A Rapid Prototyping Approach to Synthetic Data Generation For Improved 2D Gesture Recognition

Eugene M. Taranta II
University of Central Florida
Orlando, FL 32816, USA
etaranta@gmail.com

Mehran Maghoumi
University of Central Florida
Orlando, FL 32816, USA
mehran@cs.ucf.edu

Corey Pittman
University of Central Florida
Orlando, FL 32816, USA
cpittman@knights.ucf.edu

Joseph J. LaViola Jr.
University of Central Florida
Orlando, FL 32816, USA
jjl@eecs.ucf.edu

ABSTRACT

Training gesture recognizers with synthetic data generated from real gestures is a well known and powerful technique that can significantly improve recognition accuracy. In this paper we introduce a novel technique called *gesture path stochastic resampling* (GPSR) that is computationally efficient, has minimal coding overhead, and yet despite its simplicity is able to achieve higher accuracy than competitive, state-of-the-art approaches. GPSR generates synthetic samples by lengthening and shortening gesture subpaths within a given sample to produce realistic variations of the input via a process of nonuniform resampling. As such, GPSR is an appropriate rapid prototyping technique where ease of use, understandability, and efficiency are key. Further, through an extensive evaluation, we show that accuracy significantly improves when gesture recognizers are trained with GPSR synthetic samples. In some cases, mean recognition errors are reduced by more than 70%, and in most cases, GPSR outperforms two other evaluated state-of-the-art methods.

ACM Classification Keywords

H.5.2 Information interfaces and presentation: User interfaces, input devices and strategies; I.5.5 Pattern recognition: Implementation, interactive systems

Author Keywords

Synthetic gestures; stochastic resampling; rapid prototyping; gesture recognition

INTRODUCTION

Iterative design is a crucial tool for HCI research in user interfaces (UI), and techniques that aid in the rapid development and exploration of such interfaces are consequently of high value. In this context, high quality, lightweight, easy to understand and fast to implement algorithms that address common UI demands are often preferred over alternative, possibly more robust industrial strength solutions. One example lies in gesture recognition. Though UI research often involves gestural interfaces, in the past, researchers required advanced knowledge of mathematics and machine learning techniques to implement robust gesture recognition. Wobbrock *et al.* [50] began to address this problem when they introduced the elegant, well received $1 recognizer, which in turn spawned several similar recognizers [1, 2, 26, 45]. These so-called $-family recognizers, as well as other related rapid prototyping

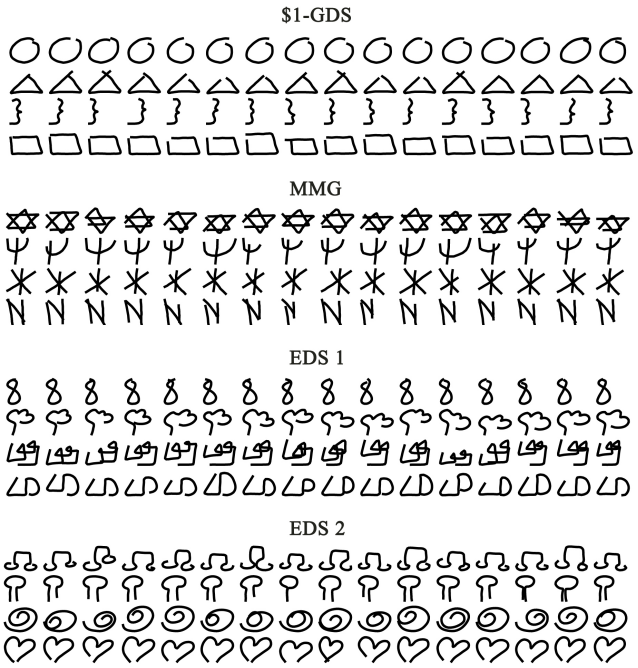


Figure 1: Example synthetic gestures from $1-GDS [50], MMG [2], EDS 1 [47], and EDS 2 [47]. The first column of each row is the original sample from which the remaining synthetic gestures are derived. All gestures are smoothed.

UIST 2016, October 16-19, 2016, Tokyo, Japan
 ISBN 978-1-4503-4189-9/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2984511.2984525

alternatives, *e.g.* [21, 22], rely on nearest neighbor template matching of candidate gestures to stored templates, and indeed accuracy improves with increased training samples. However, as Li [26] discusses, a user is unlikely to provide more than one or two samples per gesture under usual conditions, which limits the potential performance of these recognizers.

Accuracy as a function of training set size is not only limited to the aforementioned techniques, as it has been shown that many other recognizers also benefit by having access to larger datasets [6, 37, 48]. To overcome the limitations imposed by a small training dataset, researchers have utilized synthetic data generation (SDG) methods [12] in order to synthesize new samples from those already available. While SDG has proven to be useful, current techniques are unsuitable for rapid prototyping by the average developer as they are time consuming to implement, require advanced knowledge to understand and debug, or are too slow to use in realtime. The latter is especially important in UIs where users can define new gestures on demand. We address these issues by introducing a novel rapid prototyping appropriate SDG method called *gesture path stochastic resampling* (GPSR) that is accessible to the same audience as those targeted by such recognizers. Specifically, our method utilizes nonuniform resampling of gesture paths and subsequent normalization of the in-between point vectors [22] to produce realistic synthetic samples (see Figure 1).

Although our method is deceptively simple, we show through an extensive evaluation that GPSR improves the recognition accuracy of several rapid prototyping recognizers and two parametric recognizers. Further, we compare our approach to two alternative SDG methods and find that GPSR outperforms both in almost every instance.

RELATED WORK

Rapid Prototyping Gesture Recognition

One popular subset of rapid prototyping gesture recognizers is the $-family [1, 2, 26, 45, 50]. The original $1 recognizer was pioneered nearly a decade ago by Wobbrock *et al.* [50], and due to its simplicity and intuitiveness, their recognizer and its successors have enjoyed immense success. The importance of these recognizers stems from the fact that they enable HCI researchers to focus on UI design rather than fret over advanced machine learning concepts, or libraries and toolkits that may not be available for their platform. At their core, $-recognizers utilize 1-nearest neighbor pattern matching [11], where training samples are stored as *templates* and a *candidate* gesture is measured against each template. The gesture class of the best matching template is then assigned to the candidate gesture or alternatively, an *N*-best list can also be provided.

As demonstrated repeatedly in various $-family recognizer evaluations, accuracy continues to improve as the number of samples per gesture increases, and while writer-dependent recognition is already fairly high, mixed-writer and writer-independent gesture recognition can still be improved, which is one of the primary objectives of our synthetic data generation method. By writer-dependent, we mean that the recognizer is trained and used by the same person, whereas with writer-independent, the recognizer is used by writers that are different from those who trained it. Mixed-writer falls in between: training samples come non-uniformly from multiple writers who use the recognizer along with those who may not have contributed training samples.

Synthetic Data Generation

The paucity of correctly labeled training data is a common problem in the field of pattern recognition [31]. Crowdsourcing can help alleviate this issue, although with potentially high cost. Another alternative is to synthesize new data from that which is already available. This process of synthetic data generation (SDG) has been used successfully in many fields, including human pose recognition [39], digital forensics [16, 28], information retrieval [17, 36] and handwriting recognition of ancient texts [15], as well as in generating [4, 13] and collecting [29] large data sets.

Early examples of SDG in gesture and handwriting recognition include works by Ha and Bunke [18] and thereafter, a number of researchers have also attacked this problem as reported in Elanwar's survey [12]. One key difference in their approaches relates to whether the data is digital ink or images of symbols. Methods that work on ink can broadly be divided into two categories: those that replicate feature distributions of the population (such as pen-lifts and velocity) and those that apply perturbations to the given data. The former approach requires at least a small set of data to begin with, which may not exist and is why we take a perturbation approach. A third option, however, involves the interactive design of procedurally generated gestures, such as that provided by Gesture Script [27], though we require a general method that does not require user intervention.

Concerning image based techniques, Helmers *et al.* [19] proposed an approach to produce synthetic text in which isolated samples of an individual's handwriting are concatenated together to form synthetic handwritten text. Ha and Bunke [18] along with Cano et al. [6] leveraged various image transformation operations (*e.g.*, erode, dilate, *etc.*) to produce variations of image samples of handwritten characters. Varga *et al.* [41, 42] used randomly generated geometrical transformations such as scaling and shearing lines of handwritten text to produce new synthetic lines. Such transformations have also been applied towards the creation of synthetic CAPTCHAs [40]. Lee *et al.* [24] generated synthetic Korean characters using Beta distribution curves while Varga *et al.* [43] used Bezier splines to generate handwritten English text. Caramiaux *et al.* [8] generated synthetic samples drawn from Viviani's curve to produce controlled data to evaluate their proposed adaptive gesture recognizer. Dinges *et al.* [10] used active shape models which rely on the linear combination of the eigenvectors of the covariance matrix built for each class of shapes to create synthetic handwritten text.

