Attentive Sequence-to-Sequence Modeling of Stroke Gestures Articulation Performance

Lokesh Kumar T. and Luis A. Leiva

***Abstract*—Production time of stroke gestures is a fundamental measure of user performance with Graphical User Interfaces. However, production time represents an *overall* quantification of the user's gesture articulation process and therefore provides an incomplete picture of such process. Moreover, previous approaches assumed stroke gestures as *synchronous* point sequences, when most gesture-driven applications have to deal with *asynchronous* point sequences. Furthermore, deep generative models of human handwriting ignore the temporal information, thereby missing a key component of the user's gesture articulation process. To solve these issues, we introduce DITTO, a sequence-to-sequence deep learning model that estimates the *velocity profile* of any stroke gesture using spatial information only, providing thus a fine-grained estimation of the *moment-by-moment* behavior of the user's articulation performance. We show that this unique capability makes DITTO remarkably accurate while handling gestures of any type: unistrokes, multistrokes, and multitouch gestures. Our model, code, and associated web application are available as open source software.**

***Index Terms*—Stroke Gestures; Touch Gestures; Human Performance; Time Estimation; Deep Learning**

I. INTRODUCTION

STROKE gestures (also known as touch, pen, stylus, finger, ink, or handwritten gestures) represent two-dimensional trajectories as pathlines and curves that make up geometric symbols which are mapped to specific actions and user interface commands. Stroke gestures are increasingly becoming a predominant input modality in today's graphical user interfaces (GUIs). Compared to traditional input techniques based on item selection from menus, stroke gestures are not only faster, but they also reduce users' cognitive load and visual attention [1] and increase usability [2].

Research in Human-Computer Interaction (HCI) has demonstrated the practical convenience and utility of employing stroke gestures as efficient *shortcuts* to access system functions [3] or specific applications [4]. For example, drawing an 'S' shape on a mobile phone screen can be used to search in the address book [3] or speed-dial some contacts. More recently, the massive online game 'Harry Potter: Wizards Unite' required players to draw stroke gestures to create spells and defeat enemies [5]. Stroke gestures also represent an effective input modality for users with low vision to interact with mobile devices [6] by providing a practical alternative to selecting touch targets that are challenging to see on small screens or that cannot be selected by visually-impaired users [7].

Research conducted under the Aalto Science Institute internship program.

Lokesh Kumar T. is with the Indian Institute of Technology Madras, Chennai, India (email: lokesh.karpagam@gmail.com). Luis A. Leiva is with the University of Luxembourg, Esch-Sur-Alzette, Luxembourg (email: name.surname@uni.lu).

Yet another direction of previous research has demonstrated the viability of stroke gesture input for user authentication [8] and biometrics [9]. Fundamentally, research on stroke gesture input has made possible new text entry techniques for mobile devices, such as shape-writing [10], ubiquitous on today's smartphones.

Designing stroke gestures often involves user studies and experiments in an iterative design process consisting of prototyping, implementation, verification, and validation steps. Gestures should be recognized robustly by a computer [11] and, at the same time, they should represent a good fit to the functions they effect [12], be easy to articulate [13] and straightforward to recall by users [14]. Despite the practical benefits of involving actual users in this process, conducting user studies and experiments to collect gesture data and inform design takes an important amount of time, effort, and resources [15], which unnecessarily delays the launch date of software applications.

Modeling stroke gestures, therefore, is an important and challenging task. Instead of recruiting users for controlled studies, the alternative option for GUI designers and practitioners is to rely on computational models of human performance with stroke gesture input [16]–[19]. Such models and their associated prediction techniques can save precious time and provide insightful information about suitable gesture commands right from the very early stages of design. To begin with, having such estimations would represent a valuable asset for practitioners, enabling them to explore various gesture set designs with minimum effort.

Our approach makes it possible to anticipate *how* gestures are likely to be executed by end users, assuming no more information than a sequence of (x, y) points. A practical use case is illustrated in Figure 1. Imagine a GUI designer willing to implement a gesture to trigger a delete command and they are unsure whether to use either a 'd' or a 'Z' letter shape. By having a moment-by-moment estimator of gesture articulation, they can foresee the temporal evolution of the articulation process and make an informed decision about the suitability of such gesture candidates. For example, as shown in the figure, the first stroke of the 'd' letter is executed very quickly (200 ms) and with more determination than the remaining stroke. This behavior is also observed in the second and third sub-movements of the 'Z' letter, which is stereotypical of a planned action, or *aimed movement*. Aimed movements are goal-directed [20] and performed almost without thinking [21], therefore they are preferred since they are easier to perform. The designer concludes that the 'Z' shape should be used to trigger the delete command.

Other similar examples can be easily imagined by the

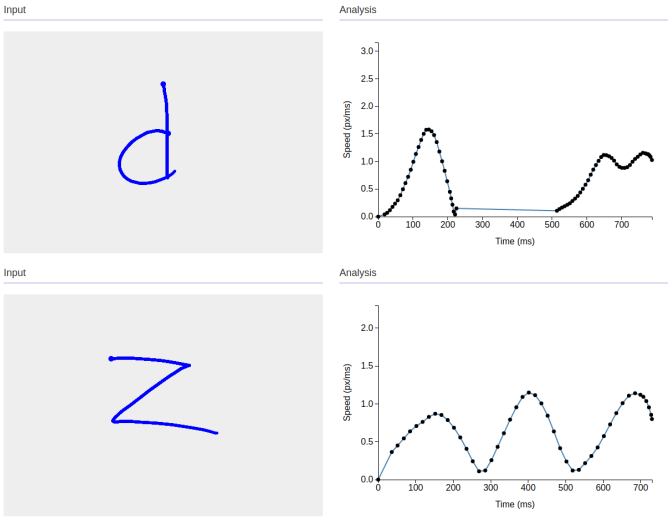


Fig. 1: Screenshots of our companion web application. DITTO takes any stroke gesture, defined as a sequence of consecutive (x, y) points, and computes the most probable *moment-by-moment* articulation, thereby providing detailed information about such articulation beyond its (overall) production time. Users can upload their own gestures in JSON format.

reader, such as predicting the perceived user's articulation difficulty [13] or quantifying variation in gesture input [22]. Overall, predictive models of gesture production time can help the design and evaluation of existing or future gesture interfaces by quantitatively predicting their *a-priori* efficiency, before running any user studies. In sum, predictive models of stroke gesture articulation can save both precious time and cost, by allowing HCI stakeholders (e.g. GUI designers, developers, researchers, and practitioners) to anticipate plausible performance results with a particular set of gestures.

A. Contributions and Applications

Current state-of-the-art computational models of stroke gestures production time have relied on the Kinematic Theory [23] to produce *overall* estimations of production times [18], [22], assuming *synchronous* sequences of 2D points with associated timestamps. However, in most gesture-driven applications, including web-based and smartphone-based, the point sequences captured by the application are *asynchronous*, i.e. the sampling rate is not constant.[1] Moreover, in many stroke gesture datasets the temporal information is missing [25] and even corrupted [26], so previous models like KeyTime [18] and GATO [22] cannot be used to estimate production times in those cases, at least not directly.[2]

To address these issues, we have developed DITTO (Deep tIme esTimaTOr), a sequence-to-sequence deep learning model that estimates the velocity profile of any stroke gesture using spatial information as only input, providing thus a fine-grained analysis of the *moment-by-moment* behavior of user's articulation performance. Our model can handle asynchronous stroke gestures with remarkable accuracy, thereby providing a comprehensible picture of stroke gesture articulation. Also importantly, DITTO opens the door to using modern synthetic data generation techniques such as GPSR [25] and SketchRNN [27], which ignore the temporal information, to produce a complete picture of human-like artificial samples.

