

# Efficient communication with one or two buttons

David J.C. MacKay\*, Chris J. Ball\* and Mick Donegan†

\**Cavendish Laboratory, Cambridge, CB3 0HE, United Kingdom*

†*ACE Centre, 92 Windmill Road, Oxford OX3 7DR, United Kingdom*

**Abstract.** We discuss how the arithmetic-coding-based communication system, Dasher, could be driven by discrete button presses. We describe several prototypes and predict their information rates.

Dasher is a communication system based on a beautiful idea from information theory, called arithmetic coding (Witten et al., 1987; MacKay, 2003, Chapter 6). Arithmetic coding is an optimal method for text-compression using a language model. By turning arithmetic coding on its head, we obtain an optimal method for text-*generation*.

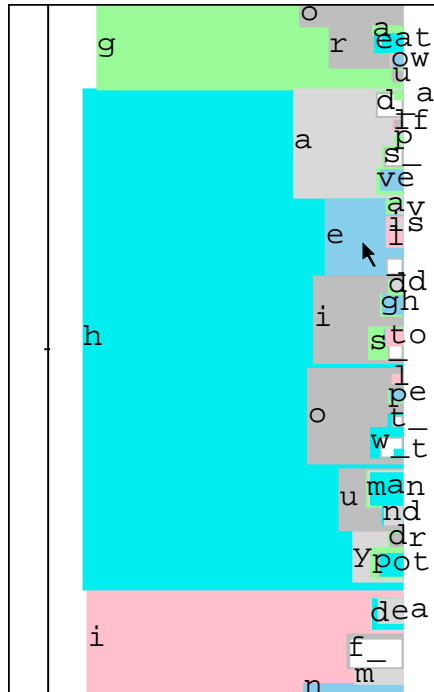
We view a person's gestures as a source of information, and the sentences they wish to communicate as the sink of information. Good interface design maximizes the number of bits per second that are conveyed from the user into text. Poor interfaces waste the user's time either by failing to extract all the bits that the user could easily generate, or by diverting the user's bits into redundant activity.

The Dasher approach to interface design decouples the issues of efficient bit-generation and efficient language-generation. Unlike in most interfaces, a Dasher-user's gestures have no relationship to particular symbols in the language. Instead, they control *navigation* in a continuous space whose contents are laid out using a language model. For demonstrations, or to try Dasher for yourself – it's free – please visit [www.inference.phy.cam.ac.uk/dasher/](http://www.inference.phy.cam.ac.uk/dasher/).

## HOW DASHER WORKS

Imagine writing a piece of text by going into the library that contains *all possible books*, and finding the book that contains exactly that text. In this way, writing can be turned into a navigational task. What is written is determined by where the user goes. In Dasher's idealized library, the 'books' are arranged alphabetically on one enormous shelf. When the user points at a part of the shelf, the view zooms in continuously on that part of the shelf. To write a message that begins 'hello', one first steers towards the section of the shelf marked h, where all the books beginning with h are found. Within this section are sections for books beginning ha, hb, hc, etc.; one enters the he section, then the hel section within it, and so forth.

To make the writing process efficient we use a language model, which predicts the probability of each letter in a given context, to allocate the shelf-space for each letter of the alphabet, as illustrated in figure 1. The shelf is recursively chopped up in such a way that the amount of shelf-space devoted to a string is proportional to its probability. The user's gestures are turned into steering commands, controlling the portion of the display



**FIGURE 1.** A Screenshot of Dasher when the user starts writing `hello`. The shelf of the alphabetical ‘library’ is displayed vertically. The space character, ‘\_’, is included in the alphabet after `z`. Here, the user has zoomed in on the portion of the shelf containing messages beginning with `g`, `h`, and `i`. Following the letter `h`, the language model makes the letters `a`, `e`, `i`, `o`, `u`, and `y` easier to write by giving them more space. Common words such as `had` and `have` are visible. The pointer’s vertical coordinate controls the point that is zoomed in on, and its horizontal coordinate controls the rate of zooming; pointing to the left makes the view zoom out, allowing the correction of recent errors.

zoomed into. If the user can generate information at a rate of, say, 5 bits per second, then our aim is to feed these bits to Dasher in such a way that, each second, the display zooms in by a factor of  $2^5 = 32$  on the region containing the text required by the user. When the language model’s predictions are accurate, many successive characters can be selected by a single gesture. The language model we use, PPMD5 (Cleary and Witten, 1984; Teahan, 1995), generates text at an exchange rate of about two bits per character. Thus the user will be able to write at  $5/2$  characters per second, or 30 words per minute. We could only beat this writing speed by enhancing the rate at which the user generates bits, or improving the predictions of the language model.

