Using Bayes’ Theorem for Command Input: Principle, Models, and Applications

Suwen Zhu¹, Yoonsang Kim¹, Jingjie Zheng², Jennifer Yi Luo¹; Ryan Qin¹; Liuping Wang³, Xiangmin Fan¹, Feng Tian¹, Xiaojun Bi¹

¹Department of Computer Science, Stony Brook University, Stony Brook, NY, United States
²Google, Kitchener, Ontario, Canada
³Institute of Software, Chinese Academy of Sciences, Beijing, China

{suzh, yoonsakim, xiaojun}@cs.stonybrook.edu, jingjiezeng@gmail.com, {jluo9191, ryanqin15}@gmail.com, wangliuping17@mails.ucas.ac.cn, {xiangmin, tianfeng}@iscas.ac.cn

ABSTRACT
Entering commands on touchscreens can be noisy, but existing interfaces commonly adopt deterministic principles for deciding targets and often result in errors. Building on prior research of using Bayes’ theorem to handle uncertainty in input, this paper formalized Bayes’ theorem as a generic guiding principle for deciding targets in command input (referred to as “BayesianCommand”), developed three models for estimating prior and likelihood probabilities, and carried out experiments to demonstrate the effectiveness of this formalization. More specifically, we applied BayesianCommand to improve the input accuracy of (1) point-and-click and (2) word-gesture command input. Our evaluation showed that applying BayesianCommand reduced errors compared to using deterministic principles (by over 26.9% for point-and-click and by 39.9% for word-gesture command input) or applying the principle partially (by over 28.0% and 24.5%).

Author Keywords
Bayes’ theorem; command input; point-and-click; word-gesture shortcuts; touchscreen.

CCS Concepts
•Human-centered computing → Human computer interaction (HCI); Interaction techniques; User studies;

INTRODUCTION
Command input is essential to human-computer interaction. There is no exception in the era of mobile and wearable computing where people regularly issue commands on touchscreens with finger touch or gesture [1, 17, 38, 42]. These input modalities are natural to use, but they inevitably introduce uncertainty. For example, touch input is notoriously known to be noisy and imprecise due to occlusion and the uncertainty of converting a “fat” contact region into a single touchpoint [26, 32, 33, 63, 64, 65]; a recognizer in a gestural command input system may yield erroneous output if the input gestures deviate from the predefined templates [43, 71].

Despite the wide existence of uncertainty, the existing command input methods are ill-positioned for handling it because they often adopt a deterministic principle to decide which command will be issued. For example, to trigger a command with finger touch, the user needs to land the touchpoint precisely within the target boundaries; to input a command with a gesture, the decoded command name should match the exact command label.

Inspired by prior work of adopting Bayes’ theorem to decide touch selection target [10], this work incorporates prior probability into the target deciding process, and generalizes the Bayes’ theorem as a principle (referred to as BayesianCommand) for both point-and-click and gestural command input. Our experiments (explained later) showed incorporating the prior increased command input accuracy by more than 20% (relatively) over using likelihood only [10]. Our work focuses on command input, which also contributes to a large body of research of using a probabilistic framework to handle uncertainty in interaction (e.g., text entry [27], navigation [45], file

\[ e^* \ = \ \arg \max_{c = 1, \ldots, n} P(c|s) \]

\[ e^* \ = \ \arg \max_{c = 1, \ldots, n} \frac{P(s|c)P(c)}{P(s)} \]

Figure 1. Overview of BayesianCommand. Given an input signal s and a set of n commands \( C = \{ c_1, \ldots, c_n \} \), the goal of a command input task is to find \( e^* \) that maximizes \( P(c|s) \). BayesianCommand views the input signal s as a random variable carrying likelihood information, and uses Bayes’ theorem to combine it with the prior probability \( P(c_i) \) to infer \( P(c|s) \). The target command information is then used to update the prior probability model.
BayesianCommand sets out from a probabilistic perspective to interpret the ambiguity in command input: the input signals are viewed as a random variable that carries likelihood information of the target command. The posterior belief is formed accordingly via Bayes’ theorem. The candidate with the highest posterior probability should be decided as the target. This information is in turn used to update the prior probability model for future command input. Figure 1 provides an overview of BayesianCommand. Because BayesianCommand is algorithmic and requires no visual change of the layout, it is advantageous to frequency-based layout rearrangement and less likely to slow users down [20].

Overall, we made the following three contributions: (1) We formally described how to use Bayes’ theorem for command input, and established that it should be the principle for deciding the target command; (2) We proposed three models for applying BayesianCommand: a prior probability model, a dual-Gaussian likelihood model for point-and-click input, and a two-step likelihood model for recall-based gestural command input; (3) We conducted experiments to demonstrate the effectiveness of BayesianCommand over using the existing deterministic principles or applying them partially.

RELATED WORK
This work is related to handling uncertainty in user input, touch pointing and gestural command input technologies.

Handling Uncertainty in User Input
Probabilistic frameworks have been proposed to deal with the uncertainty in input processes, such as considering the input as a continuous control process in which the system continuously infers a distribution over potential user goals [13, 67, 70], or carrying the uncertainty of input forward all the way through the interaction [61, 62]. Other examples include Dasher [66], which used probabilistic models to adapt screen layouts, and Semantic pointing [12], which adapted the control-to-display ratio according to cursor distance to nearby targets. Bayes’ theorem has also been adopted to reduce uncertainty in interaction, such as the statistical decoding algorithm of soft keyboards [27], and the Bayesian Information Gain (BIG) framework [45, 46].

Distinct from the previous work, we focus on command input, adding to the vast body of research of using probabilistic frameworks to handle input uncertainty. BayesianCommand uses Bayes’ theorem to incorporate the previous command input history to improve input accuracy. Although previous research has explored adapting the menu visuals (e.g., Morphing menu [16]) to accommodate the command frequency, BayesianCommand is algorithmic and requires no visual change, making the UI visual consistent to users throughout the interaction.