Perturbation models such as Perlin noise [32] and the Sigma-Lognormal [35] model have been proven to be strong contenders for SDG. Since we will be using these methods in our evaluations, a more in depth description of each follows.

Perlin Noise. Davila *et al.* [9] used Perlin noise maps [32], a well-known technique in computer graphics for producing

natural looking synthetic textures, to generate synthetic math symbols. Each Perlin map consists of a grid of points. The number of points defines the resolution of the map. A gradient direction is assigned to each point and random noise is generated based on the direction of the gradient. Synthetic samples are created by coinciding the gesture's stroke points on the noise map and moving each stroke's points along the grid's gradient direction. A discriminating factor of Perlin noise compared to our proposed approach is that our method modifies the gesture's path whereas Perlin noise modifies individual sample points.

Sigma-Lognormal Model. Although numerous models have been proposed to describe human movement, the kinematic theory of rapid human movement [34] and its associated Sigma-Lognormal ($\Sigma\Lambda$) model [35] has been shown to have superior performance in modeling human movement, and has been successfully applied to a large range of applications [30]. The $\Sigma\Lambda$ equations (including Equations 1 and 2) attempt to model the complex interactions of a neuromuscular network executing an action plan. That is, a stroke is described by a set of overlapping primitives connecting a series of virtual targets [25], where each primitive is described by a lognormal equation. Formally, the velocity profile of a trajectory is given by:

$$\vec{v}(t) = \sum_{i=1}^{N} \vec{v}_i(t) = \sum_{i=1}^{N} D_i \begin{bmatrix} \cos\phi_i(t) \\ \sin\phi_i(t) \end{bmatrix} \Lambda(t; t_0, \mu_i, \sigma_i^2), \quad (1)$$

which is the vectorial summation of N primitives. Each primitive is a four parameter lognormal function scaled by D_i and time shifted by t_i, where μ_i represents a neuromuscular time delay and σ, the response time. The angular position of a primitive is also given by:

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

where θ_{s_i} and θ_{e_i} are starting and ending angles of the ith primitive. A parameter extractor such as that described by Martín-Albo *et al.* [30] is used to find the individual primitives and their associated model parameters for a given stroke. Perturbations to model parameters create realistic variations in the trajectory and can be used to create synthetic gestures, such as for whiteboard note generation [14]. Closer to our interests, Leiva *et al.* [25] recently introduced the Gestures à Go Go (G3) web service to provide synthetic samples from real data using kinematic theory. They were also able to show that $-family recognizers trained with only synthetically generated samples could perform as well as recognizers trained with only human samples.

Similar to perturbations on $\Sigma\Lambda$ model parameters that change the velocity profile by modifying delay and response times, we also think of GPSR as a way of changing the timing profile of a stroke, so that gesture subpaths are lengthened and shortened as a result of our resampling strategy. However, our approach uses the input directly, without requiring the overhead of model parameter extraction.

GESTURE PATH STOCHASTIC RESAMPLING

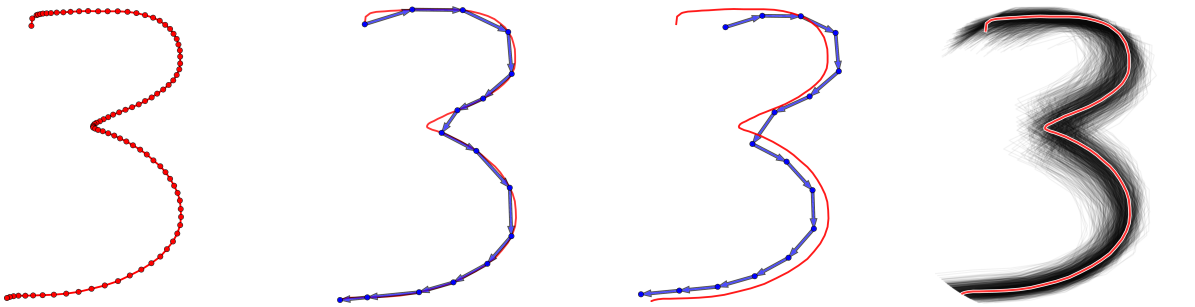
The design of our synthetic data generation method is motivated by several crucial objectives. Foremost, as a rapid prototyping technique, the approach should be easily accessible to the average developer: understood with little effort and without expert knowledge, and consequently be fast to implement as well as easy to debug. Yet even with its reduced complexity, improvements in recognition accuracy must remain competitive as compared to other state-of-the-art SDG methods. For the sake of adoptability, the approach ought to utilize only spatial coordinates given that timing and pressure information may be unreliable or even unavailable, but more importantly, artificial gestures should be synthesized with minimal computational overhead. Further, synthetic gestures should have a realistic appearance, not only for display purposes, but because severely deformed synthetic samples may lead to poor recognizer performance. Finally, the method should fit orthogonally and be complementary to already available gesture recognition techniques so that existing machinery might be leveraged without significant modification if desired. In our view, *gesture path stochastic resampling* (GPSR) satisfies these aforementioned criteria. We now describe GPSR, for which the pseudocode listing can be found in Appendix A[1].

Similar in nature to the $\Sigma\Lambda$ model [25], one may think of a gesture as being described by a canonical velocity profile that is based on an action plan defining the gesture's shape. To recreate the gesture, a writer must execute the plan with a certain level of fidelity. Minor variability due to perturbations in the velocity profile will yield recognizable, yet uniquely different shapes, and so long as the writer's variation is reasonable, global characteristics of the shape will remain intact. However, rather than extract complex model parameters from a given sample and perturb the parameters post hoc, we can simulate reasonable perturbations on the sample directly.

As a first step, we consider perturbations that result in the lengthening or shortening of gesture subpaths. To simulate such deviations from the action plan, consider resampling a stroke to n points so that the path length is *nonuniform* between points. Call any vector between two contiguous points along the gesture path an in-between point direction vector [22]. Now normalize all in-between point vectors to unit length and observe that for any two arbitrary vectors, the ratio of their lengths are altered as a result of the transformation. This approach is analogous to modifying the gesture path length between each pair of contiguous points, which is illustrated in Figure 3—some in-between point vectors will have been shortened while others will have been lengthened. It is easy to see that each nonuniform resampling and subsequent normalization will lead to a different shape, and with repeated application, a distribution of synthetic gestures can be generated from a single sample.

Formally, let $\xi_1 = 0$ and $\xi_2, \ldots, \xi_n$ be a random sample of size $n-1$ from a uniform $\mathcal{U}(1, 1 + \sqrt{12 * \sigma^2})$ population. Define

[1]Further, an interactive demo of GPSR is available at http://www.eecs.ucf.edu/isuelab/demo/stochastic-resampling



(a) Original Sample **(b)** Stochastically Resampled **(c)** Post Normalization **(d)** Synthetic Samples Distribution

Figure 2: Illustration of the gesture path stochastic resampling process.

an ordered list of gesture path ratios using the random sample:

$$r = \left(r_i = \frac{\sum_{j=1}^{i} \xi_j}{\sum \xi} \;\middle|\; i = 1 \ldots n \right), \tag{3}$$

so that $0 = r_1 < r_i < r_{i+1} < r_n = 1$. Further, a stroke is defined as an ordered list of 2D points $p = (p_i = (x_i, y_i) \mid i = 1 \ldots n)$, $\mathcal{L}$ is the arc-length of the gesture path through all points from p_1 to p_n, and $\mathcal{L}(p_i)$ is the arc-length distance to point p_i [46]. Similarly, we denote $\mathcal{L}^{-1}(d)$ as the inverse arc-length function that returns the point p_x at distance d along the gesture path. Now define an ordered list of stochastic points using the ratios as follows:

$$q = \left(q_i = \mathcal{L}^{-1}\left(r_i \mathcal{L}\right) \;\middle|\; i = 1 \ldots n \right). \tag{4}$$

The in-between point vectors derived from the stochastic points are $v = (v_i = (q_{i+1} - q_i) \mid i = 1 \ldots n - 1)$, from which a synthetic stroke is generated:

$$p' = \left(p'_i = p'_{i-1} + \frac{v_{i-1}}{\| v_{i-1} \|} \;\middle|\; i = 2 \ldots n \right), \tag{5}$$

where $p'_1 = (0, 0)$. The synthetic stroke p' can then be scaled, translated, rotated, and smoothed as desired.

