Gestures à Go Go: Authoring Synthetic Human-like Stroke Gestures Using the Kinematic Theory of Rapid Movements

LUIS A. LEIVA and DANIEL MARTÍN-ALBO, Universitat Politècnica de València
RÉJEAN PLAMONDON, École Polytechnique de Montréal

Training a high-quality gesture recognizer requires providing a large number of examples to enable good performance on unseen, future data. However, recruiting participants, data collection and labeling, etc. necessary for achieving this goal are usually time-consuming and expensive. Thus, it is important to investigate how to empower developers to quickly collect gesture samples for improving UI usage and user experience. In response to this need, we introduce Gestures à Go Go (G3), a web service plus an accompanying web application for bootstrapping stroke gesture samples based on the kinematic theory of rapid human movements. The user only has to provide a gesture example once, and G3 will create a model of that gesture. Then, by introducing local and global perturbations to the model parameters, G3 generates from tens to thousands of synthetic human-like samples. Through a comprehensive evaluation, we show that synthesized gestures perform equally similar to gestures generated by human users. Ultimately, this work informs our understanding of designing better user interfaces that are driven by gestures.

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces; I.5.2 [**Pattern Recognition**]: Design Methodology; I.5.4 [**Pattern Recognition**]: Applications

General Terms: Algorithms, Design, Human Factors

Additional Key Words and Phrases: Gesture Synthesis; Bootstrapping; Gesture Recognition; Strokes; Marks; Symbols; Unistrokes; Multistrokes; Multitouch; Kinematics; User Interfaces; Rapid Prototyping

1. INTRODUCTION

Gestures are increasingly becoming a predominant input modality in today's user interfaces (UIs). Gesture interaction is possibly one of the most researched areas in Human-Computer Interaction (HCI), with a long history that started as early as 1960, with the Sketchpad project [Sutherland 1963] and the RAND tablet [Davis and Ellis 1964]. Gestures can be mid-air (more prominent in gaming applications) or stroke based (more prominent in mobile applications). We are particularly interested in the latter type, motivated by the fact that stroke gestures are becoming more and more relevant to mainstream products such as touchscreen-capable devices like smartphones and tablets.

Strokes gestures represent the movement trajectory of one or more contact points on a sensitive surface. Stroke gestures are sometimes also called "pen gestures", "hand drawn marks", "hand drawn gestures", "hand markings", or "markings" [Zhai et al. 2012]. Stroke gestures tend to give richer perceptual cues to the user, to form an association between the shape of the gesture and the meaning of the command [Appert and Zhai 2009]. Compared to traditional interactions, stroke gestures have the potential to lower cognitive load and the need for visual attention [Zhai et al. 2012]. Stroke

This work is part of the Valorization and I+D+i Resources program of VLC/CAMPUS and has been funded by the Spanish MECD as part of the International Excellence Campus program. This work has also been partially supported by the Spanish MINECO under grant TIN2014-37475 and FPU scholarship AP2010-0575, and by NSERC-Canada grant RGPIN-2014-04946.
Authors' addresses: L. A. Leiva (**llt@acm.org**, corresponding author) and D. Martín-Albo (**damarsi1@upv.es**) are with the PRHLT Research Center, Universitat Politècnica de València. Camí de Vera, 46022 València, Spain; Réjean Plamondon (**rejean.plamondon@polymtl.ca**) is with the Laboratoire Scribens, Département de Génie Électrique, École Polytechnique de Montréal. 6079, succursale Centre-Ville, Montréal, QC, Canada.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

gestures also may improve the usability of UIs, by replacing standard shortcuts by more accessible triggers.

Stroke gestures have existed in the industry for decades. Early examples of commercial products that successfully incorporated gestures are, e.g., PDAs like the Palm Pilot or the Apple Newton, and the Windows Tablet. These devices featured the Graffiti and Unistroke shorthand writing systems, which used a single stroke Roman letter-like gesture vocabulary. Today, stroke gestures are mostly used in consumer devices for executing simple actions, such as pinching a picture to zoom in/out, swiping to reveal an options menu, or panning to switch between apps. Nevertheless, stroke gestures are increasingly being incorporated to facilitate random access to smartphone contents, such as invoking a command hidden in an advanced settings menu or quickly searching for a friend's email in the contacts list. Therefore, it is expected that strokes gestures will make a notable impact in consumers' lives.

In general, any application that is driven by gestures must rely on some recognition-based techniques. These techniques often require expert knowledge in pattern recognition or machine learning, something that is typically beyond the reach of many developers and UI designers. Furthermore, recruiting participants, data collection and labeling, etc. necessary for using these techniques are usually time-consuming and expensive. Thus, it is important to investigate how to empower developers to quickly collect gesture samples for improving UI usage and, as a result, user experience (UX).

In this article, we introduce Gestures à Go Go (G3), a web service plus an accompanying web tool for bootstrapping stroke gesture samples based on the kinematic theory of rapid human movements [Plamondon 1995a; Plamondon 1995b]. The user only has to perform a gesture sample once, and G3 will generate from tens to thousands of synthetic human-like samples. This aims for creating better gesture recognizers, eliminating the overhead of recruiting and data collection, and reducing the need for expert knowledge in machine learning. In addition, the user can get a gesture recognizer together with the synthesized data. As such, the outcome of G3 can be directly incorporated into production-ready applications. This work is framed within the so-called programming by example paradigm [Halbert 1984], where an end-user teaches a computer new behaviors by providing examples instead of programming them through machine commands. Ultimately, this work informs our understanding of designing better UIs that are driven by gestures.

This article offers the following contributions:

(1) A RESTful web service aimed at bootstrapping stroke gestures from just one given example.
(2) A web application that interfaces with our web service and allows users to design their own gesture sets.
(3) Integration of popular gesture recognizers in our web application, aimed at building a working prototype in few clicks.
(4) Extensive validation of this work through a number of experiments and an informal use case.

1.1. Organization

The remainder of this article is organized as follows. Section 2 discusses related work. Section 3 introduces the theoretical framework on which this work is based. Section 4 and Section 5 describe G3 implementation details. Section 6 and Section 7 evaluate G3 from different perspectives. Section 8 discusses the implications and limitations of this work. Finally, Section 9 gives a number of concluding remarks and provides opportunities for future work.

A Note for Practitioners. We believe this article will be useful for both practitioners and researchers. However, practitioners are likely to be mostly interested in a small portion of the article: the system details (Section 4 and Section 5) and the general discussion (Section 8). We encourage practitioners to read also Section 7 as that section provides insights on an informal use case of our system.

2. RELATED WORK

There is a huge body of research on gesture interaction. Over the past few years, new touchscreen-based products have taken off rapidly, boosting the popularity of stroke gestures as commands and

symbols. An excellent integrative review of the state-of-the-art research on stroke gestures is provided by Zhai et al. [2012]. Below we comment on prior works that bear direct relevance to this article. We comment on a later section the theoretical framework on which G3 is based.

2.1. Gesture Recognition

One of the first research studies on stroke gestures was conducted by Wolf and Morrel-Samuels [1987], who reported that "the use of gestures is of particular interest in an interface which allows the user to write directly on the surface of a display with a stylus." Stroke gestures led to good intra-subject consistency and there was consensus regarding gestures being perceived as easy to use and remember. Later on, Goldberg and Richardson [1993] introduced the general philosophy of simplifying gesture sets, motivated by the fact that simpler is faster to write, less prone to recognition error, and can be entered in an "eyes-free" manner, which requires little space onscreen.

Gesture recognition has its own roots in sketching and handwriting recognition. Classification methods include, among others: linear discriminant analysis [Rubine 1991], template matching [Connell and Jain 2000], decision trees [Belaid and Haton 1984], neural networks [Marzinkewitsch 1991], hidden Markov models [Koschinski et al. 1995], parsing grammars [Costagliola et al. 2004], support vector machines [Bahlmann et al. 2001], principal component analysis [Deepu et al. 2004], or *ad-hoc* recognizers [Leiva et al. 2014]. Typically, gesture recognition takes place after a "pointer up" event, although it is possible to perform it continuously, in an incremental fashion [Bau and Mackay 2008].

Most gesture recognizers for prototyping UIs are based on the template matching (or instance-based) approach: a query gesture is geometrically compared against a number of stored templates, using 1 nearest-neighbor for classification and either Euclidean distance or a Mean Square Error (MSE) score as dissimilarity measures. Template matchers are a very viable and a relatively simple solution for recognizing gestures, and can be adapted to personalized user gestures. Popular examples of these recognizers among the HCI literature are part of the so-called "$ family": $1 [Wobbrock et al. 2007], $N [Anthony and Wobbrock 2010], and their newer versions Protractor [Li 2010] and $N-Protractor [Anthony and Wobbrock 2012], respectively—the only difference with their previous versions is a closed-form algorithm to match gesture templates that provides a significantly better performance. More recently, Vatavu et al. [2012a] introduced $P, a sequential-agnostic recognizer where strokes are treated as a cloud of 2D points, discarding thus stroke number, order, and direction.

2.2. Gesture Boostrapping

Training data is the key factor to build a competitive gesture recognizer. For instance, the Freehand Formula Entry System (FFES) suggests 20–40 examples per symbol per user [Smithies et al. 2001]. Koch et al. [2010] studied gesture recognition using a Nintendo Bluetooth Wiimote controller as input, and found that 120 training patterns of accelerometer-based data are a lower bound; below that threshold the error rate increased dramatically. Another important factor worth mentioning toward the adoption of one gesture recognizer over another is performance, typically represented by execution time and memory usage. It is here where the above-mentioned template matchers usually excel, and the main reason why we chose them to conduct our experiments (Section 6).

A number of approaches are aimed at simplifying the process of designing gesture sets. An interesting example is Gesture Script [Lü et al. 2014], which allows developers to describe the structure of a gesture and its parts. With this information, Gesture Script synthesizes new gesture samples by changing the relative scale of each part and their rotation angles. Unfortunately, Gesture Script can only deal with unistroke gestures that are performed in a unique way. Besides, having to provide too detailed information for each gesture can be time-consuming for the user.

Gesture Marks [Ouyang and Li 2012] allows users to access applications and websites using gestures without having to define them first, by means of crowdsourcing and the combination of gesture and handwriting recognizers. Gestalt [Patel et al. 2010] supports the entire process of applying machine learning: implementing a classification pipeline, analyzing data as it moves through that pipeline, and easily transitioning between them. CrowdLearner [Amini and Li 2013] enables a

developer to quickly acquire a usable recognizer for their specific application by spending a moderate amount of money (about $10) in a short period of time (about 2 hours).

Notable systems aimed at building gesture recognizers tailored to developers and end-users include MAGIC [Ashbrook and Starner 2010; Kohlsdorf and Starner 2013] and Gesture Follower [Caramiaux et al. 2014]. Both systems provide the user with a means of generating synthetic gesture samples in 3D space. MAGIC performs local perturbations to the resampled points of a gesture, whereas Gesture Follower introduce some variations to a gesture template using Viviani's curve formulation. These tools decrease the number of iterations needed to build a fast and stable gesture recognition interface, however there is no evidence that they can produce human-like samples. Further, these artificially generated samples usually perform poorly since they do not illustrate sufficient variation required for high-quality training [Plamondon et al. 2014]. However, these prior projects demonstrate the ongoing importance of and interest in improving gesture recognition by acquiring large data samples.

2.3. Gesture Design Tools

Gesture design tools are well studied in the HCI community [Hong and Landay 2000; Kin et al. 2012]. Example-based approaches like GRANDMA [Rubine 1991], Agate [Landay and Myers 1993], GDT [Long et al. 1999], Gesture Coder [Lü and Li 2012], or Gesture Studio [Lü and Li 2013] allow developers to create and test gestures by recording examples.

There are a number of similar systems tailoring end-users. For instance, EventHurdle [Kim and Nam 2013] is a visual gesture-authoring tool to support designers' explorative prototyping. It supports remote gestures from a camera, handheld gestures with physical sensors, and touch gestures by utilizing a touchscreen. Also, designers can visually define and modify gestures through interaction workspace and graphical markup language with hurdles. A CAPpella [Dey et al. 2004] is another programming by demonstration environment intended for end-users. It allows users to "program" their desired behavior without writing any code, by demonstrating it to the system and by annotating the relevant portions of the demonstration. GestIT [Spano et al. 2013] allows declarative and compositional definition of gestures for different categories (e.g. multitouch and full-body gestures).