Understanding and Improving Finger Touch
There has been extensive research on understanding and improving touch pointing accuracy. On a capacitive touchscreen, a touchpoint is converted from the finger’s contact region with noise and uncertainty in the converting process. Factors such as hand posture [14, 26], finger angle [32, 33], and body movement [25, 55] may affect the size and shape of the contact region, unintentionally altering the touch position. The lack of feedback on where the finger lands due to occlusion (the “fat finger” problem) further exacerbates the issue [32, 33, 63, 64, 65]. Previous research has explored various approaches to improve touch accuracy. Examples include compensating for the offset caused by different finger input angles [32, 33] or location on screens [31], displaying the touch location in a non-occluded area [64], and using the back of the device for selection [68, 69]. Others also explored using various finger gestures to assist target selection, including crossing [2, 15, 51, 52, 58], sliding [14, 54, 72, 73], rubbing [57, 60], circling [34], and multi-touch gestures [8].

Our first application of using BayesianCommand to improve point-and-click command input is in particular related to Bayesian Touch Criterion (BTC) [10]. The main difference is that BayesianCommand involves both prior and likelihood probability calculation, while BTC ignored the prior probability and only used the likelihood. We compared BayesianCommand with BTC in our user study. We showed that incorporating prior is an essential step toward truly adopting Bayes’ theorem, and it substantially improves the touch accuracy over BTC. Apart from point-and-click input, our second application of gestural command input is different from BTC [10].

Word-Gesture Command Input
Gestural input has been widely explored as a command input method on touchscreen devices, thanks to the human’s ability in memorizing pictorial information [56]. It has been adopted in marking menu [39, 40, 41] and its variants [5, 6, 22, 23, 75, 76], gesture-based interfaces [7, 24, 42, 47, 48, 49, 50], and multi-touch gesture frameworks [35, 36].

To assist users in memorizing the mappings between commands and gestures, previous researchers have explored using word-gestures [37, 74] for command input – entering a command by gliding finger over letters in the command name on a virtual keyboard. Word-gesture was initially invented for text entry on touchscreen devices [37, 74], which was later extended as a method for command input. For example, Command Strokes [38] and CommandBoard [1] support triggering a command by drawing its word-gesture on a soft keyboard, and HotStrokes [17] supports word-gesture command input on a laptop trackpad.

The existing gestural command input systems (e.g., [1, 17, 38] often adopt a deterministic principle to decide the target command: the decoder matches the input gesture with the predefined gesture template of each command candidate; the candidate with highest matching score is the target command. It has little room for handling uncertainty and would result in errors if the input gesture deviates greatly from the template or some commands share the similar predefined templates. In this paper we proposed BayesianCommand to replace the typical deterministic principle to handle such uncertainty.
BayesianCommand: A BAYESIAN PERSPECTIVE ON COMMAND INPUT

From a Bayesian perspective, a command input task can be described as follows: assuming \( C = \{c_1, c_2, \ldots, c_n\} \) is a set of \( n \) available commands, given the input signal \( s \), the goal of a command input task is to find \( c^* \) in \( C \) that maximizes \( P(c|s) \). According to Bayes’ theorem, it can be calculated as:

\[
c^* = \arg \max_{c \in C} P(c|s) = \arg \max_{c \in C} \frac{P(s|c)P(c)}{P(s)}, \quad (1)
\]

Assuming \( P(s) \) is a constant across \( c \) (because \( s \) is a fixed value for a given input), we can further simplify Equation (1) to:

\[
c^* = \arg \max_{c \in C} [P(s|c)P(c)], \quad (2)
\]

where \( P(c) \) is the prior probability of \( c \) being the intended command without the observation of \( s \), and \( P(s|c) \) describes how probable \( s \) is if the intended target is \( c \) (the likelihood). We refer to this principle as BayesianCommand in this paper.

Obtaining the prior and likelihood is the key to apply BayesianCommand. We developed one prior probability model, and two likelihood models: a dual-Gaussian likelihood model for point-and-click and a two-step likelihood model for command shortcut. The dual-Gaussian likelihood model was inspired by the dual Gaussian distribution hypothesis [9, 10]. Note that these models represent only one approach of estimation. We use them to establish that BayesianCommand is the principled framework for command input.

PRIOR PROBABILITY MODEL

We first developed a model to predict \( P(c) \) – the prior probability of the candidate \( c \) being the intended target – from the command input history.

In the prior probability model, we assume that the distribution of the intended command among candidates is not entirely random, and the command input history is observable. We formed this assumption based on the findings that the patterns of menu selection [16, 44], command triggering [3, 19], and smartphone app launching [53] are not random and often follow certain distributions (e.g., Zipfian distributions). These are all scenarios involving frequent command input.

Before deriving the model, we define two criteria that the model should satisfy:

(1) Without observing any selection history, each candidate is equally probable as the target.

(2) With a large number of observations, \( P(c) \) approximates the frequency that the candidate \( c \) was selected as the target in the past.

We propose the frequency model as follows. Assuming we observe that the candidate \( c_i \) has been selected \( t_i \) times in the past as the target, \( P(c_i) \) is calculated as:

\[
P(c_i) = \frac{k + t_i}{k \cdot n + \sum_{i=1}^{n} t_i}, \quad (3)
\]

where \( n \) is the number of available commands (e.g., the number of items in a menu), and \( k \) is the update rate, a positive constant which determines how fast \( P(c_i) \) will be learned from the selection history.

The proposed model (Equation (3)) satisfies aforementioned criteria (1) and (2). If no selection history is observed, i.e., \( t_i = 0, i \in [1,n] \), Equation (3) shows \( P(c_i) = \frac{1}{n} \). It indicates that each candidate is equally probable as the target. On the other hand, if we have a large number of observations on selection history (i.e., \( t_i \gg k \cdot n \) and \( t_i \gg k, i \in [1,n] \)), Equation (3) shows \( P(c_i) \approx \frac{t_i}{\sum t_i} \), which is the frequency of \( c_i \) being the target in the past.

