Gestimator - Shape and Stroke Similarity Based Gesture Recognition

Yina Ye, Petteri Nurmi
Helsinki Institute for Information Technology HIIT
Department of Computer Science, PO Box 68, FI-00014, University of Helsinki, Finland
yina.ye.yyn@gmail.com
petteri.nurmi@cs.helsinki.fi

ABSTRACT

Template-based approaches are currently the most popular gesture recognition solution for interactive systems as they provide accurate and runtime efficient performance in a wide range of applications. The basic idea in these approaches is to measure similarity between a user gesture and a set of pre-recorded templates, and to determine the appropriate gesture type using a nearest neighbor classifier. While simple and elegant, this approach performs well only when the gestures are relatively simple and unambiguous. In increasingly many scenarios, such as authentication, interactive learning, and health care applications, the gestures of interest are complex, consist of multiple sub-strokes, and closely resemble other gestures. Merely considering the shape of the gesture is not sufficient for these scenarios, and robust identification of the constituent sequence of sub-strokes is also required. The present paper contributes by introducing Gestimator, a novel gesture recognizer that combines shape and stroke-based similarity into a sequential classification framework for robust gesture recognition. Experiments carried out using three datasets demonstrate significant performance gains compared to current state-of-the-art techniques. The performance improvements are highest for complex gestures, but consistent improvements are achieved even for simple and widely studied gesture types.

Categories and Subject Descriptors

I.5.2 [**Pattern Recognition**]: Design Methodology: Feature evaluation and selection; I.5.4 [**Pattern Recognition**]: Applications: Signal processing; H.4.m [**Information Systems**]: Information Systems Applications: Miscellaneous

General Terms

Algorithms, Experimentation

Keywords

Gesture Recognition; Segmentation; Pattern Matching

1. INTRODUCTION

Popularity of touch screen devices combined with increased availability of low-cost gesture tracking technologies, such as consumer-grade depth cameras and accelerometer equipped wearable devices, has resulted in a surge of gesture-based interfaces. Examples of platforms for gesture-based interfaces include gaming consoles [15], mobile devices [1, 9], and smart watches [10]. At the same time, highly accurate and easy to implement gesture recognizers have become available [3, 14, 23], providing increasingly opportunities for developing and prototyping gesture-based interfaces.

Template-based algorithms are currently the most popular gesture recognition approach for interactive systems as they provide several advantages over their counterparts. Firstly, they are computationally efficient, which is particularly important for emerging gesture recognition solutions operating on resource constrained devices, such as smartwatches, smartphones, or other wearable devices [1, 2, 9]. Secondly, template-based recognizers have been shown to provide robust and accurate recognition rates (typically over 90%) even with small amounts of training data [3, 14]. Thirdly, template-based algorithms are intuitive to understand and easy to implement, making it possible to develop and deploy them without expertise in machine learning techniques.

The basic idea in template-based gesture recognizers is to measure similarity between a user's gesture and previously seen gestures, or templates. The user's gesture can then be classified as the type of gesture with highest overall similarity. While simple and elegant, this approach performs well only when gestures are simple and unambiguous. This problem is illustrated in Figure 1 which shows two motion trajectories, one corresponding to the character '{' and the other to '['. When only the similarity of the entire trajectory is considered, these gestures can be easily mistaken, resulting in a wrong interpretation. The issue is further exacerbated by the fact that visual (dis)similarity does not directly translate into gesture (dis)similarity due to the design of gesture recognizers. In fact, as will be later shown, even relatively dissimilar gestures such as a triangle and a circle can be misinterpreted by contemporary gesture recognizers, making it difficult for designers to determine gesture sets that only include unambiguous gestures. At the same, novel domains for gesture interaction are emerging where gestures are inherently complex and ambiguous. Examples of these domains include authentication [17], interactive learning [24], interactive performances [8], and health care [5]. Providing an easy to implement, accurate, and run-time efficient gesture recognition algorithm that can cope with complex and ambiguous gestures currently remains an open challenge.

The present paper contributes by developing *Gestimator* as a novel stroke-based gesture recognizer. The key intuition in Gestimator is to segment gestures into strokes and to match gestures by comparing the similarity of their constituent strokes. Segmenting gestures into strokes helps to resolve ambiguity in the recognition phase, improving overall gesture recognition performance. This is illustrated on the right-hand side of Figure 1 which shows how the strokes of '{' and '[' help to distinguish the two gestures. Segmenting gestures into constituents, however, is far from straightforward as noise and variations in the gesture trajectories, resulting from variations in how users perform gestures, make it difficult to produce a consistent and unique segmentation for a gesture. To overcome these issues, Gestimator builds on a novel adaptive segmentation and matching framework that adapts to the noise level of each gesture and that provides consistent and accurate recognition performance even in the presence of segmentation differences for a specific gesture.

We demonstrate the effectiveness of Gestimator through extensive benchmark experiments carried out using three gesture datasets: a set of unistroke command gestures [23], the Execution Difficulty Set (EDS1) of Vatavu et al. [20], and a dataset consisting of mid-air gestures from a device pairing scenario [2]. As part of the experiments we compare Gestimator against three state-of-the-art gesture recognition techniques: the \$1 recognizer [23], Protractor [14], and a DTW-based recognizer [13]. Results of the experiments demonstrate significant performance improvements for all datasets considered in the experiments. The improvements are largest for complex gestures, but consistent improvements are achieved even for simple and widely studied gesture types. As part of the experiments, we also demonstrate that Gestimator results in robust and consistent segmentation performance and that the runtime performance of Gestimator is comparable to state-of-the-art template-based gesture recognizers.