We foresee several applications of DITTO, among which we should emphasize the following ones:

1) **Analysis tool:** Understanding stroke gesture articulation performance beyond overall production times.
2) **Recovery method:** Fixing datasets where the temporal information is missing, corrupted, or ill-estimated.
3) **Augmentation aid:** Complementing the output of generative models with plausible temporal information.

In sum, DITTO substantially advances the state of the art and represents the only general-purpose, comprehensive, and detailed production time estimation model for stroke gesture input. DITTO can be accessed as a web application and as a web service (RESTful API), so it can be easily integrated with third party apps; e.g. as a software plugin of a GUI design program such as Sketch[3] or Figma.[4]

II. RELATED WORK

Velocity profiles (Figure 1) are asymmetric bell-shaped curves generally observed in human handwriting, including gesturing and sketching, as well as in other rapid movements such as head and eye movements [23], [28]. Velocity profiles are thus a key component in human movement analysis to understand the organization and production of such movements, including those involved in the articulation of stroke gestures. From a practical perspective, human handwriting models based on velocity profiles not only provide a parametric description of movement signals, but they can also be used to segment complex movements and to compress and store data efficiently [29].

The Kinematic Theory [23], [30] is a very popular framework to synthesize and analyze human handwriting with the so-called Sigma-Lognormal ($\Sigma\Lambda$) model [28]. However, practical applications of the Kinematic Theory in HCI have been primarily directed at generating gesture data [6], [25], [31] and estimating gesture production times [18], [22]. These models enforce a synchronous stream of 2D coordinates as input, for which they resample the data to a constant sampling rate [31]. In contrast, DITTO operates over the original (unmodified) stroke data.

A. Stroke Gestures

Simple forms of stroke-based input, such as pointing and item selection from menus, have been studied using Fitts' law [32] and its several variations. However, more complex stroke input, such as handwriting, shape-writing, or free-form

[1] In web development, for example, browser events are placed in an event queue and the browser dispatches these events at unpredictable times [24] by invoking any established handlers.

[2] When timing information is missing or corrupted, previous techniques can only assume a synchronous, fixed sampling rate. This is actually a workaround, to help circumventing those cases in practice.

[3] https://www.sketch.com/

[4] https://www.figma.com/

gesture paths, requires more sophisticated models to characterize human performance effectively. A comprehensive survey in this area is provided by Müller et al. [33]. Interestingly, in a cross-cultural study involving forty people from nine countries, Mauney [34] noticed that the most common gesture for the "Print" function was drawing a letter, though the cultural background of the participants had a strong influence on the choice of the gestures overall, e.g., Chinese participants employed symbolic gestures more frequently than participants from other countries. In a large-scale in-the-wild study with 388 participants, Poppinga et al. [35] observed that mobile users preferred to use symbolic and letter-shaped gestures for launching mobile apps.

There are many ways to produce a stroke gesture depending on the number of strokes or the number of fingers and hands touching the screen. For example, common touchscreen technology found in commodity smartphones and tablets can detect multiple discrete touch points at once, which enables practitioners to access a rich design space of unistrokes, multistrokes, multitouch, and bimanual gestures for their user interfaces and applications. In addition, expert gesture designs often involve the use of multiple fingers [36], various finger parts [37], or even the entire hand for expressive input [38]. At the same time, users are known for exhibiting variations in articulating multistroke and multitouch gestures in terms of the number of strokes and fingers touching the screen [39] especially when there are no constraints imposed. Therefore, an ideal human performance estimation technique should be able to handle all sorts of measurable variation in stroke gesture input.

B. User Performance with Stroke Gesture Input

Early work by Isokoski [17] proposed a very simple model based on line counting to estimate handwriting time. Unfortunately, this model makes no specific time quantification for a given gesture. Instead, the model is useful for computing a relative ranking of a given set of gestures, provided that they are distinct enough. For example, Isokoski's model would predict no difference between the letters 'L', 'T', 'V', and 'X', as all of them have the same number of lines. However, 'L' and 'V' are drawn with a single stroke, whereas 'T' and 'X' are drawn with two strokes, so the latter letters usually take more time and effort to draw that the formers.

Cao and Zhai's CLC model [16] was specifically designed to estimate the magnitudes of production time for *unistroke* gestures. The CLC model operates by dividing the gesture shape into curves, lines, and corners, for which production times are estimated separately. The resulting time for a particular gesture is computed as the sum of the individual production times needed to articulate each of the gesture's elementary curves, lines, and corners. The CLC model works very well as a first-order estimator, however it can only provide a single estimation value, which is insufficient to characterize the variation in gesture articulation within and between users [40] (low flexibility). Also, CLC is known to overestimate the actual magnitudes of production times [16] (low accuracy), presumably because it doesn't compensate for user articulation skills. To address these issues, Leiva et al. [18] introduced KeyTime, a technique that works for unistroke gestures and accepts free-form drawing as input, and GATO [22], an extension of KeyTime for multistroke and multitouch gestures. However, as stated previously, these techniques provide an *overall* estimation of a gesture's production time and assume *synchronous* spationtemporal sequences. And if the temporal information is missing or corrupted, these techniques can only resample the original data assuming a constant frequency.

Gesture features and measures have also been used to inform the design of gesture sets. For example, Long et al. [41] were interested in gesture shapes that would be easy for users to learn and recall. Researchers have employed other gesture measures to understand differences in performance between users or between input conditions. For example, Vatavu et al. [40] used relative accuracy measures to quantify deviations from "ideal" gesture shapes or templates from a training set. Such gesture measures have proven very useful to characterize various aspects of stroke gesture input as well as to inform gesture-based user interface design. However, another line of work has focused on a more fundamental understanding of human movements during stroke gesture production by relating to key aspects from the motor control theory. We discuss this work in the following section.

C. Human Movement Models

Viviani et al. [42] were among the first to investigate the fundamental aspects of human handwriting and drawing behavior. Since then, a fruitful line of research has been the application of minimization principles to motor control, such as Flash and Hogan's Minimum-Jerk Theory [43]. Further investigations showed that lognormal-based models, such as those postulated by the Kinematic Theory [23], [44], are arguably the most accurate descriptors of human movements known today [45], compared to which *"other models can be considered as successive approximations"* [46]. Actually, it has been shown that the concepts postulated by the Minimum-Jerk Theory and the Kinematic Theory are linked and describe, with different arguments, a model of velocity profiles [47].

