, there is prior evidence that, when throwing darts, people release the dart close to the peak velocity of the throw (about 2-11 ms before peak velocity [38]). Fourth, the slope of the velocity curve is zero at the peak and therefore minimally sensitive to small errors in *when* the curve is sampled, whereas sampling the curve anywhere else where the slope is non-zero will result in a larger variance if the sample's timing (e.g. the user's timing of the button release) is slightly off.

To further illustrate the distinction between manual release and (automatic) release at peak velocity, Figure 4 shows a physical analog where both types of release are possible.

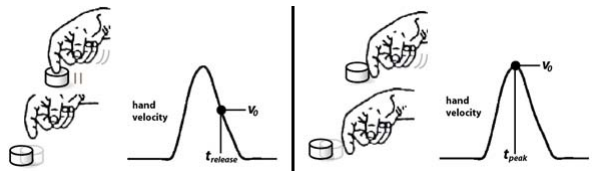

Figure 4. Two ways of physically accelerating and releasing a puck with velocity v_0. *Left:* The hand presses down and drags the puck to accelerate it, lifting off the puck to release it at some moment in time that may not coincide with the hand's peak velocity. *Right:* The hand pushes the puck forward to accelerate it. As soon as the hand decelerates (i.e. after reaching its peak velocity), contact is "automatically" lost with the puck.

For each of the 4 main conditions, there were 5 targets, displayed on-screen at distances of 2, 4, 8, 16, and 32 cm from the starting position. (In the case of tossing, each target distance corresponded to a target velocity, with the target at 32 cm corresponding to 1 m/s, and others corresponding to linearly smaller velocities. For example, the target at 2 cm corresponded to 6.25 cm/s.) For each main condition, users performed 25 practice trials (5 for each target distance) followed by 250 trials (50 repetitions for each target distance) in random order. Hence, there were 2 participants $\times$ 4 main conditions (PM-+, PM-O, TM-+, TA-+) $\times$ 5 target distances $\times$ 50 repetitions = 2000 trials in total.

For brevity, we only mention three results of the pilot study here. First, in tossing, the error in the final position was so large that users were rarely within a 1 cm radius of the targets, suggesting that even if 2 cm diameter circular targets had been used in the tossing tasks, it would have made little or no difference to the distribution of final positions. Thus, in the full experiment, for consistency with pointing tasks that traditionally use targets having some size, we used 2 cm circular targets in all conditions in the full experiment. A second result of the pilot was that the automatic release mechanism in tossing worked well, and seemed to work better than manual release. Thus, in the full experiment, we added a condition which tested a similar kind of automatic release condition for normal pointing, to make the main conditions symmetrical. Finally, a third finding from the pilot was that we observed that users sometimes started moving before a trial had begun, in anticipation of the target. To eliminate this in the full experiment, we added a random foreperiod, which we explain below.

Full Study

Participants

The full experiment involved 12 right-handed participants.

Conditions

The full experiment crossed the *technique* (Pointing or Tossing) and *release-type* (Manual or Automatic) variables, yielding 4 main conditions, abbreviated as PM, PA, TM, and TA. In all conditions, targets were circles with a 2 cm diameter. In the PA (pointing with automatic release) condition, the automatic release occurred when the mouse's speed dropped below 50 mm/s for 50 ms.

As mentioned earlier, the data collected in the pilot study indicated that users often began moving the mouse before the start of the trial. In other words, users would initiate movement slightly before pressing the mouse button down, in anticipation of the target that would appear. Thus, the mouse would already have momentum to the right when the trial began. For the full experiment, we prevented such anticipatory processes by imposing a pause of random duration, called a *foreperiod*, between the mouse button down press, and the appearance of the target. The foreperiod lasted between 600 and 1400 ms, and during this period no mouse motion was allowed. At the end of the foreperiod, the target would appear, and users were then allowed to begin moving toward it.

For each of the 4 main conditions, users were given the following instructions:

> "Your goal will be to quickly bring a cursor inside circular targets. If you are unable to bring the cursor inside the target, try to be as close as possible."

and then performed 25 practice trials (5 for each target distance) followed by 125 trials (25 repetitions for each target distance) in random order. Hence, there were 12 participants $\times$ 2 techniques (Pointing and Tossing) $\times$ 2 release-types (Manual and Automatic) $\times$ 5 target-distances (2, 4, 8, 16, and 32 cm) $\times$ 25 repetitions = 6000 trials in total. As in the pilot study, a target distance of 32 cm corresponded to a velocity of 1 m/s in the tossing conditions.

In choosing the order of presentation of conditions, we decided that the two pointing conditions should always appear before the two tossing conditions, because the pointing con-

ditions were much more familiar to users. However, within each pair of conditions we varied the ordering, resulting in a total of 4 orderings (PM,PA,TM,TA), (PM,PA,TA,TM), (PA,PM,TM,TA), (PA,PM,TA,TM) that were each assigned to one quarter of the participants.

Results and Discussion of Full Study

Movement Time

For all conditions, movement time MT was measured between the start of the movement (defined to occur when velocity went above 30 mm/s for 50 ms) and the end of the movement (defined to occur when velocity dropped below 30 mm/s for 50 ms — these thresholds are based on [11]). Thus, movement time did not include reaction time. Note that, in the TA condition, the time between onset of movement and automatic release of target (occurring at the peak velocity) is *less* than the MT we defined, however our MT better reflects the total temporal cost of the user's action, especially in the context of repeated target acquisitions (where the user would have to terminate the movement for one toss before initiating another toss).

Figure 5 shows MT broken down by target distance for each condition. Analysis of variance revealed two main effects on MT: one for technique ($F_{1,11} = 154.24$, $p < .001$), meaning that the tossing conditions required significantly less time than the pointing conditions, and another main effect for target-distance ($F_{4,44} = 524.64$, $p < .001$).