The contributions of the paper are summarized as follows:

1. We develop Gestimator, a novel stroke-based gesture recognizer.
2. We develop a novel adaptive and sequential segment and match framework for comparing gestures based on the combined similarity of their overall shape and the similarity of their constituent strokes.
3. We carry out extensive empirical evaluation of Gestimator, demonstrating that it improves gesture recognition performance compared to popular state-of-the-art gesture recognizers while at the same time preserving runtime efficiency and simplicity of implementation.

2. RELATED WORK

Gesture recognition algorithms can be categorized into two main classes. *Parametric* approaches extract different features from the user's gesture and treat recognition as a classification task. A prominent example is Rubine [16], which uses 13 features extracted from the user's gesture as input to a linear discriminant classifier. Vuurpilj and Schomaker [21] extract features related to relative position, orientation, and curvature. Willems et al. [22] introduce a set of 48 different gesture features. Generally parametric approaches can use any (supervised) classification technique, with Support Vector Machines and Hidden Markov models being popular choices; see, e.g., [11, 22]. The main limitations of parametric approaches are their need for large amounts of training data, and the difficulty of integrating them into user interfaces.

The alternative is to use a *template-based* recognizer which determines the type of gesture by comparing user's input against previously seen gestures, or templates. An early example of template-based recognizers is the SHARK2 [13] system for tablet computers, which uses dynamic time warping (DTW) to compare the similarity of gestures. Another example is the \$1 recognizer [23], which uses Euclidean distance to compare gestures. The \$1 recognizer also incorporates various preprocessing steps, such as scaling, subsampling, and alignment of gesture orientations, to provide robust performance against minor changes in gesture directions and variations in gesture tracking technology. The 1¢ recognizer [7] is a variation of the \$1 recognizer that uses distances from the gesture centroid instead of the original points for recognition. Instead of using Euclidean distance, Protractor [14] compares gestures using minimal angular distance. Kratz and Rohs propose a close-formed quaternion representation for reorienting gestures in order to extend the Protractor algorithm to support 3D gestures. Anthony and Wobbrock [3] propose the \$N recognizer, which extends the \$1 recognizer to support multi-stroke gestures. Vatavu [18] explores the effect of sampling rate on the performance of template-based recognizers, demonstrating that even small sampling rates tend to result in highly accurate performance. The \$P recognizer is another extension of the \$1 recognizer, which ignores the ordering of points within gestures during the recognition process [19].

To summarize, previous research on template-based gesture recognition has predominantly targeted development of recognizers that are easy to implement and integrate as part of gesture interfaces. These recognizers are well suited for scenarios where the user gestures are relatively simple, e.g., commands for scrolling, rotating or zooming, and do not contain ambiguity. Our work provides a novel simple-to-implement template-based recognizer that extends the scope of template-based recognition techniques to support scenarios where the gestures are complex, contain ambiguity, and significant amount of user variation.

3. GESTIMATOR: STROKE-BASED GESTURE RECOGNITION

We have developed Gestimator as a novel gesture recognizer that simultaneously compares the similarity of the entire gesture and its constituent sub-strokes. The key technical contributions of Gestimator are (i) an adaptive segmentation algorithm that breaks down the user's gesture into constituents; and (ii) a sequential matching formulation that calculates the similarity of two gestures by comparing its constituent sub-strokes. As our experiments demonstrate, the combination of these contributions provides consistent improvements in gesture recognition accuracy, in particular for complex and ambiguous gestures.

Figure 1 shows an overview of the different processing phases in Gestimator. First, in a *preprocessing* phase, noise is removed from the user's gesture and the gesture data is transformed into a scale and position invariant form. The main source of noise we consider is jitter resulting from sub-

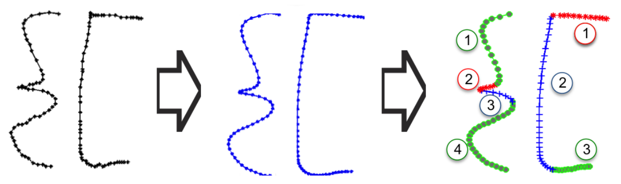

Figure 1: Overview of the gesture recognition pipeline in Gestimator. The raw gesture input (left) is first smoothed and cleaned (middle), after which it is segmented into constituent strokes (right). The gesture recognition phase sequentially compares the strokes in a gesture against previously seen templates to determine appropriate gesture type.

tle hand movements and inaccuracies in the gesture data. Next, the preprocessed gesture is *segmented* into constituent sub-strokes using an adaptive algorithm that identifies peaks in the curvature of the gesture data. Finally, *gesture matching* determines the type of the gesture by comparing the segmented gesture against previously seen gesture trajectories. The matching is performed using a novel sequential matching algorithm that allows local mismatches in the constituent sub-strokes, enabling the matching to operate accurately against different segmentations of the same gesture. In the following we describe the different phases of Gestimator in detail.