Other tools like iGesture [Signer et al. 2007] and InkKit [Plimmer and Freeman 2007] offer several algorithms through a high-level interface. However, they are more intended for recognition benchmarking. In this line, Beuvens and Vanderdonckt [2012] devised a desktop application for facilitating the integration of gestures in UIs by describing the roles of the gesture specialist and other stakeholders involved in the development life cycle. G3 preserves these core interactions, though its goal goes further. In short, our tool provides the user with an unprecedented capability in terms of time savings, since only one example per gesture class has to be specified. Furthermore, because G3 is web-based, the user does not have to install additional software to start using our tool.

2.4. Gestures as A Service

To close this related work section, we should mention a number of previous works that offered gesture development over the Web. First, van Seghbroeck et al. [2010] described WS-Gesture, a framework that allows users to control different devices based on the DPWS (Devices Profile Web Services) standard. Although an important piece of the framework, gesture recognition itself was not the topic of their research.

Second, Vatavu et al. [2012b] developed Gesture Profile for Web Services (GPWS), an event-driven architecture based on the service-oriented architecture (SOA) standard. GPWS focuses on delivering gesture recognition services, for which a user must provide the data required to train a recognizer. On the contrary, G3 bootstraps gesture generation from one gesture example provided by the user. What is more, G3 uses the Representational State Transfer (REST) web service protocol to bootstrap gestures. Among other advantages, REST is a stateless, cacheable, layered system architecture. Further, REST has gained widespread acceptance across the Web as a simpler alternative to other web service protocols such as WSDL.

Third, MAGIC 2.0 [Kohlsdorf et al. 2011] is a web-based prototype that allows users to interface with the MAGIC framework [Ashbrook and Starner 2010]. Users can design motion gesture datasets while testing for and preventing false positives, which is important for sensor-based prototypes, as the user may accidentally trigger unintended gestures.[1] Thus MAGIC is related to gesture classification systems, instead of gesture bootstrapping. G3 fills this gap and optionally allows for quickly creating a ready-to-use, simple gesture recognizer that is suitable for use on prototypes in different programming languages.

3. KINEMATIC THEORY OF RAPID HUMAN MOVEMENTS

Many models have been proposed to study human movement production; e.g., models relying on neural networks [Bullock and Grossberg 1988], equilibrium point models [Feldman 1966], behavioral models [Thomassen et al. 1983], coupled oscillator models [Hollerbach 1981], kinematic models [Meyer et al. 1990; Plamondon 1995a], or models exploiting minimization principles [Neilson 1993; Flash and Hogan 1985]. Some models exploit the properties of various functions to reproduce human movements; e.g., exponentials [Plamondon and Lamarche 1986], second order systems [Denier Van Der Gon and Thuring 1965], gaussians [Leclerc et al. 1992], beta functions [Alimi 2003], splines [Morasso et al. 1983], Viviani's curves [Viviani and Flash 1995], and trigonometrical functions [Maarse 1987]. Among these, the kinematic theory [Plamondon 1995a; Plamondon 1995b] and its associated Sigma-Lognormal model [Plamondon and Djioua 2006] provides the most solid framework to date for the study of the production of human movements. This framework takes into account different psychophysiological features, such as the neuromuscular response time, and has been shown to outperform many other approaches [Plamondon et al. 1993]; see Section 8 for a discussion. The Sigma-Lognormal is the latest instantiation of this framework, and very recently has been used to explore gesture recognition, showing that synthesized gestures are beneficial in many ways. For instance, using synthetic samples, a classifier resists better when introducing new classes and it is able to re-estimate rapidly all its parameters and to improve rapidly the recognition performance for the old and the new gestures [Almaksour et al. 2011; Plamondon et al. 2014].

The reader should note that the use of the word "rapid" in the kinematic theory name is due to historical reasons, as the first model (Delta-lognormal) was aimed at studying truly rapid movements, such as those involved in creating handwritten signatures. However, the Sigma-Lognormal model generalizes to any type of movements [Plamondon and Djioua 2006] and has been used to analyze, among others, handwriting learning in children, aging phenomenon in adults, neurodegenerative disorders, or brain stroke risk factors [Plamondon et al. 2013].

3.1. Mathematical Formulation

At a high-level representation, the Sigma-Lognormal model assumes that a complex handwritten trajectory (e.g. a character, a digit, a word, a signature, or a gesture) is composed of a series of primitives[2] (circular arcs) connecting a sequence of virtual targets. This series of primitives is known as the "action plan" of the user, and it is fed through the neuromuscular network to produce a trajectory that leaves a handwritten trace, as illustrated in Figure 1.

Mathematically, the magnitude of the velocity of the ith primitive is described by a lognormal shaped function scaled in amplitude by a command parameter D_i and time-shifted by the time occurrence t_{0_i} of this command:

$$\|\vec{v}_i(t)\| = D_i \Lambda(t; t_{0_i}, \mu_i, \sigma_i^2) = \frac{D_i}{\sigma_i \sqrt{2\pi}(t - t_{0_i})} \exp\left(\frac{-[\ln(t - t_{0_i}) - \mu_i]^2}{2\sigma_i^2}\right) \quad (1)$$

where μ_i and σ_i define the variability of the neuromuscular execution of the ith motor command.

[1] This phenomenon is related to the well-known "Midas Touch problem" in eye-gaze interaction: the sensors cannot be used directly as a pointer device, because the sensors are always "on."

[2] In the gesture recognition literature, the term "stroke" denotes the trajectory between two consecutive pointer-down and pointer-up events. In the kinematic theory, a "stroke" is what we call "primitive" in this article.

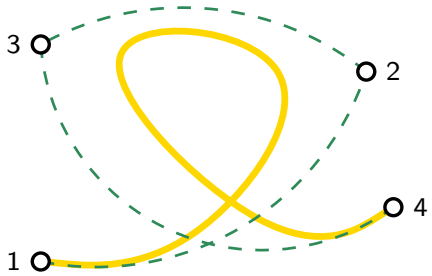


Fig. 1. A gesture stroke (orange thick line) is described by the temporal overlap of a series of primitives (green dashed arcs) connecting a sequence of virtual targets (1–4 black circles). Each primitive is described by a lognormal velocity profile.

Then, the trajectory that produces the human movement $\vec{v}(t)$ is computed as the temporal overlap of each primitive's velocity $\vec{v}_i(t)$:

$$\vec{v}(t) = \sum_{i=1}^{N} \vec{v}_i(t) = \sum_{i=1}^{N} \begin{bmatrix} \cos\phi_i(t) \\ \sin\phi_i(t) \end{bmatrix} D_i \Lambda(t; t_{0_i}, \mu_i, \sigma_i^2) \tag{2}$$

where the angular position $\phi_i(t)$ is obtained by:

$$\phi_i(t) = \theta_{s_i} + \frac{\theta_{e_i} - \theta_{s_i}}{2} \left[1 + \operatorname{erf}\left(\frac{\ln(t - t_{0_i}) - \mu_i}{\sigma_i \sqrt{2}} \right) \right] \tag{3}$$

being θ_{s_i} and θ_{e_i} the starting angle and the end angle of a given primitive, respectively.

Finally, the reconstruction of the original trajectory can be computed using the following compact notation [O'Reilly and Plamondon 2009]:

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \sum_{i=1}^{N} \frac{D_i}{\theta_{e_i} - \theta_{s_i}} \begin{bmatrix} \sin\phi_i(t) & - & \sin\theta_{s_i} \\ -\cos\phi_i(t) & + & \cos\theta_{s_i} \end{bmatrix} \tag{4}$$

Previous works have demonstrated the connection between the distortion of the Sigma-Lognormal parameters and the intra-variability found in complex human movements [Djioua and Plamondon 2009; Martín-Albo et al. 2014]. This results in the generation of realistic human-like synthetic samples, which in turn improves an existing recognizer's accuracy [Plamondon et al. 2014]. We have developed a web service that implements this framework and we would like to make it available to the research community via this article. Further, we have developed an open source web application that interfaces with our web service. To date, this is the first tool of these characteristics that is publicly available. In the next section we provide the implementation details of G3 components. Then we describe the G3 interface through a practical usage example.

4. IMPLEMENTATION AND ALGORITHMS

In this section, we describe the current implementation of our gesture bootstrapping system. At its core, we have incorporated the Sigma-Lognormal extractor proposed by O'Reilly and Plamondon [2009] to identify primitives and compute their Sigma-Lognormal parameters. Then, we built a web service that makes the system available to others as well as a web application that interfaces with the web service.

4.1. Sigma-Lognormal Extractor

In the following, we describe the different steps performed to identify and extract the lognormals from a given gesture example.

4.1.1. Preprocessing. Initially we apply a preprocessing step focused on enhancing the quality of a given gesture trajectory. To begin, the trajectory is interpolated using cubic splines and then resampled at 200 Hz. As the parameter estimator used is velocity-based, instead of shape-based, it is more prone to spatial deviation errors (drifting) as the input size gets larger; i.e., when the number of strokes and/or number of points increase. Then, the final velocity is computed as the sum of each primitive's velocity. Therefore, some errors in the angle estimation could be propagated over the reconstructed trajectory, leading to increased drifting for larger gestures [Fischer et al. 2014]. For this reason, we used a more robust representation, where we split the original trajectory into smaller pieces, considering each of them as an independent primitive. After that, zero crossing velocity is enforced for each primitive, by keeping the original trajectory artificially at the start and end positions.

4.1.2. Primitive Identification. As previously stated, to estimate the Sigma-Lognormal parameters, a number of primitives must be identified in the gesture velocity profile. This results in five characteristic points for each lognormal. Each characteristic point is located at a certain time t and has magnitude $\|\vec{v}(t)\|$; see Figure 2.

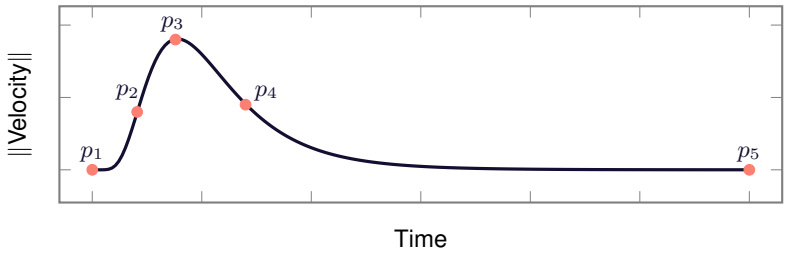


Fig. 2. The velocity profile of a gesture primitive follows a lognormal function. The red dots indicate, from left to right: the beginning of the lognormal (p_1), first inflexion point (p_2), local maximum velocity (p_3), second inflexion point (p_3) and the end of the lognormal (p_5).

It is important to remark that the selection of these characteristic points must be robust, given that noise and/or lognormal superpositions may drift their location, generating thus false point series. Thus, their location is compensated taking into account the expected variability of the μ and σ parameters. After that, two practical criteria are applied to retain the meaningful characteristic point series. The first criterion states that the area under the curve delimited by p_1 and p_5 must be greater than the mean minus one standard deviation of the area under the curve of all computed characteristic point series. The second criterion states that the maximum value of a characteristic point series (that is, $\|\vec{v}(t_3)\|$) must be at least 15 times smaller than the maximum value of $\|\vec{v}(t)\|$.

4.1.3. Primitive Extraction. After primitive identification, a strategy to extract multiple lognormals is adopted, which proceeds in two different modes. In the first one, all characteristic point series are sequentially processed. For each set of characteristic points, Sigma-Lognormal parameters are estimated using the Robust XZERO (RX_0) algorithm [O'Reilly and Plamondon 2009]. This mode is preferred, because it provides a better framework to isolate each lognormal, given that it minimizes the superposition effect from direct neighboring lognormals. However, if the end of the gesture trajectory is reached without having obtained a satisfactory estimation, the extractor toggles to a secondary mode. Here, the characteristic point series are processed in descending order according to their area under the curve, that is, according to the importance of their effect on the gesture velocity.