The update rate \( k \) in the model controls the balance between two extreme views on calculating \( P(c_i) \):

(A) \( P(c_i) \) is identical to the frequency of \( c_i \) being the target in the past.

(B) all the candidates are equally probable as the target.

If \( k = 0 \), \( P(c_i) = \frac{1}{\sum t_i} \), which is the view (A). If \( k \to +\infty \), \( P(c_i) \approx \frac{1}{n} \), which is the view (B). A positive \( k \) controls the weights between these two views. Later we explain how we used a simulation-based approach to determine an optimal \( k \) in our applications.

LIKELIHOOD MODELS

We have developed two likelihood models: a dual-Gaussian likelihood model for point-and-click and a two-step likelihood model for recall-based gestural command input. Because each of the likelihood models is tightly connected to the specific command input method, we describe how to obtain them when describing applications.

After obtaining prior probability and likelihood, we can apply BayesianCommand to decide the target command. Algorithm 1 shows how the BayesianCommand principle works.

Algorithm 1 BayesianCommand

1: Input: \( s \) – the input signal \( s \), \( C = \{c_1, \ldots, c_n\} \)
2: Output: the target command \( c^* \)
3: for \( i = 1, 2, \ldots, n \) do
4: \( \) obtain prior probability \( P(c_i) \) from Equation (3)
5: \( \) calculate \( P(s|c_i) \) from the likelihood model
6: \( \) select \( c^* = \arg \max \{P(s|c_i)\} \) as the target command
8: update prior probability \( P(c_i) \) for each \( c_i \) based on Equation (3), given that \( c^* \) is the selected command.

APPLICATION 1: APPLYING BayesianCommand TO POINT-AND-C Click COMMAND SELECTION

We first applied BayesianCommand to improve the command input accuracy on a touch-based point-and-click interface: triggering a command by touch pointing the corresponding icon, button, or menu item. We expect that BayesianCommand will improve the command input accuracy over the typical boundary criterion, which decides the target by examining whether the touchpoint falls within the target boundary.
To apply BayesianCommand to point-and-click command selection, we used the prior probability model (Equation (3)) to estimate \( P(c) \). Next, we explain how to obtain \( P(s|c) \) for point-and-click interfaces.

Dual-Gaussian Likelihood Model for Point-and-Click

We adopted a dual-Gaussian likelihood model to calculate \( P(s|c) \), assuming that the touchpoints approximately follow a Gaussian distribution [4, 31, 32]. Assuming for a 2-dimensional target we observe a touchpoint \( s \) as \((s_x, s_y)\), \( P(s|c) \) can be calculated as:

\[
P(s|c) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left\{ -\frac{r^2}{2(1-\rho^2)} \right\},
\]

where

\[
z \equiv \frac{(s_x - \mu_x)^2}{\sigma_x^2} - \frac{2\rho(s_x - \mu_x)(s_y - \mu_y)}{\sigma_x\sigma_y} + \frac{(s_y - \mu_y)^2}{\sigma_y^2}.
\]

(\( \mu_x, \mu_y \)) is the target center, \( \sigma_x \) and \( \sigma_y \) are the standard deviations of users’ touchpoints, and \( \rho \) is the correlation coefficient between \( x \) and \( y \). We followed the next two steps to estimate the parameters of Equations (4) and (5).

First, we assumed that the center of touchpoint distribution \((\mu_x, \mu_y)\) co-locates with the center of the target. Previous research showed that \((\mu_x, \mu_y)\) has only a small offset from the target center, and the magnitude and direction of the offset are affected by various factors including the target position on the screen, users’ postures, and finger angle, etc. [4, 31, 32, 67, 77]. Without further knowledge on these factors, we assume \((\mu_x, \mu_y)\) is located at the target center. Similarly, previous research also showed the correlation coefficient \( (\rho) \) between \( x \) and \( y \) largely depends on a variety of factors such as on-screen location, hand posture, and finger angle. Similar to the approach adopted in Bi and Zhai [11], we assume \( \rho \approx 0 \) without further knowledge of these factors.

Second, we adopted the dual Gaussian distribution hypothesis [9, 10] to estimate \( \sigma_x \) and \( \sigma_y \). For a point-and-click interface, the dual distribution hypothesis [9, 10] states that the variance of touchpoints \( (\sigma) \) has a linear relationship to \( d^2 \):

\[
\sigma^2 = \sigma_x^2 + \sigma_y^2 = \alpha \times d^2 + \sigma_a^2,
\]

where \( \alpha \) and \( \sigma_a \) are empirically determined parameters, and \( d \) is the target size.

Parameterizing the Dual-Gaussian Likelihood Model

Following the procedure reported in the previous research [10], we conducted a target acquisition study to obtain \( \alpha \) and \( \sigma_a \) values for Equation (6).

We recruited 36 participants (12 female) aged between 19 and 37 (average 25.4±4.2). Each participant was instructed to naturally select a circular target, which randomly appeared on a Nexus 5X touchscreen device. The study included four levels of target size (diameter): 8, 12, 16, 20 mm, each with 20 trials. To avoid over-fitting, we randomly divided the data into two sets: 29 (~80%) participants as the training set and the rest as the test set. Both data sets included a mix of two postures (index finger, thumb).

We established the touch model of the training set following the procedure described in [10]. More specifically, we first calculated the mean and standard deviation of the touchpoints relative to the target center. As conventional in Android and iOS, we assume the positive \( x \)-direction is right, and the positive \( y \)-direction is down. Table 1 shows the touch model parameters (in mm) of the training set data.