We chose to use a uniform random distribution after finding inconsequential differences between the uniform, normal, exponential, and beta distributions, which was contrary to the poor performing log-normal distribution. The lower bound of the uniform distribution was set to 1 only to avoid any probability of drawing 0 and the upper bound is a function of variance so that the spread of the distribution can optionally be tuned.

Removals

Another type of variation occurs when a writer skips over some detail of the action plan, such as when a gesture stroke is not fully articulated or when a corner is cut. To simulate this in a general way we introduce the removal count x that, when specified, indicates how many points from the stochastic stroke q are randomly removed before generating the synthetic stroke p'. When removals are used, n should be adjusted to account for this reduction. Therefore, from hereon and for clarity, n refers to the length of the final synthetic stroke after the removal of x points, and implicitly the length of ξ, r, and q become $n + x$.

Multistroke Support

So far, we have described how to create single stroke synthetic gestures. With only a few modifications, gesture path stochastic resampling can also work with multistroke gestures. In a preprocessing step, first randomly permute the stroke set and randomly reverse a subset of the these strokes before combining them together into a single ordered list of points[2] p. In addition to 2D coordinate data, each point also now possesses an integer stroke ID. Stochastically resample p as before, while also interpolating the stroke ID as one does with 2D coordinates. Last, generate the synthetic sample, but break apart the unistroke p' by discarding "over the air" points whose stroke IDs are between integer values. This results in a synthetic multistroke gesture, and example results are shown in the second four rows of Figure 1.

PARAMETER SELECTION

Gesture path stochastic resampling requires the selection of three parameters: variance σ^2, removal count x, and resampling count n. With respect to variance σ^2, we found in early testing that this parameter had little influence on recognizer accuracy—any variance setting was sufficient to achieve good results. Therefore, we settled for $\sigma^2 = 0.25$ and held the parameter constant for the remainder of our analysis. Removal count x, on the other hand, had a noticeable impact on synthetic gesture quality. At $x = 2$, recognition accuracy results were indeed improved, but as x increased, gesture quality rapidly deteriorated. For this reason, we also decided to hold the removal count constant at two, and focus on the one parameter having the most significant impact on accuracy and gesture quality, the resampling count n.

As illustrated in Figure 3, one can see that the selection of n significantly impacts synthetic gesture generation. With a low resolution such as with $n = 16$, the left curly brace [50] has reasonable variation between samples, but most triangle-chain [47] samples are unrecognizable. At $n = 64$ there is practically no variation between left curly brace samples, though triangle-chain samples are now improved and have a healthy variation. Finally, at $n = 256$, there appears to be almost no variation

[2]This approach was inspired by $N's [1] handling of multistroke gestures.

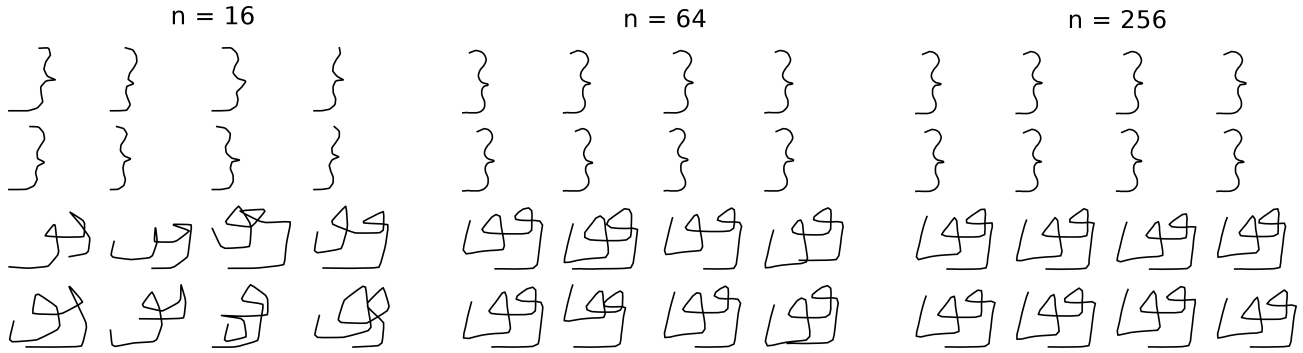


Figure 3: Effect of n on gestures of different complexity – left curly brace [50] (top) and triangle chain [47] (bottom)

for either gesture. These observations motivate us to find a function of n based on properties of the gesture that yield reasonable results—a function that can analyze an individual gesture sample and select an optimal resampling count n to apply.

In order to decide on an optimal resampling strategy, we consider the effect of n on various geometric relative accuracy measures[3] [46]. Namely, we use these measures to analyze how well a population of synthetic gesture samples matches a population of human generated samples. To start, Vatavu *et al.* [46] define the shape error (ShE) of a candidate gesture to be a measure of deviation from the gesture class's canonical form:

$$\mathrm{ShE}(p) = \frac{1}{n}\sum_{i=1}^{n} \|p_{\sigma(i)} - \overline{p}_i\|, \tag{6}$$

where $\overline{p}$ is the canonical point series and $\sigma(i)$ is a permuter of the candidate gesture points, which is used to find an optimal alignment. Shape variability (ShV) measures the standard deviation of the individual shape errors. Thereafter, bending error (BE) is a measure of average deviation from the absolute curvature of the gesture class's canonical form:

$$\mathrm{BE}(p) = \frac{1}{n}\sum_{i=1}^{n-2} \left|\theta_{\sigma(i)} - \overline{\theta}_i\right|, \tag{7}$$

where θ_i is the turning angle between in-between point vectors i and $i+1$. Bending variability (BV) again is the variance of these individual errors.

Using a similar evaluation methodology to that described shortly, we calculated all four relative accuracy measures for $n \in \{8, 16, 32, 64, 128\}$. We found that the ShE and ShV were significantly correlated ($r(549) = .91, p < .0001$), as were BE and BEV ($r(549) = .80, p < .0001$). Given this strong correlation between the error and variance metrics, we decided to focus on ShE and BE only. Specifically, we were interested in the mean ShE and mean BE for a population of gesture samples. To differentiate between the real population and a synthetic population we refer to these means as the real ShE (real BE) and syn ShE (syn BE), respectively. Further, the *mean ShE percentage error* is given in Equation 8, where each summand compares the syn and real ShE for one gesture, which gives us an error term for that gesture. Thus the equation averages these errors together, and our goal is to minimize this overall error:

$$\text{ShE \% Err} = \frac{100}{G}\sum_{i=1}^{G} \frac{\left|\text{real ShE} - \text{syn ShE}\right|}{\text{real ShE}}, \tag{8}$$

where G is the number of gestures under consideration, and the mean BE percentage error is defined similarly.

Name	Ref	Multistroke	Gestures	Participants
$1-GDS	[50]	No	16	10
EDS 1	[47]	No	18	14
EDS 2	[47]	No	20	11
LP Training	[22]	No	40	24
MMG	[2]	Yes	16	20

Table 1: Datasets used in evaluations.

To carry out our investigation, we utilized the five datasets described in Table 1. All of these datasets have been widely used in various studies except LP Training [22]; we included this dataset only to help avoid overfitting given that it replicates a number of gestures found in the other datasets.

For a given gesture, we first scaled and aligned[4] all samples within the population, after which we followed the procedure described by Vatavu *et al.* [46] to select the average template. That is, we uniformly resampled all samples to a fixed number of points, calculated the mean of the population, and selected the sample closest to the mean to serve as the canonical (centroid) sample of the gesture class. However, because all strokes were first aligned, we choose our permuter $\sigma(i)$ to be the identity permutation. With the canonical gesture in hand, we found

[3] In addition to geometric measures, Vatavu *et al.* [46] also define a set of kinematic accuracy measures that we do not include in our analysis given that GPSR does not attempt to synthesize time stamps. For similar reasons, the articulation accuracy measures were also not studied.

[4] Alignment was decided by finding the per sample stroke order permutation and direction that minimized the non-GSS $1 [50] distance across the population.

the optimal n that minimized the ShE percentage error[5]—for each value of n tested, we generated 512 synthetic samples from the centroid. With these samples, we then calculated the syn ShE for the population. Note that ShE was prioritized at this stage because the metric relies on point correspondences which directly, and sometimes indirectly, relate to the distance metric employed by various rapid prototyping gesture recognizers. Further, we also assumed that improvements in ShE error would lead to improvements in BE error.