Finally, the quality of the estimated primitives is measured using the signal to noise ratio (SNR) with respect to the original velocity profile. SNR is computed as follows:

$$\text{SNR} = 10 \log \left(\frac{\int_0^T \|\vec{v}(t)\|^2}{\int_0^T \|\vec{v}(t) - \vec{v}^*(t)\|^2} \right) \tag{5}$$

where $\vec{v}(t)$ is the original velocity, $\vec{v}^*(t)$ is the reconstructed velocity and T is the duration of the gesture. In practice, the Sigma-Lognormal parameters are considered to be well estimated when SNR $\geq$ 15 dB; see Figure 4.

4.1.4. Velocity Estimation. As mentioned previously, the RX_0 algorithm was used to estimate the velocity-related parameters (σ, t_0, D, μ) for every detected lognormal. This algorithm exploits time and velocity constraints on three of the lognormal characteristic points, p_j:[3]

$$\begin{aligned} t_j &= t'_j \\ \|\vec{v}(t_j)\| &= \|\vec{v}^*(t'_j)\| \end{aligned} \tag{6}$$

where $j \in \{2, 3, 4\}$. Left-side terms are observed values, obtained from the velocity profile and the right-side terms are calculated analytically.

The estimation of σ, μ, t_0 and D can be carried out using different combinations of the lognormal characteristic points and the Equations (7) to (10) that are derived from the previous constraints:

$$\sigma^2 = \begin{cases} -2 - 2\ln r_{\alpha\beta} - \dfrac{1}{2\ln r_{\alpha\beta}}, & \text{if } \alpha = 2,\ \beta = 3 \\ -2 + 2\sqrt{1 + \ln^2 r_{\beta\alpha}}, & \text{if } \alpha = 2,\ \beta = 4 \\ -2 - 2\ln r_{\beta\alpha} - \dfrac{1}{2\ln r_{\beta\alpha}}, & \text{if } \alpha = 3,\ \beta = 4 \end{cases} \tag{7}$$

with $r_{ij} = \|\vec{v}(t_i)\| / \|\vec{v}(t_j)\|$. Then,

$$\mu = \ln \left(\frac{t_\alpha - t_\beta}{e^{-a_\alpha} - e^{-a_\beta}} \right) \tag{8}$$

$$t_0 = t_\alpha - e^{\mu - a_\alpha} \tag{9}$$

$$D = \|\vec{v}(t_\alpha)\| \, \sigma \sqrt{2\pi} \exp \left(\mu + \frac{a_\alpha^2}{2\sigma^2} - a_\alpha \right) \tag{10}$$

where $\alpha, \beta \in \{2, 3, 4\}$, $\alpha < \beta$ and

$$a_j = \begin{cases} \frac{3}{2}\sigma^2 + \sigma\sqrt{\frac{\sigma^2}{4} + 1}, & \text{if } j = 2 \\ \sigma^2, & \text{if } j = 3 \\ \frac{3}{2}\sigma^2 - \sigma\sqrt{\frac{\sigma^2}{4} + 1}, & \text{if } j = 4 \end{cases} \tag{11}$$

The parameters are computed using all of the possible combinations of p_2, p_3, and p_4, in order to provide more robustness to such estimation. Once these estimates are available, the best solution is the one that minimizes the least-square error with respect to the original velocity profile.

[3] Although constraints are also met for p_1 and p_5, they are not very robust in practice and therefore we do not recommend to take them into account.

4.1.5. Angle Estimation. At this point, the velocity parameters θ_s and θ_e remain still unknown. Nevertheless, given that each lognormal primitive occurs along a pivot, it can be proved that the angular variation is proportional to the traveled distance along the trajectory [Plamondon 1995a]. This property is exploited to perform a linear interpolation that computes the angular parameters θ_s and θ_e using Equations (3) and (14):

$$\begin{aligned} \theta_s &= \phi(t_3) - \Delta\phi \cdot \big[\mathrm{d}(t_3) - \mathrm{d}(t_1)\big] \\ \theta_e &= \phi(t_3) - \Delta\phi \cdot \big[\mathrm{d}(t_5) - \mathrm{d}(t_3)\big] \end{aligned} \tag{12}$$

where

$$\Delta\phi = \frac{\phi(t_4) - \phi(t_2)}{\mathrm{d}(t_4) - \mathrm{d}(t_2)} \tag{13}$$

$$\mathrm{d}(t_j) = \begin{cases} 0, & \text{if } j = 1 \\ \frac{D}{2}\left[1 + \mathrm{erf}\left(\frac{-a_j}{\sigma\sqrt{2}}\right)\right], & \text{if } j = 2, 3, 4 \\ D, & \text{if } j = 5 \end{cases} \tag{14}$$

4.2. Artificial Sample Generation

In the past, the artificial generation of human movements has been approached by applying deformations at the coordinate level. However, this produces unrealistic results in many cases [Plamondon 1995b; Plamondon et al. 2014]. In contrast, as discussed earlier, previous works have demonstrated the connection between the distortion of the Sigma-Lognormal parameters and the intra-variability found in human handwriting [Djioua and Plamondon 2009; Martín-Albo et al. 2014], which results in the generation of realistic human-like synthetic samples. Therefore, once the gesture primitives have been extracted and a model of the original (human) gesture sample is available, we introduce uniformly-distributed perturbations to the following parameters:

$$\begin{aligned} l_i^* &= l_i\,(1 + \xi \cdot u_{l_i}) \\ g_i^* &= g_i\,(1 + \xi \cdot k) \end{aligned} \tag{15}$$

where $l_i = \{\mu_i, \sigma_i\}$ denote local perturbations in the peripheral parameters μ and σ, with $u_{l_i} = \mathcal{U}(-n_l, n_l)$ being the (local) noise level applied to each primitive; $g_i = \{D, \theta_s, \theta_e\}$ denote global perturbations in the control parameters D and θ, with $k = \mathcal{U}(-n_g, n_g)$ being the constant (global) noise level applied to each primitive; $\xi \in [0, 1]$ is a user-modifiable parameter of the G3 interface (to be described in the next section); and $n_\mu = 0.15$, $n_\sigma = 0.35$, $n_D = 0.25$, $n_\theta = n_{\theta_s} = n_{\theta_e} = 0.3$ denote the different noise values. These values have been empirically tunned in previous work [Martín-Albo et al. 2014]. The remaining Sigma-Lognormal parameter t_{0_i} is left untouched, as suggested by others, because it is very sensitive to small perturbations [Fischer et al. 2014].

4.3. G3 Web Service

Our web service was designed with simplicity and ease of use in mind. Therefore, a single URL is made available as endpoint, accepting: the number of desired synthetic samples (10 by default), the degree of gesture variability (ξ, 1.0 by default) and an example gesture.

For instance, the following HTTP request will return 15 synthetic samples having a relatively high variability degree:

```
POST /synthesize HTTP/1.1
Host: http://g3.service.url
Content-Type: application/json

{ "gesture": "{···}", "num_samples": 15, "variability": 0.75 }
```

where **"gesture"** points to a JSON-formatted string representing a stroke sequence:

```
{ "id1":[[x1,y1,t1]···[xM,yM,tM]], ···, "idN":[···] }
```

Each stroke is a tuple of 2D coordinates plus timestamp, and has an ID in order to make the format compatible with multitouch gestures.

We should mention that the ξ parameter is just meant for fine-tuning while operating the G3 interface: while 1.0 is a reasonable choice (it provides more diverse gesture exemplars), setting it to 0.0 is useful for research (e.g. inspect sample reconstruction, test and compare different noise generation techniques) or user adaptation purposes (replicating the same sample is equivalent to weighing its importance in a dataset). As can be seen in Figure 3, ξ can be controlled by a slider and, according to Equation 15, the noise levels range between 0.0 and their empirically tuned values.

On the other hand, if no timestamps are available, the web service accepts an optional third parameter to set a sampling frequency (200 Hz by default). For instance, assuming that in the previous example the coordinates do not have associated timestamps but we know they were acquired at 100 Hz, we can specify such sampling rate as follows:

```
POST /synthesize HTTP/1.1
Host: http://g3.service.url
Content-Type: application/json

{ "gesture": "{···}", "num_samples": 15, "variability": 0.75, "rate": 100 }
```

The web service returns a JSON-encoded string:

```
HTTP/1.1 200 OK
Connection: close

{ "success": true, "error_code": null, "samples": [gesture01, ···, gesture15] }
```

The **success** property informs about the bootstrapping result: **true** if success, **false** otherwise. Each synthesized gesture sample has the same number of strokes as the original gesture, and is resampled according to the user's articulation speed[4] Finally, JSONP requests are possible, in order to enable cross-domain communication.

4.4. G3 Web Application

Our web service alone might not be of practical use for novice developers or UI designers, as they may not be familiar with machine learning techniques or gesture recognition algorithms. For that reason, we release a web-based application interfacing with our web service that allows users to make the most of its potential. A detailed overview is given in the next section.

The web application incorporates a number of template-matching recognizers (see next section), so that it is possible to build a working prototype together with the synthesized data. Further, being open source, our application allows anyone interested to contribute to improving it. For instance, an envisioned task is that of including a particular gesture recognizer in a particular programming language. To do so, currently a developer must put a the source code of said recognizer in a special folder inside the application's working directory together with a **g3manifest.json** file. The following is an example:

```
{
  "name": "NN-DTW",
  "description": "Nearest Neighbor classifier with Dynamic Time Warping",
  "author": "mlpy",
  "version": "3.4.0",
  "website": "http://example.com/dtw/",
  "language": "Python",
  "capabilities": ["unistrokes"],
  "preprocessor": "/path/to/file"
}
```

[4]Estimated either from the timestamps or the optional **rate** parameter.

Here, the interesting part is the file indicated in the **`preprocessor`** property. This file takes as input a gesture set in JSON format and outputs a working gesture recognizer in a ZIP file. The preprocessor instructs our tool how to build the recognizer, e.g. feeding a template library with the new samples provided or even training a sophisticated classifier. Thus, our tool abstracts away the recognizer's implementation. The **`capabilities`** option provides additional information about the recognizer; e.g. whether it can deal with multistroke gestures or if it only targets unistroke gestures. This is used to inform the user prior selecting a combination of recognizer plus programming language (see Figure 3).

5. INTERACTING WITH G3

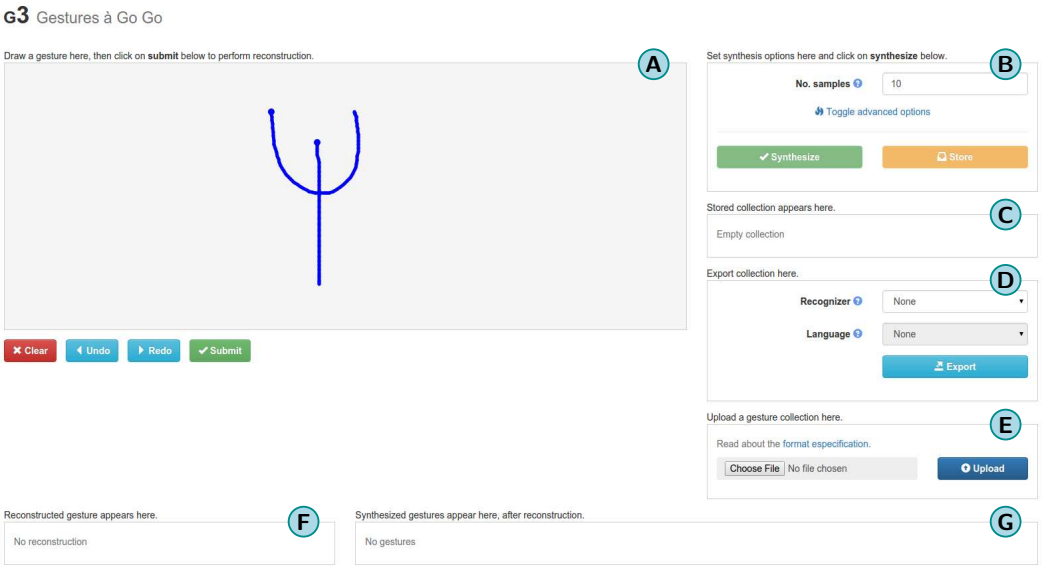


Fig. 3. G3 user interface. **A**: Drawing area. The dot indicates the starting point of each gesture stroke. **B**: Options area. Besides the number of gestures to generate, advanced options allow the user to indicate e.g. a desired variability degree. **C**: Collection area. Each gesture is presented as an ordered list, with the possibility of adding or removing gesture examples. **D**: Export area. The user can optionally export a gesture recognizer available in different programming languages. **E**: Import area. A JSON file comprising a collection of gesture examples can be submitted. **F**: Reconstruction area. A reconstruction of the user gesture is used for later synthesis. **G**: Synthetic gestures area. Generated samples appear here.