Dasher was first developed to be driven by *continuous* two-dimensional gestures, delivered, for example, via a mouse, touch screen, or gazetracker. Our experiments showed that, with Dasher, it is easy to spell correctly and hard to make spelling mistakes. Using an ordinary mouse, typical novice users reach a writing speed of 25 words per minute after 60 minutes of practice, and expert users can write at 35 words per minute (Ward et al, 2002). Results using Dasher with an gazetracker were record-breaking: after 60 minutes’ practice, novice users can drive Dasher using an eyetracker at a speed of about 15 words per minute; expert users can write at 25 words per minute, and make almost no spelling mistakes (Ward and MacKay, 2002).

In this paper, we discuss how Dasher could instead be used with button-pushes.

## BUTTONS

We are concerned with several categories of users, whom we model as follows.

**Two or three buttons, no timing, no noise.** We do not want to give these users time-critical tasks. Conveying information through the precise timing of a button-push is not an option. These users can select among two or three buttons. We assume that their selections are reliable. An example user who might match this category is someone with cerebral palsy.

If we assume that the typical time between button presses is  $t$ , the capacity of the communication channel is  $(\log_2 K)/t$ , where  $K$  is the number of buttons they can choose among. A more sophisticated model might discriminate between the time between two presses of the same button, and the time taken to switch buttons.

**Two or three buttons, no timing, noisy.** If we assume the user does not always press the correct button, then the capacity of the communication channel is significantly reduced. For example, if a user has two buttons and presses the wrong one a fraction  $f = 1/10$  of the time, then the capacity is reduced by nearly a factor of two. The capacity is

$$C = \frac{\log_2 K}{t} (1 - H_2(f)), \quad (1)$$

where  $H_2$  is the binary entropy function.

A user who might match this category is someone communicating via a brain-computer interface. Some researchers attempt to reduce the noise level  $f$  of their brain-computer interfaces by increasing the time  $T$  between decisions. But increasing  $T$  may not be optimal in terms of capacity. Perhaps we can design a more efficient interface that does not require reliable decisions.

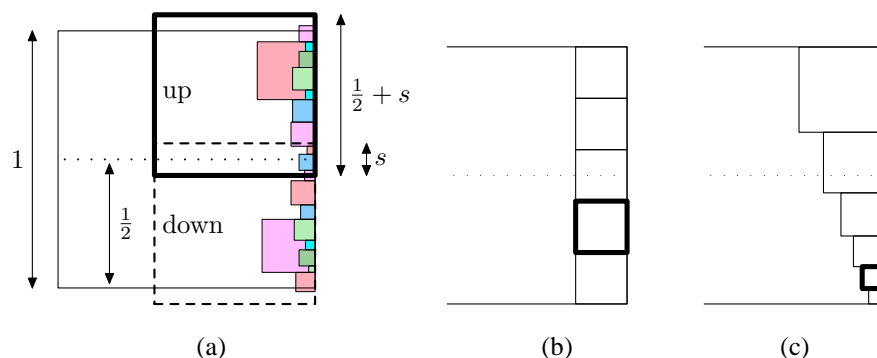
**One button.** A user who presses only one button can convey information by the timing of those events.

**One button, short and long presses.** Some users can control how long they hold a button down. One possible model, analogous to that used in Morse code, allows the user to distinguish between short and long presses.

## TWO-BUTTON DASHER

We can imagine two ways in which two-button Dasher could work: ‘direct’ and ‘menu’ two-button Dasher. Both approaches rely on the assumption that the language model is doing its job, so that the user’s destination is equally likely to be any point on the right-hand side of the display.

In **direct two-button Dasher** (figure 2a), each button is associated with an immediate zooming action. If each button zooms in on a different half of the display, then the user can convey information at a rate of one bit per button press. We can’t do any better than this.



**FIGURE 2.** Alternative ways in which two-button Dasher can work. (a) **‘Direct’ two-button Dasher.** Each button is associated with a zooming action: in-and-up, and in-and-down respectively. If these two regions fill the current view without overlap then the system has the maximum possible information rate, one bit per button press. For robustness, we imagine each target region is bigger, by an amount  $s$ , than the ideal size of  $1/2$ . The information rate is thus  $\log(1/(s + 1/2))$  per button push. In this figure, we show a representative painting of the Dasher world with little boxes associated with the text being written. We emphasize that none of the navigation methods presented in this paper refer to those boxes. This paper discusses navigation methods that will work for any language model. The language model’s sole job is to arrange that the probability distribution of our destination is uniform over the right hand side of the display. We omit the language model’s boxes from now on. (b) **‘Menu’ two-button interface.** One button is used to cycle through a list of options; the other button is used to select the highlighted option. In this illustration, there are five options, each associated with a region of width  $(1/5 + s)$ , where  $s$  is the small robustness parameter. A menu-based interface has the advantage of allowing additional options (such as zoom back, or stop) to be included at the end of the menu. (c) **Menu with non-uniform-sized boxes.** The sizes of these six boxes are in a geometric progression.