<table>
<thead>
<tr>
<th>( d )</th>
<th>( \mu_x )</th>
<th>( \mu_y )</th>
<th>( \sigma_x )</th>
<th>( \sigma_y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.472</td>
<td>0.327</td>
<td>1.372</td>
<td>1.598</td>
</tr>
<tr>
<td>12</td>
<td>0.648</td>
<td>0.348</td>
<td>1.756</td>
<td>2.010</td>
</tr>
<tr>
<td>16</td>
<td>0.628</td>
<td>0.411</td>
<td>1.843</td>
<td>2.350</td>
</tr>
<tr>
<td>20</td>
<td>0.973</td>
<td>0.348</td>
<td>2.138</td>
<td>2.451</td>
</tr>
</tbody>
</table>

Table 1. Touchpoint distribution for different target sizes. All units are in mm. The target center is \((0,0)\), \( d \) is the diameter of the target. \( \mu_x, \mu_y \) are the mean of the touchpoints. \( \sigma_x \) and \( \sigma_y \) are the standard deviations of the touchpoints.

We then ran linear regression for the variance of \( x \) and \( y \) directions against \( d^2 \). The estimations are shown in Figure 2. The \( \alpha \) and \( \sigma_a \) values serve as the parameters for Equation (6). To verify the trained parameters, we tested them on the \( \sigma \) values on the test dataset, the mean (SD) RMSE were 0.10 (0.11) \( mm \) on \( \sigma_x \), and 0.12 (0.04) \( mm \) on \( \sigma_y \) across different \( d \). This confirmed the validity of the model.

![Figure 2. Regression between the variance in \( x \) direction (\( \sigma_x^2 \)) and \( y \) direction (\( \sigma_y^2 \)) and the target width (\( d^2 \)).](image)

In this section, we provide the dual-Gaussian likelihood model for point-and-click command input by building a touch model that predicts the probability of observing a touchpoint \( s \) given \( c \) as the intended command. This touch model not only serves as the likelihood function for BayesianCommand but can also be used to generate touchpoints in the following simulation study in which we determined how fast the prior probability model should be updated from the command input history.

Determining the Update Rate of Prior Probability Model

After obtaining the prior probability and likelihood models, we investigated how fast the prior probability \( P(c_i) \) will be updated from the selection history. In other words, we decided the optimal \( k \) value in Equation (3) via a simulation study.

The simulation worked as follows. We first designed a 6 by 4 touchscreen grid layout for command selection (Figure 3). Each cell in the grid corresponded to a command candidate. We then implemented BayesianCommand as the principle for deciding the target on this grid interface, using the previously
We implemented a set of BayesianCommand-based criteria with different \( k \) values in the prior probability models. We used the touch models of test set users to generate the touchpoints, fed the touchpoints into this grid layout and evaluated the accuracy of different BayesianCommand models to determine which \( k \) value led to optimal performance.

We ran the simulation using the data collected in the previous study. We used the training set to train a touch model which served as the likelihood model in BayesianCommand (Figure 2). We developed independent touch models for each user in the test set to generate the touchpoints for testing.

On this grid layout, we assumed the target frequency follows the Zipfian distribution [79]:

\[
f(l; s, N) = \frac{1/l^s}{\sum_{n=1}^{N} (1/n^s)}, \tag{7}
\]

where \( N \) is the number of elements, \( l \in \{1, 2, \ldots, N\} \) is the rank of the element, and \( s \) is the value of the exponent characterizing the distribution. We randomly picked 12 square targets from a grid layout (Figure 3) and simulated two different distributions with exponent \( s = 1 \) and \( s = 2 \), based on 600 total selections. The generated frequencies were \((216, 106, 98, 79, 52, 25, 7, 6, 4, 4, 2, 1)\) for \( s = 1 \) and \((430, 142, 14, 3, 2, 2, 2, 1, 1, 1, 1)\) for \( s = 2 \). We assumed these 12 frequencies showed how frequently a target would be the intended command, and assigned these 12 frequencies to the selected 12 targets.

![Figure 3. The grid layout used in the simulation. The yellow block shows the simulated target. The finger illustrates the simulated touchpoint.](image)

Seven target sizes (4, 5, 6, 7, 8, 9, and 10 mm) were tested. We ran the simulation for every user in the test set separately. The target order was randomized. In each simulation trial, we picked one candidate as the target, and generated a touchpoint for selecting this target following the test user’s individual touch model. Given the touchpoint location, we then determined the selected target using BayesianCommand with different \( k \) values in the prior probability models. We repeated the procedure five times.

In total, the simulation included: 2 Zipfian distributions \( \times 7 \) target sizes \( \times 7 \) test users \( \times 600 \) trials \( \times 5 \) repetitions = 294000 simulation trials.

To determine which \( k \) should be used in the prior probability model, we compared the following \( k \) values when applying BayesianCommand:

1. Optimal \( k \). We searched for the optimal \( k \) in the prior probability model by initializing \( k \) to 0.1, and increasing it to 20 with a step length of 0.1. The \( k \) that led to the highest accuracy was optimal.

2. \( k = 1 \). We used \( k = 1 \) across users and conditions. We discovered that \( k = 1 \) performed well in pilot simulation runs and would like to see if it could be generalized.

**Results**

We calculated the target acquisition accuracy of each repetition as the total number of correct selections divided by the total number of selections averaged across the users in the test set. Figure 4 shows the average accuracy over the five repetitions.

![Figure 4. Average target acquisition accuracy by target size for different target frequency (Zipfian distribution with \( s = 1 \) or \( s = 2 \)) and \( k \).](image)

The optimal value of \( k \) varied across different runs, but generally fell within the range of \([0.5, 3]\). As shown in Figure 4, when \( k = 1 \), the average accuracy was close to the optimal accuracy. Therefore, we chose \( k = 1 \) in the prior probability model (Equation (3)) and used this value for implementation. Note that the choice of \( k \) is specific to our particular application. Different values should be selected depending on the actual scenario.

After determining the update rate (i.e., the \( k \) parameter), we conducted a study to evaluate BayesianCommand for point-and-click command input, using the proposed prior probability model and the dual-Gaussian likelihood model.

**Experiment I: Evaluating BayesianCommand for Point-and-Click Command Input**

The purpose of the study was to evaluate BayesianCommand for point-and-click command input. We expected BayesianCommand to outperform the typical boundary criterion because BayesianCommand was a more principled approach to handle the ambiguity in touch pointing input. We were also interested in comparing BayesianCommand with the BTC [10]. As explained in the related work section, BTC ignores the prior probability and only uses the likelihood probability to decide the target.