Finally, we should mention the gesture path stochastic resampling (GPSR) technique [25], which is strongly focused on rapid GUI prototyping, is computationally efficient, and has minimal coding overhead. However, GPSR does not synthesize timestamps, thereby precluding a fine-grained analysis of handwriting behavior. Overall, current generative models of human handwriting ignore the temporal component; c.f. SketchRNN [27] or FG-SBIR [48] to name some recent examples. This is presumably so because there is a fundamental issue from a modeling perspective: the same spatial trajectory can be articulated in several ways. For example, even after years of practice, the same handwritten trace executed twice will probably be different due to the inherent variability of our motor control system. Ultimately, so, time is a fundamental component of human movement. Thus, ignoring the temporal component provides an incomplete picture for researchers trying to create accurate models of human movement behavior.

III. System Overview

Using the spatial information of a handwriting movement (represented as a sequence of 2D points) as sole input data, Ditto learns an internal representation that decodes the latent temporal information, which is "hidden" in the input sequence. At the technical level, Ditto estimates the most probable time offset (in milliseconds) between two consecutive spatial coordinates. As a result, Ditto generates the expected velocity profile of such an input sequence. This task can be formulated as a sequence-to-sequence learning problem and thus can be effectively solved using modern deep learning techniques.

Ditto follows an encoder-decoder architecture, as shown in Figure 2. The core component of our computational model is the Long Short Term Memory (LSTM) network, a specific type of recurrent neural net (RNN) that can handle long-term non-linear dependencies within the data. RNNs are networks with loops, allowing information to persist, and have been successfully used in a variety of problems, from speech recognition to language modeling, machine translation, and image captioning. However, a particularly annoying issue with RNNs is that they can run quickly into vanishing gradient problems.[5] The LSTM network was designed to solve this issue and so, since stroke gestures are of sequential nature by definition, it seemed natural for us to use the LSTM network to model the articulation of stroke gestures.

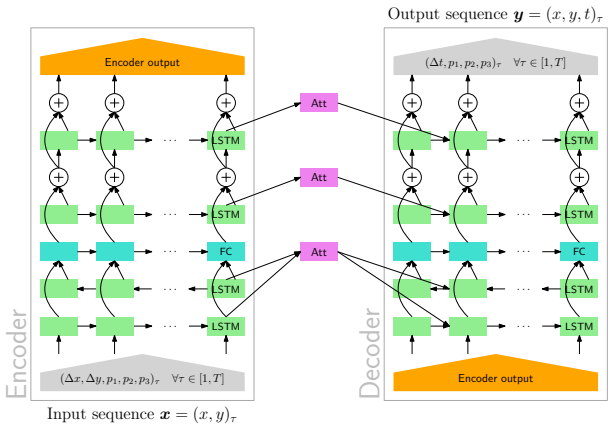


Fig. 2: System architecture of Ditto. The encoder transforms the input sequence $\boldsymbol{x}$ into a sequence of hidden representations, which the decoder extracts through an attention mechanism and generates the output sequence $\boldsymbol{y}$ autoregressively for every timestep τ. FC stands for 'Fully Connected' layer.

As observed in Figure 2, we stack several LSTM layers in order to increase the learning capability of Ditto, by feeding the last LSTM layer with the hidden states of the previous LSTM layer. The final number of LSTM layers is determined in a later experiment; see Section V. We use L2 weight regularization (weight decay) with $\alpha = 10^{-3}$ to prevent overfitting. Further, to stabilize training and improve convergence, we use residual connections [49] and clipnorm of 1.0, the latter value being widely used to avoid exploding gradients. We also add a Dropout layer right before the decoder output with drop rate of 0.5, which means that 50% of the neurons are randomly dropped during training. This increases the generalization capability of our model. Finally, we train Ditto with the popular Adam optimizer, using learning rate $\eta = 0.001$ and momentum terms $\beta_1 = 0.9, \beta_2 = 0.999$. We set a maximum number of 100 epochs but use early stopping with patience of 20 epochs while monitoring the validation loss, i.e., training ends as soon as no further gains in performance are achieved in 20 consecutive epochs.

In order to speed up training on a single GPU card, we set a maximum sequence length of 100 timesteps per stroke, which is larger than the median gesture length in the datasets we have tested (Section IV). Without this fixed length, we would have to use very small batch sizes for training, in order to account for outliers and exceedingly large gestures, thereby unnecessarily slowing down the training process.

A. From Spatial Encoding to Temporal Decoding

At each timestep τ, Ditto processes a 5-dimensional vector $(\Delta x, \Delta y, p_1, p_2, p_3)$, where the first two components represent the spatial offset, or displacement, from the previous position in the horizontal (x) and vertical (y) axes, respectively, and the last three components represent a one-hot encoded vector of 3 possible pen tip states [27]. The first pen state, p_1, indicates that the pen is currently touching the paper and that a line will be drawn connecting the next point with the current point. The second pen state, p_2, indicates that the pen will be lifted from the paper after processing the current point and that no line will be drawn afterwards. The final pen state, p_3, indicates that the drawing has ended and that subsequent points, including the current point, will not be further considered. By using such one-hot encoded pen states, the decoder can better anticipate the end of a stroke, which is particularly helpful for modeling multistroke gestures. The decoded output sequence follows a similar formulation but predicts a temporal offset Δt, therefore the decoder produces a 4-dimensional vector $(\Delta t, p_1, p_2, p_3)$.

For the input layer of the encoder network, we used the bidirectional LSTM variant (BLSTM) which takes as input the concatenation of the input sequence in both forward and backward direction (hence the bidirectional name), because BLSTMs can learn representations from both past and future timesteps. This contributes to a better understanding of the sequence context and thus can eliminate potential ambiguities. In our experiments, we have observed little difference between stacking 3 BLSTM layers and using only one BLSTM layer followed by 2 LSTM layers, therefore we decided not to use more than one BLSTM and keep the model simple. The output of the encoder network is a fixed latent vector which is then passed to the decoder network, i.e., the cell states of the input layer of the decoder network are initialized with the cell states of the output layer of the encoder network; see Figure 2. The decoder network is also a stacked deep LSTM network, with the same specifications as the encoder network plus a self-attention mechanism, which we describe in the next section.

[5] In neural networks, information back-propagates during training in the form of error/loss gradients (partial derivatives) to update the network weights accordingly. When gradients are very small, information gain will eventually become zero as they flow back to earlier layers, resulting in a lack of learning.

B. Attention Mechanism

The encoder-decoder architecture enables the use of recurrent neural networks to address challenging sequence-to-sequence problems. One of the main limitations of this architecture is that the decoder is initialized with a fixed-length latent vector so it is easy to lose information and perform worse while decoding longer sequences. Attention is a popular extension to mitigate this issue. With attention, DITTO can provide a richer encoding of the input sequence from which to construct a context vector that can then be used by the decoder network. This allows the model to learn what parts of the input sequence to pay attention to (and to what degree) during the prediction of the output sequence. Currently there are several popular attention "flavors", depending on the task or problem at hand. For sequential data, we can mention local [50] and global [51] attention. Overall, global attention is a simplification of local attention and sometimes achieves better results [51]. For this reason, we decided to use global attention in DITTO. The main difference between them is how to score the alignment between decoder input and encoder outputs. Both local and global attention have proved very successful in machine translation tasks. The computer vision community differentiates between soft and hard attention, which essentially are equivalent to global and local attention, respectively [52]. A third category is that of self-attention [53], which focuses on different positions of the input sequence to compute a representation of the same sequence. It has been shown to be very useful in machine reading and summarization tasks.