For robustness, however, we have to allow the user a little lee-way. We pad each target out with padding of thickness  $s$  ( $s/2$  on either side). To make clear to the user the effect of the two possible actions, we need only to ensure that the display includes a horizontal dividing line at a height of  $1/2$ . Direct two-button Dasher has the disadvantage that it offers the user no alternatives apart from ‘up’ and ‘down’. One fix is to provide a third button, for occasional use only, to enable zooming back out or other control functions (such as changing to another application). In the absence of a third button, these control functions can be included in the language model in the Dasher canvas. Such a *control mode* is already available in Dasher. But we think that a second version of two-button Dasher may be preferable.

In **menu two-button Dasher** (figure 2b,c), one button (‘rotate’) is used to cycle between a set of  $B$  options. The other button is used to select the highlighted option. In keeping with the Dasher philosophy, the options are *not* associated with the text being written on the Dasher canvas. Rather, each option is associated with a particular zooming action. The set of  $B$  options can be displayed simultaneously by overlaying  $B$  light boxes on the Dasher canvas, of which one is highlighted. We arrange that pressing the ‘rotate’ button  $B$  times returns the highlight to the first option in the list. While this circular arrangement is slightly information-inefficient, we think it makes the interface more user-friendly. Other non-zooming options can be included in the cycle.

We could give the alternative zooming-in options equal size on the screen (figure 2b), but it is more information-efficient to have the first option in the list be more probable

(*i.e.*, larger) than the second, and so forth (figure 2c). We will explore this issue in more detail in a moment.

Which is better, direct or menu two-button Dasher? One concern with direct two-button Dasher is that the user might require a little time to get oriented every time the display zooms in. Thus while the most information-efficient two-button solution for an ideal user is direct two-button Dasher, a real user might be able to go faster in menu two-button Dasher. The menu approach allows each zooming event to zoom in by a bigger factor, and thus allows the number of zooms to be reduced. We model a user with three parameters:  $t_r$ , the time taken to press the ‘rotate’ button;  $t_s$ , the time taken to press the ‘select’ button; and  $t_t$ , the time required to take in the new view after a zoom has occurred. The time  $t_t$  might also include a delay imposed by the software, if the zooming event takes a substantial time to render. For simplicity, we now assume  $t_r = t_s = 1$ , and we use the delay  $t_t$  and the number of boxes  $B$  as the two parameters to explore the choice between these two versions of two-button Dasher. We consider two versions of menu-Dasher, ‘uniform’, and ‘non-uniform’. In the uniform menu interface, all boxes have size  $\frac{1}{B} + s$ , where  $s$  is the small robustness parameter, here set to 0.05. In the non-uniform menu interface, the sizes are  $\{p_1 + s, p_2 + s, \dots, p_B + s\}$ , where  $\sum_b p_b = 1$ . Our objective function is the user’s bit rate,

$$R = \frac{I(\mathbf{p})}{T(\mathbf{p})}, \quad (2)$$

where the average number of bits per zoom is

$$I(\mathbf{p}) = \sum_b p_b \log_2 \frac{1}{p_b + s} \quad (3)$$

and, assuming the user always takes the shortest path to his target, the average time between zooms is

$$T(\mathbf{p}) = t_t + t_s + t_r \left[ \sum_b p_b (b - 1) \right]. \quad (4)$$

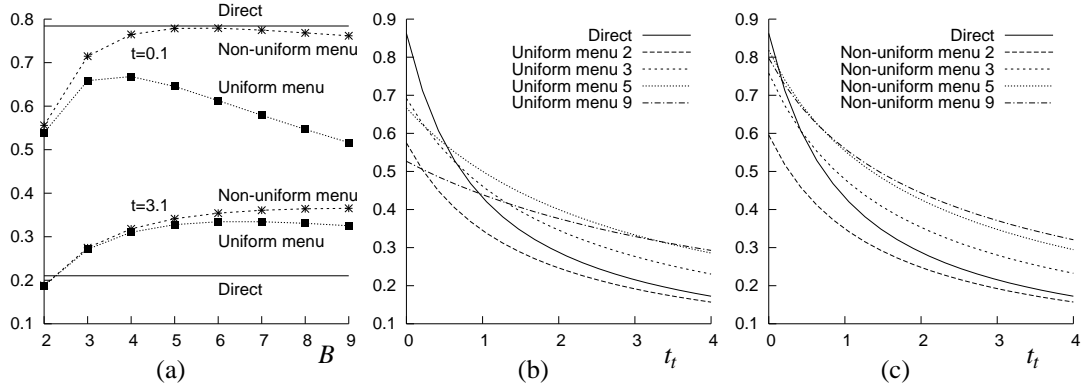
In the special case of uniform menus, the objective function takes the value

$$R_{\text{uniform}}(B, t_t) = \frac{\log_2(1/(1/B + s))}{t_t + t_s + \frac{1}{2}(B - 1)t_r}. \quad (5)$$