Like prior example-based systems, our web application allows developers to quickly create a gesture set. However, within G3 interface the developer simply demonstrates one example of each gesture. The web application currently supports unistroke, multistroke, and multitouch gestures drawn on a 2D canvas. Below we introduce our tool by following a developer as she creates a set of customized gestures. The developer, Alice, needs to build a recognizer that handles 5 different gestures in her application.

5.1. Example-based Demonstration of Gestures

For each gesture, Alice records just one example by drawing on the web canvas. We are not aware of any other tool that requires such a small user intervention at this time. Next, G3 verifies that the gesture data is of enough quality for later synthesis. It is considered so when SNR $\geq$ 15 dB (this applies to all of the strokes in a multistroke or multitouch gesture). When the SNR of the provided gesture example is below that threshold, G3 informs Alice so that she can draw the gesture again.

The reasons why a gesture might not achieve a high-quality SNR include: too fast-paced execution (low number of captured points),[5] under-resourced hardware (e.g. an old smartphone), or simply a symptom of possible problems in the user's motor control system [Plamondon et al. 2013].

5.2. Synthesizing Additional Examples

Once a gesture has been drawn, Alice chooses the desired number of samples that will be synthesized as well as the degree of variability for such synthetic samples. Low variability values will return synthetic samples that are mostly close to the gesture drawn by the user, whereas high variability values will make room for generating more diverse samples. Next, she clicks on the 'submit' button and G3 displays a list with the synthetic gesture samples.

5.3. Iterative Refinement

Alice does not like 2 of the synthesized gestures, so she clicks on the unwanted gestures to remove them from the current set and requests 2 additional samples. When she is happy with the result, she clicks on the 'store' button and the samples of the current gesture are saved. The original hand-made gesture is also stored together with the synthetic samples.

5.4. Exporting Gesture Data

After gesture synthesis, Alice immediately has a gesture set of an arbitrary size (e.g., 20 samples for each of the 5 gestures provided, 100 samples overall). By clicking on the 'export' button, the data are exported as a JSON file. G3 will remember the generated gesture set, in case Alice wants to modify some of the gestures later.

Alice also decides to try a number of the recognizers provided by the G3 interface, so she selects the desired combination of recognizer plus programming language and clicks on the 'export' button. As a result, Alice gets a recognizer together with the synthesized gesture set. G3 currently features the "$ family" and Dynamic Time Warping in different programming languages (JavaScript, ActionScript, Python, C#, Java, C++, PHP), which are well-suited for experimentation by developers and UI designers. Excepting JavaScript, not all combinations of recognizer plus programming language are available at this time, but it is our hope that these will be available in the future.

5.5. Incorporating a Recognizer

In case a recognizer is created within G3, incorporating it in the developer's application is analogous to prior works [Amini and Li 2013; Lü and Li 2012; Lü et al. 2014]. Alice will add the recognition module containing the implemented algorithms and the gesture files as created and exported from G3. We have decided not to incorporate recognizers that require extensive training because such approaches do not fit our desire of quickly providing users with a simple and ready-to-use gesture recognizer.

5.6. Importing Gesture Data

Alice remembers that some time ago she downloaded a small dataset consisting of 2 examples per gesture, 16 different gestures in total. Now she wants to generate a bigger dataset to train a custom recognizer, so she prepares a JSON file according to the following specification:

```
{
  "a gesture label": [gesture01, ···, gestureN],
  ···,
  "another gesture label": [gesture01, ···, gestureN]
}
```

where each gesture follows the format described in Section 4.3. She then clicks on the 'upload' button and the collection module gets updated (Figure 3, D). Next, she goes through each imported gesture and requests the desired number of samples. As previously commented, if some of the uploaded

[5] As discussed in Section 4.1.2, at least 5 characteristic points per stroke are needed for reconstruction.

samples were reconstructed with SNR < 15 dB, G3 interface will inform Alice. Finally, she clicks on the 'export' button and she gets delivered the synthesized dataset, and optionally an accompanying gesture recognizer.

6. EVALUATION

We conducted a rigorous experimentation over 3 public datasets, in order to illustrate the value of our bootstrapping service as a means to building or replicating human-generated datasets. We compared the performance of the synthetic gesture samples with that of human samples in terms of: articulation speed, size of gesture vocabulary, input device, and gesture variability. Further, in the next section we provide anecdotal evidence on the usage of G3 interface.

6.1. Recognizers

We tested all recognizers currently implemented in G3 interface: the \$ family (\$1, \$N, \$P) and Dynamic Time Warping (DTW). \$1 (actually Protractor) is an instance-based unistrokes recognizer using 1 nearest-neighbor classification with Euclidean distance; see Related Work. \$N (actually \$N-Protractor) is an extension of \$1 to handle multistrokes. \$P uses an optimization of the Hungarian algorithm for classification. DTW performs elastic matching between two gestures, by computing a warping matrix of point-wise Euclidean distances. Contrary to the recognizers in the \$ family, our DTW implementation does not perform any preprocessing on gesture samples (e.g., resampling, rotating, or scaling). All of these recognizers are publicly available in JavaScript, so we used node.js to conduct the experiments. We used the default configuration of each recognizer excepting gesture resampling, which was set to 32 points in order to speed up recognition time without sacrificing accuracy [Vatavu 2011].

6.2. Datasets

We generated synthetic samples for the following public datasets via our web service.

\$1-GDS: Available at `https://depts.washington.edu/aimgroup/proj/dollar/xml.zip`. This is a reference dataset in HCI to test unistroke-based recognizers. The dataset comprises 16 unistroke gesture classes, 5,280 samples in total. 10 users (plus 1 pilot user) provided 10 samples per class at 3 articulation speeds (slow, medium, fast) using an iPAQ Pocket PC (stylus as input device). For slow speed, users were asked to "be as accurate as possible;" for medium speed, users were asked to "balance speed and accuracy;" for fast speed, users were asked to "go as fast as you can" [Wobbrock et al. 2007].

MMG: Available at `https://depts.washington.edu/aimgroup/proj/dollar/mmg.zip`. This is a reference dataset in HCI to test multistroke-based recognizers. The dataset comprises 16 multistroke gesture classes, 9,600 samples in total. 20 users provided 10 samples per class at 3 articulation speeds (slow, medium, fast) using either finger (half of the users) or stylus as input devices on Tablet PCs. Speed definitions are the same as in the \$1-GDS dataset.

chars74k: Available at `http://www.ee.surrey.ac.uk/CVSSP/demos/chars74k/EnglishHnd.tgz`. The dataset comprises 62 handwritten classes (0-9, A-Z, a-z), unistrokes and multistrokes, 3,410 samples in total. 55 users provided 1 sample per class using a Tablet PC at a constant sampling rate of 100 Hz. This dataset is interesting to us for three reasons. First, previous works have found that letters are commonly used by end-users as gestures [Ouyang and Li 2012; Poppinga et al. 2014]. Second, this dataset is more complex and has a large number of classes in comparison to the other datasets. Third, unlike \$1-GDS and MMG datasets, no timestamps are available in the chars74k dataset, so handwriting velocity must be estimated for reconstruction from the given sampling rate.

6.3. Method

All gesture samples in each dataset were reconstructed according to Section 4.1. We picked the worst and best sample of each gesture in terms of SNR (Equation 5), and generated as many synthetic samples as human samples were in each dataset, using $\xi = 1.0$. As shown in Figure 4, SNR ≥ 15 dB is an acceptable lower bound to work with. Thus, each gesture's worst reconstructed sample had

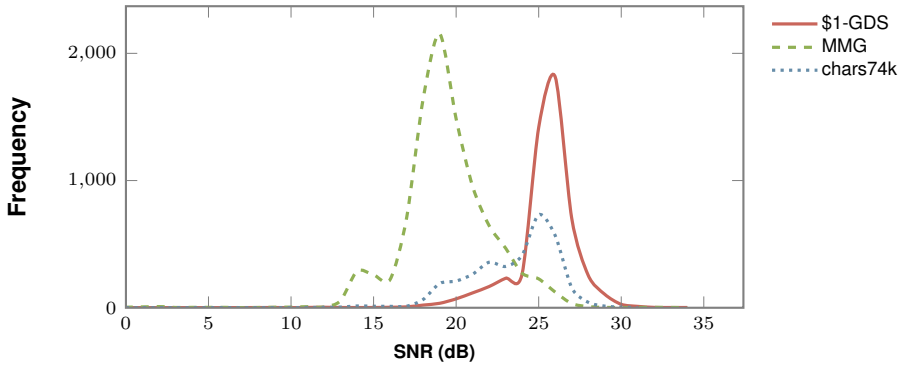


Fig. 4. Histograms of all reconstructed human gesture samples.

at least 15 dB, whereas each gesture's best reconstructed sample had the highest SNR among all available samples of such gesture, typically around 30 dB. Samples with SNR below 15 dB were not considered as worst case examples. In total, 36,580 samples were used for reconstruction.

Figure 5 provides an overview of the $1-GDS and MMG datasets, together with some examples of the synthesized gestures that G3 generates. The chars74k dataset is not shown for the sake of brevity—all samples are just digits and uppercase and lowercase English letters.

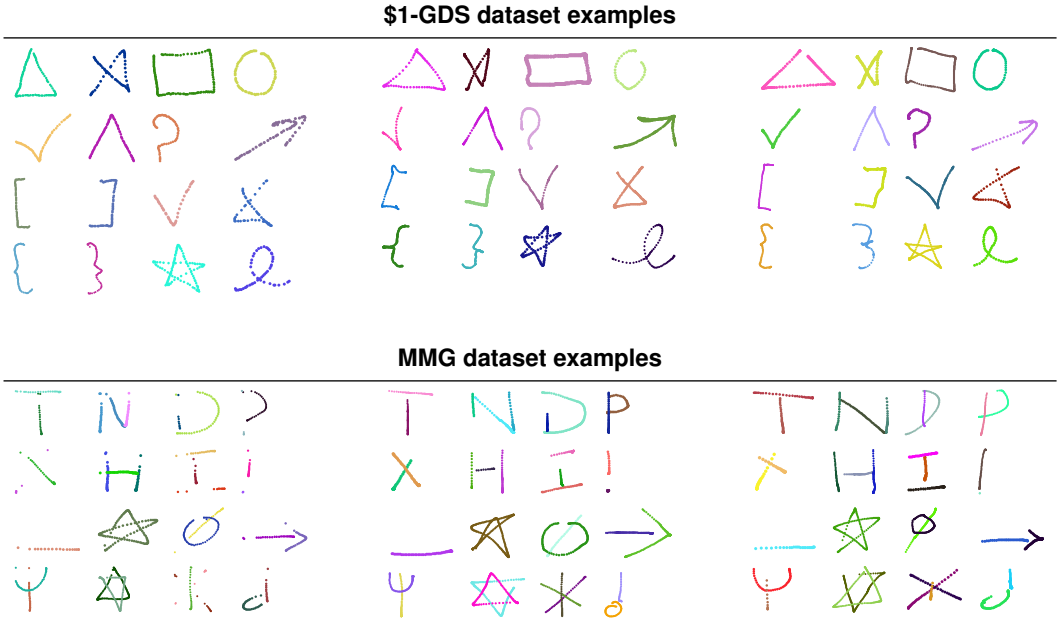


Fig. 5. Human and synthetic gesture datasets, all samples picked at random. Can you guess which datasets are human generated and which ones were synthesized? See the answer at the end of this article.

6.4. Impact of Synthetic Samples on Articulation Speed

These experiments were conducted over the $1-GDS and MMG datasets, as they were the ones that provided up to 3 articulation speeds: slow, medium, and fast. We decided to test the $1-GDS dataset

with $1 and DTW, whereas MMG dataset would be tested with $N and $P. Multistroke recognizers could also have been tested on the $1-GDS dataset, and even DTW could have been adapted to handle multistroke gestures. However, the evaluation goal is comparing the performance of synthetic gesture samples to that of human samples. Thus, we sought a balance on recognizer plus dataset combinations.