3.1 Preprocessing

Gestimator operates on multi dimensional measurements containing timestamped position vectors. In the experiments we consider motion gestures captured from a touchscreen and mid-air gestures captured using a Kinect[1]. We first perform jitter removal by applying a centered moving average filter on the measurements. Jitter results from inaccuracies in the gesture tracking and from subtle, unconscious hand movements. The latter is particularly prominent with in-air gestures, where the hand performing the gesture never is completely stationary. Performing noise removal is essential for ensuring consistent segmentation performance as otherwise small errors in the gesture trajectory can result in additional strokes being detected.

After noise removal, we perform start and end point removal on the gesture data. In our experiments we have observed that users often start and end gestures by having a short stationary period during which subtle hand movements cause changes in the gesture trajectory. These periods can degrade the performance of the segmentation phase unless they are dealt with appropriately. We perform start and end point removal using a density-based clustering algorithm that iteratively merges the first (resp. last) points in the trajectory until a consistent movement trajectory is identified. All points in the detected clusters are removed as noise; see left-hand side of Fig. 1 for illustration. After noise removal, in line with current best practices [14, 23], we normalize the gesture by unit scaling and centering it,

[1]We consider gestures to be multi-stroke whenever they can be seen to contain multiple constituents. In the case of handwriting or touch-screen gestures, constituents can be identified from touch or pen motions. However, in the context of mid-air gestures clear boundaries seldom can be identified.

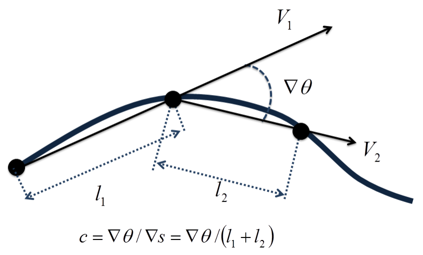

Figure 2: The curvature of a gesture is defined as the ratio of change over a given distance.

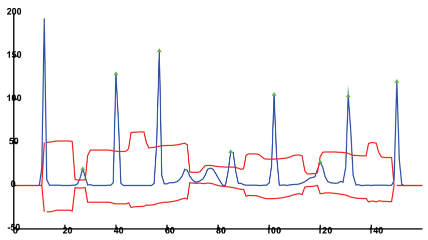

Figure 3: Curvature plot for a gesture together with the confidence bound used for segmentation and the associated peaks corresponding to segment points.

thus achieving scale invariance. Rotation invariance can be achieved by normalizing the orientation of the gesture [14, 23]. In our experiments only scale invariance is considered.

3.2 Stroke Segmentation

Gestimator builds on the novel idea of comparing gestures as sequences of sub-strokes. Segmenting the gesture enables resolving ambiguity by focusing on differences in smaller subsequences of the gesture instead of comparing the entire gesture. To ensure best possible recognition performance, a robust and accurate method for segmenting the gesture into constituent strokes is required. In particular, the segmentation should be consistent across different repetitions of the same gesture and able to identify the most distinguishable parts of the gesture. To preserve the main benefits of template-based recognizers, we also need to ensure the segmentation is sufficiently lightweight to remain runtime efficient even on resource constrained devices.

To accomplish these goals, Gestimator considers an adaptive segmentation algorithm that identifies peaks in the *curvature* of the gesture (see Fig. 2 for a definition). Curvature is well suited for segmentation as it provides a unified measure for changes in the speed and direction. Changes in speed have been widely used to segment sketches into constituents [6], whereas changes in orientation are a staple of line simplification methods, used, e.g., in computer graphics and trajectory monitoring [4, 12]. In practice, however, gestures tend to contain significant variation, e.g., some users simplify gestures in order to perform them more smoothly whereas others consistently monitor their speed to perform the gesture shape as accurately as possible. As we demonstrate in the experiments, using curvature enables robust segmentation across these variations.

Our approach for segmenting gestures into strokes is summarized in Algorithm 1. We first construct a curvature vector V from the preprocessed gesture and apply a sliding win-

Algorithm 1 Segmentation Algorithm

```
1: A sliding window stroke segmentation method:
2: Inputs: $V$ vector of input gesture curvature, $\alpha_w$ window
   length to total length ratio.
3: Outputs: $T$ vector containing the segmentation indexes
   of the gesture.
4: procedure $T$ = StrokeSegment($V$, $\alpha_w$)
5:     $l_w \leftarrow \lfloor$ Length( $V^*$ ) $\times \alpha_w \rfloor$
6:     $P \leftarrow \emptyset$
7:     for $i = 1 : \lfloor l_w \rfloor :$ Length($V$) do
8:         $l^* \leftarrow$ getWindow($i$)
9:         $I \leftarrow$ DetectSegmentPoint($l^*$)
10:        $P := P \bigcup I$
11:    end for
12:    for $i =$ Length($V$) $: \lfloor l_w \rfloor : 1$ do
13:        $l^* \leftarrow$ getWindow($i$)
14:        $I \leftarrow$ DetectSegmentPoint($l^*$)
15:        $P := P \bigcup I$
16:    end for
17:    return $T$
18: end procedure
```

dow based peak detection on V. To ensure the segmentation can cope with gestures of different length, we determine the size of the sliding window adaptively based on the overall length of the gesture (line 5). For each window, we apply a peak detection method (lines 9 and 14) that identifies points where the curvature changes significantly. Specifically, our peak detection method first constructs a 95% confidence interval for the curvature values within a window. Successive points that fall outside the confidence interval form a so-called *peak area* and the maximal value within each peak area is then chosen as a segmentation point; see Figure 3 for an illustration. To ensure all relevant segment points can be detected, we apply the peak detection in both forward (lines 7 - 11) and backward directions (lines 12 - 16). Once the segment points are detected, we represent the gesture as a sequence of its constituents. Formally, we denote the segmented gesture using $a =< a_1, \ldots, a_k >$ where k is the number of sub-strokes in the gesture.