Since our goal was to find a function of n based on properties of a given sample, we considered seven features derived from summaries provided by Blagojevic *et al.* [5]: curvature, absolute curvature, closedness, direction changes, two density variants, and stroke count. In addition to optimal n, we also extracted these features from the centroid sample. Using techniques from [23], we found that the optimal n had a logarithmic relationship with its predictors, which is why we first transformed the response. Thereafter, using stepwise linear regression, we identified those features that best explained the variability of the 110 gestures (see Table 1). In the end, we identified two features that together achieved high performance:

$$\text{closedness} = 1 - \frac{\| p_n - p_1 \|}{\text{diag}}, \tag{9}$$

and,

$$\text{density} = \frac{\mathcal{L}}{\text{diag}}, \tag{10}$$

where diag is the gesture's bounding box diagonal length. Absolute curvature actually accounted for negligibly higher variability over density, but since the feature may be unreliable due to jitter or wobbling and because destiny is a simpler approach, we favored a parsimonious solution.

Now with good features selected, we performed multiple linear regression to find an equation for optimal n. A significant regression equation was found ($R^2 = .59$, $F(2, 109) = 75.62$, $p < .0001$). The intercept ($p < .0001$), density ($p < .0007$) and closedness ($p < .0001$) parameters were significant, yielding the following equation:

$$n = \exp\{1.67 + 0.29\ \text{density} + 1.42\ \text{closedness}\} \tag{11}$$

Further, the cross validated coefficient of determination[6] ($Q^2 = .56$) was approximately equal to $R^2 = .59$, which is another indication of a good fit. Residuals were also confirmed to be statistically normal using a Shapiro-Wilks test ($W = .99$, $p = .37$).

We consider $R^2 = .59$ to be a good result because there is a great deal of variability between datasets. For instance, some selected differences in real ShE are the heart gesture at .061 and .142; rectangle at .045 and .054; five point star at .099 and .135; and triangle at .056 and .081. These differences are likely related to how the datasets were collected, including the device, instructions, and software used. By applying the optimal n equation to each of the 110 gesture centroids from Table 1, we find that the n values range from 16 to 69, and have a mean of 31 (SD=13.1).

[5]We set $n = 16$ as a lower bound, since lower values can result in poorly malformed gestures.

[6]$Q^2 = 1 - \frac{PRESS}{TSS}$.

n	Rec % Error	ShE % Err	BE % Err
Optimal	3.47 (3.81)	26.06 (21.24)	21.15 (15.23)
8	—	1137.72 (9538.78)	95.86 (85.58)
16	4.68 (5.05)	80.22 (67.82)	33.37 (37.34)
32	3.80 (3.98)	38.85 (25.14)	25.27 (17.28)
64	4.20 (4.18)	44.44 (24.04)	33.63 (14.18)

Table 2: Mean gesture recognition percentage error (and SD) over all template matching recognizers for one and two training samples per gesture, from which 64 gestures are synthesized per training sample (see Evaluation) on $1-GDS [50], MMG [2], EDS 1 [47], and EDS 2 [47], as well as the ShE and BE percentage errors. Note that the optimal n value ranges from 16 to 69 depending on the gesture's centroid.

Evaluation of Optimal n

To understand if optimal n (Equation 11) is effective at simulating a realistic distribution, we calculated the relative metrics over varying $n \in \{8, 16, 32, 64\}$ and optimal n. Results can be found in Table 2. Overall, optimal n had the lowest ShE percentage error ($M = 26\%, SD = 21\%$) and, as compared to its runner up $n = 32$ ($M = 39\%, SD = 25\%$), the result was statistically significant based on a Wilcoxon signed-ranks test ($p < .0001$). Similarly, optimal n also had the lowest BE percentage error ($M = 21\%$, $SD = 15\%$) that again, compared to $n = 32$ ($M = 25\%, SD = 17\%$), was significant ($p < .004$).

Further, to ensure our approach did not degrade recognizer performance (for instance, by reducing or over inflating population variance as compared to other static values of n), we also evaluated recognition accuracy for each level. Optimal n achieved the lowest mean error 3.47% ($SD = 3.81$) for one and two training samples expanded into 64 synthetic gestures per sample, using unistroke recognizers on unistroke datasets and multistroke recognizers on MMG [1]. The second closest was $n = 32$, having a mean error of 3.80% ($SD = 3.98$); although, the difference between levels was not significant ($F(3, 220) = .4093, n.s.$).

As a result of this analysis, we determined that optimal n (which is unique per sample) is able to synthesize a gesture population more precisely than any static value of n. This is highly desirable, because for unknown datasets, we need to reduce the probability of generating unrealistic samples that cross the decision boundary between gestures classes, since malformed gestures have the potential to ruin recognizer performance. On the other hand, we also do not want to restrict synthetic data generation so much so that there is insufficient variation to be of any use to a gesture recognizer, and therefore optimal n is utilized to strike a balance between these objectives. For this reason, optimal n was used throughout the remainder of our evaluations.

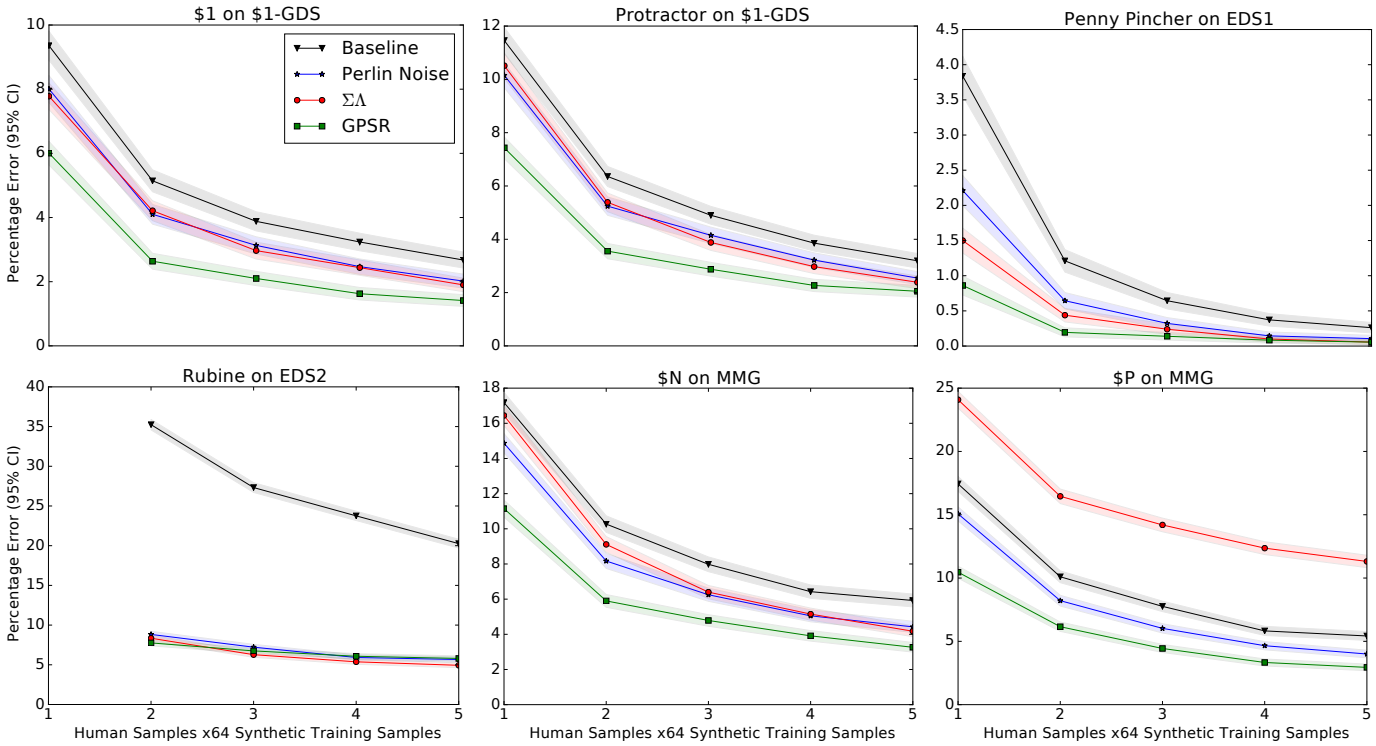


Figure 4: Accuracy results for various configurations. In each graph, the horizontal axis is the number of human samples per gesture used for training, where $S = 64$ synthetic samples were created per real sample. Results were randomly selected so as not to highlight any one particular recognizer and dataset. However, across the board, one will notice that mean recognition errors are significantly reduced using GPSR with gestures being stochastically resampled to optimal n.