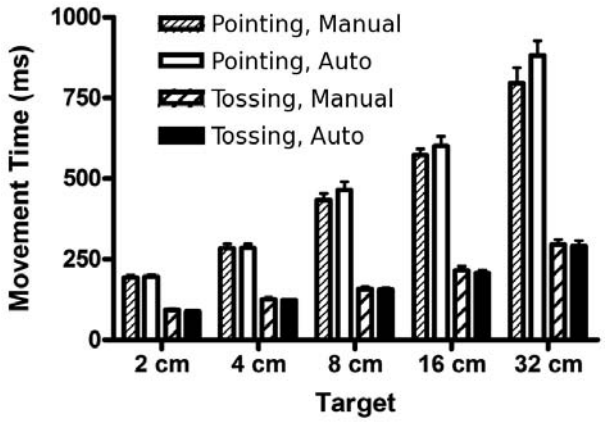

Figure 5. The movement time for each target distance, by condition. Error bars show the standard error of the mean (i.e., $\sigma/\sqrt{N}$). In the tossing conditions, each target distance corresponds to a target velocity (2 cm corresponds to 6.25 cm/s, 32 cm corresponds to 1 m/s, etc., in a linear fashion).

For each of the main conditions, the movement times were fit to a straight line with linear regression with respect to index of difficulty (Table 1). As is often done in experiments involving Fitts' law, the regression was performed on aggregated data, i.e. on average movement times, so only 5 data points were used for each regression. Table 1 completely specifies the 5 data points used in each regression, as can be verified by the reader. We note that the correlation coefficient r is at least 0.99 in all conditions. Consulting a standard table of significance levels, we find that any r value greater than 0.9, computed from 5 data points, is significant ($p < .05$). Thus, Fitts' law can be used to model average movement time in all conditions.

	Movement Time MT in ms, for each Index of Difficulty $ID = \log_2(\text{(Target Distance in cm)}/2+1)$ in bits					Linear Regression of MT with respect to ID		
	1.000 bits	1.585 bits	2.322 bits	3.170 bits	4.087 bits	slope m_{MT}	intercept b_{MT}	r
PM:	192.947	284.017	433.801	572.673	795.101	193.276	−14.505	0.997
PA:	195.493	285.087	465.386	600.402	880.844	218.708	−46.642	0.993
TM:	92.599	126.446	156.841	215.706	295.337	64.360	20.806	0.991
TA:	86.705	122.725	155.508	206.173	290.413	63.796	17.099	0.990

Table 1. Average movement times MT in ms for each target distance, by condition, and the result of linear regressions. The conditions PM, PA, TM, TA correspond to Pointing with Manual and Automatic release, and Tossing with Manual and Automatic release, respectively.

Error in Final Position

For each target distance d, there is a distribution of final positions achieved by the user with mean μ and standard deviation σ. We characterize the distribution by its constant error $CE = \mu - d$ (i.e., a negative CE indicates systematic undershoot) and variable error $VE = \sigma$ (a synonym for the standard deviation). Tables 2 and 3 show the values of CE and VE for all conditions, and the results of linear regressions, now with respect to target distance rather than index of difficulty ID. (We also performed regressions of CE and VE with respect to ID, but these result in r values closer to zero.) Again, regressions were performed on aggregated data, with 5 data points for each regression, hence all r values greater than 0.9 (or less than -0.9) are significant ($p < .05$).

	Constant Error CE in cm, for each Target Distance in cm					Linear Regression of CE with respect to Target Distance		
	2 cm	4 cm	8 cm	16 cm	32 cm	slope m_{CE}	intercept b_{CE}	r
PM:	0.0209	−0.1095	−0.0440	0.0839	−0.0749	−0.0005	−0.0191	−0.071
PA:	−0.1154	−0.3666	−0.3304	−0.4214	−1.2714	−0.0348	−0.0697	−0.951
TM:	−0.0810	−0.8796	−2.3695	−5.1826	−7.9906	−0.2631	−0.0378	−0.983
TA:	0.5751	0.5326	−0.1025	−2.4999	−4.6407	−0.1847	1.0638	−0.986

Table 2. Constant error CE in cm for each target distance, by condition, and the result of linear regressions. Most of the CE values are negative, indicating undershoot (i.e. the mean of the distribution is to the left of the target center). Not surprisingly, CE is small in the pointing conditions. In the tossing conditions, CE increases negatively with target distance, in a roughly linear fashion (r somewhat close to -1), and is reduced in the automatic release condition.

	Variable Error VE in cm, for each Target Distance in cm					Linear Regression of VE with respect to Target Distance		
	2 cm	4 cm	8 cm	16 cm	32 cm	slope m_{VE}	intercept b_{VE}	r
PM:	0.2911	0.3188	0.3516	0.3567	0.3135	0.0003	0.3227	0.131
PA:	0.4986	0.7247	1.1395	2.1059	3.4555	0.0992	0.3552	0.997
TM:	1.3773	1.6560	2.6330	4.6122	8.0084	0.2247	0.8709	0.999
TA:	1.1587	1.1038	1.5511	2.9463	5.4755	0.1508	0.5767	0.995

Table 3. Variable error VE in cm (i.e., the standard deviation) for each target distance, by condition, and the result of linear regressions. Not surprisingly, VE is small when pointing with manual release. In the tossing conditions, VE increases with target distance, in a linear fashion (r close to 1), and is reduced in the automatic release condition.

For CE, analysis of variance revealed three main effects: technique ($F_{1,11} = 46.94$, $p < .001$), release-type ($F_{1,11} = 5.94$, $p < .05$), and target-distance ($F_{4,44} = 44.73$, $p < .001$). For VE, there were two main effects: technique ($F_{1,11} = 77.23$, $p < .001$), and target-distance ($F_{4,44} = 51.39$, $p < .001$). It follows that pointing had a significantly lower CE and VE than tossing. Among the interactions found, two

2-way interactions were notable: a technique by release-type interaction ($F_{1,11} = 14.70$, $p < .01$) for CE, and a technique by release-type interaction ($F_{1,11} = 33.44$, $p < .001$) for VE. These interactions reflect the fact that automatic release made *pointing* significantly *worse* in terms of CE and VE, whereas automatic release also made *tossing* significantly *better* in terms of CE and VE.

Figures 6-9 depict the distribution of final positions as a function of target distance for each of the 4 main conditions, and Figure 10 shows a theoretical case. In all five figures 6-10, the axes are set to the same scale, and the height of the error bars is set to 4.133 standard deviations (i.e. $\pm\sqrt{\pi e/2}\sigma \approx \pm 2.066\sigma$ around the mean, or a width of 4.133 times the VE values in Table 3). This width encompasses

$$\mathrm{erf}(\sqrt{\pi e/2}/\sqrt{2}) \approx 96\%$$

of the distribution, corresponding to an error rate of about 4%. 4.133σ is also the effective width W_e of the targets, as defined and explained in [29].