**Participants and Apparatus**

18 adults (4 females) aged between 21 and 35 (average 27.3 ± 3.3) participated in the study. 16 participants were right-handed. The self-reported average usage time of mobile phones was 24.5 hours per week. We used a Ticwatch S Smartwatch with a 45mm diameter screen in the study (Figure 5a).

**Experiment Setup**

The study was a within-subject design. There were 2 independent variables: target size and target deciding principle. We evaluated two target sizes: 3mm and 4mm square targets on a 4 by 6 grid layout. The target deciding principles included three levels:
We randomly selected 12 items as targets. We used the same set of targets across participants and conditions. The target positions were fixed, as shown in Figure 5b. Target item frequencies were generated according to Zipfian distribution with exponent $s = 1$ based on 30 selections. The generated frequencies, i.e., the number of occurrences, were $(7, 5, 4, 4, 2, 2, 1, 1, 1, 1, 1, 1)$. The frequency assignments were randomized across participants and conditions. Participants were not informed of the frequency distribution of the items or the position of the most frequent items.

BayesianCommand reduced the error rates: on 4mm targets, it reduced the error rate by 37.2% and 39.3% compared to the boundary criterion and BTC; on 3mm targets, the error rate reduction was 26.9% and 28.0% respectively. ANOVA showed there was a significant main effect of the target deciding principle on the error rates ($F_{2,34} = 7.98, p < .005$). Pairwise comparisons with Bonferroni adjustment showed that the difference was significant between BayesianCommand vs. BTC ($p = .004$) and between BayesianCommand vs. boundary criterion ($p = .017$). The 4mm targets were less error-prone and easier to select than the 3mm targets. ANOVA showed the differences were significant for target size ($F_{1,17} = 39.93, p < .005$). We did not observe a significant interaction effect of target deciding principle × target size ($F_{2,34} = 0.33, p = .72$).

Target acquisition time. We compared the average target acquisition time, which was the elapsed time from a target being highlighted on the screen to the time the participant made the first selection. We only considered the first attempt in every trial, regardless of whether it was correct or not.
16 participants could correctly finish all trials within 3 attempts. Although we only allowed 3 failed attempts per trial, our investigation showed it had minor effects on the overall error rates. The empirical study showed that BayesianCommand outperformed the boundary criterion and BTC. It also showed that learning was advantageous to frequency-based menu adaptation (e.g., touchpoint-to-center distance). Since the targets were of the same size and were arranged in a grid with no gaps between them in our experiment, BTC was equivalent to boundary criterion: the item whose boundary contains the touchpoint is also the target that has the shortest distance to the touchpoint. Although we only allowed 3 failed attempts per trial, our investigation showed it had minor effects on the overall error rates. 16 participants could correctly finish all trials within 3 attempts in all conditions. The rest two failed 0.56% (= 4/720) of the trials 3 times, which were for 3mm targets in the Bayesian-Command condition. These 2 participants were able to select the targets correctly when the same trials repeated, indicating that these items remained accessible for them.

How can designers or developers leverage the benefits of BayesianCommand? Many applications and software have collected usage patterns of menus, buttons, and commands, e.g., the command usage frequency of Microsoft Word 2003 [29]. These accumulated frequencies and patterns could serve as the prior probability for adopting BayesianCommand; the system can then adapt the prior probability as a user is interacting with the system. If no prior command history is available, the system can assume every command is equally probable and learn the distribution probability as more actions are observed. We would also like to point out that BayesianCommand works under the assumption that the command input distribution model can be established. It might not show significant benefits for some applications if their command frequency model is not that obvious (e.g., Maps).

**APPLICATION 2: APPLYING BayesianCommand TO WORD-GESTURE SHORTCUTS**

In this application, we investigated how to apply BayesianCommand for word-gesture shortcuts – entering a command by drawing the word-gesture [37, 74] of the command name (e.g., Figure 7). We first propose a two-step likelihood model for gestural input and combine it with the previously proposed prior probability model. A user study showed using BayesianCommand outperformed the existing deterministic principle, which selects command simply based on the highest matching score from a gesture decoder.

Note that the two-step likelihood model is not a gesture decoder; it is a model that uses a gesture decoder and combines the decoding outcomes with available command candidates to estimate \( P(s|c) \), where \( s \) is an input gesture and \( c \) is a command. It is independent of a gesture decoder. In this application, we used the i’sFree gesture decoder [78] as an example, but it can be replaced with other decoders such as \( SHARK^2 \) [37].

**Table 2. Average target acquisition time in seconds.**

<table>
<thead>
<tr>
<th>Target size</th>
<th>Boundary criterion</th>
<th>BTC [10]</th>
<th>BayesianCommand</th>
</tr>
</thead>
<tbody>
<tr>
<td>4mm</td>
<td>0.74 ± 0.25</td>
<td>0.78 ± 0.26</td>
<td>0.71 ± 0.15</td>
</tr>
<tr>
<td>3mm</td>
<td>0.86 ± 0.26</td>
<td>0.88 ± 0.24</td>
<td>0.94 ± 0.44</td>
</tr>
</tbody>
</table>

The target size had a main effect on the target acquisition time \( (F_{1,7} = 6.98, p = .017) \). We did not observe a main effect of the target deciding principle \( (F_{2,34} = 0.30, p = .75) \) or any interaction effect \( (F_{2,34} = 1.45, p = .25) \). As shown in Table 2, using different target deciding principles had little effect on the target acquisition time.

**Subjective feedback.** We used a subset of NASA-TLX [30] questions to measure the perceived workload of the task, including mental demand, physical demand, and effort. The rating was from 0 to 10. The lower the rating the better.

For 4mm targets, the median ratings were 4 (mental demand), 3 (physical demand), and 3 (effort) for BayesianCommand; 3, 4, 4 for the boundary criterion, and 3.5, 4, 4 for BTC. For 3mm targets, the median ratings were 4, 5, 5 for BayesianCommand; 5, 5, 5 for the boundary criterion, and 5, 6 for BTC. BayesianCommand was perceived slightly less mentally demanding than the other two principles on 3mm targets.