	$ 1-GDS [50]				**EDS 1 [47]**				**EDS 2 [47]**				**MMG [1]**			
	Pincher, $S = 8$		Pincher, $S = 64$		\$1, $S = 8$		\$1, $S = 64$		Pincher, $S = 8$		Pincher, $S = 64$		\$P, $S = 8$		\$P, $S = 64$	
	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%	M% (SD)	±%
None	8.59 (6.56)	—	8.59 (6.56)	—	2.16 (3.29)	—	2.16 (3.29)	—	2.13 (3.09)	—	2.13 (3.09)	—	17.4 (9.20)	—	17.4 (9.20)	—
GPSR	6.05 (5.61)	30	4.76 (5.08)	45	1.11 (2.39)	49	0.46 (1.60)	79	1.11 (2.33)	48	0.59 (1.64)	72	14.3 (8.21)	18	10.5 (7.54)	40
ΣΛ	7.32 (5.98)	15	6.41 (5.71)	25	1.92 (3.21)	11	1.38 (2.78)	36	1.37 (2.56)	36	0.92 (2.10)	57	26.6 (9.96)	-49	24.1 (9.59)	-38
PN	7.34 (6.27)	15	6.89 (6.02)	20	1.67 (2.96)	22	1.31 (2.64)	39	1.58 (2.67)	26	1.24 (2.39)	42	15.0 (8.79)	09	15.1 (8.79)	14

Table 3: Recognizer percentage error rates (SD) and their associated percentage error rate reductions from baseline (without SDG) given one real training sample per gesture ($T = 1$), comparing gesture path stochastic resampling (GPSR), ΣΛ and Perlin noise (PN) for $S = 8$ synthetic samples per real gesture and $S = 64$ across four datasets. The gesture recognizer shown is the one that achieved the lowest error rate for the given dataset and S. In all cases, GPSR achieves the best performance.

EVALUATION: RECOGNITION ACCURACY

Our evaluation strategy was similar to that used by Wobbrock *et al.* [50]. One key difference in our protocol is that instead of performing writer-dependent testing, we use a mixed-writer protocol. Given a particular dataset comprised of G gestures, all samples from all participants are pooled together. Then for each gesture class, T real samples are selected from the pool and 1 remaining, independent sample is selected for testing. If synthetic data generation is being used, S synthetic samples per real sample are generated for training, which means the recognizer is trained with $G * T * S$ samples. Once the recognizer is trained, each sample in the validation test set is classified, which results in G recognition tests. Note that the validation set comprised only real samples, not synthetic samples. These G results are subsequently combined into a single average error rate. This procedure is repeated 1000 times and the result from each iteration is further averaged into a single overall recognition error rate.

In this evaluation, the levels of T are increased from 1 to 5, and the levels of S are $\{8, 16, 32, 64\}$. We note that it may seem biased to test a recognizer trained with T real samples to a recognizer trained with $T * S$ synthetic samples, but it is important to note that in both cases, exactly the same number of real samples are provided. The key difference is in what advantage SDG can provide with respect to the recognition error rate. One reason why we train with more synthetic samples is because there are a lot of redundancies in the synthetic sample distribution, where some synthetic samples provide no additional value, which is discussed further in the limitations section.

All datasets appearing in Table 1 except LP Training were utilized in our recognition accuracy tests, as these are familiar datasets commonly appearing in the literature. We consider the problem described in the previous section of finding an optimal n function that produces realistic distributions to be different from that of finding a suitable n that results in high recognition accuracy. However, it is important to note that since we leverage the same datasets, it is possible that optimal n is overfit in these tests. Static values $n = 16, 32$, and 64 were also evaluated and as before. We found that $n = 32$ achieves the lowest average error rate of all static values of n, and their is no statistical difference between optimal n and $n = 32$. Thus, we do not believe there was overfitting in this case. For this reason, we report only optimal n in this section. Further, since we are primarily concerned with recognition accuracy when only one or two samples are given per gesture, we restrict our formal analysis to $T \in [1, 2]$, and the remaining levels are only used to show that the trends continue as T increases, as is shown in Figure 4. As a last note, since error rates tend to be skewed towards zero and violate ANOVA assumptions, we used the Aligned Rank Transform (ART) method [49] to carry out our analysis.

SDG Methods

In addition to running all recognizers without synthetic data generation (the baseline), three SDG methods were evaluated: GPSR, Perlin noise and ΣΛ. GPSR was implemented as discussed where samples are stochastically resampled according optimal n (Equation 11). We used the Perlin noise implementation developed for [9]. This implementation includes several parameters that influence the shape of the generated synthetic sample (such as map resolution, amount of noise, *etc.*). Suitable values for these parameters were established via personal contact with the authors. Moreover, the authors expressed that the noise map was further smoothed via Gaussian blur prior to application, and these considerations are incorporated in our implementation. It is worth mentioning Perlin noise map generation is time consuming, which makes synthetic sample generation slow as compared to GPSR or ΣΛ (post parameter extraction). To reduce the time needed to run our experiments, we precomputed 2048 Perlin noise maps and cached them to disk prior to application. Although considerable speed improvements were observed, these cached maps required 64 MiB of storage space.

Our ΣΛ implementation is based on the recent parameter extraction algorithms described by Martín-Albo *et al.* [30]. Since parameter extraction can also be a time consuming process, we first extracted the ΣΛ models for all samples in the aforementioned datasets and used only sufficiently high quality models in our evaluation. That is, per [25], we required that the signal to noise ratio of a reconstructed model be 15dB or greater; otherwise the sample was excluded. Since the success of parameter extraction is dependent on signal quality, low resolution strokes can lead to difficult situations. One example problematic scenario occurs with the MMG [1] dataset where multiple points have the same timestamp, which causes an incorrect velocity calculation. We addressed these issues as much as possible, but it should be noted that our implementation is unlikely to be as robust as alternative propriety implementations. This is why we consider our results to be a reasonable lower bound on what is possible with the ΣΛ SDG method.

Recognizers

Template Matching Recognizers

Since gesture path stochastic resampling is designed to be used as a rapid prototyping technique, we expect $-family and other related recognizers to be paired with our method. Therefore, we decided to evaluate GPSR with six such recognizers: $1 [50], Protractor [26], $N [1], $N-protractor [2], $P [45], and Penny Pincher [22]. An additional benefit is that a variety of distance metrics are used throughout this suite of recognizers; so although these methods are all template matching, a large variety of techniques are represented.

Parametric Recognizers

Two parametric recognizers, Rubine's linear classifier [38] and naive Bayes were implemented and trained using the same set of features. We found that the original set of 13 features in [38] were inadequate for mixed-writer gesture recognition, and some features were unusable, since GPSR does simulate timestamps for example. To overcome these issues, we included some of the most prominent features described in [5]. The final features we settled for were the cosine ([38]:1)[7] and the sine ([38]:2) of the initial angle, angle of the bounding box's diagonal ([38]:4), the distance between the endpoints ([38]:5), cosine ([38]:6) and the sine ([38]:7) of the angle between the first and the last point, aspect ([5]:7-2), total angle traversed ([38]:9) as well as some convex hull related features such as length:perimeter ratio ([5]:2-6), perimeter efficiency ([5]:7-16) and perimeter:area ([5]:7-17) ratio.

Recognition Errors (Accuracy)

Figure 4 shows results for various recognizers on different datasets. These results were selected so as not to highlight any particular recognizer, dataset, or SDG method, though the results are consistent across all tested scenarios. One exception is naive Bayes (discussed below). Further, the reader may notice that ΣΛ performance is below baseline performance on the $P MMG dataset, but this result is compatible with those reported in [25]. Table 3 gives detailed recognition error rates for the best performing recognizer for each dataset given one real training sample per gesture. GPSR in all cases achieves the best performance. Minimally with 8 synthetic samples per training sample loaded, GPSR reduces the error rate by 18% on MMG and 30% on $1-GDS, whereas with the other two datasets, improvements approach 50%. At 64 synthetic samples per gesture, $P on MMG sees a 40% improvement, and $1 on EDS 1 enjoys a 79% reduction in the error rate. ΣΛ and Perlin noise also see improvements, but to lesser extent as can be seen in the table. Since we ran a vast number of tests, in what follows, we discuss average results across all recognizers and datasets. However, the full set of results can be found on our website.

Template Matching Gesture Recognizers

[7] This notation signifies the feature number as presented in the paper referenced.