Figures 6-9 contain the following 3 kinds of lines: (1) a dotted line $y = x$ passing through the center of the targets, (2) a solid line $y = x + m_{CE}x + b_{CE}$ showing the line that best fits the centers of the distributions, where m_{CE} and b_{CE} are the slope and intercept taken from the appropriate linear regression in Table 2, and (3) dashed lines $y = x + m_{CE}x + b_{CE} \pm 2.066(m_{VE}x + b_{VE})$ showing the effective target width, where m_{VE} and b_{VE} are similarly taken from the regressions in Table 3.

Not surprisingly, in the pointing with manual release (PM) condition, the CE and VE are small and approximately constant across target distances (Figure 6). This is because the user can reduce the error in the final location by simply performing corrective submovements as required, prolonging the movement time. In the other three conditions, however, especially TM and TA, the correlation coefficients in Tables 2 and 3 are high, and the corresponding Figures 7-9 indicate there is a linear relationship between CE and target distance, and also between VE and target distance.

Unfortunately, the VE in the tossing conditions is so high that the error bars in Figures 8 and 9 overlap vertically to a large degree. This means that, in a practical setting, tosses by the user toward the same 5 targets would be misinterpreted by the computer much more often than 4% of the time. For reliably distinguishable targets (i.e. with an error rate of 4% or less), in the TM condition, there appears to be room for only two targets in the velocity domain: one at 2 cm (i.e. 6.25 cm/s), and one at 32 cm (i.e. 1 m/s). In the TA condition, however, the error is smaller, and there appears to be room for *three* reliably distinguishable targets, as shown in Figure 10.

Indices of Performance of Pointing and Tossing

Of the four techniques tested in the full experiment, the most successful pointing and tossing techniques were PM and TA, respectively. There are a couple of issues to consider to inform a proper comparison of these two techniques.

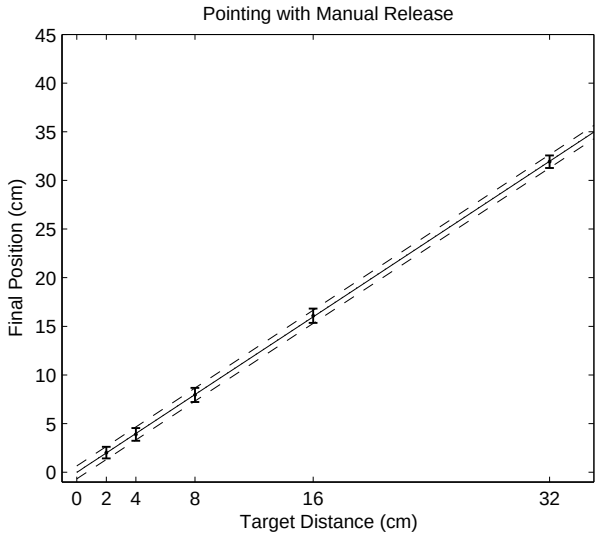

Figure 6. Performance in the pointing with manual release condition. Not surprisingly, the distributions of final positions are tightly clustered around the line $y = x$, because in general users were able to fall within each target regardless of target distance.

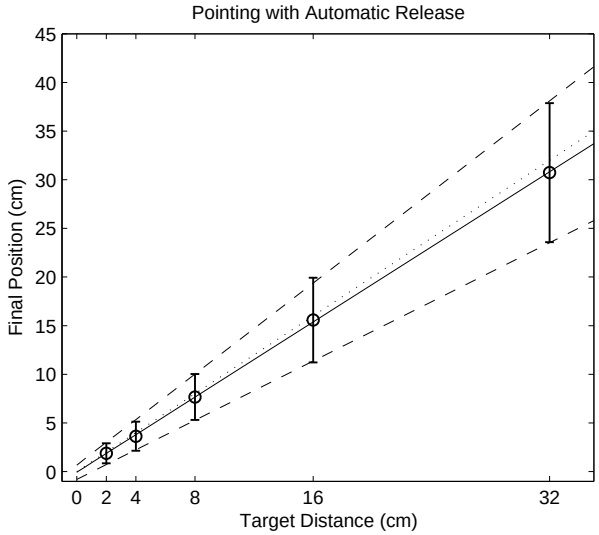

Figure 7. Performance in the pointing with automatic release condition. A small negative constant error (shown by the solid line appearing below the dotted line) is seen, as well as a variable error that grows with target distance.

First, as shown in Figure 5 and Table 1, tossing with automatic release can be performed faster than pointing with manual release, however this comes at the cost of a higher constant error CE and variable error VE (Tables 2 and 3). Thus, there is a tradeoff between the PM and TA techniques that complicates comparison.

The second, more subtle issue to consider, is related to the *target distance* that was manipulated as an independent vari-

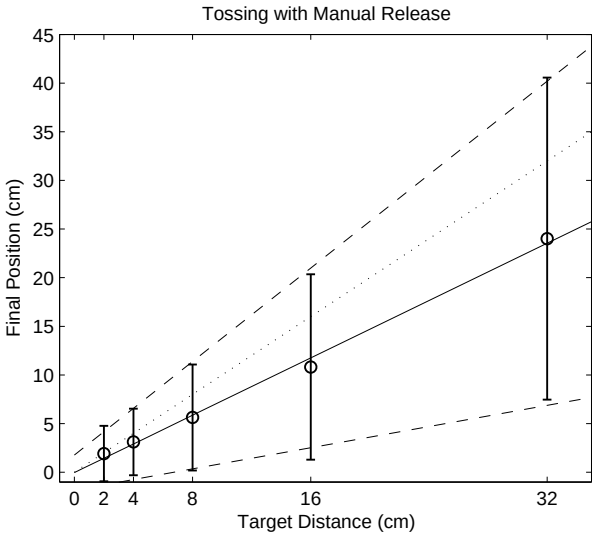

Figure 8. Performance in the tossing with manual release condition. The effective widths of the targets at 2 and 32 cm do not overlap in space, however there does not seem to be room to add a 3rd non-overlapping target in between them.