6.4.1. User-dependent Tests. To begin, we conducted a number of user-dependent tests in the same vein as previous works have tested the $ family of recognizers [Anthony and Wobbrock 2012; Li 2010; Vatavu et al. 2012a; Wobbrock et al. 2007]. For each user, the recognizer is trained using a number of the user's gesture examples, and one example is used for testing. This is repeated for all users, and results are aggregated. Each user provided 10 examples of each gesture, so we increased the number of templates from 1 to 9. In total, 265,848 recognition tests were performed. The results are shown in Figure 6.

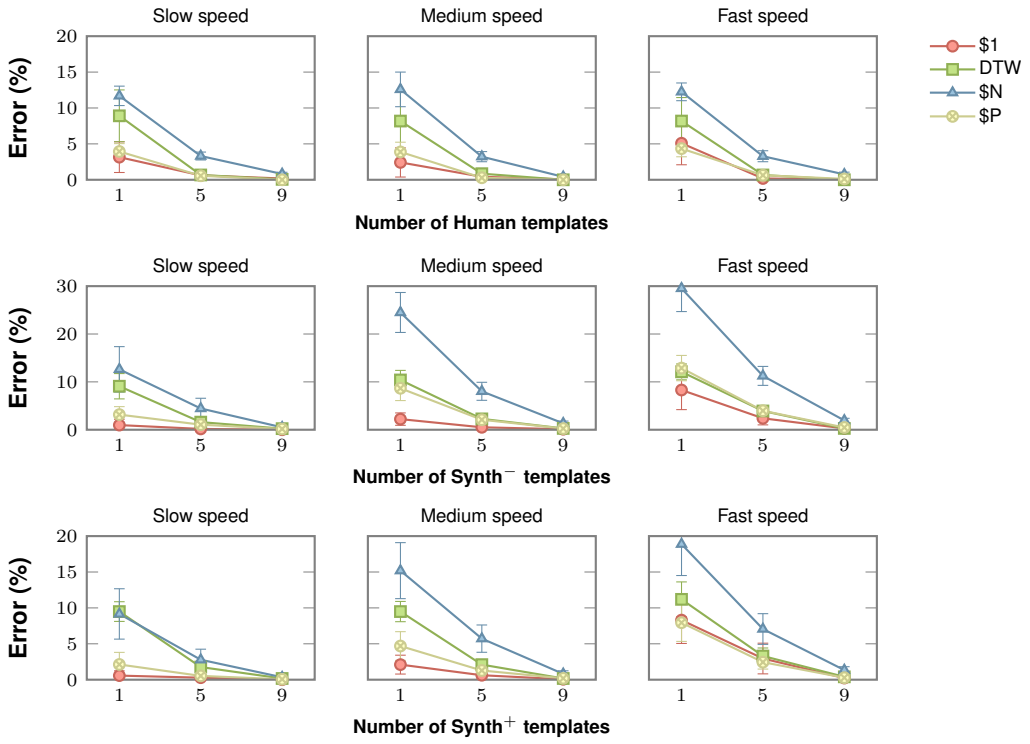


Fig. 6. Impact of synthetic samples on articulation speed. User-dependent tests. Error bars denote 95% confidence intervals. Synth$^-$ and Synth$^+$ denote synthesized samples using the worst and best reconstructed human sample of each gesture, respectively.

Synthetic samples were found to achieve very similar performance to that of human samples. This observation was consistent for all articulation speeds and number of templates, using either the best and worst reconstructed human samples. Differences between human and synthetic samples (either worst and best case examples) were not statistically significant (two-tailed paired t-tests with Bonferroni correction, $p > 0.05/6$). It is interesting to note that $1 and $P perform quite well with just one loaded template; these recognizers are about as twice accurate as DTW and $N, respectively. Then, when the number of templates increases, differences fall away. With all user's templates loaded, all recognizers are +99% accurate.

6.4.2. User-independent Tests. We also conducted user-independent tests, as it is a regular procedure in machine learning experiments. We used a k-fold leaving-one-out procedure: for each user, the recognizer is trained using the rest of the users and one user is left out for testing. This is repeated for all users, and results are aggregated. Each user provided 10 examples of each gesture, so we increased the number of templates from 1 to 10. In total, 4,479,420 recognition tests were performed. The results are shown in Figure 7.

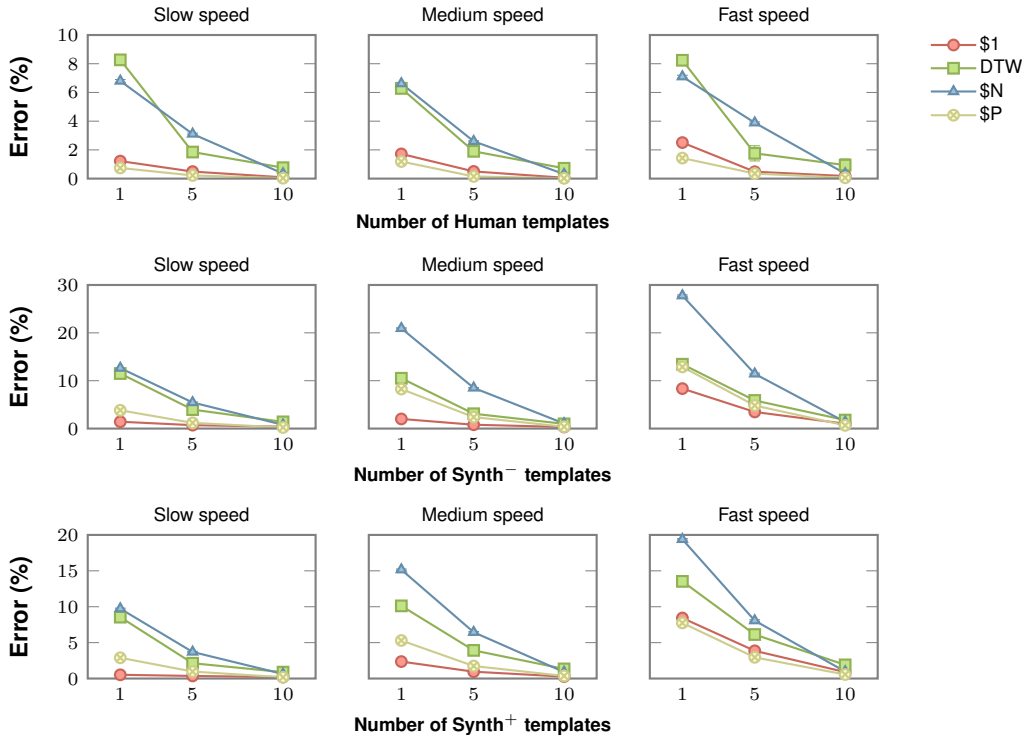


Fig. 7. Impact of synthetic samples on articulation speed. User-independent tests. 95% confidence intervals are below 0.1%. Synth$^-$ and Synth$^+$ denote synthesized samples using the worst and best reconstructed human sample of each gesture, respectively.

As in the previous experiments, we noticed that synthetic samples provide similar performance to that of human samples. This was also consistent for all articulation speeds and number of templates, using either the best and worst reconstructed human samples. Differences between human and synthetic samples were not found to be statistically significant (two-tailed paired t-tests with Bonferroni correction, $p > 0.05/6$). Again, \$1 and \$P performed better than DTW and \$N with just one loaded template. With all user's templates loaded, all recognizers provide similarly competitive advantage.

6.5. Experiments on a Large Gesture Vocabulary

Usually, when the number of gesture classes in a dataset increases, the accuracy of a recognizer tends to decrease. This is so because of potential collisions introduced by perceptually similar classes. To test this hypothesis, we used the chars74k dataset. Some examples of such potential collisions occur in this dataset with `i` and `j`; `o`, `O`, and `0`; `c` and `C`, etc.

In the chars74k dataset 55 users provided 1 sample per gesture class, so we can follow a procedure akin user-dependent tests; see previous section. Given that this dataset includes multistroke samples,

we tested $N and $P. In total, 613,800 recognition tests were performed. The results are shown in Figure 8.

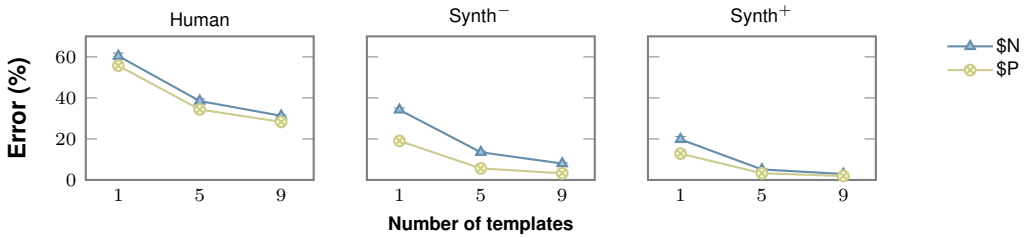


Fig. 8. Impact of synthetic samples on a large gesture vocabulary (chars74k dataset, 62 classes). 95% confidence intervals are below 1%. Synth$^-$ and Synth$^+$ denote synthesized samples using the worst and best reconstructed human sample of each gesture, respectively.

While differences between human and synthetic samples were not statistically significant (two-tailed paired t-tests with Bonferroni correction, $p > 0.05/2$), the recognizers performed worse in comparison to the results they achieved on the MMG dataset, as predicted; see Figure 6. Overall, it was found that synthetic samples achieved better results. This is true for samples synthesized from the best reconstructed samples as well as for samples synthesized from the worst reconstructed samples. For instance, with one loaded template, error rates surpassed 50% in case of human samples, while $N achieved 34.1% and 19.7% for the worst and best cases, whereas $P achieved 19.0% and 12.8%, respectively.

As usual, increasing the number of templates improved recognition accuracy. However, this time the best improvements were achieved by far by the synthetic samples. Both $N and $P stabilized around 30% error using up to 9 human samples as gesture templates, while achieving competitive results with synthetic samples, the error ranging between 8.0% ($N, worst case) and 1.9% ($P, best case). These results are encouraging and of potential interest for the design of gesture sets that use a large number of classes as part of their gesture vocabulary.

6.6. Impact of Synthetic Samples on Input Device

We wondered if there was any difference when users draw gestures with different input devices. Luckily, the MMG dataset allows us to test two conditions: finger and stylus. We conducted both user-dependent and user-independent tests. The 3 articulation speeds were averaged for these experiments.

6.6.1. User-dependent Tests. We followed the same procedure as in the previous experiments. In total, 172,788 recognition tests were performed. The results are shown in Figure 9.

This time, human samples performed better that their synthetic counterparts for 1 loaded template. This was found to be statistically significant for finger (two-tailed paired t-tests with Bonferroni correction, $p < 0.05/4$) but not for stylus with best case examples. Then, as soon as the number of templates increased, all conditions performed equally and statistically similar. With 9 templates loaded, all recognizers achieved around 99% of accuracy for both devices, using either human or synthetic samples.

6.6.2. User-independent Tests. We followed the same procedure as in the previous experiments. In total, 3,455,760 recognition tests were performed. The results are shown in Figure 10.

We observed the same pattern as in the previous user-dependent experiments. With 1 loaded template, human samples achieved better accuracy. This was found to be statistically significant for both devices (two-tailed paired t-tests with Bonferroni correction, $p < 0.05/4$). With 5 templates per gesture, the best case samples performed equally similar to the original dataset samples. Then, with 9 templates loaded, all conditions performed equally and statistically similar. Further, all recognizers

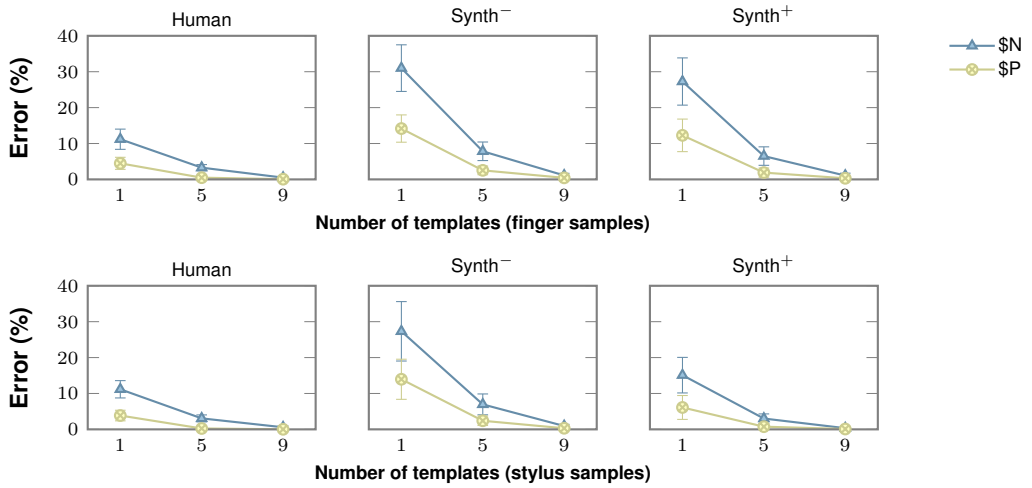


Fig. 9. Impact of synthetic samples on input device. User-dependent tests. Error bars denote 95% confidence intervals. Synth$^-$ and Synth$^+$ denote synthesized samples using the worst and best reconstructed human sample of each gesture, respectively.