It is possible to optimize the objective function  $R$  with respect to  $\mathbf{p}$  and the number of boxes  $B$ , for a given user parameter  $t_t$ . However, we are interested in finding a robust solution that will work well across a range of values of  $t_t$ . We therefore examine a simple exponential family of solutions,

$$p_b = \frac{1}{Z(C)} 2^{-C t_r (b-1)}, \quad (6)$$

where  $Z(C) = (1 - a^B)/(1 - a)$ , with  $a = 2^{-C t_r}$ . Equation (6) gives the optimal solution if the robustness parameter  $s$  is 0; at the optimum, the parameter  $C$  is identical to the



**FIGURE 3.** Information rate for menu-style interfaces as a function of the number of menu boxes and the delay parameter  $t_t$ . Vertical axis: information rate in bits per unit time. Button presses take unit time. (a) Information rate versus number of boxes, for two values of the delay parameter  $t_t$ . The upper curves ( $t_t = 0.1$ ) illustrate the relative merits of uniform and non-uniform menus if there is very little delay associated with each zooming action: non-uniform menus can get very close to the information-rate of the direct two-button interface; uniform menus with 3, 4, or 5 boxes can get within 15%. The lower curves assume that there is a substantial pause ( $t_t = 3.1$ ) after each zoom: the user waits 3.1 times the duration of a button press before pressing another button. In this case, both uniform and non-uniform menus can be more than twice as good as the direct two-button interface. (b) Information-rates of menu-style interfaces with uniformly-sized menu boxes as a function of the delay  $t_t$  associated with each zoom. (c) Information-rates of menu-style interfaces with non-uniformly-sized menu boxes. For all calculations the robustness parameter  $s$  was set to 0.05.

bit rate  $I(\mathbf{p})/T(\mathbf{p})$  (MacKay, 2003, p. 125). We approximate the optimum non-uniform solution for a given  $t_t$  and  $B$  by iterating a couple of times the assignments of  $\mathbf{p}$  (6) and  $C = R(2)$ , starting from  $C = R_{\text{uniform}}(B, t_t)$ .

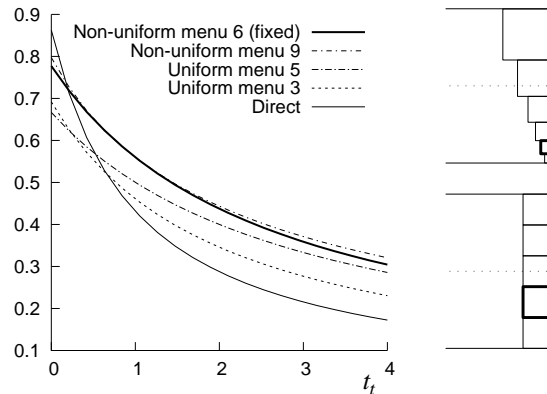
The resulting values of the objective function  $R(B, t_t)$  are shown in figure 3.

Figure 3a shows that, if there is very little delay associated with each zoom ( $t_t = 0.1$ ), then the direct two-button solution has the best information rate ( $R = 0.78$ ): but non-uniform menus with  $B = 4, 5, 6$ , or 7 boxes can get very close (these solutions have boxes whose sizes are roughly  $1/2, 1/4, 1/8, \dots$ ); uniform menus with 3, 4, or 5 boxes can also get within 15% of the best solution.

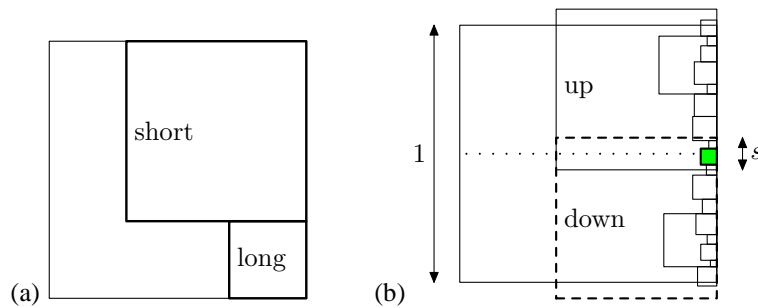
What if we assume there is a substantial pause ( $t_t = 3.1$ ) after each zoom? (So the user waits for three times the duration of a button press before pressing another button.) In this case, the lower curves in figure 3a show that both uniform and non-uniform menus can be *more than twice as good* as the direct two-button solution.

Figure 3b shows the dependence of the uniform menu system's information rate on  $t_t$  for different numbers of boxes. Figure 3c shows equivalent curves for the non-uniform menu system.