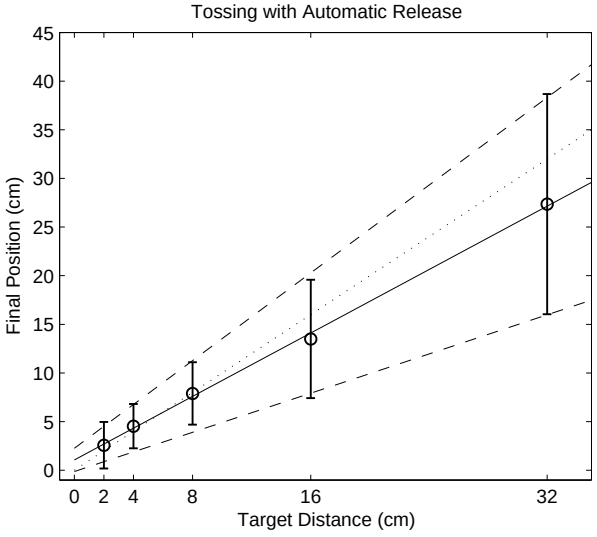

Figure 9. Performance in the tossing with automatic release condition. The effective widths of the targets at 2 and 32 cm do not overlap in space, and furthermore there appears to be room to fit a 3rd non-overlapping target in between them (see Figure 10).

able. As already mentioned, in the pointing conditions, the C:D ratio was 1, so the target distance was the same on-screen and in the physical domain. For convenience, the same on-screen target distances were used in the tossing conditions, however in the physical domain these target distances corresponded to *velocities* to be acquired by the user. The fact that an on-screen target distance of 2 cm corresponded to a velocity of 6.25 cm/s is simply a consequence of the C:D ratio used in our tossing conditions. Had we used a different C:D ratio for tossing, but kept the same target velocities, this would have changed the on-screen target dis-

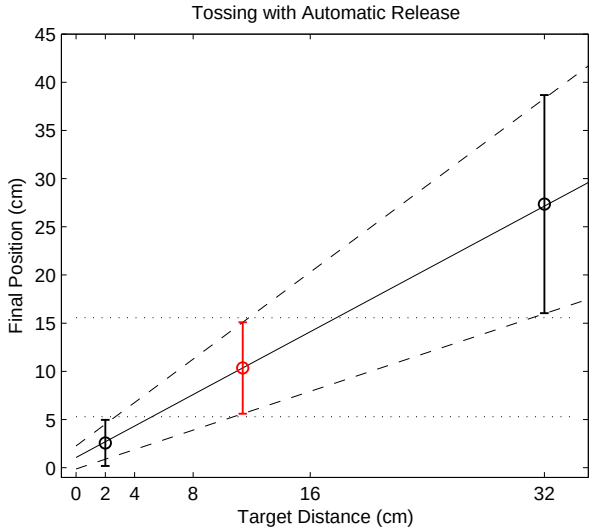

Figure 10. Theoretical performance in the tossing with automatic release condition. Two of the experimentally measured distributions are shown (the ones at target distances of 2 and 32 cm from Figure 9) in black. In addition, a theoretical distribution, found through interpolation, is shown in red. The dotted lines clearly show that the error bars do not overlap in space, hence there appears to be room for 3 targets with non-overlapping effective widths.

tances for tossing. Thus, comparisons between techniques at each target distance, such as those in Figure 5 and Tables 1-3, are arguably misleading.

Therefore, to properly compare the PM and TA techniques, we calculate their respective index of performance IP (i.e. their bandwidth, or throughput). To do this, we first need to properly account for the large VE with tossing by using the effective width W_e of the targets for each technique [29]. In addition, because of the large CE with tossing, we also calculate the effective distance D_e of each target for each technique. Table 4 shows these effective quantities and Table 5 shows the resulting regressions.

PM	Target Distance	2.000	4.000	8.000	16.000	32.000
	CE	0.021	−0.110	−0.044	0.084	−0.075
	D_e = Target Distance + CE	2.021	3.891	7.956	16.084	31.925
	VE	0.291	0.319	0.352	0.357	0.314
	$W_e = 4.133\,VE$	1.203	1.318	1.453	1.474	1.296
	$ID_e = \log_2(D_e/W_e + 1)$	1.422	1.983	2.695	3.574	4.680
	MT	192.947	284.017	433.801	572.673	795.101
TA	Target Distance	2.000	4.000	8.000	16.000	32.000
	CE	0.575	0.533	−0.103	−2.500	−4.641
	D_e = Target Distance + CE	2.575	4.533	7.898	13.500	27.359
	VE	1.159	1.104	1.551	2.946	5.476
	$W_e = 4.133\,VE$	4.789	4.562	6.410	12.176	22.629
	$ID_e = \log_2(D_e/W_e + 1)$	0.621	0.995	1.158	1.076	1.143
	MT	86.705	122.725	155.508	206.173	290.413

Table 4. Effective distance D_e, effective width W_e, effective index of difficulty ID_e, and related quantities for the pointing with manual release (PM) and tossing with automatic release (TA) conditions.

To find the index of performance IP for each technique, one approach is to define $IP = 1/b$ where b is the slope of the regression line. This results in $IP = 1/(184.438\ \text{ms/bit}) = 5.42\ \text{bits/s}$ for the PM condition, and $IP = 1/(253.773\ \text{ms/bit}) = 3.94\ \text{bits/s}$ for TA. However, the $IP = 1/b$ definition has the

	Linear Regression of MT with respect to ID_e		
	slope	intercept	r
PM:	184.438	−73.798	0.999
TA:	253.773	−81.182	0.707

Table 5. Linear regressions of MT using effective index of difficulty.

disadvantage of not taking into account the intercept a of the regression. Furthermore, we note that in Table 4, all the ID_e values for TA are below 1.2 bits (making it all the more important to take a into account), and the r value for TA in Table 5 is only about 0.7. Thus, it is doubtful that the slope of the regression line for TA will, by itself, accurately reflect the bandwidth of the technique.

An alternative definition of the index of performance is $IP = \overline{ID_e}/\overline{MT}$, where $\overline{ID_e}$ and $\overline{MT}$ are the average effective index of difficulty and average movement time, respectively (see [41, 39] for discussion of these and other approaches to calculating index of performance). This definition implicitly takes into account the intercept a, but has the disadvantage that it depends on the range of (distance,width) target values used in the experiment. However, as already mentioned, the target distances were chosen to cover most, if not all, of the full range of reasonable tossing speeds, thus we expect this definition to yield more meaningful values of IP. Performing the calculation, in the PM condition, $IP = (2.871 \text{ bits})/(455.708 \text{ ms}) = 6.30 \text{ bits/s}$, and in the TA condition, $IP = (0.999 \text{ bits})/(172.305 \text{ ms}) = 5.80 \text{ bits/s}$. The difference in IP values is less than 10%.