3.3 Gesture Matching

Gestimator recognizes gestures by comparing the sequence of sub-strokes in a gesture against the sub-strokes of previously seen templates. The general principle is that the higher the similarity between each sub-stroke, the more likely the gesture corresponds to the given template. A challenge with this scheme, however, is that variations in the way gestures are performed can cause two segmentations of the same gesture to differ from each other, e.g., a square can be segmented into three sub-strokes when one of the turns is performed as a round corner. To ensure high recognition performance in the presence of segmentation differences, we consider an accumulative matching framework where subsequent strokes can be merged if they produce higher similarity than the individual strokes alone would.

The intuition behind the accumulative matching framework considered in Gestimator is illustrated in Figure 4. Given two gestures $a =< a_1, \ldots, a_k >$ and $b =< b_1, \ldots, b_l >$, we represent them as binary trees and progressively traverse the corresponding trees by comparing different combinations of strokes. Specifically, each node in the tree correspond to one stroke of the corresponding gesture. The left child of a node corresponds to the case where the subsequent stroke is compared independently of the current stroke, whereas the right child correspond to an accumulative comparison where the two strokes are merged. At each node, the child with higher similarity is chosen. We use DTW as the similarity measure for the strokes since it enables an elastic comparison that accounts for different execution speeds and lengths of the strokes. Note that, as the comparison is performed separately for each stroke, DTW cannot use overall similarity to compensate for intermediate gaps, which is the main problem of current DTW-based recognizers [13]. To ensure efficient runtime performance, we consider a simple traversal heuristic that only considers the subtree specified by the most recent stroke. Consequently, we only need to perform three comparisons at each level of the binary tree, resulting in linear runtime complexity. In practice, the runtime of the matching phase in Gestimator is comparable to that of using a DTW-based recognizer.

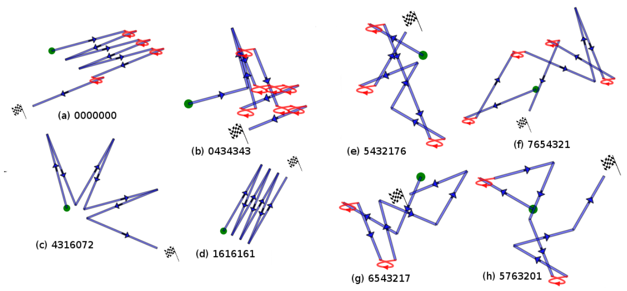

Figure 5: The gesture types in the **Authentication** dataset. The green circle indicates the start of the gesture, and the checkered flag represents the end. Arrows indicate direction of motion. The red swirl gestures are used to reverse the direction of next stroke to constrain the overall gesture within a hand reachable region.

4. EXPERIMENTAL SETUP

We demonstrate the effectiveness of Gestimator through rigorous and extensive benchmark evaluations carried out using three datasets. As part of the evaluation, we compare Gestimator against three popular state-of-the-art gesture recognizers: the \$1 recognizer [23], Protractor [14], and a DTW-based recognizer [13]. The preprocessing phases and parameters of the baseline algorithms were chosen according to the original articles, and we verified our results by comparing them against the original publications. The datasets we consider were chosen to contain gestures of different complexity and ambiguity, ranging from simple unistroke command gestures to complex multi-stroke authentication gestures. The datasets considered in our evaluation are summarized as follows:

1. **Unistroke** is a dataset of pen-based unistroke gestures[2], collected by Wobbrock et al. [3] from 10 participants. For each participant, the dataset contains 10 repetitions of 16 different command gestures performed with 3 different execution speed levels: fast, medium and slow. The Unistroke dataset has been

[2]Available from: `http://depts.washington.edu/aimgroup/proj/dollar/`.