As mentioned previously, DITTO implements global attention, where every encoder state and all decoder states prior to the current state are taken into account to compute the alignment weights. Let h_τ represent the hidden state vector of the encoder network at timestep τ. Given that our encoder network begins with a BLSTM network, there are 2 hidden state vectors at each timestep, corresponding to the forward ($\overrightarrow{h_\tau}$) and backward ($\overleftarrow{h_\tau}$) directions. Therefore, the effective hidden state of the encoder is the concatenation of both hidden state vectors. Our decoder network has hidden state $s_\tau = f(s_{\tau-1}, y_{\tau-1}, c_\tau)$ for the output sequence at timestep τ, where the final output of the encoder is kept as context vector c_τ:

$$c_\tau = \sum_{j=1}^{T} a_{\tau j} h_j \tag{1}$$

where each weight $a_{\tau j}$ is the amount of attention paid to the corresponding encoder output h_j, which is computed using (1) the last hidden state $s_{\tau-1}$ and h_j, marginalizing over all hidden states, and (2) a feedforward neural network that is jointly learned with the rest of the model.

C. Loss Function

DITTO uses a dual-objective loss function for end-to-end training of both the encoder and decoder networks:

$$\mathcal{L} = \ell_{\text{MSE}} + \ell_{\text{state}} \tag{2}$$

where ℓ_{MSE} is the mean squared error loss between the true time offset Δt and the predicted time offset $\hat{\Delta} t$, and ℓ_{state} is the cross-entropy loss between the true pen state p_i and the predicted pen state probability q_i, respectively:

$$\ell_{\text{MSE}} = \frac{1}{|\boldsymbol{x}|} \sum_{\tau=1}^{T} (\Delta t - \hat{\Delta} t)_\tau^2 \tag{3}$$

$$\ell_{\text{state}} = \frac{1}{|\boldsymbol{x}|} \sum_{\tau=1}^{T} \left(- \sum_{i=1}^{3} p_i \log q_i \right) \tag{4}$$

where $\boldsymbol{x}$ denotes a stroke sequence with τ timesteps. This loss function is jointly optimized so that at every timestep DITTO predicts the most likely temporal offset between consecutive coordinates, guided by the most probable pen state. We deliberately omit the τ sub-index in Equation 4 for the variables p_i and q_i in order to make the equation more readable.

IV. EVALUATION

We evaluate the performance of DITTO on a variety of datasets and a wide range of stroke gesture types, aimed at covering the full spectrum of stroke gestures; i.e., unistrokes, multistrokes, and multitouch gestures. Figure 3 and Table I provide an overview of these.

A. Gesture Datasets

In the following we describe the public datasets we used for evaluation. They include examples of both simple and complex gestures, produced with different devices and under different execution speeds. Taken together, these datasets constitute a relevant testbed for conducting replicable research on stroke gestures.

1) The GDS[6] dataset comprises $4,800$ samples of 16 distinct **unistroke** gestures (see Figure 3a) performed by 10 participants with a stylus on an iPAQ Pocket PC [54]. Because participants were asked to articulate gestures at three different speeds (slow, medium, and fast), we split the dataset according to each articulation speed in our analysis:
   a) GDS-fast: $1,600$ gestures performed by all participants at fast speed, 10 executions per participant per gesture type. Participants received the instruction "go as fast as you can"; see Wobbrock et al. [54] (p.164).
   b) GDS-medium: $1,600$ gestures performed at medium speed by all participants, 10 executions per participant per gesture type. Participants received the instruction "balance speed and accuracy".
   c) GDS-slow: $1,600$ gestures performed at slow speed by all participants, 10 executions per participant per gesture type. Participants received the instruction "be as accurate as possible".

2) The NicIcon[7] dataset comprises $13,860$ samples of 14 distinct **multistroke** gestures (see Figure 3b) performed by 33 participants with a stylus on a Wacom Intuos2

[6]https://depts.washington.edu/acelab/proj/dollar/
[7]http://unipen.nici.ru.nl/NicIcon/

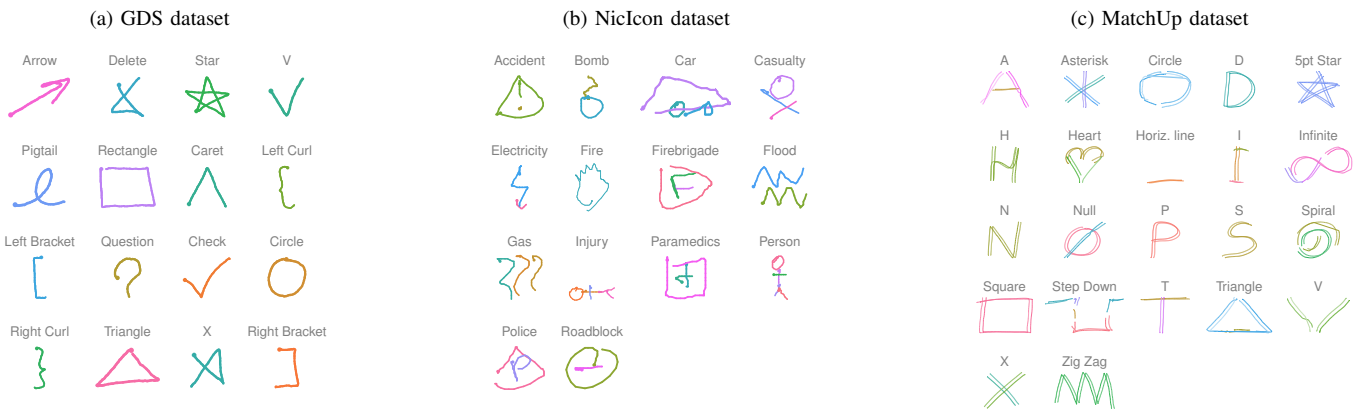


Fig. 3: Gesture types (actual examples) from our evaluation datasets. Each gesture stroke is signified with a random color.

tablet [55]. Each participant provided 30 executions per gesture type.

3) The MatchUp[8] dataset comprises 5,155 samples of 22 distinct **multistroke-multitouch** gestures (see Figure 3c) performed by 16 participants with the finger on a 3M C3266PW 32" multitouch display [56]. For each gesture type, participants were asked to produce as many different variations as possible, using either: one or two hands, one or multiple fingers, one or multiple strokes. Each participant provided 5 executions per variation and gesture type.

Our datasets contain a good mixture of geometrical shapes and symbols, including popular [54], challenging [55], and unfamiliar [56] ones, with a wide range of execution complexity [17], ranging from unistrokes to multistrokes and multitouch gestures, the latter performed using one or both hands [56]. Taken together, the three datasets contain a diverse type of stroke gestures articulated under various input conditions regarding number of strokes, fingers, and hands.