**Discussion**

The empirical study showed that BayesianCommand outperformed the boundary criterion and BTC. Using BayesianCommand substantially reduced the touch pointing error rate for both large and small menu targets. The reduction was especially remarkable for small targets: around 26% over both boundary criterion and BTC. It also showed that learning the prior probability distribution and combining it with the likelihood function outperforms using the likelihood function alone. Since BayesianCommand is algorithmic, these improvements were achieved without altering any UI layout, which was advantageous to frequency-based menu adaptation (e.g., morphing menu [16]), and thus less likely to slow users down or reduce user satisfaction [20].

BTC had almost identical error rates to the boundary criterion. According to the definition of BTC (Equation (1) in [10]), when the target sizes are equal, BTC is equivalent to comparing the distance from the touchpoint to the target center (touchpoint-to-center distance). Since the targets were of the same size and were arranged in a grid with no gaps between them in our experiment, BTC was equivalent to boundary criterion: the item whose boundary contains the touchpoint is also the target that has the shortest distance to the touchpoint. Although we only allowed 3 failed attempts per trial, our investigation showed it had minor effects on the overall error rates. 16 participants could correctly finish all trials within 3 attempts in all conditions. The rest two failed 0.56% (= 4/720) of the trials 3 times, which were for 3mm targets in the Bayesian-Command condition. These 2 participants were able to select the targets correctly when the same trials repeated, indicating that these items remained accessible for them.

How can designers or developers leverage the benefits of BayesianCommand? Many applications and software have collected usage patterns of menus, buttons, and commands, e.g., the command usage frequency of Microsoft Word 2003 [29]. These accumulated frequencies and patterns could serve as the prior probability for adopting BayesianCommand; the system can then adapt the prior probability as a user is interacting with the system. If no prior command history is available, the system can assume every command is equally probable and learn the distribution probability as more actions are observed. We would also like to point out that BayesianCommand works under the assumption that the command input distribution model can be established. It might not show significant benefits for some applications if their command frequency model is not that obvious (e.g., Maps).
If we view $s$, $w$, and $c$ as random variables, the following graphical model describes their dependencies (Figure 8):

![Conditional dependencies:]

$$P(s|c) = \sum_{i=1}^{N} P(s, w_i | c) = \sum_{i=1}^{N} P(s|w_i, c)P(w_i|c), \quad (8)$$

where $s$ is the input gesture, $w_i$ is a decoded word candidate from a gesture typing decoder, $N$ is the total number of decoded word candidates, and $c$ is a command candidate.

The graphical model (Figure 8) suggests that $c$ and $s$ are conditional independent given $w$. Therefore, Equation (8) can be further expressed as:

$$P(s|c) = \sum_{i=1}^{N} P(s|w_i, c)P(w_i|c) = \sum_{i=1}^{N} P(s|w_i)P(w_i|c). \quad (9)$$

Equation (9) is our two-step likelihood model. As shown, the key of using this model is to obtain $P(s|w_i)$ and $P(w_i|c)$. These two terms can be calculated as follows.

The term $P(s|w_i)$ represents the probability of observing the input gesture $s$ if $w_i$ is the target word. From a gesture typing decoder’s perspective, it is the spatial score of $w_i$ given $s$ is the input gesture [37, 74]. In this research, we adopted the eyes-free gesture decoder [78] to obtain it. We swapped the original language model used in the eyes-free gesture decoder [78] with the command set $C = \{c_1, c_2, \ldots, c_n\}$, because our goal was to predict an available command in a command set, rather than as a general text entry method.

The term $P(w_i|c)$ represents the probability of inputting the word $w_i$ if the $c$ is the intended command. Since a command might be triggered by different words (e.g., launching a clock with “clock,” “timer,” or “watch”), we calculate $P(w_i|c)$ as follows. For a given command $c$, we first form a set of words corresponding to it: $M = \{m_0, m_1, m_2, \ldots, m_K\}$ from a thesaurus (e.g., thesaurus.com), where $m_i$ is a valid word for triggering $c$. If a decoded word candidate $w_i$ does not belong to this set, we assume $P(w_i|c) = 0$. Otherwise, $P(w_i|c) = \frac{1}{K}$, assuming that each word in this thesaurus has equal probability for triggering command $c$.

Experiment II: Evaluating BayesianCommand for word-gesture shortcuts

We conducted a user study to evaluate using BayesianCommand for word-gesture shortcuts. We compared BayesianCommand with the typical deterministic strategy for deciding target command.

Participants and Apparatus

18 adults (4 females) aged between 23 to 31 (average 26.9±2.5) participated in the study. The self-reported average usage time of mobile phones was 30.1 hours per week. 17 participants were right-handed. The median of self-reported familiarity with Qwerty layout (1: not familiar at all, 5: very familiar) was 4.5. The median familiarity with gesture typing was 3. A Google Pixel running Android 9.0 was used for the study, as shown in Figure 9b.

Experiment Setup

The study was a within-subject design. The independent variable was the command deciding principles with three levels:

- **BayesianCommand**: we applied BayesianCommand (Algorithm 1) as the principle to decide the target command, using Equation (3) to calculate prior probability and the two-step likelihood model in Equation (9) to calculate likelihood. Similar to Experiment I, we chose $k = 1$ in the prior probability model.

- **Likelihood-only**: the command candidate with the highest likelihood value (the two-step likelihood model in Equation (9)) is the intended target. It uses likelihood value only. We included this condition to understand how much performance gain in BayesianCommand was provided by the prior probability, and how much gain was provided by the likelihood function. This approach can also be viewed as using BayesianCommand but assuming all the command candidate has equal prior probability: under this assumption $P(c|s)$ will be determined by the likelihood $P(s|c)$ only.
• Deterministic approach. This is the typical target deciding principle for gestural command input. The gesture decoder used a set of available command names as the dictionary and matched the input gesture with the words in this dictionary. The word with the highest matching score was the intended command.