Unistroke Gestures. Compared to the baseline percentage error ($M = 5.49, SD = 4.03$), without SDG, all methods showed an improvement in accuracy. GPSR achieved the lowest error ($M = 3.10, SD = 3.04$), which was followed by $\Sigma\Lambda$ ($M = 4.33, SD = 3.72$) and Perlin Noise ($M = 4.32, SD = 3.51$). These differences were statistically significant ($F(3, 464) = 8.05, p < .0001$), and a post hoc analysis using Tukey's HSD indicated that GPSR is significantly different from all other methods ($p < .005$), although baseline, $\Sigma\Lambda$, and Perlin noise were not significantly different from each other.

Multistroke Gestures. With the MMG [2] dataset, results were similar. GPSR ($M = 10.10$, $SD = 3.29$) achieves the highest performance, compared to baseline ($M = 14.67$, $SD = 4.42$), which was followed by Perlin noise ($M = 12.68$, $SD = 3.86$), and $\Sigma\Lambda$ ($M = 16.98$, $SD = 5.25$). Again, these differences were significant ($F(3, 74) = 11.25, p < .0001$), and a post hoc analysis showed that all SDG methods were significantly different from the baseline ($p < .0002$). However, the SDG methods were not significantly different from one another. Further, GPSR was significantly different from the baseline ($p < .04$), although Perlin noise and $\Sigma\Lambda$ were not significantly different from the baseline.

Parametric Gesture Recognizers

Both parametric recognizers were substantially improved by all SDG methods. The best performing method was Perlin noise ($M = 13.27$, $SD = 6.52$), which was very closely followed by GPSR ($M = 13.75$, $SD = 6.10$). $\Sigma\Lambda$ ($M = 15.57$, $SD = 7.01$) was also well below the baseline ($M = 32.96$, $SD = 9.47$). These results were statistically significant ($F(3, 152) = 10.998, p < .0001$), and again only GPSR was significantly different from the baseline ($p < .0001$).

Upon further inspection, we found that with Rubine, GPSR achieved the lowest mean error ($M = 11.46$, $SD = 5.18$), which was followed by Perlin Noise ($M = 13.42$, $SD = 5.85$). Conversely, with naive Bayes, Perlin noise achieved the lowest mean error ($M = 13.12$, $SD = 7.25$), followed by GPSR ($M = 16.04$, SD=6.18). Naive Bayes appears to be the only case where Perlin noise achieved a better result than GPSR.

EVALUATION: RUNTIME PERFORMANCE

To determine the average time required to generate one synthetic sample, we ran additional tests specifically to measure synthetic gesture generation speed with $1-GDS and MMG data sets. The performance of Perlin noise was evaluated both with and without using cached maps. The tests were performed on a Surface Pro™ 3 featuring a low-power dual core Intel Core-i7 mobile processor running at 2.30 GHz and 8 GiB of RAM. Table 4 summarizes the results.

According to table 4, the only method that is marginally faster than GPSR is cached Perlin noise. However, this superiority comes at the cost of additional storage needs. As mentioned before, caching 2048 Perlin noise maps requires 64 MiB of storage which may constrain its use on devices where available memory for applications is limited to a few hundred megabytes. Without caching, Perlin noise was the slowest method tested. Further, it is evident that all methods performed slightly worse on generating synthetic multistroke samples which coincides with expectations.

	$ 1-GDS		**MMG**	
	M (μs)	SD	M (μs)	SD
GPSR	6.75	0.43	8.34	0.45
$\Sigma\Lambda$	125.46	14.96	129.32	12.09
Perlin	1101.16	79.43	1091.89	30.31
Perlin†	6.82	0.49	7.38	0.64

Table 4: Average time required to generate one synthetic sample by each SDG method. All values are reported in microseconds and are averaged over 256000 trials. Perlin† set is performed using cached Perlin maps, and the $\Sigma\Lambda$ time does not include parameter extraction, which can take several seconds.

DISCUSSION

In all objectives set forth, gesture path stochastic resampling succeeds. Foremost, we believe the technique is rapid prototyping appropriate. For example, GPSR is a straightforward generalization of uniform resampling, i.e., with $\sigma^2 = 0, x = 0$, and the multistroke extension utilizes concepts from $N [1] and $P [45] without being more complex than either. Despite GPSR's reduced complexity, as compared to other state-of-the-art SDG methods, we see from the evaluation that our method is very competitive, for example, achieving a 79% error rate reduction for EDS 1 [47]. Indeed the accuracy of all recognizers are improved and, in most cases, GPSR is the best performing method. With respect to synthetic gesture realism, using optimal n, GPSR achieves a low ShE and BE percentage error for populations of synthetic gestures generated around centroid samples. Finally, in terms of computational complexity, we see that if noise maps are precomputed, then cached Perlin noise is slightly faster than GPSR; otherwise, GPSR is able to synthesize samples much faster than the other methods, and so our approach can be used in realtime to generate potentially hundreds of synthetic samples.

We see that as the number of real training samples increases, GPSR continues to show significant improvements in the error rates, though we can also observe that accuracy is converging between recognizers trained with and without synthetic data, see Figure 4. The biggest difference in performance occurs when only one or two real samples are available, which is our primary research goal and is especially important for gesture customization such as for gesture shortcuts [3]. As Appert and Zhai remark: "Users tend to be reluctant to invest time and effort upfront to train or adjust software before using it." This sentiment is echoed by Li [26] who notes that users are unwilling to provide more than a small set of samples for training. Consequently, methods that achieve more with less are of high value in gesture recognition. Still, even in situations where a user can provide numerous samples, the increase in accuracy was substantial for all levels of real training samples with the tested parametric recognizers. This result shows that for custom gestures, even when many samples are provided, synthetic data generation remains quite useful.

It is interesting to compare uniform and stochastic resampling with respect to the resampling rate n. Vatavu [44] was able

to show that template matching recognizers employing a Euclidean or angular distance metric are able to achieve high recognition rates with as little as $n = 6$. High accuracy is possible because corresponding points along the gesture path of two samples from the same class are equivalent (the distance between points in the feature space is small), which is generally untrue for samples from different gesture classes. With stochastic resampling, on the other hand, small values of n dramatically shifts points along the gesture path, which has potential to move points out of correspondence and this helps to explain why, in part, GPSR is useful in generating new variations. Unlike uniform resampling, however, we found that GPSR depends on a unique n per gesture to achieve a reasonable ShE. If realism is unimportant, static $n = 32$ is also a good compromise where smaller values result in too much variability and larger values do not provide enough.

Limitations

Although we strived to develop a general approach that works well for most situations, there are still a number of limitations one should bear in mind before choosing to use our method. First, like many synthetic data generation methods, GPSR does not synthesize timestamps, which is an important part of some gesture recognition strategies, *e.g.*, for features based on velocity and duration [38] or stroke segmentation [20]. Timestamps from an original sample can be interpolated and applied to synthetic samples, but this naïve approach may not produce realistic results. An alternative approach may use the findings of Cao and Zhai [7] in modeling gesture stroke production times based on characteristics of a given gesture, but we leave this for future work.

Another limitation of GPSR is that synthetic samples are bound to the seed samples from which they are derived, and so our method may not sufficiently capture form variability, especially between writers. If there are valid alternative productions of a gesture, such as skewed edges on a right square bracket, GPSR will not generate these forms given that we modify the subpath *length* between points, not the subpath *direction*. Additional geometric transformations like shearing, scaling, and bending, applied globally or to a subpath may help overcome this issue, an approach that has been successfully used to generate synthetic textlines [41].

Since synthetic samples do not deviate substantially from the seeds, GPSR generates a number of samples that appear redundant in the recognition space—the samples do not improve recognition accuracy because of their similarity with the seed and other synthetic samples. This limitation leads to two issues: we need to generate a large distribution of synthetic samples to achieve good coverage and these redundant samples may cause overfitting in a parametric recognizer. Template matching recognizers will be impacted by the distribution size since recognition time increases linearly with the number of templates. Penny Pincher [22] is designed specifically for speed and large template counts, but practitioners may need to be careful to balance the training set size with application and domain specific requirements when using other recognizers. As part of our future work, we intend to investigate how to cull a synthetic sample distribution down to just a few good samples. Pittman *et al.* [33] recently suggested to use random mutation hill climb to aid in template selection for $-family recognizers, and they were able to achieve high accuracy with significantly fewer templates, but the process is offline. We prefer an online process in which samples can be discarded immediately based on those samples already synthesized.

Also not addressed in this work is how to handle small strokes in a multistroke gesture. For instance, the dot in an 'i' or '!' may be discarded with a nonzero removal count $x > 0$; or if a small stroke appears in the middle of a gesture, the resampling algorithm may not produce any points within the associated subpath[8]. However, until a more elegant solution is identified, logic can be added to ensure small strokes are not skipped.