Thus, although tossing is faster than pointing, these calculations indicate that the loss of precision with tossing is such that information cannot be expressed quite as efficiently with tossing as it can be with pointing. Of course, our experiment only tested one set of conditions, and future work may find different results. However, assuming that tossing has indeed a smaller bandwidth than pointing, it would seem unwise to design interfaces where input is done exclusively through tossing, without any pointing. However, a case can still be made for interfaces that use a mixture of pointing and tossing for input. In a later section of this paper, we discuss mechanisms by which a user could switch between normal pointing and tossing. Performing such a switch is one way of telling the system that the user is interested in a different class of targets or a different class of actions, and would allow the user to perform fast, imprecise, distant displacements with tossing whenever desired. More generally, an interface could also be designed to support switching between normal pointing and one of several other possible input schemes (such as object pointing [22], a bubble cursor [21, 10], or normal pointing with a larger gain). However, having tossing as the 2nd mode has the advantage that tossing has a familiar, natural metaphor in real life, and may be easier to understand for users than other input schemes.

ERROR RATE AS A FUNCTION OF NUMBER OF TARGETS

Although Figure 10 indicates that there should be room for a 3rd target, this is based on targets having an effective width of 4.133 standard deviations (i.e. $\pm\sqrt{\pi e/2}\sigma \approx \pm 2.066\sigma$ around the mean), corresponding to an error rate of

$$1 - \text{erf}(\sqrt{\pi e/2}/\sqrt{2}) \approx 4\%$$

If a higher error rate is tolerable (or a lower error rate is required), or the experimental conditions are modified, then the number of allowable targets may change. Hence, we now show how to calculate, in general, the optimal placement of N of targets, and the estimated error rate that should result from this.

Let x_1 and x_N be target distances within the range of distances that a user can reasonably achieve, with x_1 near the lower end, and x_N near the upper end. We wish to choose intermediate target distances $x_2, \ldots, x_{N-1}$ that minimize the overlap between the distributions of final positions and thus minimize the error rate. Let $b_{CE}, m_{CE}, b_{VE}, m_{VE}$ be slope and intercept regression constants for constant error and variable error, as in Tables 2 and 3. To simplify our analysis, we initially assume that $b_{CE} = b_{VE} = 0$. Figure 11 illustrates. For each target distance x, the distribution of final positions is centered around the mean $\mu = x + m_{CE}x$, corresponding to a point on the solid line in Figure 11 whose equation is $y = x + m_{CE}x$. The portion of a target's distribution that does not overlap (in y) with other targets corresponds to $y = \mu \pm z\sigma$, where z is a z-score. These non-overlapping portions are delimited by the dashed lines in Figure 11, whose equations are $y = x + m_{CE}x \pm z(m_{VE}x)$. (In Figures 6-10, the dashed lines correspond to a z-score of $z = 4.133/2$, whereas z has an unknown value in Figure 11.) As can be seen, the non-overlapping portions of the distributions form the vertical segments of a staircase pattern.

Given N, x_1, x_N, we would like to calculate z and the associated error rate. The staircase pattern in Figure 11 results in the values $x_1, \ldots, x_N$ forming a geometric sequence. (To see this, consider the triangles bounded by the horizontal axis $y = 0$, the (dotted) vertical lines $x = x_i$ for $i = 1, \ldots, N$, and any of the (solid or dashed) diagonal lines in the figure, and note how these triangles are similar.) The scale factor of the geometric sequence is

$$s = \frac{x_2}{x_1} = \ldots = \frac{x_N}{x_{N-1}} = \sqrt[N-1]{\frac{x_N}{x_1}}$$

Next, consider the value of y^*, at the boundary between the last two targets, located between y_{N-1} and y_N. The value of y^* can be expressed either as that of y_{N-1} *plus* half a step (of appropriate size), or as y_N *minus* half a step (again, of appropriate size). To be precise,

$$y^* = y_{N-1} + z(m_{VE}x_{N-1}) = y_N - z(m_{VE}x_N)$$

Expanding the expressions for y_{N-1} and y_N,

$$(1 + m_{CE})x_{N-1} + z(m_{VE}x_{N-1}) = (1 + m_{CE})x_N - z(m_{VE}x_N)$$

Substituting in $x_N = s\, x_{N-1}$ and solving for z, we find

$$z = \frac{(1 + m_{CE})}{m_{VE}} \frac{(s-1)}{(s+1)}$$

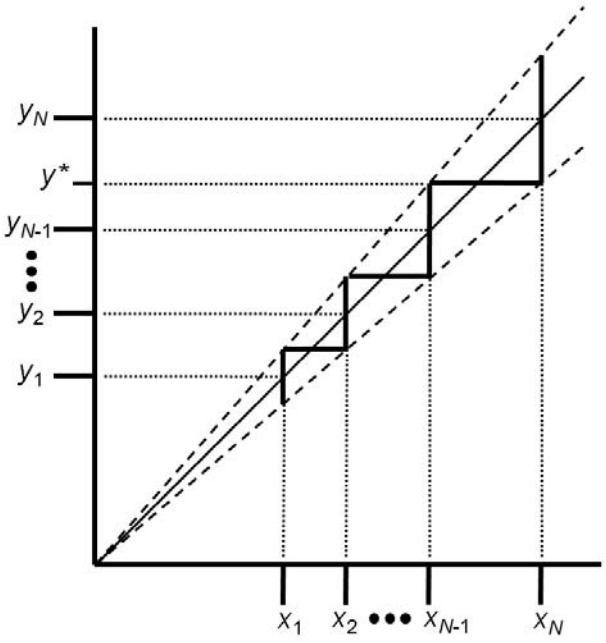

Figure 11. A theoretical "staircase" pattern formed by having N targets to toss to (in this case, $N = 4$). The dashed lines delimit the vertical segments of the "stairs", i.e. the portions of the distributions that do not overlap. Any tosses that fall outside the dashed lines are errors, because they will be misinterpreted as tosses to a different target. In other words, each target is effectively a vertical segment centered at some y_i, and the user must toss within the vertical segment.

Finally, assuming normal distributions, the error rate that should result from this non-overlap is $1 - \mathrm{erf}(z/\sqrt{2})$.

If we now allow for non-zero values of b_{CE} or b_{VE}, it turns out that the only calculation that changes is of the scale factor, which becomes

$$s = \sqrt[N-1]{\frac{x_N + b_{VE}/m_{VE}}{x_1 + b_{VE}/m_{VE}}}$$

The values of z and of the error rate are calculated using the same expressions as before.