We used the same gesture decoder [78] across all three conditions. We swapped the language model in the original decoder with the command set used the study (including all the trigger words for each command in Appendix A). The composition of the command set is explained in detail later.

Before the study, participants were shown the 20 commands and their corresponding graphical representations. Participants needed to memorize ≥ 80% of the commands before they could proceed to the formal study: they had to recall at least one of the trigger words of the commands. This procedure ensured that the results wouldn’t be affected by participants’ familiarity with the commands, or any external cause other than the three principles.

For each trial, an icon was first displayed on the screen as the target command. The participants then gestured the word in the white space below it to trigger the command. The input command name was shown to the participants after the finger lifted off from the screen, regardless of whether it’s the intended command or not, as shown in Figure 9a. A trial was completed if the input command was correct or three failed attempts were made. For each condition, participants first performed a warm-up session of two trials, followed by 60 trials divided into two blocks. Participants were allowed to take a short break after the completion of each block. Each participant performed the task three times, with different target deciding principles each time. The orders of three target deciding principles were fully counterbalanced across participants.

A subset of 12 commands was picked as the targets. The same set of commands were used across participants. We used the same item frequencies as in Experiment I, i.e., the number of occurrences for the commands was (7, 5, 4, 4, 2, 2, 1, 1, 1, 1, 1, 1). Participants were not informed of the frequency distribution of the items. The rest of the experiment design is similar to Experiment I. For each command, a set of 10 additional words for triggering this command was created from [18]. The list of commands is shown in Table 3. The command set included 20 commands. Each command has 11 corresponding trigger words (10 synonyms and the command name). The command set includes 220 words in total, which was incorporated into the decoder used in the study.

In total, the study included: 18 participants × 3 principles × 60 trials = 3240 trials.

Results

Error rates. This metric measures the ratio of the number of incorrect gesture inputs over the total number of trials. The average error rates are shown in Figure 10.

BayesianCommand lowered the error rate by 39.9% and 24.5% compared to the deterministic and the likelihood-only approaches. ANOVA showed a significant main effect of the command deciding principle on the error rates ($F_{2,34} = 5.56, p < .01$). Pairwise comparisons with Bonferroni adjustment showed that the difference was significant between BayesianCommand vs. deterministic strategy ($p = .01$), but not for BayesianCommand vs. likelihood-only ($p = .13$) or likelihood-only vs. deterministic strategy ($p = .54$).

Command triggering time. We compared the average command triggering time, which was the elapsed time from a target command icon being shown on the screen to the completion of a gesture command. The average command triggering time was $2.91 ± 1.32$ seconds for the deterministic strategy, $2.83 ± 0.98$ seconds for the likelihood-only approach, and $2.98 ± 1.35$ seconds for BayesianCommand. We did not observe a main effect of the principle ($F_{2,34} = 0.19, p = .83$). This result also indicates that using different principles had little effect on the overall command triggering time.

Use of trigger words. We examined the trigger words of each command in the deterministic condition. We counted the number of unique trigger words (i.e., the decoded word from the gesture recognizer) of each command when it was successfully triggered. We excluded the BayesianCommand

Table 3. List of the 20 commands. The underlined commands were tested in the experiment.
and the likelihood-only conditions as they used probabilistic approaches, and the decoded word did not always correspond to the command label. The average number of trigger words was 2.92 ($SD=1.38$) across the 12 tested commands. This result supported the validity of the graphical model (Figure 8) and our hypothesis that multiple trigger words could be used for a command.

Subjective feedback. A subset of NASA-TLX [30] questions was used to measure the perceived workload of the task. The range of the ratings was 1 to 10 (the smaller the rating, the better). The median ratings were 4 (mental demand), 2.5 (physical demand), and 3 (effort) for BayesianCommand; 5, 4, 5 for the deterministic method, and 4.5, 4, 5 for likelihood-only. BayesianCommand was perceived less demanding than the other two principles in all questions.

Discussion
The results showed that BayesianCommand effectively improved the input accuracy for word-gesture shortcuts. It reduced the command triggering error rate by 39.9% compared to the deterministic method. Notably, it performed better than the deterministic strategy when the gesture decoder failed to distinguish commands in similar shapes. For example, the average input error rate for the command “cut” was 34.2% for the deterministic method, because its gesture trace was very similar to “copy” on a Qwerty keyboard. 65% of input for “cut” was misrecognized to “copy” for the deterministic method. BayesianCommand reduced the error rate to 16.2%, showing that combining prior and likelihood resolved some ambiguity introduced in the gesture decoding. BayesianCommand also outperformed the likelihood-only approach by 24.5%. The results also substantiated our claim that fully applying the Bayes’ theorem could be adopted in various applications to deal with the input uncertainty.

Likewise, limiting the number of failed attempts to 3 had minor effects on the results. Nine participants correctly finished all trials in under three tries in all conditions. For the rest 9 participants, the average percentage of trials that failed three times was $1.30 \pm 1.67\%$ for the deterministic approach, $1.02 \pm 1.53\%$ for likelihood-only, and $1.30 \pm 2.78\%$ for BayesianCommand. Compared to the other two conditions, BayesianCommand did not introduce more trials that failed three times or contained inaccessible commands. The percentage increased over Experiment I as gesture input is a more complex procedure with higher cognitive and motor execution demands.

While the application focused on word-gesture shortcuts, BayesianCommand could be extended to other gestural command input methods, e.g., Command Strokes [38], CommandBoard [1], or HotStrokes [17]. The prior probability and the likelihood models are independent of the gesture decoder, thus being applicable to other gestural command input methods with minor modification. Investigation on the generalization and other recall-based methods are interesting future work.