Future Work

Although we set out to design a rapid prototyping technique to complement such gesture recognizers, we discovered that GPSR is quiet capable; and so to better understand the limitations of gesture path stochastic resampling and also to understand if GPSR is appropriate for general use, further studies utilizing additional recognizers and SDG methods are warranted. Since we evaluated recognition accuracies in a mixed-writer scenario, we also require additional testing for pure writer-dependent and writer-independent scenarios. Our hope is that our method can help further bridge the gap between robust, high quality, commercial grade gesture recognition and rapid prototyping recognizers. For instance, it is desirable to evaluate the performance of our proposed approach when used for training support vector machines or random forests. Conversely, we also wish to evaluate if GPSR is suitable for testing recognizers, rather than training only.

We also plan to study user perception on synthetic gestures. A preliminary study in which participants rated the realism of synthetic and real gestures revealed significances between GPSR, $\Sigma\Lambda$, and Perlin noise samples, but not between GPSR and real samples. These results show promise that GPSR may be an appropriate method for rendering synthetic gestures that look real, but more work is needed to test and validate these results. Finally, beyond 2D gestures, we found that GPSR works reasonably well with signatures, sentences, and illustrations, though refinement is certainly necessary. For instance, we found that optimal n was unsuitable for such complex structures and, for instance, sentences will often run off their baseline. Finally, we would also like to explore how GPSR can be applied towards 3D gesture recognition.

CONCLUSION

In this paper we presented a novel technique for synthetic data generation that significantly improves gesture recognition accuracy. Through an extensive evaluation we demonstrated that GPSR is highly competitive with current state-of-the-art SDG techniques, achieving as much as a 70% reduction in recognition error rates. Using optimal n, GPSR is also able to generate synthetic gesture distributions that approximate real distributions based on the ShE and BE relative accuracy measures. Further, due to its minimalistic design and low computational cost, GPSR is also applicable to rapid prototyping.

[8]This is also true for uniform resampling.

ACKNOWLEDGMENTS

This work is supported in part by NSF CAREER award IIS-0845921. We also thank the other ISUE lab members at UCF for their support as well as the anonymous reviewers for their helpful feedback.

REFERENCES

1. Anthony, L., and Wobbrock, J. O. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010*, GI '10, Canadian Information Processing Society (Toronto, Ont., Canada, Canada, 2010), 245–252.
2. Anthony, L., and Wobbrock, J. O. $n-protractor: A fast and accurate multistroke recognizer. In *Proceedings of Graphics Interface 2012*, GI '12, Canadian Information Processing Society (Toronto, Ont., Canada, Canada, 2012), 117–120.
3. Appert, C., and Zhai, S. Using strokes as command shortcuts: Cognitive benefits and toolkit support. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, ACM (New York, NY, USA, 2009), 2289–2298.
4. Awal, A.-M., Mouchere, H., and Viard-Gaudin, C. Towards handwritten mathematical expression recognition. In *Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on* (July 2009), 1046–1050.
5. Blagojevic, R., Chang, S. H.-H., and Plimmer, B. The power of automatic feature selection: Rubine on steroids. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, Eurographics Association (Aire-la-Ville, Switzerland, Switzerland, 2010), 79–86.
6. Cano, J., Perez-Cortes, J.-C., Arlandis, J., and Llobet, R. *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, ch. Training Set Expansion in Handwritten Character Recognition, 548–556.
7. Cao, X., and Zhai, S. Modeling human performance of pen stroke gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, ACM (New York, NY, USA, 2007), 1495–1504.
8. Caramiaux, B., Montecchio, N., Tanaka, A., and Bevilacqua, F. Adaptive gesture recognition with variation estimation for interactive systems. *ACM Trans. Interact. Intell. Syst. 4*, 4 (Dec. 2014), 18:1–18:34.
9. Davila, K., Ludi, S., and Zanibbi, R. Using off-line features and synthetic data for on-line handwritten math symbol recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, IEEE (2014), 323–328.
10. Dinges, L., Elzobi, M., Al-Hamadi, A., and Aghbari, Z. A. *Image Processing and Communications Challenges 3*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, ch. Synthizing Handwritten Arabic Text Using Active Shape Models, 401–408.
11. Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2001.
12. Elanwar, R. I. The state of the art in handwriting synthesis. In *2nd International Conference on New Paradigms in Electronics & information Technology (peit'013), Luxor, Egypt* (2013).
13. Farooq, F., Jose, D., and Govindaraju, V. Phrase-based correction model for improving handwriting recognition accuracies. *Pattern Recogn. 42*, 12 (Dec. 2009), 3271–3277.
14. Fischer, A., Plamondon, R., O'Reilly, C., and Savaria, Y. Neuromuscular representation and synthetic generation of handwritten whiteboard notes. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on* (Sept 2014), 222–227.
15. Fischer, A., Visani, M., Kieu, V. C., and Suen, C. Y. Generation of learning samples for historical handwriting recognition using image degradation. In *Proceedings of the 2Nd International Workshop on Historical Document Imaging and Processing*, HIP '13, ACM (New York, NY, USA, 2013), 73–79.
16. Galbally, J., Fierrez, J., Martinez-Diaz, M., and Ortega-Garcia, J. Synthetic generation of handwritten signatures based on spectral analysis. In *SPIE Defense, Security, and Sensing*, International Society for Optics and Photonics (2009), 730629–730629.
17. Gatos, B., Konidaris, T., Ntzios, K., Pratikakis, I., and Perantonis, S. J. A segmentation-free approach for keyword search in historical typewritten documents. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, ICDAR '05, IEEE Computer Society (Washington, DC, USA, 2005), 54–58.
18. Ha, T. M., and Bunke, H. Off-line, handwritten numeral recognition by perturbation method. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*, 5 (May 1997), 535–539.
19. Helmers, M., and Bunke, H. Generation and use of synthetic training data in cursive handwriting recognition. In *Pattern Recognition and Image Analysis*. Springer, 2003, 336–345.
20. Herold, J., and Stahovich, T. F. Speedseg: A technique for segmenting pen strokes using pen speed. *Computers & Graphics 35*, 2 (2011), 250 – 264. Virtual Reality in BrazilVisual Computing in Biology and MedicineSemantic 3D media and contentCultural Heritage.
21. Herold, J., and Stahovich, T. F. The 1¢ recognizer: A fast, accurate, and easy-to-implement handwritten gesture recognition technique. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, Eurographics Association (Aire-la-Ville, Switzerland, Switzerland, 2012), 39–46.
22. II, E. M. T., Vargas, A. N., and Jr., J. J. L. Streamlined and accurate gesture recognition with penny pincher. *Computers & Graphics 55* (2016), 130 – 142.
23. Kutner, M. H., Nachtsheim, C. J., Neter, J., and Li, W. *Applied linear statistical models*, vol. 5. McGraw-Hill Irwin New York, 2005.
24. Lee, D.-H., and Cho, H.-G. A new synthesizing method for handwriting korean scripts. *International Journal of Pattern Recognition and Artificial Intelligence 12*, 01 (1998), 45–61.
25. Leiva, L. A., Martín-Albo, D., and Plamondon, R. Gestures À go go: Authoring synthetic human-like stroke gestures using the kinematic theory of rapid movements. *ACM Trans. Intell. Syst. Technol. 7*, 2 (Nov. 2015), 15:1–15:29.
26. Li, Y. Protractor: A fast and accurate gesture recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, ACM (New York, NY, USA, 2010), 2169–2172.
27. Lü, H., Fogarty, J. A., and Li, Y. Gesture script: Recognizing gestures and their structure using rendering scripts and interactively trained parts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, ACM (New York, NY, USA, 2014), 1685–1694.
28. Lundin, E., Kvarnström, H., and Jonsson, E. A synthetic fraud data generation methodology. In *Proceedings of the 4th International Conference on Information and Communications Security*, ICICS '02, Springer-Verlag (London, UK, UK, 2002), 265–277.
29. MacLean, S., Tausky, D., Labahn, G., Lank, E., and Marzouk, M. Tools for the efficient generation of hand-drawn corpora based on context-free grammars. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, ACM (New York, NY, USA, 2009), 125–132.