Taking the experimental results in the tossing with automatic release condition, we set $x_1 = 2$, $x_N = 32$, $m_{CE} = -0.1847$, $m_{VE} = 0.1508$, and $b_{VE} = 0.5767$. We can then find the theoretical error rate as a function of the number of buttons N, as reported in Table 6.

N	s	z	error rate
3	2.480	2.299	2.148 %
4	1.832	1.589	11.214 %
5	1.575	1.207	22.743 %

Table 6. Theoretical prediction of the error rate, and other parameters, as a function of the number of discrete tossing targets N, for tossing with automatic release.

Although we have not tested the $N = 3$ case with an appropriately positioned intermediate target, both Figure 10 and the above calculations indicate that users should be able to toss toward 3 targets in the same direction with an error rate below 4%.

The theoretical error rate of 11% for $N = 4$ is rather high, however it may be acceptable in certain applications where the cost of an error is low. It is also possible that with a different input device, and/or a better mechanism for performing automatic release, the error rate for $N = 4$ might be reduced.

INTEGRATING TOSSING IN STANDARD USER INTERFACES

For practical applications, an interesting design question is how to allow users to use tossing without interfering with already existing mouse actions (pointing, clicking, and normal dragging) in status quo user interfaces. If tossing is only possible when moving with the mouse button (or stylus tip) held down, then clearly tossing will not interfere with normal pointing and clicking (i.e. press-releasing). The only problem that would remain is disambiguating normal dragging from tossing. Furthermore, of all the objects (icons, windows, etc.) in an interface that might be dragged, the designer may deem some of these objects as "non-tossable". Initiating a drag on such objects, then, would never be interpreted as a toss. Objects that *can* be tossed might be indicated as such by a special highlighting color or cursor shape when the cursor is over them. To allow both tossing and normal dragging of such "tossable" objects, various approaches are possible for disambiguating the two actions. The approach taken in [37, 2] is to use the velocity at release time to disambiguate dragging (a release velocity near zero) and tossing (a release velocity above some threshold). However, this approach cannot be used in conjunction with automatic release at peak velocity, which our experiment showed to be more precise than manual release. The following are some alternative approaches that assume tossing is performed with automatic release:

1. To toss an object, press over it and drag faster than some minimum threshold speed. Dragging slower than the threshold is interpreted as a normal drag. (Unfortunately, this could prevent users from dragging an object to a desired location as quickly as they would like to, and could also increase the lowest target speed for tossing, thus reducing the available range $[x_1, x_N]$ for discrete tossing targets.)

2. To toss an object, press over it and drag. To drag an object normally, press-pause over it (i.e. press and then wait for some small span of time, such as 300ms), waiting for some visual confirmation (such as a change in cursor shape) that the object will not be tossed, and then drag normally.

3. To toss or normally drag an object, press over it and drag. During dragging, the system continually looks for velocity peaks in the user's motion, and highlights the location that the object would be tossed to if the toss is accepted by the user. (Alternatively, the object could be tentatively moved to the final location, and would spring back if the user declines the offered toss.) To accept the offered final position of a toss, the user releases the mouse button immediately. To ignore or decline a tossed position, the

user can continue dragging to new locations, and can also dwell over a location (say, for 300ms) to decline the most recently offered toss and then release to drop the object at the position dragged to. This approach is similar to the previous one, except that the pause is moved to the end of the normal drag action.

4. To toss an object, press over it and drag. To drag an object normally, press-release (or press-release-press-release, as in a double click) over it to "attach" it to the cursor, then move to the destination location (without holding down the button), and press-release to "drop" the object. (This approach has the disadvantage that press-release and double clicking are often already used for selecting and opening an object, respectively.)

5. To toss an object, press over it and drag. To drag an object normally, press-release-press (i.e. one-and-a-half clicks) over it to grab it, drag to the destination location (while holding down the button), and release to drop the object.

6. Use one button for normal dragging, and another button (or the same button with a keyboard modifier key) for tossing. (This approach has the disadvantage that it is more difficult to use with a stylus, and it requires using two hands or may require using a button that is already mapped to a function such as the common right-click menu.)

7. Use a virtual button or widget to switch between normal dragging and tossing. For example, the *trailing widget* in [19] could be used to perform such switching, generally remaining close to the cursor for easy access but also usually out of the user's way.

Approach #6 above is an obvious one to take when using a mouse with more than one button and/or a keyboard. The fact that the right mouse button may already be mapped to a right-click menu need not be a major obstacle, since a right-press-release (or right-press-pause) could still open the menu normally, while a right-press-drag would initiate a toss (or a normal drag, as the case may be).

Approach #5 listed above may be the most viable, since it can be used with a stylus without any extra buttons, does not require slowing the user down with any imposed pauses, and makes use of a one-and-a-half click action that, as far as we know, is not currently mapped to an operation in any existing interface.

Tossing within an interface could be done toward a continuous range of locations, or toward a set of discrete targets. Examples of cases where it could make sense to have a set of discrete tossing targets defined include (1) tossing icons into windows or into folders, (2) tossing thumbnails of photos into piles or groups of photos, (3) tossing windows toward the edges of the screen or toward empty spaces[2] located between other windows, (4) tossing the thumb of a scrollbar toward chapter boundaries within a document or toward the first or last page of the entire document, (5) tossing the cursor itself toward widgets (buttons, menus, tool palettes, etc.) that it can interact with, as a shortcut to pointing.

Since Figure 10 and Table 6 indicate that only 3 discrete targets could be reliably tossed to in the same direction, we must consider what to do if there are more than 3 discrete targets in a given direction. One possibility is, when the user moves their cursor over a tossable object, the system could highlight the 3 most likely tossing targets in each direction (where likelihood could be a function of past usage and/or of the compatibility of object and target types (see §4.1 of [5] for related strategies for determining candidate drop sites)). If the user tosses the object, the "slow", "medium", and "fast" tossing speeds would be mapped to the closest, intermediate, and farthest highlighted targets in the appropriate direction. This approach would mean that if the user wishes to move the object to a target that isn't highlighted, they would probably drag it normally instead of tossing.