Dataset	Samples	Classes	Users	Trials	Timesteps
GDS-fast	1,600	16	10	10	54
GDS-medium	1,600	16	10	10	66
GDS-slow	1,600	16	10	10	79
NicIcon	13,860	14	33	30	88
MatchUp	5,155	22	16	5	82

TABLE I: Overview of the evaluation datasets. The 'Timesteps' column denotes the median number of timesteps, which we use to inform the size of the input layer (number of neurons) of our model.

B. Data Ingestion

Every gesture is presented to DITTO as a flat point sequence (Section III-A) of variable length, but the input layer of the network has a finite (fixed) capacity and thus is unable to process sequences that exceed the network capacity. As explained in Section III, DITTO has a maximum sequence length of 100 timesteps by design. In practice, longer sequences can be truncated, though we eventually decided to ignore those in our experiments (roughly 20% of the data) because processing incomplete sequences could mislead the results. On the other hand, sequences shorter than the capacity of the input layer (100 timesteps) are padded with dummy values, as usual, since the input layer of the network has a fixed capacity.

Regarding model training, we use 60:10:30 (train:validation:test) randomized splits; i.e., we train DITTO on a training partition comprising 60% of the data from each dataset and reserve the remaining 30% of the data for testing the trained model. We also reserve 10% of the original data for fine-tuning the model hyperparameters. This way, the testing partition simulates unseen data and therefore provides an estimation of the model performance in practice.

In addition, we should mention that DITTO uses batch processing and so it can process several gesture samples at the same time, both during training and testing time. In the latter case it is possible to speed up data ingestion further, e.g. by running different threads or child processes, since DITTO uses reentrant code, thereby allowing for parallelizing data processing.

C. Evaluation Measures

First, we report the following evaluation measures of overall production times, to make this work comparable with previous work such as KeyTime [18] or GATO [22]:

1) *Mean Absolute Error* (MAE), which averages the absolute differences between predicted and actual production times, in milliseconds.
2) *Root Mean Squared Error* (RMSE), which averages the squared differences (and thus penalizes larger errors) between predicted and actual production times, in milliseconds.
3) *Pearson's r* (correlation coefficient), which measures the linear correlation between predicted and actual production times, and can range between -1 and 1 (with 0 implying no correlation).

Next, we compare predictions delivered by DITTO and the ground-truth data using macro-average point-wise articulation

[8]https://sites.google.com/site/yosrarekikresearch/projects/

metrics, i.e., measured at every timestep along the articulation path of each gesture and then aggregated in a single value.

1) *Cumulative time offset*: the distribution of the (cumulative sum) of time differences between consecutive timesteps, in milliseconds. The computed value at the end of this curve is the overall production time.
2) *Velocity profile alignment* between predicted and actual gesture execution. For this, we use the same error and correlation measures as defined above but computed at every timestep.

D. Baseline Model

We further compare DITTO's performance against a naive time estimator, which is used under the hood by the $\Sigma\Lambda$ model [23] and similar approaches. Remember that state-of-the-art models of production time for stroke gestures [18], [22] use the $\Sigma\Lambda$ model to get an initial reconstruction of the gesture and then bootstrap the time computation with synthetic samples. As discussed in Section I, these state-of-the-art models produce *overall* estimations of production times assuming *synchronous* sequences of 2D points with associated timestamps.

The naive time estimator model assumes that the input device operates at a fixed frequency, or sampling rate, while recording a stroke gesture $\mathcal{G}$, so the overall production time T can be computed by:

$$T = f \sum_{\boldsymbol{x} \in \mathcal{G}} \frac{1}{N} \sum_{p \in \boldsymbol{x}} p_{/\!/} \tag{5}$$

where f denotes the sampling frequency of the acquisition device (in Hz), $\boldsymbol{x}$ denotes a stroke, $p_{/\!/}$ denotes the stroke points drawn in parallel, to account for multitouch gestures (where several fingers are drawing on the screen at the same time), and $N = |p_{/\!/}|$. Essentially, Equation 5 computes the average number of points that happened at a particular timestep and multiplies that number by the sampling rate f to get an estimation of the overall time production. Notice that for single-touch gestures $N = 1$ (only one finger is used) and for unistroke gestures $|\mathcal{G}| = 1$ (only one stroke is articulated). We will test different sampling rates for the naive time estimator, since each dataset may have a preferred indicative value, or sweet spot. This estimator therefore qualifies as a reasonable baseline model to compare against DITTO.

V. RESULTS

Note that most gesture datasets in the research literature are available at one input speed only, typically a "medium" speed definition, where participants have to balance speed and accuracy and operate at their own tradeoff [54], [57]–[59]. Therefore, we begin by analyzing the GDS dataset, since it has three different articulation speeds and therefore can be used to better inform the design of DITTO's model architecture.

A. Model Architecture

In our first experiment we analyze the impact of model depth, as determined by the number of stacked recurrent layers, and the incorporation of self-attention mechanisms for the task of predicting overall production times. Table II reports the results of this experiment. As discussed in Section III-A, the first recurrent layer in any case is always a BLSTM layer, and further recurrent layers are LSTM layers. We also report in Table II the Pearson's correlation coefficient (r) between the model predictions and the ground-truth values. Since each dataset comprises repeated measures, i.e. the same participant performed the same gesture under different trials (Table I), we aggregate the data by participant and gesture type.

		Without attention			With attention		
Depth	**Speed type**	**MAE**	**RMSE**	r	**MAE**	**RMSE**	r
1	Slow	88	114	0.908	145	178	0.752
1	Medium	104	124	0.926	120	152	0.823
1	Fast	108	113	0.988	165	173	0.975
1	*Overall*	100	117	0.969	144	168	0.911
2	Slow	92	12	0.890	89	107	0.947
2	Medium	84	101	0.932	155	166	0.952
2	Fast	91	98	0.987	169	175	0.987
2	*Overall*	89	108	0.964	138	152	0.974
3	Slow	108	130	0.915	130	178	0.859
3	Medium	151	164	0.940	72	86	0.876
3	Fast	136	142	0.992	39	49	0.989
3	*Overall*	132	146	0.970	80	118	0.953

TABLE II: Analysis of model depth (number of stacked recurrent layers) and self-attention mechanism over the testing partition of the GDS datasets. Errors are expressed in milliseconds. All correlations are statistically significant at the $p < .001$ level.

We can see that a non-attentive computational model can benefit from stacking up to 2 recurrent layers; adding a third layer degrades performance, possibly due to a shortage of training data. But if we incorporate self-attention, then a deeper model achieves the best results overall. While for slow gestures the errors are slightly larger than the non-attentive counterpart, the attentive model outperforms the estimations of medium and fast gesture executions, often by a large margin; e.g. for fast gestures MAE is near four times smaller (39 vs 136 ms) and RMSE is about three times smaller (49 vs 142 ms). Moreover, the non-attentive model is systematically skewed (see Figure 4) and so correlates a bit better than the attentive model, though differences are not statistically significant. We also tested the effect of adding self-attention to the first and last recurrent layers only, but they proved to work less effectively than adding attention to all recurrent layers.

We conclude that the attentive 3-layered deep model is the best performer overall. The non-attentive 2-layered model should be considered only when the estimation of slow gestures is important for the task at hand. Nevertheless, this experiment informs only about the overall production time, so we need to look into velocity profile alignment measures to get a better understanding about the articulation process of stroke gestures input; i.e., the moment-by-moment estimations, at every timestep. In a later experiment we precisely conduct this investigation.