30. Martín-Albo, D., Plamondon, R., and Vidal, E. Improving sigma-lognormal parameter extraction. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on* (Aug 2015), 286–290.
31. Navaratnam, R., Fitzgibbon, A. W., and Cipolla, R. The joint manifold model for semi-supervised multi-valued regression. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on* (Oct 2007), 1–8.
32. Perlin, K. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, ACM (New York, NY, USA, 1985), 287–296.
33. Pittman, C., Taranta II, E. M., and LaViola, Jr., J. J. A $-family friendly approach to prototype selection. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, IUI '16, ACM (New York, NY, USA, 2016), 370–374.
34. Plamondon, R. A kinematic theory of rapid human movements. *Biological cybernetics 72*, 4 (1995), 295–307.
35. Plamondon, R., and Djioua, M. A multi-level representation paradigm for handwriting stroke generation. *Human movement science 25*, 4 (2006), 586–607.
36. Rodriguez-Serrano, J. A., and Perronnin, F. Synthesizing queries for handwritten word image retrieval. *Pattern Recognition 45*, 9 (2012), 3270 – 3276. Best Papers of Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA'2011).
37. Rowley, H. A., Goyal, M., and Bennett, J. The effect of large training set sizes on online japanese kanji and english cursive recognizers. In *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on* (2002), 36–40.
38. Rubine, D. Specifying gestures by example. *SIGGRAPH Computer Graphics 25*, 4 (July 1991), 329–337.
39. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. Real-time human pose recognition in parts from single depth images. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, IEEE Computer Society (Washington, DC, USA, 2011), 1297–1304.
40. Thomas, A. O., Rusu, A., and Govindaraju, V. Synthetic handwritten captchas. *Pattern Recogn. 42*, 12 (Dec. 2009), 3365–3373.
41. Varga, T., and Bunke, H. Generation of synthetic training data for an hmm-based handwriting recognition system. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on* (Aug 2003), 618–622 vol.1.
42. Varga, T., and Bunke, H. Offline handwriting recognition using synthetic training data produced by means of a geometrical distortion model. *International Journal of Pattern Recognition and Artificial Intelligence 18*, 07 (2004), 1285–1302.
43. Varga, T., Kilchhofer, D., and Bunke, H. Template-based synthetic handwriting generation for the training of recognition systems. In *Proceedings of the 12th Conference of the International Graphonomics Society* (2005), 206–211.
44. Vatavu, R.-D. The effect of sampling rate on the performance of template-based gesture recognizers. In *Proceedings of the 13th International Conference on Multimodal Interfaces*, ICMI '11, ACM (New York, NY, USA, 2011), 271–278.
45. Vatavu, R.-D., Anthony, L., and Wobbrock, J. O. Gestures as point clouds: A $p recognizer for user interface prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction*, ICMI '12, ACM (New York, NY, USA, 2012), 273–280.
46. Vatavu, R.-D., Anthony, L., and Wobbrock, J. O. Relative accuracy measures for stroke gestures. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, ICMI '13, ACM (New York, NY, USA, 2013), 279–286.
47. Vatavu, R.-D., Vogel, D., Casiez, G., and Grisoni, L. Estimating the perceived difficulty of pen gestures. In *Proceedings of the 13th IFIP TC 13 International Conference on Human-computer Interaction - Volume Part II*, INTERACT'11, Springer-Verlag (Berlin, Heidelberg, 2011), 89–106.
48. Velek, O., and Nakagawa, M. *Document Analysis Systems V: 5th International Workshop, DAS 2002 Princeton, NJ, USA, August 19–21, 2002 Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, ch. The Impact of Large Training Sets on the Recognition Rate of Off-line Japanese Kanji Character Classifiers, 106–110.
49. Wobbrock, J. O., Findlater, L., Gergle, D., and Higgins, J. J. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, ACM (New York, NY, USA, 2011), 143–146.
50. Wobbrock, J. O., Wilson, A. D., and Li, Y. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, ACM (New York, NY, USA, 2007), 159–168.

APPENDIX A: PSEUDOCODE

In this appendix, we present the gesture path stochastic resampling pseudocode. Note that Stochastic-Resample is adapted from Resample in [50]. Use of optimal n is considered optional, and we recommend using $n = 32$ otherwise.

Make-Synthetic-Sample (*strokes*)

```
/*** Creates a synthetic sample from the given sample.          ***/
unistroke ←Make-Stochastic-Unistroke(strokes, true)
n ←Optimal-N(unistroke)
x ← 2
σ² ← 0.25
unistroke ←Stochastic-Resample(unistroke, n, x, σ²)
multistroke ← Make-Multistroke(unistroke)

return multistroke
```

Stochastic-Resample (*points*, n, x, σ^2)

```
/*** Generate n + x − 1 normalized intervals.                   ***/
n ← n + x
total ← 0

for i ← 0 to n − 2 do
    intervals_i ←Rand-Float(min = 1, max = 1 + √(12 + σ²))
    total ← total + intervals_i

for i ← 0 to n − 2 do
    intervals_i ← intervals_i / total

/*** Perform resampling.                                        ***/
pathDistance ←Path-Length(points)
cnt ← 0
I ← pathDistance ∗ intervals_cnt
D ← 0
v ← {points_0}

foreach p_i in points for i ≥ 1 do
    d ←Distance(p_i, p_i−1)
    if D + d ≥ I then
        t ←Min(Max( (I − D) / d, 0), 1)
        prev ← p_i−1
        q_x ← (1 − t) ∗ prev_x + t ∗ p_ix
        q_y ← (1 − t) ∗ prev_y + t ∗ p_iy
        q_stroke_id ← (1 − t) ∗ prev_stroke_id + t ∗ p_i, stroke_id
        Append(v, q)
        InsertAt(points, i, q)
        D ←0
        cnt ← cnt+1
        I ← pathDistance ∗ intervals_cnt
    else
        D ← D + d

/*** Remove random points.                                      ***/
for i ← 1 to x do
    idx ← Rand-Int(min = 0, max = n − i)
    Remove(v, idx)

/*** Normalize vectors.                                         ***/
normalizedPoints ← {Point(0,0, v_0,stroke_id)}
for i ← 1 to |v| − 1 do
    prev ← v_i−1
    dx ← v_ix − prev_x
    dy ← v_iy − prev_y
    len ← √(dx² + dy²)
    q_x ← normalizedPoints_i−1x + dx/len
    q_y ← normalizedPoints_i−1y + dy/len
    q_stroke_id ← v_i, stroke_id
    Append(normalizedPoints, q);
return normalizedPoints
```

Distance (Point a, Point b)

```
return √((a_x − b_x)² + (a_y − b_y)²)
```

Make-Stochastic-Unistroke (*strokes*, *randomize*)

```
/*** Permute strokes with Fisher-Yates shuffle,                 ***/
/*** and handle stroke direction invariance.                    ***/
strokeCount ← Length(strokes)
if randomize then
    for i ← strokeCount − 1 downto 0 do
        j ← Rand-Int(min = 0, max = i)
        Swap(strokes_i, strokes_j)
        if Rand-Float(min=0, max=1) < 0.5 then
            Reverse(strokes_i)

/*** Combine strokes into a unistroke.                          ***/
v = {}
for i ← 0 to strokeCount − 1 do
    pointCount ← Length(stroke_i.points)
    for j ← 0 to pointCount − 1 do
        Append(v, strokes_i.points_j)

return v
```

Optimal-N (*points*)

```
/*** Computes the optimal stochastic resampling factor.         ***/
endpointsDist ← Distance(points_0, points_|points|−1)
pathLength ← Path-Length(points)
diagonal ← Diagonal(points)

density ← pathLength / diagonal
closedness ← 1 − endpointsDist / diagonal

/*** Compute the optimal n, Equation 11                         ***/
n ← exp(1.67 + 0.29 ∗ density + 1.42 ∗ closedness)

if n < 16 then
    n ← 16
return n
```

Make-Multistroke (*points*)

```
/*** Converts a unistroke to a multistroke.                     ***/
multistroke ← { }, temp ← { }
id ← 0

foreach p_i in points for i ≥ 0 do
    prev_id ← id
    id ← p_i.stroke_id

    /*** The point belongs to a stroke part                     ***/
    /*** if its id is an integer.                               ***/
    if IsInteger(id) then
        if temp ≠ {} and prev_id ≠ id then
            Append(multistroke, temp)
            temp ← {}
        Append(temp,p_i);

/*** Append any remaining points.                               ***/
if temp ≠ {} then
    Append(multistroke, temp);

return multistroke
```

Path-Length (Points *points*)

```
d ← 0
for i ← 1 to |points| − 1 do
    d ← d + Distance(points_i−1, points_i)
return d
```

Diagonal (Points *points*)

```
minPoint ← Point(Min_x(points), Min_y(points))
maxPoint ← Point(Max_x(points), Max_y(points))

return Distance(minPoint, maxPoint)
```