Now, imagine that we wish to make a single piece of software that does a good job for users with a range of values of  $t_t/t_r$ . We could make an interface whose parameters  $B$  and  $\mathbf{p}$  adapt optimally to the measured properties of the user. But, for simplicity, can we find a fixed setting of  $B$  and  $\mathbf{p}$  that gives good performance across the board? After inspecting figure 3, we selected one uniform and one fixed non-uniform menu solution whose performance looks good for  $t_t/t_r$  ranging from 0 to 4 (figure 4). The uniform solution has  $B = 5$ . The non-uniform solution has parameters  $B = 6$ ,  $\mathbf{p} \simeq \{.33, .24, .17, .12, .09, .06\}$ .



**FIGURE 4.** Information-rate (in bits per unit time) of a fixed non-uniform menu-system with six boxes as a function of the parameter  $t_t$ . Also highlighted is the uniform five-box menu interface. For comparison, good menu systems from figure 3 are shown with lighter lines.



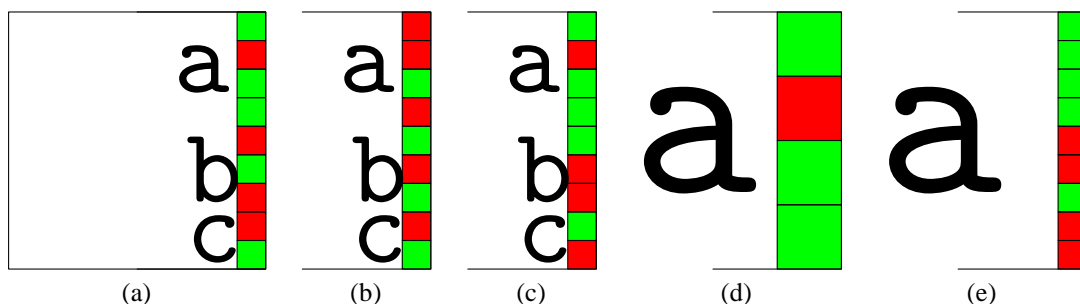
**FIGURE 5.** (a) Two-button Dasher modified for a single-button user who can make short and long presses. (b) Dasher for the blind. At each step, the system identifies a string whose interval lies entirely in the interval  $(0.5 - s/2, 0.5 + s/2)$ , as illustrated by the small highlighted box; any such string will do, but the best choice of string might be the most probable one that is pronounceable. This string is read to the user. The user selects between the two targets according to whether the desired string is alphabetically above or below that string.

We propose to include in the cycle of options a ‘zoom-out’ option that zooms straight out by a factor of  $1/(p_{\max} + s)$ , where  $p_{\max}$  is the biggest probability.

Two-button Dasher could also be used by a single-button user who can make short and long presses. The menu system could be used with short presses mapped to ‘rotate’ and long presses to ‘select’. The direct two-button method would be optimized by giving the short press a larger target area than the long (figure 5a).

## Dasher for the blind

Dasher was conceived as an intensely visual communication interface, but direct two-button Dasher could nevertheless be used in an auditory mode. At each step, the interface reads to the user a piece of text whose Dasher-interval is wholly contained in the interval of width  $s$  surrounding the centre of the display – the interval marked by the small arrow in figure 5b. The user then decides whether her intended piece of text is alphabetically



**FIGURE 6.** Proposal for noisy two-button Dasher, for use in brain–computer interfaces, for example. If the user wishes to enter the top box, he presses the light button at step (a), the dark at step (b), and the light at step (c). This sequence is not unique to the top box; the fourth box has the same sequence. And because of the possibility of noise, the system must assign some probability to other targets too. At some point, the system may judge that the posterior probability is sufficiently concentrated that the view can zoom in on a region of the display, as shown in (d). The process continues with a new random colouring (e), which may be chosen to distinguish between the most probable hypotheses (here, the regions associated with the original first and fourth squares).

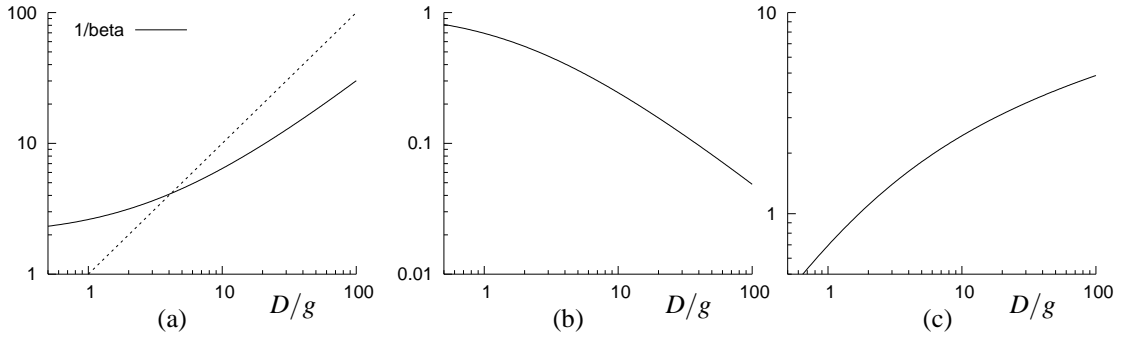
before or after the piece of text that is read to her, and presses ‘up’ or ‘down’ accordingly. Once the piece of text to be read is longer than a few words, it would probably suffice to read the last few words only. Such a system might be useful for a blind person who cannot speak or type, but who can hear, and press buttons.