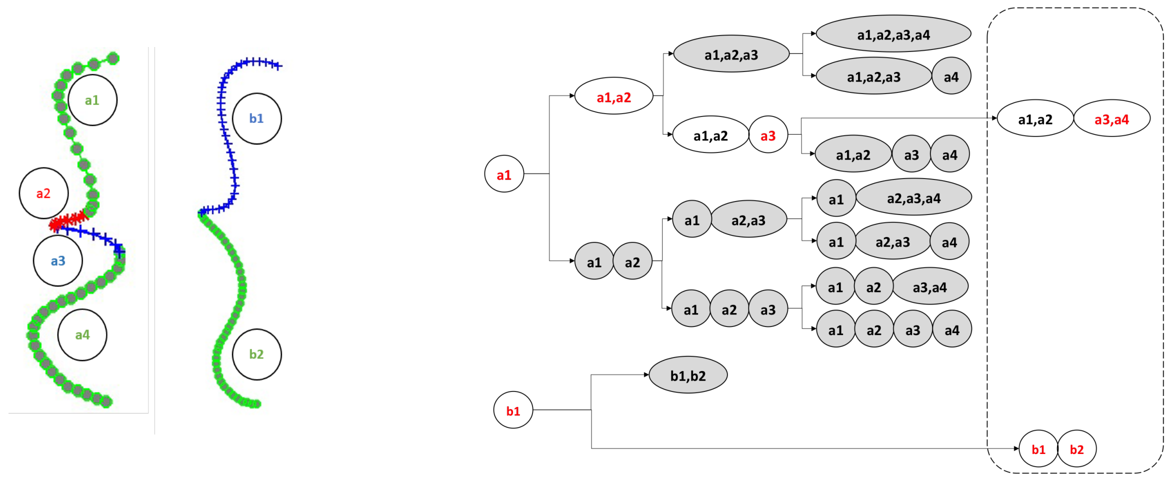

Figure 4: Illustration of the accumulative gesture matching process. The constituents of gestures $a = < a_1, a_2, a_3, a_4 >$ and $b = < b_1, b_2 >$ are progressively compared by traversing the corresponding binary trees and assessing the similarity of the constituents. The highest similarity is obtained for the combination $(a_1, a_2), (a_3, a_4)$ and b_1, b_2.

widely used to evaluate template-based recognizers and we included it to provide a baseline for comparison for the performance of Gestimator.

2. **Execution Difficulty Set #1 (*EDS1*)** is a dataset of pen stroke gestures[3] collected by Vatavu et al. [20]. The dataset contains 20 repetitions of 18 gesture types by 14 participants.
3. **Authentication** consists of continuous mid-air gestures recorded using a Kinect[4]. The gestures are generated automatically and consist of 7 straight line segments and additional swirl strokes, which are used to ensure the gestures remain confined to a sufficiently small region. The dataset contains measurements from 13 participants who performed 8 different gestures for 6 times; see Fig. 5 for the gesture types. As can be observed, the gesture patterns are complex and contain ambiguity, enabling us to assess the performance of Gestimator in a complex usage scenario of practical interest. More details about the application scenario and the experimental setup are given in [2].

5. RESULTS

To assess the performance of Gestimator, we have used the datasets described in the previous section to carry out an extensive and rigorous set of benchmark experiments. In our evaluation we consider the performance of Gestimator both in user-dependent (i.e., personalized) and user-independent gesture recognition, and compare Gestimator against state-of-the-art baselines. We also separately consider the performance of Gestimator in the presence of ambiguous gestures, and evaluate the accuracy and consistency of the stroke segmentation algorithm.

5.1 User-Dependent Recognition

We first consider the recognition performance of Gestimator in user-dependent testing, i.e., in a personalized scenario where training samples from the end user are available. We perform our evaluation by considering a cross-validation scheme that is analogous to the one used by Wobbrock et al. [23] to evaluate the $1 recognizer. Specifically, we test each user separately, taking k randomly sampled repetitions of each gesture as training data and using the remaining samples as test data. In the experiment k was varied from 1 to 5 and the experiment was carried out 100 times for each level of k, i.e., for a given user and gesture we performed 500 recognition tests.

The results of the evaluation are summarized in Table 1. From the results we can observe that all algorithms generally achieve high recognition accuracy on the simpler pen gestures, but their performance decreases on the authentication gestures. Gestimator has best overall performance, with an average recognition error of 1.85%. Compared to the baselines, Gestimator is able to consistently outperform

#Training:	**1**	**2**	**3**	**4**	**5**	**Avg.**
			Unistroke			
DTW	5.24	2.61	1.92	1.43	1.19	2.48
$1	6.26	3.18	2.06	1.72	1.36	2.91
Protractor	3.61	1.60	1.22	1.00	0.76	1.64
Gestimator	2.36	1.27	0.94	0.84	0.73	1.23
			EDS1			
DTW	4.52	2.50	0.95	0.75	0.79	1.90
$1	5.02	2.29	1.39	1.13	0.84	2.13
Protractor	5.50	2.46	1.54	0.96	0.92	2.28
Gestimator	3.18	0.83	0.42	0.19	0.13	0.94
			Authentication			
DTW	8.44	3.80	2.91	2.73	0.99	3.77
$1	10.87	5.35	3.92	3.42	0.90	4.89
Protractor	17.91	9.23	6.87	5.87	1.82	8.34
Gestimator	8.82	3.55	2.04	1.89	0.68	3.39

Table 1: Recognition error of Gestimator and the baseline algorithms as a function of amount of training data.

[3]Available from `http://www.eed.usv.ro/~vatavu/index.php?menuItem=pengestures2011`.
[4]Available for research purposes from `universe.hiit.fi/gesture`

them, particularly on the EDS1 and authentication datasets, which contain complex and ambiguous gesture pairs. We analyzed statistical significance of the performance differences by conducting McNemar's tests with Yates' correction on the on the confusion matrices of the recognizers. The results of the significance analysis showed the performance differences between Gestimator and the baseline methods to be statistically significant for all datasets ($p < .001$).