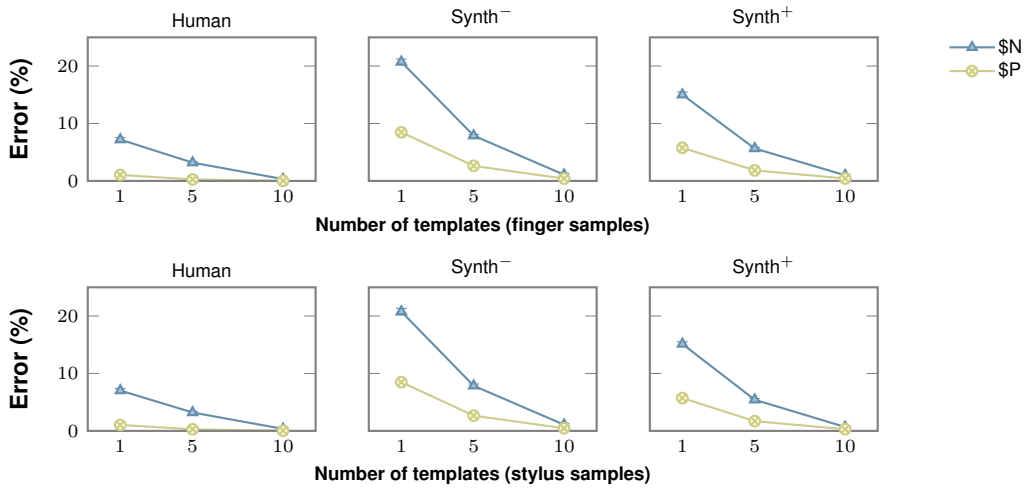


Fig. 10. Impact of synthetic samples on input device. User-independent tests. 95% confidence intervals are below 1%. Synth$^-$ and Synth$^+$ denote synthesized samples using the worst and best reconstructed human sample of each gesture, respectively.

achieved more than 99% of accuracy for both devices, using 10 samples either of human or synthetic nature.

6.7. Impact of Synthetic Samples on Gesture Variability

Usually, a gesture recognizer would not perform well on unseen data if gesture samples do not exhibit sufficient variation. We investigated this topic with our datasets, by comparing the variability of the synthesized samples against their original human samples, using different values of the user-configurable parameter ξ that is available in G3 interface.

Samples were synthesized in batches of $N \in \{10, 100, 1000\}$ samples each with $\xi \in \{0.0, 0.5, 1.0\}$; 0.0 denoting no variability.[6] Then, we computed the mean squared error (MSE) of each synthetic sample with respect to the human sample from which it was generated. MSE values close to zero indicate that synthetic samples and their human counterparts look very similar. Conversely, higher MSE values indicate that synthetic samples diverge in shape from their human counterparts. Strokes were resampled in such a way that a human sample and its synthesized samples had the same length. MSE was averaged for each batch, variability level, and dataset. No difference was found regarding the use of worst and best case examples. Table I shows the results.

Table I. Gesture variability (mean squared error) for different number of synthesized samples and different values of the user-configurable parameter ξ.

N	ξ	$1-GDS			MMG			chars74k		
		Mean	SD	SE	Mean	SD	SE	Mean	SD	SE
10	0.0	554.9	715.9	56.6	169.7	175.7	9.5	75.2	101.3	3.6
	0.5	579.4	774.4	61.2	250.1	271.7	14.7	359.8	407.6	14.7
	1.0	593.6	731.1	57.8	490.4	704.9	38.2	1181.5	1449.3	52.5
100	0.0	554.9	713.9	17.8	169.7	175.7	3.0	75.2	101.3	1.1
	0.5	576.8	754.7	18.8	256.1	273.4	4.6	377.2	408.3	4.6
	1.0	622.1	823.1	20.5	493.4	628.2	10.7	1298.9	1448.9	16.6
1000	0.0	554.9	713.6	5.6	169.7	175.7	0.9	75.2	101.3	0.3
	0.5	572.5	741.3	5.8	261.8	280.3	1.5	386.4	426.8	1.5
	1.0	621.4	813.3	6.4	498.7	646.0	3.5	1316.2	1452.7	5.2

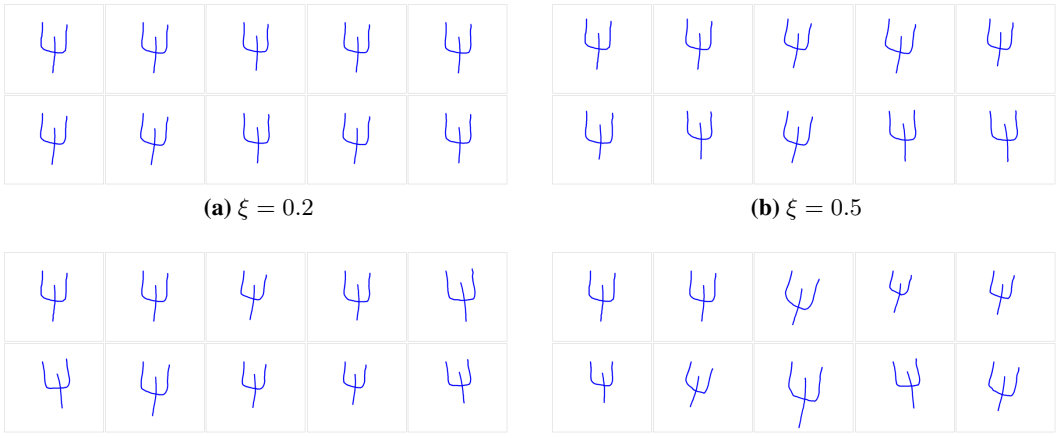
As expected, it was found that synthetic samples are more variable as ξ increases. Figure 11 illustrates the visual effect of this parameter on gesture variability. In general, we observed that requesting a small number of synthetic samples (10 samples per gesture) provides slightly less variable samples. Interestingly, for a given value of ξ, variability was found to increase as the number of requested synthetic samples increases, though we suspect it is because the MSE is underestimated for small batch sizes. Indeed, the standard error (SE) gets smaller as the number of samples gets larger, because the mean of a large sample is likely to be closer to the true population mean.

We also examined the intra-class variability of human gestures, distance-wise; i.e., how variable is a human gesture sample as compared to the rest of the human samples that belong to the same gesture class. No difference was found regarding the use of worst and best case examples, neither regarding the number of requested synthetic samples. The Pearson's correlation coefficient was found to be greater than 0.94 in *all* datasets, which indicates, for the human datasets, a large agreement regarding how users articulated gestures. Then, comparing synthetic gestures with their human counterparts resulted in Pearson's correlation coefficients decreasing as ξ increased; see Table II. This was unsurprising, and indicates that samples synthesized with a low variability degree look much more similar to the human samples from which they were generated.

Table II. Correlation for different values of the user-configurable parameter ξ. Batch size did not make any difference in this study.

Dataset	Pearson's ρ		
	$\xi = 0.0$	$\xi = 0.5$	$\xi = 1.0$
$1-GDS	0.94	0.94	0.93
MMG	0.94	0.93	0.89
chars74k	1.00	0.99	0.97

[6] In this case, G3 returns N copies of the same reconstruction of each human gesture sample.



(a) $\xi = 0.2$ **(b)** $\xi = 0.5$ **(c)** $\xi = 0.7$ **(d)** $\xi = 1.0$

Fig. 11. Visualizing the effect of the ξ parameter on gesture variability while requesting 8 synthetic samples. In these subfigures, the first sample (top-left) is the original user's gesture, whereas the second sample (top second to the left) is the reconstruction of the original gesture. With $\xi = 0.0$, G3 simply returns 8 copies of the reconstructed user's gesture.

7. INFORMAL USER TESTS

We informally evaluated our web application with 2 participants. Both were professional developers but have never programmed gestures. To conduct the tests, we used a Nexus 4 smartphone running Android 4.4.4 (KitKat). The smartphone's touchscreen was placed in landscape mode. Participants browsed the web application using Firefox mobile.

Participants were asked to recreate a mobile shortcut gestures dataset (Figure 12) that is not publicly available at the moment [Poppinga et al. 2014, personal communication]. The dataset comprises 20 unistroke gesture classes for the top-20 most used commands on smartphones, and was compiled with the Gestify app, available at the Android Market. In the original (in-lab) study, 18 users provided 1 sample of each gesture [Poppinga et al. 2014].

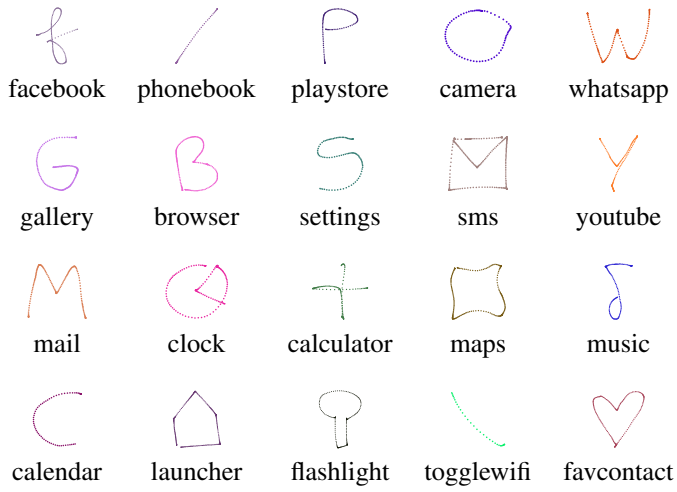


Fig. 12. Mobile shortcut gestures dataset. All samples were synthesized by picking at random one human sample of each gesture.

Participants were given a brief description of the G3 interface, after which they could play with it for 5 minutes at most. Then, they were given a paper-based cheatsheet with the gestures they had to draw. Participants were told to use the default options: request 10 synthesized samples, variability slider at the middle position, no built-in recognizer. This way the data would be collected under the same settings. Participants were instructed to perform as fast and accurate as possible, so they operate at their own speed-accuracy tradeoff point.

Eventually, participants spent 10.3 and 12.1 minutes each while creating their datasets. Each dataset comprised 220 gesture samples (10 synthesized samples per gesture plus the originally drawn sample), so we eventually collected 440 samples for analysis. On average, each gesture took 2.6 seconds to draw for the user and 9.4 seconds to synthesize for the system. 18 out of the 20 gestures were successfully drawn at first try; i.e., participants were happy with how they executed each gesture. The Maps and Clock gestures were the ones that took a few tries to draw, though they were reconstructed with high-quality SNR (M=24.8, SD=2.0). This reveals that our web application achieves adequate sampling performance for data gathering on a mobile device. In addition, no synthesized sample was re-generated within the interface, indicating that participants were happy with G3 results.

Participants commented that G3 interface was simple and intuitive to use, and expressed a desire to use it in case they would need to create a gesture-driven application. All in all, enabling a developer who has never programmed gestures to generate a high-quality dataset (and optionally a gesture recognizer) in about ten minutes is promising and of potential interest both for practitioners and researchers.

No recognition tests were performed by the original dataset authors [Poppinga et al. 2014], who shipped Gestify with $1 and assumed that it would achieve 97% accuracy with one loaded template. Therefore we also wondered how $1 would actually perform on this dataset, and whether another template-matching recognizer could perform better. To achieve this, we ran user-dependent tests with the collected data, using all recognizers currently implemented in G3 interface. As in previous experiments, we increased the number of training templates from 1 to 9, and left 1 for testing. Templates were picked at random, so we repeated this procedure up to 10 times to reduce the effects of chance, since we had a relatively small number of samples. In total, 50,400 recognition tests were performed. The results are shown in Figure 13.