Another possible approach is to highlight the 3 closest targets in each direction, and to allow the user to *re-toss* an object one or more times before releasing the mouse button. In effect, the first velocity peak detected would cause a toss, but also rigidly "connect" the cursor to the object (now possibly far away), enabling additional tosses to be performed, until the button is released at which point the connection would disappear. Such re-tossing would be useful not just for tossing farther than the 3 closest targets, but also for correcting an erroneous toss.

In the superflick [37] technique, the tossed object is released when the user lifts the stylus tip (equivalent to releasing a mouse button), at which point the final position of the toss is immediately displayed, and an animation also plays out, showing the motion of the object to its final position. During this animation, the user may click and drag a second time to displace the final position in a relative manner, again as if there were a rigid connection between the final toss location and the pointing device. (Also, during such corrective displacements, the C:D ratio is temporarily changed to 1:4.) Thus, tossing and correcting with superflick requires *two* press-drag-release actions. In contrast, if an automatic release technique were used, tossing could be performed by press-dragging, and then corrections could be performed (either by re-tossing, as describing in the previous paragraph, or by displacing the final position with relative motions) all within a *single* drag. Such a technique would improve the precision of the initial toss, and also have the advantage that the user would not have to wait for an animation to terminate before clicking and dragging to initiate some other action. Although an animation could still be played out, the user could release immediately if they see that no correction is necessary, and move on to some other action before the animation has finished. An additional improvement to the superflick technique would be to change the C:D ratio during corrections to k:v_0, where k is some constant and v_0 is the release velocity. Thus, the C:D ratio would scale with the release velocity, rather than being fixed at 1:4. This is motivated by the fact that our experiment found that the standard

[2]Such empty spaces could be automatically found using the technique in [6].

deviation in final tossed position varies linearly with v_0, thus faster tosses will usually require larger corrections.

Finally, after a toss is performed, there are a few forms of visual feedback that could be displayed. [37] displays the final location of the toss as well as an animation of the tossed object moving toward that location. To save time, an alternative form of feedback that might be easier to see when played out at high speed would be a visual "smoke trail" or semi-transparent streak toward the final location, that fades out over a short period of time. Also of potential use would be visual feedback indicating the peak velocity achieved, with respect to the "slow", "intermediate", and "fast" speeds, perhaps on a small linear scale that is popped up for a moment. This would allow the user to see if their toss was nearly (or only slightly) misinterpreted, enabling them to calibrate their speed in subsequent tosses. Finally, to allow easy undoing of unintentional tosses, a small virtual button could be temporarily popped up close to the cursor's position. Moving toward the button and clicking on it would undo the toss, and moving away from it (which would usually happen naturally if the button is ignored) would cause the virtual button to disappear.

CONCLUSIONS

Our experimental study confirmed that tossing is significantly faster than pointing, but is also significantly less precise. The movement time associated with tossing was found to increase linearly with the index of difficulty of the targets, and the constant error and variable error were found to increase linearly with the distance to the target. We found no evidence that users can trade off movement time for precision during tossing, as they can in traditional pointing. From these results, we calculated the index of performance associated with tossing, and found it to be less than 10% lower than the index of performance associated with pointing. We also showed how to calculate the optimal arrangement of a given number of targets in velocity space, and calculate the predicted error rate that results from such an arrangement. We predict that 3 optimally positioned targets along a single direction would allow for tossing with an error rate below 4%.

We also demonstrated that tossing precision is significantly improved by automatically releasing at the peak velocity, without requiring a longer movement time. On the other hand, automatically releasing during traditional pointing when the velocity drops below a threshold significantly worsened precision.

Finally, we have discussed several approaches for integrating tossing within user interfaces alongside normal pointing and dragging.

FUTURE DIRECTIONS

One obvious direction for future work would be to test the error rates that we predict for arrangements of three or four discrete targets in the velocity domain, and to test this for tossing movements in different directions. We have also discussed ways that the superflick technique [37] could be improved to allow tossing with correction in a single press-drag-release, using automatic release, and using a C:D ratio during correction that depends on the release velocity. This improved superflick technique could also be tested experimentally.

We also suspect that the automatic release technique that we described for *pointing*, which was shown to worsen performance, might be improved if the threshold speed used to determine when the user has stopped was made proportional to the peak velocity. Such automatic detection of the endpoint of a pointing motion might be useful in interfaces where the user cannot perform explicit clicks, e.g. with eye trackers, and might prove superior to using a dwell time threshold for "clicking".

Future work could also test the effect of different input devices (e.g. mouse versus stylus) as well as different feedback. For example, showing an animation of a virtual object decelerating to a final position, as done in [37], would be more congruent with our everyday real world experience, and might allow users to better adapt their subsequent motions. Different friction models could be tested for driving such animation, such as the kinetic and fluid friction we discussed. (Also of interest might be a friction model that results in distance traveled D being a logarithmic function of release velocity v_0, so that the magnitude of the variable error in D is independent of D, "evenly distributing" error across distance.) Haptic feedback might also be useful for improving tossing performance, to provide, for example, cues about the weight of a virtual object, and/or signal when release has occurred.

It could also be worthwhile to explore designs for widgets based on velocity input. For example, "speed-dependent marking menus" have been proposed [35] where options are selected with strokes of varying direction and speed (analogous to the pressure marking menu in [36] that is sensitive to pen pressure, or bullseye menus [20] that are sensitive to stroke length, in addition to direction). Taking this idea further, researchers might try to design a graphical interface that uses tossing and velocity-based actions as much as possible, perhaps creating a "Tossy" interface in a spirit similar to CrossY [3] which is designed around crossing actions.

ACKNOWLEDGEMENTS

This work was performed while Dragicevic and McGuffin were members of the Dynamic Graphics Project (DGP) lab at University of Toronto, and was supported by NSERC. Thanks to Andres Palomino for help collecting the experimental data. Thanks also to Anand Agarawala for suggesting the idea of a Tossy UI to us, and to Ravin Balakrishnan, Joe Laszlo, other members of the DGP lab, and Gordon Kurtenbach for valuable discussion.

REFERENCES

1. A. Agarawala and R. Balakrishnan. Keepin' it real: Pushing the desktop metaphor with physics, piles and the pen. In *Proc. CHI*, pages 1283–1292, 2006.
2. D. Aliakseyeu, P. Irani, A. Lucero, and S. Subramanian.

Multi-flick: An evaluation of flick-based scrolling techniques for pen interfaces. In *Proc. CHI*, 2008.