One of the main benefits of template-based recognizers is that they are capable of achieving high recognition rates with very small amounts of training data. From the results in Table 1 we can observe this to be the case also for Gestimator, which outperforms the baselines with a single exception. On the authentication dataset Gestimator performs slightly worse than DTW when only a single training sample is available. However, the difference between the algorithms is not statistically significant and both algorithms perform an order of magnitude better the other two baselines. From the results we can also observe that, on the simpler pen-based gesture datasets, the performance difference between Gestimator and the baseline techniques is highest when only one or two training samples per gesture are available.

We also assessed the runtime performance of the different algorithms as part of the experiment. Protractor has the fastest overall performance, reaching an average matching time of 28ms. The next best is $1 (mean 37ms) followed by DTW (mean 52ms). The matching time of Gestimator is expectedly slowest (mean 86 ms), especially since it builds on DTW. While slower than that of other techniques, it remains more than feasible for most practical applications of gesture recognition. Overall the results thus demonstrate that Gestimator is capable of providing significant improvements in the robustness and accuracy of gesture recognition in a diverse set of scenarios. Moreover, these improvements are reached even when very small amounts of training data are available. These improvements are obtained with a small decrease in runtime performance. However, overall the results suggest that Gestimator is the best suited algorihtm, unless the gestures are very simple and sufficient amounts of training data are available.

5.2 User-Independent Recognition

We next consider the generalization performance of Gestimator in a user independent gesture recognition scenario. We carry out the evaluation by randomly choosing one user as test subject. To train the different algorithms, we construct a training dataset by randomly choosing p other users. For each training user, we randomly choose k different repetitions of each gesture type. The randomization is performed 100 times for each different value of p and k.

The results of this evaluation are summarized in Table 2. Generally the performance of all algorithms suffers in the user-independent case, with particularly Protractor resulting in significant performance degradation. When training from only one or two test users is available, Gestimator loses against the DTW recognizer. However, when data from three or more users is available, Gestimator already starts to consistently outperform the other algorithms. The decrease in performance is likely to be caused by the fact that Gestimator assumes gestures are performed in the same order, whereas there are some variations in the order people perform gestures. As illustrated in the previous section, Gestimator only requires very limited amount of data from each different type of user, thus demonstrating generally a good generalization performance.

Table 2: Error rates (percentage) of user independent testing

# Training	**1**	**2**	**3**	**4**	**5**
			Unistroke		
DTW	13.19	8.22	5.86	4.99	3.96
$1	14.27	8.79	6.79	5.52	4.72
Protractor	16.46	9.96	7.54	6.38	4.83
Gestimator	14.11	8.06	4.41	3.26	2.57
			EDS1		
DTW	2.35	0.53	0.20	0.19	0.17
$1	3.33	1.06	0.56	0.25	0.13
Protractor	10.37	4.80	2.96	2.25	1.73
Gestimator	8.15	2.41	0.27	0.00	0.00
			Authentication		
DTW	7.42	6.23	5.25	5.53	4.59
$1	24.86	12.71	10.11	6.33	6.50
Protractor	28.66	20.27	12.86	11.98	11.71
Gestimator	16.56	6.75	4.62	3.40	3.06

5.3 Gesture Ambiguity

One of the main design goals for Gestimator has been to improve recognition rates in the presence of ambiguous gestures. We demonstrate this is the case by carrying out an evaluation where we consider all gesture with error rate of 2.5% or higher with any of the algorithms considered in the evaluation. The results of this evaluation are summarized in Table 3. From the table, we can observe that ambiguity is highly algorithm dependent, with different pairs being problematic for different recognizers. Besides the example of detecting '{' and '[' being generally difficult, also somewhat surprising pairs such as triangle vs. rectangle and circle vs. 6 cause difficulties to recognizers that rely on overall shape similarity. The results also show that ambiguity can depend on the characteristics of the dataset as the gesture pair rectangle vs. circle is more difficult for all recognizers in the EDS1 dataset than in the Unistroke dataset.

For the majority of the ambiguous gesture pairs (21/35), we can observe that Gestimator outperforms the baseline algorithms, and even in the remaining cases Gestimator is usually within 1% error of the best performing algorithm. Only two exceptions can be observed, both of which are on the authentication dataset (0434343 vs. 0000000 and 6543217 vs. 5432176; see Fig. 5). The reason for errors in these gesture pairs is that the pairs have many matching strokes, despite having different overall similarity. As we have used an elastic similarity measure, the mismatching strokes are not always sufficient for separating the gesture pairs. This issue can be mitigated by adjusting the similarity measure to assign higher gap penalty for mismatches between stroke pairs. However, even for these highly complex gesture pairs the recognition rate of Gestimator is high, achiving smaller than 3.5% error rate.

5.4 Segmentation Performance

The performance of Gestimator depends on the accuracy of the stroke segmentation algorithm described in Sec. 3.2.