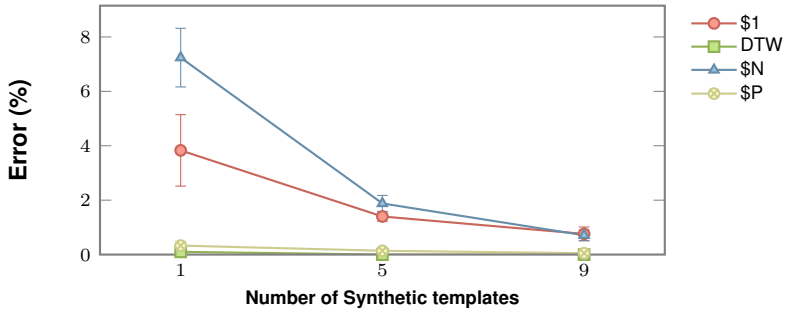


Fig. 13. Results on the mobile shortcut gestures dataset. Error bars denote 95% confidence intervals.

We verified that $1 can indeed achieve 97% of accuracy with just one loaded template. However, under the same conditions DTW and $P achieved 99.9% and 99.6% accuracy rates, respectively. Then, when the number of templates increases all recognizers provide similar competitive accuracy, as expected. Therefore, we would recommend the Gestify application to ship either $P or DTW in future releases, as they will perform better with less number of loaded templates. Further, both recognizers will allow users to create multistroke gestures.

8. GENERAL DISCUSSION

This article focuses on a web service to bootstrap gesture generation and the web application that makes the service available to others. Our experiments have proved impact on the design of gesture datasets, allowing the user to select the best recognizer for each use case. This way, it is really easy to quickly collect gesture samples and build a competitive recognizer for gesture-driven UI prototypes.

The fundamental advantage of G3 over other approaches is that the user just needs to draw one example of each gesture. Then, samples are synthesized using lognormal-based deformations on velocity profiles, which produces human-like results and ultimately helps a recognizer to perform well on unseen data. In fact, our results confirm previous works that have reported improved accuracy when training with a dataset that is extended with synthetic data [Galbally et al. 2012; Fischer et al. 2014; Martín-Albo et al. 2014; Plamondon et al. 2014]; e.g., from words or signatures collected from various writers sitting in front of a digitizer tablet, to sentences written on a whiteboard using full arm movements while standing up.

From a fundamental perspective, it has been proved that lognormal-based models are the most accurate descriptors of human movements and that other models can be considered as successive approximations [Djioua and Plamondon 2009]. In fact, Plamondon et al. [1993] compared 23 different models to describe human movements and found that the lognormal approach outperfomed all of the other approaches. And even though these lognormal-based models have been mainly applied to handwriting analysis, many studies have shown that they can be successfully applied to other types of movements. For example: reproducing wrist movement and eye saccades [Plamondon 1995b], 2D and 3D arm movements [Leduc and Plamondon 2001], and more recently, stroke gestures [Almaksour et al. 2011]. In other words, while in practice there might be inherent differences between handwriting and stroke gesture input, it has been shown that Sigma-Lognormal synthesized gestures are actually reflective of how users produce stroke gestures.

The previous instantiation of the Kinematic Theory (the Delta-Lognormal model) assumed that the production of a stroke requires the synergetic activation of two neuromuscular systems, one agonist and the other antagonist to the direction of the movement. These synchronous commands propagate in parallel across the two neuromuscular systems, each of which is described by a lognormal impulse response and has its own timing properties. On the contrary, the Sigma-Lognormal model does not assume that the two neuromuscular systems are working in precisely opposite directions. The output velocity is thus described by a vectorial summation of the contribution of each neuromuscular system involved in the production of a stroke. This model is actually very general, and is not limited to a single stroke description [Plamondon and Djioua 2006; O'Reilly and Plamondon 2009]. Our work builds upon this fundamental notion, however we are the first to use the Sigma-Lognormal model to study finger writing behavior. This corroborates the prediction of the Kinematic Theory, where it is theorized that every human movement has a lognormal impulse response that results from the limiting behavior of a large number of interdependent neuromuscular networks.

According to the Kinematic Theory, the actual variability in handwriting articulation might come from two sources: the "action plan" of the user (dashed lines in Figure 1) and the actual execution process. This is reflected by fluctuations in the control parameters (t_0, D, θ) and in the peripheral parameters (μ, σ). In our implementation, we introduce noise in μ, σ, D and θ. The μ and σ variations mimic peripheral noise, like a writer who instantiates the same gesture intention and executes it with an upper limb slightly different from one trial to another. The D and θ variations refer to central fluctuations that might occur in the position of the virtual targets of the action plan from one trial to another, reflecting for example attention changes. Combining both types of variations, the central and peripheral noise values being empirically tuned, reflects real-life situations like performing the same movement under different psychophysiological conditions.

With G3, the user can request more or less diverse samples, according to the ξ parameter. However, we should mention that such parameter is just for fine-tuning. This is so because it has been shown that generally people tend to perform gestures consistently and it is hard to manually introduce variation [Lü et al. 2014]. That said, we believe that practitioners would use $\xi = 1.0$ most of the time,

as it provides more diverse gesture exemplars, whereas researchers might want to experiment with other values.

In most interactive applications, a gesture is generally submitted as a stream of asynchronous timestamped events. Interestingly, because G3 has access to a model of the original human gesture, it can return reconstructed samples in a variety of ways, e.g., unevenly or uniformly distributed coordinates either in space or time. Right now G3 transforms the original stream into constant frequency; i.e., the resulting synthesized coordinates are uniformly distributed in time. This allows to "fix" downsampled strokes that were acquired with under-resourced hardware; see e.g. the last 2 examples of human-made gestures in the MMG dataset (Figure 5). Ultimately, this approach may have a significant impact on the design of template-based gesture recognizers, since some preprocessing steps prior to recognition (e.g. resampling) could be omitted.

We should point out that a synthesized gesture has the same number of strokes as the original gesture after reconstruction, because G3 uses only 1 example for bootstrapping, aimed at unburdening the user. While this is not a limitation for synthesizing unistroke gestures (for obvious reasons), one might want synthesized multistroke gestures to have a different number of strokes than their human counterparts. If so, the user can simply provide another example of the same gesture executed with a different number of strokes.

An actual limitation of our current implementation is related to temporal performance. While G3 actually does not mind generating either ten or thousand gesture samples from one given example, the payload resides in the reconstruction process, which is proportional to the number of points (stroke-wise) and the number of strokes. For instance, generating 1,000 samples of each gesture in the $1-GDS dataset took on average 68.7% of the processing time (SD=13.1%) in reconstructing each human sample. The same procedure took a bit more time in the MMG dataset (M=82.1%, SD=16.5%) since there are multi-stroke samples. Results with the chars74k dataset stayed in between (M=76.1% SD=15.5%) since there are less multi-stroke samples than unistrokes.

In addition to the experiments conducted in this article with template-matching recognizers, which typically achieve competitive accuracy with few training examples per gesture class, previous works have also proved the suitability of using synthetic samples generated by the Sigma-Lognormal model in recognizers that require a large number of training samples. For example, by doubling the training set in this way, Fischer et al. [2014] reported an increase in accuracy while recognizing handwritten texts with a neural network. Similarly, by generating thousands of these synthetic samples, Martín-Albo et al. [2014] have successfully performed writer adaptation with hidden Markov models.

Finally, we should mention that Plamondon et al. have conducted several studies regarding human perception toward synthetic samples; e.g. Galbally et al. [2012] showed that users cannot tell real and synthetic signatures apart. Our evaluation stands out from previous studies by providing the most comprehensive analysis (performance-wise) to date on stroke gestures bootstrapping. It is our hope that G3 will allow UI researchers and practitioners to easily incorporate gestures in their prototypes. Looking forward, we believe our work suggests future research opportunities for researchers, practitioners, and developers that wish to design better user interfaces that are driven by gestures.

9. CONCLUSION AND FUTURE WORK

We have presented G3, a tool for bootstrapping human-like stroke gesture samples based on the kinematic theory of rapid movements and its associated Sigma-Lognormal model. Within an intuitive web application, our tool offers active design exploration by prototyping gesture datasets. As in other design tools, it enhances example-based learning preserving the low threshold of example-based gesture tools while raising the ceiling of the recognizers created in such tools [Myers 1988].

Although not the focus of this article, we should point out that the Sigma-Lognormal model can be used directly as the basis for a recognition algorithm. In this article, we have used the model as a generative application, but it could be used e.g. for logistic regression in a wealth of domains. For example, others have used the model to make predictions about user's age and detect brain stroke

risk [Plamondon et al. 2013]. There are thus many interesting research avenues worth pursuing in future work.

At the moment, G3 can only process 2D trajectories. Thus, in future work we would like to extend it for processing 3D gestures, derived from e.g. accelerometers, Kinects, Wii controllers, and similar devices. Actually, this kind of data can be reconstructed using lognormals provided that the torsion is taken into account [Leduc and Plamondon 2001], which just requires introducing an additional parameter to the Sigma-Lognormal model. Also, to make this possible in practice, we would need to rewrite some parts of our web application, especially those related to device input; e.g. switching to WebGL instead of using a web canvas.

END NOTES

More than half a million gestures were synthesized while preparing this manuscript. We will release the data so that others can build upon our work.

Our web application can be publicly accessed at **`http://personales.upv.es/luileito/g3/`**. Due to legal restrictions, the core software for gesture synthesis cannot be made publicly available as a standalone software. The interested people must sign an agreement with the École Polytechnique de Montréal through a collaborative project to get a license.

Solution to Figure 5

In each subfigure, the first dataset at the left is human generated, the dataset in the middle was synthesized using the worst reconstructed human sample of each gesture, and the dataset at the right was synthesized using the best reconstructed human sample.

ACKNOWLEDGMENTS

We wish to thank Benjamin Poppinga for sharing the “20 popular mobile actions” dataset prior to the official release date. We also thank the anonymous TIST referees for providing feedback to improve this article.

REFERENCES

M. A. Alimi. 2003. Beta neuro-fuzzy systems. In *TASK Quarterly J., Special Issue on Neural Networks*, W. Duch and D. Rutkowska (Eds.). Vol. 7. 23–41.

A. Almaksour, E. Anquetil, R. Plamondon, and C. O’Reilly. 2011. Synthetic handwritten gesture generation using Sigma-Lognormal model for evolving handwriting classifiers. In *Proc. Biennial Conf. of the Intl. Graphonomics Society (IGS)*.

S. Amini, and Y. Li. 2013. CrowdLearner: Rapidly creating mobile recognizers using crowdsourcing. In *Proc. Annual ACM Symp. on User Interface Software and Technology (UIST)*. 163–172.

L. Anthony, and J. O. Wobbrock. 2010. A lightweight multistroke recognizer for user interface prototypes. In *Proc. Graphics Interface (GI)*. 245–252.

L. Anthony, and J. O. Wobbrock. 2012. $N-protractor: a fast and accurate multistroke recognizer. In *Proc. Graphics Interface (GI)*. 117–120.

C. Appert, and S. Zhai. 2009. Using strokes as command shortcuts: Cognitive benefits and toolkit support. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 2289–2298.

D. Ashbrook, and T. E. Starner. 2010. MAGIC: A motion gesture design tool. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 2159–2168.

C. Bahlmann, B. Haasdonk, and H. Burkhardt. 2001. On-line handwriting recognition with support vector machines: A kernel approach. In *Proc. Intl. Workshop on Frontiers in Handwriting Recognition (IWFHR)*. 49–54.

O. Bau, and W. E. Mackay. 2008. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proc. Annual ACM Symp. on User Interface Software and Technology (UIST)*. 37–46.

A. Belaid, and J. Haton. 1984. A syntactic approach for handwritten formula recognition. *IEEE T. Pattern Anal.* 6(1), 105–111.

F. Beuvens, and J. Vanderdonckt. 2012. Designing graphical user interfaces integrating gestures. In *Proc. ACM Intl. Conf. on Design of Communication (SIGDOC)*. 313–322.

D. Bullock, and S. Grossberg. 1988. The VITE model: a neural command circuit for generating arm and articulator trajectories. In *Dynamic Patterns in Complex Systems*. 305–326.