### Two or more noisy buttons

If a user’s button presses are noisy – with the wrong button being pressed occasionally – then a radical redesign is required. We propose the following solution. The user is presented with a Dasher canvas. Superposed on the Dasher canvas are  $B$  boxes, just like the  $B$  boxes in uniform-menu two-button Dasher, though the number of boxes should be quite large, perhaps 16. (Each box should be viewed as having a cushion of width  $s/2$  surrounding it, though for display purposes, the cushion may be omitted.) These boxes are pseudorandomly coloured with  $K$  colours (figure 6a), where  $K$  is the number of buttons available to the user. To avoid obscuring the features of the Dasher landscape, the coloured boxes could be placed adjacent to the Dasher canvas instead of on it. The user is asked to choose her destination box by pressing the appropriate button. We assume she does this noisily. A Bayesian inference is used to update the probability distribution of the user’s intended destination. If she presses the light button rather than the dark button, say, and the noise level is  $f$ , then all boxes that are currently light get their posterior odds increased by a factor  $(1 - f)/f$ . If side-information is available that reveals how reliable each individual press is (in the form of a likelihood function) then this information can be used in the inference.

After the Bayesian inference, the system should decide whether it is safe to zoom in at all. Multiple presses will often be required before a move takes place. Zooming-in is safe only if the probability mass associated with the region zoomed in on is above some threshold such as 0.99. Let’s assume no zooming is currently possible. The system





**FIGURE 7.** Capacity of a single button. (a) The optimal value of  $1/\beta$  as a function of the delay-to-precision ratio  $D/g$ . As a guide to the eye, the line  $y = x$  is also shown.  $1/\beta$  is the mean time the user waits, after the delay  $D$ , before clicking. (b,c) The capacity of the channel, in bits per duration  $g$ , and bits per duration  $D$ , respectively.

presents a new pseudorandom colouring of the landscape to the user (figure 6b), and gets more noisy data from her. The system can choose each pseudorandom colouring in a way that depends on the current probability distribution of the destination – so as to maximize the expected amount of information from the user, or to minimize the expected time until a significant zoom can be made.

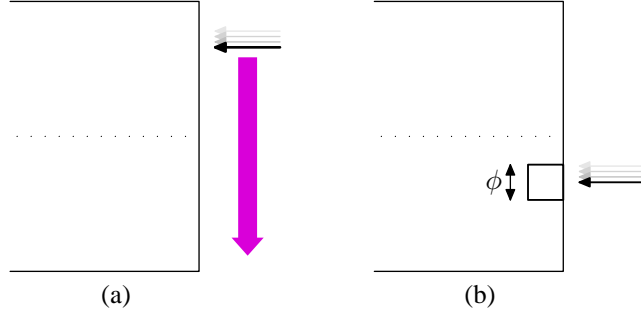
Eventually, a zoom onto a particular region will become safe. This zoom will not necessarily be a zoom right into one of the  $B$  boxes; it might be a lesser zoom that zooms in on a *collection* of boxes that includes the most probable boxes and a few others (figure 6d). When this zoom is made, the zoomed-in region is immediately coloured with boxes in the same way as before (figure 6e).

To allow the possibility of zooming back when mistakes are made, an additional target can be included somewhere on the display, whose meaning is ‘none of the current boxes’. This target can be randomly coloured, and its probability updated, just like the other targets. If the probability of this target gets big, then the display can zoom back and revive hypotheses that had been judged unlikely. This system will make progress in a way similar to the sequential decoding algorithm for tree codes (Johannesson and Zigangirov, 1999).

There is another possibility: the inferences about the user’s destination could be used to update the probabilities within Dasher’s own language model. However, we expect that it is more user-friendly to keep the Dasher landscape as a non-deforming world in which the user moves her point of view.

## ONE-BUTTON DASHER

For the simplest model of a single-button user, we assume that the user controls only the times of presses, not the times of releases. We model the user with two parameters: a timing accuracy  $g$ , and a recovery time  $D$ . The user clicks within a time  $\pm g/2$  around each intended click time, and then requires a delay of duration  $D$  before she can click again. Before describing our two proposed versions of one-button Dasher, we compute the capacity of the channel as a function of the parameters  $D$  and  $g$ .