Unistroke		DTW	$1	Prot.	Gest.
check	v	3.09	2.67	**1.56**	1.67
circle	rectangle	5.60	5.24	1.82	**0.85**
circle	triangle	2.69	3.13	**0.05**	1.73
{	[	3.29	4.67	3.36	**1.34**
?	caret	2.55	3.44	**0.51**	0.82
?	]	2.00	3.02	**1.13**	1.33
rectangle	circle	0.07	0.22	3.36	**0.07**
triangle	circle	3.47	4.56	1.75	**1.72**
triangle	rectangle	4.29	6.18	0.87	**0.75**
v	check	3.07	2.00	**1.91**	2.69
EDS1		DTW	$1	Prot.	Gest.
8	reversed-pi	1.14	2.83	1.97	**0.08**
8	strike-through	4.00	3.07	2.30	**0.00**
8	turn-90	0.86	3.43	2.49	**0.00**
circle	6	6.00	5.34	4.96	**0.00**
circle	rectangle	4.29	3.99	4.89	**0.80**
rectangle	circle	0.86	**0.31**	2.46	1.12
steep-hill	sail-boat	0.43	2.73	2.87	**0.00**
triangle	6	3.29	2.80	2.80	**0.00**
triangle	circle	2.86	2.53	3.66	**0.37**
Authentication		DTW	$1	Prot.	Gest.
0000000	0434343	**1.92**	3.52	2.91	2.06
0000000	1616161	2.69	1.20	**1.02**	1.13
0000000	4316072	0.93	2.18	3.89	**0.88**
0434343	0000000	1.78	**0.78**	0.98	3.49
0434343	4316072	1.38	2.55	2.98	**0.15**
1616161	4316072	1.55	4.63	7.12	**0.56**
4316072	1616161	1.52	3.05	3.38	**1.47**
4316072	6543217	0.15	0.11	4.50	**0.13**
5432176	5763201	3.63	**2.34**	2.36	3.11
5432176	6543217	3.57	3.51	3.76	**2.24**
5763201	0000000	0.14	1.23	3.81	**0.14**
5763201	4316072	0.38	1.78	4.86	**0.35**
6543217	5432176	3.20	**1.57**	2.46	3.33

Table 3: Error rates of different algorithms for gesture pairs with the highest error rates.

Specifically, accurate segmentation ensures consistent subsequences of the original gesture are being compared during the matching phase. Accurate segmentation also translates into faster runtime performance by reducing the number of look-ahead steps in the matching phase. As the next step of evaluation we demonstrate that Gestimator is capable of segmenting gestures accurately and consistently.

We assess segmentation accuracy by comparing the number of strokes resulting from the gesture segmentation against the actual number of strokes in a gesture. Ground truth for the Unistroke and EDS1 datasets was obtained through majority voting on the results of five independent annotators. Inter-rater reliability was assessed using Cohen's kappa and found to be substantial ($\kappa > .7, p < .001$). For the authentication dataset, the number of strokes was predetermined by the encoding scheme used to generate the gestures. Segmentation consistency was measured by calculating the variance and range of the number of strokes for each repetition of the same gesture.

Figure 6 shows the cumulative distribution plot of the segmentation errors for the different datasets. The lowest performance is obtained for the authentication dataset where 97.8% of the gestures being segmented with at most 2 additional strokes. The main reason for errors in the authentication dataset are the circular swirl motions, which sometimes are segmented into multiple segments. For the other two datasets segmentation accuracy is much higher. On the Unistroke dataset 98.2% of the gestures result in accurate segmentation. On the EDS1 dataset 90.5% of the gestures are segmented with an error of at most a single stroke. The results for segmentation consistency, summarized in Table 4, closely mirror the results of segmentation error. On the Unistroke and EDS1 datasets the average variance of the number of strokes for the same gesture is below 0.5, suggesting that most gestures are segmented with at most a single stroke difference. On the authentication dataset the number of strokes varies bit more, with an average difference of 1.5 strokes and a maximum error of 4 strokes. Analogously to the segmentation error results, most of these differences are due to variations in the way users perform swirl motions. To summarize, overall the results demonstrate that Gestimator has accurate segmentation performance and is capable of providing consistent segmentation across different users and across variations in user performance.

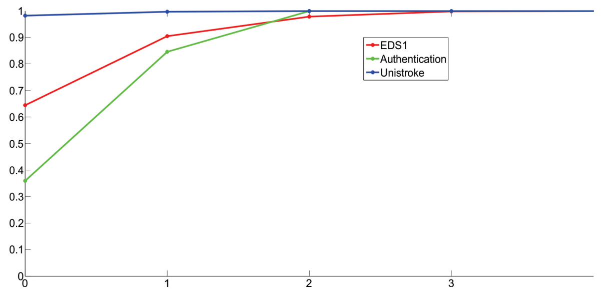

Figure 6: Cumulative probability distribution plot for segmentation error. The values on the x-axis correspond to the error in number of strokes, whereas the y-axis shows the cumulative probability.

Dataset	Variance	Range
Unistroke	0.43	0 - 2
EDS1	0.41	0 - 2
Authentication	1.48	0 - 4

Table 4: Summary of the results for evaluation of segmentation consistency.

6. SUMMARY AND CONCLUSION

We have contributed by presenting Gestimator, a novel template-based gesture recognizer that compares gestures by considering the similarity of stroke sequences in them. The main technical contributions of our work have been an algorithm for segmenting gestures into strokes, and an algorithm for assessing the similarity of stroke sequences, taking into account possible errors in segmentation. Experiments carried out using three datasets demonstrate significant performance gains compared to existing state-of-the-art techniques. The results also demonstrate that Gestimator is capable of accurately and consistently segmenting gestures into strokes, and that it has good generalization performance against different sources of variation in gesture trajectories.