3. G. Apitz and F. Guimbretière. CrossY: A crossing-based drawing application. In *Proc. UIST*, pages 3–12, 2004.
4. T. Asano, E. Sharlin, Y. Kitamura, K. Takashima, and F. Kishino. Predictive interaction using the Delphian desktop. In *Proc. UIST*, pages 133–141, 2005.
5. P. Baudisch et al. Drag-and-pop and drag-and-pick: techniques for accessing remote screen content on touch- and pen-operated systems. In *Proc. Interact*, pages 57–64, 2003.
6. B. A. Bell and S. K. Feiner. Dynamic space management for user interfaces. In *Proc. UIST*, 2000.
7. A. Bezerianos and R. Balakrishnan. The vacuum: Facilitating the manipulation of distant objects. In *Proc. CHI*, pages 361–370, 2005.
8. L. G. Carlton. Visual processing time and the control of movement, 1992. In L. Proteau and D. Elliott (eds.) Vision and Motor Control (pp. 3–31), Amsterdam: North Holland.
9. G. Casiez, D. Vogel, R. Balakrishnan, and A. Cockburn. The impact of control-display gain on user performance in pointing tasks. *Human-Computer Interaction*, 23:215–250, 2008.
10. O. Chapuis, J.-B. Labrune, and E. Pietriga. DynaSpot: Speed-dependent area cursor. In *Proc. CHI*, 2009.
11. R. Chua and D. Elliott. Visual regulation of manual aiming. *Human Movement Science*, 12, 1993.
12. M. Collomb and M. Hascoët. Speed and accuracy in throwing models. In *Proceedings of HCI*. British HCI Group, 2004.
13. M. Collomb, M. Hascoët, P. Baudisch, and B. Lee. Improving drag-and-drop on wall-size displays. In *Proc. GI*, 2005.
14. E. R. F. W. Crossman and P. J. Goodeve. Feedback control of hand-movement and Fitts' law. *Quarterly J. of Experimental Psychology*, 35A:251–278, 1983.
15. P. Dragicevic. Building input configurations, 2004. Unpublished webpage. http://inputconf.sourceforge.net/examples.html.
16. M. S. Dulberg, R. S. Amant, and L. S. Zettlemoyer. An imprecise mouse gesture for the fast activation of controls. In *Proc. Interact*, pages 375–382, 1999.
17. D. Elliott, W. F. Helsen, and R. Chua. A century later: Woodworth's (1899) two-component model of goal-directed aiming. *Psychological Bulletin*, 127(3):342–357, 2001.
18. P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *J. of Experimental Psychology*, 47, 1954.
19. C. Forlines, D. Vogel, and R. Balakrishnan. HybridPointing: fluid switching between absolute and relative pointing with a direct input device. In *Proc. UIST*, pages 211–220, 2006.
20. N. Friedlander, K. Schlueter, and M. Mantei. Bullseye! When Fitts' law doesn't fit. In *Proc. CHI*, 1998.
21. T. Grossman and R. Balakrishnan. The bubble cursor: Enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proc. CHI*, 2005.
22. Y. Guiard, R. Blanch, and M. Beaudouin-Lafon. Object pointing: a complement to bitmap pointing in GUIs. In *Proc. GI*, 2004.
23. M. Hascoët. Throwing models for large displays. In *Proceedings of HCI*. British HCI Group, 2003.
24. E. R. Hoffmann. Capture of moving targets: A modification of Fitts' law. *Ergonomics*, 34, 1991.
25. R. J. Jagacinski and J. M. Flach. *Control Theory for Humans: Quantitative Approaches to Modeling Performance*. 2003.
26. R. J. Jagacinski, D. W. Repperger, S. L. Ward, and M. S. Moran. A test of Fitts' law with moving targets. *Human Factors*, 22(2):225–233, 1980.
27. S. W. Keele. Movement control in skilled motor performance. *Psychological Bulletin*, 70, 1968.
28. G. Kurtenbach and W. Buxton. The limits of expert performance using hierarchic marking menus. In *Proc. CHI*, pages 482–487, 1993.
29. I. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7:91–139, 1992.
30. I. S. MacKenzie, A. Sellen, and W. A. S. Buxton. A comparison of input devices in element pointing and dragging tasks. In *Proc. CHI*, pages 161–166, 1991.
31. M. J. McGuffin and R. Balakrishnan. Fitts' law and expanding targets: Experimental studies and designs for user interfaces. *TOCHI*, 12, 2005.
32. D. E. Meyer, R. A. Abrams, S. Kornblum, C. E. Wright, and J. E. K. Smith. Optimality in human motor performance: Ideal control of rapid aimed movements. *Psychological Review*, 95(3):340–370, 1988.
33. D. E. Meyer, J. E. K. Smith, S. Kornblum, R. A. Abrams, and C. E. Wright. Speed-accuracy tradeoffs in aimed movements: Toward a theory of rapid voluntary action. In M. Jeannerod, editor, *Attention and Performance XIII*, pages 173–226. Lawrence Erlbaum, Hillsdale, NJ, 1990. http://www.umich.edu/˜bcalab/Meyer_Bibliography.html.
34. M. Moyle and A. Cockburn. A flick in the right direction: A case study of gestural input. *Behaviour and Information Technology*, 24(4):275–288, 2005.

35. M. Nancel and M. Beaudouin-Lafon. Extending marking menus with integral dimensions: Application to the dartboard menu. Technical Report #1503, LRI, France, 2008.

36. G. Ramos and R. Balakrishnan. Pressure marks. In *Proc. CHI*, pages 1375–1384, 2007.

37. A. Reetz, C. Gutwin, T. Stach, M. Nacenta, and S. Subramanian. Superflick: a natural and efficient technique for long-distance object placement on digital tables. In *Proc. GI*, pages 163–170, 2006.

38. J. B. J. Smeets, M. A. Frens, and E. Brenner. Throwing darts: timing is not the limiting factor. *Experimental Brain Research*, 144:268–274, 2002.

39. R. W. Soukoreff and I. S. MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of Fitts' law research in HCI. *International J. of Human-Computer Studies*, 61:751–789, 2004.

40. N. Walker, D. E. Meyer, and J. B. Smelcer. Spatial and temporal characteristics of rapid cursor-positioning movements with electromechanical mice in human-computer interaction. *Human Factors*, 35(3):431–458, 1993.

41. S. Zhai. Characterizing computer input with Fitts' law parameters–the information and non-information aspects of pointing. *International Journal of Human-Computer Studies*, 61:791–809, 2004.