**FIGURE 8.** How static one-button Dasher works. (a) A pointer appears from one side of the display. As long as the user doesn't click, the pointer sweeps to and fro over the range of possible destinations. (b) The user clicks when the pointer is alongside his required destination. The display zooms in on a fraction  $\phi$  surrounding the pointer, and the process repeats.

We estimate the capacity by chopping time into boxes of duration  $g$ . We write the probability that the user waits for a time  $D + ng$  from one click to the next as

$$p_n = (1 - \beta)^n \beta, \quad n \in \{0, 1, \dots\}. \quad (7)$$

[MacKay (2003, p. 125) explains why the optimal probability has this form.] The mean time to the next press is  $D + g/\beta$ , and the average information content generated by the selection of a waiting time  $n$  is the entropy of  $\mathbf{p}$ ,

$$H_2(\beta)/\beta \text{ bits}. \quad (8)$$

The rate of information generation is thus

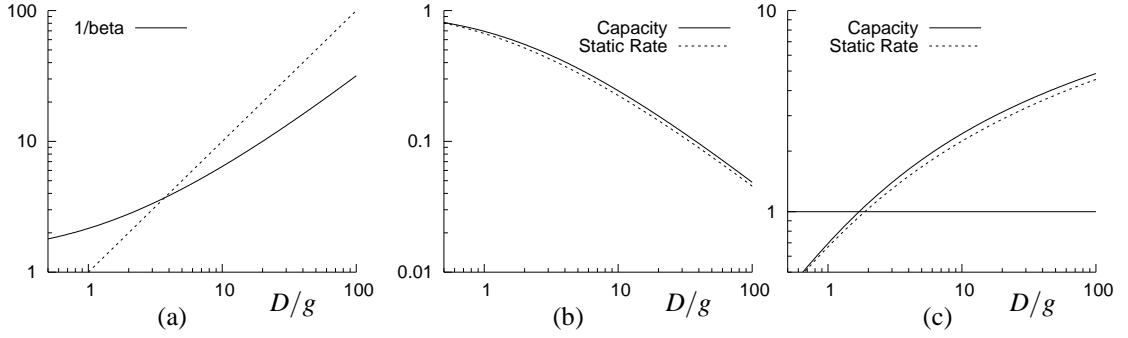
$$R(\beta) = \frac{H_2(\beta)}{\beta D + g}. \quad (9)$$

Figure 7 shows the optimal value of  $1/\beta$  and the capacity as a function of the delay-to-precision ratio  $D/g$ .

We propose two versions of one-button Dasher, 'static' and 'dynamic'.

**Static one-button Dasher** behaves rather like a continuous version of two-button menu-Dasher, or the 'EZ keys' software ([www.words-plus.com](http://www.words-plus.com)). Between clicks, the Dasher display is stationary, and a pointer sweeps up and down the screen. When the user clicks the button, the display zooms in on the region surrounding the pointer (figure 8).

Static one-button Dasher has two parameters: the time  $E$  taken for the pointer to pass across the Dasher canvas once, and the fraction of the display  $\phi$  into which we zoom. We assume that the user's clicks are uniformly distributed in an interval  $\pm g/2$  around the intended click time. We fix  $E$  as a function of  $\phi$  and the clicking accuracy  $g$ :  $E = g/\phi$ . If we zoom in on a fraction  $\phi$ , we generate information at a rate of  $\log_2 1/\phi$  bits per click. We define  $\beta = 2\phi$ . The average time between clicks is  $D + E/2 = D + g/\beta$ , if the pointer always starts from one edge of the display, and there is a recovery-time  $D$  between clicks.



**FIGURE 9.** Optimized performance of static one-button Dasher. (a) The optimal value of  $1/\beta$  as a function of the delay-to-precision ratio  $D/g$ . As a guide to the eye, the line  $y = x$  is also shown.  $1/\beta$  is the mean time (in units of  $g$ ) that the user waits, after the delay  $D$ , before clicking. This chart should be used to set the parameters of static Dasher as a function of  $D$  and  $g$ :  $E = (2/\beta)g$ , and  $\phi = \beta/2$ . (b,c) The rate of communication, in bits per duration  $g$  and bits per duration  $D$ , respectively. The channel capacity is also shown.

The information rate achieved by static one-button Dasher is thus

$$R(\beta) = \frac{\log_2(2/\beta)}{D + g/\beta}, \quad (10)$$

which is maximized with respect to  $\beta$  when

$$\frac{D}{g} = \frac{1}{\beta} \left[ \ln \left( \frac{2}{\beta} \right) - 1 \right]. \quad (11)$$

According to the resulting information rate, plotted in figure 9, static one-button Dasher appears to be able to get within 10% of the channel capacity.

**Dynamic one-button Dasher** also features a moving pointer that sweeps to and fro across the display. Unlike static Dasher, dynamic Dasher moves all the time, with the direction of motion being controlled by the current pointer position. The user's clicks reverse the direction of motion of the pointer. Dynamic one-button Dasher does not exploit high temporal accuracy, so it is expected to be useful only for users whose ratio  $D/g$  is small.