B. Self-attention mechanism

Next, we study the effect of using self-attention mechanisms on velocity profiles estimation, for different articulation speeds, for which we look at the cumulative time distribution; i.e. at each timestep we plot the accumulated time offset, aggregated by participant and gesture class. Figure 4 and Figure 5 show the results of this experiment. We can see that the plots follow a non-linear steeper curve, signaling thus the asynchronous nature of stroke gestures execution. Again, we can observe that incorporating self-attention increases model performance. Without attention, the model estimations tend to degrade over time. This is caused by a drift effect, where small errors accumulate over time (Figure 4) and thus the estimated time is far off the ground-truth time at the end of the gesture execution. This behavior is noticeably more accentuated in faster executions. Indeed, if a handwriting movement is executed at slow speed, the model can easily estimate the temporal offsets because these are more predictable. For fast speeds, however, the estimation becomes much more challenging, and only the model with self-attention (Figure 5) is able to capture best the articulation behavior exhibited by the ground-truth movements.

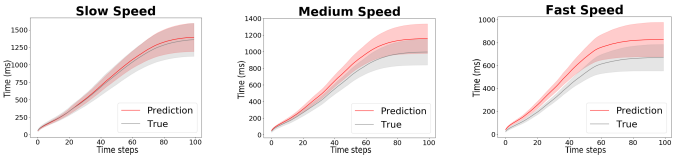


Fig. 4: Cumulative time offset distribution in the GDS dataset (test partition), using the non-attentive 2-layered model.

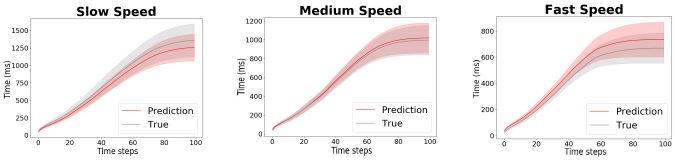


Fig. 5: Cumulative time offset distribution in the GDS dataset (test partition), using the attentive 3-layered model.

C. Gesture Articulation

So far, our experiments have provided a practical justification of the theoretical grounds underlying DITTO's model architecture and self-attention mechanism. The encoder-decoder network that we use in further experiments is the network with 3 stacked recurrent layers (BLSTM→LSTM→LSTM) and self-attention on all recurrent layers. Figure 6 show some examples of the velocity profile estimations produced by DITTO on the GDS dataset. In order to remove noise from the sensor of the acquisition device,[9] we apply a Savitzky-Golay filter of window size 21 and polynomial order 2. These filter parameters were the best compromise solution for all datasets. The critical parameter is the window size, as its role is to attenuate the raw signal's jitter without being too aggressive as to flatten the signal.

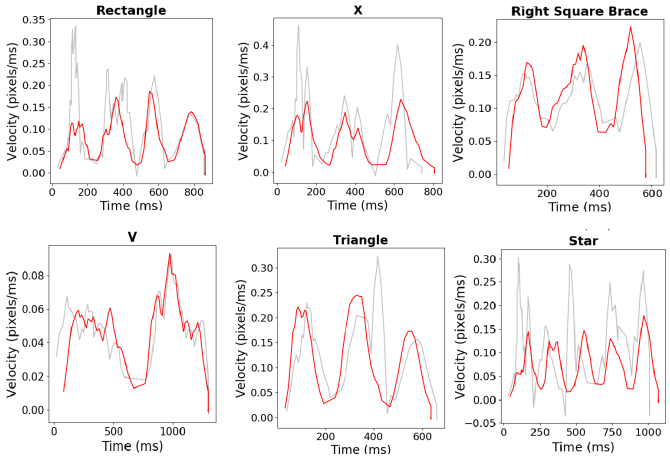


Fig. 6: Velocity profile estimations in the GDS dataset (test partition). Gestures examples picked at random.

Table III summarizes the prediction results for the remaining datasets, including both production time and velocity profile alignment measures. For completeness, and by way of summary, we also report the results over the GDS dataset. As can be observed in the table, DITTO is off by a very small margin and correlates highly with ground-truth data. The MatchUp dataset is particularly challenging, since each gesture could be executed with different number of fingers, hands, and strokes [56], thereby entailing greater variability than in the other datasets.

The naive time estimator is unable to deliver competitive results, although if we have a good estimation of the recording frequency of the input device, then a naive estimator could be used to achieve reasonable results. For example, the GDS gesture samples seems to have been acquired at a sampling rate close to 60 Hz, as suggested by the prediction errors achieved by the naive estimator operating at that frequency. Further, the NicIcon gestures were acquired with a Wacom Intuos2 tablet, which operates at a constant (synchronous) sampling rate of 100 Hz,[10] and we can see that, unsurprisingly, the naive estimator operating at that frequency achieves the lowest prediction errors. Still, DITTO achieved very small errors as well, in the order of tens of milliseconds, and we have to remark the fact that DITTO is device-agnostic and can handle asynchronous stroke data by design, without being aware of the operational device's sampling rate.

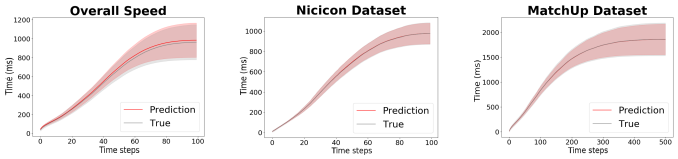
Figure 7 reports the results on the NicIcon (multistroke gestures) and MatchUp (multitouch-multistroke gestures) datasets. For completeness, we also report the overall production times from the GDS dataset. As can be observed in these plots, DITTO achieves notable accuracy, considering again that it is a device-agnostic data-driven approach. At every timestep, the cumulated difference between predicted and ground-truth time offsets is very small. We can conclude that the estimations delivered by DITTO are on par with users' actual performance with stroke gesture input.

[9] Computers periodically poll the input devices for new data [60], thereby producing artificial velocity jitter.

[10] See https://www.neuroscript.net/tablets/

	Dataset	Production time MAE	RMSE	r	Velocity profile align. MAE	RMSE	r
DITTO	GDS	80	118	0.953	25	52	0.912
	NicIcon	12	16	0.998	8	18	0.957
	MatchUp	212	330	0.935	7	269	0.965
N.30	GDS	1177	1193	0.924	287	370	0.492
	NicIcon	2537	2630	0.999	232	241	0.827
	MatchUp	6851	7186	0.990	539	616	0.810
N.60	GDS	110	134	0.924	180	267	0.492
	NicIcon	723	749	0.999	132	138	0.898
	MatchUp	2563	2687	0.723	382	441	0.903
N.100	GDS	432	479	0.924	141	172	0.492
	NicIcon	4	3	0.999	0.36	0.37	0.910
	MatchUp	847	889	0.990	239	269	0.942

TABLE III: Experimental results over our datasets (test partition). Errors are expressed in milliseconds. All correlations are statistically significant at the $p < .001$ level.



(a) GDS (b) NicIcon (c) MatchUp

Fig. 7: Cumulative time distributions in the evaluated datasets (test partition).