B. Caramiaux, N. Montecchio, A. Tanaka, and F. Bevilacqua. 2014. Adaptive gesture recognition with variation estimation for interactive systems. *ACM T. on Interactive Intell. Syst.* 4(4), 18:1–18:34.

S. D. Connell, and A. K. Jain. 2000. Template-based on-line character recognition. *Pattern Recogn.* 34(1), 1–14.

G. Costagliola, V. Deufemia, G. Polese, and M. Risi. 2004. A parsing technique for sketch recognition systems. In *Proc. Symp. on Visual Languages and Human-Centric Computing (VLHCC)*. 19–26.

M. Davis, and T. Ellis. 1964. The RAND tablet: A man-machine graphical communication device. In *Proc. American Federation of Information Processing Societies (AFIPS)*. 325–331.

V. Deepu, S. Madhvanath, and A. G. Ramakrishnan. 2004. Principal component analysis for online handwritten character recognition. In *Proc. Intl. Conf. on Pattern Recognition (ICPR)*. 327–330.

J. Denier Van Der Gon, and J. Thuring. 1965. The guiding of human movements. *Kybernetik* 14, 145–148.

A. K. Dey, R. Hamid, C. Beckmann, I. Li, and D. Hsu. 2004. A CAPpella: Programming by demonstration of context-aware applications. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 33–40.

M. Djioua, and R. Plamondon. 2009. Studying the variability of handwriting patterns using the Kinematic Theory. *Hum. Mov. Sci.* 28(5), 588–601.

A. Feldman. 1966. Functional tuning of the nervous system with control of movement or maintenance of a steady posture. *Biophysics* 11(1), 565–578.

A. Fischer, R. Plamondon, C. O'Reilly, and Y. Savaria. 2014. Neuromuscular representation and synthetic generation of handwritten whiteboard notes. In *Proc. Intl. Conf. on Frontiers in Handwriting Recognition (ICFHR)*. 222–227.

T. Flash, and N. Hogan. 1985. The coordination of arm movements: an experimentally confirmed mathematical model. *J. Neurosci.* 5(7), 1688–1703.

J. Galbally, J. Fierrez, J. Ortega-Garcia, and R. Plamondon. 2012. Synthetic on-line signature generation. Part II: Experimental validation. *Pattern Recogn.* 45(7), 2622–2632.

D. Goldberg, and C. Richardson. 1993. Touch-typing with a stylus. In *Proc. INTERCHI'93 Conf. on Human Factors in Computing Systems*. 80–87.

D. Halbert. 1984. *Programming by Example*. Ph.D. Dissertation. U.C. Berkeley.

J. Hollerbach. 1981. An oscillation theory of handwriting. *Biol. Cybern.* 39(2), 139–156.

J. I. Hong, and J. A. Landay. 2000. SATIN: A toolkit for informal ink-based applications. In *Proc. Annual ACM Symp. on User Interface Software and Technology (UIST)*. 63–72.

J.-W. Kim, and T.-J. Nam. 2013. EventHurdle: Supporting designers' exploratory interaction prototyping with gesture-based sensors. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 267–276.

K. Kin, B. Hartmann, T. DeRose, and M. Agrawala. 2012. Proton: Multitouch gestures as regular expressions. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 2885–2894.

P. Koch, W. Konen, , and K. Hein. 2010. Gesture recognition on few training data using slow feature analysis and parametric bootstrap. In *Proc. IEEE Intl. Joint Conf. on Neural Networks (IJCNN)*. 1–8.

D. Kohlsdorf, T. E. Starner, and D. Ashbrook. 2011. MAGIC 2.0: A web tool for false positive prediction and prevention for gesture recognition systems. In *Proc. IEEE Intl. Conf. on Automatic Face & Gesture Recognition and Workshops (FG)*. 1–6.

D. K. H. Kohlsdorf, and T. E. Starner. 2013. MAGIC Summoning: Towards automatic suggesting and testing of gestures with low probability of false positives during use. *J. Mach. Learn. Res.* 14(1), 209–242.

M. Koschinski, H. J. Winkler, and M. Lang. 1995. Segmentation and recognition of symbols within handwritten mathematical expressions. In *Proc. Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*. 2439–2442.

J. A. Landay, and B. A. Myers. 1993. Extending an existing user interface toolkit to support gesture recognition. In *Proc. INTERCHI'93 Conf. on Human Factors in Computing Systems*. 91–92.

F. Leclerc, R. Plamondon, and G. Lorette. 1992. Des gaussiennes pour la modélisation des signatures et la segmentation des tracés manuscrits. *Trait. Signal* 9(4), 347–358.

N. Leduc, and R. Plamondon. 2001. A new approach to study human movements: The three dimensional Delta-Lognormal model. In *Proc. Biennial Conf. of the Intl. Graphonomics Society (IGS)*. 98–102.

L. A. Leiva, V. Alabau, V. Romero, A. H. Toselli, and E. Vidal. 2014. Context-aware gestures for mixed-initiative text editing UIs. *Interact. Comput.* 27(1).

Y. Li. 2010. Protractor: a fast and accurate gesture recognizer. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 2169–2172.

A. C. Long, J. A. Landay, and L. A. Rowe. 1999. Implications for a gesture design tool. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 40–47.

H. Lü, J. A. Fogarty, and Y. Li. 2014. Gesture Script: Recognizing gestures and their structure using rendering scripts and interactively trained parts. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 1685–1694.

H. Lü, and Y. Li. 2012. Gesture Coder: A tool for programming multi-touch gestures by demonstration. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 2875–2884.

H. Lü, and Y. Li. 2013. Gesture Studio: Authoring multi-touch interactions through demonstration and declaration. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 257–266.

F. J. Maarse. 1987. *The Study of Handwriting Movement: Peripheral Models and Signal Processing Techniques*. Swets & Zeitlinger.

D. Martín-Albo, R. Plamondon, and E. Vidal. 2014. Training of on-line handwriting text recognizers with synthetic text generated using the kinematic theory of rapid human movements. In *Proc. Intl. Conf. on Frontiers in Handwriting Recognition (ICFHR)*. 543–548.

R. Marzinkewitsch. 1991. Operating computer algebra systems by hand-printed input. In *Proc. Intl. Symp. on Algorithms and Computation (ISAAC)*. 411–413.

D. E. Meyer, J. E. K. Smith, S. Kornblum, R. A. Abrams, and C. E. Wright. 1990. Speed-accuracy tradeoffs in aimed movements: Toward a theory of rapid voluntary action. *Atten. and Perform.* 13(23), 173–226.

P. Morasso, F. A. Mussa Ivaldi, and C. Ruggiero. 1983. How a discontinuous mechanism can produce continuous patterns in trajectory formation and handwriting. *Acta Psychol.* 54(1), 83–98.

B. Myers. 1988. *Creating user interfaces by demonstration*. Academic Press.

P. Neilson. 1993. The problem of redundancy in movement control: the adaptive model theory approach. *Psychol. Res.* 55(1), 99–106.

C. O'Reilly, and R. Plamondon. 2009. Development of a Sigma-Lognormal representation for on-line signatures. *Pattern Recogn.* 42(12), 3324–3337.

T. Ouyang, and Y. Li. 2012. Bootstrapping personal gesture shortcuts with the wisdom of the crowd and handwriting recognition. In *Proc. SIGCHI Conf. on Human Factors in Computing Systems (CHI)*. 2895–2904.

K. Patel, N. Bancroft, S. M. Drucker, J. Fogarty, A. J. Ko, and J. Landay. 2010. Gestalt: Integrated support for implementation and analysis in machine learning. In *Proc. Annual ACM Symp. on User Interface Software and Technology (UIST)*. 37–46.

R. Plamondon. 1995a. A kinematic theory of rapid human movements. Part I: Movement representation and control. *Biol. Cybern.* 72(4), 295–307.

R. Plamondon. 1995b. A kinematic theory of rapid human movements. Part II: Movement time and control. *Biol. Cybern.* 72(4), 309–320.

R. Plamondon, A. M. Alimi, P. Yergeau, and F. Leelerc. 1993. Modelling velocity profiles of rapid movements: a comparative study. *Biol. Cybern.* 69(1), 119–128.

R. Plamondon, and M. Djioua. 2006. A multi-level representation paradigm for handwriting stroke generation. *Hum. Mov. Sci.* 25(4–5), 586–607.

R. Plamondon, and F. Lamarche. 1986. Modelization of handwriting: A system approach. In *Graphonomics: Contemporary Research in Handwriting*, H. S. R. Kao, G. P. van Galen, and R. Hoosain (Eds.). 169–183.

R. Plamondon, C. O'Reilly, J. Galbally, A. Almaksour, and E. Anquetil. 2014. Recent developments in the study of rapid human movements with the Kinematic Theory: Applications to handwriting and signature synthesis. *Pattern Recogn. Lett.* 35, 225–235.

R. Plamondon, C. O'Reilly, C. Rémi, and T. Duval. 2013. The lognormal handwriter: learning, performing, and declining. *Front. Psychol.* 4(1), 945:1–945:14.

B. Plimmer, and I. Freeman. 2007. A toolkit approach to sketched diagram recognition. In *Proc. British HCI Group Annual Conference on People and Computers*. 205–213.

B. Poppinga, A. S. Shirazi, N. Henze, W. Heuten, and S. Boll. 2014. Understanding shortcut gestures on mobile touch devices. In *Proc. ACM Conf. on Human-computer interaction with mobile devices and services (MobileHCI)*. 173–182.

D. Rubine. 1991. Specifying gestures by example. *Proc. annual Conf. on Computer graphics and interactive techniques (SIGGRAPH)* 25(4), 329–337.

B. Signer, U. Kurmann, and M. C. Norrie. 2007. iGesture: A general gesture recognition framework. In *Proc. Intl. Conf. on Document Analysis and Recognition (ICDAR)*. 954–958.

S. Smithies, K. Novins, and J. Arvo. 2001. Equation entry and editing via handwriting and gesture recognition. *Behav. Inform. Technol.* 20(1), 53–67.

L. D. Spano, A. Cisternino, F. Paternò, and G. Fenu. 2013. GestIT: A declarative and compositional framework for multiplatform gesture definition. In *Proc. ACM SIGCHI Symp. on Engineering Interactive Computing Systems (EICS)*. 187–196.

I. E. Sutherland. 1963. *Sketchpad: A man-machine graphical communication system*. Technical Report 296. Lincoln Laboratory, MIT.

A. J. W. M. Thomassen, P. J. G. Keuss, and G. van Galen. 1983. Motor aspects of handwriting. *Acta Psychol.* 54(1–3), 354.

G. van Seghbroeck, S. Verstichel, F. D. Turck, and B. Dhoedt. 2010. WS-Gesture, a gesture-based state-aware control framework. In *Proc. IEEE Intl. Conf. on Service-Oriented Computing and Applications (SOCA)*. 1–8.

R.-D. Vatavu. 2011. The effect of sampling rate on the performance of template-based gesture recognizers. In *Proc. Intl. Conf. on Multimodal Interaction (ICMI)*. 271–278.

R.-D. Vatavu, L. Anthony, and J. O. Wobbrock. 2012a. Gestures as point clouds: a $P recognizer for user interface prototypes. In *Proc. Intl. Conf. on Multimodal Interaction (ICMI)*. 273–280.

R.-D. Vatavu, C.-M. Chera, and W.-T. Tsai. 2012b. Gesture profile for web services: An event-driven architecture to support gestural interfaces for smart environments. In *Proc. Ambient Intelligence (AmI)*. 161–176.

P. Viviani, and T. Flash. 1995. Minimum-jerk, two-thirds power law, and isochrony: converging approaches to movement planning. *J. Exp. Psychol.* 21(1), 32–53.

J. O. Wobbrock, A. D. Wilson, and Y. Li. 2007. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proc. Annual ACM Symp. on User Interface Software and Technology (UIST)*. 159–168.

C. G. Wolf, and P. Morrel-Samuels. 1987. The use of hand-drawn gestures for text editing. *IJMMS* 27(1), 91–102.

S. Zhai, P. O. Kristensson, C. Appert, T. H. Anderson, and X. Cao. 2012. Foundational issues in touch-surface stroke gesture design — an integrative review. In *Foundations and Trends in Human-Computer Interaction*. Vol. 5. 97–205.