Dynamic one-button Dasher has two parameters, the zooming rate  $z$  of the interface (in nats per second), and the sweeping rate  $r$  of the pointer. [The zooming rate in bits per second,  $z_2$ , is  $z_2 = z/\ln 2$ .]  $1/r$  is the time taken for the pointer to sweep across the unit width of Dasher display. We now calculate what values  $z$  and  $r$  can take for a user whose minimum time between clicks is  $D$ .

Imagine that the target is at vertical coordinate  $y(t)$  at time  $t$ , and the pointer is associated with vertical coordinate  $p(t)$ . Then Dasher's dynamics, zooming at rate  $z$ , will move  $y$  away from the pointer in accordance with

$$\dot{y} = z(y - p). \quad (12)$$

The pointer position obeys  $\dot{p} = \pm r$ , with sign determined by the user's clicks. The user tries to keep the gap  $|y - p|$  small. How big can  $z$  be without  $|y - p|$  diverging? The

critical value of  $|y - p|$  is  $r/z$ ; any bigger, and the gap grows exponentially. If the user clicks optimally, the biggest that the gap can become in time  $t$  is  $rt(1 + zt/2)$ , to leading order. Setting  $t = D$ , we find the maximum possible zooming rate is  $z_{\text{crit}} = \sqrt{3} - 1/D \simeq 0.73/D$ , independent of  $r$ , and thus the communication rate is bounded by

$$R_{\text{max}} = \frac{0.73}{\ln 2} \simeq 1.05 \text{ bits per duration } D. \quad (13)$$

Having fixed the zooming rate  $z$ , the rotation rate  $r$  should be set such that the critical value of  $|y - p| = r/z$  is comfortably large – bigger than the display width, 1, for example. We recommend setting  $r \simeq z$ .

Comparing the maximum bit rate of dynamic one-button Dasher with the curves for static Dasher (figure 9), we predict that faster communication will be achieved with static Dasher by any user whose ratio of recovery time  $D$  to clicking precision  $g$  exceeds 2. Since the user is free to do nothing for a while, static Dasher will probably feel a less demanding interface than dynamic Dasher, which requires perpetual attention and control. To make dynamic Dasher feel safer, we map the extreme pointer positions to Dasher’s origin, so that when the user is inactive, Dasher shortly stops moving.

If the user is able to distinguish between short and long presses, then the capacity increases by up to one bit per button press. Can either of our interfaces exploit this capacity? We have augmented dynamic Dasher such that long presses increase the rotation rate  $r$ , for the duration of the press. Static Dasher could have *two* pointers, sweeping over separate regions in the display; a short or long press would zoom in on the upper or lower pointer, respectively.

Dasher is free software, distributed under the GNU General Public License, and available from [www.inference.phy.cam.ac.uk/dasher/](http://www.inference.phy.cam.ac.uk/dasher/).

## ACKNOWLEDGMENTS

We thank Caroline Gray, David Colven, Iain Murray, and Matthew Garrett for helpful discussions, and Kimber Gross for comments on the manuscript.

We gratefully acknowledge the support of the Gatsby Charitable Foundation.

## REFERENCES

- Witten, I. H., Neal, R. M., and Cleary, J. G., Arithmetic coding for data compression, *Communications of the ACM*, **30**, 520–540 (1987).
- MacKay, D. J. C., *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press (2003). Available from [www.inference.phy.cam.ac.uk/mackay/itila/](http://www.inference.phy.cam.ac.uk/mackay/itila/).
- Cleary, J. G., and Witten, I. H., Data compression using adaptive coding and partial string matching, *IEEE Trans. on Communications*, **32**, 396–402 (1984).
- Teahan, W. J., Probability estimation for PPM. In *Proc. of the N.Z. Comp. Sci. Research Students’ Conf.* (1995). Available from [citeseer.nj.nec.com/teahan95probability.html](http://citeseer.nj.nec.com/teahan95probability.html).
- Ward, D. J., Blackwell, A. F., and MacKay, D. J. C., Dasher – A data entry interface using continuous gestures and language models, *Human-Computer Interaction*, **17** (2002).
- Ward, D. J., and MacKay, D. J. C., Fast hands-free writing by gaze direction, *Nature*, **418**, 838 (2002).

Johannesson, R., and Zigangirov, K. S., *Fundamentals of Convolutional Coding*, IEEE Press, Piscataway, N.J. (1999).

First published July 18, 2004. Version 3.2 – June 22, 2005  
Appeared in *Bayesian Inference and Maximum Entropy Methods  
in Science and Engineering*, proceedings of *Maximum Entropy and Bayesian Methods*,  
2004. (Minor corrections made Wed 22/6/05; thanks to Satoshi Koyama.)