Finally, some examples of velocity profiles estimations are reported for the NicIcon dataset in Figure 8 and for the MatchUp dataset in Figure 9. All velocity profiles were randomly selected from their respective test partitions, so the plots help to illustrate the capability of DITTO to estimate the articulation of unseen gestures. As can be observed, DITTO's predicted velocity profiles are often inline with ground-truth profiles, providing thus further evidence and highlighting the accuracy of our model to estimate temporal information using spatial information as sole input.

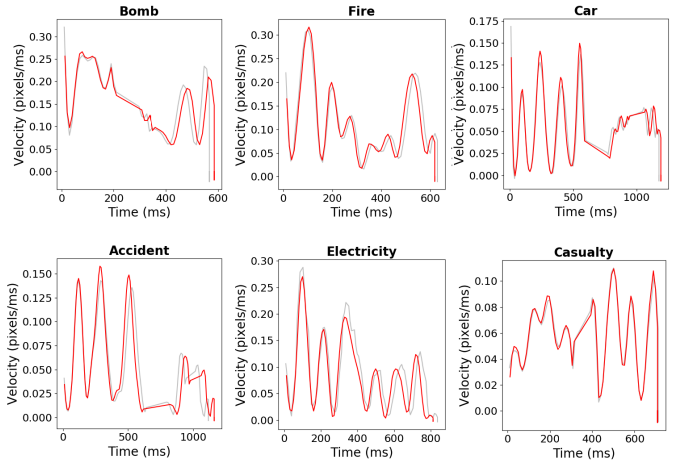


Fig. 8: Velocity profile estimations in the Nicicon dataset (test partition). Gestures examples picked at random.

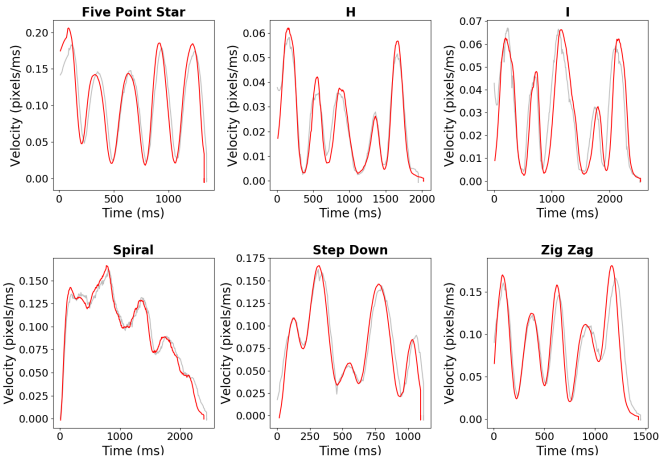


Fig. 9: Velocity profile estimations in the MatchUp dataset (test partition). Gestures examples picked at random.

VI. Limitations and Future Work

One limitation of computational models like DITTO is that they do not provide a closed-form solution nor cannot reason about the data they produce. As with any machine learning model, DITTO learns from examples and so these should be as diverse as possible, in order to generalize to new data.

Currently we have set a maximun model capacity to 100 timesteps, which has proved to be effective in handling the analyzed datasets. Nevertheless, this number should be increased to handle much longer stroke sequences, such as in signatures or mid-air pointing traces. In the future we plan to explore other handwriting movements like these just mentioned.

VII. Conclusion

DITTO is a comprehensive sequence-to-sequence deep learning model that delivers a remarkably accurate estimation of the velocity profile of *any* stroke gesture, including unistrokes, multistrokes, and multitouch gestures. DITTO contributes to advancing our capacity as a community to model, analyze, and understand users' stroke gesture articulation. From this perspective, DITTO takes a major step by moving away from *overall* estimations and instead delivering *moment-by-moment* estimations of production times. DITTO is available to practitioners, developers, and researchers as a web application and associated service at https://luis.leiva.name/ditto/.

This work contributes to our understanding on gesture input articulation, as well as our ability to make accurate estimations of user performance. We would like to restate that efforts like DITTO are essential to help the community shape, consolidate, and advance its knowledge.

Acknowledgments

We acknowledge the computational resources provided by the Aalto Science-IT project and funding from the Helsinki Institute for Information Technology (HIIT) via the Finnish Center for Artificial Intelligence (FCAI).

REFERENCES

[1] S. Zhai, P. O. Kristensson, C. Appert, T. H. Anderson, and X. Cao, "Foundational issues in touch-surface stroke gesture design — an integrative review," in *Foundations and Trends in Human-Computer Interaction*. Now publishers, 2012, vol. 5, no. 2.

[2] L. A. Leiva, V. Alabau, V. Romero, A. H. Toselli, and E. Vidal, "Context-aware gestures for mixed-initiative text editing UIs," *Interact. Comput.*, vol. 27, no. 1, 2014.

[3] Y. Li, "Gesture Search: A tool for fast mobile data access," in *Proc. UIST*, 2010.

[4] C. Zhang, N. Jiang, and F. Tian, "Accessing mobile apps with user defined gesture shortcuts: An exploratory study," in *Proc. ISS*, 2016.

[5] WB Games and Niantic, "Harry potter: Wizards unite," Available at https://wizardsunitehub.info/, 2019.

[6] L. A. Leiva, D. Martín-Albo, and R.-D. Vatavu, "Synthesizing stroke gestures across user populations: A case for users with visual impairments," in *Proc. CHI*, 2017.

[7] S. K. Kane, J. P. Bigham, and J. O. Wobbrock, "Slide rule: Making mobile touch screens accessible to blind people using multi-touch interaction techniques," in *Proc. ASSETS*, 2008.

[8] Y. Yang, G. D. Clark, J. Lindqvist, and A. Oulasvirta, "Free-form gesture authentication in the wild," in *Proc. CHI*, 2016.

[9] L. A. Leiva, M. Diaz, M. A. Ferrer, and R. Plamondon, "Human or Machine? it is not what you write, but how you write it," in *Proc. ICPR*, 2020.

[10] P. O. Kristensson and S. Zhai, "Command strokes with and without preview: Using pen gestures on keyboard for command selection," in *Proc. CHI*, 2007.

[11] J. Allan C. Long, J. A. Landay, and L. A. Rowe, "Implications for a gesture design tool," in *Proc. CHI*, 1999.

[12] J. O. Wobbrock, M. R. Morris, and A. D. Wilson, "User-defined gestures for surface computing," in *Proc. CHI*, 2009.

[13] R.-D. Vatavu, D. Vogel, G. Casiez, and L. Grisoni, "Estimating the perceived difficulty of pen gestures," in *Proc. INTERACT*, 2011.

[14] M. A. Nacenta, Y. Kamber, Y. Qiang, and P. O. Kristensson, "Memorability of pre-designed and user-defined gesture sets," in *Proc. CHI*, 2013.

[15] J. Nielsen and T. K. Landauer, "A mathematical model of the finding of usability problems," in *Proc. INTERCHI*, 1993.

[16] X. Cao and S. Zhai, "Modeling human performance of pen stroke gestures," in *Proc. CHI*, 2007.