The results also demonstrate that the performance gains of Gestimator are particularly significant for gesture pairs that contain ambiguity or a high level of complexity.

7. ACKNOWLEDGMENTS

The work has been partially funded by DIGILE D2I SHOK programme. The work only reflects the authors' views.

8. REFERENCES

[1] S. Agrawal, I. Constandache, S. Gaonkar, R. R. Choudhury, K. Caves, and F. DeRuyter. Using mobile phones to write in air. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services (MobiSys 2011)*, 2011.

[2] I. Ahmed, Y. Ye, S. Bhattacharya, N. Asokan, G. Jacucci, P. Nurmi, and S. Tarkoma. Checksum Gestures: Continuous Gestures as an Out-of-Band Channel for Secure Pairing. In *Proceedings of The 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*, 2015.

[3] L. Anthony and J. O. Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Graphics Interface*, 2010.

[4] J. Brault and R. Plamondon. Segmenting handwritten signatures at their perceptually important points. *Pattern Analysis and Machine Intelligence*, 15:953 – 957, 1993.

[5] M. Fan, D. Gravem, D. M. Cooper, and D. J. Patterson. Augmenting gesture recognition with erlang-cox models to identify neurological disorders in premature babies. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing (UbiComp)*, 2012.

[6] J. Herold and T. F. SpeedSeg: A technique for segmenting pen strokes using pen speed. *Computers and Graphics*, 35:250 – 264, 2011.

[7] J. Herold and T. F. Stahovich. The One Cent Recognizer: A Fast, Accurate, and Easy-to-Implement Handwritten Gesture Recognition Technique. In *Proceedings of the Symposium on Sketch Based Interfaces and Modeling (SBM)*, pages 39–46. Eurographics Association, 2012.

[8] H.-S. Ip, K. Law, and B. Kwong. Cyber Composer: Hand Gesture-Driven Intelligent Music Composition and Generation. In *Proceedings of the 11th International Multimedia Modelling Conference (MMM)*, 2005.

[9] W. Kienzle and K. Hinckley. Writing handwritten messages on a small touchscreen. In *Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 2013.

[10] J. Kim, J. He, K. Lyons, and T. Starner. The Gesture Watch: A Wireless Contact-free Gesture based Wrist Interface. In *Proceedings of the 11th IEEE International Symposium on Wearable Computers (ISWC)*, 2007.

[11] J.-B. Kim, K.-H. Park, W.-C. Bang, and Z. Bien. Continuous gesture recognition system for Korean sign language based on fuzzy logic and hidden Markov model. In *Proceedings of the 2002 IEEE International Conference on Fuzzy Systems*, pages 1574–1579. IEEE, 2002.

[12] M. B. Kjærgaard, S. Bhattacharya, H. Blunck, and P. Nurmi. Energy-efficient trajectory tracking for mobile devices. In *Proceedings of the 9th International Conference on Mobile Systems, Applications and Services (MobiSys)*, 2011.

[13] P. O. Kristensson and S. Zhai. SHARK^2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2004.

[14] Y. Li. Protractor: A Fast and Accurate Gesture Recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2010.

[15] M. Raptis, D. Kirovski, and H. Hoppe. Real-Time Classification of Dance Gesturesfrom Skeleton Animation. In *Proceedings of the 2011 Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, 2011.

[16] D. Rubine. Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1991.

[17] M. Sherman, G. Clark, Y. Yang, S. Sugrim, A. Modig, J. Lindqvist, A. Oulasvirta, and T. Roos. User-generated free-form gestures for authentication: Security and memorability. In *The 12th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2014.

[18] R.-D. Vatavu. The effect of sampling rate on the performance of template-based gesture recognizers. In *Proceedings of the 13th International Conference on Multimodal Interfaces (ICMI)*, 2011.

[19] R.-D. Vatavu, L. Anthony, and J. O. Wobbrock. Gestures as point clouds: a $P recognizer for user interface prototypes. In *Proceedings of the International Conference on Multimodal Interaction (ICMI)*, 2012.

[20] R.-D. Vatavu, D. Vogel, G. Casiez, and L. Grisoni. Estimating the perceived difficulty of pen gestures. In *Human-Computer Interaction–INTERACT 2011*, pages 89–106. Springer, 2011.

[21] L. Vuurpijl and L. Schomaker. Finding structure in diversity: a hierarchical clustering method for the categorization of allographs in handwriting. In *Proceedings of the 4th International Conference Document Analysis and Recognition (ICDAR)*, 1997.

[22] D. Willems, R. Niels, M. van Gerven, and L. Vuurpijl. Iconic and multi-stroke gesture recognition. *Pattern Recognition*, 12:3303–3312, 2009.

[23] J. O. Wobbrock, A. D. Wilson, and Y. Li. 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)*, pages 159–168. ACM, 2007.

[24] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti. American sign language recognition with the kinect. In *Proceedings of the 13th International Conference on Multimodal Interfaces (ICMI)*, 2011.