LIMITATIONS AND FUTURE WORK
A side effect of incorporating prior probability is that it could make the less frequent items difficult to select. Although our experiments did not show severe consequences, the infrequent items would become more and more challenging to select as their prior probabilities are decreasing [44]. We could mitigate the problem by adding a lower bound for command frequency to ensure that no command will become hard to access or inaccessible. In real-world applications, we could leverage user actions to address them. For example, if the previous selection is an error (back/cancel button is pressed immediately), the probability of this command will decrease for the subsequent command input, preventing users from repeatedly selecting the same incorrect command and increasing the chance of selecting the intended one.

Our investigation on point-and-click input was under the assumption that the target size decided the likelihood model $P(s|c)$. Such a hypothesis did not reflect the possibility that users may adapt their interaction behavior as frequent items were becoming easier to select. It is worth investigating whether adapting $P(s|c)$ according to user interaction experience would lead to more accurate likelihood models.

Additionally, BayesianCommand is essentially adjusting the command activation space according to command frequencies. In the current investigation, we did not communicate such adjustment to users via visuals to avoid disruption caused by interface visual changes. It is worth investigating whether communicating this adjustment would affect users’ interaction behavior and how we develop more accurate likelihood models to capture it.

Our two examples (i.e., point-and-click and gestural command input) were two experiments demonstrating the effectiveness of BayesianCommand. Many design choices for the models (e.g., the value of $k$, the command set, and the trigger words) were specific to these experiment settings. As shown in [21], we could make more mature decisions for real-world applications with more contextual information such as the interaction scenarios, command set sizes, and users’ preferences. For example, if we can access detailed command input history (e.g., command usage patterns for Microsoft Word [29]), we may build a more advanced prior model (e.g., n-gram command sequence) and follow the similar principle proposed in this paper to improve the command selection accuracy.

CONCLUSIONS
In this paper, we have formalized Bayes’ theorem as a guiding framework for deciding the target in command input. To support this principle, we have developed three models: (1) a prior probability model, (2) a dual-Gaussian likelihood model for point-and-click, and (3) a two-step likelihood model for word-gesture shortcuts. Our experiments showed that applying BayesianCommand with the proposed models substantially improved the command input accuracy. Compared to the deterministic principles or applying the principle partially, BayesianCommand reduced the command input error rate by 26.9% and 28.0% for point-and-click, and by 39.9% and 24.5% for word-gesture command input.

REFERENCES


Table 4 shows the 11 trigger words of the 20 commands in Experiment II. The words in bold were the trigger words used by the participants to trigger the corresponding commands in the deterministic condition. Note that these words represent the decoding output from the gesture recognizer, not necessarily what the participants intended to input.

<table>
<thead>
<tr>
<th>Command</th>
<th>Trigger words</th>
</tr>
</thead>
<tbody>
<tr>
<td>calculator, calculators, calculate, calculation, computer, computation, microcomputer, count, appraise, spreadsheet</td>
<td></td>
</tr>
<tr>
<td>camera, cameras, camcorder, video, photograph, photographer, camerman, videocamera, tripod, lens, projector</td>
<td></td>
</tr>
<tr>
<td>clock, clocks, timer, time, dial, watch, stopwatch, alarm, tick, seconds, wristwatch</td>
<td></td>
</tr>
<tr>
<td>copy, copying, copyist, replicate, replica, imitate, reproduce, emulate, duplicate, plagiarize, clone</td>
<td></td>
</tr>
<tr>
<td>cut, cutting, slice, trim, reduce, prune, shorten, truncate, curtail, scissors, clippers</td>
<td></td>
</tr>
<tr>
<td>delete, deleting, deleted, deletes, deletion, remove, uninstall, eliminate, omit, overwrite, discard</td>
<td></td>
</tr>
<tr>
<td>download, downloads, downloadable, upload, redownload, load, downloder, browse, access, file-sharing, homepage</td>
<td></td>
</tr>
<tr>
<td>edit, editing, editor, edits, edited, annotate, annotated, essay, alter, revise, rewrite</td>
<td></td>
</tr>
<tr>
<td>file, files, filing, filename, filed, refile, folder, document, documents, archive, directory</td>
<td></td>
</tr>
<tr>
<td>help, helping, helps, helped, assist, assistance, aid, support, avail, advice, service</td>
<td></td>
</tr>
<tr>
<td>keyboard, keyboards, touchpad, trackpad, keypad, qwerty, stylus, numberpad, typewriter, typing, laptop</td>
<td></td>
</tr>
<tr>
<td>mail, mails, mailbox, mailing, e-mail, email, spam, letter, postal, post, mailed</td>
<td></td>
</tr>
<tr>
<td>network, networks, networked, net, internet, web, cable, channel, connectivity, networking, interconnect</td>
<td></td>
</tr>
<tr>
<td>print, printing, printer, printed, reprint, handwritten, photocopy, publish, publication, booklet, distribute</td>
<td></td>
</tr>
<tr>
<td>recent, subsequent, recently, latest, previous, past, earlier, prior, preceding, later, coming</td>
<td></td>
</tr>
<tr>
<td>rotate, rotation, rotational, tilted, pivot, tilt, rotating, rotated, revolving, swivel, spin</td>
<td></td>
</tr>
<tr>
<td>search, searches, searching, retrieve, discover, check, find, look, quest, searcher, scour</td>
<td></td>
</tr>
<tr>
<td>share, shared, sharing, exchange, swap, commonality, pool, combine, express, collect, common</td>
<td></td>
</tr>
<tr>
<td>weather, inclement, meteorological, windy, forecast, forecaster, winter, foggy, thunderstorm, meteorologist, blizzard</td>
<td></td>
</tr>
<tr>
<td>zoom, zoom-in, close-up, enlarge, magnify, magnifier, scroll, augment, enhance, expand, amplify</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. The trigger words for the 20 commands used in Experiment II.

APPENDIX

TRIGGER WORDS OF THE COMMANDS

Table 4 shows the 11 trigger words of the 20 commands in Experiment II. The words in bold were the trigger words used by the participants to trigger the corresponding commands in the deterministic condition. Note that these words represent the decoding output from the gesture recognizer, not necessarily what the participants intended to input.