[17] P. Isokoski, "Model for unistroke writing time," in *Proc. CHI*, 2001.

[18] L. A. Leiva, D. Martín-Albo, R. Plamondon, and R.-D. Vatavu, "KeyTime: Super-accurate prediction of stroke gesture production times," in *Proc. CHI*, 2018.

[19] L. A. Leiva, R.-D. Vatavu, D. Martín-Albo, and R. Plamondon, "Omnis Praedictio: Estimating the full spectrum of human performance with stroke gestures," *Int. J. Hum. Comput. Stud.*, vol. 142, 2020.

[20] R. Woodworth, "The accuracy of voluntary movement," *Psychol. Rev.*, vol. 3, 1899.

[21] J. Smeets, L. O. Wijdenes, and E. Brenner, "Movement adjustments have short latencies because there is no need to detect anything," *Motor Control*, vol. 20, 2016.

[22] L. A. Leiva, D. Martín-Albo, and R.-D. Vatavu, "GATO: Predicting human performance with multistroke and multitouch gesture input," in *Proc. MobileHCI*, 2018.

[23] R. Plamondon, "A kinematic theory of rapid human movements. Part I: Movement representation and control," *Biol. Cybern.*, vol. 72, no. 4, 1995.

[24] J. Resig, B. Bibeault, and J. Maras, *Secrets of the JavaScript Ninja*, 2nd ed. Manning Publications, 2016.

[25] E. M. Taranta II, M. Maghoumi, C. R. Pittman, and J. J. LaViola Jr., "A rapid prototyping approach to synthetic data generation for improved 2D gesture recognition," in *Proc. UIST*, 2016.

[26] L. Anthony and J. O. Wobbrock, "A lightweight multistroke recognizer for user interface prototypes," in *Proc. GI*, 2010.

[27] D. Ha and D. Eck, "A neural representation of sketch drawings," in *Proc. ICLR*, 2018.

[28] R. Plamondon and M. Djioua, "A multi-level representation paradigm for handwriting stroke generation," *Hum. Mov. Sci.*, vol. 25, no. 4–5, 2006.

[29] D. Martín-Albo, L. A. Leiva, J. Huang, and R. Plamondon, "Strokes of insight: User intent detection and kinematic compression of mouse cursor trails," *Inform. Process. Manag.*, vol. 56, no. 6, 2016.

[30] R. Plamondon, "A kinematic theory of rapid human movements. Part II: Movement time and control," *Biol. Cybern.*, vol. 72, no. 4, 1995.

[31] L. A. Leiva, D. Martín-Albo, and R. Plamondon, "Gestures à Go Go: Authoring synthetic human-like stroke gestures using the kinematic theory of rapid movements," *ACM Trans. Intell. Syst. Technol.*, vol. 7, no. 2, 2016.

[32] P. M. Fitts, "The information capacity of the human motor system in controlling the amplitude of movement," *J. Exp. Psychol.*, vol. 47, no. 6, 1954.

[33] J. Müller, A. Oulasvirta, and R. Murray-Smith, "Control theoretic models of pointing," *ACM Trans. Comput. Hum. Interact.*, vol. 24, no. 4, 2017.

[34] D. Mauney, "What gestures do people actually use?" in *Proc. D4M*, 2010.

[35] B. Poppinga, A. S. Shirazi, N. Henze, W. Heuten, and S. Boll, "Understanding shortcut gestures on mobile touch devices," in *Proc. MobileHCI*, 2014.

[36] G. Bailly, J. Müller, and E. Lecolinet, "Design and evaluation of finger-count interaction: Combining multitouch gestures and menus," *Int. J. Hum. Comput. Stud.*, vol. 70, no. 10, 2012.

[37] C. Harrison, J. Schwarz, and S. E. Hudson, "TapSense: Enhancing finger interaction on touch surfaces," in *Proc. UIST*, 2011.

[38] F. Matulic, D. Vogel, and R. Dachselt, "Hand contact shape recognition for posture-based tabletop widgets and interaction," in *Proc. ISS*, 2017.

[39] Y. Rekik, R.-D. Vatavu, and L. Grisoni, "Understanding users' perceived difficulty of multi-touch gesture articulation," in *Proc. ICMI*, 2014.

[40] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock, "Relative accuracy measures for stroke gestures," in *Proc. ICMI*, 2013.

[41] J. Allan C. Long, J. A. Landay, L. A. Rowe, and J. Michiels, "Visual similarity of pen gestures," in *Proc. CHI*, 2000.

[42] P. Viviani and C. Terzuolo, "Trajectory determines movement dynamics," *Neuroscience*, vol. 7, no. 2, 1982.

[43] T. Flash and N. Hogan, "The coordination of arm movements: an experimentally confirmed mathematical model," *J. Neurosci.*, vol. 5, no. 7, 1985.

[44] R. Plamondon, A. M. Alimi, P. Yergeau, and F. Leclerc, "Modelling velocity profiles of rapid movements: a comparative study," *Biol. Cybern.*, vol. 69, no. 1, 1993.

[45] L. A. Leiva, D. Martín-Albo, and R. Plamondon, "The kinematic theory produces human-like stroke gestures," *Interact. Comput.*, vol. 29, no. 4, 2017.

[46] M. Djioua and R. Plamondon, "Studying the variability of handwriting patterns using the Kinematic Theory," *Hum. Mov. Sci.*, vol. 28, no. 5, 2009.

[47] ——, "The limit profile of a rapid movement velocity," *Hum. Mov. Sci.*, vol. 29, no. 1, 2010.

[48] U. R. Muhammad, Y. Yang, Y.-Z. Song, T. Xiang, and T. M. Hospedales, "Learning deep sketch abstraction," in *Proc. CVPR*, 2018.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.

[50] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. ICLR*, 2015.

[51] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," in *Proc. EMNLP*, 2015.

[52] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proc. ICML*, 2015.

[53] J. Cheng, L. Dong, and M. Lapata, "Long short-term memory-networks for machine reading," in *Proc. EMNLP*, 2016.

[54] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes," in *Proc. UIST*, 2007.

[55] D. Willems, R. Niels, M. van Gerven, and L. Vuurpijl, "Iconic and multi-stroke gesture recognition," *Pattern Recognit.*, vol. 42, no. 12, 2009.

[56] Y. Rekik, R.-D. Vatavu, and L. Grisoni, "Match-up & conquer: A two-step technique for recognizing unconstrained bimanual and multi-finger touch input," in *Proc. AVI*, 2014.

[57] S. K. Kane, J. O. Wobbrock, and R. E. Ladner, "Usable gestures for blind people: Understanding preference and performance," in *Proc. CHI*, 2011.

[58] D. E. Meyer, J. E. Keith-Smith, S. Kornblum, R. A. Abrams, and C. E. Wright, "Speed-accuracy tradeoffs in aimed movements: Toward a theory of rapid voluntary action," *Atten. and Perform.*, vol. 13, no. 23, 1990.

[59] R. Plamondon and A. M. Alimi, "Speed/accuracy tradeoffs in target directed movements," *Behav. Brain Sci.*, vol. 20, no. 2, 1997.

[60] D. Anderson, *USB System Architecture*. MindShare, Inc., 1997.